Technical Report no. 2005-01

Interactive Verification of JCSP Programs

Philipp Rümmer

CHALMERS | GÖTEBORG UNIVERSITY

Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden

Göteborg, 2005

Technical Report in Computing Science at
Chalmers University of Technology and Göteborg University

Technical Report no. 2005-01
ISSN: 1652-926X

Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden

Göteborg, 2005

Contents

I. Übersicht **1**

1. Einführung **3**
1.1. Spezifikation von Eigenschaften nebenläufiger Systeme 4

2. Communicating Sequential Processes (CSP) **5**
2.1. Modellierung von JCSP-Systemen in der CSP-Algebra 6

3. Spezifikation und Verifikation von Eigenschaften von CSP-Prozessen **7**

4. Erweiterung von CSP um P/T-Netz-Kausalität **9**

5. Algebraische Eigenschaften von CCSP **11**
5.1. Freie Stellen von Prozessen 11
5.2. Grundlegende Eigenschaften der CCSP-Operatoren 11
5.3. Elimination von Präfix und Auswahl aus Alternativen 12
5.4. Hilfsmittel für Transformation in Normalform 12
5.4.1. Abspalten von Transitionen aus Parallelismus 12
5.4.2. Abspalten von Transitionen aus den Operatoren für Benennung von Prozessen und Verbergen von Ereignissen 13
5.4.3. Abspalten von Transitionen aus den Operatoren für Verbrauch von Marken 13

6. Transformation von CCSP-Termen in Normalform **15**
6.1. Analyse des Termersetzungssystems 16

7. Zusammenfassung **19**

II. Thesis **21**

1. Introduction **23**
1.1. Calculi for CSP 24
1.2. Propositions about Processes 25
1.3. Related Work 27
1.4. How the Thesis is Organised 27

2. Communicating Sequential Processes **29**
2.1. The Considered Fragment of CSP . . . 29
2.1.1. The Fundamental CSP Operators . . . 31
2.2. Formal Definition of the CSP Operators . . . 36
2.2.1. Labelled Transition Systems . . . 37
2.2.2. Definition of the CSP Operators . . . 41
2.3. Modelling JCSP using CSP . . . 46

3. Specification and Verification of CSP Process Properties **53**
3.1. Temporal Correctness Assertions . . . 53
3.2. Hennessy-Milner-Logic . . . 54
3.3. A Calculus for CSP and Hennessy-Milner-Logic Using SOS Rules . . . 55
3.4. A Calculus for CSP and Hennessy-Milner-Logic Using Rewriting . . . 57
3.5. Analysis of CSP Process Terms in General . . . 58
3.6. Analysis of CSP Process Terms Using a Causal Extension of CSP . . . 61

4. Extending CSP with P/T Net Causality **65**
4.1. The Variant of P/T Nets Used in the CSP Extension . . . 65
4.1.1. Monotonic Behaviour of P/T Nets . . . 68
4.2. The Extension CCSP of CSP . . . 70
4.2.1. An Operational Semantics for Monotonic P/T Nets . . . 70
4.2.2. P/T Net Causality for Arbitrary Processes . . . 71
4.2.3. Operators of CCSP . . . 73
4.3. Formal Definition of the CCSP Operators . . . 80
4.4. Embedding of CSP in CCSP . . . 83

5. Algebraic Properties of CCSP **85**
5.1. Free Places of a Process . . . 85
5.2. Fundamental Properties of the CCSP Operators . . . 87
5.3. Reduction of Prefixing and Choice to Interleaving and P/T Net Causality 91
5.4. Reasoning about Closed Process Terms . . . 93
5.5. Transition Propagation through Parallelism . . . 98
5.6. Transition Propagation through Labelling and Hiding . . . 103
5.7. Transition Propagation through Token Consumption . . . 104
5.7.1. Duplication of Places in Monotonic P/T Nets . . . 104
5.7.2. Example of Distribution of Token Consumption . . . 106
5.7.3. Duplication of Places in Processes . . . 107
5.7.4. Merging of Places in Processes . . . 111
5.7.5. Distribution of Token Consumption . . . 118

6. Transformation of CCSP Terms to Head Normal Form **121**
6.1. Prerequisites . . . 122
6.2. Syntax of CCSP . . . 123
6.3. Interpretation of CCSP Syntax . . . 125
6.3.1. Free Places of Processes Described by CCSP Terms . . . 126
6.3.2. Renaming of Places in a Process Term . . . 126

6.4. A Rewriting System for Analysing CCSP Process Terms 127
6.5. Example 134
6.6. Correctness of the Rewriting System 140
6.7. Normal Forms Established by Rewriting 143
6.8. Reasoning about Open Process Terms 144
6.8.1. Evaluation of the Prefix of an Irreducible Process Term 145
6.8.2. Syntactic Analysis of Irreducible Process Terms 146
6.9. Simplification of CCSP Terms 151
6.10. Calculi for CCSP and Hennessy-Milner-Logic Using Rewriting 153
6.10.1. Direct Evaluation of Hennessy-Milner-Logic Formulas 153
6.10.2. Evaluation of Hennessy-Milner-Logic Formulas Using Marking Relations 157

7. Conclusion and Future Work **159**

A. Proofs **161**
A.2. Communicating Sequential Processes 162
A.4. Extending CSP with P/T Net Causality 162
A.5. Algebraic Properties of CCSP 164
A.6. Transformation of CCSP Terms to Head Normal Form 206

List of Tables

2.1. Plotkin-style Structural Operational Semantics of CSP 36
2.2. Dynamics of the LTS $(\mathcal{S}, \alpha_{\mathcal{S}})$ Defining the CSP Operators 44

3.1. Gentzen-style Sequent Calculus for Hennessy-Milner-Logic 56

4.1. Operational Semantics of CCSP: Causal State Operator 77
4.2. Operational Semantics of CCSP: Parallelism 77
4.3. Operational Semantics of CCSP 78
4.4. Characteristic Equations Defining CCSP 81
4.5. Characteristic Equations Defining CCSP: Parallelism 82

5.1. SOS Rule for Application of Approximations to Processes 96
5.2. Example of Propagation of Transitions through Parallelism 99
5.3. Online Algorithm for Merging Two Streams of Transitions 101
5.4. Example of Propagation of Transitions through Parallelism: Rewriting . 102
5.5. SOS Rules for Duplicating and Merging Places of Processes 108
5.6. The Relation $(m, n) \lhd_{p+q} (m', n')$ for Merging Places of Processes 112

List of Figures

1.1. Example Process Consisting of Three States 26

2.1. Simplified UML Class Diagrams of JCSP 46
2.2. Example Network of Two Processes in JCSP 48

5.1. Example of Splitting Places and Transitions in P/T Nets 105
5.2. Example of "Inclusive Or" Consumption of Tokens in P/T Nets 105
5.3. Example of Duplication of Places in a CCSP Process Term 106
5.4. Nesting and Commuting of Duplicating and Merging Places of Processes 115

6.1. Example of P/T Net that Results from Rewriting Parallel Composition . 138

Definitions and Lemmas

Definition 2.2.1	Labelled Transition System	37
Definition 2.2.2	LTS-Homomorphism	38
Definition 2.2.3	Bisimulation Equivalence	39
Lemma 2.2.5	LTS Categories	40
Definition 4.1.1	Monotonic P/T Nets	65
Definition 4.1.2	Dynamics of Monotonic P/T Nets	66
Definition 4.1.3	Addition of I/O	67
Lemma 4.1.4	Addition of I/O	67
Definition 4.1.5	Approximations of P/T Net Markings	68
Lemma 4.1.6	Approximations of P/T Net Markings	69
Lemma 4.4.1	Translation of CSP to CCSP	84
Definition 5.1.2	Free Places of a Process	87
Lemma 5.1.3	Free Places and CCSP	87
Lemma 5.2.1	Basic Properties of Choice and Parallelism	87
Lemma 5.2.2	Commutation of Causal State Operator	88
Lemma 5.2.3	Properties of Token Consumption	89
Lemma 5.2.4	Properties of Transitions	91
Lemma 5.3.1	Reduction of Prefixing and Choice	93
Definition 5.4.1	Application of Approximations	95
Lemma 5.4.3	Application of Approximations	97
Definition 5.5.1	Transition Merging	103
Lemma 5.5.2	Propagation through Parallelism	103
Lemma 5.6.1	Propagation through Labelling and Hiding	104
Lemma 5.7.3	Properties of Duplicating Places	110
Lemma 5.7.5	Properties of Merging Places	113
Lemma 5.7.6	Elimination of Place Merging	117
Lemma 5.7.7	Propagation through Token Consumption	118
Definition 6.6.1	Well-formed CCSP Process Term	141
Lemma 6.6.2	Rewriting of Well-formed Terms	142
Lemma 6.7.1	Properties of Rewriting System	143
Definition 6.8.3	Blocked Process Term	147
Lemma 6.8.4	Blocked Process Term	148

The whole document is summarised in the first part on the following pages. According to requirements for a diploma thesis in Germany, this part is written in German. Because it does not provide any technical contents beyond what is given in the remainder starting on page 23 (in English), it has not been translated to English but has been kept as an add-on for readers capable of understanding German.

The origin of this document is:

Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe

Diplomarbeit

Interactive Verification of JCSP Programs

Philipp Rümmer

30. September 2004

Verantwortlicher Betreuer: Prof. Dr. Peter H. Schmitt
Betreuer: Vladimir Klebanov, Steffen Schlager

Danksagung

In allererster Linie möchte ich meinen Betreuern Vladimir Klebanov und Steffen Schlager danken, die in den letzten Monaten harte Arbeit mit unausgereiften Zwischenversionen dieses Dokuments hatten, und die mich trotzdem stets mit sinnvollen Kommentaren und Verbesserungsvorschlägen versorgt haben. In ähnlicher Weise hat mir Professor Peter Schmitt geholfen, der vor allem in der Anfangsphase in diversen Diskussionen viel zur Klärung von grundsätzlichen Fragen beigetragen hat, vielen Dank dafür.

Besondere Anerkennung gebührt Thomas Arts, Gennady Chugunov, Mads Dam, Dilian Gurov, Lars-Åke Fredlund, sowie den übrigen Mitgliedern von SICS die bereitwillig Tage investiert haben, um uns in die Geheimnisse des EVT Systems einzuweihen.

Den Mitgliedern des KeY-Projektes in Karlsruhe, die mir während der Implementierungsphase viele Probleme mit dem KeY-System aus dem Weg geräumt haben, sei hiermit für die umfangreiche geleistete Hilfe gedankt.

Schließlich möchte ich meinen Eltern und meiner Schwester für Motivation und finanzielle Unterstützung während des Studiums danken.

Part I.

Übersicht

1. Einführung

Das KeY-Projekt ist ein Gemeinschaftsprojekt der Universität Karlsruhe, der Chalmers Universität in Göteborg und der Universität Koblenz mit der Zielsetzung, Hilfsmittel für die Spezifikation und Verifikation von Software zu entwickeln. Zentrale Teil des KeY-Projekts ist die Entwicklung eines integrierten interaktiven und automatischen Beweisers für die Verifikation von JavaCard-Programmen (JavaCard ist eine Variante von Java für SmartCard-Software), der auf einer dynamischen Logik für diese Sprache basiert.

Die vorliegende Arbeit behandelt einen Ansatz zur Erweiterung des bestehenden Beweisers, bei dem die bereits existierende Behandlung sequentieller Java-Programme zur Verifikation von nebenläufigen Systemen verwendet werden soll. Dabei werden Systeme betrachtet, die nicht mit Hilfe der Java-eigenen Primitive (Threads, shared memory, Monitore) Nebenläufigkeit realisieren, sondern die auf die Java-Bibliothek JCSP aufbauen. JCSP ist eine Sammlung von Java-Klassen zur Erstellung von mehrfädigen Systemen, die ein Prozessmodell ähnlich dem CSP Modell von Hoare implementiert, und bei der die zentralen Softwarekomponenten über Nachrichtenkanäle miteinander kommunizierende Prozesse sind. Informationsübertragung durch shared memory ist in diesem Modell nicht vorgesehen.

Es ist anzunehmen, dass die Beschränkung des zu entwickelnden Beweissystems auf ein Prozessmodell einer höheren Ebene (als das Java-Thread-Modell) die Verifikation von komplexeren Systemen ermöglicht, da Teile von Beweisen, die andernfalls die Korrektheit von Kommunikationsvorgängen sicherstellen, entfallen.

Als Grundlage des Kalküls für die Behandlung von JCSP-Systemen wurde eine Modellierung von JCSP-Primitiven mit Hilfe einer generischen Prozessalgebra (CSP) ausgewählt. Dabei wird der bestehende Kalkül des KeY-Beweisers, der eine symbolische Ausführung von Java-Programmen vornimmt, erweitert um die Möglichkeit, CSP-Prozessterme (die in diesem Fall als atomare Bestandteile JCSP-Prozesse in Form von sequentiellen Java-Programmen enthalten) symbolisch abzuarbeiten. Der größte Teil dieser Arbeit beschreibt und analysiert einen Kalkül für die CSP-Prozessalgebra, der besonders gut für diesen Zweck geeignet ist.

Da das Einbetten von JCSP-Prozessen in CSP-Terme in jedem Fall zu Formeln und Sequenzen von erheblicher Größe führt, und da für eine interaktive Benutzung des Kalküls auch die Übersichtlichkeit der Beweise wesentlich ist, muss ein geeigneter CSP-Kalkül eine kompakte Repräsentation von Zwischenzuständen der Prozesse ermöglichen. Ein Anwachsen von Termen während der Abarbeitung muss nach Möglichkeit vermieden werden. Da diese Anforderung mit einer direkten Verwendung von CSP nicht zu erreichen war, basiert die beschriebene Methode auf einer Erweiterung von CSP um einen Kausalitätsbegriff ähnlich dem von Petrinetzen.

1.1. Spezifikation von Eigenschaften nebenläufiger Systeme

Für die Spezifikation von Eigenschaften nebenläufiger Systeme sind die für sequentielle Programme üblichen Konzepte von Vor- und Nachbedingungen nur bedingt geeignet, da nebenläufige Systeme häufig (als reguläres Verhalten) nicht terminieren, und da für eine Verwendung der Spezifikation zur modularen Programmentwicklung neben Anfangs- und Endzuständen auch das Verhalten in Zwischenzuständen wichtig ist. Diese Arbeit verwendet daher zur Darstellung von solchen Systemen Prozesse, d.h. im Allgemeinen unendliche Zustandsübergangssysteme, deren Übergänge mit beobachtbaren Ereignissen beschriftet sind. Zur Beschreibung von Eigenschaften von Prozessen werden modale oder temporale Logiken eingesetzt, wobei sich die Arbeit auf die Behandlung der besonders einfachen Hennessy-Milner-Logik beschränkt. Da Zustandsübergangssysteme als Kripke-Rahmen multimodaler Logiken betrachtet werden können, lassen sich Formeln dieser Logiken für Prozesse auswerten.

2. Communicating Sequential Processes (CSP)

Die Arbeit behandelt eine eingeschränkte Version der CSP-Algebra, bei durch verbliebene Verknüpfungen definierbare Operatoren weggelassen sind. Um CSP einzuführen, wird eine Menge von Prozessen (der Prozessraum) mit zweierlei Strukturen versehen: Einer Koalgebra-Struktur, welche das dynamische Verhalten die einzelnen Prozesse beschreibt, sowie einer Algebra-Struktur durch die Definition der CSP-Verknüpfungen. Da die Festlegung der Verknüpfungen Bezug auf das dynamische Verhalten der behandelten Prozesse nimmt, hat die Koalgebra-Struktur primären Charakter; sie ist im übrigen auch unabhängig von der Algebra CSP.

Die definierten Operatoren von CSP sind:

$stop$:	Inaktiver Prozess
$a \to P$:	Präfix
$P \,\Box\, Q$:	Auswahl aus Alternativen
$c \,?\, x \to P$:	Lesen von Daten aus Kanälen
$P \,[\![X]\!]\, Q$:	Parallelkomposition
$l : P$:	Benennen von Prozessen
$P \setminus X$:	Verbergen von Ereignissen

Im Wesentlichen werden diese Operatoren mit der Standardsemantik nach Hoare versehen. Die einzige (nicht völlig unwesentliche) Ausnahme von dieser Regel ist die Behandlung unsichtbarer Berechnungsschritte: Statt einem einzigen Element τ für die Darstellung interner Ausführungsschritte wird eine Menge $\Sigma_\tau = \{\tau(a) \,:\, a \in \Sigma\}$ solcher Symbole eingeführt. Der Operator $P \setminus X$ nimmt eine Umbenennung von Ereignissen $a \in \Sigma$ zu $\tau(a) \in \Sigma_\tau$ vor.

Als Abkürzung wird ein eigenes Symbol für Interleaving eingeführt durch

$$P \,|||\, Q \quad := \quad P \,[\![\varnothing]\!]\, Q$$

Zur formalen Definition der Operatoren wird zunächst eine geeignete Kategorie $\underline{\mathrm{LTS}}_{\mathcal{L}}$ von Zustandsübergangssystemen (Koalgebren) definiert, und die Existenz finaler Objekte der Kategorie festgestellt. Ein solches finales Objekt repräsentiert die Menge aller mit dem gewähltem Prozessbegriff darstellbaren Prozesse (bis auf Bisimularität), da beliebige Prozesse – repräsentiert durch Objekte der Kategorie – eineindeutig homomorph in das finale Objekt abgebildet werden können. Der Zustandsraum des finalen Objekts wird deshalb als Menge $Proc_{\Sigma_0}$ aller existierenden Prozesse betrachtet (die Menge $\mathcal{L}$ von Kantenbeschriftungen des Zustandsübergangssystems, wie auch die Mengen Σ und Σ_τ sind vom atomaren CSP-Alphabet Σ_0 abgeleitet).

Bezugnehmend auf eine gegebene strukturelle operationale Semantik werden die genannten CSP-Operatoren als Verknüpfungen auf $Proc_{\Sigma_0}$ definiert. Dazu wird das Verfahren der so genannten Koinduktiven Definition eingesetzt, das auf Eigenschaften finaler Objekte von $\underline{\mathrm{LTS}}_{\mathcal{L}}$ basiert, und das die Existenz von Prozessen mit beliebig gewähltem Ausführungsverhalten (als Elementen von $Proc_{\Sigma_0}$) zusichert.

2.1. Modellierung von JCSP-Systemen in der CSP-Algebra

Für eine Auswahl stark vereinfachter Klassen der JCSP-Bibliothek wird die Darstellung durch CSP-Verknüpfungen beschrieben. Die behandelten Klassen realisieren die Parallelkomposition von JCSP-Prozessen und synchrone Nachrichtenkanäle. Die Modellierung durch CSP umfasst die folgenden Schritte:

- Definition eines geeigneten CSP-Alphabets Σ_0, um die durch JCSP-Kanäle ausgetauschten Nachrichten in der CSP-Welt darstellen zu können.
- Definition einer Interpretationsfunktion, die für jedes JCSP-System einen entsprechenden Prozess $P \in Proc_{\Sigma_0}$ konstruiert. Für den Basisfall (JCSP-Systeme, die nur aus einem atomaren Prozess, d.h. einem sequentiellen Java-Programm bestehen) wird dabei auf eine operationale Semantik von Java zurückgegriffen, die ergänzt ist um spezielle Regeln für die Methoden der Nachrichtenkanal-Klasse. Zusammengesetzte Systeme werden rekursiv interpretiert, wobei die Komposition der erhaltenen Unterprozesse mit Hilfe von CSP-Verknüpfungen beschrieben ist.

Die Beschreibung der JCSP-Kompositionsprimitive anhand von CSP ergibt eine direkte Übersetzung von JCSP-Systemen zu äquivalenten CSP-Termen, auf die ein Beweissystem aufgesetzt werden kann.

3. Spezifikation und Verifikation von Eigenschaften von CSP-Prozessen

Als Basis und Stellvertreter für mächtigere temporale oder modale Logiken, die zur Spezifikation von Prozessen auch in der Praxis verwendbar sind, wird Hennessy-Milner-Logik (HML) eingeführt. Formeln Φ dieser Logik werden dabei im Rahmen von temporalen Korrektheitszusicherungen

$$T : \Phi$$

verwendet, wobei T Beschreibung eines Prozesses durch einen Term (in dieser Arbeit durch CSP) ist. Die gegebene Zusicherung drückt aus, dass der durch T dargestellte Prozess die Eigenschaft Φ hat, d.h. dass die Auswertung der Formel Φ im durch den Prozess gegebenen Kripke-Rahmen den Wahrheitswert ‚wahr' ergibt (HML enthält keine Prädikate oder aussagenlogische Variable, weswegen die Begriffe ‚Kripke-Rahmen' und ‚Kripke-Struktur' zusammenfallen).

Um Kalküle für Korrektheitszusicherungen zu konstruieren, ist es notwendig, das dynamische Verhalten von durch T beschriebenen Prozessen zu bestimmen. Die Lösung dieser Aufgabe kann schrittweise erfolgen, da es bei einer top-down-Abarbeitung der Formel Φ ausreicht, vollständige Information über alle möglichen ersten Berechnungsschritte solcher Prozesse zu erhalten. Beispielsweise ist es in der Situation

$$T : \langle a \rangle \Phi$$

zur Untersuchung des Wahrheitsgehaltes der Aussage notwendig, unter den initialen Ausführungsschritten von durch T repräsentierten Prozessen die Ereignisse a zu untersuchen. Die Problemstellung ist daher

> Gegeben ist ein Prozessterm T. Bestimme alle initialen Ausführungsschritte von dargestellten Prozessen sowie die zugehörigen Ereignisse und Darstellungen der Nachfolgeprozesse.

Es werden zwei Ansätze beschrieben, um Kalküle für Korrektheitszusicherungen der gezeigten Form zu konstruieren:

- Die Verwendung von expliziten Übergangszusicherungen

  $$P \xrightarrow{a} Q,$$

  welche die Existenz eines bestimmten Zustandsübergangs (beschriftet mit dem Wert von a) zwischen den von den Termen P, Q dargestellten Prozessen ausdrücken. Die Verwendung von Formeln dieser Art erlaubt das direkte Übernehmen von Regeln einer strukturellen operationalen Semantik (von CSP) in ein Beweissystem, und damit einen top-down-Ansatz für die Erforschung des Verhaltens eines Prozesses.

- Die Formulierung von Termersetzungssystemen zur Überführung von Prozesstermen in bestimmte Normalformen, anhand derer das Verhalten dargestellter Prozesses durch syntaktische Analyse festgestellt werden kann. Daraus ergibt sich eine bottom-up-Analyse von Prozesstermen: Es werden rekursiv zunächst Unterterme auf Normalform gebracht, und anschließend werden die normalisierten Unterterme entsprechend der Semantik des verknüpfenden Operators kombiniert.

Beide Ansätze führen zu Kalkülen denen bestimmte Nachteile anhaften:

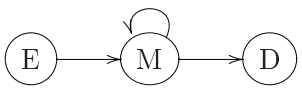
- Es werden viele Kopien von Untertermen eines Prozessterms erzeugt, was insbesondere die interaktive Benutzung erschwert.
- Ausführungsschritte des Prozesses werden total geordnet, und es wird ein Berechnungsbaum des Prozesses, d.h. ein Zustandsübergangssystem in Form eines Baums, ermittelt. Dadurch geht Information des Prozessterms über Unabhängigkeit von bestimmten Berechnungsschritten verloren.

Als Alternative zu den beschriebenen Arten von Kalkülen wird eine Analyseprozedur für CSP-Prozessterme beschrieben, bei der diese unerwünschten Eigenschaften durch Verwendung einer Erweiterung von CSP vermieden werden. Die Prozedur entspricht zu einem gewissen Grad der Bestimmung von Abwicklungsnetzen bei Petrinetzen. Die CSP-Erweiterung und die Methode der Prozesstermanalyse werden in den folgenden Kapiteln näher erläutert.

4. Erweiterung von CSP um P/T-Netz-Kausalität

Es wird eine Erweiterung von CSP eingeführt, in der zusätzliche Operatoren Kausalzusammenhänge zwischen verschiedenen Teilen eines Prozessterms herstellen können. Der verwendete Kausalitätsbegriff lehnt sich an Stellen-Transitions-Netze (P/T-Netze) an.

Um eine Basis für günstigere algebraische Eigenschaften zu schaffen, wird eine eingeschränkte Variante von 1-sicheren P/T-Netzen definiert, so genannte monotone P/T-Netze. Die Netze zeichnen sich durch eine modifierte Stellensemantik aus, bei der eine einzelne Stelle drei verschiedene Werte annehmen kann (im Gegensatz zu den zwei möglichen Werten ‚leer' und ‚markiert' bei normalen 1-sicheren Netzen): ‚leer'(E), ‚markiert' (M) und ‚tot' (D). Die Entwicklung der Wertebelegung einer Stelle während der Ausführung eines monotonen P/T-Netzes wird beschrieben durch den Zustandsgraph



Die wesentliche Eigenschaft des Graphs ist das Fehlen von (nicht-trivialen) Zyklen, was sich überträgt auf den Zustandsgraphen eines monotonen P/T-Netzes im Ganzen. Dementsprechend ist es möglich, für die Markierung (d.h. die Abbildung, die jeder Stelle eines Netzes ihren Wert zuordnet) eines kompletten Netzes weitgehende Abschätzungen über Veränderungen bei Ausführung des Netzes zu formulieren.

Das Konzept monotoner P/T-Netze wird in CSP integriert (die Erweiterung wird CCSP genannt) durch folgende Anpassungen:

- Für Prozessdarstellungen als Zustandsübergangssysteme wird bei den Beschriftungen von Übergängen ein spezielles Format verwendet: Die Beschriftungen sind Tripel (m, e, n), bestehend aus zwei endlichen Mengen $m, n \subseteq S$ von Stellen (aus einer global festgelegten Menge S) und einem CSP-Ereignis e. Entsprechend der kategoriellen Konstruktion von Kap. 2 ergibt sich daraus ein reichhaltigerer Prozessraum $CProc_{\Sigma_0,S}$ als Trägermenge eines finalen Zustandsübergangssystems, in das sich der ursprüngliche CSP-Prozessraum $Proc_{\Sigma_0}$ kanonisch einbetten lässt.

- Die Einführung von zusätzlichen Operatoren:

${}^{o}P$:	Verbrauch von Marken
${}^{o*}P$:	Verbrauch von Marken bei sichtbaren Ereignissen
${}^{o}a^{r}$:	Transitionen
$p[v] : P$:	Festlegen des Wertes einer Stelle

Die ersten beiden Operatoren entsprechen (in etwa) Eingangsstellen einer Transition in einem P/T-Netz, und konstruieren Prozesse deren erster Ausführungsschritt (bzw. deren erster sichtbarer Ausführungsschritt) nur unter Verbrauch von Marken aus den angegebenen Stellen o vollzogen werden kann.

Transitionen ${}^{o}a^{r}$ haben weitgehend dasselbe Verhalten wie Transitionen eines P/T-Netzes, d.h. bei Feuern werden Marken von Stellen o konsumiert, und Marken für Stellen r produziert.

Der kausale Zustandsoperator $p[v] : \cdot$ legt den Wert einer Stelle p auf v fest, und beschreibt darüberhinaus die Stellensemantik, d.h. den obigen Zustandsgraphen für Werte von Stellen.

- Der Operator für Parallelkomposition wird verändert, um die Realisierung der Prozesstermanalyse zu erleichtern. Die neue Verknüpfung hat die Form

$$P \,\|[\, L[X]R \,]\|\, Q$$

wobei P, Q, X dieselbe Bedeutung wie beim ursprünglichen CSP-Operator haben, während die endlichen Multimengen L, R von Tripeln (m, e, n) (wie im ersten Punkt) bei der Parallelverknüpfung von Transitionen als Puffer verwendet werden (genauer erläutert im nächsten Kapitel).

Wie in Kap. 2 wird zur formalen Definition der neuen und veränderten Verknüpfungen von CCSP auf dem Prozessraum $CProc_{\Sigma_0,S}$ das Koinduktive Definitionsverfahren verwendet.

5. Algebraische Eigenschaften von CCSP

Als Basis für die Definition eines Termersetzungssystems zur Prozesstermanalyse werden in diesem Kapitel algebraische Eigenschaften der CCSP-Operatoren erarbeitet. Die grundlegende Zielsetzung dabei ist die Formulierung von Gleichungen, welche die Auflösung eines beliebigen Prozessterms in durch Interleaving verknüpfte Transitionen ermöglichen, d.h. die Transformation in die Form

$$ {}^{o_1}a_1{}^{r_1} \;|||\; \cdots \;|||\; {}^{o_k}a_k{}^{r_k} \qquad (*) $$

Prozessterme dieser Art können ohne Schwierigkeiten syntaktisch untersucht werden.

5.1. Freie Stellen von Prozessen

Als wesentliche Voraussetzung für spätere Lemmata wird der Begriff von freien Stellen eines Prozesses $P \in \mathit{CProc}_{\Sigma_0,S}$ eingeführt, welcher diejenigen Stellen $FP(P) \subseteq S$ auszeichnet, die in Beschriftungen (m, e, n) des Zustandsübergangsgraphen von P auftauchen. Die Konzept derartiger Stellen ist wichtig, da genau für freie Stellen $p \in FP(P)$ die Anwendung des Kausalitätsoperators $p[v] : \cdot$ auf einen Prozess P eine verändernde Wirkung hat, oder umgekehrt ausgedrückt: Ein Prozess ist genau dann Fixpunkt von $p[v] : \cdot$, wenn die Stelle p im Prozess nicht frei vorkommt.

Freie Stellen sind vergleichbar mit freien Variablen in Termen oder Formeln. Die Bindung einer solchen Variablen, beispielsweise durch Quantoren, entspricht der Bindung einer Stelle durch den Kausalitätsoperator.

5.2. Grundlegende Eigenschaften der CCSP-Operatoren

- Versehen mit dem CCSP-Operator $\Box$ für Auswahl aus Alternativen bildet der Prozessraum $\mathit{CProc}_{\Sigma_0,S}$ einen kommutativen Monoiden.
- Der Operator für Parallelkomposition $\|[L[X]R]\|$ ist symmetrisch und – unter bestimmten Bedingungen – assoziativ.
- Der Kausalitätsoperator $p[v] : \cdot$ kann mit den übrigen CCSP-Operatoren, sowie mit sich selbst, kommutiert werden, vorausgesetzt dass dabei keine Kollisionen auftreten.
- Verschachtelte Auftreten der Operatoren für den Verbrauch von Marken können kontrahiert oder kommutiert werden.

- Prozesse, die beschrieben sind durch die Anwendung der Operatoren für die Benennung von Prozessen $l : \cdot$, das Verbergen von Ereignissen $\cdot \setminus X$ oder den Verbrauch von Marken ${}^{o}(\cdot)$, ${}^{o}*(\cdot)$ auf Transitionen ${}^{o}a^{r}$ sind entweder inaktiv, oder können wiederum durch Transitionen dargestellt werden.

5.3. Elimination von Präfix und Auswahl aus Alternativen

Im Gegensatz zu CSP sind die Verknüpfungen für das Hinzufügen von Präfixen $a \rightarrow \cdot$ und die Auswahl aus Alternativen $\Box$ in CCSP nicht unabhängig, sondern können simuliert werden unter Verwendung von P/T-Netz-Kausalität. Diese Beobachtung wird im Wesentlichen durch die folgenden Gleichungen beschrieben:

$$\begin{array}{lcll} a \rightarrow P & = & p[\mathrm{E}] : ({a}^{\{p\}} \;|||\; {}^{\{p\}}P), & p \notin FP(P) \\ P \,\Box\, Q & = & p[\mathrm{M}] : ({}^{\{p\}}*P \;|||\; {}^{\{p\}}*Q) & p \notin FP(P) \cup FP(Q) \end{array}$$

Beide Aussagen erfordern das Einführen einen „neuen“ Stelle p, d.h. einer Stelle, die in den beteiligten Prozessen nicht frei vorkommt. Für eine effizientere Handhabung von Vorkommen der Operatoren für den Verbrauch von Marken können die gegebenen Umformungen noch verfeinert werden.

5.4. Hilfsmittel für Transformation in Normalform

Die folgenden drei Abschnitte ermöglichen die rekursive Transformation von Prozesstermen in die Form (*). Dazu wird jeweils davon ausgegangen, dass ein Unterterm eines der verbliebenen CCSP-Operatoren in der Form $P := {}^{o}a^{r} \;|||\; Q$ vorliegt, d.h. dass durch bereits erfolgte Transformationen eine Transition abgespalten wurde. In dieser Situation kann die Transition von P einen weiteren Schritt nach außen gezogen werden:

5.4.1. Abspalten von Transitionen aus Parallelismus

Falls P als Argument des Operators für Parallelkomposition auftritt, d.h.

$$A \,[\![\, L[X]R \,]\!]\, ({}^{o}a^{r} \;|||\; Q),$$

dann kann die Transition ${}^{o}a^{r}$ im Falle von Interleaving direkt vor den Parallelismusoperator gezogen werden. Andernfalls (${}^{o}a^{r}$ repräsentiert ein Ereignis, das die Synchronisation mit dem Prozess A erfordert) werden die Puffer L, R verwendet, um eine systematische Verschmelzung der Transition mit dem Gegenprozess A zu erreichen. Dies erfolgt in zwei Schritten:

- Der Puffer L enthält alle Transitionen aus A, für die das beschriebene Verfahren bereits angewendet wurde. ${}^{o}a^{r}$ wird mit allen kompatiblen Elementen (m, a, n) aus L zu einer neuen Transition ${}^{m \cup o}a^{n \cup r}$ verschmolzen, welche aus dem Skopus des Operators für Parallelkomposition herausgezogen wird.

- Damit ${}^{o}a^{r}$ mit noch nicht ermittelten Transitionen von A kombiniert werden kann, wird es als Tripel (o, a, r) dem Puffer R hinzugefügt.

Insgesamt erhält man die Gleichung

$$A \,[\![\, L[X]R \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q) \quad = \quad T \;|||\; \Big(A \,[\![\, L[X]R' \,]\!]\, Q\Big)$$

wobei T eine Liste von Transitionen (entsprechend dem auftretenden Fall, und verknüpft durch Interleaving) und $R' \in \{R, R \uplus \{(o, a, r)\}\}$ der neue Wert des rechten Puffers ist.

5.4.2. Abspalten von Transitionen aus den Operatoren für Benennung von Prozessen und Verbergen von Ereignissen

Da für die CCSP-Operatoren für die Benennung von Prozessen $l : \cdot$ und das Verbergen von Ereignissen $\cdot \setminus X$ Distributivgesetze im Verbindung mit Interleaving gelten, lässt sich ihre Anwendung auf das Argument P ohne Schwierigkeiten durchführen:

$$\begin{aligned} l : ({}^{o}a^{\,r} \;|||\; Q) \quad &= \quad (l : {}^{o}a^{\,r}) \;|||\; (l : Q) \\ ({}^{o}a^{\,r} \;|||\; Q) \setminus X \quad &= \quad ({}^{o}a^{\,r} \setminus X) \;|||\; (Q \setminus X) \end{aligned}$$

Aufgrund der in Abschnitt 5.2 festgestellten Eigenschaften lassen sich die resultierenden Ausdrücke $l : {}^{o}a^{\,r}$ und ${}^{o}a^{\,r} \setminus X$ zu Transitionen oder zu *stop* umformen.

5.4.3. Abspalten von Transitionen aus den Operatoren für Verbrauch von Marken

Den schwierigsten Fall stellen die CCSP-Operatoren für den Verbrauch von Marken ${}^{o}(\cdot)$, ${}^{o*}(\cdot)$ dar, da die notwendigen Umformungen für das Abspalten einer Transition ausgehend von

$${}^{m}({}^{o}a^{\,r} \;|||\; Q), \qquad {}^{m*}({}^{o}a^{\,r} \;|||\; Q),$$

im Allgemeinen eine globale Wirkung haben. Bei der durchgeführten Modifikation ändert sich zwangsläufig auch der Wert der dargestellten Terme, weshalb zum Ausgleich ein umgebender Kontext angepasst werden muss. Das Abspalten einer Transition wird durchgeführt durch die Transformationen ($q \in S$ ist eine unbenutzte Stelle)

$$\begin{aligned} {}^{\{p\}}({}^{o}a^{\,r} \;|||\; Q) \quad &\rightsquigarrow \quad {}^{\{p\}}({}^{o}a^{\,r}) \;|||\; {}^{\{q\}}Q \\ {}^{\{p\}*}({}^{o}a^{\,r} \;|||\; Q) \quad &\rightsquigarrow \quad {}^{\{p\}*}({}^{o}a^{\,r}) \;|||\; {}^{\{q\}*}Q \end{aligned}$$

und muss durch die Verdopplung der Stelle p (durch Hinzufügen von q) in einem Kontext des Terms ausgeglichen werden. Diese Verdopplung wird durch Termersetzung beschrieben, und führt beispielsweise auf

$$p[v] : {}^{\{p\}}({}^{o}a^{\,r} \;|||\; Q) \quad = \quad p[v] : q[v] : \Big({}^{\{p\}}({}^{o}a^{\,r}) \;|||\; {}^{\{q\}}Q\Big)$$

6. Transformation von CCSP-Termen in Normalform

Unter Verwendung der in Kap. 5 beschriebenen Eigenschaften wird ein Termersetzungssystem definiert für die Umformung von CCSP-Termen auf gewisse Normalformen, anhand derer Aussagen über die Zustandsübergangssysteme dargestellter Prozesse gemacht werden können. Dazu wird zunächst eine formale Syntax für CCSP-Terme eingeführt, welche im Wesentlichen induktiv aus den Operatoren aus Kap. 4 aufgebaut werden. Für die Beschreibung von Ereignissen (das Argument a in $a \rightarrow P$), der Werte von Stellen (Argument v in $p[v] : P$) und von Schnittstellenmengen (Argument X in $P \,|[\, L[X]R \,]|\, Q$) wird die Existenz von passenden Termalgebren vorausgesetzt, wobei die genaue Wahl dieser Algebren (mit einigen Einschränkungen, die weiter unter ausgeführt sind) keine Rolle für das weitere Verfahren spielt. Für eine feste Wahl der Termalgebren wird die Menge der CCSP-Prozessterme mit $Proc_S$ bezeichnet, und ist parametrisiert mit einem Vokabular S von Stellen.

Die Auswertung von CCSP-Prozesstermen wird induktiv definiert als Fortsetzung von entsprechenden Auswertungsfunktionen der zugrundeliegenden Termalgebren für Ereignisse, Schnittstellenmengen und Stellenbelegungen. Die CCSP-Symbole werden dabei interpretiert mit den in Kap. 4 definierten zugehörigen Verknüpfungen auf dem Prozessraum $CProc_{\Sigma_0,S}$.

Die Regeln des Termersetzungssystems für die Normalisierung von Prozesstermen entsprechen größtenteils den in Kap. 5 aufgestellten Gleichungen. In einigen Punkten ist die Modifikation der Gleichungen nötig, um der Unentscheidbarkeit von semantisch formulierten Bedingungen Rechnung zu tragen, zum Beispiel der Gleichheit des Wertes zweier Terme (solche Bedingungen treten bei der Abspaltung von Transitionen aus Parallelismus auf). Zu diesem Zweck werden Fallunterscheidungen mit Hilfe des Kausalitätsoperators eingeführt, und es wird entsprechend die Existenz geeigneter Ausdrücke für den Wert von Stellen verlangt. Eine solche Fallunterscheidung ist beispielsweise

$$p[e \doteq e'] : {}^{\{p\}} P$$

wobei der Ausdruck $e \doteq e'$ die Gleichheit der Werte der Terme e, e' darstellt. Je nach Auswertung von $e \doteq e'$ wird die Stelle p mit verschiedenen Werten belegt, und dadurch die Ausführung des durch P beschriebenen Prozesses ermöglicht oder blockiert.

Im einzelnen enthält das Termersetzungssystem die folgenden Gruppen von Regeln:

- Elimination von Präfix und Auswahl aus Alternativen
- Ordnen von mit Interleaving verknüpften Prozessen: Es werden Terme der Form

$$^{o}\, a \,^{r} \;|||\; Q$$

hergestellt, die die Voraussetzung für die Anwendbarkeit von Regeln anderer Gruppen ist

- Herausziehen von Kausalitätsoperatoren: Dadurch werden Terme auf eine Pränexform
$$p_1[v_1] : p_2[v_2] : \ldots p_k[v_k] : P'$$
(wobei P' keine Kausalitätsoperatoren mehr enthält) gebracht
- Duplizieren von Stellen: Hilfsregeln für das Abspalten von Transitionen aus den Operatoren für Verbrauch von Marken
- Anwendung von verschiedenen Operatoren (z.B. Benennung von Prozessen) auf Transitionen
- Abspalten von Transitionen aus Parallelismus
- Abspalten von Transitionen aus den Operatoren für Benennung von Prozessen und Verbergen von Ereignissen
- Abspalten von Transitionen aus den Operatoren für Verbrauch von Marken.

6.1. Analyse des Termersetzungssystems

Die Anwendung des Termersetzungssystems terminiert in jedem Fall, und führt grundsätzlich zu Termen der Form
$$r_1[\Phi_1] : \cdots r_l[\Phi_l] : \Big(T \;|||\; R \Big)$$
wobei Term T nur aus durch Interleaving verknüpften Transitionen besteht, und R weder Transitionen noch Kausalitätsoperatoren enthält. Aus einem Term dieser Form lassen sich Rückschlüsse auf Zustandübergänge von dargestellten Prozessen ziehen:

- T stellt (zumindest einen Teil) aller möglichen ersten Ausführungsschritte des Prozesses dar. Ob Transitionen von T feuern können, hängt ausschließlich von der Belegung beteiligter Stellen $r_1, \ldots, r_l$ ab, welche wiederum von den Ausdrücken $\Phi_1, \ldots, \Phi_l$ bestimmt wird. Damit ist die Belegung im Allgemeinen nicht berechenbar, was aber durch Fallunterscheidungen und das Hinzuziehen von geeigneten Kalkülen umgangen werden kann. Es ergibt sich eine Menge möglicher Markierungen der Stellen, die jeweils versehen sind mit Bedingungen an die Werte von $\Phi_1, \ldots, \Phi_l$.
- Wird eine konkrete Wertebelegung von $r_1, \ldots, r_l$ betrachtet, so lassen sich die aktivierten Transitionen von T berechnen. Ob dadurch das gesamte potentielle Verhalten des Prozesses im ersten Schritt erfasst ist, hängt vom Restterm R ab. Es lassen sich hinreichend schwache syntaktische Kriterien dafür formulieren, dass R für eine bestimmte Wertebelegung von Stellen keine Berechnungsschritte durchführen kann, d.h. inaktiv ist. Wenn das der Fall ist, werden initiale Schritte des Prozesses ausschließlich von T verursacht, und lassen sich damit syntaktisch bestimmen.

Das Verfahren für die Analyse von Prozesstermen lässt sich in den in Kap. 3 beschriebenen Rahmen für Kalküle für Korrektheitszusicherungen einbetten, und hat gegenüber den dort beschriebenen Ansätzen der Vorteil, dass bei der Termumformung keine Totalordnung vor Berechnungsschritten vorgenommen wird, und dass Redundanzen durch Vervielfachen von Teiltermen vermieden werden.

7. Zusammenfassung

Es wurden verschiedene Aspekte eines Beweissystems für die Verifikation von JCSP-Programmen behandelt, insbesondere wurde ein Verfahren für die Analyse von CSP-Prozesstermen beschrieben, das als Teil eines Kalküls für die symbolische Ausführung von JCSP-Systemen verwendet werden kann. Zu diesem Zweck wurde eine Erweiterung von CSP um kausale Konzepte ähnlich denen von Petrinetzen eingeführt und untersucht.

Die beschriebenen Teile des Beweissystems für JCSP wurden komplett implementiert, und die behandelten Methoden sind größtenteils aufgrund von Erfahrungen mit der Implementierung entstanden. Punkte die weitere Arbeit erfordern sind unter anderem:

- Vervollständigung der unterstützten Klassen von JCSP
- Betrachtung von mächtigeren Logiken, möglicherweise des modalen μ-Kalküls.

Part II.

Thesis

1. Introduction

The KeY project [ABB+04], which is joint project of the University of Karlsruhe, the Chalmers University of Technology, Gothenburg, and the University of Koblenz, aims at the development of tools that enable the formal specification and verification of software. The central component that is worked on is a prover tailored to the verification of programs written using the JavaCard language [Sun02] (a variant of the Java language [GJSB00] for smart card software), which is supposed to be usable both for interactive and for automated proving. The underlying logic is a variant of dynamic logic for JavaCard [Har84, Bec01].

At the time, the KeY system does not allow the treatment of concurrent programs, but is restricted to purely sequential code. This is consistent with the features of the JavaCard language, which does not contain multi-threading (in contrast to Java); nevertheless, the (almost) complete treatment of sequential JavaCard programs—and of many features of Java—by the KeY prover makes the addition of support for concurrent programs a promising challenge.

This thesis describes aspects of one approach to this end, in which the JavaCard language is augmented with high-level primitives of concurrency that strongly resemble the Communicating Sequential Processes (CSP) model [Hoa85]. The starting point therefore is the Java Communicating Sequential Processes (JCSP) library [WA, PJ00], which is a library of Java classes that enable the development of concurrent programs. Unlike the native means of communication between threads that are present in the full-featured Java language, which are essentially shared memory and synchronisation by monitors, communication in JCSP is based on message passing between processes. The concepts used in this respect are very similar to the Occam language [Bar92].

Though in practice JCSP is implemented by using the existing multi-threading concept of Java, and though the communication primitives of JCSP internally use shared memory for passing data, programs that exclusively employ JCSP for establishing concurrent behaviour can equivalently be described within a higher-level process model. In this model, for instance address spaces of different processes are regarded as strictly separated, and processes can interact only through well-defined primitives of JCSP. As the model differs from the real implementation of JCSP, i.e. threads—which are used for realising JCSP processes—are in Java actually able to access memory regions of other threads, for programs that are supposed to be represented using the higher-level process model it has to be ensured that no such "illegal" operations are performed. This could be done by the programmer or by a static approximation of the program behaviour. A further possibility would be a more rigid implementation of JCSP (that is consistent with the same process model though).

When concentrating on the verification of such "well-formed" JCSP programs (i.e. on a subset of the domain of Java programs), instead of referring to the general thread model of the Java language the higher-level process model realised by JCSP can be

used as semantic basis. This has the advantage of delegating certain aspects of the verification to the well-formedness condition, i.e. either to the programmer or to a static analysis ensuring that communication between processes is only performed using the JCSP library. Consequently, the correctness of inter-process communication itself does not have to be proved as part of the actual verification, which can concentrate on the more essential matter of showing that the behaviour of a concurrent system—built using primitives whose correctness is guaranteed—satisfies certain specifications. We believe that this approach makes verification more efficient, and thus enables the treatment of more complex systems than the consideration of the full range of concurrent Java programs.

The basic idea that was followed when constructing a proof system for JCSP programs was to model the elements of JCSP by using operators of a generic process algebra. Because of the relation between JCSP and the CSP algebra, as a natural choice the latter algebra was used for this purpose; nevertheless, the same construction could be carried out using other process algebras. Because the KeY prover already provides an axiomatisation of the operational semantics of sequential JavaCard programs, which can—with little effort—also be applied to the different situation of JavaCard code that is embedded in process terms of the CSP algebra, a calculus for concurrent programs can be obtained essentially by adding formalisations of the JCSP classes and rules for CSP. The present thesis mostly describes a CSP calculus that has been found to be particularly suited therefore.

1.1. Calculi for CSP

The approach of using the CSP algebra for modelling JCSP classes leads to terms in which CSP operations are mixed with Java code. For instance the parallel composition of two JCSP processes described by Java programs α and β is represented by a term like

$$T(\alpha) \mid\mid\mid T(\beta)$$

in which T is a symbol that denotes the interpretation of Java code as a process of the CSP world, and $\mid\mid\mid$ is the CSP operation of interleaving.

The analysis of sequential Java code is by the KeY system performed according to the symbolic execution paradigm. This essentially means that the rules of the underlying calculus implement an operational semantics of Java, and that the verification of program properties, i.e. the creation of a proof for a formula of dynamic logic, can be regarded as the execution of the program for unspecified memory contents.

For the verification of JCSP systems, the symbol execution method is carried over to CSP containing Java fragments. This execution of a process term thus consists of

- the symbolic execution of concerned Java code within the term, using the existing calculus for sequential programs, and
- the consideration of process behaviour in combination, i.e. the evaluation of the CSP operators as soon as the execution of sequential Java code yields results that are relevant for the behaviour of the system as a whole.

When implementing the verification concept, we have found that for the calculus part treating CSP operators primarily the following characteristics are important:

- Using a single process term for assembling basic processes that are described by Java code leads to formulas that become quite large, because for each particular Java process both the executed source code and current memory contents have to be stored. Hence, it is essential that the CSP rules do not introduce redundancy, for instance by duplicating subterms. The absence of such redundancy is also necessary to obtain a calculus that can be used interactively.

- In practice, when verifying a JCSP system, the majority of rule applications for symbolic execution treats sequential Java code. It is reasonable to spend some effort designing CSP rules that postpone branching of a proof as far as possible to avoid duplicated Java execution thus.

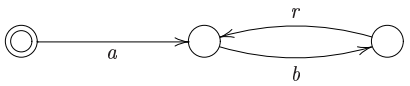
Unfortunately, the sole usage of the CSP algebra has proved to be insufficient when designing a calculus that meets these requirements, because the concepts of the CSP operators enforce a fairly "rigid" representation of JCSP networks, and requires early case distinctions. For this reason, the CSP calculus given in this thesis is based on an extension of CSP along the lines of [BB01], which describes a combination of traditional process algebra with Petri nets.

1.2. Propositions about Processes

Requirements on formalisms for specifications about concurrent systems are different from those that can be formulated when treating purely sequential systems. Common choices for the latter case are Hoare Logic [Hoa69] or Dynamic Logic [Har84], both of which allow propositions about programs to be made that (semantically) refer to the input/output relation of the program, i.e. about the relation between the states (the memory content of a machine model) in which the program execution is started and in which it terminates. Because concurrent systems are often systems whose normal behaviour is non-terminating, and because for the composition of concurrent systems from simpler components it is usually necessary to consider not only input/output, but also the behaviour of components during execution, this approach is not appropriate for concurrent systems.

A common way to give specifications about (concurrent) systems, which is also the approach this report concentrates on, is to interpret the behaviour of the system as a *process*. Processes are in this thesis semantically regarded as (usually infinite) state machines, and are described by a set of states, a labelled transition relation on this state space and a distinguished initial state (the exact definition is given in Sect. 2.2.1). An example of a process is given as a diagram in Fig. 1.1. Labels of transitions are regarded as the communication of the process with an (unknown) environment, and make up the behaviour of the process that is observable. Because each node of the graph can be regarded as an initial state, the given diagram actually describes three different processes that all have the same state space and the same transition relation. This idea is carried on in the sequel by fixing a certain state space and a transition

Figure 1.1. Example Process Consisting of Three States



relation; within this context, processes can then be described uniquely by choosing an initial state. The chosen state space is hence also called *process space*.

The representation of systems using the process framework includes the following decisions:

- The distinction of interesting communication aspects of the system as potential transition labels, i.e. of those aspects that are regarded as relevant for making proposition about the behaviour of a system, or that are important for the embedding of the system in an environment,
- the construction of a suitable state space based on possible (internal) states of the system and
- the encoding of dynamic behaviour as the transition relation on the state space, for instance as the relation describing atomic execution steps of the system (operational semantics).

There are different approaches for making propositions about processes of the described kind, i.e. for distinguishing certain sets of processes that adhere to particular properties. Perhaps the two methods (that for certain situations are also equivalent) used most often are

- The definition of a *refinement* relation on the set of processes, which is a preorder that relates processes whose behaviour is regarded as similar, but the "accuracy" of whose behaviour is different. Using this approach, specifications of systems are again processes; to compare the specification with an implementation, it is checked whether the processes are related by the preorder (the implementation is required to have a behaviour that is at least as precise as the specification). Details that can be neglected in the specification include internal computation steps and determinism.
- The specification of properties using formulas of *modal or temporal logics*. These formalisms can be employed for propositions about processes, because the state graphs with labelled transitions can equivalently be regarded as Kripke frames of multi-modal logics (see for instance [Sti92]), in which such formulas can be evaluated.

While the methods described in this thesis primarily apply to specifications using modal formulas, i.e. to the second item, most probably it is possible to treat refinement verification in a similar way. We also concentrate on a simple modal logic, namely Hennessy-Milner-Logic [HM80], which can be regarded as the basis for more powerful logics like CTL [CE82] or the μ-calculus (an introduction to this logic is [BS01]).

1.3. Related Work

- Numerous projects and articles treat the verification of temporal or modal properties for processes, both using model checking and deduction. See for instance different parts of [BPS01].
- The EVT system [ACD+03, Fre01] is a very complete proof system for deductive verification of Erlang programs [AVWW96] based on the μ-calculus. Several ideas for the JCSP proof system, in particular the approach of verification using a higher-level process model than the Java thread model, originate from discussions with members of the EVT project.
- The combination of process algebra and Petri nets, or the definition Petri net semantics for process algebras, has a long tradition (the following list is by no means complete):
  - [Gol87], which treats the process algebra CCS, and describes translation of CCS processes to Petri nets
  - [BDK01, BDK02] relates process algebras with Petri nets, and in particular describes the Petri Box Calculus, which is a process algebra that is primarily equipped with Petri net semantics
  - [BB01], which describes a process algebra that is enriched with concepts of Petri nets, and on which parts of this thesis are based.

  To our knowledge, however, a procedure similar to the lazy symbolic execution of CSP processes using Petri nets, which is described in this thesis, has not yet been formulated.

1.4. How the Thesis is Organised

Sect. 2 describes and defines the considered fragments of the CSP process algebra and the JCSP library, and introduces basic notions about processes that are used throughout the thesis

Sect. 3 introduces Hennessy-Milner-Logic as a simple formalism for specification of processes, and describes two approaches for calculi treating CSP in combination with Hennessy-Milner-Logic

Sect. 4 defines an extension of the CSP algebra, in which CSP is enriched with further operators that resemble Petri nets, and which is later used as the basis of a CSP calculus

Sect. 5 collects properties of the CSP extension given in Sect. 4, and describes how derived algebraic laws can be used to obtain normal forms of process terms

Sect. 6 defines a formal syntax of the CSP extension of Sect. 4, and gives a term rewriting system that analyses process terms by deriving normal forms. The rewriting system is used to formulate calculi for the CSP algebra and Hennessy-Milner-Logic.

2. Communicating Sequential Processes

This section gives an introduction to a fragment of the communicating sequential processes algebra (CSP, [Hoa85]). A selection of the most important CSP operators is described and formally defined. CSP is at this point regarded as a concrete algebra, i.e. we define its operators as mappings on a certain process space.[1] In order to construct this space, and to provide means of reasoning about processes, basic concepts of coalgebra are introduced.

The end of the section sketches the usage of CSP for representing the process model realised by the JCSP library.

2.1. The Considered Fragment of CSP

In the following pages, we define the CSP algebra to be a set of *processes*—a process space—that is equipped with two structures:

- A *coalgebra* structure, which is given by a labelled transition relation, and which defines the dynamic behaviour of processes (processes can, as in the introduction, be seen as state machines). This structure is described in detail in Sect. 2.2.1.

- An *algebra* structure, which is made up from the CSP connectives that are described below. As the connectives are defined referring to the transition relation (the coalgebra structure), the algebra structure can be regarded as secondary.

Comparing CSP with other process algebras, CSP consists of a particularly rich set of operators, many of which are in fact derived and can be simulated using more basic ones. While this large vocabulary makes CSP suited for specifications beyond mere toy applications, the number of operations is of course reflected by the number of rules a calculus for CSP consists of. For the more theoretical considerations of this thesis, we therefore stick to a fragment of CSP, which is large enough, however, to observe important aspects of CSP calculi, and which is also sufficient to model most parts of the JCSP library. Nevertheless, it has been found that the efficiency of a CSP-based proof system for JCSP programs can be substantially improved by the usage of higher-level CSP operators, which should thus also be supported by an implementation of a CSP calculus that is supposed to be used in practice.

[1] A corresponding term algebra—whose elements are constructed according to a formal syntax of process terms—is for an extension of CSP defined in Sect. 6.

The CSP operators we are going to define are (we mainly use the notations of [Ros98]):

$$\begin{aligned} stop: &\quad \text{Blocked Process} \\ a \rightarrow P: &\quad \text{Prefixing} \\ P \,\Box\, Q: &\quad \text{Choice} \\ c\,?\,x \rightarrow P: &\quad \text{Prefix Choice} \\ P \,[\![X]\!]\, Q: &\quad \text{Parallelism} \\ l : P: &\quad \text{Labelling} \\ P \setminus X: &\quad \text{Hiding} \end{aligned}$$

The set of CSP processes is parametrised by an alphabet Σ of events, which is supposed to be non-empty and fixed a priori. This notion of an alphabet differs from the original CSP definition by Hoare, which assigns alphabets not to the space of processes as a whole, but to single processes. As a consequence thereof, also the operators of parallelism in [Hoa85] follow a slightly different concept than pursued in [Ros98], whereby in our experience the former one is less intuitive and more difficult to handle by calculi. Thus, we stick closely to [Ros98] in this respect.

The operators for labelling and hiding modify the communication behaviour of a process by exchanging the events the process is able to engage in. Both functions are derived from corresponding mappings of the set Σ of events:

- We distinguish between atomic and composed events, and therefore we require that the alphabet Σ is given the structure of a free semi-group with an arbitrary (non-empty) basis $\Sigma_0 \subseteq \Sigma$. The elements of the basis are regarded as the *atomic events* of Σ. Depending on the situation, composed events are written using one of the following notations (both terms actually describe the same event):

  $$\begin{aligned} c.a \quad &\ldots \quad \text{an event (without indicating a direction of information flow)} \\ c\,!\,a \quad &\ldots \quad \text{sending of a value } a \text{ through a channel } c \end{aligned}$$

  To achieve a consistent notation, at times we also use the empty word ϵ as neutral operand of the event connective '.'; nevertheless, ϵ is not regarded as an independent event. Besides that, in some situations an exclamation mark is also added as a prefix to events as pure syntactic sugar:

  $$!\,a \quad = \quad a$$

- It is necessary to introduce a concept of *internal actions*[2] for defining the hiding operator. Contrary to [Ros98], in our variant of CSP there is not a single internal event τ, but instead for each visible event $a \in \Sigma$ we introduce a corresponding internal (invisible) event $\tau(a)$. The benefit of this non-standard convention (apart from some technical advantages in Sect. 2.2) is that in an interactive prover it will

[2] Other common names that can be found in literature are *silent steps* and *invisible actions*. A detailed treatment of silent steps can be found in [AFV01].

be easier to track the development of a CSP process term (to which, for instance, reduction rules are being applied) if information about events is not removed by calling all internal steps τ. According to our experience with the CSP calculi that are provided in this thesis, the usage of a single internal event does in fact make it almost impossible to understand the meaning of a process term whenever multiple internal steps are simultaneously turning up.

Formally, for each alphabet Σ of visible events we require Σ_τ to be a disjoint set of the same cardinality containing invisible events, and τ to be a bijection

$$\tau : \Sigma \to \Sigma_\tau$$

mapping visible events to invisible events.

The space of processes for a given set Σ_0 of atomic events is denoted by $Proc_{\Sigma_0}$; a formal definition of this set is given in Sect 2.2.2. Note that we introduce the following functions on a purely semantic (algebraic) level, i.e. the operators are not regarded as symbols from which terms are built, but as mappings according to their particular signature. In contrast, *process terms* are for an extension of the CSP algebra formally defined in Sect. 6.

2.1.1. The Fundamental CSP Operators

In the following pages a short and informal description of the operators of CSP is given, in particular w.r.t. non-standard modifications made in this thesis. For each operation we also provide the rules of its formal (Plotkin-style) structural operational semantics (SOS), on which all further reasoning is based.[3] These rules determine the connection between the algebra and the coalgebra structures of the process space. Algebraic laws are only given as far as they are important for the remainder of the thesis, a full account can for instance be found in [Hoa85, Ros98]. As our primary semantics of CSP is operational, equational laws are in any case derived and not necessary for a complete characterisation.

Blocked Process

$$stop$$

The process *stop* is the simplest of all processes, namely it is the process that is unable to perform any execution step at all. *stop* is mostly interesting as neutral element of some of the other CSP operators, or as "base case" when constructing more complex processes, as for instance

$$a \to b \to c \to stop$$

Prefixing

$$a \to P, \qquad a \in \Sigma, \quad P \in Proc_{\Sigma_0}$$

[3] The syntax of the rules conforms to [AFV01].

Following the standard semantics of prefixing, the process $a \to P$ is in its first step able to engage in the event a (and only in a), and afterwards behaves like P. As the described process is committed to the single event a, this engagement can also be regarded as the *sending* of a value a, encoded as an event, to the environment of the process. A complementary process that tries to read data from its environment can be obtained by applying the prefix choice constructor.

$$\frac{}{(a \to P) \xrightarrow{a} P}$$

(External) Choice

$$P \,\Box\, Q, \qquad P, Q \in Proc_{\Sigma_0}$$

The operator of external choice can be used to construct a process $P \,\Box\, Q$ that is able to behave either like P or like Q; the process enters into a commitment to one of the two possibilities when engaging in the first visible event that is raised by P or by Q (the standard semantics as in [Hoa85]). Internal events are interleaved and do not enforce commitment to one of the two subprocesses. The choice is called external, because an environment (added through the operator of parallelism, see below) which accepts only events offered by one of the two processes P, Q compels $P \,\Box\, Q$ to follow this decision.

$$\frac{P \xrightarrow{\tau(a)} P'}{P \,\Box\, Q \xrightarrow{\tau(a)} P' \,\Box\, Q} \qquad \frac{Q \xrightarrow{\tau(a)} Q'}{P \,\Box\, Q \xrightarrow{\tau(a)} P \,\Box\, Q'}$$

$$\frac{P \xrightarrow{a} P' \qquad a \in \Sigma}{P \,\Box\, Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q' \qquad a \in \Sigma}{P \,\Box\, Q \xrightarrow{a} Q'}$$

The further choice operators that can be found in [Hoa85] are not regarded as essential parts of CSP in this thesis:

- The restricted (external) choice connective

  $$a \to P | b \to Q$$

  is immediately subsumed by $\Box$. In [Hoa85], $|$ is introduced as an operator that is inherently related to the prefix operator $\to$, and that must only be applied to a tuple of subprocesses described using the prefix operator, provided that the first events raised by the subprocesses are distinct.

- The operator of internal (nondeterministic/demonic) choice

  $$P \sqcap Q$$

  which differs from external choice in the way the commitment to a particular subprocess is entered. While for external choice this decision is made by the environment of the process (at least provided that the subprocesses can be distinguished by looking at their first events), the internal choice between two processes

is made by the composed process itself in a nondeterministic way, not taking into account the events offered by the environment.

As the different semantics of internal and external choice cannot directly be modelled as state transition systems, in [Ros98] to this end (and only for defining the operational semantics of CSP) *internal actions* τ are introduced. The nondeterminism of internal choice can then be modelled by treating the entering of the commitment to either P or Q as an internal transition τ that can fire at any time regardless of communication with the environment.

A process with the same behaviour can be constructed using external choice together with the hiding operator: If a distinguished event $c \in \Sigma$ is given that does not turn up in normal situations, then by exploiting algebraic identities internal choice can be rewritten to

$$P \sqcap Q \quad \rightsquigarrow \quad (c \to P \sqcap c \to Q) \setminus \{c\} \quad = \quad (c \to P \,\Box\, c \to Q) \setminus \{c\}.$$

Prefix Choice

$$c\,?\,x \to P, \qquad c \in \Sigma \cup \{\epsilon\}, \quad P : \Sigma_0 \to Proc_{\Sigma_0}$$

As a generalisation of external choice, by using the prefix choice operator a process can be constructed that offers each atomic event $a \in \Sigma_0$ (labelled with a *channel name* c, provided that $c \neq \epsilon$), as its first possible step. The second parameter P of the operator determines the processes that describe the behaviour after the first step; P is a function that assigns each value $a \in \Sigma_0$ the corresponding continuation process $P(a)$. If Σ_0 is finite, then occurrences of prefix choice can equivalently be written using ordinary choice:

$$c\,?\,x \to P \quad = \quad c.v_1 \to P(v_1) \,\Box\, \cdots \,\Box\, c.v_n \to P(v_n) \qquad \Sigma_0 = \{v_1, \ldots, v_n\}$$

As the process is not committed to any particular member of Σ_0, but is willing to accept an arbitrary atomic event offered to it by an environment, prefix choice can be used to *receive* data encoded as an event. Two processes sending resp. receiving data can be integrated to form a communicating network by using the operator of parallelism, which is introduced below.

$$\frac{v \in \Sigma_0}{(c\,?\,x \to P) \xrightarrow{c.v} P(v)}$$

In literature about CSP (in particular in [Ros98]), the prefix choice operator is usually defined in a more general way. It is common practice to allow the embedding of $?\,x$ in complex event descriptions to combine multiple send or receive transactions in a single event; to restrict the range of values that can be received by a process, it is possible to use typed variables x by adding a set A of events:

$$c\,?\,x : A, \qquad c\,?\,x\,?\,y, \qquad c\,?\,x\,!\,v\,?\,y$$

In this document, the restricted variant of prefix choice is retained for reasons of simplicity, and because the extended possibilities are in our experience rarely needed. The

generalisation to more complex communication patterns should be straightforward, however.

Instead of writing the mapping P explicitly as a function—for instance using λ-notation—we usually regard the prefix choice operator as binding of a variable. For instance we write

$$c\,?\,msg \rightarrow c\,!\,msg \rightarrow stop \quad := \quad c\,?\,x \rightarrow \lambda\, y.\Big(c\,!\,y \rightarrow stop\Big)$$

Parallelism

$$P \,[\![\, X \,]\!]\, Q, \qquad X \subseteq \Sigma, \quad P, Q \in Proc_{\Sigma_0}$$

The central constructor of CSP is the connective that makes two processes run in parallel. The variant of this mapping given above is introduced as *generalised parallel* in [Ros98], where it is also found to subsume all other kinds of parallelism used in CSP. High-level operators defined by Hoare, like *subordination* or *pipes*, which combine parallelism with certain transformations of events, are left out in this document, but have been found to be a valuable extension to a practical CSP calculus enabling modelling that is both more concise and more efficient.

The set X of visible events is called the *interface* of the parallel composition shown above, and determines those events on which the processes P, Q have to agree, i.e. which can only be raised by P, Q simultaneously (*synchronised*). All other events are *interleaved* and can be engaged in independently by P, Q.

$$\frac{P \xrightarrow{a} P' \qquad a \notin X}{P \,[\![\, X \,]\!]\, Q \xrightarrow{a} P' \,[\![\, X \,]\!]\, Q} \qquad \frac{Q \xrightarrow{a} Q' \qquad a \notin X}{P \,[\![\, X \,]\!]\, Q \xrightarrow{a} P \,[\![\, X \,]\!]\, Q'}$$

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{a} Q' \qquad a \in X}{P \,[\![\, X \,]\!]\, Q \xrightarrow{a} P' \,[\![\, X \,]\!]\, Q'}$$

For the common case of pure interleaving of two processes we also use the abbreviation

$$P \,|||\, Q \quad := \quad P \,[\![\, \varnothing \,]\!]\, Q$$

corresponding to the native operator $|||$ of the original CSP algebra from [Hoa85].

Labelling

$$l : P, \qquad l \in \Sigma, \quad P \in Proc_{\Sigma_0}$$

The labelling operator can be used to structure a process network by adding a common prefix to all visible events a process engages in. While we use the standard semantics from [Hoa85], the more complex *multiple labelling* defined in the book is omitted in this thesis.

$$\frac{P \xrightarrow{a} P' \qquad a \in \Sigma}{l : P \xrightarrow{l.a} l : P'} \qquad \frac{P \xrightarrow{\tau(a)} P'}{l : P \xrightarrow{\tau(a)} l : P'}$$

It is also easy to generalise the labelling operator to arbitrary *renaming* determined by a binary relation on Σ, as it is done in [Ros98]; renaming also subsumes the hiding operator as it is defined below. Since the power of a general renaming operator is not needed for most modelling purposes, however, we only treat simple labelling.

In practice, labelling is mostly used to give server/slave processes of networks names by which they can be accessed. A common construction looks like

$$s : S \,[\![\, s.\Sigma \,]\!]\, C$$

in which S denotes a server process and C a client. C is able to communicate with S by prepending the prefix s to messages (the processes synchronise on such messages because of the operator $[\![s.\Sigma]\!]$), but can also refer to an environment without interaction with S through other events (in contrast to the server process). In the same way, multiple servers can be attached to one client:

$$s_1 : S \,[\![\, s_1.\Sigma \,]\!]\, \Big(s_2 : S \,[\![\, s_2.\Sigma \,]\!]\, C\Big) \qquad \text{or} \qquad \Big(s_1 : S \,|||\, s_2 : S\Big) \,[\![\, s_1.\Sigma \cup s_2.\Sigma \,]\!]\, C$$

Hiding

$$P \setminus X, \qquad X \subseteq \Sigma, \quad P \in Proc_{\Sigma_0}$$

Complementary to the labelling of processes, it is possible to structure a process network by hiding all events a process P engages in, provided that they are element of a given set X. We equip the hiding operator with a non-standard semantics differing from the definitions in [Hoa85, Ros98] that is in our opinion more appropriate for an (essentially) interactive calculus:

- As in the operational semantics in [Ros98], the hiding of an event is actually a renaming that replaces the original event with an internal action. This realisation is necessary to correctly represent CSP processes using transition systems, similarly to the situation above regarding the operator of internal choice. In order to avoid (intuitively) inconsistent algebraic and operational semantics, we have decided to generally regard hiding as renaming.
- The hiding operator renames each event $a \in X$ that is raised by the process P to the related internal action $\tau(a)$, i.e. only "marks" it as being internal. The advantages of keeping information about the original event are described in the beginning of the section.

$$\frac{P \xrightarrow{a} P' \qquad a \notin X}{P \setminus X \xrightarrow{a} P' \setminus X} \qquad\qquad \frac{P \xrightarrow{a} P' \qquad a \in X}{P \setminus X \xrightarrow{\tau(a)} P' \setminus X}$$

When attaching servers to clients, as shown in the examples for the labelling operator, it is usually also necessary to regard communication between server and client as internal and to hide this communication from potential environments. The label s can in the following process be regarded as a local identifier, because messages $s.a$ can never be observed from outside:

$$\Big(s : S \,[\![\, s.\Sigma \,]\!]\, C\Big) \setminus s.\Sigma$$

Table 2.1. Plotkin-style Structural Operational Semantics of CSP

$$\frac{}{(a \to P) \xrightarrow{a} P}$$

$$\frac{P \xrightarrow{\tau(a)} P'}{P \,\Box\, Q \xrightarrow{\tau(a)} P' \,\Box\, Q} \qquad \frac{Q \xrightarrow{\tau(a)} Q'}{P \,\Box\, Q \xrightarrow{\tau(a)} P \,\Box\, Q'}$$

$$\frac{P \xrightarrow{a} P' \qquad a \in \Sigma}{P \,\Box\, Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q' \qquad a \in \Sigma}{P \,\Box\, Q \xrightarrow{a} Q'}$$

$$\frac{v \in \Sigma_0}{(c\,?\,x \to P) \xrightarrow{c.v} P(v)}$$

$$\frac{P \xrightarrow{a} P' \qquad a \notin X}{P \,[\![X]\!]\, Q \xrightarrow{a} P' \,[\![X]\!]\, Q} \qquad \frac{Q \xrightarrow{a} Q' \qquad a \notin X}{P \,[\![X]\!]\, Q \xrightarrow{a} P \,[\![X]\!]\, Q'}$$

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{a} Q' \qquad a \in X}{P \,[\![X]\!]\, Q \xrightarrow{a} P' \,[\![X]\!]\, Q'}$$

$$\frac{P \xrightarrow{a} P' \qquad a \in \Sigma}{l : P \xrightarrow{l.a} l : P'} \qquad \frac{P \xrightarrow{\tau(a)} P'}{l : P \xrightarrow{\tau(a)} l : P'}$$

$$\frac{P \xrightarrow{a} P' \qquad a \notin X}{P \setminus X \xrightarrow{a} P' \setminus X} \qquad \frac{P \xrightarrow{a} P' \qquad a \in X}{P \setminus X \xrightarrow{\tau(a)} P' \setminus X}$$

The syntax of the rules conforms to [AFV01].

If communication was not hidden in this situation, then an environment would be able to interfere and block messages (accidently):

$$\Big(s : S \,[\![s.\Sigma]\!]\, C\Big) \,[\![s.\Sigma]\!]\, \mathit{stop}$$

2.2. Formal Definition of the CSP Operators

The rules of structural operational semantics given in the last pages are collected in Table 2.1, and are in this section complemented with the construction of labelled transition systems (LTSs). The definition of the systems is in principle independent from the CSP semantics, and does in particular not refer to CSP process terms. We rather construct an LTS that (to a certain degree) covers all processes definable—using arbitrary formal notations for processes—and define the CSP operators as mappings on this LTS that are induced by the operational rules. We prefer this approach, as for concrete applications of the process algebra we intend to enrich CSP syntax with further elements, e.g. to reason about processes that are described using the Java language

(possibly using the JCSP library), a function symbol can be defined that embeds a fragment of Java code in a process term (Sect. 2.3). Hence, it is desirable to abstract from the concrete algebra CSP and treat processes in a more general setting.

2.2.1. Labelled Transition Systems

A labelled transition system over a set $\mathcal{L}$ of labels is usually (for instance in [BS01]) defined to be a tuple $(\mathcal{S}, \rightarrow)$ consisting of a set $\mathcal{S}$ of states and a transition relation $\rightarrow \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$, i.e. it describes a directed graph whose edges are labelled with elements of $\mathcal{L}$. The diagram of Fig. 1.1 represents a labelled transition system (LTSs do not have distinguished initial states, however). An LTS can be regarded as the "semantics" of a process algebra by identifying the nodes $\mathcal{S}$ of the system with processes (elements of the algebra), and the edges $P \xrightarrow{a} Q$ with the willingness of a process P to engage in an event a, causing the original process P to pass in the process Q. In computer science, the theory of transition systems has been investigated extensively, in particular considering special cases like (finite) automata, or from the perspective of (multi-)modal or temporal logics (Kripke frames that are used to describe semantics of multi-modal logics can be identified as labelled transition systems, see for instance [Sti92]).

For a systematic treatment of LTSs, we mostly follow the documents [Rut95, Rut96] and regard transition systems as coalgebras. Notions like (strong) bisimularity between processes, which is the semantics underlying all considerations in this report, emerge in this framework in a natural way, and likewise the coinduction principle poses an elegant way to give the operators of CSP semantics by defining them as mappings on a set of processes (this concept is consistent with the operational semantics given in Table 2.1). While a systematic examination of the relation between operational semantics (in particular in the GSOS congruence format) and coalgebras has (and is) recently been performed in [TP97, Pow03], to cope with the simple operational semantics of our CSP fragment we stick to the more basic tools provided in [Rut96].

In the next paragraphs, we adapt a number of basic definitions and theorems of universal coalgebra to the special situation of labelled transition systems as used in this report; the general versions can be found in [Rut96]. We also use a definition of LTSs different from the one given above that captures the "flavour" of coalgebra more directly. Both notions of transition systems are—up to a restriction that is omitted in the first case—equivalent, however. A systematic comparison of the two representations is given in [CGRH00].

2.2.1 Definition (Labelled Transition System): Suppose that $\mathcal{L}$ is a set of labels. A *labelled transition system (LTS)* over $\mathcal{L}$ is a pair $T = (\mathcal{S}, \alpha_{\mathcal{S}})$, which consists of a set $\mathcal{S}$ of states (the *carrier* $\mathcal{S} = \mathit{Carrier}(T)$) together with a function $\alpha_{\mathcal{S}} : \mathcal{S} \rightarrow \mathcal{L} \rightarrow \mathcal{P}_{\mathrm{fi}}\mathcal{S}$ (the *dynamics* $\alpha_{\mathcal{S}} = \mathit{Dynamics}(T)$) that assigns each combination of a state and a label a finite set of states. *

Intuitively, the function $\alpha_{\mathcal{S}}$ collects the outgoing arcs of each state of the LTS, indexed by the labels the edges are marked with. The correspondence between an LTS $(\mathcal{S}, \alpha_{\mathcal{S}})$ as in Def. 2.2.1 and the notion of LTSs as pairs $(\mathcal{S}, \rightarrow)$ of a set of states and a transition

relation can be seen as follows: In either direction the carrier $\mathcal{S}$ can be retained. Given a mapping $\alpha_{\mathcal{S}} : \mathcal{S} \to \mathcal{L} \to \mathcal{P}_{\text{fi}}\mathcal{S}$, a transition relation can be constructed by

$$\to \quad := \quad \{(P, a, Q) \mid Q \in \alpha_{\mathcal{S}}(P)(a)\},$$

and, given a transition relation $\to$, the values of a matching function $\alpha_{\mathcal{S}}$ are determined by

$$\alpha_{\mathcal{S}}(P)(a) \quad := \quad \{Q \mid P \xrightarrow{a} Q\}, \qquad P \in \mathcal{S}, \quad a \in \mathcal{L}$$

provided that these sets are finite. This finiteness is known as *image finite* condition on an LTS (e.g. see [AFV01]): From a particular state p and exclusively using edges labelled with a, only a finite number of target states can be reached in one step. We require the image finite condition at this point to obtain a class of LTSs with more convenient properties (details are given below).

As the two definitions of labelled transition systems are interchangeable, we also use the notation $P \xrightarrow{a} Q$ as an abbreviation of $Q \in \alpha_{\mathcal{S}}(P)(a)$ in the sequel. At times, we also add an index as in $P \xrightarrow{a}_{\mathcal{S}} Q$ to the transition arrow, if the LTS the statement refers to cannot unambiguously be determined from the context.

As a basis for equipping labelled transition systems with algebraic connectives (like the CSP operators), a concept of morphisms between labelled transition systems is needed:

2.2.2 Definition (LTS-Homomorphism): Suppose that $(\mathcal{S}, \alpha_{\mathcal{S}})$, $(\mathcal{T}, \alpha_{\mathcal{T}})$ are two labelled transition systems over the same set $\mathcal{L}$ of labels. A function $f : \mathcal{S} \to \mathcal{T}$ is called a *homomorphism* between $(\mathcal{S}, \alpha_{\mathcal{S}})$ and $(\mathcal{T}, \alpha_{\mathcal{T}})$ if f commutes with the dynamics of the systems:[4]

$$\forall\, a \in \mathcal{L}, P \in \mathcal{S}. \quad f(\alpha_{\mathcal{S}}(P)(a)) = \alpha_{\mathcal{T}}(f(P))(a). \qquad *$$

The same condition can be formulated in two parts referring to the traditional definition of an LTS in the following way ($a \in \mathcal{L}$, $P \in \mathcal{S}$):

- '$\subseteq$': For $Q \in \mathcal{S}$: If $P \xrightarrow{a}_{\mathcal{S}} Q$, then $f(P) \xrightarrow{a}_{\mathcal{T}} f(Q)$
- '$\supseteq$': For $R \in \mathcal{T}$: If $f(P) \xrightarrow{a}_{\mathcal{T}} R$, then there is a $Q \in \mathcal{S}$ with $P \xrightarrow{a}_{\mathcal{S}} Q$ and $f(Q) = R$.

Homomorphisms of labelled transition systems preserving transition structures (as well as the more general notion of coalgebra-homomorphisms) can be observed to have essentially the same properties as homomorphisms of algebras preserving applications of the algebra operators. In the same way as algebra morphisms can be used to obtain a concept of congruence, LTS morphisms are intimately related to the idea of *bisimulation*.[5]

Bisimulation is a concept that relates certain processes of LTSs whose behaviour is regarded as equivalent; an extensive discussing of bisimulation and further equivalence relations on labelled transition systems can be found in [Gla01]. As we are able to

[4] The function f is continued to the power set $\mathcal{PS}$ in the usual way.

[5] In [Rut96], bisimulation equivalences are defined as a special case of bisimulations, which are binary relations between two not necessarily identical LTSs. On the other hand, in [AFV01] only the greatest bisimulations on an LTS are called bisimulation equivalences.

construct transition systems that do not contain distinct processes with the same behaviour up to bisimulation (Lem. 2.2.5 below), processes can in this system be uniquely defined through their behaviour.

2.2.3 Definition (Bisimulation Equivalence): Suppose that $(\mathcal{S}, \alpha_\mathcal{S})$ is an LTS over the set $\mathcal{L}$ of labels. A *bisimulation equivalence* Rel on $(\mathcal{S}, \alpha_\mathcal{S})$ is an equivalence relation Rel on $\mathcal{S}$ that only identifies states with the same adjacent processes (modulo Rel):[6]

$$\forall (P, Q) \in Rel, a \in \mathcal{L}. \quad \alpha_\mathcal{S}(P)(a) \cdot Rel = \alpha_\mathcal{S}(Q)(a) \cdot Rel. \qquad *$$

Again it is possible to give an equivalent formulation of this condition using the traditional notion of LTSs:

- For $(P, Q) \in Rel$, $a \in \mathcal{L}$: If $P \xrightarrow{a} P'$, then there is a $Q' \in \mathcal{S}$ with $Q \xrightarrow{a} Q'$ and $P'\ Rel\ Q'$
- Rel is an equivalence relation on $\mathcal{S}$.

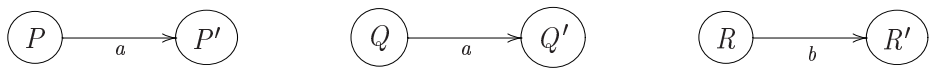
A further way to introduce bisimulation equivalences—which is not referred to in this report, however—is their connection to homomorphisms: The kernel of a homomorphism f is defined to be the equivalence relation that relates those elements which are mapped to the same image elements:

$$\ker f \quad := \quad \{(x, y) \in \mathcal{S} \times \mathcal{S} \mid f(x) = f(y)\}$$

It can be observed that bisimulation equivalences and kernels of homomorphisms coincide, i.e. that kernels are always bisimulations equivalences, and that on the other hand for each bisimulation equivalence Rel we can find a certain LTS $(\mathcal{T}, \alpha_\mathcal{T})$ and a morphism $f : (\mathcal{S}, \alpha_\mathcal{S}) \to (\mathcal{T}, \alpha_\mathcal{T})$, such that the kernel of f is Rel.

2.2.4 Example: The criterion of Def. 2.2.3 states that a bisimulation equivalence has to be closed regarding to the progression of processes, i.e. if two processes are identified, then also the processes that are reachable have to be regarded as equivalent. Accordingly, for the LTS $(\mathcal{S}, \alpha_\mathcal{S})$ given by the diagram below, the equivalence relations Rel_1 and Rel_2, which are generated by the sets $\{(P', Q')\}$ and $\{(P', Q'), (P', R')\}$ (resp.), are bisimulation equivalences.

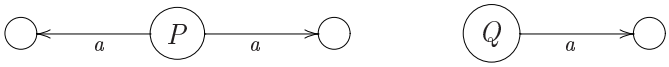
In contrast, relation Rel_3 is not a bisimulation, because the processes P and Q are identified, but the adjacent processes P', Q' are not. Rel_3 can be enlarged by adding the generator (P', Q') to create a bisimulation Rel_4. The relation Rel_5 is not a bisimulation, and is not a subset of any bisimulation equivalence either, because for all equivalence relations A we have $\alpha_\mathcal{S}(P)(a) \cdot A \neq \varnothing = \alpha_\mathcal{S}(R)(a) \cdot A$.



$$Rel_1 := <\{(P', Q')\}> \qquad Rel_2 := <\{(P', Q'), (P', R')\}>$$
$$Rel_3 := <\{(P, Q)\}> \qquad Rel_4 := <\{(P, Q), (P', Q')\}>$$
$$Rel_5 := <\{(P, R)\}> \qquad *$$

[6] With $A \cdot Rel$ we denote the usual product of a set A and a binary relation Rel, which is defined as follows: $A \cdot Rel := \{y \ : \ \exists x \in A.\ (x, y) \in Rel\}$

An interesting observation is the existence of greatest bisimulation equivalences for arbitrary LTSs (w.r.t. the ordering given by set inclusion). Pairs of processes that are element of the greatest bisimulation are also called *bisimular*. In the whole thesis, processes that are bisimular are regarded as (behavioural) equivalent. Following this concept, for instance the states P, Q of the following system are representations of the same process:



In order to justify the existence of greatest bisimulation equivalences, we can observe that the set of bisimulation equivalences on an LTS $(\mathcal{S}, \alpha_{\mathcal{S}})$ even forms a complete lattice: The bounds of a collection $(Rel_k)_{k \in I}$ of bisimulation equivalences can be derived from the corresponding operations on sets in the following way:

$$\bigvee_{k \in I} Rel_k \quad := \quad \bigcap \{Rel \mid Rel \text{ bis. eq., } Rel \supseteq \bigcup_{k \in I} Rel_k\}$$
$$\bigwedge_{k \in I} Rel_k \quad := \quad \bigcup \{Rel \mid Rel \text{ bis. eq., } Rel \subseteq \bigcap_{k \in I} Rel_k\}$$

To construct "canonical" transition systems that are usable for defining the CSP operators as connectives, we collect three further statements in the next lemma. It is not surprising that LTS morphisms complement the class of labelled transition systems to be a category, and it can be shown that in correspondence to initial algebras this category contains final labelled transition systems (coalgebras), which in a certain way represent all other LTSs.

2.2.5 Lemma (LTS Categories): Suppose that $\mathcal{L}$ is a set of labels.

1. Together with the LTS-homomorphisms, the class of labelled transition systems over $\mathcal{L}$ form a category $\underline{\mathrm{LTS}}_{\mathcal{L}}$.

2. The category $\underline{\mathrm{LTS}}_{\mathcal{L}}$ contains a final element, i.e. there is a labelled transition system $\mathrm{FinLTS}_{\mathcal{L}}$ over $\mathcal{L}$ (that is unique up to isomorphism) such that for each LTS $(\mathcal{S}, \alpha_{\mathcal{S}})$ over $\mathcal{L}$ there is exactly one homomorphism $f : (\mathcal{S}, \alpha_{\mathcal{S}}) \to \mathrm{FinLTS}_{\mathcal{L}}$.

3. The LTS $\mathrm{FinLTS}_{\mathcal{L}}$ is *simple*, i.e. there is exactly one bisimulation equivalence on $\mathrm{FinLTS}_{\mathcal{L}}$, which is the identity $=$. *

As each LTS over a certain label set $\mathcal{L}$ can be mapped to the final LTS $\mathrm{FinLTS}_{\mathcal{L}}$, homomorphic images of all possible LTSs can be found by looking at the sub-LTSs of $\mathrm{FinLTS}_{\mathcal{L}}$. Besides that, for each LTS $(\mathcal{S}, \alpha_{\mathcal{S}})$ the kernel of the uniquely determined morphism $f : (\mathcal{S}, \alpha_{\mathcal{S}}) \to \mathrm{FinLTS}_{\mathcal{L}}$ is the greatest bisimulation equivalence on $(\mathcal{S}, \alpha_{\mathcal{S}})$. This means that $\mathrm{FinLTS}_{\mathcal{L}}$ represents a space of processes that is "complete" in the sense of: Each process that can be described (or constructed) as a state of an arbitrary LTS also is an element of $\mathrm{FinLTS}_{\mathcal{L}}$ (up to bisimulation). This observation is called *coinductive definition principle* and can be used to show the existence of processes with certain properties within $\mathrm{FinLTS}_{\mathcal{L}}$, and also to equip $\mathrm{FinLTS}_{\mathcal{L}}$ with the CSP algebra structure (which is the topic of the next section).

The simplicity of FinLTS$_{\mathcal{L}}$, on the other hand, entails that FinLTS$_{\mathcal{L}}$ contains each process at most once, i.e. no two different elements of FinLTS$_{\mathcal{L}}$ are bisimular. This reflects the convention of regarding bisimular processes as equivalent from above. Thus, FinLTS$_{\mathcal{L}}$ is a complete and redundance-free collection of all existing processes.

To prove that two processes of FinLTS$_{\mathcal{L}}$ are equal, it is sufficient to show the bisimularity of the processes. This method can be used in combination with coinductive definitions to prove algebraic identities, and is known as *coinductive proof principle*.

A number of coinductive proofs can be found in Appendix A. Typical statements for which the principle can be applied are equations

$$P_1 = P_2 \qquad (*)$$

that compare expressions P_1, P_2 denoting processes of FinLTS$_{\mathcal{L}}$. Usually, the proof of an equation like (*) is constructed using the following pattern:

1. An equivalence relation *Rel* on the set *Carrier*(FinLTS$_{\mathcal{L}}$) is constructed that subsumes (*), i.e. in a way such that $(x_1, x_2) \in Rel$ holds for all values (x_1, x_2) the arguments (P_1, P_2) of the equation can be evaluated to. *Rel* has to be formulated in a sufficiently strong way, which is a task comparable to finding a suitable hypothesis when proving by induction.

2. By proving the condition of Def. 2.2.3, it is shown that *Rel* is a bisimulation:

   $$\forall (P, Q) \in Rel, a \in \mathcal{L}. \quad \alpha_{\mathcal{S}}(P)(a) \cdot Rel = \alpha_{\mathcal{S}}(Q)(a) \cdot Rel.^{7} \qquad (**)$$

   If *Rel* is known to be the reflexive, symmetric and transitive closure of a set G of generators (for instance by definition), i.e. $Rel = < G >$, then because of the properties of equality it is possible to simplify condition (**) and only show the statement for the generators G:

   $$\forall (P, Q) \in G, a \in \mathcal{L}. \quad \alpha_{\mathcal{S}}(P)(a) \cdot Rel = \alpha_{\mathcal{S}}(Q)(a) \cdot Rel.$$

3. The third step is accomplished by Lem. 2.2.5: The bisimulation equivalence *Rel* is in fact the equality $=$ on the set *Carrier*(FinLTS$_{\mathcal{L}}$). In particular this proves the equation $P_1 = P_2$, because by construction pairs (x_1, x_2) compared by the equation are members of *Rel*.

2.2.2. Definition of the CSP Operators

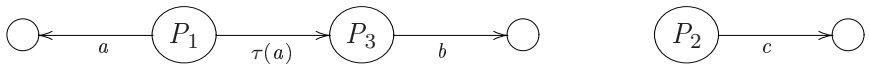
After the short introduction to labelled transition systems in the previous section, the description of the fragment of CSP in Sect. 2.1 can now be complemented with a formal definition. As most basic prerequisite therefore, the process space $Proc_{\Sigma_0}$ that is referred to in Sect. 2.1 is defined to be a final labelled transition system, on which it is then possible to construct the operators of CSP by using coinduction. The labels of the LTS cover both visible and invisible events of CSP:

$$\mathcal{L}_{CSP} \quad := \quad \Sigma \cup \Sigma_\tau, \qquad\qquad Proc_{\Sigma_0} \quad := \quad Carrier(\text{FinLTS}_{\mathcal{L}_{CSP}}).$$

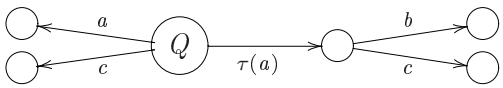
[7] In proofs we also use the notation $\alpha_{\mathcal{S}}(p)(a) \sim_{Rel} \alpha_{\mathcal{S}}(q)(a)$ when continuing *Rel* to sets.

In order to define the semantics of the CSP operators, we use the following pattern, which is also described in [RT94, Rut96] in detail. The rather simple pattern can be applied because of the format of the operational rules in Table 2.1, which is essentially GSOS (for a definition see [AFV01]). In this format, each rule makes a proposition about the result of an application of operator *op* to arbitrary arguments represented by variables (the left part of conclusions of the rules), which can directly be translated into the dynamics of an LTS. In our case, the situation is particularly simple, since also the right terms of conclusions do not contain CSP operators apart from *op* (despite the fact that GSOS would also allow more general rules). To motivate the construction, we start with an example.

2.2.6 Example (Dynamics of External Choice): We construct the process $P_1 \,\Box\, P_2 \in Proc_{\Sigma_0}$ that results from the application of external choice to the processes $P_1, P_2 \in Proc_{\Sigma_0}$, which are given by diagrams of their LTSs:



According to operational semantics, $P_1 \,\Box\, P_2$ is either able to engage in event a or in event c, which causes commitment to one of the subprocesses, or to perform the internal step $\tau(a)$ and pass in a process $P_3 \,\Box\, P_2 \in Proc_{\Sigma_0}$. Likewise, $P_3 \,\Box\, P_2$ is able to raise one of the events b, c. An LTS of $P_1 \,\Box\, P_2 =: Q$ is hence



A homomorphic image of the constructed process Q can be located in $\mathrm{FinLTS}_{\mathcal{L}_{CSP}}$ by using coinduction: The system shown in the diagram is an object $(\mathcal{S}, \alpha_{\mathcal{S}})$ of the category $\underline{\mathrm{LTS}}_{\mathcal{L}}$ (we can simply choose $\mathcal{S}$ to be the set of nodes of the diagram, and the dynamics $\alpha_{\mathcal{S}}$ can be defined as a representation of the arcs of the diagram). By finality, there is a unique morphism $f : (\mathcal{S}, \alpha_{\mathcal{S}}) \to \mathrm{FinLTS}_{\mathcal{L}_{CSP}}$, and because f preserves transition structures the behaviour of the process $f(Q) \in Proc_{\Sigma_0}$ conforms to the LTS that is shown. Thus, we can define the value of $P_1 \,\Box\, P_2$ to be $f(Q)$. ∗

The construction of the example is in the next pages performed for the CSP operators in general. A discussion of similar constructions for rules of the full GSOS format can be found in [TP97, Pow03], whereas for our convenient operational semantics the recipe is as follows:

1. The rules of the operational semantics in Table 2.1 are examined, and (as in the example) an LTS $(\mathcal{S}, \alpha_{\mathcal{S}})$ is defined that describes the meaning of the operators in a way equivalent to the operational rules (together with the assumption that processes can perform at most those execution steps whose existence is provable by using the rules). To construct the carrier $\mathcal{S}$ of the LTS, the process terms that occur in conclusions of the operational rules are collected, and for each term all possible instances that can be obtained by replacing contained variables with elements of the respective sets (according to the signatures of the operators) are

included in $\mathcal{S}$.[8] To avoid misconceptions and clarify the usage of terms as states of the LTS at this point, we add an index t as in op_t to CSP operators used in such terms. Considering for instance the rule that describes the prefix operator, the following terms would be added to the carrier (the conclusion of the rule contains the two process terms $a \to P$ and P):

$$\{a \to_t P \mid a \in \Sigma, P \in Proc_{\Sigma_0}\} \cup Proc_{\Sigma_0}$$

The LTS is equipped with a dynamics $\alpha_{\mathcal{S}}$ according to the operational semantics: For a rule with conclusion $t_1 \xrightarrow{a} t_2$, the LTS contains a transition labelled with a between all instances of the terms t_1, t_2 of $\mathcal{S}$, provided that the constituents of the instances satisfy the premisses of the operational rule. For the running example that treats the prefixing operator, the resulting transitions are

$$\big(a \to_t P\big) \xrightarrow{a} P \qquad a \in \Sigma, P \in Proc_{\Sigma_0}$$

2. The behaviour of a process $op_t(v_1, \ldots, v_n) \in \mathcal{S}$ is by construction precisely the behaviour that the SOS rules demand for $op(v_1, \ldots, v_n) \in Proc_{\Sigma_0}$. We are able to locate $op(v_1, \ldots, v_n)$ as an element of $\mathrm{FinLTS}_{\mathcal{L}_{CSP}}$ by using coinduction: Provided that the LTS $(\mathcal{S}, \alpha_{\mathcal{S}})$ is well-defined, by the finality of $\mathrm{FinLTS}_{\mathcal{L}_{CSP}}$ there is a unique homomorphism $f : (\mathcal{S}, \alpha_{\mathcal{S}}) \to \mathrm{FinLTS}_{\mathcal{L}_{CSP}}$. As f preserves transition structures, $f(op_t(v_1, \ldots, v_n)) \in Proc_{\Sigma_0}$ has the same behaviour as $op_t(v_1, \ldots, v_n)$ up to bisimularity, and because f is unique a reasonable definition of op is:

$$op(v_1, \ldots, v_n) \quad := \quad f(op_t(v_1, \ldots, v_n)) \tag{2.1}$$

Performing the two steps is mostly mechanical:

Definition of the Carrier: The carrier of the LTS consists of the instances of conclusion terms; we also have to add the unmodified set $Proc_{\Sigma_0}$ of processes, because the conclusions of some of the rules (for instance of prefixing) contain terms in which no CSP operators turn up.[9]

$$\begin{aligned}
\mathcal{S} \quad := \quad & Proc_{\Sigma_0} \\
& \cup \{\ stop_t\ \} \\
& \cup \{\ a \to_t P \quad \mid a \in \Sigma,\ P \in Proc_{\Sigma_0}\ \} \\
& \cup \{\ P \,\Box_t\, Q \quad \mid P, Q \in Proc_{\Sigma_0}\ \} \\
& \cup \{\ c\,?\,x \to_t P \quad \mid c \in \Sigma \cup \{\epsilon\},\ P \in (Proc_{\Sigma_0})^{\Sigma_0}\ \} \\
& \cup \{\ P \,[\![\, X \,]\!]_t\, Q \quad \mid X \subseteq \Sigma,\ P, Q \in Proc_{\Sigma_0}\ \} \\
& \cup \{\ l :_t P \quad \mid l \in \Sigma,\ P \in Proc_{\Sigma_0}\ \} \\
& \cup \{\ P \setminus_t X \quad \mid X \subseteq \Sigma,\ P \in Proc_{\Sigma_0}\ \}
\end{aligned}$$

[8] The usage of "mixed terms"—which are built from the principal syntactic operators as well as from semantic entities—as elements of the carrier is described in [RT94]. Equivalently, we could require the existence of a constant symbol for each particular semantic element.

[9] We assume that the parts that make up $\mathcal{S}$ are disjoint, which means that $Proc_{\Sigma_0}$ must not contain any of the mixed terms of the other subsets. This is no restriction, because $\mathrm{FinLTS}_{\mathcal{L}_{CSP}}$ is only unique up to isomorphism, and hence it is always possible to choose the set $Proc_{\Sigma_0}$ appropriately.

Table 2.2. Dynamics of the LTS $(\mathcal{S}, \alpha_{\mathcal{S}})$ Defining the CSP Operators

$$
\begin{array}{rcl}
\alpha_{\mathcal{S}}(P) & := & \alpha_{CSP}(P) \\
\alpha_{\mathcal{S}}(stop_t)(e) & := & \varnothing \\
\alpha_{\mathcal{S}}(a \rightarrow_t P)(e) & := & \begin{cases} \{P\} & e = a \\ \varnothing & e \neq a \end{cases} \\
\alpha_{\mathcal{S}}(P \,\Box_t\, Q)(e) & := & \begin{cases} \{P' \,\Box_t\, Q \mid P \xrightarrow{e} P'\} & \\ \qquad \cup\, \{P \,\Box_t\, Q' \mid Q \xrightarrow{e} Q'\} & e \in \Sigma_\tau \\ \alpha_{CSP}(P)(e) \;\cup\; \alpha_{CSP}(Q)(e) & e \in \Sigma \end{cases} \\
\alpha_{\mathcal{S}}(c\,?\,x \rightarrow_t P)(e) & := & \begin{cases} \{P(v)\} & \exists\, v \in \Sigma_0.\ e = c.v \\ \varnothing & \text{otherwise} \end{cases} \\
\alpha_{\mathcal{S}}(P \,[\![\, X \,]\!]_t\, Q)(e) & := & \begin{cases} \{P' \,[\![\, X \,]\!]_t\, Q \mid P \xrightarrow{e} P'\} & \\ \qquad \cup\, \{P \,[\![\, X \,]\!]_t\, Q' \mid Q \xrightarrow{e} Q'\} & e \notin X \\ \{P' \,[\![\, X \,]\!]_t\, Q' \mid P \xrightarrow{e} P',\ Q \xrightarrow{e} Q'\} & e \in X \end{cases} \\
\alpha_{\mathcal{S}}(l :_t P)(e) & := & \begin{cases} \{l :_t P' \mid P \xrightarrow{a} P'\} & \exists\, a \in \Sigma.\ e = l.a \\ \{l :_t P' \mid P \xrightarrow{e} P'\} & e \in \Sigma_\tau \\ \varnothing & \text{otherwise} \end{cases} \\
\alpha_{\mathcal{S}}(P \setminus_t X)(e) & := & \bigcup\limits_{\substack{a \in X \\ e = \tau(a)}} \{P' \setminus_t X \mid P \xrightarrow{a} P'\} \;\cup \\
 & & \begin{cases} \{P' \setminus_t X \mid P \xrightarrow{e} P'\} & e \notin X \\ \varnothing & \text{otherwise} \end{cases}
\end{array}
$$

Definition of the Dynamics: The dynamics of the LTS is defined by a case distinction to cover the different parts of the carrier (Table 2.2); each case corresponds to a group of operational rules that treats a particular CSP operator. We use the notation $\alpha_{CSP} := \mathit{Dynamics}(\mathrm{FinLTS}_{\mathcal{L}_{CSP}})$. The part $\mathit{Proc}_{\Sigma_0} \subseteq \mathcal{S}$ of the carrier is equipped with unmodified dynamics from $\mathrm{FinLTS}_{\mathcal{L}_{CSP}}$, which ensures that $(\mathit{Proc}_{\Sigma_0}, \alpha_{\mathcal{S}}|_{\mathit{Proc}_{\Sigma_0}})$ forms a sub-LTS of $(\mathcal{S}, \alpha_{\mathcal{S}})$ that is isomorphic (even identical) to $\mathrm{FinLTS}_{\mathcal{L}_{CSP}}$.

To ascertain that the LTS is well-defined, it is primarily necessary to check the image finite condition for each of the equations in Table 2.2, i.e. for each state $p \in \mathcal{S}$ and each label $e \in \mathcal{L}_{CSP}$ the set $\alpha_{\mathcal{S}}(p)(e)$ has to be finite. As an example we verify the condition for the equation defining external choice:

$$\alpha_{\mathcal{S}}(P \,\Box_t\, Q)(e) \quad := \quad \begin{cases} \{P' \,\Box_t\, Q \mid P \xrightarrow{e} P'\} & \\ \qquad \cup\, \{P \,\Box_t\, Q' \mid Q \xrightarrow{e} Q'\} & e \in \Sigma_\tau \\ \alpha_{CSP}(P)(e) \;\cup\; \alpha_{CSP}(Q)(e) & e \in \Sigma \end{cases}$$

Because $\mathrm{FinLTS}_{\mathcal{L}_{CSP}}$ is a well-defined LTS, the sets $\alpha_{CSP}(P)(e)$ and $\alpha_{CSP}(Q)(e)$ are finite, and consequently this also holds for all other sets that occur on the right side of the equation (apart from Σ_τ, Σ).

Definition of CSP Operators: The second part of the construction does actually not require any further manual work, as Lem. 2.2.5 guarantees the existence of the LTS-homomorphism $f : (\mathcal{S}, \alpha_{\mathcal{S}}) \to \mathrm{FinLTS}_{\mathcal{L}_{CSP}}$, and as the CSP operators can now simply be evaluated using equation (2.1). For the operator of parallelism, for instance, this leads to the definition

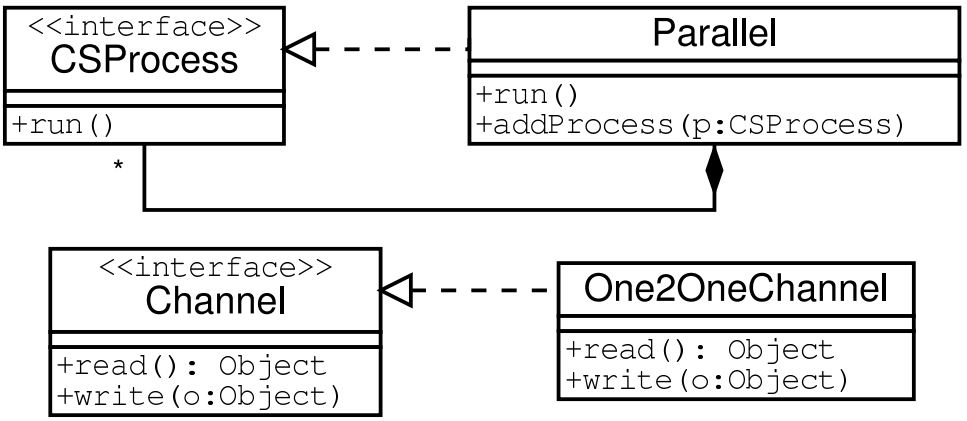
$$P \,[\![X]\!]\, Q \quad := \quad f(P \,[\![X]\!]_t\, Q).$$

The possibility remains to rewrite the equations from Table 2.2 by applying the definition of a homomorphism to gain a similar set of "characteristic equations" that no longer refer to the LTS $(\mathcal{S}, \alpha_{\mathcal{S}})$. This looks as follows for parallelism, and can analogously be performed for all other CSP operators (we omit a complete table of the characteristic equations, because the only difference to the equations of Table 2.2 is the absence of indices t):

$$\begin{aligned} \alpha_{CSP}(P \,[\![X]\!]\, Q)(e) \quad &= \quad \alpha_{CSP}(f(P \,[\![X]\!]_t\, Q))(e) \quad = \quad f(\alpha_{\mathcal{S}}(P \,[\![X]\!]_t\, Q)(e)) \\ &= \quad \begin{cases} f(\{P' \,[\![X]\!]_t\, Q \mid P \xrightarrow{e} P'\} & \\ \qquad \cup\, \{P \,[\![X]\!]_t\, Q' \mid Q \xrightarrow{e} Q'\}) & e \notin X \\ f(\{P' \,[\![X]\!]_t\, Q' \mid P \xrightarrow{e} P',\ Q \xrightarrow{e} Q'\}) & e \in X \end{cases} \\ &= \quad \begin{cases} \{P' \,[\![X]\!]\, Q \mid P \xrightarrow{e} P'\} & \\ \qquad \cup\, \{P \,[\![X]\!]\, Q' \mid Q \xrightarrow{e} Q'\} & e \notin X \\ \{P' \,[\![X]\!]\, Q' \mid P \xrightarrow{e} P',\ Q \xrightarrow{e} Q'\} & e \in X \end{cases} \end{aligned}$$

The characteristic equations uniquely define the CSP operators as connectives on the set Proc_{Σ_0} (and other sets that turn up in the signatures of the operations, like Σ). Indeed, a proof of the existence of solutions has been performed in the last paragraphs

Figure 2.1. Simplified UML Class Diagrams [OMG01] of the Interfaces and Classes of JCSP that Represent Processes and Channels



by the construction of the LTS $(\mathcal{S}, \alpha_{\mathcal{S}})$, and uniqueness can be shown using the coinductive proof principle. It can also be observed that the equations are only a different notation of the SOS rules of Table 2.1; for the operator of parallel composition, the case distinction of the equation corresponds to the two groups of rules for interleaving and synchronisation (resp.), and the sets of the right side of the equation represent all situations in which the rules can be applied.

Strictly speaking, and when regarded as mappings on $Proc_{\Sigma_0}$, the CSP operators are parametrised with the connective '.' on the set of visible events Σ (that makes the set a semi-group), and with the mapping $\tau : \Sigma \to \Sigma_\tau$ (as described in Sect. 2.1) relating internal and external steps. For instance the definition of $\alpha_{\mathcal{S}}(l :_t P)$ in Table 2.2 requires evaluations of '.', and thus a modification of '.' leads to different LTSs. For a shorter notation, we omit corresponding indices when writing the operators and assume that both functions are predetermined by the choice of Σ_0.

2.3. Modelling JCSP using CSP

The Java library JCSP [WA] implements a process model that to a certain degree resembles the CSP algebra. Processes can in this framework be implemented using the Java language, and can be run in parallel and communicate with each other by using respective primitives of JCSP. Simplified versions of some of the most basic classes and interfaces of JCSP are given in Fig. 2.1, namely of

- An interface CSProcess, objects of whose implementations represent processes in the JCSP framework. The implementing class Parallel is used to run processes in parallel, and has approximately the same meaning as the CSP interleaving operator $|||$.
- The Channel interface, which represents the primary way of realising communication between processes. JCSP contains different implementations of this inter-

face, one of which is the One2OneChannel class for sending messages between two particular processes. The standard semantics of this channel is a zero-buffered and synchronous communication; the exact behaviour can be chosen by adjusting parameters that are left out in the diagram, though. Further classes exist to establish communication between groups of processes.

A simple example of a JCSP system is given in Fig. 2.2. The network (represented by class Network) consists of a serving process that computes successors of natural numbers, and of a client that demands the computation of the successor of 41. Both processes are connected by a channel, and are run in parallel by using the JCSP class Parallel.

To represent systems as the one shown as processes, so that interdependencies can be expressed using the CSP algebra, we use a modelling of sequential Java and of the JCSP primitives that is shortly introduced in the next pages. Both for sequential Java programs and for networks of JCSP processes we can define interpretations as elements of a process space $Proc_{\Sigma_0}$. The interpretation function is a homomorphism of the JCSP primitives, which are mapped to corresponding process combinators that are described in terms of CSP. In this manner, the class Parallel can be represented by the CSP interleaving operator, and class One2OneChannel is mapped to a parallel composition with a certain CSP channel process.

For this section, we assume that a context of defined classes and interfaces is fixed.

- In the first place, we define the alphabet of CSP processes used to represent JCSP, which consists of three parts:

  - For representing messages that can be sent through channels, we assume that an appropriate encoding of the relevant information is given. If we for instance consider the statement

    ```
    c.write ( new Integer ( 41 ) );
    ```

    from Fig. 2.2, then it has to be encoded that an instance of class Integer is to be transmitted, and that a certain attribute of this instance has value 41. The identity of the object (its memory location) does not have to be transfered, because we regard the address spaces of different processes to be disjoint; memory locations of the sending process do not have any meaning for the receiver.[10]
  - To distinguish different channels, we furthermore require *Chan* to be a set of channel identifiers, and we assume that instances of implementations of the interface Channel are assigned unique identifiers.
  - For the models of JCSP channels, certain atomic events are necessary as keywords; the set of these events is denoted with *Cmds*.

[10] This assumption is in fact not realised in the implementation of JCSP from [WA]; we regard separated address spaces to be an essential feature of the process model that underlies the presented verification approach, however. Instead of implementing this separation explicitly, which would probably cause performance problems, the non-interference of distinct processes can also be ensured at compile-time by using static analysis.

Figure 2.2. Example Network of Two Processes in JCSP

```
public class Succ implements CSProcess {
    private final Channel c;
    public Succ (Channel c) { this.c = c; }

    public void run () {
        while ( true ) {
            final Integer i = (Integer)c.read();
            c.write ( new Integer ( i.intValue () + 1 ) );
        }
    }
}

public class Client implements CSProcess {
    private final Channel c;
    public Client (Channel c) { this.c = c; }

    public void run () {
        c.write ( new Integer ( 41 ) );
        final Integer n = (Integer)c.read ();
        /* do something */
    }
}

public class Network extends Parallel {
    private final Channel c = new One2OneChannel ();

    public Network () {
        addProcess ( new Succ ( c ) );
        addProcess ( new Client ( c ) );
    }
}
```

As in Sect. 2.1, the alphabet Σ is defined to be the free semi-group over a set of atomic events, which is now chosen as

$$\Sigma_0 \quad := \quad \mathit{Msg} \cup \mathit{Chan} \cup \mathit{Cmds}$$

The set $\Sigma \cup \Sigma_\tau$ of LTS labels is denoted with $\mathcal{L}_{CSP}$ as in Sect. 2.2.2.

- As base case for combining processes written using Java, a suitable operational semantics for sequential Java programs is defined that accounts for the communication primitives of JCSP. This is done by using the notion of labelled transition systems of Sect. 2.2.1: For each fragment α of Java code (possibly containing invocations of methods of the JCSP classes and interfaces) and each environment σ (describing memory contents), a processes $T(\alpha, \sigma) \in \mathit{Carrier}(\mathrm{FinLTS}_{\mathcal{L}_{CSP}})$ is distinguished. The details of the operational behaviour can be defined in different ways; we use approximately the following definition:

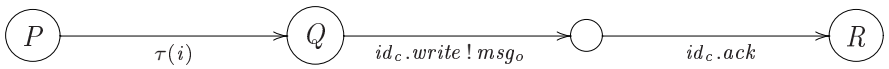
  – Normal execution steps of the Java program (apart from invocations of JCSP methods) have the usual effect to the internal machine state, and can be represented as transitions labelled with invisible actions $\tau(a) \in \Sigma_\tau$. It is also possible to omit such transitions completely from the labelled transition system.
  – Invocations of JCSP methods make the process raise certain combinations of (visible) events according to the used CSP models of JCSP classes.

  As an example, we consider the Java code fragment

  ```
  { i = 1; c.write(o); α }
  ```

  This program first assigns a value to variable i, which is in the LTS below represented as an internal execution step $\tau(i)$, and which alters an initial environment σ into σ'. Subsequently, the method One2OneChannel.write is invoked. α is an arbitrary piece of Java code, $\mathit{id}_c \in \mathit{Chan}$ is the identifier of the channel referred to by c, $\mathit{msg}_o \in \mathit{Msg}$ is the event encoding message o and $\mathit{write}, \mathit{ack} \in \mathit{Cmds}$ are keywords of the used channel model. The sequence $\mathit{write} \rightarrow \mathit{ack} \rightarrow \cdots$ is part of a protocol between the process sending a message and a dedicated channel process (defined below).

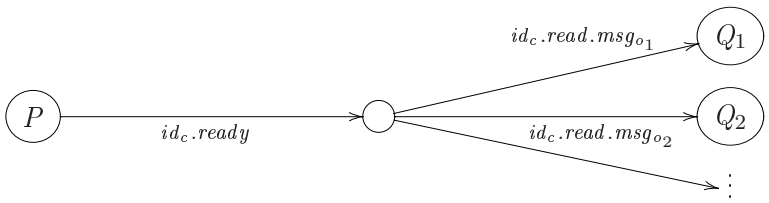
$$\begin{aligned} P \quad &:= \quad T(\ \{\ \mathrm{i} = 1;\ \mathrm{c.write(o)};\ \alpha\ \}\ , \sigma) \\ Q \quad &:= \quad T(\ \{\ \mathrm{c.write(o)};\ \alpha\ \}\ , \sigma') \\ R \quad &:= \quad T(\ \{\ \alpha\ \}\ , \sigma') \end{aligned}$$



For the channel model that we refer to in this thesis, which is described below, invocations of the method One2OneChannel.read have an effect similar to the CSP prefix choice operator, i.e. messages are read from an environment. We assume that $\mathit{Msg} = \{\mathit{msg}_{o_1}, \mathit{msg}_{o_2}, \ldots\}$ are all JCSP messages that can be transmitted,

and that $\sigma_1, \sigma_2, \ldots$ are environments that reflect the receiving of the respective messages. σ_i is obtained from an original environment σ by adding the received objects to the heap, and by assigning the reference variable r the location of the primary object transmitted. The sequence $ready \rightarrow read \rightarrow \cdots$ is again a protocol for interaction with a channel process.

$$\begin{aligned} P &:= T(\ \{\ \mathrm{r} = \mathrm{c.read}();\ \alpha\ \}\ , \sigma') \\ Q_1 &:= T(\ \{\ \alpha\ \}\ , \sigma_1) \\ Q_2 &:= T(\ \{\ \alpha\ \}\ , \sigma_2) \end{aligned}$$



- JCSP channel objects are modelled as processes, and are defined by using the CSP operators. A model for the class One2OneChannel of Fig. 2.1 is given in [PJ00], where it is also proved correct by using the model checker FDR [For97]. The version of the model we give here is slightly adapted to the present situation. The processes are determined as solutions of the following recursive equations:

$$\begin{aligned} LEFT &= write\,?\,msg \rightarrow transmit\,!\,msg \rightarrow ack \rightarrow LEFT \\ RIGHT &= ready \rightarrow transmit\,?\,msg \rightarrow read\,!\,msg \rightarrow RIGHT \\ ONE2ONECHANNEL &= \big(LEFT \,[\![\, transmit.\Sigma \,]\!]\, RIGHT\big) \setminus transmit.\Sigma \end{aligned}$$

  Keywords used in the process definitions are elements of the set $Cmds$:

$$write, ack, ready, read, transmit \in Cmds$$

  A channel process $ONE2ONECHANNEL$ controls the communication between two JCSP processes, and consists of two parts that are responsible for serving the sending and the receiving process (resp.) according to the protocols that are already shown in the previous item:

  - $LEFT$ takes over messages sent to it by JCSP processes, transmits the messages to the other component $RIGHT$, and sends back receipts ack.
  - $RIGHT$ listens to read requests $ready$ that are raised by JCSP processes, takes over messages from the opponent $LEFT$, and finally hands over messages to the requesting JCSP processes.

- The class Parallel is modelled by CSP interleaving; channels between the processes are realised by channel processes $ONE2ONECHANNEL$ that are added to the network as a whole. For the example of Fig. 2.2, the execution of the network by calling

```
new Network().run();
```

leads to a system that is described by the following CSP term (after completion of the construction phase, i.e. after the creation of the channel and after spawning client and server process). We assume that $id_c \in Chan$ is the identifier of the channel that is used for communication between client and server.

$$\left(\begin{array}{c} id_c : ONE2ONECHANNEL \\ [\![id_c.\Sigma]\!] \\ T(\ \{\ \text{Succ.run}\,();\ \}\ ,\sigma)\ |||\ T(\ \{\ \text{Client.run}\,();\ \}\ ,\sigma) \end{array} \right) \setminus id_c.\Sigma$$

3. Specification and Verification of CSP Process Properties

In this section, Hennessy-Milner-Logic (HML) is described as a basic method for expressing propositions about processes. Though in practice HML is too weak to formulate properties of communicating systems, the additional features of more powerful logics—like the (modal) μ-calculus—are essentially orthogonal to the concepts already present in HML, and can hence by treated independently. In any case it is necessary to handle the issues that already turn up for HML, on which the thesis concentrates.

Two ways of designing calculi for HML in combination with the CSP process algebra for defining processes are shortly introduced. The calculi have both been implemented as part of this thesis, and have been used for the verification of fragments of Java code embedded in CSP terms, but have also been found not to be perfectly suited for this purpose.

The end of the section formulates and discusses requirements an analysis procedure for terms of the CSP algebra should meet, and sketches the method that is described in the remainder of the thesis.

3.1. Temporal Correctness Assertions

The principal statements that we use to make propositions about processes are *temporal correctness assertions*

$$T : \Phi$$

which are for instance described in [Sim95, DG00].

- *Syntactically*, such an assertion relates a term T of a formal language for processes (like a programming language or—in our case—a formal syntax for the CSP process algebra) with a modal proposition Φ (in the sequel formulated in Hennessy-Milner-Logic) about processes. Informally, the assertion states that the process described by T has property Φ.

- T is *semantically* interpreted as a process, i.e. according to the previous section as an element $\mathrm{val}_{\mathcal{L}}(T) \in \mathit{Carrier}(\mathrm{FinLTS}_{\mathcal{L}})$ of the carrier of a final labelled transition system over a certain set $\mathcal{L}$ of labels. The complete expression $T : \Phi$ is evaluated to a truth value by the evaluation of the modal formula Φ in the labelled transition system $\mathrm{FinLTS}_{\mathcal{L}}$ (regarded as a Kripke frame) and in the state $\mathrm{val}_{\mathcal{L}}(T)$ described by T (we assume that the binary function $\mathrm{val}_{\mathrm{FinLTS}_{\mathcal{L}}}$ performs this evaluation of modal formulas):

$$\mathrm{val}_{\mathcal{L}}(T : \Phi) \quad := \quad \mathrm{val}_{\mathrm{FinLTS}_{\mathcal{L}}}(\mathrm{val}_{\mathcal{L}}(T), \Phi) \quad \in \quad \mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$$

Correctness assertions can be embedded in propositional or predicate logic, and can in this context simply be regarded as formulas. Consequently, the assertions can be connected using propositional junctors, as in $T_1 : \Phi_1 \wedge T_2 : \Phi_2$, and a proof system for correctness assertions can be built on top of calculi like Gentzen-style sequent calculi [Gen35].

3.2. Hennessy-Milner-Logic

Hennessy-Milner-Logic (HML) [HM80] is a simple modal logic, which is—when reduced to its core connectives—syntactically described by the grammar

$$\mathit{For}_{\mathrm{HML}} \quad ::= \quad \mathit{true} \mid \neg \mathit{For}_{\mathrm{HML}} \mid \mathit{For}_{\mathrm{HML}} \wedge \mathit{For}_{\mathrm{HML}} \mid \langle a \rangle \mathit{For}_{\mathrm{HML}}$$

where a ranges over a domain of expressions for events (which could for instance be a term algebra). Studying HML in the context of processes is mostly interesting because it can be observed that HML characterises bisimulation (for image finite processes), i.e. two processes satisfy the same HML formulas if and only if the processes are bisimular.

The meaning of the connectives conforms to the usual semantics of modal logics; in particular, a formula $\langle a \rangle \Phi$ states that a concerned process is able to engage in an event described by a, and that in one of the states that can be reached by such an engagement Φ holds.

In general, the evaluation of HML formulas to truth values is defined inductively as follows. We assume that a set $\mathcal{L}$ of LTS labels is fixed and omit corresponding indices. The evaluation takes place in the final LTS $\mathrm{FinLTS}_{\mathcal{L}}$, and is (as above) carried out by a mapping $\mathrm{val}_{\mathrm{HML}}$ that takes two arguments: A process $P \in \mathit{Carrier}(\mathrm{FinLTS}_{\mathcal{L}})$ that determines the state in which a formula is evaluated, and a formula $\Phi \in \mathit{For}_{\mathrm{HML}}$. We write $\alpha_{\mathcal{L}} := \mathit{Dynamics}(\mathrm{FinLTS}_{\mathcal{L}})$ for the transition structure of the LTS.

$$\begin{aligned}
\mathrm{val}_{\mathrm{HML}}(P, \mathit{true}) &= \mathrm{tt} \\
\mathrm{val}_{\mathrm{HML}}(P, \neg\Phi) &= \begin{cases} \mathrm{ff} & \mathrm{val}_{\mathrm{HML}}(P, \Phi) = \mathrm{tt} \\ \mathrm{tt} & \text{otherwise} \end{cases} \\
\mathrm{val}_{\mathrm{HML}}(P, \Phi_1 \wedge \Phi_2) &= \begin{cases} \mathrm{tt} & \mathrm{val}_{\mathrm{HML}}(P, \Phi_1) = \mathrm{tt} = \mathrm{val}_{\mathrm{HML}}(P, \Phi_2) \\ \mathrm{ff} & \text{otherwise} \end{cases} \\
\mathrm{val}_{\mathrm{HML}}(P, \langle a \rangle \Phi) &= \begin{cases} \mathrm{tt} & \exists\, Q \in \alpha_{\mathcal{L}}(P)(\mathrm{val}(a)).\ \mathrm{val}_{\mathrm{HML}}(Q, \Phi) = \mathrm{tt} \\ \mathrm{ff} & \text{otherwise} \end{cases}
\end{aligned}$$

The term $\alpha_{\mathcal{L}}(P)(\mathrm{val}(a))$ of the last case describes the set of processes that are reachable from P with a transition labelled with the value of a. An evaluation of a can at this point be necessary and non-trivial, because no assumptions are made about the kind of expressions for LTS labels that are used in formulas; a could for instance be a term whose meaning is determined by an interpretation function.

3.2.1 Example: For the following example formulas, we choose the set $\mathcal{L} = \mathcal{L}_{CSP} = \Sigma \cup \Sigma_\tau$ of labels as in the last section when defining CSP. As expressions for events $a \in \Sigma \cup \Sigma_\tau$

in formulas we use the events themselves (equivalently we could assume that each event $a \in \Sigma \cup \Sigma_\tau$ is also regarded as a homonymous constant symbol and describe events by terms over this vocabulary).

The left formula states that a process is able to perform an execution step labelled with a, whereas Φ_2 holds for processes that after two arbitrary execution steps labelled with a are necessarily able to engage in a step b.

$$\Phi_1 \quad := \quad \langle a \rangle \mathit{true} \qquad\qquad \Phi_2 \quad := \quad \neg\langle a \rangle\langle a \rangle\neg\langle b \rangle \mathit{true}$$

The two formulas can be evaluated for different processes described using the CSP algebra (we assume $a \neq b$):

$$\begin{aligned}
\mathrm{val}_{\mathrm{HML}}(\mathit{stop}, \Phi_1) &= \mathrm{ff} & \mathrm{val}_{\mathrm{HML}}(a \to \mathit{stop}, \Phi_1) &= \mathrm{tt} \\
\mathrm{val}_{\mathrm{HML}}(b \to \mathit{stop}, \Phi_1) &= \mathrm{ff} & \mathrm{val}_{\mathrm{HML}}(a \to \mathit{stop} \mid\mid\mid b \to \mathit{stop}, \Phi_1) &= \mathrm{tt} \\
\mathrm{val}_{\mathrm{HML}}(a \to \mathit{stop}, \Phi_2) &= \mathrm{tt} & \mathrm{val}_{\mathrm{HML}}(a \to a \to b \to \mathit{stop}, \Phi_2) &= \mathrm{tt} \\
\mathrm{val}_{\mathrm{HML}}(a \to \mathit{stop} \mid\mid\mid a \to \mathit{stop}, \Phi_2) &= \mathrm{ff} & \mathrm{val}_{\mathrm{HML}}(a \to b \to \mathit{stop} \mid\mid\mid a \to \mathit{stop}, \Phi_2) &= \mathrm{tt}*
\end{aligned}$$

In order to investigate whether a process—described for instance by a CSP term—satisfies a particular HML formula, it is common practice to design some kind of calculus that syntactically analyses both the process term and the HML formula. Because HML formulas make propositions about labelled transition systems, such an analysis procedure can be expected to derive from a process term some kind of (perhaps implicit) intermediate representation of LTSs. Because HML is unable to distinguish between bisimular processes, as mentioned afore, different choices for this representation exist that all lead to correct results. The two approaches of analysis that are sketched in the next pages both primarily generate tree-like LTSs.

In Sect. 2, CSP is introduced semantically as a concrete algebra, i.e. in particular the connectives of CSP are defined to be mappings. Though for the syntactic analysis of CSP terms this has to be complemented with a formal syntax—a task that is performed for an extension of CSP in Sect. 6—this step is skipped for pure CSP in the thesis, and we illustrate the two methods of the next pages by considering CSP terms that actually belong to the meta-level of the considerations. For instance, we are going to use correctness assertions

$$a \to \mathit{stop} : \langle a \rangle \mathit{true}$$

in which the (meta-level) symbols that make up the term $a \to \mathit{stop}$ are in Sect. 2 defined to denote the corresponding mappings of CSP. We believe that at this point the explicit introduction of a further level of indirection, namely of a formal syntax, whose symbols are interpreted with the concrete connectives of CSP, does neither add to understanding nor exhibits new aspects.

3.3. A Calculus for CSP and Hennessy-Milner-Logic Using SOS Rules

A possible calculus for temporal correctness assertions for CSP is obtained by explicitly modelling the transition relation of labelled transition systems through *transition*

Table 3.1. Rules of a Gentzen-style Sequent Calculus that Treat Hennessy-Milner-Logic by Using Transition Assertions

$$\frac{\Gamma \vdash \Delta}{\Gamma,\ P : \mathit{true} \vdash \Delta} \qquad \frac{}{\Gamma \vdash P : \mathit{true},\ \Delta}$$

$$\frac{\Gamma \vdash P : \Phi,\ \Delta}{\Gamma,\ P : \neg\Phi \vdash \Delta} \qquad \frac{\Gamma,\ P : \Phi \vdash \Delta}{\Gamma \vdash P : \neg\Phi,\ \Delta}$$

$$\frac{\Gamma,\ P : \Phi_1,\ P : \Phi_2 \vdash \Delta}{\Gamma,\ P : \Phi_1 \wedge \Phi_2 \vdash \Delta} \qquad \frac{\Gamma \vdash P : \Phi_1,\ \Delta \qquad \Gamma \vdash P : \Phi_2,\ \Delta}{\Gamma \vdash P : \Phi_1 \wedge \Phi_2,\ \Delta}$$

$$\frac{\Gamma,\ P \xrightarrow{a} x,\ x : \Phi \vdash \Delta}{\Gamma,\ P : \langle a \rangle \Phi \vdash \Delta} \qquad \frac{\Gamma \vdash P \xrightarrow{a} Q,\ \Delta \qquad \Gamma \vdash Q : \Phi,\ \Delta}{\Gamma \vdash P : \langle a \rangle \Phi,\ \Delta}$$

(x is a fresh variable for processes with universal semantics)

assertions

$$P \xrightarrow{a} Q$$

which state that the process described by term P is able to evolve into Q through the engagement in the event given by a. Transition assertions can—like temporal correctness assertions—be regarded as formulas. To reason about transition assertions, rules of the structural operational semantics of a process formalism can directly be integrated in the analysis calculus. The classical paper describing this approach (for SOS rules in the GSOS format) is [Sim95], and the method is for instance used in the EVT system [ACD+03, Fre01] for verifying Erlang programs [AVWW96].

A Gentzen-style sequent calculus (other kinds of calculi can be designed similarly) for CSP correctness assertions using this approach essentially consists of two groups of rules (possibly apart from rules of an enclosing logic, like of first-order predicate logic). On the one hand, there are rules treating HML formulas within correctness assertions, which could for instance be the rules given in Table 3.1; these rules analyse HML formulas in the same way as propositional formulas are treated by a sequent calculus for classical propositional logic.

The most interesting rules of Table 3.1 are the last two ones, which describe how to translate the modal operator of HML into a proposition about the labelled transition system of a process. The rules can (intuitively) be derived from the equivalence of the following first-order formulas, of which the right one is a direct adaption of the definition of $\mathrm{val}_{\mathrm{HML}}(P, \langle a \rangle \Phi)$ from above:

$$P : \langle a \rangle \Phi \quad \equiv \quad \exists\, x . P \xrightarrow{a} x \wedge x : \Phi$$

A second group of rules describes the operational semantics and treats transition assertions; this part of the calculus is specific for the used formalism for processes, in this case for CSP. In [Sim95], a general translation of GSOS rules to sequent calculus rules is given. A similar adaption of the CSP rules of Table 2.1 yields rules like the

following ones (we only treat the prefixing operator as an example):

$$\frac{\Gamma,\ a \doteq b,\ P \doteq Q\ \vdash\ \Delta}{\Gamma,\ (a \to P) \xrightarrow{b} Q\ \vdash\ \Delta} \qquad \frac{\Gamma\ \vdash\ a \doteq b,\ \Delta}{\Gamma\ \vdash\ (a \to P) \xrightarrow{b} P,\ \Delta}$$

The translation generates rules for transition assertions in the antecedent and in the succedent, and uses an equality predicate $\doteq$ do state the identity of LTS labels and of processes. Both rules express that the process given by $a \to \mathit{stop}$ is exactly able to raise the event a, and that this step deterministically leads to the process described by P.

3.4. A Calculus for CSP and Hennessy-Milner-Logic Using Rewriting

The second analysis procedure for CSP terms we describe works by transforming terms to obtain *head normal forms*. This is performed in [Ros98] in the presence of the operator of internal choice, but can be carried out for the CSP fragment of Sect. 2.1 in a similar manner. The principal idea is to establish process terms of the shape

$$a_1 \to T_1 \,\Box\, a_2 \to T_2 \,\Box\, \cdots \tag{*}$$

by rewriting. Such terms directly enable propositions about the described processes because of the following equivalences[1] (we are again using first-order formulas to render the statements)

$$\begin{aligned} (a \to P) : \langle b \rangle \Phi \quad &\equiv \quad a \doteq b \wedge (P : \Phi) \\ (P \,\Box\, Q) : \langle b \rangle \Phi \quad &\equiv \quad (P : \langle b \rangle \Phi) \vee (Q : \langle b \rangle \Phi) \end{aligned}$$

This means that we can—as above for the calculus using transition assertions—design a set of sequent calculus rules that analyse HML formulas, and complement these rules with a second group to treat process terms. In fact, most of the rules of Table 3.1 can be kept for the first group: It is only necessary to replace the last two rules of the table by using the given equivalences for the HML modal operator:

$$\frac{\Gamma,\ a \doteq b,\ P : \Phi\ \vdash\ \Delta}{\Gamma,\ (a \to P) : \langle b \rangle \Phi\ \vdash\ \Delta} \qquad \frac{\Gamma\ \vdash\ a \doteq b,\ \Delta \qquad \Gamma\ \vdash\ P : \Phi,\ \Delta}{\Gamma\ \vdash\ (a \to P) : \langle b \rangle \Phi,\ \Delta}$$

$$\frac{\Gamma,\ P : \langle b \rangle \Phi\ \vdash\ \Delta \qquad \Gamma,\ Q : \langle b \rangle \Phi\ \vdash\ \Delta}{\Gamma,\ (P \,\Box\, Q) : \langle b \rangle \Phi\ \vdash\ \Delta} \qquad \frac{\Gamma\ \vdash\ P : \langle b \rangle \Phi,\ Q : \langle b \rangle \Phi,\ \Delta}{\Gamma\ \vdash\ (P \,\Box\, Q) : \langle b \rangle \Phi,\ \Delta}$$

The normalisation of a CSP process term can be accomplished recursively, i.e. in order to normalise $P = \mathit{op}(T_1, \ldots, T_n)$, first the subterms $T_1, \ldots, T_n$ are recursively

[1] In fact, the second equivalence is not valid because of the definition of choice $\Box$ regarding internal steps $\tau(a)$, which are interleaved. Another problem turns up when treating the hiding operator, because the prefixing operator cannot directly be used for internal steps. These problems can be overcome by introducing additional "pure" versions of both operators, i.e. a choice operator that does not distinguish between visible and invisible steps, and prefixing that can be used for all kinds of events alike. The details are omitted in the thesis.

transformed to shape (*), and subsequently the subterms are combined according to the semantics of *op* to establish a head normal form of P. When using modified versions of external choice and prefixing (as described in Footnote 1) on page 57, this can be performed for all CSP operators by suitable rewriting rules. For instance we can treat the hiding operator by rules like the following ones:[2]

$$\frac{\Gamma,\ T[P \setminus X \ \Box\ Q \setminus X] : \Phi \ \vdash\ \Delta}{\Gamma,\ T[(P \ \Box\ Q) \setminus X] : \Phi \ \vdash\ \Delta}$$

$$\frac{\Gamma,\ a \in X,\ T[\tau(a) \to (P \setminus X)] : \Phi \ \vdash\ \Delta \qquad \Gamma,\ a \notin X,\ T[a \to (P \setminus X)] : \Phi \ \vdash\ \Delta}{\Gamma,\ T[(a \to P) \setminus X] : \Phi \ \vdash\ \Delta}$$

Corresponding rules for the succedent can be given alike.

3.5. Analysis of CSP Process Terms in General

When using the calculi described in the last pages in practice, primarily two disadvantages have been observed:

- Both calculi tend to duplicate parts of process terms, which leads to large terms or/and to large sequents, for instance when making case distinctions. Though it is possible to cope with duplication of subterms by sharing terms, this does not seem to be an optimal solution for a calculus that is (also) supposed to be used interactively. A typical rule that duplicates subterms is for instance the total-order characterisation of interleaving, which resembles rules of the rewriting calculus of Sect. 3.4:

$$a \to P \ |||\ b \to Q \quad \rightsquigarrow \quad a \to (P \ |||\ b \to Q) \ \Box\ b \to (a \to P \ |||\ Q)$$

- Information about independence of certain computation steps that the process term contains is destroyed (see for instance [GvLH$^+$96] for a detailed account on dependency between transitions). This means, in particular, that concurrent events that are combined by interleaving are ordered totally by the calculi. This is a second objection against the rule describing interleaving from the last item, which destroys knowledge about the independence of the engagement in the two events a and b by making a case distinction on the two possible ways of arrangement (with either a or b being the first step).

In order to investigate CSP process terms for the evaluation of HML formulas, both calculi derive transition systems from the terms first. The procedure using SOS rules (Sect. 3.3) represents the derived system explicitly using transition assertions, whereas the method of Sect. 3.4 produces process terms that contain tree-like LTS representations using the operators of external choice and of prefixing. In both cases, repeatedly knowledge about initial parts of the behaviour of a process is collected: If a HML formula is analysed in top-down direction, as it is done by both calculi, then to treat a

[2] To represent rewriting rules within a sequent calculus, we use the notation $T[T']$ for distinguishing a subterm T' within a context $T[\cdot]$. This concept is for instance introduced in [DP01].

formula whose top-level operator is the modal operator, possible first execution steps of the process have to be known. For instance to decide about the assertion

$$\Big(a \to P \;|||\; b \to Q\Big) : \langle a \rangle \neg \langle b \rangle true$$

it is necessary to examine execution steps labelled with a processes described by $a \to P \;|||\; b \to Q$ are able to engage in.

To determine how information about initial process behaviour can be gathered, we first consider the "event flow" that takes place within a CSP term when executing the represented process. The investigation is only performed in an informal way, and is based on the structural operational semantics of CSP as given in Table 2.1.

Considering the way in which events are raised by the CSP operators defined in Sect. 2, it is possible to distinguish different roles that the operators might embody:

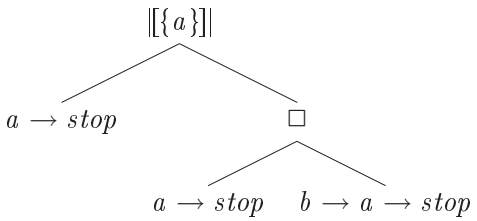
- An operator might be able to *synthesise* events, which is the case for prefixing $a \to P$ and for prefix choice $c\,?\,x \to P$.
- All other connectives only *propagate* events that are raised by their subprocesses. Propagated events are in many cases altered by the transmitting operator, e.g. replaced with events denoting internal steps by the hiding operator, or merged with events of another subprocess by synchronous parallelism. The occurrence of the operator can also be modified by the passing event, e.g. the external choice operator is removed after having delivered a visible action. These operators for themselves are passive and never offer "spontaneous" events to the environment.

Consequently, we can distinguish between two parts of a process term, namely, on the one hand, there is a collection of maximum (regarding the subterm relation) subterms whose top-level operator is active, and on the other hand these active subterms are combined by a context or *skeleton* of the term, which only consists of passive operators. The context C can be expressed as a term containing holes $\diamond$ (we use the same syntax as [DP01] for terms with holes), i.e. as an element of the term algebra over the vocabulary consisting of the passive CSP operators and the nullary symbol $\diamond$. An arbitrary process term P can be decomposed in

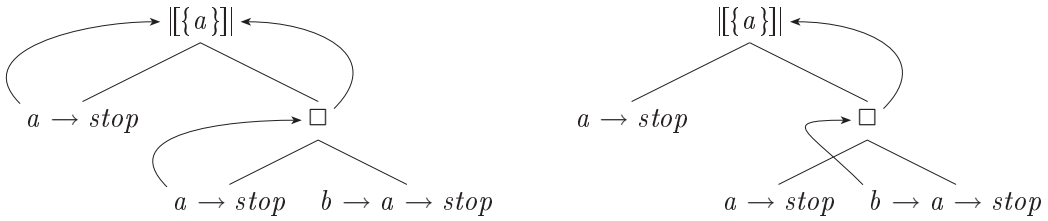
$$P \quad = \quad C[\,T_1, \ldots, T_k\,]$$

with a skeleton C and active process terms $T_1, \ldots, T_k$. This is illustrated for the following process:

$$\big(a \to stop\big) \;|[\,\{a\}\,]|\; \big(a \to stop \;\Box\; b \to a \to stop\big)$$
$$= \quad \Big(\diamond \;|[\{a\}\,]|\; (\diamond \;\Box\; \diamond)\Big)[\; a \to stop,\; a \to stop,\; b \to a \to stop\;]$$



The engagement of the complete system in events is always triggered by active subprocesses whose top-level operators offer certain events. Following the semantics of passive skeleton nodes that is given by the SOS rules, events are then handed on by these operators and "float" upwards to the root of the skeleton. The two events the example process is able to engage in are a and b:



To calculate the first steps of a process in general, it is necessary to analyse both the skeleton of the term and the first events that can be raised by active subprocesses, and to determine sets of subprocesses that are able to simultaneously engage in events, according to the skeleton. If events or parameters of skeleton operators are represented by terms (for instance the set $X = \{a\}$ in the running example), the analysis cannot be expected to be computable: To decide about the applicability of SOS rules treating parallelism, it is necessary to consider propositions like $a \in X$ and $a = b$. Hence, at least parts of the analysis have to be performed by a background procedure, for instance by a calculus for predicate logic, that provides knowledge about the particular event domain.

Requirements on a procedure for process term analysis (independently from the treatment of propositions about the event domain) that seem to be reasonable are:

- If the analysis procedure rewrites the process term, then it should not be necessary to duplicate parts of the process term.

- Information about independence of certain computation steps that the process term contains should be preserved by the calculus.

- As the structural operational semantics, which is the basis for constructing LTSs for process terms, assembles the dynamic behaviour of the process given by a term by referring to the behaviour of its subterms, most likely also a calculus will treat the skeleton in a somewhat recursive way. Intermediate results (like results about subtrees of the skeleton) that are obtained in this analysis should be stored in some way, and should be reused in later proof steps if possible. In the example, as intermediate information one could regard the fact that the subprocess $a \rightarrow stop \;\Box\; b \rightarrow a \rightarrow stop$ can either raise a or b in its first step and after that passes in the processes $stop$ or $a \rightarrow stop$.

In the remainder of the section, we sketch an analysis method that we believe to adhere to these requirements, i.e. that does not suffer from the problems described above.

3.6. Analysis of CSP Process Terms Using a Causal Extension of CSP

The procedure for analysing process terms we describe pursues an approach that is similar to the rewriting procedure for pure CSP of Sect. 3.4, but is based on an extension of the CSP algebra that is better suited for this purpose. Like in Sect. 3.4, head normal forms are established, from which it is possible to derive potential initial steps of a process, together with the particular continuations (i.e. the behaviour of the process after performing one of the initial execution steps). The analysis can also be guided by the same (bottom-up) recursive scheme as the pure CSP method, i.e. to normalise a process term $P = op(T_1, \ldots, T_n)$, first the subprocesses $T_1, \ldots, T_n$ are recursively normalised, and subsequently the subterms are combined according to the semantics of op (as mentioned afore, when using terms to represent the processes $T_1, \ldots, T_n$, then in general this step is not computable and it is necessary to introduce case distinctions). The leafs of the recursive invocation tree are determined by the active subterms (as introduced in the last section), because it is not necessary to descend any further once all possible first steps of a subprocess are known.

To encode head normal forms of processes, we extend the CSP process algebra with a secondary layer of causality, which is based on a restricted version of 1-safe Petri nets (introduced in Sect. 4.1 in detail). The head normal form that is established by the calculus of Sect. 3.4 is of the kind

$$a_1 \rightarrow T_1 \,\Box\, a_2 \rightarrow T_2 \,\Box\, \cdots \qquad (*)$$

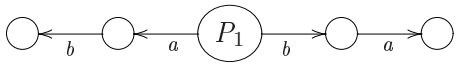
i.e. the initial behaviour of a process P is represented as a case distinction between all possible first execution steps. With the extended CSP algebra, we modify this normal form basically in two respects:

- Instead of presenting a list $T_1, T_2, \ldots$ of continuations of the process after the first step (all of which are usually very similar), possible continuations are encoded as a single term $T(M)$ that is parametrised with a configuration M. The subsequent process after step a_i can be derived by choosing the appropriate argument M_i and is represented by $T(M_i)$. This representation of continuations essentially solves the problem of duplicated process terms.

- The head normal form (*) provides no means of expressing information about *confluence* of the different alternatives. In fact, if the transformation to head normal form (*) is iterated to establish head normal form also for all subterms, then a tree-like LTS is obtained. To detect that for certain sets of (independent) steps the order of execution is actually not important, it is necessary to compare process terms afterwards, which is—given that a lot of information about independence is in fact provided by the process term *before* the transformation to head normal form—a method that is unnecessarily inefficient. To illustrate this phenomenon, we consider the process described by

$$a \rightarrow stop \;|||\; b \rightarrow stop \quad = \quad a \rightarrow b \rightarrow stop \,\Box\, b \rightarrow a \rightarrow stop \quad =: \quad P_1$$

The right term is a representation of a tree-like LTS of the process, and no information (apart from the equality of the corresponding subterms *stop*) is given from which the bisimularity of the two marginal states can derived:



This inefficiency is related to the fact that in P_1 the events a, b occur more than once, namely both as potential first and second steps. These multiple occurrences are caused by unique events a, b in the original term, which cannot be reconstructed from P_1 directly either.

To cope with both problems, we equip the collection $a_1, a_2, \ldots$ of first steps with a Petri net structure, using which the relation between the steps can be encoded accurately. In particular the independence of certain execution steps can be represented without choosing total orderings. The Petri net is not only used as a representation of one collection of initial steps, but it also contributes to the behaviour of the process after the first step and is continuously updated whenever one of the steps is performed.[3]

Head normal forms that are established by the analysis procedure can be regarded as a pair (N, T), which consists of a labelled place-transition net N (together with a marking) and a process term T. T is parametrised with markings of the Petri net N. As a characteristic property of the tuple, there is a one-to-one correspondence between initial execution steps of the net N and the initial execution steps of the original process P. First steps can thus be determined by exclusively considering the place-transition net N (which is a computable problem). The target state that is reached by performing a particular initial step is described by (N', T), where N' is obtained from N by altering the attached marking accordingly.

3.6.1 Example: The process described by the term

$$a \to stop \;|||\; b \to c \to stop$$

offers the independent initial steps a and b. The Petri net N of these first steps consists of two unconnected transitions that are labelled with a and b:

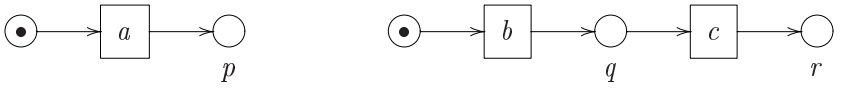
The parametrised continuation term has the shape

$$T \quad = \quad G_p(stop) \;|||\; G_q(c \to stop)$$

[3]While the analysis procedure can in principle also be used to translate a process term to a P/T net as a whole—provided that the term is only built from the CSP operators of Sect. 2.1, and does for instance not use recursion—we only perform this translation for initial steps. A complete translation is in general not viable, at least not for the kind of Petri nets used in this thesis, because for processes that cannot be described by finite state graphs also the resulting nets would have to be infinite.

where the symbols G_p, G_q represent guards that establish the relation between initial steps and the continuations. G_q states that step c can only be performed after step b. To determine whether the first step that was raised by the process is a or b, G_q refers to the marking of place q.

The net N is a complete description of the initial behaviour of the process, but it is incomplete regarding subsequent execution steps. We can extend N to cover also the second execution step by further translating the continuation T, which leads to the following net N' and continuation T':



$$T' \quad = \quad G_p(stop) \;|||\; G_r(stop)$$

For more complex process terms, this procedure can be iterated to explore LTSs of the corresponding processes up to an arbitrary (finite) depth. The used kind of P/T nets enables the description of arbitrary acyclic labelled transition systems, which is a representation of processes that is more efficient than the tree-like normal form (*). $*$

To realise the described analysis of CSP terms, we extend the process algebra with elements of Petri nets, mostly following the approach in [BB01]. Therefore, CSP is basically extended in three respects:

- A notion of *transitions* similar to the transitions of Petri nets is introduced. The corresponding operator is an extension of the CSP prefixing operator. Transitions allow the specification of token consumption and production requirements.
- The concept of *places* known from Petri nets is added to CSP: An operation on processes is defined for the declaration of places to which transitions can refer. Places are able to store tokens that are produced by transitions, and to provide stored tokens to transitions that need to consume any.
- To establish the *communication* between transitions and places, the event concept of CSP is extended. An execution step is no longer only labelled with an event $e \in \Sigma \cup \Sigma_\tau$, but furthermore LTS labels carry information about places from which tokens have to be consumed, or to which tokens have to be produced to perform the respective step.

4. Extending CSP with P/T Net Causality

We describe a process algebra based on CSP that provides additional features to specify causal relationships. This algebra uses concepts described in [BB01] for combining traditional process algebras with Petri nets.

The first part of the section defines and discusses a restricted variant of 1-safe place/-transition nets, on which the CSP extension is based. Subsequently, the operators of the CSP extension are described and formally defined. In order to relate the CSP extension with the original CSP algebra, we define an embedding of the latter formalism in the former one.

4.1. The Variant of P/T Nets Used in the CSP Extension

While most of the fundament of our CSP extension sticks closely to [BB01], instead of using general P/T nets for the modelling of causality we introduce a simplified version of 1-safe P/T nets, in which the semantics of places has been modified to restrict expressiveness. These nets, which are in the following pages called *monotonic P/T nets*, have a strong relation to occurrence nets [NPW80] that can be used to unfold general P/T nets or to represent partially order multisets (pomsets) as Petri nets. Occurrence nets are defined as a subclass of P/T nets by the imposition of certain properties, in particular of acyclicity, and it can be observed that on this subclass the monotonic modification of place semantics does (essentially) not have any influence on the meaning of nets. Monotonic P/T nets are hence sufficient to represent occurrence nets, and also to model the kinds of causality necessary for our CSP extension, but when compared to general P/T nets have the advantage of allowing for more convenient transformations and analysis.

The basic definitions of monotonic P/T nets essentially conform to the usual definition of 1-safe P/T nets, apart from a slightly non-standard representation of arcs that implicitly prevents a place to turn up more than once as input or output of a single transition. Differences to the standard definitions occur in the concept of markings of nets; we do not regard initial markings as inherent components of a P/T net either.

4.1.1 Definition (Monotonic P/T Nets): A *monotonic P/T net* is a tuple $(S, T, pre, post)$ consisting of two disjoint sets S, T of *places* and *transitions* together with two mappings $pre, post : T \to \mathcal{P}_{\text{fi}} S$, which assign each transition finite sets of input and output places. The pair $(pre(t), post(t))$ is called the *I/O* of a transition t.

A *marking* of a net $(S, T, pre, post)$ is a function $M : S \to \{\text{E}, \text{M}, \text{D}\}$ assigning each place one of the three states *empty*, *marked* and *dead*. The set of all markings is denoted with $\mathcal{M}_S$ and only depends on the set S of places the net contains. ∗

The different life cycles of places of normal 1-safe (boolean) P/T nets and of monotonic nets are given by the state graphs below. While the value of an ordinary place can alternate arbitrarily often during the execution of a P/T net, the places of monotonic P/T nets have a monotonic behaviour (which explains the denomination) and, once having been marked by putting a token in storage, can never get empty again but are only able to pass over to a terminal state D as soon as the token is consumed.

The dynamics of monotonic P/T nets, which is total-order according to the taxonomy of [BB01], is formalised in the next definition:

4.1.2 Definition (Dynamics of Monotonic P/T Nets): Suppose $(S, T, pre, post)$ is a monotonic P/T net and $M \in \mathcal{M}_S$ is a marking.

1. A transition $t \in T$ is called *enabled* for M, if input places are marked and pure output places are empty:

$$M(pre(t)) \subseteq \{\mathrm{M}\} \ \wedge \ M(post(t) \backslash pre(t)) \subseteq \{\mathrm{E}\}$$

   Because this condition only depends on the I/O of a transition, we also write $(pre(t), post(t)) \in \mathrm{En}(M)$ and define the set $\mathrm{En}(M) \subseteq (\mathcal{P}_{\mathrm{fi}} S)^2$ to consist of all I/O pairs that are enabled (which are not necessarily only pairs that really turn up in the P/T net).

2. If a transition $t \in T$ is enabled for M, then the P/T net is able to perform an execution step as t is able to *fire*. The effect of t firing is a modification of the marking M, the new marking after the execution step is $M' := M + (pre(t), post(t))$ and defined by

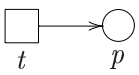
$$(M + (m,n))(p) \quad := \quad \begin{cases} \mathrm{D} & p \in m \backslash n \\ \mathrm{M} & p \in n \\ M(p) & \text{otherwise} \end{cases} \qquad \begin{array}{l} M \in \mathcal{M}_S, \\ m, n \in \mathcal{P}_{\mathrm{fi}} S \end{array}$$

*

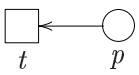
Note that the second condition $p \in n$ of the last case distinction could be strengthened to $p \in n \backslash m$, because the proposition $M(pre(t) \cap post(t)) \subseteq \{\mathrm{M}\}$ holds in any case for an enabled transition t. The difference between the definitions is important only when using the mapping + also for disabled transitions, which is done in some places of the next section.

The effect of firing of a transition can be illustrated by identifying the edges of the state graph of a monotonic place (as shown above) with the different kinds of I/O that can be attached to a transition: The edge between E and M corresponds to a pure output place of a transition, the self-loop of M to a place that is both input and output, and the edge between M and D is related to a pure input place. The firing of a transition causes the state of each place connected to the transition in one of the three different ways to progress according to the respective arc of the state graph, and a transition is enabled if (and only if) these state transitions are possible.

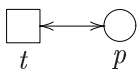
As usual, monotonic P/T nets can be depicted as bipartite directed graphs in which the two kinds of nodes are places (circles) and transitions (squares), and whose arcs represent the I/O of the transitions:



Production
$pre(t) = \varnothing$,
$post(t) = \{p\}$



Consumption
$pre(t) = \{p\}$,
$post(t) = \varnothing$



Testing
$pre(t) = \{p\}$,
$post(t) = \{p\}$

To denote values of places, at times we use the following notations for empty, marked and dead places. The first two representations of places conform to standard notation:

Empty Place

Marked Place

Dead Place

Conforming to intuition, and as it is the case for general P/T nets, the firing of a transition only affects the state of those places that are directly connected to the transition, which are also the only places the enabledness of a transition depends on. Besides that, the result of the firing is independent from the marking of other places:

$$M|_{m\cup n} = M'|_{m\cup n} \quad \Longrightarrow \quad \big((m,n) \in \mathrm{En}(M) \Longleftrightarrow (m,n) \in \mathrm{En}(M')\big) \tag{4.1}$$

$$(M + (m,n))|_{S\setminus(m\cup n)} \quad = \quad M|_{S\setminus(m\cup n)} \tag{4.2}$$

$$(M + (m,n))|_{m\cup n} \quad = \quad (M' + (m,n))|_{m\cup n} \tag{4.3}$$

To treat the CSP operator of parallelism $|[X]|$ in the CSP extension, it is eventually necessary to combine the input and output places of two transitions, i.e. to construct for two given transitions a third one whose behaviour resembles the behaviour of the two separate transitions in total. This is only possible for transitions that are *independent*, i.e. that have no directly connected places in common, as Def 4.1.1 imposes well-formedness conditions on monotonic P/T nets. It is for instance impossible to formulate nets that contain transitions unable to fire under any circumstances (there will always be markings of a net for which a transition is enabled).

4.1.3 Definition (Addition of I/O): Two I/O pairs (m_1, n_1), (m_2, n_2) are called *independent* if

$$(m_1 \cup n_1) \cap (m_2 \cup n_2) = \varnothing. \tag{4.4}$$

The sum of two independent pairs (m_1, n_1), (m_2, n_2) is defined to be

$$(m_1, n_1) + (m_2, n_2) \quad := \quad (m_1 \cup m_2, n_1 \cup n_2). \qquad *$$

4.1.4 Lemma (Addition of I/O): Suppose $(S, T, pre, post)$ is a monotonic P/T net, $M \in \mathcal{M}_S$ is a marking of the net and (m_1, n_1), (m_2, n_2), (m_3, n_3) are I/O pairs.

1. Provided that the three I/O pairs are pairwise independent, the addition of the pairs is associative. Furthermore, both sides of the equation are defined:

$$\big((m_1, n_1) + (m_2, n_2)\big) + (m_3, n_3) = (m_1, n_1) + \big((m_2, n_2) + (m_3, n_3)\big)$$

2. Addition of I/O is compatible with firing of transitions, provided that the concerned I/O pairs are independent:

$$M + \big((m_1, n_1) + (m_2, n_2)\big) = \big(M + (m_1, n_1)\big) + (m_2, n_2)$$

3. A partial ordering on I/O pairs is given by

$$(m, n) \leqslant (m', n') \quad :\Longleftrightarrow \quad \exists (m'', n''). \, (m', n') = (m, n) + (m'', n'').$$

The set $\mathrm{En}(M)$ is left-closed w.r.t. this ordering:

$$(m_1, n_1) \leqslant (m_2, n_2) \;\wedge\; (m_2, n_2) \in \mathrm{En}(M) \quad \Longrightarrow \quad (m_1, n_1) \in \mathrm{En}(M) \qquad *$$

4.1.1. Monotonic Behaviour of P/T Nets

The term "monotonic" as part of the name of our P/T nets is justified in the previous section by considering the life cycle of places, but it can as well be observed that the states of monotonic P/T nets as a whole evolve in a monotonic and acyclic way by firing transitions, which is a property that is used extensively in later sections to describe transformations of process terms. Therefore, the set $\{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ of place values is equipped with the total ordering $\mathrm{E} < \mathrm{M} < \mathrm{D}$, and it is clear that firing transitions always replaces the states of concerned places with new states in a non-decreasing manner.

The ordering on $\{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ induces a partial ordering on the set $\mathcal{M}_S$ of possible markings of a P/T net $(S, T, \mathit{pre}, \mathit{post})$ by point-wise comparison of markings:

$$M \leqslant M' \quad :\Longleftrightarrow \quad \forall \, p \in S. \; M(p) \leqslant M'(p)$$

From the life cycles of places, it follows immediately that the firing of a transition alters a marking in a non-decreasing way. Provided that the transition also consumes or produces tokens (and is not only testing the value of places), a marking is even strictly increased. These properties are ascertained in a slightly more general version in a lemma below.

The ordering on markings makes it possible to introduce a concept of *approximations*, and can be used to estimate the behaviour of a P/T net in the future following certain firing sequences.

4.1.5 Definition (Approximations of P/T Net Markings): Suppose that $(S, T, \mathit{pre}, \mathit{post})$ is a monotonic P/T net. A marking $M' \in \mathcal{M}_S$ of the net is called an *approximation* of the marking $M \in \mathcal{M}_S$, if $M' \leqslant M$. A transition $t \in T$ is called *potentially enabled* for a marking M', if t is enabled for an arbitrary marking M that is approximated by M'. Similarly to the notation for enabled transitions we write $(\mathit{pre}(t), \mathit{post}(t)) \in \mathrm{PotEn}(M)$ in this case and define the set $\mathrm{PotEn}(M) \subseteq (\mathcal{P}_{\mathrm{fi}} S)^2$ to consist of all I/O pairs that are potentially enabled. *

The notion of approximations is useful as it allows to make propositions about the state and the progression of P/T nets that are known only partially. In particular, the following lemma entails that a transition that is not potentially enabled for the current

marking of a net will never get enabled at all, as the state of the net does not become smaller by transitions firing. To make this statement, it is not necessary to know the transitions the P/T net actually consists of.

From the fact that the relation $\leqslant$ on markings is an ordering, it also follows that state graphs of any P/T net are acyclic, and that the execution of a finite P/T net necessarily terminates.

4.1.6 Lemma (Approximations of P/T Net Markings): Suppose $(S, T, pre, post)$ is a monotonic P/T net, $M, M' \in \mathcal{M}_S$ are markings of the net and (m, n), (m', n') are I/O pairs.

1. Every enabled I/O pair is also potentially enabled:
$$\mathrm{En}(M) \subseteq \mathrm{PotEn}(M)$$
2. The set of potentially enabled I/O pairs is left-closed:
$$(m, n) \leqslant (m', n') \quad \wedge \quad (m', n') \in \mathrm{PotEn}(M) \quad \Longrightarrow \quad (m, n) \in \mathrm{PotEn}(M)$$
3. A direct characterisation of potentially enabled I/O pairs is (we write $A \leqslant \mathrm{M}$ as abbreviation for $\forall\, x \in A.\ x \leqslant \mathrm{M}$):
$$(m, n) \in \mathrm{PotEn}(M) \quad \Longleftrightarrow \quad M(m) \leqslant \mathrm{M} \quad \wedge \quad M(n \backslash m) \leqslant \mathrm{E}$$
4. Firing of enabled transitions alters a marking in a non-decreasing way:
$$(m, n) \in \mathrm{PotEn}(M) \quad \Longrightarrow \quad M \leqslant M + (m, n)$$
5. If an enabled transition consumes or produces tokens, then its firing increases the marking:
$$(m, n) \in \mathrm{PotEn}(M) \quad \wedge \quad m \ominus n \neq \varnothing^1 \quad \Longrightarrow \quad M < M + (m, n)$$
6. A transition that consumes or produces tokens can be fired at most once:
$$m \ominus n \neq \varnothing \quad \Longrightarrow \quad (m, n) \notin \mathrm{PotEn}(M + (m, n))$$
7. The mapping $\mathrm{PotEn}(\cdot)$ is an order-isomorphism between the set $\mathcal{M}_S$ of markings and the set $\mathrm{PotEn}(\mathcal{M}_S)$ of sets of I/O pairs, equipped with the superset relation:
$$M \leqslant M' \quad \Longleftrightarrow \quad \mathrm{PotEn}(M') \subseteq \mathrm{PotEn}(M)$$
8. The firing of transitions is monotonic in the first argument:
$$M \leqslant M' \quad \Longrightarrow \quad M + (m, n) \leqslant M' + (m, n)$$
9. W.r.t. the ordering of I/O pairs of Lem. 4.1.4, the firing of transitions is monotonic in the second argument, provided that the concerned pairs are potentially enabled:
$$(m, n) \leqslant (m', n') \quad \wedge \quad (m', n') \in \mathrm{PotEn}(M) \quad \Longrightarrow \quad M + (m, n) \leqslant M + (m', n')$$
10. Adding input places to a transition alters the effect of firing in a non-decreasing way:
$$m \subseteq m' \quad \Longrightarrow \quad M + (m, n) \leqslant M + (m', n) \qquad *$$

[1] With $A \ominus B$ we denote the symmetric difference of two sets A, B: $A \ominus B := A \backslash B \cup B \backslash A$.

4.2. The Extension CCSP of CSP

In this section, we introduce an extension of CSP (called CCSP[2]), which we use to define an analysis procedure for process terms, and we describe the differences between the original CSP fragment—given in Sect. 2—and the extended CSP. The fundamental idea underlying the extension is the concept presented in [BB01], which is an approach that combines a traditional process algebra (in this case ACP [BK84]) with the causality model that governs the execution of a Petri net. To motivate this combination, and to discuss the differences between CSP and Petri nets, we start with equipping the monotonic P/T nets of the last section with a total-order operational semantics, i.e. we use nets as a description of processes. It needs to be stressed that in this construction two notions of transitions turn up, which—though being related—do not coincide:

- The transitions that make up a P/T net, i.e. the elements of component T of a P/T net $(S, T, pre, post)$.
- The accessibility relation of labelled transition systems $(\mathcal{S}, \alpha_{\mathcal{S}})$ (as in Sect. 2), i.e. the mapping $\alpha_{\mathcal{S}}$, which for each state/process $P \in \mathcal{S}$ determines possible execution steps, together with the assigned labels and the respective target processes.

4.2.1. An Operational Semantics for Monotonic P/T Nets

We assume that a monotonic P/T net $(S, T, pre, post)$ and a corresponding marking $M \in \mathcal{M}_S$ are given. To connect P/T nets with labelled transition systems, we furthermore assume that the transitions are assigned labels, namely each one an external or internal event as defined for CSP, by a mapping $\lambda : T \to \mathcal{L}_{CSP}$.

To regard the P/T net as a process description, a dynamic behaviour of the process, i.e. the possible execution steps and their effect, has to be defined. As usual, this is conducted by regarding the firing of transitions of the net as execution steps. A transition $t \in T$ is enabled for the marking M if (and only if) $(pre(t), post(t)) \in \mathrm{En}(M)$ (as in the last pages), and in this case the firing of t has the effect of updating M to $M + (pre(t), post(t))$. If we use the notation

$$ptProc((S, T, pre, post), M, \lambda)$$

to denote the process that is described by the net $(S, T, pre, post)$ and the initial marking M, and whose execution steps are labelled with events $\lambda(t)$, then the operational rule for monotonic P/T nets is

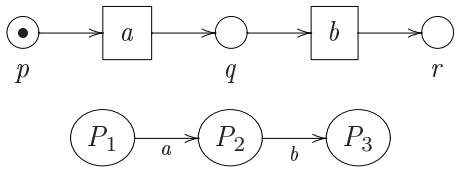
$$\frac{t \in T, \qquad (pre(t), post(t)) \in \mathrm{En}(M)}{ptProc((S, T, pre, post), M, \lambda) \xrightarrow{\lambda(t)} ptProc((S, T, pre, post), M + (pre(t), post(t)), \lambda)}$$

As for CSP processes, the rule describes labelled transition systems, and likewise it is possible to prove the existence of corresponding (and uniquely determined) states in the final system $\mathrm{FinLTS}_{\mathcal{L}_{CSP}}$ by using coinduction.[3]

[2] There is no direct relation to other concepts called CCSP, as for instance [EHD86].

[3] Strictly speaking, it is necessary to ascertain the image finite condition for this construction, which does not hold for monotonic P/T nets in general. Therefore, it would be necessary to enforce certain finiteness conditions, like that the P/T net is only allowed to contain finitely many transitions.

4.2.1 Example: We consider the P/T net N depicted in the upper diagram (labels $\lambda(t)$ of transitions are directly written in the transition boxes). The labelled net, together with the initial marking M_1 that is given in the diagram as well, describe the process $P_1 = ptProc(N, M_1, \lambda)$ represented by the labelled transition system below. All states of the system are of the kind $P_i = ptProc(N, M_i, \lambda)$, where the corresponding markings M_i are given in the table on the right.



	p	q	r
M_1	M	E	E
M_2	D	M	E
M_3	D	D	M

∗

4.2.2. P/T Net Causality for Arbitrary Processes

When comparing the concept of describing processes using P/T nets with the description by CSP terms, we can observe (besides other features) the following difference:

- When executing the process defined by a P/T net, the net itself, as well as the labelling λ, is not altered; more exactly, processes that are reachable by performing execution steps can alway be described using the same net and the same labelling function. The only component of the term $ptProc((S, T, pre, post), M, \lambda)$ that is not constant is the marking M, and when constructing labelled transition systems the carrier (i.e. the state space) can consequently be chosen to be the set $\mathcal{M}_S$ of markings. The transitions outgoing from each state can be identified as a subset of the P/T net transitions, namely as those transitions of the net that are enabled for the effective marking.

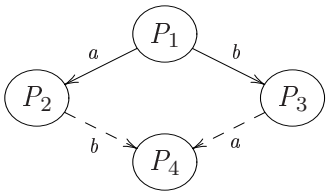
- CSP process terms (or terms of other process algebras) are by execution "worked off". It is in particular necessary to reduce occurrences of prefixing to synthesise events. When considering an LTS described by a process term, then in contrast to the P/T net case there is no direct relation between the collections of outgoing transitions of adjacent states, i.e. the offering of next execution steps can be altered arbitrarily by actually performing a step.

For combining the process algebra CSP with the P/T net causality concept, we add to CSP the idea of having a "reservoir" of execution steps available in principle, in which subsequently (by a selection method, namely by markings M and the semantics of places) enabled transitions are distinguish from disabled ones. The combination differs from P/T nets in the ability of arbitrarily modifying the transition reservoir in each step, which is the aspect taken from the process algebra world.

Potential Execution Steps: For the description of such reservoirs, in the first place we assume that an arbitrary process $P \in Proc_{\Sigma_0}$ is given. We are going to regard the collection $\alpha_{CSP}(P)$ of outgoing transitions (α_{CSP} as in Sect. 2.2.2) as the reservoir of possible next execution steps of P, or more exactly the set

$$R(\alpha_{CSP}(P)) \quad := \quad \{(e, Q) \in \mathcal{L}_{CSP} \times Proc_{\Sigma_0} \ : \ Q \in \alpha_{CSP}(P)(e)\}$$

For the following process, this means for instance



$$R(\alpha_{CCSP}(P_1)) = \{(a, P_2), (b, P_3)\}$$

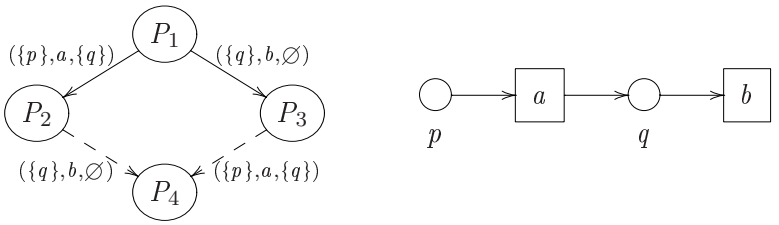
To realise a concept of secondary selection of steps, we assign the set $R(\alpha_{CSP}(P))$ a monotonic P/T net $(S, R(\alpha_{CSP}(P)), pre, post)$, i.e. the collection is regarded as set of transitions of a net. The set S of places is constant and chosen a priori. For defining the functions *pre*, *post* describing edges of the net, we enrich LTS labels (which are just events for the original CSP) with I/O information. This means that labels of the labelled transition systems used are then given by the set

$$\mathcal{L}_{CCSP} \quad := \quad \mathcal{P}_{\text{fi}} S \times \mathcal{L}_{CSP} \times \mathcal{P}_{\text{fi}} S \quad = \quad \mathcal{P}_{\text{fi}} S \times (\Sigma \cup \Sigma_\tau) \times \mathcal{P}_{\text{fi}} S$$

where the first and the last component of a triple $(m, e, n) \in \mathcal{L}_{CCSP}$ are used as an I/O pair (m, n) as in Sect. 4.1. For $((m, e, n), Q) \in R(\alpha_{CSP}(P))$, *pre* and *post* are defined by

$$pre\big(((m, e, n), Q)\big) \; := \; m \qquad post\big(((m, e, n), Q)\big) \; := \; n$$

The example process from above could be adapted to the different label format for instance as follows (the I/O information of labels is chosen arbitrarily). $R(\alpha_{CCSP}(P_1))$ is depicted as a net in the right diagram, and consists of the two transitions of P_1 together with the places S.[4]

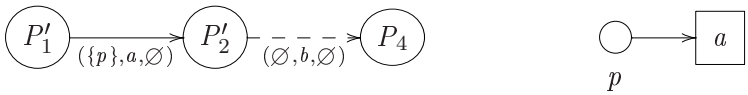


Enabled Execution Steps: In order to perform the secondary selection of *enabled* steps, we introduce a *causal state operator*, which both assigns (initial) states to places, and which hides places of the process by removing them from the labels the LTS contains. The assignment of a particular initial value to a place removes all transitions from the LTS that are disabled for this value, leaving over only a subset of enabled ones. The version of the state operator that is used in this thesis assigns a value to one place at a time (contrary to the operator defined in [BB01], which treats sets of places), and can for instance be used to assign the initial value E to place q in the running example:

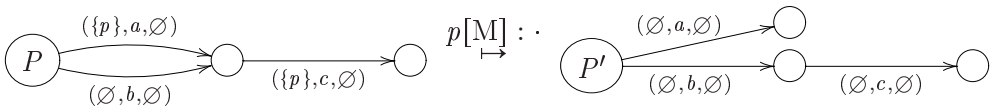
$$P_1' \quad := \quad q[\mathrm{E}] : P_1$$

[4] To make diagrams of P/T nets more concise, we leave out places that are not referred to by any transition.

In the resulting LTS (below), the transition labelled with event b is no longer available as a first step of the process, so the state graph is no longer branching. Accordingly, the P/T net of the first steps of P_1' only consists of a single transition. Note that the place q does not turn up in labels of the LTS anymore, as the state operator also hides places.



The state operator does not only modify the first steps of a process, but it also realises the progression of the value of a place, and accordingly can remove further disabled transitions. If we for instance assign place p the initial value M, then the process described by the left LTS (below) is mapped to a process that is unable to perform the trace ac. Namely, the step labelled with $(\{p\}, a, \varnothing)$ alters the value of p by consuming a token, and for this reason the subsequent step $(\{p\}, c, \varnothing)$ is disabled and has to be removed.



Process Spaces: Contrary to the CSP fragment defined in Sect. 2.1, the process space of CCSP is not only parametrised with an alphabet Σ defined through a set Σ_0 of atomic events, but also with an unstructured set S of places. Similarly to Sect. 2.2.2, processes of CCSP are regarded as elements of the final labelled transition system $\mathrm{FinLTS}_{\mathcal{L}_{CCSP}}$, and we use the notations

$$\begin{aligned} \mathit{CProc}_{\Sigma_0,S} &:= \mathit{Carrier}(\mathrm{FinLTS}_{\mathcal{L}_{CCSP}}) \\ \alpha_{CCSP} &:= \mathit{Dynamics}(\mathrm{FinLTS}_{\mathcal{L}_{CCSP}}) \end{aligned}$$

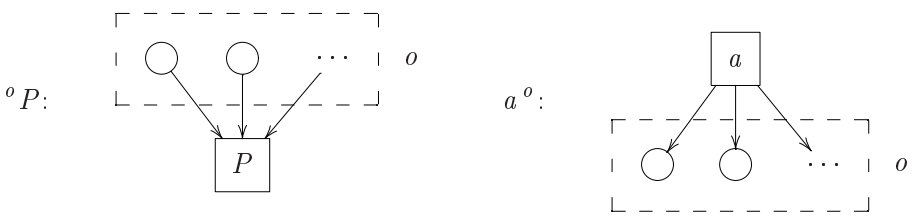
for the carrier and the dynamics of the LTS.

4.2.3. Operators of CCSP

Compared to CSP, and apart from the different kind of LTS labels, for CCSP a number of new operators are introduced, and in one case we also modify an operator of vanilla CSP and define it in a more general way. Process terms of the original CSP can be translated to CCSP in a canonical way, however, and it is proved in the end of the section that this translation preserves the semantics of processes (Sect. 4.4).

In the next paragraphs, operators specific to CCSP are described informally. For all operators that are not mentioned in the following pages, the introduction from Sect. 2.1.1 applies. The only exception from this scheme is the prefix choice operator, which we leave out for CCSP in the report. Prefix choice can be treated using concepts similar to those given for prefixing, however. The implementation of the described procedure that was created as part of the thesis also contains full support for prefix choice.

While the operators for expressing processes that consume or produce P/T net tokens differ from the corresponding ones in [BB01] in a number of details, we essentially use the same notation. This means that writing a set of places to the left of a term—as in ${}^{o}P$—in general points to a process that consumes tokens as specified by o, whereas the addition of a set on the right side—as in a^{o}—expresses the production of tokens. The diagrams below show the meaning of these notations schematically. In the left diagram, P actually represents a whole process or a whole P/T net instead of only a single transition. Nesting of processes in diagrams resembles a situation that can only turn up when using terms; regarding plain P/T nets, the meaning of the left diagram is undefined.



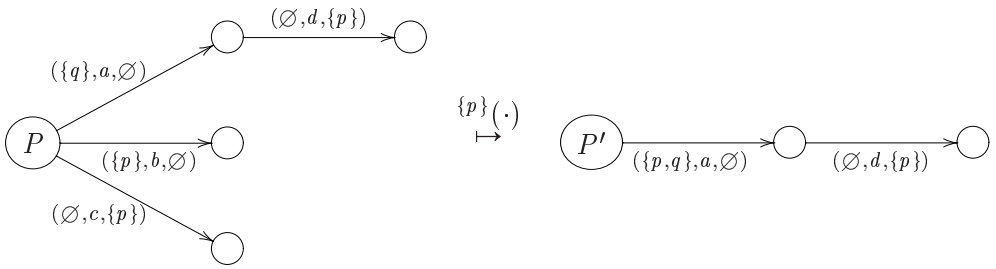
Token Consumption

$$ {}^{o}P, \qquad o \in \mathcal{P}_{\text{fi}}S, \quad P \in CProc_{\Sigma_0,S} $$

The first operator specific to CCSP makes processes consume tokens from each element of a finite set o of places. This consumption is performed in the first step the resulting process performs (and only then), i.e. to any triple (m, a, n) that can be raised by the subprocess P in the first step the places o are added as input places: In total, the triple that is raised is $(m \cup o, a, n)$, provided that $(m \cup n) \cap o = \emptyset$.[5]

$$ \frac{P \xrightarrow{(m,a,n)} P' \qquad (m \cup n) \cap o = \emptyset}{{}^{o}P \xrightarrow{(m \cup o,a,n)} P'} $$

As an example, we apply the mapping ${}^{\{p\}}(\cdot)$ to the process of the following diagram. The transitions labelled with b and c are removed by the application, because place p already turns up in the attached labels.



[5] We do not allow the step to be performed for $m \cap o \neq \emptyset$. Intuitively, this would require places of P/T nets to contain more than one token, which is not possible for 1-safe nets. The case $n \cap o \neq \emptyset$ is forbidden because this would cause token consumption to be replaced with testing for the places in $n \cap o$, which is a behaviour that leads to algebraic properties less convenient.

Token Consumption Upon (In)Visible Steps

$$^{o}{*}\,P, \qquad o \in \mathcal{P}_{\mathrm{fi}} S, \quad P \in CProc_{\Sigma_0,S}$$

To be able to define reduction rules for the operator of external choice, a second operator for token consumption is defined, which distinguishes between visible and invisible steps performed by the subprocess P. To any leading invisible step, i.e. to any engagement in a triple $(m, \tau(a), n)$, the places o are both added as input and output places. Consequently, such steps test the values of the places o without consuming tokens (this corresponds to bidirectional arcs in a P/T net). Upon the first visible step, the resulting process behaves like ^{o}P, i.e. it consumes tokens from o.

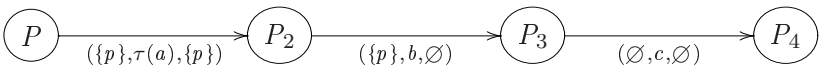
$$\frac{P \xrightarrow{(m',\tau(a),n')} P' \qquad (m,n) = (m',n') + (o,o)}{^{o}{*}\,P \xrightarrow{(m,\tau(a),n)} {}^{o}{*}\,P'}$$

$$\frac{P \xrightarrow{(m',a,n)} P' \qquad a \in \Sigma, \;\; (m \cup n) \cap o = \varnothing}{^{o}{*}\,P \xrightarrow{(m \cup o,a,n)} P'}$$

To illustrate the operator, we consider the following process, which first performs an invisible and then two visible steps:

$$P \quad := \quad ^{\{p\}}{*}\big((a \to b \to c \to stop) \setminus \{a\}\big)$$

An LTS that describes the process is



Transitions

$$^{o}\,a\,^{r}, \qquad a \in \Sigma \cup \Sigma_\tau, \quad o, r \in \mathcal{P}_{\mathrm{fi}} S$$

As a simple way of expressing processes that are able to perform exactly one deterministic execution step, we introduce a new operator for transitions. The special case $^{\varnothing}\,a\,^{\varnothing}$ with empty sets of places has about the same meaning as the process $a \to stop$, with the only difference that for transitions we also allow a to be an internal event (contrary to prefixing). In general, the process $^{o}\,a\,^{r}$ is exactly able to raise the triple (o, a, r), which means that the left set o of places corresponds to the argument of the token consumption operator, whereas r represents places for which tokens are produced by execution.

In CCSP, transitions are the only way of describing processes that produce tokens, contrary to the process algebra given in [BB01], where an operator symmetric to token consumption exists for token production. For our purpose of creating a CSP calculus, such a general operator is not necessary, however, because the strategy of the calculus is to break up process terms by reducing everything to transitions. The existence of a token production operator would make transitions as an independent operation

superfluous, because transitions can be derived from prefixing and token consumption/production.

$$\overline{{}^{o}a^{\,r} \xrightarrow{(o,a,r)} stop}$$

To achieve a shorter notation, we leave out empty sets as arguments of the transition operator, and thus we write

$$a \;:=\; {}^{\varnothing}a^{\varnothing}, \qquad {}^{o}a \;:=\; {}^{o}a^{\varnothing}, \qquad a^{\,r} \;:=\; {}^{\varnothing}a^{\,r}$$

The concept of CCSP transitions is very similar to the transitions of P/T nets, and the pair (o, r) is comparable to the I/O $(pre(t), post(t))$ of such a transition. Essentially, there are two differences, namely on the one hand CCSP transitions are always labelled with events (which is not the case at least for our definition of P/T net transitions). On the other hand, a P/T net transition is in principle able to fire repeatedly, contrary to CCSP transitions, which entails that in general duplicating a CCSP transition (and connecting the instances with interleaving) will modify the behaviour of a process. For P/T nets, duplication of transitions has no effect. Using monotonic P/T nets, the second distinction is less important, however, because according to Sect. 4.1.1 a transition with $pre(t) \ominus post(t) \neq \varnothing$ is never able to fire more than once either. This issue is discussed in more detail in Sect. 5.4.

Given a P/T net $(S, T, pre, post)$ with only finitely many transitions T and a labelling function $\lambda : T \to \mathcal{L}_{CSP}$ (as in Sect. 4.2.1), we can construct a CCSP term that has—up to the different notions of multiplicity of transitions—the same meaning as the labelled P/T net:

$${}^{pre(t_1)}\lambda(t_1)^{post(t_1)} \;|||\; \cdots \;|||\; {}^{pre(t_k)}\lambda(t_k)^{post(t_k)} \qquad T = \{t_1, \ldots, t_k\}$$

Causal State Operator

$$p[v] : P, \qquad p \in S, \quad v \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}, \quad P \in CProc_{\Sigma_0,S}$$

The state operator is described in the beginning of the section. Intuitively, the application of $p[v] : \cdot$ to a process assigns place p the initial value v, and also removes p from labels that are attached to transitions of the process. The semantics of monotonic places is directly encoded in the four operational rules that describe the mapping (Table 4.1). The rules correspond to the possible relations between a P/T net transition and a place: The two entities are either unrelated, or the place is an input or an output place of the transition (or both).

Parallelism

$$P \;[\![\, L[X]R \,]\!]\; Q, \qquad X \subseteq \Sigma, \quad L, R \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP}, [6] \quad P, Q \in CProc_{\Sigma_0,S}$$

The parallel operator of CCSP is an extended version of the corresponding CSP operator $|[X]|$. As for the original one, the steps (m, a, n) with $a \in X$ raised by subprocesses

[6] With $\mathrm{mult}_{\mathrm{fi}}\, A$ we denote the set of finite multisets of A. Formally, we define $\mathrm{mult}_{\mathrm{fi}}\, A$ to be the free commutative monoid with basis A, and we use the common set-like notation for multisets. The connective of the monoid is denoted with $\uplus$.

Table 4.1. Operational Semantics of CCSP: Causal State Operator

$$\frac{P \xrightarrow{(m,a,n)} P' \qquad p \notin m,\ \ p \notin n}{p[v] : P \xrightarrow{(m,a,n)} p[v] : P'}$$

$$\frac{P \xrightarrow{(m',a,n')} P' \qquad (m',n') = (m,n) + (\{p\},\{p\})}{p[\mathrm{M}] : P \xrightarrow{(m,a,n)} p[\mathrm{M}] : P'}$$

$$\frac{P \xrightarrow{(m',a,n')} P' \qquad (m',n') = (m,n) + (\varnothing,\{p\})}{p[\mathrm{E}] : P \xrightarrow{(m,a,n)} p[\mathrm{M}] : P'}$$

$$\frac{P \xrightarrow{(m',a,n')} P' \qquad (m',n') = (m,n) + (\{p\},\varnothing)}{p[\mathrm{M}] : P \xrightarrow{(m,a,n)} p[\mathrm{D}] : P'}$$

Table 4.2. Operational Semantics of CCSP: Parallelism

$$\frac{P \xrightarrow{(m,a,n)} P' \qquad a \notin X}{P \,[\![\, L[X]R \,]\!]\, Q \xrightarrow{(m,a,n)} P' \,[\![\, L[X]R \,]\!]\, Q} \qquad \frac{Q \xrightarrow{(m,a,n)} Q' \qquad a \notin X}{P \,[\![\, L[X]R \,]\!]\, Q \xrightarrow{(m,a,n)} P \,[\![\, L[X]R \,]\!]\, Q'}$$

$$\frac{P \xrightarrow{(m_1,a,n_1)} P' \qquad Q \xrightarrow{(m_2,a,n_2)} Q' \qquad a \in X,\ \ (m,n) = (m_1,n_1) + (m_2,n_2)}{P \,[\![\, L[X]R \,]\!]\, Q \xrightarrow{(m,a,n)} P' \,[\![\, L[X]R \,]\!]\, Q'}$$

$$\frac{P \xrightarrow{(m_1,a,n_1)} P' \qquad R = R' \uplus (m_2,a,n_2) \qquad a \in X,\ \ (m,n) = (m_1,n_1) + (m_2,n_2)}{P \,[\![\, L[X]R \,]\!]\, Q \xrightarrow{(m,a,n)} P' \,[\![\, L[X]R' \,]\!]\, Q}$$

$$\frac{L = L' \uplus (m_1,a,n_1) \qquad Q \xrightarrow{(m_2,a,n_2)} Q' \qquad a \in X,\ \ (m,n) = (m_1,n_1) + (m_2,n_2)}{P \,[\![\, L[X]R \,]\!]\, Q \xrightarrow{(m,a,n)} P \,[\![\, L'[X]R \,]\!]\, Q'}$$

Table 4.3. Operational Semantics of CCSP

$$\frac{}{(a \to P) \xrightarrow{(\varnothing,a,\varnothing)} P} \qquad \frac{}{{}^{o}a^{\,r} \xrightarrow{(o,a,r)} stop}$$

$$\frac{P \xrightarrow{(m,a,n)} P' \qquad (m \cup n) \cap o = \varnothing}{{}^{o}P \xrightarrow{(m \cup o,a,n)} P'}$$

$$\frac{P \xrightarrow{(m',\tau(a),n')} P' \qquad (m,n) = (m',n') + (o,o)}{{}^{o*}P \xrightarrow{(m,\tau(a),n)} {}^{o*}P'}$$

$$\frac{P \xrightarrow{(m',a,n)} P' \qquad a \in \Sigma,\ \ (m \cup n) \cap o = \varnothing}{{}^{o*}P \xrightarrow{(m \cup o,a,n)} P'}$$

$$\frac{P \xrightarrow{(m,\tau(a),n)} P'}{P \,\square\, Q \xrightarrow{(m,\tau(a),n)} P' \,\square\, Q} \qquad \frac{Q \xrightarrow{(m,\tau(a),n)} Q'}{P \,\square\, Q \xrightarrow{(m,\tau(a),n)} P \,\square\, Q'}$$

$$\frac{P \xrightarrow{(m,a,n)} P' \qquad a \in \Sigma}{P \,\square\, Q \xrightarrow{(m,a,n)} P'} \qquad \frac{Q \xrightarrow{(m,a,n)} Q' \qquad a \in \Sigma}{P \,\square\, Q \xrightarrow{(m,a,n)} Q'}$$

$$\frac{P \xrightarrow{(m,a,n)} P' \qquad a \in \Sigma}{l : P \xrightarrow{(m,l.a,n)} l : P'} \qquad \frac{P \xrightarrow{(m,\tau(a),n)} P'}{l : P \xrightarrow{(m,\tau(a),n)} l : P'}$$

$$\frac{P \xrightarrow{(m,a,n)} P' \qquad a \notin X}{P \setminus X \xrightarrow{(m,a,n)} P' \setminus X} \qquad \frac{P \xrightarrow{(m,a,n)} P' \qquad a \in X}{P \setminus X \xrightarrow{(m,\tau(a),n)} P' \setminus X}$$

have to be synchronised, whereas in other cases interleaving takes place. Synchronisation of two triples (m, a, n), (m', a', n') is only possible for $a = a'$ and if the two steps do not refer to the same places: $(m \cup n) \cap (m' \cup n') = \varnothing$. The label of the merged execution step is in this case $(m \cup m', a, n \cup n')$. Interleaving does not alter the label of a step. This gives for instance

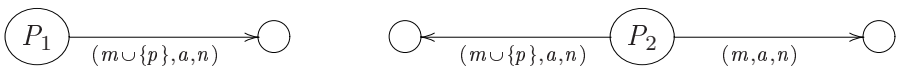
$$\begin{aligned} {}^{\{p\}}a^{\{q\}} \,[\![\, \varnothing[\{a\}]\varnothing \,]\!]\, {}^{\{r\}}a &= {}^{\{p,r\}}a^{\{q\}} \\ {}^{\{p\}}a^{\{q\}} \,[\![\, \varnothing[\{a\}]\varnothing \,]\!]\, {}^{\{p\}}a &= stop \\ {}^{\{p\}}a^{\{q\}} \,[\![\, \varnothing[\{a\}]\varnothing \,]\!]\, {}^{\{p\}}b &= {}^{\{p\}}b \end{aligned}$$

Compared to Sect. 2.1.1, the operator is extended by adding two "buffers" L, R, which are represented by finite multisets of LTS labels. This does not enlarge the expressiveness of the operator in principle, but is used for book-keeping when designing a reduction rule for parallelism (see Sect. 5.5). A triple $(m, a, n) \in R$ with $a \in X$[7] has to a certain degree the same meaning as a transition ${}^{m}a^{n}$ attached to subprocess Q using interleaving, e.g.

$$\underbrace{P \,[\![\, \{(\varnothing, a, \varnothing)\}[X]\{(m, a, n)\} \,]\!]\, Q}_{=:P_1} \quad \approx \quad \underbrace{P \,[\![\, \{(\varnothing, a, \varnothing)\}[X]\varnothing \,]\!]\, \left({}^{m}a^{n} \;|||\; Q\right)}_{=:P_2}$$

The buffer element ${}^{m}a^{n} \in R$ is regarded as "passive", however, and can only be raised in combination with an engagement of the opposing subprocess P, and *not* combined with an element of the opposing buffer L. For $P = Q = stop$, LTSs of the two processes shown are thus

If the situation is modified by choosing $P = {}^{\{p\}}a^{\varnothing}$ for $p \notin m \cup n$, then the processes described by the terms are



The operational rules for parallelism are shown in Table 4.2. The first two rules are responsible for interleaving events, whereas the other three ones treat synchronisation of events from different sources, namely merging of steps performed by the subprocesses P and Q, and merging of a step that is performed by either P or Q with an element of the opposing buffer.

Buffers are used for the intermediate storage of events that are offered by subprocesses, but that are synchronised and are still waiting for a matching event raised by the opposing subprocess. The notion of "passiveness" of events is introduced to ensure that the reduction of CCSP process terms terminates, i.e. to ascertain that a reduction rule describing synchronisation is not repeatedly applied to the same pair of events (Sect. 5.5).

[7] Triples with $a \notin X$ as elements of buffers do not have any effect and can simply be removed without altering the process. The buffers L, R are only used to treat synchronised steps.

For the common case of arguments that are empty sets, we define the following abbreviations:

$$P \mid\!\mid\!\mid Q \quad := \quad P \,[\![\, \varnothing[\varnothing]\varnothing \,]\!]\, Q, \qquad P \,[\![\, X \,]\!]\, Q \quad := \quad P \,[\![\, \varnothing[X]\varnothing \,]\!]\, Q$$

4.2.2 Example: The new operators can be used to describe causal relationships between parts of a process term. Provided that p is a "new" place, we have for instance (this transformation is described in detail in Sect. 5.3)

$$a \to P \quad = \quad p[\mathrm{E}] : \Big(a^{\,\{p\}} \mid\!\mid\!\mid^{\{p\}} P \Big)$$

Causal relationships can also be imposed across other operators, for instance to pull out interleaved events (this is treated in Sect. 5.5):

$$\big(a \to P\big) \,[\![\, \{b\} \,]\!]\, Q \quad = \quad p[\mathrm{E}] : \Big(a^{\,\{p\}} \mid\!\mid\!\mid \big({}^{\{p\}}P \,[\![\, \{b\} \,]\!]\, Q \big) \Big)$$

Note that a corresponding transformation only using CSP operators does not exist, in particular prefixing cannot be pulled out directly:

$$\big(a \to P\big) \,[\![\, \{b\} \,]\!]\, Q \quad \neq \quad a \to \big(P \,[\![\, \{b\} \,]\!]\, Q\big) \qquad *$$

4.3. Formal Definition of the CCSP Operators

The operational rules for the CCSP operators introduced so far are collected in the Tables 4.1, 4.2 and 4.3, and are mostly an adaption of the CSP SOS rules (Table 2.1) to the different label format and the different parallel composition operator. A few rules are added to treat the new operators for token consumption, transitions and the causal state operator. To handle I/O pairs, the addition of such pairs defined in Def. 4.1.3 is used extensively.

As in Sect. 2.2.2, for formal reasoning about processes we define the CCSP operators as mappings on the final labelled transition system $\mathrm{FinLTS}_{\mathcal{L}_{CCSP}}$. Therefore, a translation of the operational rules into equivalent characteristic equations is given in Table 4.4 and Table 4.5. The existence of (unique) solutions is ensured using coinduction, and by performing the same construction as in Sect. 2.2.2: An LTS is defined whose carrier consists of those terms that are necessary to describe the semantics of the operators (as the dynamics of the LTS), and by employing the finality of $\mathrm{FinLTS}_{\mathcal{L}_{CCSP}}$.

To make the equations more concise, in Table 4.4 (and also in the remainder of the thesis) we use the following notations:

- We write
$$P \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad := \quad \alpha_{CCSP}(P)((m, e, n))$$
for the finite set of processes reachable from P in one step by the engagement in the triple (m, e, n). $\cdot \Big\downarrow \begin{smallmatrix} \cdot \\ \cdot \\ \cdot \end{smallmatrix} \Big\downarrow$ is defined to have precedence over any other operator in the left argument.

Table 4.4. Characteristic Equations Defining CCSP

$$stop\Big\downarrow^{m}_{e}_{n}\Big\downarrow = \varnothing$$

$$(a \to P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow = \begin{cases} \{P\} & (m,e,n) = (\varnothing, a, \varnothing) \\ \varnothing & \text{otherwise} \end{cases}$$

$${}^{o}a^{r}\Big\downarrow^{m}_{e}_{n}\Big\downarrow = \begin{cases} \{stop\} & (m,e,n) = (o,a,r) \\ \varnothing & \text{otherwise} \end{cases}$$

$$({}^{o}P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow = \begin{cases} P\Big\downarrow^{m\backslash o}_{e}_{n}\Big\downarrow & o \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases}$$

$$({}^{o*}P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow = \begin{cases} {}^{o*}\left(P\Big\downarrow^{m\backslash o}_{e}_{n\backslash o}\Big\downarrow\right) & e \in \Sigma_\tau, o \subseteq m \cap n \\ P\Big\downarrow^{m\backslash o}_{e}_{n}\Big\downarrow & e \in \Sigma, o \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases}$$

$$(P \,\Box\, Q)\Big\downarrow^{m}_{e}_{n}\Big\downarrow = \begin{cases} P\Big\downarrow^{m}_{e}_{n}\Big\downarrow \,\Box\, Q \;\cup\; P \,\Box\, Q\Big\downarrow^{m}_{e}_{n}\Big\downarrow & e \in \Sigma_\tau \\ P\Big\downarrow^{m}_{e}_{n}\Big\downarrow \;\cup\; Q\Big\downarrow^{m}_{e}_{n}\Big\downarrow & e \in \Sigma \end{cases}$$

$$(l : P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow = \bigcup_{e \in \Sigma_\tau} l : P\Big\downarrow^{m}_{e}_{n}\Big\downarrow \;\cup\; \bigcup_{\substack{a \in \Sigma \\ e = l.a}} l : P\Big\downarrow^{m}_{a}_{n}\Big\downarrow$$

$$(P \setminus X)\Big\downarrow^{m}_{e}_{n}\Big\downarrow = \bigcup_{e \notin X} P\Big\downarrow^{m}_{e}_{n}\Big\downarrow \setminus X \;\cup\; \bigcup_{\substack{a \in X \\ e = \tau(a)}} P\Big\downarrow^{m}_{a}_{n}\Big\downarrow \setminus X$$

For $p \notin m \cup n$: (otherwise: $(p[v] : P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow = \varnothing$)

$$(p[v] : P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow = \; p[v] : P\Big\downarrow^{m}_{e}_{n}\Big\downarrow \;\cup$$
$$\begin{cases} p[\mathrm{M}] : P\Big\downarrow^{m}_{e}_{n\cup\{p\}}\Big\downarrow & v = \mathrm{E} \\ p[\mathrm{D}] : P\Big\downarrow^{m\cup\{p\}}_{e}_{n}\Big\downarrow & \\ \quad \cup\, p[\mathrm{M}] : P\Big\downarrow^{m\cup\{p\}}_{e}_{n\cup\{p\}}\Big\downarrow & v = \mathrm{M} \\ \varnothing & v = \mathrm{D} \end{cases}$$

Table 4.5. Characteristic Equations Defining CCSP: Parallelism

$$(P \,|[\, L[X]R \,]|\, Q)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow = \begin{cases} P\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \,|[\, L[X]R \,]|\, Q & \\ \qquad \cup\ P \,|[\, L[X]R \,]|\, Q\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow & e \notin X \\ D(P \,|[\, L[X]R \,]|\, Q, (m,e,n)) & e \in X \end{cases}$$

where D is defined as follows:

$$D(P \,|[\, L[X]R \,]|\, Q, (m,e,n)) = \bigcup_{\substack{m_1,n_1,m_2,n_2 \in \mathcal{P}_{\text{fi}} S \\ (m,n)= \\ (m_1,n_1)+(m_2,n_2)}} \begin{array}{ll} & P\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,|[\, L[X]R \,]|\, Q\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\ \cup & P \,|[\, L\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R \,]|\, Q\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\ \cup & P\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,|[\, L[X]R\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow]|\, Q \end{array}$$

- To handle the operator of parallelism from above, the notation for dynamics is continued to the multisets L that turn up as buffers of this operator. Because of the semantics of parallelism, a buffer

  $$L \quad := \quad \{(o_1, a_1, r_1), \ldots, (o_k, a_k, r_k)\} \quad \in \text{mult}_{\text{fi}}\, \mathcal{L}_{CCSP}$$

  has about the same meaning as a process consisting of interleaved transitions:

  $$P \quad := \quad {}^{o_1}a_1{}^{r_1} \,|||\, \cdots \,|||\, {}^{o_k}a_k{}^{r_k}$$

  The definition of the notation $L\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$ corresponds to the dynamics of P, which is exactly able to perform one of the steps (o_i, a_i, r_i), and thereby passes in a target process P' that consists of the same transitions as P apart from ${}^{o_i}a_i{}^{r_i}$:

  $$L\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad := \quad \begin{cases} \{L \backslash \{(m,e,n)\}\} & (m,e,n) \in L \\ \varnothing & \text{otherwise} \end{cases}$$

- All operators are continued in canonical manner to sets of processes or other kinds of arguments, and this is also performed for operators with more than one argument. This gives for instance

  $$\begin{array}{rcl} l : P\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow & = & \{l : P' \mid P' \in P\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\} \\ P\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,|[\, L[X]R \,]|\, Q\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow & = & \{P' \,|[\, L[X]R \,]|\, Q' \mid P' \in P\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow, Q' \in Q\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow\} \end{array}$$

We explain two of the characteristic equations in detail:

- The equation treating the parallel composition operator (Table 4.5) primarily makes a distinction between the interleaving and the synchronisation case. For

the former case, the first two operational rules of Table 4.2—which describe that each of the subprocesses can make an execution step that is directly passed on—are translated into set notation.

Synchronisation is handled by the second part of the characteristic equation, which ranges over all possible decompositions of the I/O pair (m, n), and which for each case collects the steps that can either be performed by the subprocesses simultaneously, of by one of the subprocesses in combination with the opposing buffer. The decomposition of I/O corresponds to the fact that I/O is joined when merging steps of subprocesses, as it is defined by the last three rules of Table 4.2. Note that we consider the equation $(m, n) = (m_1, n_1) + (m_2, n_2)$ to imply that (m_1, n_1) and (m_2, n_2) are independent, i.e. $(m_1 \cup n_1) \cap (m_2 \cup n_2) = \varnothing$, because otherwise the addition of I/O pairs is undefined.

- For the causal state operator (the last part of Table 4.4), we can first observe that the operator hides the argument place p, and hence resulting processes are never able to engage in steps (m, e, n) that contain p. The right side of the characteristic equation essentially consists of four terms, which correspond to the four operational rules for the state operator in Table 4.1. The terms that are part of the case distinction on v encode the life cycle of a monotonic place, and describe that the value of place p can progress from 'empty' to 'marked', or from 'marked' to 'dead'. As a third situation, the value 'marked' can be tested without altering it. The fourth term—that is not part of the case distinction—states that transitions that are unrelated to p do not have any effect on its value.

4.4. Embedding of CSP in CCSP

While CSP and CCSP differ both in syntax and semantics, a translation (at least in one direction) it possible with little effort. Therefore, we can first observe that the category $\underline{\mathrm{LTS}}_{\mathcal{L}_{CSP}}$ can be regarded as a sub-category of $\underline{\mathrm{LTS}}_{\mathcal{L}_{CCSP}}$, because the reservoir $\mathcal{L}_{CSP}$ of labels of the former one can be embedded into the set $\mathcal{L}_{CCSP}$ by regarding CSP events as CCSP triples with empty I/O:

$$E : \begin{cases} \mathcal{L}_{CSP} \to \mathcal{L}_{CCSP} \\ e \mapsto (\varnothing, e, \varnothing) \end{cases}$$

Accordingly, each labelled transition system over the set $\mathcal{L}_{CSP}$ can also be regarded as an LTS over $\mathcal{L}_{CCSP}$: The finite image condition (Def. 2.2.1) will not violated by renaming labels of an LTS, if the renaming is injective.

We can in particular regard the final CSP process space $\mathrm{FinLTS}_{\mathcal{L}_{CSP}}$ as an object of $\underline{\mathrm{LTS}}_{\mathcal{L}_{CCSP}}$. Hence, there is a unique "homomorphism" $\mathrm{FinLTS}_{\mathcal{L}_{CSP}} \to \mathrm{FinLTS}_{\mathcal{L}_{CCSP}}$ into the CCSP process space, i.e. a mapping

$$cc_{\Sigma_0,S} : Proc_{\Sigma_0} \to CProc_{\Sigma_0,S}$$

between the carriers of the labelled transition systems that preserves the accessibility relation up to renaming of labels: For all processes $P \in Proc_{\Sigma_0}$ and all events $e \in \Sigma \cup \Sigma_\tau$,

we have

$$cc_{\Sigma_0,S}(P)\Big\downarrow\begin{smallmatrix}\varnothing\\ e\\ \varnothing\end{smallmatrix}\Big\downarrow \quad = \quad cc_{\Sigma_0,S}(\alpha_{CSP}(P)(e))$$

and for $m, n \in \mathcal{P}_{\text{fi}} S$ with $m \cup n \neq \varnothing$:

$$cc_{\Sigma_0,S}(P)\Big\downarrow\begin{smallmatrix}m\\ e\\ n\end{smallmatrix}\Big\downarrow \quad = \quad \varnothing$$

The existence of the mapping ensures that arbitrary processes of the CSP world can also be considered as processes of the CCSP framework. The effective translation of processes that are described using CSP terms is supported by the following lemma, which states that $cc_{\Sigma_0,S}$ is essentially also a homomorphism of the CSP operators (that are mapped to their CCSP counterparts). For this reason, the evaluation of CSP terms using the CSP and the CCSP semantics is basically equivalent. Proofs of the given propositions can be conducted by using coinduction.

4.4.1 Lemma (Translation of CSP to CCSP): Suppose that $P, Q \in Proc_{\Sigma_0}$ are CSP processes, $a, l \in \Sigma$ are visible events and $X \subseteq \Sigma$ is an interface set.

The following propositions hold, in which the symbols on the left side denote the CSP operators, whereas on the right side CCSP operators are used:

$$\begin{aligned}
cc_{\Sigma_0,S}(stop) &= stop \\
cc_{\Sigma_0,S}(a \to P) &= a \to cc_{\Sigma_0,S}(P) \\
cc_{\Sigma_0,S}(P \,\Box\, Q) &= cc_{\Sigma_0,S}(P) \,\Box\, cc_{\Sigma_0,S}(Q) \\
cc_{\Sigma_0,S}(P \,[\![\, X \,]\!]\, Q) &= cc_{\Sigma_0,S}(P) \,[\![\, \varnothing[X]\varnothing \,]\!]\, cc_{\Sigma_0,S}(Q) \\
cc_{\Sigma_0,S}(l : P) &= l : cc_{\Sigma_0,S}(P) \\
cc_{\Sigma_0,S}(P \setminus X) &= cc_{\Sigma_0,S}(P) \setminus X \qquad *
\end{aligned}$$

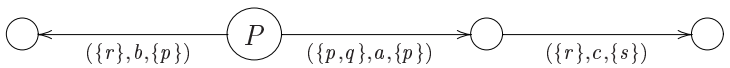
5. Algebraic Properties of CCSP

In this section, a number of algebraic properties of CCSP are described and proved that underly the calculus for CCSP. The section starts with two parts consisting of general notions and propositions, which are later referred to in all other considerations, but that are (in most cases) not directly related to a calculus for CCSP. A third part that is more fundamental, which is in particular used in the subsequent Sect. 5.5, is Sect. 5.4. The remaining issues treated correspond to certain parts of the calculus for CCSP, namely

- Synthesis of transitions from prefixing, and elimination of external choice
- Distribution of the general operator of parallelism through interleaving
- Distribution of labelling and hiding through interleaving
- Distribution of token consumption through interleaving.

5.1. Free Places of a Process

As a prerequisite for later propositions, we introduce a notion of *free places* of a process, which are those places that can turn up in a triple (m, a, n) that is raised by the process after an arbitrary number of execution steps. For instance, the process P represented by the following LTS is considered to contain the free places p, q, r, s.



We call places that turn up in arbitrary (reachable) transitions *free*, because such places can be bound (and are hidden) by the application of the causal state operator. It is also possible to define collision-free α-conversions, which do not alter a process, in the usual way; a method for renaming places is given in Sect. 6.3.

It needs to be stressed that the notion of free places is *not* related to the place value 'empty' (which is defined in Sect. 4.1 about monotonic P/T nets). On the contrary: The free occurrence of a place in a process entails that a value of the place is yet undetermined. Given the process P from above, in which p is a free place, it is possible to derive different instances by assigning p an initial value (which at the same time binds p, and hence in the resulting processes p is no longer free):

$$p[\mathrm{E}] : P, \qquad p[\mathrm{M}] : P, \qquad p[\mathrm{D}] : P$$

For processes that are described using CCSP terms, it is simple to derive safe estimations on the set of places occurring (the exact set depends on the dynamic behaviour

of a process), but as processes are treated on a purely semantic level at the time, we also have to define a semantic concept of free places. Therefore, free places of a process are collected by using fixed-point iteration along the corresponding labelled transition systems.

5.1.1 Example: A few processes together with the sets of free places are

$$\begin{aligned} FP(a^{\,o}) &= o \\ FP(p[\mathrm{E}] : {}^{m}a^{\,n}) &= \begin{cases} (m \cup n)\backslash\{p\} & p \notin m \\ \varnothing & p \in m \end{cases} \\ FP(a^{\,\{p\}} \;|||\; b^{\,\{q\}} \;|||\; c^{\,\{p\}}) &= \{p, q\} \\ FP({}^{m}a^{\,n} \;|[\,\{a\}\,]|\; {}^{l}b^{\,o}) &= l \cup o \end{aligned}$$

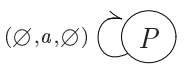
The value of $FP(p[\mathrm{E}] : {}^{m}a^{\,n})$ is explained by the observation that for $p \in m$ we have $p[\mathrm{E}] : {}^{m}a^{\,n} = \mathit{stop}$ (in this case the transition either needs to consume a token from the place p, or it is testing the value of p, which is both not possible if p is empty). Otherwise the state operator at least conceals p.

The last process ${}^{m}a^{\,n} \;|[\,\{a\}\,]|\; {}^{l}b^{\,o}$ is only able to perform step (l, b, o), because the left transition requires synchronisation, but there is no suitable step offered by the right side. *

The equation describing the set of free places of a process is well-known from program analysis and from model-checking, and defines the set in a recursive way by referring to the free places of adjacent processes, together with the places that occur in labels of outgoing transitions:

$$FP(P) = \bigcup_{\substack{(m,e,n)\in\mathcal{L}_{CCSP},\\ Q\in P\,\downarrow\,{}^{m}_{n}\,\downarrow}} m \cup n \cup FP(Q), \qquad P \in \mathit{CProc}_{\Sigma_0,S}. \tag{5.1}$$

Solutions FP of the family of equations are not unique, because for processes P that are part of cycles of the labelled transition system it is possible to choose $FP(P)$ almost arbitrarily. For instance in



any set $FP(P)$ satisfies the corresponding equation. Because we would like FP only to report those places that really turn up in any label, we are interested in the smallest (simultaneous) solution of the equations. This solution is constructed by employing the Knaster-Tarski fixed-point theorem [Tar55]. The domain of the iteration is the complete lattice of all mappings of the carrier of $\mathrm{FinLTS}_{\mathcal{L}_{CCSP}}$ to the power set of the set S of places:

$$C := \mathit{CProc}_{\Sigma_0,S} \to \mathcal{P}S \tag{5.2}$$

We can regard the right side of Eq. (5.1) as an order-preserving function $I : C \to C$ almost directly; the existence of a smallest fixed-point of I is then guaranteed by the

Knaster-Tarski theorem.

$$I(c)(P) \quad := \quad \bigcup_{\substack{(m,e,n)\in\mathcal{L}_{CCSP},\\ Q\in P\left\downarrow {m \atop e \atop n}\right\downarrow}} m \cup n \cup c(Q), \qquad c \in C,\ \ P \in CProc_{\Sigma_0,S}.$$

5.1.2 Definition (Free Places of a Process): For a process $P \in CProc_{\Sigma_0,S}$, the set of *free places* of P is defined to be

$$FP(P) \quad := \quad (\mathrm{lfp}\, I)(P) \qquad *$$

We conclude the section with a lemma that relates the free places of processes with the effect of CCSP operators. A lot of further propositions of the same kind hold, but are omitted in this report.

5.1.3 Lemma (Free Places and CCSP): Let $P \in CProc_{\Sigma_0,S}$ be a process, $p \in S$ be a place and $v \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ be a place value.

1. Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label. Places that turn up in LTS labels of initial steps a process is able to perform are free places of the process, and the places that occur free in adjacent processes also turn up in a process itself:

$$m \cup n \nsubseteq FP(P) \quad \Longrightarrow \quad P\left\downarrow {m \atop e \atop n}\right\downarrow = \varnothing \tag{5.3}$$

$$Q \in P\left\downarrow {m \atop e \atop n}\right\downarrow \quad \Longrightarrow \quad FP(Q) \subseteq FP(P) \tag{5.4}$$

2. The application of the causal state operator to a process P has no effect, if the concerned place p does not occur free in P:

$$p \notin FP(P) \quad \Longrightarrow \quad p[v] : P = P$$

3. The causal state operator hides places:

$$FP(p[v] : P) \quad \subseteq \quad FP(P) \backslash \{p\} \qquad *$$

5.2. Fundamental Properties of the CCSP Operators

The first CCSP identities we describe state that the operators of parallelism and external choice are commutative, and with certain restrictions associative. These properties are well-known and are only recapitulated at this point together with a few further ones briefly.

5.2.1 Lemma (Basic Properties of Choice and Parallelism): Suppose P, Q, $T \in CProc_{\Sigma_0,S}$ are processes, $L, R \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP}$ are finite multisets of labels, $X \subseteq \Sigma$ is an interface set and $(o, a, r) \in \mathcal{L}_{CCSP}$ is a label.

1. The set $CProc_{\Sigma_0,S}$ of CCSP processes together with the external choice operator $\Box$ form a commutative monoid,[1] i.e. the following properties hold:

$$\begin{array}{rrcl} \text{Commutativity:} & P \,\Box\, Q & = & Q \,\Box\, P \\ \text{Associativity:} & (P \,\Box\, Q) \,\Box\, T & = & P \,\Box\, (Q \,\Box\, T) \\ \text{Neutral element:} & P \,\Box\, stop & = & P \end{array}$$

2. For parallelism, the following laws hold, provided that $a \notin X$:

$$P \,[\![\, L[X]R \,]\!]\, Q = Q \,[\![\, R[X]L \,]\!]\, P \tag{5.5}$$
$$(P \,[\![\, X \,]\!]\, Q) \,[\![\, X \,]\!]\, T = P \,[\![\, X \,]\!]\, (Q \,[\![\, X \,]\!]\, T) \tag{5.6}$$
$$P \,[\![\, L \uplus \{(o,a,r)\}[X]R \,]\!]\, Q = P \,[\![\, L[X]R \,]\!]\, Q \tag{5.7}$$
$$P \,|||\, stop = P \tag{5.8}$$
$$P \,[\![\, L[X]R \,]\!]\, stop = P \,[\![\, \varnothing[X]R \,]\!]\, stop \tag{5.9}$$
$$stop \,[\![\, X \,]\!]\, stop = stop \tag{5.10}$$

*

The next class of CCSP laws deals with the causal state operator. The laws can be used to normalise process terms by pulling out occurrences of the operator, i.e. we show that the state operator commutes with other CCSP operators, provided that no collisions occur. Together with laws that enable bound renaming of places, it is possible to turn an arbitrary CCSP term into a prenex form

$$P \quad \rightsquigarrow \quad p_1[v_1] : p_2[v_2] : \ldots p_k[v_k] : P'$$

which makes the analysis of terms much easier. The prenex form can also be used to design a calculus that deals with open process terms, i.e. with terms that describe processes in which places occur free, which enables the encoding of multiple processes through a single process term and a set of different markings of the free places (Sect. 6).

The identities for pulling out the state operator are collected in the following lemma. For the commutative operators (parallelism and external choice) only one instance of the law is provided, the symmetric versions hold likewise and can be derived by using Lem. 5.2.1.

5.2.2 Lemma (Commutation of Causal State Operator): Suppose that $P, Q \in CProc_{\Sigma_0,S}$ are processes, $p, q \in S$ are places, $v, u \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ are place values, $o \in \mathcal{P}_{\mathrm{fi}} S$ is a finite set of places, $a, l \in \Sigma$ are visible events, $L, R \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP}$ are finite multisets of labels and $X \subseteq \Sigma$ is an interface set.

Two occurrences of the state operator treating different places can be permuted: For $p \neq q$:

$$p[v] : q[u] : P \quad = \quad q[u] : p[v] : P \tag{5.11}$$

[1] Note that by our notion of processes the law of idempotence does not hold for external choice: The identity on the set $CProc_{\Sigma_0,S}$ distinguishes members that are weakly bisimular, i.e. that only differ in invisible steps. For instance we have $\tau(a) \neq \tau(a) \,\Box\, \tau(a)$.

Prefixing, labelling and hiding can be commuted with the causal state operator:

$$a \to p[v] : P \quad = \quad p[v] : a \to P \tag{5.12}$$

$$l : p[v] : P \quad = \quad p[v] : l : P \tag{5.13}$$

$$(p[v] : P) \setminus X \quad = \quad p[v] : (P \setminus X) \tag{5.14}$$

Consumption of tokens can be commuted with the state operator, provided that the treated places are disjoint: For $p \notin o$:

$$^{o}p[v] : P \quad = \quad p[v] : {}^{o}P \tag{5.15}$$

$$^{o*}p[v] : P \quad = \quad p[v] : {}^{o*}P \tag{5.16}$$

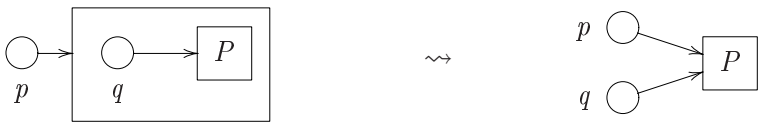
Provided that no collisions occur, the state operator can be pulled out of arguments of external choice and parallelism: For $p \notin FP(Q)$, $p \notin FP(L)$, $p \notin FP(R)$:[2]

$$(p[v] : P) \,\square\, Q \quad = \quad p[v] : (P \,\square\, Q) \tag{5.17}$$

$$(p[v] : P) \,|[\, L[X]R \,]|\, Q \quad = \quad p[v] : (P \,|[\, L[X]R \,]|\, Q) \tag{5.18}$$

*

The two operators for the consumption of tokens are parametrised with finite sets of places. Nested occurrences can in both cases be contracted by joining the parameter sets, for instance $^{o}(^{r}P) = {}^{o \cup r}P$, provided that the sets are disjoint. Intuitively, this situation is depicted by the two diagrams, in which the inner process ^{r}P is represented by "nesting" P/T nets. For $o \cap r \neq \varnothing$, the term $^{o}(^{r}P)$ expresses the consumption of multiple tokens from particular places, which is not possible for 1-safe nets.



It is also possible to commute alternating nested occurrences as in $^{o}(^{r*}P) = {}^{r*}(^{o}P)$. Not surprisingly the operators have no effect if the parameter set is empty. These observations are collected in the following lemma:

5.2.3 Lemma (Properties of Token Consumption): Suppose that $P \in CProc_{\Sigma_0,S}$ is a process and $o, r \in \mathcal{P}_{\mathrm{fi}}S$ are finite sets of places.

Nested consumption of tokens can be contracted:

$$^{o}(^{r}P) \quad = \quad \begin{cases} {}^{o \cup r}P & r \cap o = \varnothing \\ stop & \text{otherwise} \end{cases} \tag{5.19}$$

$$^{o*}(^{r*}P) \quad = \quad \begin{cases} {}^{o \cup r*}P & r \cap o = \varnothing \\ stop & \text{otherwise} \end{cases} \tag{5.20}$$

In any case alternating token consumption can be commuted:

$$^{o}(^{r*}P) \quad = \quad {}^{r*}(^{o}P) \tag{5.21}$$

[2] We write $p \notin FP(L)$ for the statement that p must not occur in any element of the multiset L, i.e. that for all labels $(m, e, n) \in L$ the proposition $p \notin m \cup n$ holds.

Alternating token consumption from the same places blocks a process: For $r \cap o \neq \varnothing$:

$$ {}^{o}({}^{r*}P) \quad = \quad {}^{r*}({}^{o}P) \quad = \quad stop \tag{5.22} $$

Token consumption has no effect for empty sets of places:

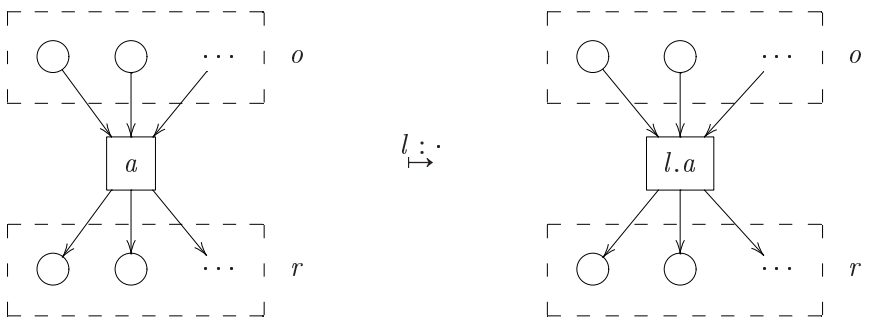
$$ {}^{\varnothing}P \quad = \quad {}^{\varnothing*}P \quad = \quad P \tag{5.23} $$

Blocked processes are not altered by token consumption:

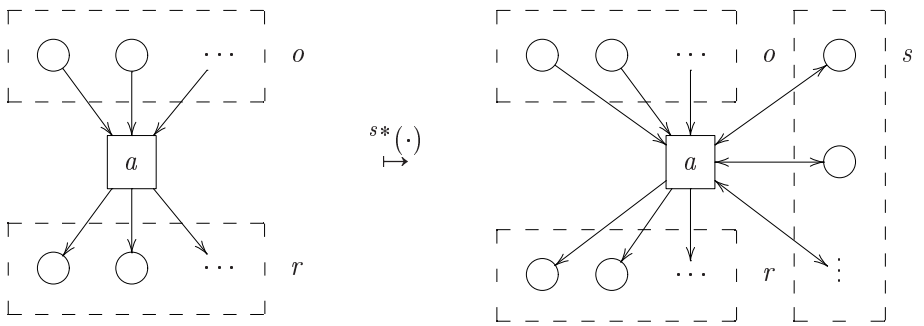
$$ {}^{o}stop \quad = \quad {}^{o*}stop \quad = \quad stop \tag{5.24} $$

*

The calculus we are about to define reduces complex CCSP terms to terms that only consist of interleaving and transitions. Because of the particularly simple behaviour of transitions, the application of "passive" operators like hiding or token consumption to transitions can easily be carried out. For labelling and hiding, it is only necessary to alter the event of a transition; the application of the former operator to a transition ${}^{o}a^{r}$ can for instance be depicted as (for $a \in \Sigma$, as labelling has no effect for internal steps)[3]



The application of the token consumption operators to transitions ${}^{o}a^{r}$ has an effect similar to the contraction of nested token consumption (Lem. 5.2.3), and can be performed by modifying only the I/O pair (o, r). The most interesting case is the application of starred token consumption, as in ${}^{s*}({}^{o}a^{r})$, which for visible steps $a \in \Sigma$ has the same meaning as normal token consumption ${}^{s}({}^{o}a^{r})$, whereas to transitions denoting internal steps $a \in \Sigma_\tau$ the places of the set o are attached both as input and output. The last situation is shown in the following diagrams:



Algebraic laws that describe the transformations are collected in the next lemma.

[3] The sets o, r are not necessarily disjoint, i.e. in this respect the representation as diagrams can lead to a wrong conception.

5.2.4 Lemma (Properties of Transitions): Suppose that $o, r, s \in \mathcal{P}_{\mathrm{fi}} S$ are finite sets of places, $a \in \Sigma \cup \Sigma_\tau$ is an event, $l \in \Sigma$ is a visible event and $X \subseteq \Sigma$ is an interface set. The following equations hold for the application of diverse CCSP operators to transitions:

$$ {}^{s}({}^{o}a^{\,r}) = \begin{cases} {}^{o\cup s}a^{\,r} & s \cap (o \cup r) = \varnothing \\ stop & \text{otherwise} \end{cases} \tag{5.25} $$

$$ {}^{s*}({}^{o}a^{\,r}) = \begin{cases} {}^{o\cup s}a^{\,r\cup s} & a \in \Sigma_\tau, s \cap (o \cup r) = \varnothing \\ {}^{o\cup s}a^{\,r} & a \in \Sigma, s \cap (o \cup r) = \varnothing \\ stop & \text{otherwise} \end{cases} \tag{5.26} $$

$$ l : {}^{o}a^{\,r} = \begin{cases} {}^{o}(l.a)^{\,r} & a \in \Sigma \\ {}^{o}a^{\,r} & a \in \Sigma_\tau \end{cases} \tag{5.27} $$

$$ {}^{o}a^{\,r} \setminus X = \begin{cases} {}^{o}(\tau(a))^{\,r} & a \in X \\ {}^{o}a^{\,r} & \text{otherwise} \end{cases} \tag{5.28} $$

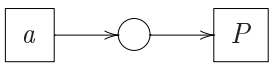
*

5.3. Reduction of Prefixing and Choice to Interleaving and P/T Net Causality

As described in Sect. 3.6, our calculus for CSP explores the behaviour of a process term essentially by transforming the term to a certain normal form recursively. In the form that is established, the operators of prefixing and external choice do no longer turn up, but are instead simulated through the causality concept that is based on monotonic P/T nets.

The pattern that is used for the elimination of prefixing is described in [BB01], where a theorem is formulated stating that sequential composition can generally be reduced to the explicit causality represented by the causal state operator, combined with operators for the production and consumption of tokens. The latter enforcement of causal relationships has the advantage of not requiring "spatial proximity" of the concerned subterms within a process term, contrary to prefixing, which depends on a rigid term structure that assembles the constituents according to the signature of the operator. With P/T net-based causality, it is possible to arrange the atomic parts (transitions) of the term loosely without omitting information, and to design rewriting rules that establish "flows" of transitions within the term under development by incrementally propagating parts towards the root.

The basic idea of the representation of CSP prefixing $a \to P$ through P/T nets is the schema



that guards the postponed "transition" P with a (fresh) place that is only marked once the primary transition a has fired. Provided that the place p does not occur free in P, the same net is given by the CCSP term

$$ a \to P = p[\mathrm{E}] : ({}^{\{p\}}a \;|||\; {}^{\{p\}}P), \qquad p \notin FP(P) $$

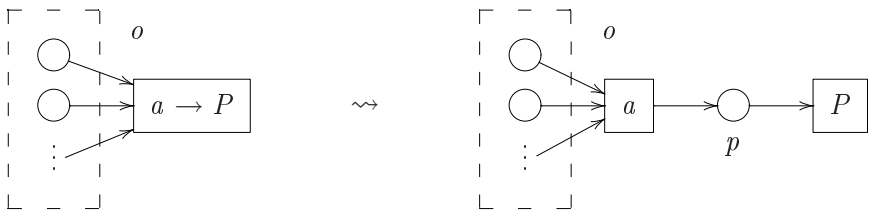
While the equation above is already sufficient to formulate a corresponding rule for a CCSP rewrite system, its efficiency can be increased by additionally taking care of leading token consumption, in order to diminish the number of places necessary in total. This shows up in the (common) situation

$$ {}^{o}(a \to P) \quad = \quad {}^{o}(p[\mathrm{E}] : (a\,{}^{\{p\}} \;|||\; {}^{\{p\}}P)), \qquad p \notin FP(P) $$

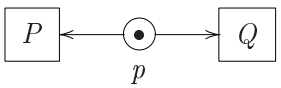
in which after the reduction of the prefixing operator it is necessary to somehow treat the consumption of tokens in combination with interleaving. The required—and unfortunately quite heavy—machinery therefore is described in Sect. 5.7, and involves the introduction of further places. By employing knowledge about the relation between $a\,{}^{\{p\}}$ and ${}^{\{p\}}P$, namely that the second process is never able to perform a step before the first one (the essence of sequentiality), and that the consumption of tokens is thus due to the first process, it is possible to derive a simpler representation of the same process:

$$ {}^{o}(a \to P) \quad = \quad p[\mathrm{E}] : ({}^{o}a\,{}^{\{p\}} \;|||\; {}^{\{p\}}P), \qquad p \notin FP(P), p \notin o $$

Schematically, this transformation can be illustrated when treating processes as transitions of P/T nets:

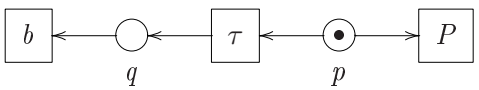


Like sequential composition, also the operator of external choice can be found to be definable in terms of P/T net causality. A first (naive[4]) approach to perform this translation is to relate a choice $P \,\Box\, Q$ with concurrent token consumption of two processes from the same place:



$$ p[\mathrm{M}] : ({}^{\{p\}}P \;|||\; {}^{\{p\}}Q), \qquad p \notin FP(P), p \notin FP(Q) $$

This simple approach does not account for the exceptional behaviour of CSP external choice w.r.t. leading internal steps of the subprocesses, however. Internal steps are by choice treated in the manner of interleaving, as they can be performed by the subprocesses P, Q without causing a definite decision to be made on which of the two processes is supposed to take over. For this (and only for this) reason, in CCSP there is a second starred operator of token consumption, which models the treatment of internal steps by choice using bidirectional arcs in the resulting P/T net. A concrete situation in which the usage of the unstarred consumption leads to wrong results is

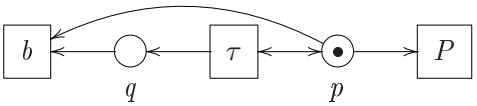


$$ \begin{aligned} &(a \to b \to \mathit{stop}) \setminus \{a\} \,\Box\, P \\ &\neq p[\mathrm{M}] : q[\mathrm{E}] : ({}^{\{q\}}b \;|||\; {}^{\{p\}}\tau(a)\,{}^{\{q\}} \;|||\; {}^{\{p\}}P) \end{aligned} $$

A P/T net that embodies the correct behaviour of the process is shown below, and is in the process algebra world given by a term that contains starred token consumption

[4] In fact the first implementation of the calculus described in this thesis contained an erroneous treatment of external choice.

instead of the unstarred one. The semantics of choice is captured correctly, as the firing of transition $\tau(a)$ does not consume the token of the place guarding the choice, but instead it is possible that step $\tau(a)$ is followed by a commitment to process P.



$$
\begin{aligned}
&(a \to b \to stop) \setminus \{a\} \,\Box\, P \\
&= p[\mathrm{M}] : ({}^{\{p\}*}((a \to b \to stop) \setminus \{a\}) \;|||\; {}^{\{p\}*}P) \\
&= p[\mathrm{M}] : q[\mathrm{E}] : ({}^{\{p,q\}}b \;|||\; {}^{\{p\}}\tau(a)\,{}^{\{p,q\}} \;|||\; {}^{\{p\}}P)
\end{aligned}
$$

Algebraic laws that enable the transformations described so far, and also some further ones, are collected by the next lemma. The first two items of the lemma are auxiliary propositions needed to prove the main parts.

5.3.1 Lemma (Reduction of Prefixing and Choice): Suppose that $P, Q \in CProc_{\Sigma_0,S}$ are processes, $p \in S$ is a place, $o, r \in \mathcal{P}_{\mathrm{fi}}S$ are finite sets of places and $a \in \Sigma$ is an event.

1. Provided that p does not turn up freely in P, the addition of p as a (marked) guard has no effect:

$$p \notin FP(P) \quad \Longrightarrow \quad p[\mathrm{M}] : {}^{\{p\}}P \quad = \quad p[\mathrm{M}] : {}^{\{p\}*}P \quad = \quad P \tag{5.29}$$

2. No tokens can be consumed from dead places:

$$p[\mathrm{D}] : {}^{\{p\}}P \quad = \quad p[\mathrm{D}] : {}^{\{p\}*}P \quad = \quad stop \tag{5.30}$$

3. Prefixing can be expressed in terms of P/T nets by the introduction of a mediating place (that does not already turn up in the concerned entities). All token consumption the whole process has to perform can be imposed on the first (external) step denoted with a. For $o \cap r = \varnothing$ and $p \notin FP(P) \cup o \cup r$:

$$ {}^{o*}({}^{r}(a \to P)) \quad = \quad p[\mathrm{E}] : ({}^{o \cup r}a\,{}^{\{p\}} \;|||\; {}^{\{p\}}P) \tag{5.31}$$

4. External choice can by rewritten to interleaving by guarding the subprocesses with a new place (and using the starred version of token consumption). Further token consumption can be distributed among the subprocesses immediately. For $p \notin FP(P) \cup FP(Q) \cup o$:

$$ {}^{o*}(P \,\Box\, Q) \quad = \quad p[\mathrm{M}] : ({}^{o \cup \{p\}*}P \;|||\; {}^{o \cup \{p\}*}Q) \tag{5.32}$$

*

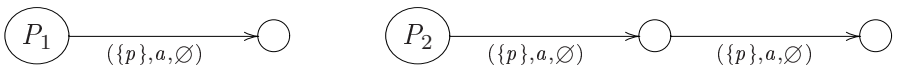
5.4. Reasoning about Closed Process Terms

The formulation of the most central transformation of CCSP expressions, which is the propagation of transitions through parallelism, requires a further prerequisite. When designing the necessary algebraic law, we meet the problem that the notion of (strong) bisimulation introduced in Sect. 2.2.1, which is used as the basis of all reasoning about CCSP so far, is a behavioural equivalence that is too accurate. This conflict shows up

whenever *open* processes are considered, which we define to be processes that contain free places. As a simple instance of the situation we consider the two terms

$$P_1 \quad := \quad {}^{\{p\}}a\,, \qquad P_2 \quad := \quad {}^{\{p\}}a \;|||\; {}^{\{p\}}a\,,$$

that both describe open processes containing the free place p. The processes are not bisimular, as the corresponding transition systems differ fundamentally:[5]



On the other hand, depicting the processes P_1, P_2 as P/T nets, as it is done in the last sections, leads to nets that are regarded as equivalent:

The different points of view towards multiplicity of transitions are caused by the fact that CCSP transitions do not have exactly the same meaning as P/T net transitions: The former ones are able to fire at most once, and then pass in the process *stop* (i.e. cannot be represented as self-loops in an LTS), which is a restriction that does not exist for (monotonic) P/T net transition. Transitions of nets are solely controlled by the marking of the net, and in principle it is possible that a transition fires more than once. Hence, the duplication of a transition in a P/T net does not alter the dynamic behaviour of the net.

Nevertheless, the representation as P/T nets captures the behaviour of the processes P_1, P_2 in a meaningful way, because for monotonic P/T nets additionally the semantics of places has to be taken into account. As the transitions of the two diagrams are all connected to place p, it is in any case (for any initial value of the place) not possible that a transitions fires repeatedly, or that both transitions of the right net fire. Namely, it is not possible to repeatedly consume tokens from a place in monotonic P/T nets; after a first consumption the value of the place is 'dead', and the place is unable to store any further tokens.

This observation holds likewise for processes, and while the processes P_1 and P_2 are different, we can as well observe that transitions of their terms have to consume tokens from place p to fire. Thus, also in this case the multiplicity of the transitions is irrelevant (to a certain degree).[6] Namely, for any value $v \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ we have

$$p[v] : P_1 \quad = \quad p[v] : P_2$$

This proposition can be generalised (strengthened) to arbitrary occurrences of P_1, P_2 in closed process terms, i.e. in terms in which no place turns up free: The value of a closed CCSP term is not altered by replacing an occurrence of P_1 by P_2.

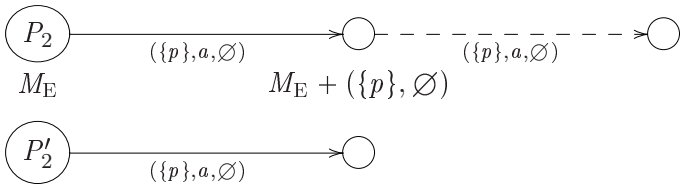
[5] In the LTS of P_2, the two intermediate states that result from different possible interleaving orders are bisimular, and are thus actually represented by the same element of a simple LTS (see Lem. 2.2.5 for the term simple). Hence, the diagram shows one possible linear LTS of the process; branching and non-simple LTSs, which for instance could have the expected diamond shape, exist as well.

[6] Generally, this holds for transitions ${}^{o}a^{r}$ with $o \ominus r \neq \varnothing$ because of Lem. 4.1.6, 5. Only for those CCSP terms whose transitions have this property a representation as P/T nets is meaningful.

The processes P_1, P_2 are prototypical for the situation that occurs in the next pages, when dealing with parallelism. The rule to treat the parallel composition operator is based on the possibility of duplicating parts of process terms without changing the meaning of the represented process. To justify the correctness of the transformations performed, we introduce a concept of equivalence of processes, which formalises the relation between P_1, P_2 from above. The concept is based on the notion of approximations of P/T net markings (discussed in Sect. 4.1.1). We show that the equivalence relation derived is essentially a congruence relation of the CCSP operators; this entails that modifications applied to a subprocess—without leaving the congruence class of the subprocess—also preserve the meaning of the whole process up to congruence. Because the equivalence relation coincides with identity on closed processes, the resulting notion of behavioural equivalence conforms to the informal idea of equivalence of P_1 and P_2.

The relation is defined as the kernel of a mapping on the set $\mathit{CProc}_{\Sigma_0,S}$ of processes: For each process $P \in \mathit{CProc}_{\Sigma_0,S}$ a "cleaned" process $M : P$ is constructed by removing certain traces that are inconsistent, and that cannot occur if the place semantics is enforced. Therefore, we choose an approximation M that is consistent with all possible (initial) place values for which the considered process is to be executed (an approximation that suits all situations is the minimum marking M_{E}, which is defined to assign all places the value 'empty': $M_{\mathrm{E}}(S) \subseteq \{\mathrm{E}\}$). All transitions outgoing from P that are necessarily disabled for M, i.e. all first execution steps labelled with (m, e, n), such that the pair (m, n) is disabled for all markings approximated by M, are removed. Adjacent processes—that can be reached by tracking the remaining transitions—are recursively simplified by the imposition of an approximation that is strengthened according to the label of the transition.

For the example process P_2 and the approximation M_{E}, this looks as follows. The first step $(\{p\}, a, \varnothing)$ of P_2 can be performed, because according to Lem. 4.1.6, 3 the pair $(\{p\}, \varnothing)$ is potentially enabled for M_{E}. On the other hand, to simplify the process that is reached by performing the step, M_{E} is strengthened to the approximation $M_{\mathrm{E}} + (\{p\}, \varnothing)$. Because of $(\{p\}, \varnothing) \notin \mathrm{PotEn}(M_{\mathrm{E}} + (\{p\}, \varnothing))$, this entails that the second transition is removed and does not turn up in $P_2' := M_{\mathrm{E}} : P_2$.



Two processes P, P' are regarded as equivalent (for an approximation M), if the simplified processes $M : P$, $M : P'$ are equal. For the running example, we can observe that in any case P_1 and P_2 are unified by the simplification, and are thus equivalent.

We primarily define the mapping ':' using a characteristic equation, as it is done in Table 4.4 for the CCSP operators. An equivalent representation using an SOS rule is given in Table 5.1. The existence of a (unique) solution of the characteristic equation can as always be proved using coinduction.

Table 5.1. SOS Rule for Application of Approximations to Processes

$$\frac{P \xrightarrow{(m,a,n)} P' \qquad (m,n) \in \mathrm{PotEn}(M)}{M : P \xrightarrow{(m,a,n)} (M + (m,n)) : P'}$$

5.4.1 Definition (Application of Approximations): The application of an approximation to a process is defined by the operator

$$\cdot : \cdot \begin{cases} \mathcal{M}_S \times \mathit{CProc}_{\Sigma_0,S} \to \mathit{CProc}_{\Sigma_0,S} \\ (M,P) \mapsto M : P \end{cases}$$

that for each marking $M \in \mathcal{M}_S$, each process $P \in \mathit{CProc}_{\Sigma_0,S}$ and each LTS label $(m,e,n) \in \mathcal{L}_{CCSP}$ satisfies the following characteristic equation (the notations of which are defined in Sect. 4.1)

$$(M : P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow \quad = \quad \begin{cases} (M + (m,n)) : P\Big\downarrow^{m}_{e}_{n}\Big\downarrow & (m,n) \in \mathrm{PotEn}(M) \\ \varnothing & \text{otherwise} \end{cases}$$

The kernel of $M : \cdot$ is denoted with $\sim_M$, i.e.

$$P \sim_M Q \quad :\Longleftrightarrow \quad M : P = M : Q \qquad *$$

5.4.2 Example: For the processes P_1, P_2 of the running example we have

$$\begin{aligned} &M_1 : P_1 = P_1 && M_1(p) \in \{\mathrm{E}, \mathrm{M}\} \\ &M_2 : P_1 = \mathit{stop} && M_2(p) = \mathrm{D} \\ &M_1 : P_2 = P_1 \\ &M_2 : P_2 = \mathit{stop} \end{aligned}$$

and thus $P_1 \sim_{M_1} P_2$ and $P_1 \sim_{M_2} P_2$. *

The application of approximations is comparable to the causal state operator, because in both cases information about the values of places is used to control the behaviour of a process. The main differences are:

- When applying approximations, information about *all* places of S is imposed, instead of only the value of a single place (as it is done by the state operator).
- In Def. 5.4.1 the enabledness of a transition is determined by the set $\mathrm{PotEn}(M)$ of *potentially enabled* transitions, whereas the causal state operator requires the value of a place to match the I/O of a transition exactly.
- The application of approximations does not conceal occurring places, contrary to the effect of the state operator.

Properties of approximations in combination with the CCSP operators are collected by the next lemma. We refer to the partial ordering on markings defined in Sect. 4.1.1 implicitly.

5.4.3 Lemma (Application of Approximations): Suppose that $M, \tilde{M} \in \mathcal{M}_S$ are markings of S, $P, Q, \tilde{P}, \tilde{Q} \in CProc_{\Sigma_0,S}$ are processes, $p \in S$ is a place, $o \in \mathcal{P}_{\mathrm{fi}} S$ is a finite set of places, $v \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ is a place value, $a \in \Sigma \cup \Sigma_\tau$ is an event, $l \in \Sigma$ is a visible event, $L, R \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP}$ are finite multisets of labels, $X \subseteq \Sigma$ is an interface set and $(o, a, r) \in \mathcal{L}_{CCSP}$ is a label.

1. $\sim_\cdot$ is a monotonic mapping of markings to sets, i.e.

$$M \leqslant \tilde{M} \quad \Longrightarrow \quad \sim_M \subseteq \sim_{\tilde{M}}$$

or equivalently

$$P \sim_M Q \wedge M \leqslant \tilde{M} \quad \Longrightarrow \quad P \sim_{\tilde{M}} Q \tag{5.33}$$

2. $M : \cdot$ is a projection:

$$M : M : P \quad = \quad M : P$$

An equivalent formulation, which follows directly from the definition of $\sim_M$, is

$$M : P \sim_M P \tag{5.34}$$

3. Applying approximations to closed processes has no effect:

$$FP(P) = \varnothing \quad \Longrightarrow \quad M : P = P \tag{5.35}$$

4. $\sim_M$ is a congruence relation w.r.t. the CCSP operators:

$$P \sim_M Q \quad \Longrightarrow \quad a \to P \;\sim_M\; a \to Q \tag{5.36}$$
$$P \sim_M Q \quad \Longrightarrow \quad {}^{o}P \;\sim_M\; {}^{o}Q \tag{5.37}$$
$$P \sim_M Q \quad \Longrightarrow \quad {}^{o*}P \;\sim_M\; {}^{o*}Q \tag{5.38}$$
$$P \sim_M \tilde{P} \wedge Q \sim_M \tilde{Q} \quad \Longrightarrow \quad P \,\Box\, Q \;\sim_M\; \tilde{P} \,\Box\, \tilde{Q} \tag{5.39}$$
$$P \sim_M \tilde{P} \wedge Q \sim_M \tilde{Q} \quad \Longrightarrow \quad P \,|[\, L[X]R \,]|\, Q \;\sim_M\; \tilde{P} \,|[\, L[X]R \,]|\, \tilde{Q} \tag{5.40}$$
$$P \sim_M Q \quad \Longrightarrow \quad l : P \;\sim_M\; l : Q \tag{5.41}$$
$$P \sim_M Q \quad \Longrightarrow \quad P \setminus X \;\sim_M\; Q \setminus X \tag{5.42}$$

For the causal state operator it has to be ensured that the used approximation is not too strong, i.e. that P and Q do not differ too much. According to the introduction, chosen approximations M have to cover all initial place values for which a process is supposed to be executed. Choosing M with $M(p) > v$ violates this requirement, because M does not approximate markings M' with $M'(p) = v < M(p)$.

$$\begin{array}{r} M(p) = v \wedge P \sim_M Q \\ \wedge M|_{S\setminus\{p\}} = \tilde{M}|_{S\setminus\{p\}} \end{array} \quad \Longrightarrow \quad p[v] : P \sim_{\tilde{M}} p[v] : Q \tag{5.43}$$

5. A relation similar to Impl. (5.40) can be derived for the buffers of the operator of parallelism:

$$(o,a) \notin \mathrm{PotEn}(M) \quad \Longrightarrow \quad P \,|[L \uplus \{(o,a,r)\}[X]R]|\, Q \;\sim_M\; P \,|[L[X]R]|\, Q \tag{5.44}$$

6. Transitions are judged by the potential enabledness of their I/O:

$$M : {}^{o}a^{\,r} \quad = \quad \begin{cases} {}^{o}a^{\,r} & (o,r) \in \mathrm{PotEn}(M) \\ \mathit{stop} & \text{otherwise} \end{cases} \tag{5.45}$$

*

5.4.4 Example: The congruence properties of the last lemma allow simple modifications of CCSP process terms: (P_1, P_2, M_E as in the whole section)

$$\begin{aligned} P_1 \sim_{M_\mathrm{E}} P_2 \quad &\Longrightarrow \quad (P_1 \,\Box\, Q) \sim_{M_\mathrm{E}} (P_2 \,\Box\, Q) \\ &\Longrightarrow \quad a \to (P_1 \,\Box\, Q) \sim_{M_\mathrm{E}} a \to (P_2 \,\Box\, Q) \end{aligned}$$

Subterms of closed process terms can be modified without altering the represented process. Besides that, by Eq. (5.43) it is possible to exploit the exact value of places that is expressed by state operators: For instance for the marking

$$M'(q) \quad = \quad \begin{cases} \mathrm{D} & q = p \\ \mathrm{E} & q \neq p \end{cases}$$

we have

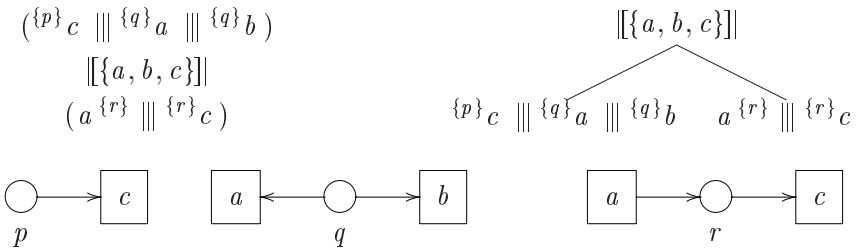
$$\begin{aligned} p[\mathrm{D}] : \left({}^{\{p\}}a \;|||\; {}^{\{p\}}a \right) &\overset{(5.35)}{=} M_\mathrm{E} : p[\mathrm{D}] : \left({}^{\{p\}}a \;|||\; {}^{\{p\}}a \right) \\ &\overset{(5.43)}{=} M_\mathrm{E} : p[\mathrm{D}] : M' : \left({}^{\{p\}}a \;|||\; {}^{\{p\}}a \right) \\ &\overset{(5.40)}{=} M_\mathrm{E} : p[\mathrm{D}] : M' : \left(M' : {}^{\{p\}}a \;|||\; M' : {}^{\{p\}}a \right) \\ &\overset{(5.45)}{=} M_\mathrm{E} : p[\mathrm{D}] : M' : (\mathit{stop} \;|||\; \mathit{stop}) \\ &= \quad \cdots \quad = \quad p[\mathrm{D}] : \mathit{stop} \end{aligned}$$

*

5.5. Transition Propagation through Parallelism

The rewriting system for CCSP process terms, which is introduced in Sect. 6, normalises terms by pulling out transitions, i.e. by moving expressions of the kind ${}^{o}a^{\,r}$ towards the root of the term. Therefore, the rewriting system has to provide rules to treat the operators that make up the skeleton of the term: If a transition turns up immediately below such an operator, there need to be rules to commute the two entities and lift the transition one step further towards the root. This propagation follows the diagrams given in Sect. 3.5 for event propagation performed by the SOS rules of CSP, with the only difference that the order in which transitions are propagated is irrelevant. Different orderings of the reduction lead to equivalent processes, and migrating one transition to the root before another one does not imply that there is any causal relationship

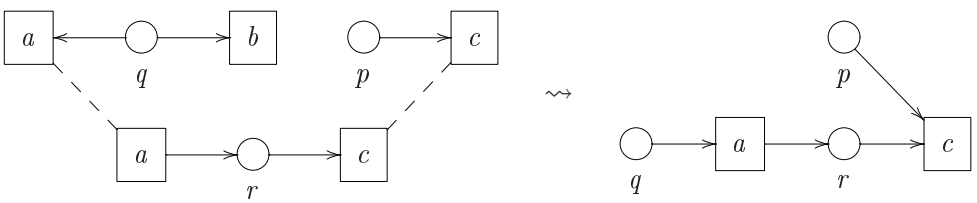
Table 5.2. Example of Propagation of Transitions through Parallelism



between them. Existing interdependencies are preserved by expressing them through the P/T net-like notion of causality available in CCSP.

The essential CCSP operator from which the skeleton is built is the operator of parallelism, because it is the only binary operator that remains after removing the derived external choice as described in Sect. 5.3. To motivate the algebraic identity the rewriting rule for parallelism is based on, in the whole section we stick to the example that is given in Table 5.2, and that consists of two lists of transitions connected by parallelism. In the term, the interleaving operator turns up a number of times besides a single occurrence of the full-featured parallelism operator. Despite the fact that interleaving is only a special case of general parallelism, it is not necessary to pay a lot of attention to this connective, as it does not introduce any dependencies between its operands. We use interleaving mostly as a way to represent bags of processes or transitions, which is possible because of the properties of interleaving that are given in Lem. 5.2.1 (in particular commutativity and associativity).

Both transition lists of Table 5.2 can separately be displayed as monotonic P/T nets, which do not describe the behaviour of the complete system when standing alone, however. Therefore, it is necessary to consider the combination of the two nets by parallelism. We describe a procedure to merge the two nets into one, which works by identifying matching transitions. The approach is based on the observation that the parallel composition operator requires synchronising events to be raised by the argument subprocesses simultaneously; if the subprocesses are described through lists of transitions, this means that such events are created by the simultaneous firing of two transitions. A net that resembles the combined behaviour of the two parallel P/T nets can thus be constructed by relating transitions with the same events:



To describe the procedure in an algebraic way, we assume that two bags of transitions

$$P = {}^{o_1}a_1{}^{r_1} \;|||\; \cdots \;|||\; {}^{o_k}a_k{}^{r_k} \qquad Q = {}^{m_1}e_1{}^{n_1} \;|||\; \cdots \;|||\; {}^{m_l}e_l{}^{n_l}$$

are given that turn up as arguments of the parallel composition operator $|[X]|$. To merge the represented processes/nets, first of all we can simply passed through those transitions ${}^{o}a^{r}$ of both lists whose events are interleaved, i.e. $a \notin X$. This is possible because of the identity

$$P \,|[\, L[X]R \,]|\, \Big({}^{o}a^{r} \;|||\; Q \Big) \quad = \quad {}^{o}a^{r} \;|||\; \Big(P \,|[\, L[X]R \,]|\, Q \Big) \qquad \text{for } a \notin X$$

Hence, we can assume that all events of the transitions of P, Q need to by synchronised. The recipe to perform this is: Determine all pairs $\left({}^{o_i}a_i{}^{r_i},\ {}^{m_j}e_j{}^{n_j} \right)$ of transitions that are compatible, merge the transitions of each pair into a single transition, and finally connect all resulting transitions by interleaving.

Two terms of this procedure need to be explained:

- We regard two transitions ${}^{o_i}a_i{}^{r_i}$, ${}^{m_j}e_j{}^{n_j}$ as compatible, if the events a_i, e_j are equal, $a_i = e_j$, and if the two I/O pairs (o_i, r_i), (m_j, n_j) are independent:

  $$(o_i \cup r_i) \cap (m_j \cup n_j) = \varnothing$$

  The notion of compatible transitions conforms to the operational rules for the operator of parallelism, in which the given criteria are precisely the conditions for the synchronisation of two transition labels (o_i, a_i, r_i), (m_j, e_j, n_j).

- Two compatible transitions are merged by joining their I/O sets, i.e. the result of merging ${}^{o_i}a_i{}^{r_i}$, ${}^{m_j}e_j{}^{n_j}$ is the transition

  $${}^{o_i \cup m_j}a_i{}^{r_i \cup n_j}$$

  This combination of I/O pairs can also be described by using addition of such pairs (Def. 4.1.3), which leads exactly to the same result.

For the bags of transitions given in Table 5.2, there are only two pairs of compatible transitions (labelled with a and c), and the result of performing the merging procedure corresponds to the P/T net given above:

$${}^{\{q\}}a^{\{r\}} \;|||\; {}^{\{p,r\}}c$$

The first version of the merging procedure assumes that the two bags P, Q of transitions are completely known a priori, but we can also give an online variant, which works on streams of transitions (Table 5.3). The algorithm successively reads transitions from the two input streams, and—depending on whether a transition represents an event that has to be synchronised—it either writes the transition to the output stream directly, or it merges the transition with all (compatible) transitions that have been read from the opposing stream so far, and writes the resulting transitions to the output stream. The given pseudo-code contains indeterministic decisions, in particular it could be possible to read transitions from both input streams. This does only affect the order in which the output stream receives resulting transitions, however, and—because interleaving (with which the transitions are eventually interconnected) is a commutative and associative operation—does not cause problems.

Table 5.3. Online Algorithm for Merging Two Streams of Transitions

```
MERGE-TRANSITION-STREAMS(I_1, I_2, R, X)
                                   ▷ I_1, I_2 are input streams of transitions
                                   ▷ R is output stream for results
                                   ▷ X is the interface event set
 1  B_1 ← empty bag                ▷ Buffer of transitions for I_1
 2  B_2 ← empty bag                ▷ Buffer of transitions for I_2
 3  while I_i provides further transition (i ← 1 or i ← 2)
 4        do T ← read(I_i)
 5           if T needs to be synchronised w.r.t. X
 6              then B_i ← add(B_i, T)
 7                   for T' in B_{3−i}
 8                        do if T and T' are compatible
 9                                then write(R, merge(T, T'))
10              else write(R, T)
```

The algorithm of Table 5.3 is implemented in Sect. 6 as a rewriting system that transforms process terms containing the parallel composition operator. The two buffers used by the procedure are represented by the buffer arguments L, R of a term $P \,\|[\, L[X]R \,]\|\, Q$, and the processes P, Q can be regarded as streams from which transitions are read.

The rewriting of the process term of the running example by the system is schematically shown in Table 5.4, in which each modification step corresponds to one iteration of the loop of the merging algorithm. The process term only contains synchronised transitions, so in each step one transition ${}^{m}a^{n}$ of the subprocesses is propagated by migrating it as a triple (m, a, n) to the corresponding buffer of the parallel operator occurrence, and by merging it with compatible elements of the opposing buffer.

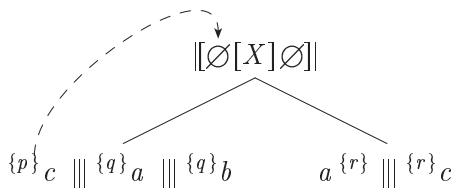
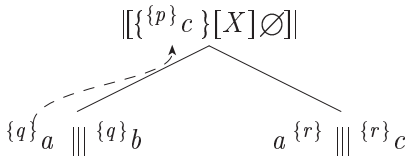
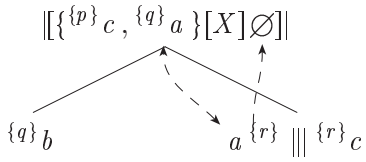
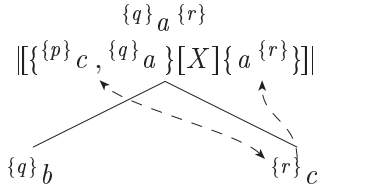
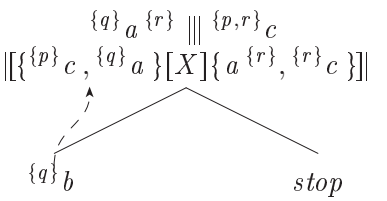
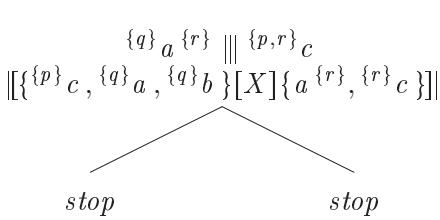
The effect of the term reduction is a bag of transitions (interconnected by interleaving) on the same level as the parallel operator, which accumulates interleaved and merged transitions of the subprocesses. The pattern of buffering and merging passing transitions ensures that each pair of synchronised transitions originating from different subprocesses is checked exactly once for compatibility.

The correctness of the adaption of the merging algorithm as rewriting rules is justified by the lemma below. Therefore, we need a further technical concept, because actually the algorithm makes an assumption about concerned transitions that does not hold in general: The merging of different pairs of transitions can lead to an effect that can be regarded as duplication of transitions. For instance rewriting of the term

$$ {}^{\{p\}}a\,{}^{\{p\}} \,\|[\,\{a\}\,]\|\, \left({}^{\{r\}}a \;|||\; {}^{\{s\}}a \right) $$

leads to the transition list ${}^{\{p,r\}}a\,{}^{\{p\}} \;|||\; {}^{\{p,s\}}a\,{}^{\{p\}}$, which does *not* represent an equivalent process: Contrary to the original process, the new one is able to perform two subsequent execution steps with event a. The problem is caused by the duplication of

Table 5.4. Example of Propagation of Transitions through Parallelism: Rewriting

$({}^{\{p\}}c \;|||\; {}^{\{q\}}a \;|||\; {}^{\{q\}}b\,)$ $\|[X]\|$ $(\,a\,{}^{\{r\}} \;|||\; {}^{\{r\}}c\,)$	
$({}^{\{q\}}a \;|||\; {}^{\{q\}}b\,)$ $\|[\{{}^{\{p\}}c\,\}[X]\varnothing]\|$ $(\,a\,{}^{\{r\}} \;|||\; {}^{\{r\}}c\,)$	
${}^{\{q\}}b$ $\|[\{{}^{\{p\}}c\,,{}^{\{q\}}a\,\}[X]\varnothing]\|$ $(\,a\,{}^{\{r\}} \;|||\; {}^{\{r\}}c\,)$	
${}^{\{q\}}a\,{}^{\{r\}} \;|||\; \left(\begin{array}{c} {}^{\{q\}}b \\ \|[\{{}^{\{p\}}c\,,{}^{\{q\}}a\,\}[X]\{\,a\,{}^{\{r\}}\}]\| \\ {}^{\{r\}}c \end{array}\right)$	
${}^{\{q\}}a\,{}^{\{r\}} \;|||\; {}^{\{p,r\}}c \;|||$ $\left(\begin{array}{c} {}^{\{q\}}b \\ \|[\{{}^{\{p\}}c\,,{}^{\{q\}}a\,\}[X]\{\,a\,{}^{\{r\}},{}^{\{r\}}c\,\}]\| \\ stop \end{array}\right)$	
${}^{\{q\}}a\,{}^{\{r\}} \;|||\; {}^{\{p,r\}}c \;|||$ $\left(\begin{array}{c} stop \\ \|[\{{}^{\{p\}}c\,,{}^{\{q\}}a\,,{}^{\{q\}}b\,\}[X]\{\,a\,{}^{\{r\}},{}^{\{r\}}c\,\}]\| \\ stop \end{array}\right)$	

Triples (m, a, n) within buffers of the parallel operator are written as transitions ${}^{m}a\,{}^{n}$ to make the terms shorter. X has the value $\{a, b, c\}$.

transition ${}^{\{p\}}a^{\{p\}}$ (as part of the two resulting transitions), which is in this case not a correct transformation, as firing of the transition does neither consume nor produce tokens (and thus does not alter the marking of any places). We can avoid these inconsistencies by demanding that only transitions ${}^{o}a^{r}$ with $o \ominus r \neq \varnothing$ are merged, which is not a significant restriction (see Sect. 6.4).

5.5.1 Definition (Transition Merging): 1. Suppose that $L \in \mathrm{mult_{fi}}\,\mathcal{L}_{CCSP}$ is a finite multiset of labels and $(o,a,r) \in \mathcal{L}_{CCSP}$ is a label. The *merge* of (o,a,r) with L is the process[7]

$$mrg\big((o,a,r),L\big) \quad := \quad \Big|\!\Big|\!\Big|_{\substack{(o',a,r')\in L \\ (o'',r'')=(o,r)+(o',r')}} {}^{o''}a^{r''}$$

2. A triple (o,a,r) or a transition ${}^{o}a^{r}$ is called *duplicable*, if $o \ominus r \neq \varnothing$. ∗

The duplicability property ensures that a transition can be cloned without modifying the behaviour of a process, up to a congruence $\sim_M$ as defined in the last section: By Lem. 4.1.6, 6, the condition $o \ominus r \neq \varnothing$ entails $(o,r) \notin \mathrm{PotEn}(M + (o,r))$, which means that a process $M : P$ can never raise a triple (o,e,r) twice, and in particular ${}^{o}e^{r} \sim_M {}^{o}e^{r} \,|||\, {}^{o}e^{r}$.

5.5.2 Lemma (Propagation through Parallelism): Suppose P, $O \in CProc_{\Sigma_0,S}$ are processes, $M \in \mathcal{M}_S$ is a marking, $L, R \in \mathrm{mult_{fi}}\,\mathcal{L}_{CCSP}$ are finite multisets of labels, $X \subseteq \Sigma$ is an interface set and $(o,a,r) \in \mathcal{L}_{CCSP}$ is a label.

Provided that (o,a,r) and each element of L is duplicable, the following relation holds:

$$P \,[\![\, L[X]R \,]\!]\, ({}^{o}a^{r} \,|||\, Q) \sim_M \begin{cases} {}^{o}a^{r} \,|||\, (P \,[\![\, L[X]R \,]\!]\, Q) & a \notin X \\ mrg\big((o,a,r),L\big) \,|||\, (P \,[\![\, L[X]R \uplus \{(o,a,r)\} \,]\!]\, Q) & a \in X \end{cases} \tag{5.46}$$

∗

5.6. Transition Propagation through Labelling and Hiding

Constituents of the skeleton that can be treated with little effort are the CSP operators for labelling and for hiding. These operators distribute through interleaving

$$l : (P \,|||\, Q) = (l : P) \,|||\, (l : Q), \qquad (P \,|||\, Q) \setminus X = (P \setminus X) \,|||\, (Q \setminus X)$$

and hence transitions can be pulled out of the scope of the operators merely by adjusting their events accordingly (also employing Lem. 5.2.4). Likewise, for labelling distribution through general settings of parallelism could be performed by altering the interface set and the buffers of the parallel operator appropriately:

$$l : (P \,[\![\, L[X]R \,]\!]\, Q) = (l : P) \,[\![\, L'[l : X]R' \,]\!]\, (l : Q)$$

This observation is not formalised in the report, however. Hiding cannot be treated in a similar way in general, because the interface set X is by definition not allowed to contain invisible events.

[7] Because interleaving $|||$ is an associative and commutative operator, we can for finite multisets A use the recursive definition $|||_{x\in A\uplus\{a\}} P(x) := P(a) \,|||\, |||_{x\in A} P(x)$ for "interleaving comprehension".

5.6.1 Lemma (Propagation through Labelling and Hiding): Suppose that $P, Q \in CProc_{\Sigma_0,S}$ are processes, $l \in \Sigma$ is a visible event and $X \subseteq \Sigma$ is a set of events. The following equations hold:

$$l : (P \,|||\, Q) \quad = \quad (l : P) \,|||\, (l : Q) \tag{5.47}$$

$$(P \,|||\, Q) \setminus X \quad = \quad (P \setminus X) \,|||\, (Q \setminus X) \tag{5.48}$$

*

5.7. Transition Propagation through Token Consumption

The kind of propagation still missing is propagation through the operators for token consumption. Corresponding to the algebraic identities described in the last two sections, the situations in which appropriate laws for consumption are necessary are

$$^{o}(P \,|||\, Q), \qquad ^{o*}(P \,|||\, Q),$$

but the problem can as well be generalised to distribution of the consumption operators through the full-featured operator of parallelism (and not only through interleaving).

The left term expresses that the first step performed by $P \,|||\, Q$, i.e. either by P or by Q, has to consume tokens from the places o. Analogously, in the second process, leading internal steps of P and Q have to test the values of the places o (i.e. such steps can only be performed if each element of o contains a token), until finally one of the processes performs a visible step and consumes tokens from o. Finding equivalent terms in which the consumption is in some way distributed among P and Q, that means terms that have the shape $P' \,|||\, Q'$, is in this case more difficult than for the operators of the last sections, since the transformation must not introduce new interdependencies between P and Q. For this reason the naive approach

$$^{o}(P \,|||\, Q) \quad \rightsquigarrow \quad ^{o}P \,|||\, ^{o}Q$$

does not preserve the meaning of the term, as in the right term the subprocess performing the first step will inevitably block the other process (unless o is empty). A modification—like the one shown—that duplicates the consumption of tokens has to be compensated by somehow adjusting the capacity of concerned places and the number of circulating tokens. The next pages describe an approach that performs such an adjustment by a suitable transformation of processes (and of process terms). To motivate the method, we first carry out the construction for the "special case" of monotonic P/T nets.

5.7.1. Duplication of Places in Monotonic P/T Nets

For 1-safe P/T nets, the capacity of a single place or the weight of an arc (the number of tokens that firing of the connected transition produces or consumes) cannot be altered, but to achieve a similar effect it is possible to duplicate a place, together with all edges connected to it. Such a modification has a "global" character and potentially changes the complete P/T net, because all connections between the particular place and transitions have to be tracked and duplicated. Still, by using this method (as a

Figure 5.1. Example of Splitting Places and Transitions in P/T Nets

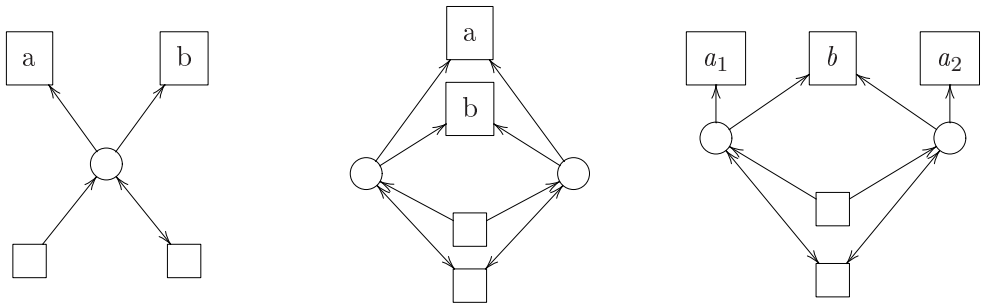
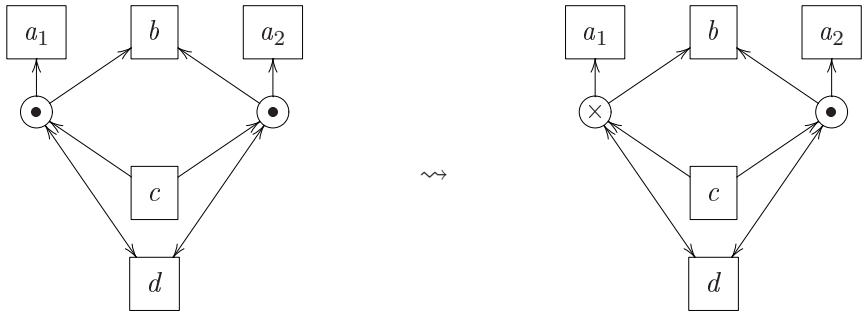


Figure 5.2. Example of "Inclusive Or" Consumption of Tokens in P/T Nets



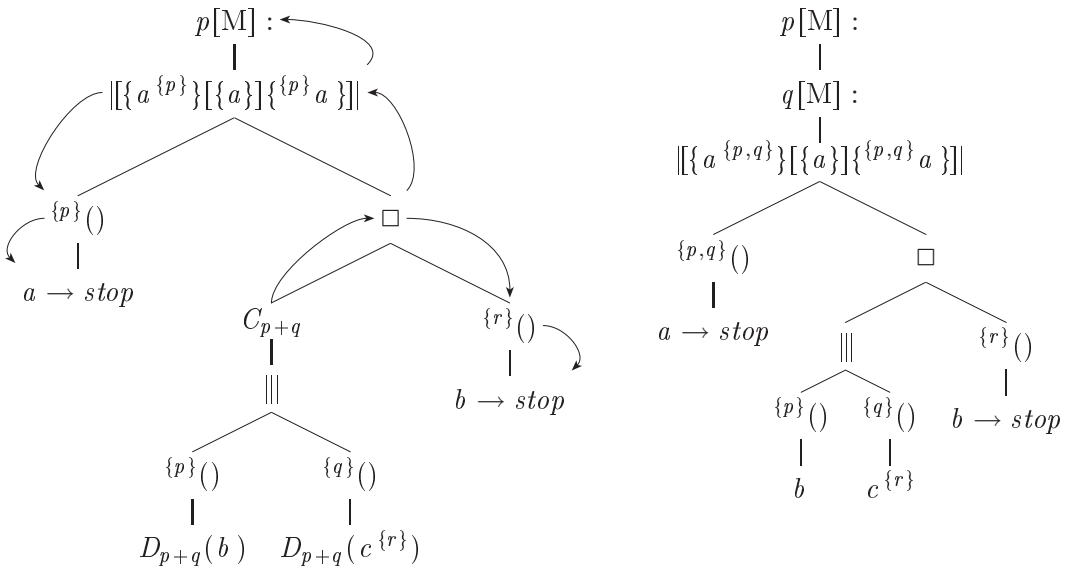
first step) it is possible to transform an arbitrary P/T net into a larger net that has exactly the same behaviour, and that contains a particular place "twice", i.e. there is a pair p, q of distinct places that have the same I/O. An illustration thereof is given in Fig. 5.1, where the diagram in the middle is obtained by duplicating the place of the left net. If the modified net is executed with an initial marking that assigns both places the same value, then the values of the places will not diverge on any execution trace of the net either, because firing of transitions has the same effect to both places. If the places are given different values, on the other hand, connected transitions will necessarily be disabled, because neither production to nor consumption from the places is possible in this case.[8]

The redundancy of the middle net in Fig. 5.1 can be employed to alter the behaviour of the net, which is performed in the right net, and which eventually leads to a correct method for distributing token consumption in process terms. In this net, additionally transition a—which in the left diagram is connected to the concerned place—is split in two, and each of the copies a_1, a_2 is connected to one of the cloned places. In the resulting net the pair a_1, a_2 of transitions *together* competes against the third transition b for tokens: Considering the transitions in the top row in the right diagram, firing of a_1 or of a_2 blocks transition b, whereas the firing of b disables both a_1 and a_2.

More generally, we can observe that in monotonic P/T nets the result of splitting both a place and one consuming transition a is a net that realises a kind of "inclusive

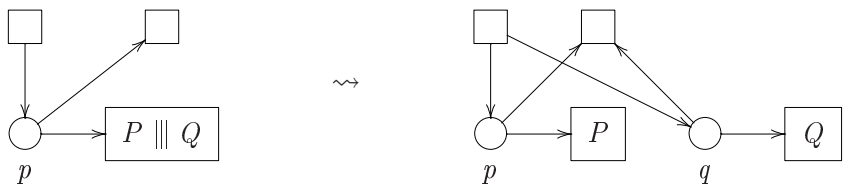
[8] This also holds for traditional 1-safe P/T nets whose place semantics is not monotonic.

Figure 5.3. Example of Duplication of Places in a CCSP Process Term



or" consumption:[9] For the remainder of the net, the firing of a_1 *or* a_2 has an effect equivalent to the firing of a in the original net, namely all transitions that are connected to both of the split places—regardless of the kind of the connection—are disabled at once. At the same time, the consumption does not prohibit the successive firing of a_1 *and* a_2. This is illustrated in the diagrams of Fig. 5.2, of which the right one shows the state of the net after the firing of transition a_1, which disables all transitions apart from a_2.

Duplication of places is a way to distribute token consumption through interleaving, because disjunctive consumption of tokens is precisely the behaviour desired in this situation. A solution for token distribution is thus shown schematically in the following diagrams, where place q of the right one is obtained by cloning place p. In the left P/T net the first execution step—performed by either P or Q—consumes a token from p, whereas in the right net a token is consumed from p or from q. For the remainder of the net, the two situations are indistinguishable.



5.7.2. Example of Distribution of Token Consumption

Before we define the necessary operators in detail, we show an example of distributing token consumption in a term. The procedure carries over the idea of duplicating places

[9] The same construction can be used to achieve an "and" behaviour for transitions producing tokens, which is also a standard pattern for designing concurrent systems using general 1-safe P/T nets.

to process terms. The term that is treated in the example is

$$p[\mathrm{M}:\Big(\Big({}^{\{p\}}\big(a \to stop\big)\Big) \,\|[\,\{a\,{}^{\{p\}}\}[\{a\}]\{{}^{\{p\}}a\,\}\,]\| \,\Big(\underline{{}^{\{p\}}\big(\,b \;|||\; c\,{}^{\{r\}}\big)}\,\Box\,{}^{\{r\}}\big(b \to stop\big)\Big)\Big)$$

in which at the underlined position the token consumption is supposed to be distributed. The first step of the transformation is the actual distribution of the consumption; this introduces three operators describing the necessary duplication of place p in the whole term. The modification preserves the meaning of the term up to the congruence relation $\sim_M$ (as introduced in Sect. 5.4; $M_{\mathrm{E}} \in \mathcal{M}_S$ is the marking that assigns all places the value E).

$${}^{\{p\}}(b \;|||\; c\,{}^{\{r\}}) \quad \sim_{M_{\mathrm{E}}} \quad C_{p+q}\Big({}^{\{p\}}D_{p+q}(\,b\,) \;|||\; {}^{\{q\}}D_{p+q}(\,c\,{}^{\{r\}})\Big)$$

The indices $p+q$ in the right term specify the intention to introduce place q as a copy of p (q does not turn up in the original term). D_{p+q} is a mapping that alters processes by duplicating places, whereas C_{p+q} is the (left) inverse of D_{p+q}, and is responsible for identifying the places p and q. The complete situation after the first step is shown in the left tree of Fig. 5.3.

The second step of the distribution consists of the duplication of place p. This is performed by applying the operators D_{p+q}, C_{p+q}, whose effect is related to the CCSP operators in the next pages. By using suitable algebraic laws, the symbol D_{p+q} can be moved downwards to the leaves of the term. Occurrences of place p within arguments of CCSP operators passing by are adjoined with the "clone" place q, so in the end in a whole subterm the single place p is replaced with the pair p, q. In the tree representation in Fig. 5.3, the progression of D_{p+q} is represented by the arrows pointing downwards; the application to the subterms b and $c\,{}^{\{r\}}$ has no effect, since p does not turn up in the terms.

In a similar way, the operator C_{p+q} can be moved upwards to the root of the term (which is shown by arrows pointing upwards in Fig. 5.3). In this action, again occurrences of p are duplicated by adding q, and side branches of the term met are treated by applying D_{p+q} to them. The result of the duplication is shown in the right tree of Fig. 5.3, and differs from the original term in the additional occurrences of q. The meaning of the original term is by the whole transformation preserved up to the relation $\sim_{M_{\mathrm{E}}}$.

$$p[\mathrm{M}:q[\mathrm{M}:$$
$$\Big(\Big({}^{\{p,q\}}\big(a \to stop\big)\Big) \,\|[\,\{a\,{}^{\{p,q\}}\}[\{a\}]\{{}^{\{p,q\}}a\,\}\,]\| \,\Big(\big({}^{\{p\}}b \;|||\; {}^{\{q\}}c\,{}^{\{r\}}\big)\,\Box\,{}^{\{r\}}\big(b \to stop\big)\Big)\Big)$$

5.7.3. Duplication of Places in Processes

In the CCSP calculus, we use the scheme of "inclusive or" consumption in P/T nets to realise the distribution of token consumption through parallelism. We therefore give an analogue of the duplication of places that operates on processes in general. The duplication is primarily formulated as a semantic process transformation, and is complemented with algebraic identities that can be used to perform the duplication by rewriting terms. Syntactically, the result is a term in which any occurrence of the

Table 5.5. SOS Rules for Duplicating and Merging Places of Processes

$$\frac{P \xrightarrow{(m,a,n)} P' \qquad m' = A_{p+q}(m),\ n' = A_{p+q}(n)}{D_{p+q}(P) \xrightarrow{(m',a,n')} D_{p+q}(P')}$$

$$\frac{P \xrightarrow{(m,a,n)} P' \qquad (m',n') \lhd_{p+q} (m,n)}{C_{p+q}(P) \xrightarrow{(m',a,n')} C_{p+q}(Cl_{p+q}(m,n,P'))}$$

place to be duplicated is complemented with the clone place, i.e. in arguments of the token consumption operators and of transitions, in the buffers of the CCSP operator of parallelism, and as argument of the causal state operator.

5.7.1 Example: Below, for some processes (described by process terms) the results of duplicating place p by adding place q are given. For the last process term no modification is necessary, because p is bound by the causal state operator and does not turn up free.

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}} {}^{\{p\}}a & \rightsquigarrow & {}^{\{p,q\}}a & a^{\{p\}} & \rightsquigarrow & a^{\{p,q\}} \\ {}^{\{p,r\}}(a \to stop) & \rightsquigarrow & {}^{\{p,q,r\}}(a \to stop) & p[\mathrm{M}] : {}^{\{p\}}a & \rightsquigarrow & p[\mathrm{M}] : {}^{\{p\}}a \qquad * \end{array}$$

We define two mutually inverse operators D_{p+q} and C_{p+q}, which describe duplication and identification of places of a process in a semantic way. Both functions only treat the special case of creating *one* copy of a single place, and the merging of a pair of places into a single place, but by an iterated application arbitrary (finite) numbers of duplicates can be generated. We can use both mappings to formulate the law for distributing token consumptions, as shown in the example above. In the sequel, we always denote the "original" place that is to be duplicated with p, and the copy of the place with q. $p, q \in S$ are supposed to be distinct.

Duplicating Places: For the definition of the first operator, we need an auxiliary function that complements a given (finite) set of places by ensuring that the set either contains both places p and q or none of them. The function is parametrised with the pair (p,q), which we denote by adding $p+q$ as an index; the same notation is also used for all other parametrised tools we introduce in this section:

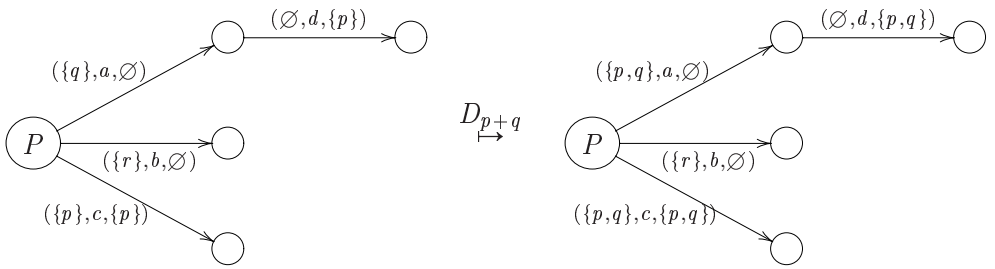
$$A_{p+q}(m) \quad := \quad \begin{cases} m \cup \{p,q\} & m \cap \{p,q\} \neq \varnothing \\ m & \text{otherwise} \end{cases} \qquad m \in \mathcal{P}_{\mathrm{fi}} S,\ \ p,q \in S$$

The operator D_{p+q} is defined to be the (uniquely determined) mapping on $CProc_{\Sigma_0,S}$ that for each process $P \in CProc_{\Sigma_0,S}$ and for each LTS label $(m,e,n) \in \mathcal{L}_{CCSP}$ satisfies the equation

$$(D_{p+q}(P))\Big\downarrow \begin{matrix} m \\ e \\ n \end{matrix} \Big\downarrow \quad = \quad \bigcup_{\substack{m',n' \in \mathcal{P}_{\mathrm{fi}} S \\ m = A_{p+q}(m') \\ n = A_{p+q}(n')}} D_{p+q}(P\Big\downarrow \begin{matrix} m' \\ e \\ n' \end{matrix} \Big\downarrow)$$

The existence of a unique solution to this family of equations can be shown by using coinduction. An equivalent characterisation of the operator by an SOS rule is given in Table 5.5. D_{p+q} maps every process P to a process that behaves like P, apart from the fact that whenever P is able to perform a step that produces or consumes tokens from any of the places p, q, the process $D_{p+q}(P)$ offers a corresponding step that produces or consumes tokens from *both* p and q. Compared with monotonic P/T nets, this transformation can already be regarded as the splitting of a place p by adding an equivalent place q. In fact q is not *added* as a new place, however, because the reservoir S of existing places is not altered. While by convention we regard the place q as the "new" one, and p as the place already existing, D_{p+q} is defined in a symmetric way and can as well be applied to processes that contain q as a free place ($q \in FP(P)$). A better description in terms of P/T nets is thus: For all transitions that are connected with p, corresponding connections with q are added to the net, and vice versa. For a net that does not refer to q, i.e. in which no transition has q as part of its I/O, this notion coincides with the splitting of p into p, q.

5.7.2 Example: We apply D_{p+q} to the process described by the following labelled transition system:



*

The application of D_{p+q} to process terms can be described by a rewriting system, the laws for which are given in the next lemma. To illustrate this method, we first show how D_{p+q} can be applied in an example; in each term the part that is modified in the next step is underlined. Each step commutes the operator D_{p+q} with one of the CCSP operators.

$$
\begin{aligned}
\underline{D_{p+q}\Big(\, a^{\,\{p\}} \;|||^{\,\{p\}} \big(\, {}^{\{r\}}b \;|||^{\,\{s\}} c \,\big)\Big)} &= \underline{D_{p+q}\Big(\, a^{\,\{p\}}\Big)} \;|||\; D_{p+q}\Big(\, {}^{\{p\}}\big(\, {}^{\{r\}}b \;|||^{\,\{s\}} c \,\big)\Big) \\
&= a^{\,\{p,q\}} \;|||\; \underline{D_{p+q}\Big(\, {}^{\{p\}}\big(\, {}^{\{r\}}b \;|||^{\,\{s\}} c \,\big)\Big)} \\
&= a^{\,\{p,q\}} \;|||^{\,\{p,q\}} \underline{D_{p+q}\Big(\, {}^{\{r\}}b \;|||^{\,\{s\}} c \,\Big)} \\
&= a^{\,\{p,q\}} \;|||^{\,\{p,q\}} \Big(\underline{D_{p+q}\big(\, {}^{\{r\}}b \,\big)} \;|||\; D_{p+q}\big(\, {}^{\{s\}}c \,\big)\Big) \\
&= a^{\,\{p,q\}} \;|||^{\,\{p,q\}} \Big(\, {}^{\{r\}}b \;|||\; \underline{D_{p+q}\big(\, {}^{\{s\}}c \,\big)}\Big) \\
&= a^{\,\{p,q\}} \;|||^{\,\{p,q\}} \Big(\, {}^{\{r\}}b \;|||^{\,\{s\}} c \,\Big)
\end{aligned}
$$

For the law of the following lemma that relates D_{p+q} with the CCSP operator of parallelism, it is necessary to modify the buffers of the latter. Therefore, we continue

the mapping D_{p+q} on finite multisets $L \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP}$ of LTS labels by

$$D_{p+q}(L) \quad := \quad \{(A_{p+q}(m), e, A_{p+q}(n)) \ : \ (m, e, n) \in L\}$$

5.7.3 Lemma (Properties of Duplicating Places): Suppose that $P, Q \in \mathit{CProc}_{\Sigma_0,S}$ are processes, $p, q, s \in S$ are places with $p \neq q$, $v \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ is a place value, $o, r \in \mathcal{P}_{\mathrm{fi}} S$ is a finite set of places, $a, l \in \Sigma$ are visible events, $L, R \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP}$ are finite multisets of labels, $X \subseteq \Sigma$ is an interface set and $M \in \mathcal{M}_S$ is a marking.

1. For processes P that do not contain q, the characteristic equation of D_{p+q} can be simplified ($(m, e, n) \in \mathcal{L}_{CCSP}$, $q \notin \mathit{FP}(P)$):

$$(D_{p+q}(P))\Big\downarrow \begin{matrix} m \\ e \\ n \end{matrix} \Big\downarrow \quad = \quad \begin{cases} D_{p+q}(P\Big\downarrow \begin{matrix} m\setminus\{q\} \\ e \\ n\setminus\{q\} \end{matrix} \Big\downarrow) & m = A_{p+q}(m), n = A_{p+q}(n) \\ \varnothing & \text{otherwise} \end{cases} \tag{5.49}$$

2. Applying D_{p+q} to processes that do neither contain p nor q has no effect:

$$\{p, q\} \cap \mathit{FP}(P) = \varnothing \quad \Longrightarrow \quad D_{p+q}(P) = P$$

3. The application of D_{p+q} to processes defined using the CCSP operators can be described recursively. The first equations hold in general, and state that D_{p+q} distributes through the CCSP operators of prefixing, choice, labelling and hiding:

$$D_{p+q}(a \to P) \quad = \quad a \to D_{p+q}(P) \tag{5.50}$$
$$D_{p+q}(P \,\Box\, Q) \quad = \quad D_{p+q}(P) \,\Box\, D_{p+q}(Q) \tag{5.51}$$
$$D_{p+q}(l : P) \quad = \quad l : D_{p+q}(P) \tag{5.52}$$
$$D_{p+q}(P \setminus X) \quad = \quad D_{p+q}(P) \setminus X \tag{5.53}$$

The remaining identities hold for $q \notin o \cup r \cup \mathit{FP}(P) \cup \mathit{FP}(Q) \cup \mathit{FP}(L) \cup \mathit{FP}(R)$ (see footnote 2 in page 89 for the meaning of $\mathit{FP}(L)$), i.e. only provided that the clone place q does not already turn up in the argument of D_{p+q}.

$$D_{p+q}(P \,[\![\, L[X]R \,]\!]\, Q) \quad = \quad D_{p+q}(P) \,[\![\, D_{p+q}(L)[X]D_{p+q}(R) \,]\!]\, D_{p+q}(Q) \tag{5.54}$$

When applying D_{p+q} to transitions or to the token consumption operators, occurrences of p in the concerned sets o, r of places can simply be complemented with the additional place q:

$$D_{p+q}({}^{o}a^{\,r}) \quad = \quad {}^{A_{p+q}(o)}a^{\,A_{p+q}(r)} \tag{5.55}$$
$$D_{p+q}({}^{o}P) \quad = \quad {}^{A_{p+q}(o)}D_{p+q}(P) \tag{5.56}$$
$$D_{p+q}({}^{o*}P) \quad = \quad {}^{A_{p+q}(o)*}D_{p+q}(P) \tag{5.57}$$

It is not necessary to alter subterms in which p is bound by a causal state operator (because of item 2 and the premiss $q \notin FP(P)$). Also because of $q \notin FP(P)$, state operators for q are irrelevant and can be removed. In all other cases, D_{p+q} can be commuted with causal state operators:

$$D_{p+q}(s[v] : P) \quad = \quad \begin{cases} p[v] : P & s = p \\ D_{p+q}(P) & s = q \\ s[v] : D_{p+q}(P) & \text{otherwise} \end{cases} \tag{5.58}$$

4. As in Lem. 5.4.3, we can give congruence characterisations of equivalence relation $\sim_M$ for D_{p+q}: For $q \notin FP(P) \cup FP(Q)$:

$$P \sim_M Q \quad \Longrightarrow \quad D_{p+q}(P) \sim_M D_{p+q}(Q) \tag{5.59}$$

*

5.7.4. Merging of Places in Processes

Besides the operation D_{p+q}, for the distribution of token consumption we also need a mapping that to a certain degree is the inverse of D_{p+q}. While the modification D_{p+q} causes cannot be inverted in general (because after the application of D_{p+q} it is no longer possible to distinguish the places p and q), a left inverse C_{p+q} can be defined when restricting D_{p+q} to those processes in which q (or analogously p) does not occur free. C_{p+q} (as we define it below) reverts the duplication performed by D_{p+q} by identifying p and q, namely each occurrence of any of the two places in a triple (m, a, n) that can be raised by a process is replaced with p.

The definition of an inverse mapping is mostly necessary for our rewriting system for place duplication on CCSP terms. This system is defined in a way that makes it possible to start the duplication not only at the root of a term, but at an arbitrary position within the term: The duplication is triggered by the distribution of a token consumption operator through parallelism, which occurs anywhere in the term, and from this position spreads out on the whole term. Because up to the end of the rewriting the modification is restricted to a subterm, to justify the correctness of the transformation—by asserting that the value of the complete term is not altered—we have to introduce a function that is semantically able to undo the effect of the duplication.

Because D_{p+q} is not surjective, an interesting issue is how to define C_{p+q} for processes that are not element of the image of D_{p+q}, i.e. for processes P that are able to raise triples (m, e, n) in which p or q do not turn up pairwise. This decision is crucial to make C_{p+q} usable for the distribution of token consumption, because the concerned processes are in general *not* just the result of applying D_{p+q}. Our choice of the definition of C_{p+q} is lead by the intention of primarily treating the consumption of tokens in a meaningful way, i.e. the situation that a place turns up in m but not in n. Therefore, it is necessary to cope with the repeated consumption of tokens from p and q, which can in general be performed by a process $P \notin D_{p+q}(CProc_{\Sigma_0,S})$, and which needs to be merged to a single consumption of a token from p in the resulting process $C_{p+q}(P)$ (in which q must no longer turn up). In this case, we regard the first consumption of a token from either p or q in an LTS label as the definite consumption of a token from p, and conceal subsequent consumptions from the other place (but without blocking them).

Table 5.6. The Relation $(m,n) \lhd_{p+q} (m',n')$ for Merging Places of Processes

$(m \cap \{p,q\}, n \cap \{p,q\})$	$(m' \cap \{p,q\}, n' \cap \{p,q\})$
$(\varnothing, \varnothing)$	$(\varnothing, \varnothing)$
$(\{p\}, \varnothing)$	$(\{p\}, \varnothing),\ (\{q\}, \varnothing),\ (\{p,q\}, \varnothing),$ $(\{p,q\}, \{p\}),\ (\{p,q\}, \{q\})$
$(\varnothing, \{p\})$	$(\varnothing, \{p,q\})$
$(\{p\}, \{p\})$	$(\{p\}, \{p\}),\ (\{q\}, \{q\}),\ (\{p,q\}, \{p,q\})$

For the concealment of places we need an auxiliary function, and we define a further relation for the merging of places as follows (for $P \in CProc_{\Sigma_0,S}$ and $m, n, m', n' \in \mathcal{P}_{\mathrm{fi}} S$):

$$Cl_{p+q}(m,n,P) \quad := \quad \begin{cases} p[\mathrm{M}] : P & m \cap \{p,q\} \backslash n = \{q\} \\ q[\mathrm{M}] : P & m \cap \{p,q\} \backslash n = \{p\} \\ P & \text{otherwise} \end{cases}$$

$$\begin{aligned} (m,n) \lhd_{p+q} (m',n') \quad :\Longleftrightarrow \quad & q \notin m \cup n \\ \wedge \quad & m \backslash \{p\} = m' \backslash \{p,q\} \ \wedge\ n \backslash \{p\} = n' \backslash \{p,q\} \\ \wedge \quad & \begin{cases} m' = m \ \wedge\ n' = n & p \notin m \cup n \\ n' \cap \{p,q\} \subsetneq m' \cap \{p,q\} & p \in m \backslash n \\ m' = m \ \wedge\ n' = n \cup \{q\} & p \in n \backslash m \\ n' \cap \{p,q\} = m' \cap \{p,q\} \neq \varnothing & p \in m \cap n \end{cases} \end{aligned}$$

Intuitively, the effect of Cl_{p+q} is that whenever the I/O pair (m,n) represents a consumption from either p or q (but not from both of them), then the initial value of the other place is set to 'marked', and the place is hidden by applying the causal state operator. The relation $\lhd_{p+q}$ describes the connection between the I/O (m,n) of a triple (m,e,n) raised by a process $C_{p+q}(P)$, and the I/O (m',n') of the corresponding step of the original process P: In (m,n), occurrences of the places p and q are merged into the single remaining place p. The definition is basically a case distinction on the different combinations of input and output of (m,n) w.r.t. p. $\lhd_{p+q}$ is also described in Table 5.6 by enumerating the elements of the relation.

There is an asymmetry between the consumption and the production of tokens (cases two and three) that entails that only simultaneous production of tokens for the places p and q is merged to a production for p. This behaviour reflects the fact that to a certain degree the complexity of the processes treated has to be restricted, so that only those ones remain that can also be represented in an equivalent way after "removing" the place q. The argument is inverse to the one given in the description of duplicating places and transitions of monotonic P/T nets above, where it is explained that the combination of the transitions a_1, a_2 behaves equivalently to the original single transition a. This

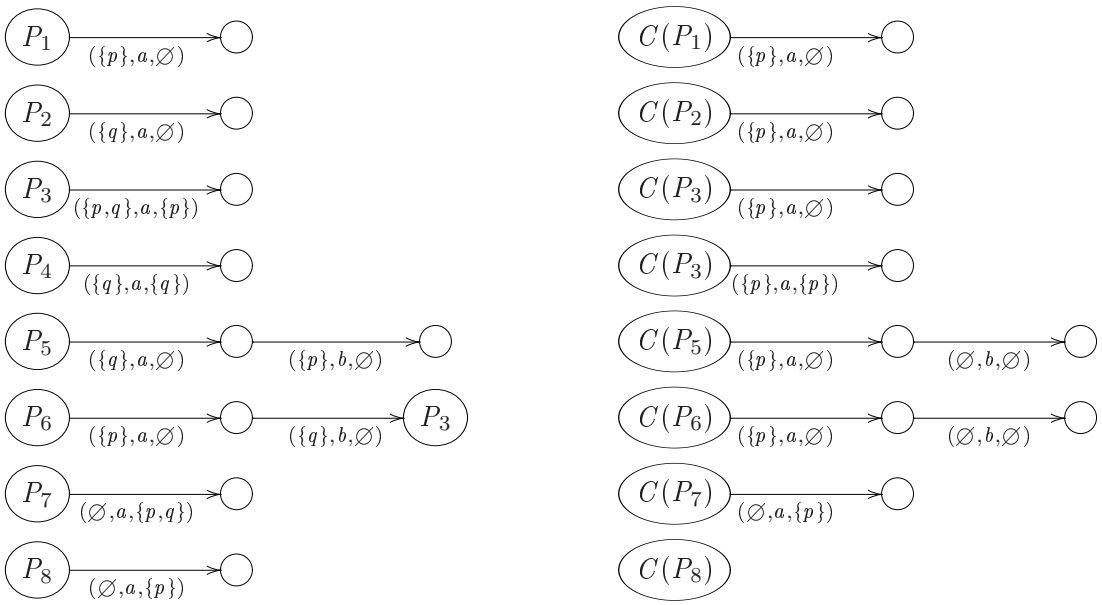
property would be destroyed by allowing that the values of the split places (that govern a_1, a_2) diverge (apart from the divergence caused by the firing of a_1, a_2). The restriction to processes that are sufficiently simple is made more explicit in the end of the section.

C_{p+q} is defined similarly to D_{p+q} by using coinduction, and as the mapping on $CProc_{\Sigma_0,S}$ that for each process $P \in CProc_{\Sigma_0,S}$ and for each LTS label $(m, e, n) \in \mathcal{L}_{CCSP}$ satisfies the equation

$$(C_{p+q}(P))\Big\downarrow \begin{matrix} m \\ e \\ n \end{matrix} \Big\downarrow \quad = \quad \bigcup_{\substack{m',n' \in \mathcal{P}_{\mathrm{fi}} S \\ (m,n) \lhd_{p+q} (m',n')}} C_{p+q}(Cl_{p+q}(m', n', P\Big\downarrow \begin{matrix} m' \\ e \\ n' \end{matrix} \Big\downarrow))$$

An alternative and equivalent characterisation of C_{p+q} using an SOS rule is given in Table 5.5.

5.7.4 Example: We apply C_{p+q} to different processes represented through LTS diagrams; C_{p+q} is abbreviated with C. The behaviour of $C_{p+q}(P_6)$ is caused by the causal state operator $q[\mathrm{M}] : \cdot$ that is added by Cl_{p+q} in the first step, and that does not allow repeated consumption from q. $C_{p+q}(P_8)$ is stamped by the mentioned aspect of the definition of $\lhd_{p+q}$, which only accepts simultaneous production of tokens for the places to be merged.



*

Properties related to the operator C_{p+q} are collected in the next lemma, in particular most of the algebraic identities that can be used for applying it syntactically.

5.7.5 Lemma (Properties of Merging Places): Suppose that $P, Q \in CProc_{\Sigma_0,S}$ are processes, $p, q, s, u \in S$ are places with $p \neq q$ and $u \neq s$, $v \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ is a place value, $o, r \in \mathcal{P}_{\mathrm{fi}} S$ is a finite set of places, $a, l \in \Sigma$ are visible events, $L, R \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP}$ are finite multisets of labels, $X \subseteq \Sigma$ is an interface set and $M, \tilde{M} \in \mathcal{M}_S$ are markings.

1. C_{p+q} is left inverse of D_{p+q} (restricted to processes that do not contain place q):

$$q \notin FP(P) \quad \Longrightarrow \quad C_{p+q}(D_{p+q}(P)) = P$$

2. Applying C_{p+q} to processes that do neither contain p nor q has no effect:

$$\{p, q\} \cap FP(P) = \varnothing \quad \Longrightarrow \quad C_{p+q}(P) = P$$

3. As in Lem. 5.4.3, we can give congruence characterisations of equivalence relation $\sim_M$ for C_{p+q}. Because $C_{p+q}(P)$ combines the behaviour of P w.r.t. the places p and q, it is necessary to ensure that the premiss $P \sim_M Q$ examines the behaviour of P and Q regarding both of the places sufficiently accurate. Hence, we have to demand $M(p) = M(q)$:

$$\begin{array}{r} M(p) = M(q) \wedge P \sim_M Q \\ \wedge\ M|_{S\setminus\{q\}} = \tilde{M}|_{S\setminus\{q\}} \end{array} \quad \Longrightarrow \quad C_{p+q}(P) \sim_{\tilde{M}} C_{p+q}(Q) \tag{5.60}$$

4. Occurrences of C_{p+q} in a term can be pulled out by cloning the places of "side terms", provided that $q \notin o \cup FP(Q) \cup FP(L) \cup FP(R)$, i.e. that the terms do not contain q. The causal state operator is not treated in full generality, as the case $r = p$ is left out. A law for this remaining case is given in Lem. 5.7.6.

$$a \rightarrow C_{p+q}(P) \quad \sim_M \quad C_{p+q}(a \rightarrow P) \tag{5.61}$$
$$C_{p+q}(P) \,\Box\, Q \quad \sim_M \quad C_{p+q}(P \,\Box\, D_{p+q}(Q)) \tag{5.62}$$
$$l : C_{p+q}(P) \quad \sim_M \quad C_{p+q}(l : P) \tag{5.63}$$
$$C_{p+q}(P) \setminus X \quad \sim_M \quad C_{p+q}(P \setminus X) \tag{5.64}$$
$$C_{p+q}(P) \,|[\, L[X]R \,]|\, Q \quad \sim_M \quad C_{p+q}(P \,|[\, D_{p+q}(L)[X]D_{p+q}(R) \,]|\, D_{p+q}(Q)) \tag{5.65}$$
$${}^{o}C_{p+q}(P) \quad \sim_M \quad C_{p+q}({}^{A_{p+q}(o)}P) \tag{5.66}$$
$${}^{o*}C_{p+q}(P) \quad \sim_M \quad C_{p+q}({}^{A_{p+q}(o)*}P) \tag{5.67}$$

For $r = q$, the causal state operator is irrelevant, because q does not turn up free in C_{p+q}(P). Otherwise, C_{p+q} can be commuted with the state operator. For $r \neq p$ (this case $r = p$ is left out):

$$s[v] : C_{p+q}(P) \quad \sim_M \quad \begin{cases} C_{p+q}(P) & s = q \\ C_{p+q}(s[v] : P) & \text{otherwise} \end{cases} \tag{5.68}$$

5. The operators for duplicating and for merging places can be commuted, provided that the concerned places are disjoint:

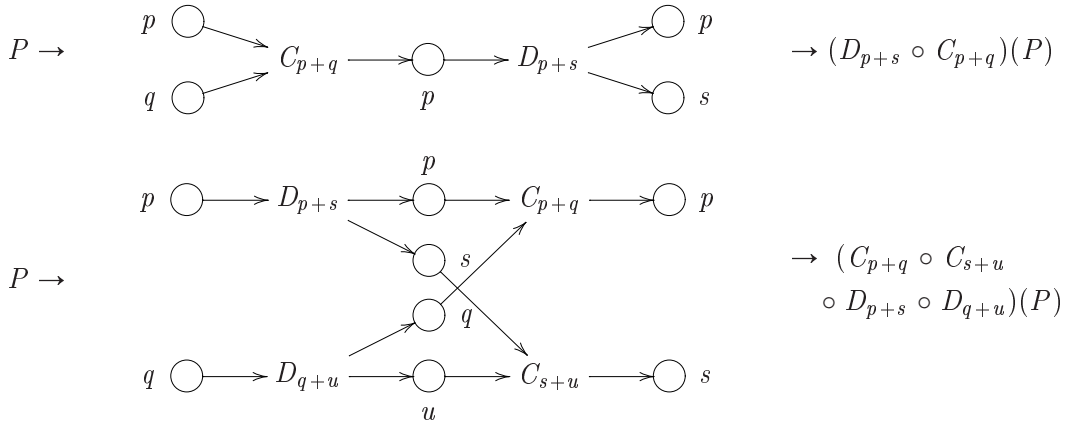
$$\{p, q\} \cap \{s, u\} = \varnothing \quad \Longrightarrow \quad C_{p+q}(D_{s+u}(P)) = D_{s+u}(C_{p+q}(P)) \tag{5.69}$$

Two operators that treat a common places p can also be permuted, but it is necessary to duplicate both occurrences:

$$\{p, q\} \cap \{s, u\} = \varnothing \ \wedge\ \{s, u\} \cap FP(P) = \varnothing \quad \Longrightarrow \\ (D_{p+s} \circ C_{p+q})(P) = (C_{p+q} \circ C_{s+u} \circ D_{p+s} \circ D_{q+u})(P) \tag{5.70}$$

*

Figure 5.4. Nesting and Commuting of Duplicating and Merging Places of Processes



To conceive the last law (Eq. 5.70) of the lemma, in Fig. 5.4 we illustrate the performed transformations. Instances of D_{q+u} create a copy u of an existing place q, whereas C_{p+q} has the effect of merging p and q. Consequently, the terms of Eq. 5.70 can be represented as networks, in which the edges describe propagation and renaming of places. Intuitively, the graphs show the equality of the two terms.

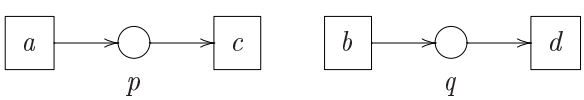
Elimination of Place Merging by Duplicating Causal State Operators: For treating the causal state operator, we first have to introduce a further tool that corresponds to the mentioned peculiarities of the definition of $\lhd_{p+q}$: Token production to the places p and q of a process P is only accounted for in $C_{p+q}(P)$, if the production is performed simultaneously to both places. The law to handle the state operator is similar to

$$p[v] : C_{p+q}(P) \quad = \quad p[v] : q[v] : P \tag{5.71}$$

but unfortunately this statement is wrong in general. This is not entirely surprising, because after all C_{p+q} merges two places p and q into one; because a single place is not able to store as much information as two, this possibly leads to clashes. With our definition of C_{p+q}, the simple process $P = a^{\{p\}}$ is already a counterexample of the equation, because the two sides are evaluate to the different processes

$$stop \quad \neq \quad a$$

By modifying the definition of C_{p+q}, the problem cannot be avoided for all processes in principle either, however, as can be conceived by considering the more complex process $Q = a^{\{p\}} \;|||\; b^{\{q\}} \;|||^{\{p\}}\; c \;|||^{\{q\}}\; d$. Q can be regarded as the P/T net



and we can formulate the following (seemingly reasonable, but inconsistent) requirements on the definition of C_{p+q}:

- The process $C_{p+q}(Q)$ must only contain place p, i.e. $FP(C_{p+q}(Q)) \subseteq \{p\}$, and in particular not q.
- Eq. (5.71) holds for Q:

  $$p[v] : C_{p+q}(Q) \quad = \quad p[v] : q[v] : Q$$

  This entails that for $v = \mathrm{E}$ the traces of the processes described by both sides of the equation are $abcd$, $bacd$, $abdc$, $badc$, $acbd$, $bdac$ (because these are the traces of the right process, whose behaviour is known). For $v = \mathrm{D}$, on the other hand, no execution is possible at all, which entails that the first triple raised by $C_{p+q}(Q)$ must in any case be $(\varnothing, x, \{p\})$ $(x \in \{a, b\})$.
- Eq. (5.71) holds for the modified process $Q \;|||^{\{p,q\}} e$:

  $$p[v] : C_{p+q}(Q \;|||^{\{p,q\}} e\,) \quad = \quad p[v] : q[v] : (Q \;|||^{\{p,q\}} e\,)$$

  The process of the right side of the equation, and thus also the left process, is for $v = \mathrm{E}$ able to raise event e, but this can only happen as the third execution step (two have to be performed by Q in the first place to produce tokens).
- Eq. (5.65) holds, which in particular means

  $$p[v] : (C_{p+q}(Q) \;|||^{\{p\}} e\,) \quad = \quad p[v] : C_{p+q}(Q \;|||^{\{p,q\}} e\,)$$

  This equation combines processes whose behaviour is considered in the previous two items. If in the first step $C_{p+q}(Q)$ of the left side necessarily produces a token for p, then the left process will have a trace in which event e is raised in the second step. This cannot happen for the process of the right side, however, because according to the observations of the last item, e can only occur as a third step in this case.

We cope with this problem by distinguishing those processes that comply to the described token production discipline, i.e. those ones that never produce tokens exclusively for one of the places p and q. When used as a premiss for Eq. (5.71), this discipline can also be weakened, because actually only an initial part of the behaviour of the process has to be investigated. As we can observe that the discipline is respected in all situations in which the elimination law is needed anyway, its imposition is not a crucial restriction. We capture the token production restriction in a formal way by defining an auxiliary mapping T_{p+q} on $CProc_{\Sigma_0,S}$, which modifies a process by simply removing all steps that violate the discipline. Admissible processes P are then characterised by the equation $T_{p+q}(P) = P$, which we also denote with

$$T^{fix}_{p+q}(P) \quad :\Longleftrightarrow \quad T_{p+q}(P) = P$$

T_{p+q} is defined by using coinduction as the function on $CProc_{\Sigma_0,S}$ that for each process $P \in CProc_{\Sigma_0,S}$ and for each LTS label $(m, e, n) \in \mathcal{L}_{CCSP}$ satisfies the equation

$$(T_{p+q}(P))\Big\downarrow^{m}_{e}_{n}\Big\downarrow \quad = \quad \begin{cases} P\Big\downarrow^{m}_{e}_{n}\Big\downarrow & A_{p+q}(n \backslash m) = n \backslash m \\ \varnothing & \text{otherwise} \end{cases}$$

Using this notion of well-formed processes, we can finally formulate a premiss that weakens statement (5.71) sufficiently. To make the property useful, we also give criteria to recognise concerned processes as well-formed. Essentially, we can observe that well-formedness is inherited by the identities given in Lem. 5.7.5, 4.

5.7.6 Lemma (Elimination of Place Merging): Suppose that $P, Q \in CProc_{\Sigma_0,S}$ are processes, $p, q, s, u \in S$ are places with $p \neq q$ and $r \neq s$, $v \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ is a place value, $o \in \mathcal{P}_{\mathrm{fi}} S$ is a finite set of places, $a, l \in \Sigma$ are visible events, $L, R \in \mathrm{mult}_{\mathrm{fi}} \mathcal{L}_{CCSP}$ are finite multisets of labels, $X \subseteq \Sigma$ is an interface set and $M \in \mathcal{M}_S$ are markings.

1. The operation C_{p+q} can be eliminated, if it is in the scope of a state operator $p[v] : \cdot$, provided that the concerned process complies to the discipline described above at least for an initial part of its behaviour (which is described by truncating further parts using an approximation M):

$$M(p) \leqslant v \wedge M(q) \leqslant v \wedge T^{fix}_{p+q}(M : P) \\ \Longrightarrow \quad p[v] : C_{p+q}(P) \quad \sim_M \quad p[v] : q[v] : P \tag{5.72}$$

2. Processes that arise from distributing token consumption (Lem. 5.7.7 below) obey the token production discipline:

$$T^{fix}_{p+q}\Big({}^{\{p\}} D_{p+q}(P) \,|[\, D_{p+q}(L)[X] D_{p+q}(R) \,]|\, {}^{\{q\}} D_{p+q}(Q) \Big) \tag{5.73}$$

$$T^{fix}_{p+q}\Big({}^{\{p\}*} D_{p+q}(P) \,|[\, D_{p+q}(L)[X] D_{p+q}(R) \,]|\, {}^{\{q\}*} D_{p+q}(Q) \Big) \tag{5.74}$$

3. Truncation of a process by applying an approximation preserves the token production discipline:

$$T^{fix}_{p+q}(P) \quad \Longrightarrow \quad T^{fix}_{p+q}(M : P) \tag{5.75}$$

4. The property T^{fix}_{p+q} is inherited by terms that are built using the equations of Lem. 5.7.5, 4: For $q \notin o \cup FP(Q) \cup FP(L) \cup FP(R)$, $s \notin \{p, q\}$ and $T^{fix}_{p+q}(P)$ we have:

$$T^{fix}_{p+q}(a \to P) \tag{5.76}$$

$$T^{fix}_{p+q}(P \,\Box\, D_{p+q}(Q)) \tag{5.77}$$

$$T^{fix}_{p+q}(l : P) \tag{5.78}$$

$$T^{fix}_{p+q}(P \setminus X) \tag{5.79}$$

$$T^{fix}_{p+q}(P \,|[\, D_{p+q}(L)[X] D_{p+q}(R) \,]|\, D_{p+q}(Q)) \tag{5.80}$$

$$T^{fix}_{p+q}({}^{A_{p+q}(o)} P) \tag{5.81}$$

$$T^{fix}_{p+q}({}^{A_{p+q}(o)*} P) \tag{5.82}$$

$$T^{fix}_{p+q}(s[v] : P) \tag{5.83}$$

The last three propositions correspond to the statements of Lem. 5.7.5, 5, and require the premisses $\{p, q\} \cap \{s, u\} = \varnothing$:

$$T^{fix}_{p+q}(D_{s+u}(P)) \tag{5.84}$$

On the right side of the last equation of Lem. 5.7.5, 5, place merging turns up twice; the next two propositions treat the particular arguments of these occurrences. Additionally to $\{p, q\} \cap \{s, u\} = \varnothing$, we need the premiss that s and u do not turn up in P, i.e. $\{s, u\} \cap FP(P) = \varnothing$:

$$T^{fix}_{s+u}(D_{p+s}(D_{q+u})(P)) \tag{5.85}$$

$$T^{fix}_{p+q}(C_{s+u}(D_{p+s}(D_{q+u})(P))) \tag{5.86}$$

*

In combination, the statements of the lemma ensure that whenever in a CCSP term the operator C_{p+q} is introduced to distribute token consumption through parallelism, the argument P of the new occurrence of C_{p+q} will have the property $T^{fix}_{p+q}(M : P)$ for an arbitrary marking M. The same holds for all other elements $P' \sim_M P$ of the M-congruence class of P (because $P' \sim_M P$ is defined as $M : P' = M : P$), which means that algebraic laws that preserve processes up to M-congruence—like in Lem. 5.7.5—can be used to alter the argument of C_{p+q}. Likewise, occurrences of C_{p+q} can be pulled out by using Lem. 5.7.5, 4, and the new argument process P' will again have the property $T^{fix}_{p+q}(M : P')$: For instance using (5.76) we have

$$T^{fix}_{p+q}(M : P) \underset{(5.36)}{\Longrightarrow} \begin{array}{l} T^{fix}_{p+q}(a \to M : P) \\ T^{fix}_{p+q}(M : (a \to P)) \end{array} \overset{(5.75)}{\Longrightarrow} T^{fix}_{p+q}(M : (a \to M : P))$$

The only CCSP operator for which this derivation does not hold in general (but still for relevant situations) is the causal state operator, because the appropriate congruence implication (5.43) does not cover arbitrary markings M.

5.7.5. Distribution of Token Consumption

By using the two operators D_{p+q} and C_{p+q}, we are able to formulate an algebraic law for the distribution of token consumption. The law is given in a general version for the unrestricted operator of parallelism, but is in the sequel mainly used to treat interleaving. As mentioned afore, we only treat single places at a time, which poses no restriction, as by Lem. 5.2.3 it is possible to split simultaneous consumption into a sequence of individual consumptions:

$${}^{o}P \quad \rightsquigarrow \quad {}^{\{p_1\}\{p_2\}\cdots}P \qquad o = \{p_1, p_2, \ldots\}$$

5.7.7 Lemma (Propagation through Token Consumption): Suppose that $P, Q \in CProc_{\Sigma_0,S}$ are processes, $p, q \in S$ are distinct places, $L, R \in \mathrm{mult_{fi}}\,\mathcal{L}_{CCSP}$ are finite multisets of labels, $X \subseteq \Sigma$ is an interface set and $M \in \mathcal{M}_S$ is a marking.

Token consumption can be distributed through parallelism according to the following equations, provided that the place q does not turn up in concerned processes or labels: $q \notin FP(P) \cup FP(Q) \cup FP(L) \cup FP(R)$.

$$
\begin{aligned}
&{}^{\{p\}}(P \,[\![\, L[X]R \,]\!]\, Q) \\
&\qquad \sim_M \quad C_{p+q}\Big({}^{\{p\}}D_{p+q}(P) \,[\![\, D_{p+q}(L)[X]D_{p+q}(R) \,]\!]\, {}^{\{q\}}D_{p+q}(Q)\Big) \qquad (5.87)\\
&{}^{\{p\}*}(P \,[\![\, L[X]R \,]\!]\, Q) \\
&\qquad \sim_M \quad C_{p+q}\Big({}^{\{p\}*}D_{p+q}(P) \,[\![\, D_{p+q}(L)[X]D_{p+q}(R) \,]\!]\, {}^{\{q\}*}D_{p+q}(Q)\Big) \qquad (5.88)
\end{aligned}
$$

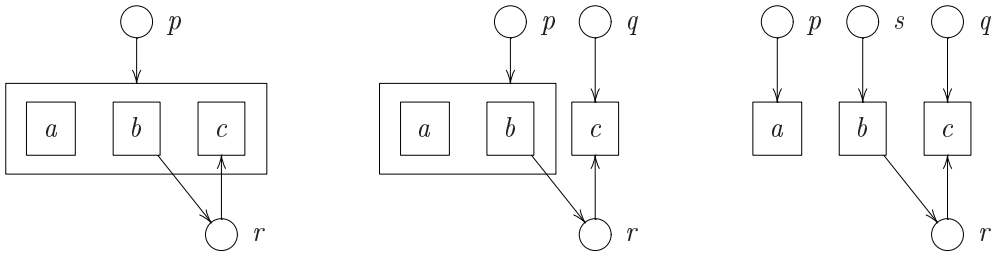
*

The necessary duplication of the place p is in the equations achieved in two parts: For subterms that are in the scope of the parallel operator by the application of D_{p+q}, and for other parts of the process term by the inverse operation C_{p+q}.

5.7.8 Example: We apply Eq. (5.87) twice to modify a process term. The application of D_{p+}. is carried out immediately (but does not have any effect, since neither p nor the places introduced turn up in the subterms). The new places that are introduced are q and s.

$$
\begin{aligned}
{}^{\{p\}}\Big((a \;|||\; b\,{}^{\{r\}}) \;|||\; {}^{\{r\}}c \Big) \quad &\rightsquigarrow \quad C_{p+q}\Big({}^{\{p\}}(a \;|||\; b\,{}^{\{r\}}) \;|||\; {}^{\{r,q\}}c \Big) \\
&\rightsquigarrow \quad C_{p+q}\Big(C_{p+s}({}^{\{p\}}a \;|||\; {}^{\{s\}}b\,{}^{\{r\}}) \;|||\; {}^{\{r,q\}}c \Big)
\end{aligned}
$$

The terms are illustrated by the following three "nested" P/T nets, which intuitively render the terms by regarding CCSP transitions as P/T net transitions. The operators C_{p+}. are ignored in the diagrams.



*

6. Transformation of CCSP Terms to Head Normal Form

The complete treatment of CCSP in the previous sections is performed semantically, and accordingly the operators described in Sect. 4.2.3 are regarded as mappings on a set of processes rather than as symbols. For the purpose of defining a calculus for CCSP based on the semantic properties given so far, we still need to define a formal syntax and a connection between the two aspects. Though the syntax introduced in the next paragraphs in general sticks closely to the semantic CCSP operators, at some points distinctions are drawn to make the construction of a calculus feasible, and in particular to make the logic extensible for elements that are not directly related to CSP (for instance to define a syntactic correspondent of the mapping $T(\cdot,\cdot)$ of Sect. 2.3 about JCSP). We do not concentrate on one way of reasoning about processes, which could for instance be performed using the modal μ-calculus or temporal logics, but instead we describe in principle how our analysis procedure for CSP terms can be embedded into these formalisms. To formulate the procedure, certain (weak) requirements have to be imposed on the framework, as it is somehow necessary to render conditions that arise from case distinctions into formulas. These requirements are for instance met by a classical first-order predicate logic.

The notions of events and of places are in the semantic world of CCSP both regarded as parameters of the coalgebra of existing processes, and can be seen as elements of the same level of abstraction in this respect. In contrast, the CCSP syntax treats events and places in fundamentally different ways: To denote events, we assume that an algebra of terms is given, whose elements are evaluated to events by an interpretation function—the exact definition and degrees of freedom thereby are mostly irrelevant. On the other hand, places are considered to be an immediate part of the syntax, so there is a one-to-one correspondence between the syntactic expressions that are regarded as places, and the places that occur in semantics. The identity of two places can thus be decided syntactically, and aliasing between syntactic descriptions of places is precluded. In fact, this notion of places is very similar to the concept of bound variables in expressions, whose distinctness or identity is also a purely syntactic problem, which is not affected by any kind of interpretation that is not fully determined.[1]

We use the P/T net causality that is available in CCSP complementary to term structure, and to establish a secondary notion of relationships between subterms; in contrast to the tree-like term structure, the connections encoded using the place concept have less restrictions, and can for instance represent arbitrary (multi-rooted) acyclic graphs. As discussed in Sect. 3.6, we regard P/T net causality to be a concept that is better suited for the representation of LTSs (or parts of LTSs) than plain CSP

[1] In the implementation of the analysis procedure that was performed as part of the thesis, places have actually been realised by using a concept of bound variables already existing.

process terms. Still, by identifying the place representations in syntax and semantics, computability in all analysis respects is sustained.

6.1. Prerequisites

For defining the CCSP syntax, we suppose that the following (not necessarily disjoint) domains of expressions exist, which could for instance be regarded as sets of expressions of certain types in a sorted predicate logic. The implementation of these prerequisites in detail is not important, however: In particular it is not necessary to handle the complete theory of sets, despite having a syntactic domain whose expressions embody sets of events. Depending on the usage of the CCSP framework, and on the range of CSP expressions that is to be treated, a very restricted event set domain can be chosen. It only has to be ensured that a tool for distinguishing those families of events is available that turn up as arguments of the parallel and the hiding operator.

- *EventVis*, *EventInt*, *SetOfEventVis* are sets of expressions describing visible and invisible events, as well as sets of visible events. The first two domains correspond to the sets Σ, Σ_τ defined in Sect. 2.1 for the semantics. We suppose that the following two function symbols exist, which correspond to the semantic function τ mapping visible events to invisible ones, and to the binary connective '.' that makes Σ a monoid (the mappings are described in Sect. 2.1):

$$\begin{aligned} \tau_t &: \mathit{EventVis} \to \mathit{EventInt} \\ \cdot_t \cdot &: \mathit{EventVis} \times \mathit{EventVis} \to \mathit{EventVis} \end{aligned}$$

- *ProcPrim* is a domain of expressions for "primary" processes, which can be used to embed other kinds of processes into CSP expressions. This set can also be chosen as empty, in which case the following considerations would apply to pure CCSP terms. When using CCSP for reasoning about concurrent Java programs, we could choose *ProcPrim* to be some kind of syntactic representation of the set of Java threads.

- *For* is a domain of formulas. The exact expressiveness of this formalism is not relevant, but we assume that the following statements are included (for $\Phi \in \mathit{For}$, $s, t \in \mathit{EventVis} \cup \mathit{EventInt}$, $X \in \mathit{SetOfEventVis}$):

$\mathit{true}, \mathit{false}$	$\in \mathit{For}$	
$\neg\Phi$	$\in \mathit{For}$	Negated formulas
$s \doteq t$	$\in \mathit{For}$	Equations of events
$s \in_t X$	$\in \mathit{For}$	Set element relation
$\mathit{int}(s)$	$\in \mathit{For}$	Test for internal steps

For the remainder of Sect. 6, we assume that these syntactic domains are fixed. Though most of the entities defined in the following pages are parametrised with the sets given so far, we omit corresponding indices to make notations more concise.

We define a *pre-interpretation* of the syntactical elements introduced so far to be a pair $(\Sigma_0, \mathrm{val}_{pre})$ consisting of

- A non-empty alphabet Σ_0 of atomic events, as in Sect. 2.1. To achieve a notation that is more concise, we again assume that by the choice of Σ_0 also the sets Σ, Σ_τ of visible and invisible events, as well as the operators '.' and τ are predetermined.

- A evaluation mapping val_{pre} that is defined on all syntactic domains, and that interprets expressions as follows:

$$\begin{aligned}
\mathrm{val}_{pre}(\mathit{EventVis}) &\subseteq \Sigma \\
\mathrm{val}_{pre}(\mathit{EventInt}) &\subseteq \Sigma_\tau \\
\mathrm{val}_{pre}(\mathit{SetOfEventVis}) &\subseteq \mathcal{P}\Sigma \\
\mathrm{val}_{pre}(\mathit{ProcPrim}) &\subseteq \mathit{Proc}_{\Sigma_0} \\
\mathrm{val}_{pre}(\mathit{For}) &\subseteq \mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}
\end{aligned}$$

  The evaluation is supposed to be consistent with the usual meaning (or the semantic counterparts) for the distinguished expressions from above, i.e. we have (for $\Phi \in \mathit{For}$, $u, v \in \mathit{EventVis}$, $s, t \in \mathit{EventVis} \cup \mathit{EventInt}$, $X \in \mathit{SetOfEventVis}$)

$$\begin{aligned}
\mathrm{val}_{pre}(\tau_t(u)) &= \tau(\mathrm{val}_{pre}(u)) \\
\mathrm{val}_{pre}(u._t v) &= \mathrm{val}_{pre}(u).\,\mathrm{val}_{pre}(v) \\
\mathrm{val}_{pre}(\mathit{true}) &= \mathrm{tt} \\
\mathrm{val}_{pre}(\mathit{false}) &= \mathrm{ff} \\
\mathrm{val}_{pre}(\neg\Phi) &= \begin{cases} \mathrm{ff} & \mathrm{val}_{pre}(\Phi) = \mathrm{tt} \\ \mathrm{tt} & \text{otherwise} \end{cases} \\
\mathrm{val}_{pre}(s \doteq t) &= \begin{cases} \mathrm{tt} & \mathrm{val}_{pre}(s) = \mathrm{val}_{pre}(t) \\ \mathrm{ff} & \text{otherwise} \end{cases} \\
\mathrm{val}_{pre}(s \in_t X) &= \begin{cases} \mathrm{tt} & \mathrm{val}_{pre}(s) \in \mathrm{val}_{pre}(X) \\ \mathrm{ff} & \text{otherwise} \end{cases} \\
\mathrm{val}_{pre}(\mathit{int}(s)) &= \begin{cases} \mathrm{tt} & \mathrm{val}_{pre}(s) \in \Sigma_\tau \\ \mathrm{ff} & \text{otherwise} \end{cases}
\end{aligned}$$

6.2. Syntax of CCSP

We define a formal syntax of CCSP, which is essentially a term algebra Proc_S built from the CCSP operators as in Sect. 4.2, as an extension of the domains of the last section. To establish a clear separation between syntactic symbols and the semantic CCSP operators, we add an index 't' to the former ones, in the same way as this issue is also handled in other parts of the report. The definition of Proc_S is inductive and follows the usual pattern.

First of all, we need to introduce a vocabulary S of places, which we require to be countably infinite. S is used both as part of the syntax and of the semantics (namely as parameter of the set $\mathit{CProc}_{\Sigma_0,S}$ of semantic processes), as mentioned afore.

Proc_S is defined to be the smallest set satisfying the following construction rules:

- Primary processes are processes: $ProcPrim \subseteq Proc_S$. We assume that the elements of $ProcPrim$ are distinct from all other terms that are defined to be members of $Proc_S$ in the remaining items, which can always be achieved by appropriate labelling in case of doubt.

- $stop_t \in Proc_S$ is the process term denoting blocked processes.

- If $a \in EventVis$ is a visible event term and $P \in Proc_S$ is a process term, then $a \rightarrow_t P \in Proc_S$ is a process term.

- If $a \in EventVis \cup EventInt$ is an event term and $o, r \in \mathcal{P}_{\text{fi}} S$ are finite sets of places, then ${}^{o}a\,{}_{t}^{r} \in Proc_S$ is a process term.

- If $P \in Proc_S$ is a process term and $o \in \mathcal{P}_{\text{fi}} S$ is a finite set of places, then ${}_{t}^{o}P \in Proc_S$ and ${}_{t}^{o*}P \in Proc_S$ are process terms.

- If $P, Q \in Proc_S$ are process terms, then $P \,\Box_t\, Q \in Proc_S$ and $P \,|||_t\, Q \in Proc_S$ are process terms as well. The introduction of a dedicated symbol $|||_t$ for interleaving is a difference between syntax and semantics of CCSP (in the sections about semantics, interleaving is always regarded as an abbreviation, and not as an operation that is introduced independently). Omitting interleaving as a symbol would lead to termination problems in the rewriting system, because the rules that treat the general operator of parallel composition introduce new occurrences of interleaving.

- If $l \in EventVis$ is a visible event term and $P \in Proc_S$ is a process term, then $l :_t P \in Proc_S$ is a process term.

- If $X \in SetOfEventVis$ is an event set term and $P \in Proc_S$ is a process term, then $P \setminus_t X \in Proc_S$ is a process term.

- If $P, Q \in Proc_S$ are process terms, $X \in SetOfEventVis$ is an event set term and

$$L, R \in \text{mult}_{\text{fi}}(\mathcal{P}_{\text{fi}} S \times (EventVis \cup EventInt) \times \mathcal{P}_{\text{fi}} S)$$

  are finite multisets of triples (m, a, n) consisting of two finite sets m, n of places and an event term a, then $P \,|[\, L[X]R \,]|_t\, Q \in Proc_S$ is a process term.

- If $p \in S$ is a place, $\Phi \in For$ is a formula and $P \in Proc_S$ is a process term, then $p[\Phi]_t : P \in Proc_S$ is a process term. The syntactic state operator can only assign two different place values to a place (namely 'empty' for $\text{val}(\Phi) = \text{ff}$ and 'marked' for $\text{val}(\Phi) = \text{tt}$), in contrast to the semantic one that covers the full meaning of places in monotonic P/T nets. Because the logic in general works with processes that contain free places, and because the representation of markings through the state operator is only used as an intermediate stage when transforming terms, this is no restriction.

- If $p, q \in S$ are places that are distinct and $P \in Proc_S$ is a process term, then $D^t_{p+q}(P) \in Proc_S$ and $C^t_{p+q}(P) \in Proc_S$ are process terms.

6.2.1 Example: Some members of the term algebra $Proc_S$ are (for $a, c \in EventVis$, $p, q \in S$, $X \in SetOfEventVis$):

$$a \to_t c \to_t stop_t \qquad \Big(a \to_t c \to_t stop_t\Big) \,[\![\, \varnothing[X]\varnothing \,]\!]_t \, \Big({}^{\{p\}}a_{\,t}\Big)$$

$$p[false]_t : \Big(a_t^{\{p\}} \,|||_t\, {}_t^{\{p\}}(c \to_t stop_t)\Big) \qquad D^t_{p+q}\big(a_t^{\{p\}}\big)$$

*

6.3. Interpretation of CCSP Syntax

The relation between the formal syntax of CCSP and semantics is straight-forward. We suppose that a pre-interpretation $(\Sigma_0, \mathrm{val}_{pre})$ of the CCSP-independent expression domains is given, and, by associating each operator with the corresponding semantic CCSP connective, we inductively continue this interpretation to the set $Proc_S$ of CCSP process terms.

Suppose that a vocabulary S of places and a pre-interpretation $(\Sigma_0, \mathrm{val}_{pre})$ is fixed. We define the evaluation $\mathrm{val} : Proc_S \to CProc_{\Sigma_0,S}$ as the unique solution of the following equations. To treat primary processes, we employ the embedding $cc_{\Sigma_0,S}$ of CSP processes into the CCSP process space, which is defined in Sect. 4.4. CCSP operators that are written without index t denote the semantic operators of Sect. 4.2, whereas operators with index t are the syntactic symbols of this section. ($P_{pri} \in ProcPrim$, $P, Q \in Proc_S$, $a \in EventVis$, $e \in EventVis \cup EventInt$, $o, r \in \mathcal{P}_{\mathrm{fi}} S$, $p, q \in S$ with $p \neq q$, $X \in SetOfEventVis$, $L, R \in \mathrm{mult}_{\mathrm{fi}}(\mathcal{P}_{\mathrm{fi}} S \times (EventVis \cup EventInt) \times \mathcal{P}_{\mathrm{fi}} S)$, $\Phi \in For$)

$$\begin{aligned}
\mathrm{val}(P_{pri}) &= cc_{\Sigma_0,S}(\mathrm{val}_{pre}(P_{pri})) \\
\mathrm{val}(stop_t) &= stop \\
\mathrm{val}(a \to_t P) &= \mathrm{val}_{pre}(a) \to \mathrm{val}(P) \\
\mathrm{val}({}^o e_t^{\,r}) &= {}^o(\mathrm{val}_{pre}(e))^{\,r} \\
{}^o_t P &= {}^o(\mathrm{val}(P)) \\
{}^{o*}_t P &= {}^{o*}(\mathrm{val}(P)) \\
\mathrm{val}(P \,\Box_t\, Q) &= \mathrm{val}(P) \,\Box\, \mathrm{val}(Q) \\
\mathrm{val}(P \,|||_t\, Q) &= \mathrm{val}(P) \,|||\, \mathrm{val}(Q) \\
\mathrm{val}(a :_t P) &= \mathrm{val}_{pre}(a) : \mathrm{val}(P) \\
\mathrm{val}(P \setminus_t X) &= \mathrm{val}(P) \setminus \mathrm{val}_{pre}(X) \\
\mathrm{val}(P \,[\![\, L[X]R \,]\!]_t\, Q) &= \mathrm{val}(P) \,[\![\, \mathrm{val}_{pre}(L)[\mathrm{val}_{pre}(X)]\, \mathrm{val}_{pre}(R) \,]\!]\, \mathrm{val}(Q) \\
\mathrm{val}(p[\Phi]_t : P) &= \begin{cases} p[\mathrm{E}] : \mathrm{val}(P) & \mathrm{val}_{pre}(\Phi) = \mathrm{ff} \\ p[\mathrm{M}] : \mathrm{val}(P) & \mathrm{val}_{pre}(\Phi) = \mathrm{tt} \end{cases} \\
\mathrm{val}(D^t_{p+q}(P)) &= D_{p+q}(\mathrm{val}(P)) \\
\mathrm{val}(C^t_{p+q}(P)) &= C_{p+q}(\mathrm{val}(P))
\end{aligned}$$

Thereby val_{pre} is continued to the buffer multisets L, R by

$$\mathrm{val}_{pre}(A) := \{(m, \mathrm{val}_{pre}(e), n) \,:\, (m, e, n) \in A\}$$

6.3.1. Free Places of Processes Described by CCSP Terms

When interpreting a process term, occurring places are simply passed on to the semantic CCSP operators without any further translation. All other kinds of arguments of the CCSP symbols are interpreted according to the chosen pre-interpretation $(\Sigma_0, \mathrm{val}_{pre})$, or are treated recursively by val. As a consequence of the direct handling of places, when describing a process through a term the set $FP(\mathrm{val}(P))$ of free places can be estimated (safely) by determining the places that syntactically occur in P. For the following relations, we define $FP(L)$—as for the buffers of the semantic parallel operator, Footnote 2 on page 89—to be the set of all places that turn up in elements of L. The function A_{p+q} is defined in Sect. 5.7, and maps a set of places to the smallest superset either containing both p and q or none of them.

$$\begin{array}{rcl}
FP(\mathrm{val}(P_{pri})) & = & \varnothing \\
FP(stop_t) & = & \varnothing \\
FP(\mathrm{val}(a \rightarrow_t P)) & = & FP(\mathrm{val}(P)) \\
FP(\mathrm{val}({}^{o}e\,{}^{r}_{t})) & = & o \cup r \\
FP(\mathrm{val}({}^{o}_{t}P)) & \subseteq & o \cup FP(\mathrm{val}(P)) \\
FP(\mathrm{val}({}^{o}{}^{*}_{t}P)) & \subseteq & o \cup FP(\mathrm{val}(P)) \\
FP(\mathrm{val}(P \,\Box_t\, Q)) & = & FP(\mathrm{val}(P)) \cup FP(\mathrm{val}(Q)) \\
FP(\mathrm{val}(P \,|||_t\, Q)) & = & FP(\mathrm{val}(P)) \cup FP(\mathrm{val}(Q)) \\
FP(\mathrm{val}(a :_t P)) & = & FP(\mathrm{val}(P)) \\
FP(\mathrm{val}(P \setminus_t X)) & = & FP(\mathrm{val}(P)) \\
FP(\mathrm{val}(P \,[\![\, L[X]R \,]\!]_t\, Q)) & \subseteq & FP(\mathrm{val}(P)) \,\cup\, FP(\mathrm{val}(Q)) \\
 & & \quad \cup\, FP(L) \,\cup\, FP(R) \\
FP(\mathrm{val}(p[\Phi]_t : P)) & \subseteq & FP(\mathrm{val}(P)) \backslash \{p\} \\
FP(\mathrm{val}(D^t_{p+q}(P))) & \subseteq & A_{p+q}(FP(\mathrm{val}(P))) \\
FP(\mathrm{val}(C^t_{p+q}(P))) & \subseteq & A_{p+q}(FP(\mathrm{val}(P))) \backslash \{q\}
\end{array}$$

6.3.2. Renaming of Places in a Process Term

Given a term $P \in Proc_S$ in which place q does not turn up (i.e. not even bound as in $q[false]_t : Q$), we can define the renaming of a place p to q in P simply by the replacement of all occurrences. A more complex method of substitution, which respects binding of places and has to avoid collisions, can be formulated as well, but is left out in this report. Some results of simple renaming (of p to q) are

$$\begin{array}{rcl}
{}^{\{p,r\}}a\,{}^{\{s\}}_{t} & \rightsquigarrow & {}^{\{q,r\}}a\,{}^{\{s\}}_{t} \\
{}^{\{p,r\}}a\,{}^{\{s\}}_{t} \,|||_t\, p[false]_t : {}^{\{p\}}_{t}stop_t & \rightsquigarrow & {}^{\{q,r\}}a\,{}^{\{s\}}_{t} \,|||_t\, q[false]_t : {}^{\{q\}}_{t}stop_t
\end{array}$$

As expected, the renaming of bound places does not alter the meaning of terms. If $P' \in Proc_S$ is obtained from P by renaming p to q, and if q does not turn up in P, then we have

$$\mathrm{val}(p[\Phi]_t : P) \quad = \quad \mathrm{val}(q[\Phi]_t : P')$$

We can also describe renaming of places in a semantic way, for instance by referring to the operators D_{p+q} and C_{q+p} of Sect. 5.7 (P' as in the last equation):

$$\mathrm{val}(P') \quad = \quad C_{q+p}(D_{p+q}(\mathrm{val}(P)))$$

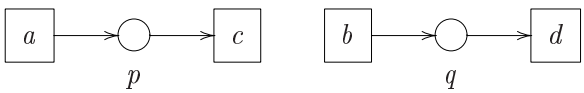
6.4. A Rewriting System for Analysing CCSP Process Terms

In Sect. 5, different kinds of algebraic laws about CCSP are collected that allow transformations of CCSP terms. Because of the tight connection between the formal syntax that is described in the last pages, and the vocabulary of the algebra CCSP from a semantic perspective, most of the laws can directly be translated to rewriting rules. For the concept of rewriting we refer to [DP01]. As discussed in Sect. 3.6, the aim of the modification is to establish a process term whose special shape allows the computation of initial parts of LTSs that describe the represented processes (different interpretations of a single process term are possible as usual). More exactly, all possible first steps of the processes are determined, and are encoded using a (computable) monotonic P/T net and a corresponding marking, together with formulas that describe case distinctions between different interpretations of the term. While the derived P/T net only captures *first* steps of the process, i.e. a complete translation of CSP processes to P/T nets is not achieved, the effect (the modification of the marking) of firing an enabled transition of the net coincides with the effect of the related step of the CSP process: To execute first steps, and to derive terms that represent the target processes, only a modification of the arguments of state operators that turn up in the term is needed.

6.4.1 Example: We consider the process represented by the following process term, which is terminal in the rewriting system we are about to define:

$$Q_t \quad = \quad p[\mathit{false}]_t : q[\mathit{false}]_t : \Big(a \,{}_t^{\{p\}}\, |||_t \; b \,{}_t^{\{q\}}\, |||_t \; {}^{\{p\}}c \,{}_t \; |||_t \; {}^{\{q\}}d \,{}_t \Big)$$

The term has a prenex form, and only consists of the state operator, interleaving and transitions. By a lemma given below, and because of the shape of Q_t, the initial behaviour of the process is represented by a P/T net that can directly be derived from the term by regarding CCSP transitions as transitions of the net. Initial markings of the net are encoded by the occurrences of formula *false* of Q_t, which in this case leads to the unique marking that assigns 'empty' to the places p, q:



The enabled transitions of the net are in the diagram labelled with a and b, and correspond to the first two transitions of the process term. Firing of transition a alters the marking by assigning p the value 'marked', which subsequently disables transition a and enables c. Analogously the firing of b adds a token to q.

While represented processes $\mathrm{val}(Q_t)$ are able to perform the same first steps as the P/T net, the effect of the steps is not only a modification of the marking represented by

the causal state operator, but furthermore the transitions are removed from the term. According to the operational semantics of CCSP, we have for instance (for simplicity we assume $\mathrm{val}_{pre}(s) = s$ for the event terms $s \in \{a, b, c, d\}$)

$$\begin{aligned} \mathrm{val}(Q_t) \quad &= \quad p[\mathrm{E}] : q[\mathrm{E}] : \Big(a^{\{p\}} \;|||\; b^{\{q\}} \;|||\; {}^{\{p\}}c \;|||\; {}^{\{q\}}d \Big) \\ &\xrightarrow{(\emptyset,a,\emptyset)} \quad p[\mathrm{M}] : q[\mathrm{E}] : \Big(stop \;|||\; b^{\{q\}} \;|||\; {}^{\{p\}}c \;|||\; {}^{\{q\}}d \Big) \end{aligned}$$

It can be observed, however, that removing the transition does not have any effect in this case. This can be seen by using the concept of approximations (Sect. 5.4), and is simply caused by the fact that the transition $a^{\{p\}}$ is not able to perform a step anyway (and will never be) if place p already contains a token.

A representation of the target process can thus be obtained from the original term Q_t by replacing the first formula *false* with *true*. If this observation is complemented with a storage of place values that is more efficient than the causal state operator, which is achieved below by adding a complete marking M as context information for a term, in fact a single open process term (i.e. a term containing free places) can be used to represent multiple processes. *

The rewriting system consists of two different classes of rules, one of which is responsible for synthesising transitions/events by translating the prefixing operator (as in Sect. 5.3; this group does actually only consist of a single rule), and the other one for the propagation of transitions through the skeleton of a process term, which is performed by employing algebraic laws of Sect. 5.5, 5.6, 5.7.

The application of several of the rules introduces fresh places $p \in S$, which are places that are required not to turn up in the whole process term or in the derivation so far (syntactically). Because the set S is countably infinite, and because a term and the derivation can only contain a finite number of places, fresh places do always exist.

Reduction of Prefixing and Choice: $P, Q \in Proc_S$ are process terms, $a \in EventVis$ is a visible event term and $p \in S$ is a fresh place.

The correctness of the rules for reducing the two operators follows from Lem. 5.3.1. Using the statements of the lemma it is also possible to formulate more powerful versions of the rules, which treat token consumption efficiently and avoid the introduction of places. We only give a simple version of the rules, which as well is sufficient to make the rewriting system handle arbitrary process terms. The first rule is the (only) rule for synthesising transitions.

$$\begin{aligned} a \rightarrow_t P \quad &\rightsquigarrow \quad p[\mathit{false}]_t : (\, a_t^{\{p\}} \;|||_t\; {}^{\{p\}}_t P) \\ P \,\Box_t\, Q \quad &\rightsquigarrow \quad p[\mathit{true}]_t : ({}^{\{p\}*}_t P \;|||_t\; {}^{\{p\}*}_t Q) \end{aligned} \tag{6.1}$$

Reordering of Interleaving: $\tilde{P} \in Proc_S$ is a process term that is not a transition (i.e. not a term of the kind ${}^{o}a_t^{r}$), $Q \in Proc_S$ is an arbitrary process term, $o, r \in \mathcal{P}_{\mathrm{fi}} S$ are finite sets of places and $e \in EventVis \cup EventInt$ is an event.

The correctness of the rules follows from Lem. 5.2.1.

$$\begin{array}{rcl}
\tilde{P} \mathbin{|||_t} {}^{o}a^{r}_{t} & \rightsquigarrow & {}^{o}a^{r}_{t} \mathbin{|||_t} \tilde{P} \\
\tilde{P} \mathbin{|||_t} \left({}^{o}a^{r}_{t} \mathbin{|||_t} Q\right) & \rightsquigarrow & {}^{o}a^{r}_{t} \mathbin{|||_t} \left(\tilde{P} \mathbin{|||_t} Q\right) \\
\left({}^{o}a^{r}_{t} \mathbin{|||_t} Q\right) \mathbin{|||_t} \tilde{P} & \rightsquigarrow & {}^{o}a^{r}_{t} \mathbin{|||_t} \left(Q \mathbin{|||_t} \tilde{P}\right)
\end{array}$$

Pulling Out the Causal State Operator: $P, Q \in \mathit{Proc}_S$ are process terms, $o \in \mathcal{P}_{\mathrm{fi}}S$ is a finite set of places, $L, R \in \mathrm{mult}_{\mathrm{fi}}(\mathcal{P}_{\mathrm{fi}}S \times (\mathit{EventVis} \cup \mathit{EventInt}) \times \mathcal{P}_{\mathrm{fi}}S)$ are buffer multisets, $a \in \mathit{EventVis}$ is a visible event term, $p \in S$ is a place that does not turn up in o, Q, L, R syntactically, $X \in \mathit{SetOfEventVis}$ is an event set term and $\Phi \in \mathit{For}$ is a formula.

The correctness of the rules follows from Lem. 5.2.2. We assume that in case of a collision, i.e. if the condition that p must not turn up in o, Q, L, R is violated, the place p is in the scope of the causal state operator implicitly renamed to a fresh place $p' \in S$.

$$\begin{array}{rcl}
a \rightarrow_t \left(p[\Phi]_t : P\right) & \rightsquigarrow & p[\Phi]_t : \left(a \rightarrow_t P\right) \\
{}^{o}_{t}\left(p[\Phi]_t : P\right) & \rightsquigarrow & p[\Phi]_t : {}^{o}_{t}P \\
{}^{o*}_{t}\left(p[\Phi]_t : P\right) & \rightsquigarrow & p[\Phi]_t : {}^{o*}_{t}P \\
\left(p[\Phi]_t : P\right) \Box_t Q & \rightsquigarrow & p[\Phi]_t : \left(P \Box_t Q\right) \\
Q \Box_t \left(p[\Phi]_t : P\right) & \rightsquigarrow & p[\Phi]_t : \left(Q \Box_t P\right) \\
\left(p[\Phi]_t : P\right) \mathbin{|\![} L[X]R \mathbin{]\!|_t} Q & \rightsquigarrow & p[\Phi]_t : \left(P \mathbin{|\![} L[X]R \mathbin{]\!|_t} Q\right) \\
Q \mathbin{|\![} L[X]R \mathbin{]\!|_t} \left(p[\Phi]_t : P\right) & \rightsquigarrow & p[\Phi]_t : \left(Q \mathbin{|\![} L[X]R \mathbin{]\!|_t} P\right) \\
\left(p[\Phi]_t : P\right) \mathbin{|||_t} Q & \rightsquigarrow & p[\Phi]_t : \left(P \mathbin{|||_t} Q\right) \\
Q \mathbin{|||_t} \left(p[\Phi]_t : P\right) & \rightsquigarrow & p[\Phi]_t : \left(Q \mathbin{|||_t} P\right) \\
a :_t \left(p[\Phi]_t : P\right) & \rightsquigarrow & p[\Phi]_t : \left(a :_t P\right) \\
\left(p[\Phi]_t : P\right) \setminus_t X & \rightsquigarrow & p[\Phi]_t : \left(P \setminus_t X\right)
\end{array}$$

Duplication of Places: $P, Q \in \mathit{Proc}_S$ are process terms, $a \in \mathit{EventVis}$ is a visible event term, $e \in \mathit{EventVis} \cup \mathit{EventInt}$ is an event term, $o, r \in \mathcal{P}_{\mathrm{fi}}S$ are finite sets of places, $L, R \in \mathrm{mult}_{\mathrm{fi}}(\mathcal{P}_{\mathrm{fi}}S \times (\mathit{EventVis} \cup \mathit{EventInt}) \times \mathcal{P}_{\mathrm{fi}}S)$ are buffer multisets, $\Phi \in \mathit{For}$ is a formula, $X \in \mathit{SetOfEventVis}$ is an event set term and $p, q, s, u \in S$ are distinct places, of which q does not turn up in o, r, P, Q, L, R syntactically.[2]

We use the function A_{p+q} that is defined in Sect. 5.7, and with L', R' we denote the buffer multisets that are derived from L, R by applying A_{p+q} to all place sets (again

[2]Contrary to the rules treating the causal state operator we do not use bound renaming to enforce this condition. Instead we show that the condition will always be satisfied anyway in Sect. 6.6.

as in Sect. 5.7), for instance

$$L' \quad := \quad \{(A_{p+q}(m), e, A_{p+q}(n)) \ : \ (m, e, n) \in L\}$$

The correctness of the first group of rules—treating D^t_{p+q}—follows from Lem. 5.7.3.

$$\begin{array}{rcl}
D^t_{p+q}(a \to P) & \rightsquigarrow & a \to D^t_{p+q}(P) \\
D^t_{p+q}(P \,\Box_t\, Q) & \rightsquigarrow & D^t_{p+q}(P) \,\Box_t\, D^t_{p+q}(Q) \\
D^t_{p+q}(a :_t P) & \rightsquigarrow & a :_t D^t_{p+q}(P) \\
D^t_{p+q}(P \setminus_t X) & \rightsquigarrow & D^t_{p+q}(P) \setminus_t X \\
D^t_{p+q}\Big(P \,[\![\, L[X]R \,]\!]_t\, Q\Big) & \rightsquigarrow & D^t_{p+q}(P) \,[\![\, L'[X]R' \,]\!]_t\, D^t_{p+q}(Q) \\
D^t_{p+q}\Big(P \,|||_t\, Q\Big) & \rightsquigarrow & D^t_{p+q}(P) \,|||_t\, D^t_{p+q}(Q) \\
D^t_{p+q}({}^{o}e^{r}_{t}) & \rightsquigarrow & {}^{A_{p+q}(o)}e^{A_{p+q}(r)}_{t} \\
D^t_{p+q}({}^{o}_{t}P) & \rightsquigarrow & {}^{A_{p+q}(o)}_{t}\Big(D^t_{p+q}(P)\Big) \\
D^t_{p+q}({}^{o*}_{t}P) & \rightsquigarrow & {}^{A_{p+q}(o)*}_{t}\Big(D^t_{p+q}(P)\Big) \\
D^t_{p+q}(p[\Phi]_t : P) & \rightsquigarrow & p[\Phi]_t : P
\end{array}$$

When respecting the well-formedness condition for process terms that is given in Sect. 6.6, the next rule is actually unnecessary, because it matches on a pattern that can never turn up in a well-formed term. More generally, in a well-formed term place q can never occur in the scope of a symbol D^t_{p+q}.

$$D^t_{p+q}(q[\Phi]_t : P) \quad \rightsquigarrow \quad D^t_{p+q}(P)$$

Note that by premiss p, q, s are distinct places, i.e. that the next rule, which handles the general case of a state operator meeting the operator for place duplication, does not subsume the previous two ones:

$$D^t_{p+q}(s[\Phi]_t : P) \quad \rightsquigarrow \quad s[\Phi]_t : D^t_{p+q}(P)$$

To treat expressions for primary processes (and to optimise the application of D^t_{p+q}) we can exploit the fact that D^t_{p+q} does not modify processes in which p, q do not turn up. Hence for process terms P in which p, q do not occur syntactically:

$$D^t_{p+q}(P) \quad \rightsquigarrow \quad P$$

In a similar way, C^t_{p+q} can be pulled out using laws of Lem. 5.7.5 and Lem. 5.7.6. Contrary to the former rules, the principal process term $\tilde{P} \in Proc_S$ of the next ones is allowed to contain place q. The correctness of the rule (*) is not obvious, because the condition $T^{fix}_{p+q}(M : \mathrm{val}(\tilde{P}))$ from Lem. 5.7.6 is not enforced as a premiss of the rule.

This matter is discussed in Sect. 6.6.

$$
\begin{array}{rcl}
a \to_t C^t_{p+q}(\tilde{P}) & \rightsquigarrow & C^t_{p+q}\Big(a \to_t \tilde{P}\Big) \\
C^t_{p+q}(\tilde{P}) \,\Box_t\, Q & \rightsquigarrow & C^t_{p+q}\Big(\tilde{P} \,\Box_t\, D^t_{p+q}(Q)\Big) \\
Q \,\Box_t\, C^t_{p+q}(\tilde{P}) & \rightsquigarrow & C^t_{p+q}\Big(D^t_{p+q}(Q) \,\Box_t\, \tilde{P}\Big) \\
a :_t C^t_{p+q}(\tilde{P}) & \rightsquigarrow & C^t_{p+q}\Big(a :_t \tilde{P}\Big) \\
C^t_{p+q}(\tilde{P}) \setminus_t X & \rightsquigarrow & C^t_{p+q}\Big(\tilde{P} \setminus_t X\Big) \\
C^t_{p+q}(\tilde{P}) \,[\![\, L[X]R \,]\!]_t\, Q & \rightsquigarrow & C^t_{p+q}\Big(\tilde{P} \,[\![\, L'[X]R' \,]\!]_t\, D^t_{p+q}(Q)\Big) \\
Q \,[\![\, L[X]R \,]\!]_t\, C^t_{p+q}(\tilde{P}) & \rightsquigarrow & C^t_{p+q}\Big(D^t_{p+q}(Q) \,[\![\, L'[X]R' \,]\!]_t\, \tilde{P}\Big) \\
C^t_{p+q}(\tilde{P}) \,|||_t\, Q & \rightsquigarrow & C^t_{p+q}\Big(\tilde{P} \,|||_t\, D^t_{p+q}(Q)\Big) \\
Q \,|||_t\, C^t_{p+q}(\tilde{P}) & \rightsquigarrow & C^t_{p+q}\Big(D^t_{p+q}(Q) \,|||_t\, \tilde{P}\Big) \\
{}^{o}_{t}\Big(C^t_{p+q}(\tilde{P})\Big) & \rightsquigarrow & C^t_{p+q}\Big({}^{A_{p+q}(o)}_{t}\tilde{P}\Big) \\
{}^{o*}_{t}\Big(C^t_{p+q}(\tilde{P})\Big) & \rightsquigarrow & C^t_{p+q}\Big({}^{A_{p+q}(o)*}_{t}\tilde{P}\Big) \\
p[\Phi]_t : C^t_{p+q}(\tilde{P}) & \rightsquigarrow & p[\Phi]_t : q[\Phi]_t : \tilde{P} \qquad (*)
\end{array}
$$

Again, the next rule is unnecessary for well-formed terms:

$$
\begin{array}{rcl}
q[\Phi]_t : C^t_{p+q}(\tilde{P}) & \rightsquigarrow & C^t_{p+q}(\tilde{P}) \\
s[\Phi]_t : C^t_{p+q}(\tilde{P}) & \rightsquigarrow & C^t_{p+q}\Big(s[\Phi]_t : \tilde{P}\Big)
\end{array}
$$

The last two rules make it possible to commute the operators for place duplication and place merging. For well-formed terms, situations apart from the ones treated by the rules cannot occur: It is not possible that the combination $D^t_{p'+q'}(C^t_{p+q}(\tilde{P}))$ with $q' \in \{p, q\}$ or $q \in \{p', q'\}$ turns up. The rules are backed by Lem. 5.7.5, 5.

$$
D^t_{s+u}(C^t_{p+q}(\tilde{P})) \quad \rightsquigarrow \quad C^t_{p+q}(D^t_{s+u}(\tilde{P}))
$$

The last rule has the premiss that s does not turn up in $\tilde{P}$, and that u is a fresh place:

$$
D^t_{p+s}(C^t_{p+q}(\tilde{P})) \quad \rightsquigarrow \quad C^t_{p+q}(C^t_{s+u}(D^t_{p+s}(D^t_{q+u}(\tilde{P}))
$$

Modification of Transitions: $e \in \mathit{EventVis} \cup \mathit{EventInt}$ is an event term, $o, r \in \mathcal{P}_{\mathrm{fi}} S$ are finite sets of places, $X \in \mathit{SetOfEventVis}$ is an event set term and $p \in S$ is a fresh place.

The rules of this group are derived from the laws of Lem. 5.2.4. We use the causal state operator to introduce case distinctions by assigning the place p values depending on the evaluation of certain formulas. As it is for instance necessary to distinguish

visible and invisible steps e when applying the labelling operator, we encode this condition as $p[int(e)]_t : \cdot$, and we attach p as an input place to the transition describing the result for this case, as in $p[int(e)]_t : {}^{\{p\}}e_{\,t}$. For $\mathrm{val}_{pre}(int(e)) = \mathrm{tt}$, the term will evaluate to the process $\mathrm{val}(e_{\,t})$, i.e. the addition of the state operator does not have any effect, whereas for $\mathrm{val}_{pre}(int(e)) = \mathrm{ff}$ the transition is always disabled and the resulting process is *stop*.

The transition for the second case in guarded in the same way (by the negation of the condition), and the two process terms are connected using interleaving. Because in any situation exactly one of the two terms has the value *stop*, by Eq. (5.8) the resulting process realises a case distinction.

$$
\begin{aligned}
{}^{s}_{t}({}^{o}e^{\,r}_{t}) \quad &\rightsquigarrow \quad \begin{cases} {}^{o\cup s}e^{\,r}_{t} & s \cap (o \cup r) = \varnothing \\ stop_t & \text{otherwise} \end{cases} \\
{}^{s*}\left({}^{o}e^{\,r}_{t}\right) \quad &\rightsquigarrow \quad \begin{cases} p[int(e)]_t : \left({}^{o\cup s\cup\{p\}}e^{\,r\cup s}_{t}\right) \\ \quad |||_t \;\; p[\neg int(e)]_t : \left({}^{o\cup s\cup\{p\}}e^{\,r}_{t}\right) & s \cap (o \cup r) = \varnothing \\ stop_t & \text{otherwise} \end{cases} \\
l :_t \left({}^{o}e^{\,r}_{t}\right) \quad &\rightsquigarrow \quad p[int(e)]_t : \left({}^{o\cup\{p\}}e^{\,r}_{t}\right) \;\; |||_t \;\; p[\neg int(e)]_t : \left({}^{o\cup\{p\}}(l :_t e)^{\,r}_{t}\right) \\
\left({}^{o}e^{\,r}_{t}\right) \setminus_t X \quad &\rightsquigarrow \quad p[e \in_t X]_t : \left({}^{o\cup\{p\}}(\tau_t(e))^{\,r}_{t}\right) \;\; |||_t \;\; p[\neg(e \in_t X)]_t : \left({}^{o\cup\{p\}}e^{\,r}_{t}\right)
\end{aligned}
$$

In practice, the introduction of case distinctions makes the process term grow rapidly, which leads to the necessity of interleaving the application of rewriting rules with rules that evaluate the conditions, and that (provided that the value can be determined) delete unnecessary parts of the process term, i.e. replace transitions that can never fire with the *stop* process. The same observation holds for the next rules, in which case distinctions are used likewise. The simplification of process terms is described in more detail in Sect. 6.5 and Sect. 6.9

Propagation of Transitions through Parallelism: $P, Q \in Proc_S$ are process terms, $o, r \in \mathcal{P}_{\mathrm{fi}}S$ are finite sets of places, $L, R \in \mathrm{mult}_{\mathrm{fi}}(\mathcal{P}_{\mathrm{fi}}S \times (EventVis \cup EventInt) \times \mathcal{P}_{\mathrm{fi}}S)$ are buffer multisets, $e \in EventVis \cup EventInt$ is an event term, $p, q, s, t \in S$ are distinct and fresh places and $X \in SetOfEventVis$ is an event set term.

To cover the equations of Lem. 5.5.2 using rewriting rules, it is again necessary to make a number of case distinctions for two kinds of conditions: The distinction between synchronisation and interleaving ($e \in_t X$), and for the former case the detection of elements of the buffers that are suitable for synchronisation ($e \doteq e'$).

For the selection of the buffer elements, we define a modified version $mrg_t\big((o, e, r), L\big)$ of the merge term (the original one is introduced in Def. 5.5.1), in which each transition is guarded by a causal state operator:

$$
mrg_t\big((o, e, r), L\big) \quad := \quad |||_t \underset{(o'',r'')=(o,r)+(o',r')}{\underset{(o',e',r')\in L}{}} s[e \doteq e'] : \left({}^{o''\cup\{s\}}e^{\,r''}_{t}\right)
$$

The rules for the propagation of transitions through parallelism combine the two equations of Lem. 5.5.2, which treat interleaving and synchronisation separately. The transition ${}^{o\cup\{q\}}e^{\,r}_{t}$ representing interleaving is guarded with the condition $q[\neg(e \in_t X)]_t : \cdot$

accordingly, whereas other parts added by the rewriting belong the synchronised case, and are guarded with $p[e \in_t X]_t : \cdot$. The four rewriting rules handle the propagation of transitions from both sides of the parallel operator; it is also necessary to account for the case of a single transition occurring as an argument (in other cases the rules for ordering interleaved processes establish the shape of the arguments on which the rules match).

$$
\begin{array}{l}
P \,[\![\, L[X]R \,]\!]_t \left({}^{o}e^{\,r}_{\,t} \;|||_t\; Q \right) \\
\quad \rightsquigarrow \quad p[e \in_t X]_t : q[\neg(e \in_t X)]_t : \quad \left({}^{o\cup\{q\}}e^{\,r}_{\,t} \right) \;|||_t\; mrg_t\big((o \cup \{p\}, e, r), L\big) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad |||_t \;\; \Big(P \,[\![\, L[X]R \uplus \{(o \cup \{p\}, e, r)\} \,]\!]\, Q \Big) \\
P \,[\![\, L[X]R \,]\!]_t \, {}^{o}e^{\,r}_{\,t} \\
\quad \rightsquigarrow \quad p[e \in_t X]_t : q[\neg(e \in_t X)]_t : \quad \left({}^{o\cup\{q\}}e^{\,r}_{\,t} \right) \;|||_t\; mrg_t\big((o \cup \{p\}, e, r), L\big) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad |||_t \;\; \Big(P \,[\![\, L[X]R \uplus \{(o \cup \{p\}, e, r)\} \,]\!]\, stop_t \Big) \\
\left({}^{o}e^{\,r}_{\,t} \;|||_t\; P \right) [\![\, L[X]R \,]\!]_t \, Q \\
\quad \rightsquigarrow \quad p[e \in_t X]_t : q[\neg(e \in_t X)]_t : \quad \left({}^{o\cup\{q\}}e^{\,r}_{\,t} \right) \;|||_t\; mrg_t\big((o \cup \{p\}, e, r), R\big) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad |||_t \;\; \Big(P \,[\![\, L \uplus \{(o \cup \{p\}, e, r)\}[X]R \,]\!]\, Q \Big) \\
{}^{o}e^{\,r}_{\,t} \,[\![\, L[X]R \,]\!]_t \, Q \\
\quad \rightsquigarrow \quad p[e \in_t X]_t : q[\neg(e \in_t X)]_t : \quad \left({}^{o\cup\{q\}}e^{\,r}_{\,t} \right) \;|||_t\; mrg_t\big((o \cup \{p\}, e, r), R\big) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad |||_t \;\; \Big(stop_t \,[\![\, L \uplus \{(o \cup \{p\}, e, r)\}[X]R \,]\!]\, P \Big)
\end{array}
$$

A technicality remains to be dealt with, namely a premiss of Lem. 5.5.2, which states that the concerned transitions and labels have to be duplicable. For the transition ${}^{o}e^{\,r}_{\,t}$ this is already assured, because the rewriting adds common (and fresh) input places p, q to the copies of the transitions. The elements of the buffers L, R, however, do in general not satisfy the premiss, which makes illegal transformations possible. Hence we assume that in case of non-duplicable elements of the buffer L (R), an application of the first two rules (the last two rules) is always preceded with appropriate applications of the following two rules, which do not alter the behaviour of a process (p is a fresh place):

$$
\begin{array}{lcl}
P \,[\![\, L \uplus \{(o, e, r)\}[X]R \,]\!]\, Q & \rightsquigarrow & p[\mathit{true}] : (P \,[\![\, L \uplus \{(o \cup \{p\}, e, r)\}[X]R \,]\!]\, Q) \\
P \,[\![\, L[X]R \uplus \{(o, e, r)\} \,]\!]\, Q & \rightsquigarrow & p[\mathit{true}] : (P \,[\![\, L[X]R \uplus \{(o \cup \{p\}, e, r)\} \,]\!]\, Q)
\end{array}
$$

Propagation of Transitions through Labelling and Hiding: $P, Q \in \mathit{Proc}_S$ are process terms, $l \in \mathit{EventVis}$ is a visible event term and $X \in \mathit{SetOfEventVis}$ is an event set term. The rewriting are direct translations of the equations given in Lem. 5.6.1.

$$
\begin{array}{rcl}
l :_t (P \;|||_t\; Q) & \rightsquigarrow & (l :_t P) \;|||_t\; (l :_t Q) \\
(P \;|||_t\; Q) \setminus_t X & \rightsquigarrow & (P \setminus_t X) \;|||_t\; (Q \setminus_t X)
\end{array}
$$

Propagation of Transitions through Token Consumption: $P, Q \in Proc_S$ are process terms, $o \in \mathcal{P}_{\text{fi}} S$ is a finite set of places and $p, q \in S$ are distinct places, of which q is required to be fresh. p must not be an element of the set o.

The rules for distributing token consumption treat one place at a time, and are based on Lem. 5.7.7. By iteration (in combination with treating the symbols D^t_{p+q} and C^t_{p+q}, the rules for which are given above) arbitrary non-empty and finite sets o of places can be worked off. To handle empty sets o, we add two rules that simply remove the token consumption in that case, which is possible by Lem. 5.2.3.

Besides interleaving, in the same way also the general operator of parallelism can be commuted with token consumption, which is left out at this point as it is not strictly necessary to make the rewriting system work. For implementing the calculus, however, it is in our experience a good strategy to push down token consumption as far as possible, as in the long term this reduces the number of places that are introduced.

$$
\begin{aligned}
{}^{o\cup\{p\}}_{t}(P \;|||_t\; Q) &\quad\rightsquigarrow\quad {}^{o}_{t}\Big(C^t_{p+q}\Big({}^{\{p\}}_{t}(D^t_{p+q}(P)) \;|||_t\; {}^{\{q\}}_{t}(D^t_{p+q}(Q))\Big)\Big)\\
{}^{o\cup\{p\}*}_{t}(P \;|||_t\; Q) &\quad\rightsquigarrow\quad {}^{o*}_{t}\Big(C^t_{p+q}\Big({}^{\{p\}*}_{t}(D^t_{p+q}(P)) \;|||_t\; {}^{\{q\}*}_{t}(D^t_{p+q}(Q))\Big)\Big)\\
{}^{\varnothing}_{t}P &\quad\rightsquigarrow\quad P\\
{}^{\varnothing*}_{t}P &\quad\rightsquigarrow\quad P
\end{aligned}
$$

6.5. Example

To illustrate the rewriting system given in the last section, we apply it to the following CSP process:

$$\Big(a \to c \to P \;[\![\, \{c\} \,]\!]\; b \to c \to Q\Big) \setminus \{c\} \;[\![\, \{a,b\} \,]\!]\; \Big(a \to U \;|||\; b \to V\Big)$$

The system is a (small) part of a CSP model of channels from the Java Communicating Sequential Processes library, and is taken from [PJ00]. In the term the top-level parallel operator separates a channel process (consisting of two transceivers, which serve a sending and a receiving process (resp.)) from two interleaved processes/threads that communicate through the channel. The two subprocesses of the channel both raise events a/b in the first place to communicate with the actual processes, and consequently they synchronise using the event c that is hidden from the environment.

For simplicity, we assume that the events $a, b, c \in \Sigma$ are directly represented as constants $a, b, c \in \mathit{EventVis}$, and that the expressions $X, Y \in \mathit{SetOfEventVis}$ stand for the occurring sets:

$$\mathrm{val}_{pre}(X) = \{c\}, \qquad \mathrm{val}_{pre}(Y) = \{a, b\}$$

The process can then be written in CCSP syntax, provided that P_t, Q_t, U_t, V_t are appropriate terms for P, Q, U, V, as

$$\Big(a \to_t c \to_t P_t \;[\![\, \varnothing[X]\varnothing \,]\!]_t\; b \to_t c \to_t Q_t\Big) \setminus_t X \;[\![\, \varnothing[Y]\varnothing \,]\!]_t\; \Big(a \to_t U_t \;|||_t\; b \to_t V_t\Big)$$

Because there is no danger of mistaking semantic CCSP operators for syntactic ones (or vice versa) in this section, we leave out the index t in the next pages. In all of the following terms, the used symbols denote the syntactic operators.

The rewriting is shown in several parts for the different subprocesses; we start with the subterm $a \to c \to P_t$, which is translated to a set of transitions and the continuation P_t:

$$\begin{aligned}
&\underline{a \to c \to P_t} \\
&\quad\rightsquigarrow\quad p[\mathit{false}] : a^{\{p\}} \;|||^{\{p\}} (\underline{c \to P_t}) \\
&\quad\rightsquigarrow\quad p[\mathit{false}] : a^{\{p\}} \;|||^{\{p\}} \underline{\Big(q[\mathit{false}] : c^{\{q\}} \;|||^{\{q\}} P_t\Big)}
\end{aligned}$$

The causal state operators can be pulled out completely using the rules of the responsible group:

$$\begin{aligned}
&\quad\rightsquigarrow\quad p[\mathit{false}] : \underline{a^{\{p\}} \;|||\; q[\mathit{false}] :^{\{p\}} \Big(c^{\{q\}} \;|||^{\{q\}} P_t\Big)} \\
&\quad\rightsquigarrow\quad p[\mathit{false}] : q[\mathit{false}] : a^{\{p\}} \;|||^{\{p\}} \underline{\Big(c^{\{q\}} \;|||^{\{q\}} P_t\Big)}
\end{aligned}$$

Subsequently, the consumption of a token from p can be distributed by introducing a copy p_2 (this step is not necessary when using a more powerful rule for reducing prefixing):

$$\rightsquigarrow\quad p[\mathit{false}] : q[\mathit{false}] : a^{\{p\}} \;|||\; C_{p+p_2}\Big({}^{\{p\}}\underline{D_{p+p_2}\big(c^{\{q\}}\big)} \;|||^{\{p_2\}} \underline{D_{p+p_2}\big({}^{\{q\}} P_t\big)}\Big)$$

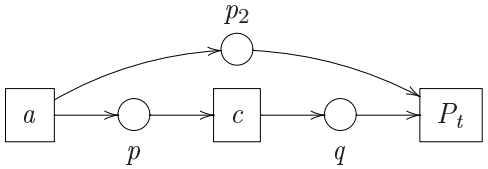
We directly remove the occurrences of D_{p+p_2}, as the places p, p_2 do not turn up in any of their arguments. C_{p+p_2} can be pulled out stepwise while inserting new occurrences of D_{p+p_2} as needed. Applying D_{p+p_2} to the transition $a^{\{p\}}$ has the effect of duplicating place p:

$$\begin{aligned}
&\overset{*}{\rightsquigarrow}\quad p[\mathit{false}] : q[\mathit{false}] : \underline{a^{\{p\}} \;|||\; C_{p+p_2}\Big({}^{\{p\}}\big(c^{\{q\}}\big) \;|||^{\{p_2\}\{q\}} P_t\Big)} \\
&\rightsquigarrow\quad p[\mathit{false}] : q[\mathit{false}] : C_{p+p_2}\Big(\underline{D_{p+p_2}\big(a^{\{p\}}\big)} \;|||\; \Big({}^{\{p\}}\big(c^{\{q\}}\big) \;|||^{\{p_2\}\{q\}} P_t\Big)\Big) \\
&\rightsquigarrow\quad p[\mathit{false}] : \underline{q[\mathit{false}] : C_{p+p_2}\Big(a^{\{p,p_2\}} \;|||\; \Big({}^{\{p\}}\big(c^{\{q\}}\big) \;|||^{\{p_2\}\{q\}} P_t\Big)\Big)} \\
&\rightsquigarrow\quad \underline{p[\mathit{false}] : C_{p+p_2}\Big(q[\mathit{false}] : a^{\{p,p_2\}} \;|||\; \Big({}^{\{p\}}\big(c^{\{q\}}\big) \;|||^{\{p_2\}\{q\}} P_t\Big)\Big)}
\end{aligned}$$

In the last step, the causal state operator for p is duplicated, and the consumption of a token from p is merged with the I/O of the transition in scope:

$$\overset{*}{\rightsquigarrow}\quad p[\mathit{false}] : p_2[\mathit{false}] : q[\mathit{false}] : a^{\{p,p_2\}} \;|||\; \Big({}^{\{p\}} c^{\{q\}} \;|||^{\{p_2\}\{q\}} P_t\Big)$$

As before, we depict the process term as a P/T net by regarding the CCSP transitions as P/T net transitions. It is obvious that the place p_2 could be removed without altering the net, but for a basic version of the rewriting system this is not strictly necessary.



A second subterm of the process term has the same shape, and can be rewritten in the same manner:

$$b \to c \to Q_t \quad \overset{*}{\rightsquigarrow} \quad r[\mathit{false}] : r_2[\mathit{false}] : s[\mathit{false}] : b\,^{\{r,r_2\}} \,|||\, \left({}^{\{r\}}c\,^{\{s\}} \,|||\, {}^{\{r_2\}\,\{s\}}Q_t \right)$$

The two parts of the process are connected with the operator $|[\varnothing[X]\varnothing]|$, and the combination is dealt with by applying the rule for propagation through parallelism. The reduction is similar to the one shown in Table 5.2. Occurrences of the causal state operator can again be moved to the root of the term, and are subsequently left out for reasons of conciseness. We are also using the transition notation ${}^{o}a\,^{r}$ for triples (o, a, r) that are element of buffers of the parallel composition operator.

As the interpretation of the set term X only contains the event c, the expected result of the rewriting is a term in which the transitions a and b are interleaved, and are thus moved out of the scope of the parallel operator (without being modified), and in which the events c raised by the subprocesses are merged and become a single transition. To recognise this pattern, it is necessary to employ background information, because the rewriting system does not know anything about the comparison of event terms, i.e. it cannot derive that a and c denote different events, or that a is not an element of X. In these cases the rewriting rules introduce case distinctions by means of the causal state operator, and essentially perform a reduction that suits all situations. This leads to rather large process terms with a lot of unnecessary transitions, together with guards that distinguish those transitions which are really needed. We carry out this "pure" rewriting in the next steps, but in practice it is necessary to evaluate conditions as they are introduced, and to remove superfluous parts of the process term immediately. Methods therefore are discussed in Sect. 6.9.

$$\begin{pmatrix} p[\mathit{false}] : p_2[\mathit{false}] : q[\mathit{false}] : a\,^{\{p,p_2\}} \,|||\, \left({}^{\{p\}}c\,^{\{q\}} \,|||\, {}^{\{p_2\}\,\{q\}}P_t \right) \\ |[\varnothing[X]\varnothing]| \\ r[\mathit{false}] : r_2[\mathit{false}] : s[\mathit{false}] : b\,^{\{r,r_2\}} \,|||\, \left({}^{\{r\}}c\,^{\{s\}} \,|||\, {}^{\{r_2\}\,\{s\}}Q_t \right) \end{pmatrix}$$

In the first place state operators are pulled out:

$$\overset{*}{\rightsquigarrow} \quad \ldots \quad \begin{pmatrix} \underline{a\,^{\{p,p_2\}}} \,|||\, \left({}^{\{p\}}c\,^{\{q\}} \,|||\, {}^{\{p_2\}\,\{q\}}P_t \right) \\ |[\varnothing[X]\varnothing]| \\ b\,^{\{r,r_2\}} \,|||\, \left({}^{\{r\}}c\,^{\{s\}} \,|||\, {}^{\{r_2\}\,\{s\}}Q_t \right) \end{pmatrix}$$

The remaining rewriting steps use the rules for propagation of transitions through parallelism. At this point the first case distinction between interleaving and synchronisation is introduced, i.e. the values of the condition $a \in_t X$ and its negation are assigned to places, which are used to guard the transitions representing the respective process behaviour. For $a \in_t X$, the transition ${}^{\{p,p_2\}}a$ is moved to the left buffer of the parallel operator (and is synchronised with the elements of the opposing buffer, which is empty), whereas in the other case it is simply moved out of scope.

$$\rightsquigarrow \quad c_1[a \in_t X] : \bar{c}_1[\neg(a \in_t X)] : {}^{\{\bar{c}_1\}}a^{\{p,p_2\}} \;|||\; \begin{pmatrix} \overline{{}^{\{p\}}c^{\{q\}} \;|||\; {}^{\{p_2\}}P_t} \\ |[\{{}^{\{c_1\}}a^{\{p,p_2\}}\}[X]\varnothing]| \\ b^{\{r,r_2\}} \;|||\; \left({}^{\{r\}}c^{\{s\}} \;|||\; {}^{\{r_2\}}Q_t\right) \end{pmatrix}$$

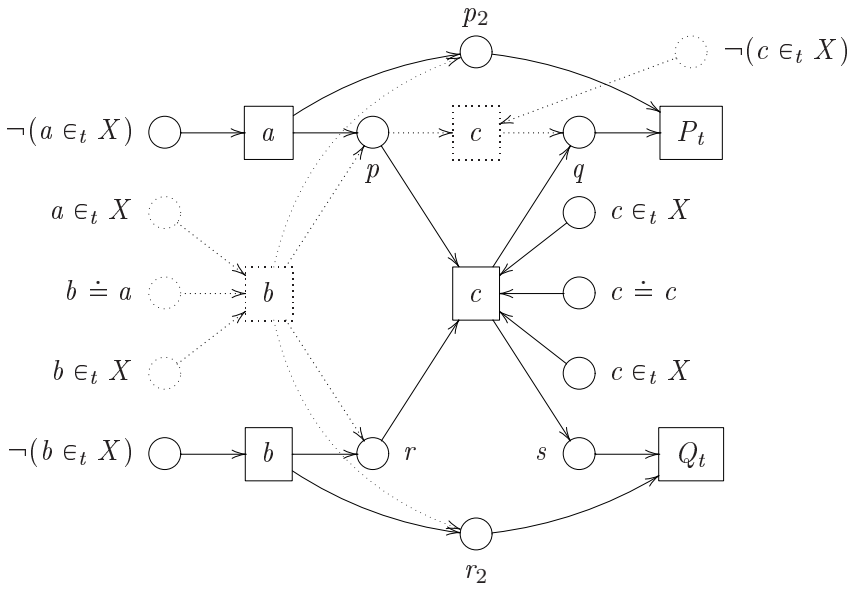
Before continuing the reduction of the process term, in practice an "external" procedure/calculus should be invoked to investigate the conditions that determine the values of the places c_1, $\bar{c}_1$. Because of the assumption that X represents the set $\{c\}$, this procedure could be expected to evaluate $a \in_t X$ to ff, and its negation to tt, and by using a deletion rule (which is not given as part of the rewriting system) this knowledge could be used to remove the element ${}^{\{c_1\}}a^{\{p,p_2\}}$ from the left buffer.
From now on, in each step we only show those transitions (outside of the parallel operator) that are generated by the particular modification. The next step propagates the second transition ${}^{\{p\}}c^{\{q\}}$:

$$\rightsquigarrow \quad \ldots \quad c_2[c \in_t X] : \bar{c}_2[\neg(c \in_t X)] : {}^{\{p,\bar{c}_2\}}c^{\{q\}} \;|||\; \begin{pmatrix} {}^{\{p_2\}}P_t \\ |[\{{}^{\{c_1\}}a^{\{p,p_2\}}, {}^{\{p,c_2\}}c^{\{q\}}\}[X]\varnothing]| \\ \underline{b^{\{r,r_2\}}} \;|||\; \left({}^{\{r\}}c^{\{s\}} \;|||\; {}^{\{r_2\}}Q_t\right) \end{pmatrix}$$

When propagating transitions of the right side, the elements of the left buffer can be used for synchronisation, which adds further conditioned transitions. Again, the evaluation of the created equations should in practice be performed as soon as possible to recognise impossible cases.

$$\begin{aligned} \rightsquigarrow \quad \ldots \quad c_3[b \in_t X] : \bar{c}_3[\neg(b \in_t X)] : {}^{\{\bar{c}_3\}}b^{\{r,r_2\}} \;|||\; & c_4[b \doteq a] : {}^{\{c_1,c_3,c_4\}}b^{\{p,p_2,r,r_2\}} \\ |||\; & c_5[b \doteq c] : {}^{\{p,c_2,c_3,c_5\}}b^{\{q,r,r_2\}} \\ |||\; & \begin{pmatrix} {}^{\{p_2\}}P_t \\ |[\{{}^{\{c_1\}}a^{\{p,p_2\}}, {}^{\{p,c_2\}}c^{\{q\}}\}[X]\{{}^{\{c_3\}}b^{\{r,r_2\}}\}]| \\ \underline{{}^{\{r\}}c^{\{s\}}} \;|||\; {}^{\{r_2\}}Q_t \end{pmatrix} \end{aligned}$$

Figure 6.1. Example of P/T Net that Results from Rewriting Parallel Composition



$$
\begin{array}{lll}
\rightsquigarrow & \ldots & c_6[c \in_t X] : \bar{c}_6[\neg(c \in_t X)] : \\
 & & \quad {}^{\{r,\bar{c}_6\}}c^{\{s\}} \quad \| \| \; c_7[c \doteq a] : {}^{\{r,c_1,c_6,c_7\}}c^{\{p,p_2,s\}} \\
 & & \quad \| \| \; c_8[c \doteq c] : {}^{\{p,r,c_2,c_6,c_8\}}c^{\{q,s\}} \\
 & & \quad \| \| \left(\begin{array}{c} {}^{\{p_2\}}\{q\}P_t \\ |[\{{}^{\{c_1\}}a^{\{p,p_2\}}, {}^{\{p,c_2\}}c^{\{q\}}\}[X]\{{}^{\{c_3\}}b^{\{r,r_2\}}, {}^{\{r,c_6\}}c^{\{s\}}\}]| \\ {}^{\{r_2\}}\{s\}Q_t \end{array} \right)
\end{array}
$$

The result of the rewriting is shown as a P/T net in Fig. 6.1. Some of the created transitions, namely some of those that are guarded with unsatisfiable conditions—provided that a, b, c, X are interpreted as supposed—are left out: The two dotted transitions represent the (wrong) attempt to merge the transitions a and b, and the (wrong) assumption that c is interleaved, and has to propagated directly. The other transitions left out are unnecessary for similar reasons, and can be identified accordingly when evaluating the conditions of the causal state operator. In most cases, names of places are in the diagram replaced with the conditions that determine the respective values.

Transitions rendered using continuous lines are those that actually make up the behaviour of the process. As expected, the events a and b are interleaved (the order of firing is not predetermined), and the synchronised transition c can only fire after a and b.

For the remaining part of the example, we assume that a background procedure (S) exists that is able to compute the values of conditions (according to the standard interpretation of the constant symbols from above), and we simplify process terms by leaving out transitions that belong to impossible cases, and by removing unnecessary

places. After this simplification, the result of the derivation so far is (without the leading state operators)

$$\begin{pmatrix} a\,^{\{p,p_2\}} \;|||\; \left(\,^{\{p\}}c\,^{\{q\}} \;|||\; ^{\{p_2\}\,\{q\}}P_t\right) \\ |[\varnothing[X]\varnothing]| \\ b\,^{\{r,r_2\}} \;|||\; \left(\,^{\{r\}}c\,^{\{s\}} \;|||\; ^{\{r_2\}\,\{s\}}Q_t\right) \end{pmatrix}$$

$$\overset{*}{\rightsquigarrow} \quad a\,^{\{p,p_2\}} \;|||\; b\,^{\{r,r_2\}} \;|||\; ^{\{p,r\}}c\,^{\{q,s\}} \;|||\; \underbrace{\begin{pmatrix} ^{\{p_2\}\,\{q\}}P_t \\ |[\{^{\{p\}}c\,^{\{q\}}\}[X]\{^{\{r\}}c\,^{\{s\}}\}]| \\ ^{\{r_2\}\,\{s\}}Q_t \end{pmatrix}}_{=:R}$$

The three transitions correspond to the transitions of the diagram that are drawn continuously, and represent the initial behaviour of the channel model, which first raises the two events a, b in arbitrary order, and subsequently c. The next part of the derivation treats the hiding of the internal action c:

$$\left(\, a\,^{\{p,p_2\}} \;|||\; b\,^{\{r,r_2\}} \;|||\; ^{\{p,r\}}c\,^{\{q,s\}} \;|||\; R\right) \setminus X$$

The first steps distribute hiding through interleaving; afterwards the hiding operator can be applied to the first transition:

$$\begin{aligned} &\overset{*}{\rightsquigarrow} \quad \underline{\left(\, a\,^{\{p,p_2\}}\right) \setminus X} \;|||\; \left(\, b\,^{\{r,r_2\}}\right) \setminus X \;|||\; \left(\,^{\{p,r\}}c\,^{\{q,s\}}\right) \setminus X \;|||\; R \setminus X \\ &\rightsquigarrow \quad \underline{c_1[a \in_t X] : \left(\,^{\{c_1\}}(\tau(a))\,^{\{p,p_2\}}\right)} \;|||\; \underline{c_1[\neg(a \in_t X)] : \left(\,^{\{c_1\}}a\,^{\{p,p_2\}}\right)} \\ &\qquad\qquad |||\; \left(\, b\,^{\{r,r_2\}}\right) \setminus X \;|||\; \left(\,^{\{p,r\}}c\,^{\{q,s\}}\right) \setminus X \;|||\; R \setminus X \end{aligned}$$

The condition $a \in_t X$ and its negation can at this point be evaluated to ff and tt (resp.), and the first case can be removed. The same steps are performed for the other two transitions, which leaves b unmodified and hides c.

$$\begin{aligned} &\overset{(S)}{\rightsquigarrow} \quad a\,^{\{p,p_2\}} \;|||\; \underline{\left(\, b\,^{\{r,r_2\}}\right) \setminus X} \;|||\; \underline{\left(\,^{\{p,r\}}c\,^{\{q,s\}}\right) \setminus X} \;|||\; R \setminus X \\ &\overset{*}{\rightsquigarrow} \quad a\,^{\{p,p_2\}} \;|||\; b\,^{\{r,r_2\}} \;|||\; ^{\{p,r\}}(\tau(c))\,^{\{q,s\}} \;|||\; R \setminus X \end{aligned}$$

This concludes the rewriting of the channel part of the example. The second part of the term, which represents the actual processes of the system, can be rewritten to a bag of transitions like the first one. For the second step that is shown, the rewriting rules for ordering nested interleaving have to be applied:

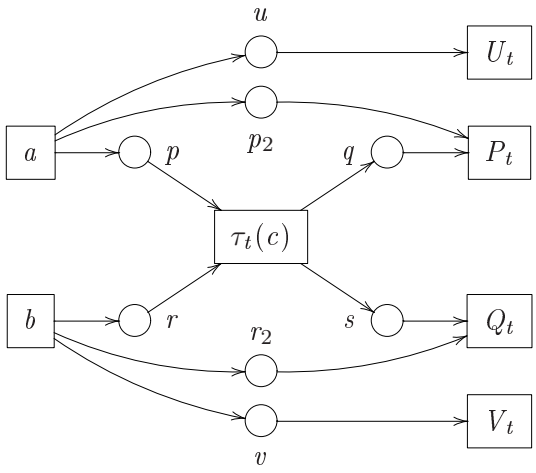
$$\begin{aligned} &\underline{a \to U_t} \;|||\; \underline{b \to V_t} \\ &\quad \overset{*}{\rightsquigarrow} \quad u[\mathit{false}] : \left(\, a\,^{\{u\}} \;|||\; ^{\{u\}}U_t\right) \;|||\; v[\mathit{false}] : \left(\, b\,^{\{v\}} \;|||\; ^{\{v\}}V_t\right) \\ &\quad \overset{*}{\rightsquigarrow} \quad u[\mathit{false}] : v[\mathit{false}] : \left(\, a\,^{\{u\}} \;|||\; b\,^{\{v\}} \;|||\; ^{\{u\}}U_t \;|||\; ^{\{v\}}V_t\right) \end{aligned}$$

We conclude the rewriting by combining the two parts using the operator $[\![\varnothing[\,Y\,]\varnothing]\!]$. The details of this last part of the reduction are skipped, as it can be performed in the same way as for the first parallel operator, and only the result after evaluating conditions is given (state operators are again left out):

$$\begin{pmatrix} a^{\,\{p,p_2\}} \;|||\; b^{\,\{r,r_2\}} \;|||\; {}^{\{p,r\}}(\tau(c))^{\,\{q,s\}} \;|||\; R \setminus X \\ [\![\varnothing[\,Y\,]\varnothing]\!] \\ a^{\,\{u\}} \;|||\; b^{\,\{v\}} \;|||\; {}^{\{u\}}U_t \;|||\; {}^{\{v\}}V_t \end{pmatrix}$$

$$\overset{*}{\rightsquigarrow} \quad a^{\,\{p,p_2,u\}} \;|||\; b^{\,\{r,r_2,v\}} \;|||\; {}^{\{p,r\}}(\tau(c))^{\,\{q,s\}} \;|||\; \begin{pmatrix} R \setminus X \\ [\![\{\,a^{\,\{p,p_2\}},\, b^{\,\{r,r_2\}}\}[\,Y\,]\{\,a^{\,\{u\}},\, b^{\,\{v\}}\}]\!] \\ {}^{\{u\}}U_t \;|||\; {}^{\{v\}}V_t \end{pmatrix}$$

We again illustrate the process term as a P/T net diagram. The diagram differs from the one given above in the renaming of c to $\tau(c)$, and in the addition of the two subprocesses U_t, V_t and their guarding places u, v.



6.6. Correctness of the Rewriting System

While for each group of rewriting rules in Sect. 6.4 we have referred to lemmas that justify the performed modifications, for some issues of correctness it is necessary to examine the rewriting system as a whole. Concerned rules are in particular those treating the duplication of places by using the symbols D^t_{p+q} and C^t_{p+q}, for which it is not completely obvious that premisses of the used lemmas hold. To ensure that this is always the case, we formulate a well-formedness condition on CCSP process terms (which is not only a syntactic condition, however, but also refers to a specific interpretation) that is preserved by the rewriting rules, and that guarantees the correctness of the rules. The necessity of such a condition is caused by the non-atomicity of the duplication of places, i.e. the insertion of symbols D^t_{p+q} and C^t_{p+q}, and the subsequent rewriting

of the term to perform the actual duplication.[3] By distributing D^t_{p+q} and C^t_{p+q} in a wrong way, it is possible to create terms that represent "illegal" intermediate states of the procedure. For instance the term

$$D^t_{p+q}({}^{\{q\}}a_{\,t})$$

is not regarded as admissible: D^t_{p+q} is supposed to introduce a new place q, which already turns up in its argument. We prove that when starting with a term that does not contain D^t_{p+q} or C^t_{p+q}, the application of rewriting rules never leads to situations like the one shown.

We assume for the next paragraphs that a set S of places is fixed as part of the vocabulary, and we denote the marking that assigns each place the value 'empty' with M_{E}:

$$M_{\mathrm{E}} : S \to \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}, \qquad M_{\mathrm{E}}(S) \subseteq \{\mathrm{E}\}$$

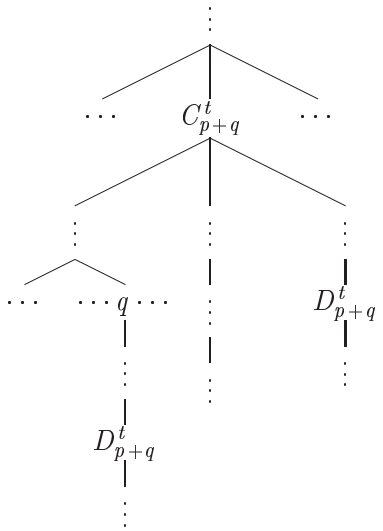
M_{E} represents the weakest possible approximation of the state of a monotonic P/T net (as introduced in Sect. 4.1.1), and we essentially show in the next lemma that all rewriting rules of the system preserve the meaning of a process term up to the congruence relation $\sim_{M_{\mathrm{E}}}$. This means, in particular, that the meaning of closed process terms (in which all places are bound by state operators) is not modified at all by rewriting.

6.6.1 Definition (Well-formed CCSP Process Term): A process term $P \in Proc_S$ is called *well-formed* for a pre-interpretation $(\Sigma_0, \mathrm{val}_{pre})$, if

1. For all pairs $(p, q) \in S \times S$, occurrences of D^t_{p+q} and C^t_{p+q} are admissible:
   - There is at most one occurrence of C^t_{p+q} in P.
   - If C^t_{p+q} turns up in P, then all occurrences of place q are in the scope of C^t_{p+q} (this entails that occurrences of D^t_{p+q} are in the scope of C^t_{p+q}).
   - q does not turn up in the scope of D^t_{p+q} (in particular D^t_{p+q} is not nested).
2. For all occurrences $C^t_{p+q}(Q)$ of a symbol C^t_{p+q} in P, the argument Q satisfies the premiss of Lem. 5.7.6, 1 regarding the marking M_{E}: $T^{fix}_{p+q}(M_{\mathrm{E}} : \mathrm{val}(Q))$. ∗

The first of the two conditions states that the symbols D^t_{p+q} and C^t_{p+q} are the lower and upper boundaries of a part of the term that may (exclusively) contain the place q. As described in Sect. 5.7, the corresponding rewriting rules move the symbol C^t_{p+q} towards the root of the term and D^t_{p+q} towards the leafs, while duplicating occurrences of p as they are met. Rewriting rules that match on D^t_{p+q} have the premiss that q must not turn up in the scope of the symbol, which can be neglected if only well-formed terms are treated.

[3] When using a complex rule that performs the duplication in one step, well-formedness requirements can be left out.



The second condition is a semantic one—though it would also be possible to formulate sufficient syntactic criteria—and prevents the mapping C_{p+q} from being applied to processes that do not adhere to the token production discipline, which is discussed in Sect. 5.7. This discipline is a premiss of the algebraic law

$$p[v] : C_{p+q}(Q) \quad \sim_{M_\mathrm{E}} \quad p[v] : q[v] : Q$$

on which the rewriting rule for eliminating C_{p+q} is based:

$$p[\Phi]_t : C^t_{p+q}(\tilde{P}) \quad \rightsquigarrow \quad p[\Phi]_t : q[\Phi]_t : \tilde{P}$$

Applying the rule to well-formed process terms is thus supported by the algebraic law, whereas for other terms the correctness is not guaranteed.

Terms in which the symbols D^t_{p+q} and C^t_{p+q} do not occur are trivially well-formed. Because we use the two operations as an intermediate representation of the procedure of duplicating places, the requirement that rewriting is only started for terms that do not contain the symbols is not a significant restriction. The next lemma states that in this case the rewriting system does not introduce new violations of the conditions either, and also that the meaning of process terms is preserved when only treating well-formed terms.

6.6.2 Lemma (Rewriting of Well-formed Terms): Suppose that $P \in \mathit{Proc}_S$ is a process term that is well-formed for the pre-interpretation $(\Sigma_0, \mathrm{val}_{pre})$, and that the term $P' \in \mathit{Proc}_S$ arises from the application of a rewriting rule to P: $P \rightsquigarrow P'$. Then the following statements hold:

1. P' is well-formed for $(\Sigma_0, \mathrm{val}_{pre})$.

2. For the pre-interpretation $(\Sigma_0, \mathrm{val}_{pre})$, the term P' denotes the same process as P up to the relation $\sim_{M_\mathrm{E}}$: $\mathrm{val}(P) \sim_{M_\mathrm{E}} \mathrm{val}(P')$. *

6.7. Normal Forms Established by Rewriting

Rule (6.1) is in this section considered to be a special case, as it is the only rule synthesising transitions. The other parts of the rewriting system only perform the task of analysing a given process term, and eventually establish a process term whose shape enables predictions about the LTSs of represented processes. While the following statements also hold in the presence of rule (6.1), they are of particular interest when excluding it from the system, or when pursuing a dedicated strategy that controls the applications of (6.1) (as discussed below).

The rewriting system has the nice property of terminating in any situation, which is entailed by the fact that its rules are consistent with an appropriate lexicographic path ordering [Der87] on the term algebra $Proc_S$. Thus, it is interesting to examine the irreducible process terms that are established by the system.

6.7.1 Lemma (Properties of Rewriting System): Suppose that $P \in Proc_S$ is a process term that does not contain D^t_{p+q} or C^t_{p+q} (for any places $p, q \in S$). The following statements hold:

1. Rewriting of P terminates.

2. If $P' \in Proc_S$ with $P \overset{*}{\rightsquigarrow} P'$ is irreducible, then P' has (up to associativity of $|||_t$) the shape

$$C^t_{p_1+q_1}\Big(\cdots C^t_{p_k+q_k}\Big(r_1[\Phi_1]_t : \cdots r_l[\Phi_l]_t : \Big(T \;|||_t\; R\Big)\Big)\cdots\Big)$$

   where $p_1, \ldots, p_k, q_1, \ldots, q_k, r_1, \ldots, r_l \in S$ are places (which are not necessarily distinct), $\Phi_1, \ldots, \Phi_l \in For$ are formulas and $T, R \in Proc_S$ are process terms. T, R have the properties

   - T only consists of transitions and the interleaving operator (possibly the part T vanishes if there are no transitions at all; for this case we choose $T = stop_t$)
   - R is a remainder of more complex CCSP operators, but does not contain transitions, the causal state operator or symbols D^t_{p+q}, C^t_{p+q}.

3. If in P no transition turns up in the scope of prefixing, then item 2 also holds if rule (6.1) for treating prefixing is removed from the rewriting system. ∗

When comparing irreducible terms with the head normal form of process terms described in Sect. 3.6, then the transition part T can be regarded as the Petri net encoding potential initial steps of a process, and part R determines the continuation of the process behaviour. Further considerations about the relation between T and R are needed, however, because in general T does not really cover the complete initial behaviour. This is only the case if R adheres to certain conditions, which are discussed in the next pages.

6.8. Reasoning about Open Process Terms

Open process terms, i.e. terms that contain places which are not bound by causal state operators, can be regarded as descriptions of processes that are parametrised with markings M of places. Instances of such templates can be derived by assigning values to the free places of the term, e.g. by adding corresponding causal state operators.

6.8.1 Example: The term

$$a \,{}^{\{p\}}_{t}\, |||_{t}\, {}^{\{p\}} b_{\,t}$$

can represent the following processes (written in plain CSP notation), depending on the value that is assigned to the place p:

$$\begin{array}{rcll} \mathrm{val}_{pre}(a) \to \mathrm{val}_{pre}(b) \to stop & \ldots & p \text{ has value E} \\ \mathrm{val}_{pre}(b) \to stop & \ldots & p \text{ has value M} \\ stop & \ldots & p \text{ has value D} \end{array} \qquad *$$

Instead of explicitly adding state operators to assign values to places, in the sequel we assume that process terms are always considered in the context of a *context marking* $M \in \mathcal{M}_S$, which determines the values of free places. To impose the initial place values given by M on a process, we introduce a mapping $init(\cdot,\cdot)$ on processes that performs the necessary assignments. This can be done by using causal state operators, because process terms can only describe processes $P \in CProc_{\Sigma_0,S}$ that contain finitely many free places:

$$init(P,M) \quad := \quad p_1[M(p_1)] : \cdots p_n[M(p_n)] : P \qquad \text{for } FP(P) = \{p_1,\ldots,p_n\}$$

Apart from the definition of $init(\cdot,\cdot)$ referring to the causal state operator, we can also give a direct characterisation using a characteristic equation. The equation can be derived from the definitions of the causal state operator and of I/O addition for P/T net markings (Def. 4.1.3), which essentially are both encodings of the semantics of places in monotonic P/T nets. We first formulate an operational rule for $init(\cdot,\cdot)$, which states that a process $init(P,M)$ can make an execution step $(\varnothing,a,\varnothing)$ whenever P is able to perform a corresponding step that is consistent with the marking M. The effect of the step regarding consumption and production of tokens is retained by altering M appropriately:[4]

$$\frac{P \xrightarrow{(m,a,n)} P' \qquad (m,n) \in \mathrm{En}(M)}{init(P,M) \xrightarrow{(\varnothing,a,\varnothing)} init(P', M + (m,n))}$$

As always, the rule can be translated into an equation by collecting all instances that match on the expression $init(P,M)$: For an LTS label $(m,e,n) \in \mathcal{L}_{CCSP}$, we have

$$init(P,M)\Big\downarrow \begin{smallmatrix}\varnothing\\ e\\ \varnothing\end{smallmatrix}\Big\downarrow \quad = \bigcup_{(m',n')\in \mathrm{En}(M)} init(P\Big\downarrow \begin{smallmatrix}m'\\ e\\ n'\end{smallmatrix}\Big\downarrow, M + (m',n'))$$

[4]Note that this rule is very similar to the SOS rule of Table 5.1 describing the application of approximations to processes. A discussion comparing approximations with the causal state operator, which also applies to $init(\cdot,\cdot)$ that is derived from the latter, can be found in Sect. 5.4.

For $m \cup n \neq \varnothing$:

$$init(P, M)\Big\downarrow \begin{matrix} m \\ e \\ n \end{matrix} \Big\downarrow \quad = \quad \varnothing$$

The union of the first equation is in general infinite and can violate the image finite condition of labelled transition systems (Def. 2.2.1). For this reason, the function $init(\cdot, \cdot)$ is only partial on the set $CProc_{\Sigma_0,S}$ of all processes, which reflects the assumption that only finitely many places must occur free in process P. In this case, the set $P\Big\downarrow \begin{matrix} m' \\ e \\ n' \end{matrix} \Big\downarrow$ is empty for all but finitely many I/O pairs (m', n').

6.8.1. Evaluation of the Prefix of an Irreducible Process Term

According to Lem. 6.7.1, in an irreducible process term a prefix

$$C^t_{p_1+q_1}\Big(\cdots C^t_{p_k+q_k}\Big(r_1[\Phi_1]_t : \cdots r_l[\Phi_l]_t :$$

can be identified. We describe how this prefix can be merged with a context marking M, so that after elimination of the prefix only an (open) process term of the kind $T \parallel_t R$ remains.

Leading symbols $C^t_{p_1+q_1}$ denote the duplication of place p_1 by introducing a copy q_1. This can be translated into an update of the context marking M, namely cell $M(q_1)$ is assigned the value $M(p_1)$. To make this observation, we examine the closed process $init(C_{p_i+p_j}(P), M)$, i.e. the instance of $C_{p_i+p_j}(P)$ that is distinguished by M. By the definition of $init(\cdot, \cdot)$ and Lem. 5.7.6, 1, and because inner state operators overwrite the values chosen by outer ones, the removal of the leading operation $C_{p_i+p_j}$ can simply be written as (we again assume $FP(P) = \{p_1, \ldots, p_n\}$)

$$\begin{aligned}
&p_1[M(p_1)] : \cdots p_n[M(p_n)] : C_{p_i+p_j}(P) \\
&\quad = \quad \cdots \quad = \quad p_1[M(p_1)] : \cdots p_n[M(p_n)] : p_i[M(p_i)] : C_{p_i+p_j}(P) \\
&\quad = \quad \cdots \quad = \quad p_1[M(p_1)] : \cdots p_n[M(p_n)] : p_i[M(p_i)] : p_j[M(p_i)] : P \\
&\quad = \quad \cdots \quad = \quad p_1[M(p_1)] : \cdots p_j[M(p_i)] : \cdots p_n[M(p_n)] : P
\end{aligned}$$

The same statement can be given using the mapping $init(\cdot, \cdot)$ as

$$init(C_{p_i+p_j}(P), M) \quad = \quad init(P, M^{M(p_i)}_{p_j}) \qquad \left(M^{M(p_i)}_{p_j}\right)(p) = \begin{cases} M(p_i) & p = p_j \\ M(p) & \text{otherwise} \end{cases}$$

Though the whole transformation is performed semantically, no requirements have been imposed on the underlying interpretation. Thus, for arbitrary interpretations it is possible to remove the symbol $C^t_{p_i+p_j}$ from a term while modifying an initial marking accordingly:

$$init(\mathrm{val}(C^t_{p_i+p_j}(P)), M) \quad = \quad init(\mathrm{val}(P), M^{M(p_i)}_{p_j})$$

In a similar way, leading occurrences $r_i[\Phi_i]_t : \cdot$ of the causal state operator in a process term can be translated into an update of cell $M(r_i)$ of a context marking M.

The new value of r_i depends on a pre-interpretation $(\Sigma_0, \mathrm{val}_{pre})$ that determines the value of Φ_i, so in general a case distinction has to be made:

$$init(\mathrm{val}(r_i[\Phi_i]_t : P), M) \quad = \quad init(\mathrm{val}(P), M^{v}_{r_i}) \qquad v = \begin{cases} \mathrm{E} & \mathrm{val}_{pre}(\Phi_i) = \mathrm{ff} \\ \mathrm{M} & \mathrm{val}_{pre}(\Phi_i) = \mathrm{tt} \end{cases}$$

Using the proposition of Lem. 6.7.1 together with the given treatment of leading modifiers for markings, the result of rewriting a process term P is a term of the kind $P' = T \;|||_t\; R$, together with a marking M that determines initial values of the places that turn up in P' (because of case distinctions, in general a number of markings M are obtained, whereby each one is equipped with a set of premisses/formulas).

6.8.2 Example: We rewrite the process term $a_{\,t}\, |[\, \varnothing[X]\varnothing \,]|_t\, (b\, {}^{\{q\}}_{t}\; |||_t\; {}^{\{q\}}(c \to_t P))$ without using the rule for the reduction of prefixing, which yields

$$p_1[a \in_t X]_t : p_2[\neg(a \in_t X)]_t : p_3[b \in_t X]_t : p_4[\neg(b \in_t X)]_t : p_5[a \doteq b]_t :$$
$$^{\{p_2\}}a_{\,t}\; |||_t\; ^{\{p_4\}}b\, ^{\{q\}}_{t}\; |||_t\; ^{\{p_1,p_3,p_5\}}b\, ^{\{q\}}_{t}\; |||_t \left(stop_t\; |[\, \{\, ^{\{p_1\}}a\, \}[X]\{\, ^{\{p_3\}}b\, ^{\{q\}}\}\,]|_t\; ^{\{q\}}(c \to_t P)\right)$$

To illustrate the removal of causal state operators, we start with the initial marking M_0 with $M_0(q) = \mathrm{E}$. The results of performing the steps as described above are given in the following table. Cases with inconsistent premisses have been removed. Because only the values of the places $q, p_1, \ldots, p_5$ are important (and determined), we write markings M as tuples $(M(q), M(p_1), \ldots, M(p_5))$.[5]

	$\mathrm{val}_{pre}(a \in_t X)$	$\mathrm{val}_{pre}(b \in_t X)$	$\mathrm{val}_{pre}(a \doteq b)$	
$M_1 = (\mathrm{E},\mathrm{E},\mathrm{M},\mathrm{E},\mathrm{M},\mathrm{E})$	ff	ff	ff	
$M_2 = (\mathrm{E},\mathrm{E},\mathrm{M},\mathrm{E},\mathrm{M},\mathrm{M})$	ff	ff	tt	
$M_3 = (\mathrm{E},\mathrm{M},\mathrm{E},\mathrm{E},\mathrm{M},\mathrm{E})$	tt	ff	ff	
$M_4 = (\mathrm{E},\mathrm{E},\mathrm{M},\mathrm{M},\mathrm{E},\mathrm{E})$	ff	tt	ff	
$M_5 = (\mathrm{E},\mathrm{M},\mathrm{E},\mathrm{M},\mathrm{E},\mathrm{E})$	tt	tt	ff	
$M_6 = (\mathrm{E},\mathrm{M},\mathrm{E},\mathrm{M},\mathrm{E},\mathrm{M})$	tt	tt	tt	$*$

6.8.2. Syntactic Analysis of Irreducible Process Terms

Blocked Process Terms

In Example 6.4.1, the relation between process terms consisting of transitions and the corresponding monotonic P/T nets is discussed. For a term of the shape $P' = T \;|||_t\; R$, the question arises in which cases the transitions of part T can be used to make propositions about the behaviour of the complete process represented. Intuitively, the initial behaviour of P' is solely determined by T if R is *not* able to perform a step, i.e. if it is blocked for the time being. For instance this is the case for

$$a\, {}^{\{p\}}_{t}\; |||_t\; {}^{\{p\}}_{t}(b \to_t P) \qquad M(p) = \mathrm{E}$$

[5] The first two rows could be combined, as the value of $a \doteq b$, which determines whether the two events can be synchronised, does not have any effect for interleaving.

where for the given context marking M the left transition is enabled, whereas the right part cannot engage in any event that is consistent with M (the event $(\{p\}, b, \varnothing)$ cannot be raised, because this would require the value 'marked' of place p). To make this more formal, we examine the behaviour of processes represented by $T \;|||_t\; R$ for a given initial marking M.

For an arbitrary interpretation val, we can determine the behaviour of the process $\mathit{init}(\mathrm{val}(P'), M) = \mathit{init}(\mathrm{val}(T \;|||_t\; R), M)$ by employing the characteristic equation of $\mathit{init}(\cdot,\cdot)$:

$$
\begin{aligned}
&\mathit{init}(\mathrm{val}(T \;|||_t\; R), M)\Big\downarrow \begin{smallmatrix}\varnothing\\ e\\ \varnothing\end{smallmatrix}\Big\downarrow \\
&\quad= \bigcup_{(m',n')\in \mathrm{En}(M)} \mathit{init}(\mathrm{val}(T \;|||_t\; R)\Big\downarrow \begin{smallmatrix}m'\\ e\\ n'\end{smallmatrix}\Big\downarrow, M + (m', n')) \\
&\quad= \bigcup_{(m',n')\in \mathrm{En}(M)} \mathit{init}(\mathrm{val}(T)\Big\downarrow \begin{smallmatrix}m'\\ e\\ n'\end{smallmatrix}\Big\downarrow \;|||\; \mathrm{val}(R) \cup \mathrm{val}(T) \;|||\; \underbrace{\mathrm{val}(R)\Big\downarrow \begin{smallmatrix}m'\\ e\\ n'\end{smallmatrix}\Big\downarrow}_{=:U}, M + (m', n'))
\end{aligned}
$$

We are interested in those situations in which only T is important for the initial behaviour of the process; the derivation entails that this is the case for $U = \varnothing$. This is translated to the following equation, which holds whenever the process $\mathrm{val}(R)$ is unable to raise event e for the place values given by M. Eq. (6.2) is a semantic argument, but we are able to formulate a sufficient syntactic criterion.

$$
\bigcup_{(m',n')\in \mathrm{En}(M)} \mathrm{val}(R)\Big\downarrow \begin{smallmatrix}m'\\ e\\ n'\end{smallmatrix}\Big\downarrow \quad = \quad \varnothing \tag{6.2}
$$

To derive a statement on the process term R that entails Eq. (6.2) regardless of the chosen interpretation val, in particular the CCSP operators for token consumption can be used to estimate the behaviour of processes $\mathrm{val}(R)$. Because places are regarded as a syntactic concept, token consumption is also a requirement that can be checked syntactically. Given a context marking M, a process that has to consume tokens from a set o of places can only make a computation step if M assigns all places of o the value 'marked'.

In the definition of the predicate $\mathit{blocked}_M$ below, which realises an approximation for detecting process terms that describe blocked processes, we leave out the causal state operator and the operators C^t_{p+q}, D^t_{p+q}, because after applying the rewriting system to a process term these symbols do no longer turn up. The predicate essentially detects processes that are blocked because of token consumption, and describes how this information can be propagated.

6.8.3 Definition (Blocked Process Term): Suppose that $M \in \mathcal{M}_S$ is a marking, $P, Q \in \mathit{Proc}_S$ are process terms, $a \in \mathit{EventVis} \cup \mathit{EventInt}$ is an event term, $l \in \mathit{EventVis}$ is a visible event term, $X \in \mathit{SetOfEventVis}$ is an event set term, $o, r \in \mathcal{P}_{\mathrm{fi}} S$ are finite sets of places and $L, R \in \mathrm{mult}_{\mathrm{fi}}(\mathcal{P}_{\mathrm{fi}} S \times (\mathit{EventVis} \cup \mathit{EventInt}) \times \mathcal{P}_{\mathrm{fi}} S)$ are buffer multisets.

The predicate $\mathit{blocked}_M$ on the set Proc_S of CCSP process terms is defined inductively

as the strongest predicate with

$$\begin{array}{rcl} & & \mathit{blocked}_M(\mathit{stop}_t) \\ (o,r) \notin \mathrm{En}(M) & \Longrightarrow & \mathit{blocked}_M({}^{o}a\,{}^{r}_{t}) \\ (o,\varnothing) \notin \mathrm{En}(M) \vee \mathit{blocked}_M(P) & \Longrightarrow & \mathit{blocked}_M({}^{o}_{t}P),\ \mathit{blocked}_M({}^{o}{}^{*}_{t}P) \\ \mathit{blocked}_M(P) & \Longrightarrow & \mathit{blocked}_M(l :_t\ P),\ \mathit{blocked}_M(P \setminus_t X) \\ \mathit{blocked}_M(P) \wedge \mathit{blocked}_M(Q) & \Longrightarrow & \begin{array}{l}\mathit{blocked}_M(P \,\Box_t\, Q),\ \mathit{blocked}_M(P \,|||_t\, Q), \\ \mathit{blocked}_M(P \,|[\, L[X]R \,]|_t\, Q)\end{array} \end{array} \quad *$$

Processes that are syntactically blocked cannot engage in any event, provided that the context marking M determines the initial values of places:

6.8.4 Lemma (Blocked Process Term): Suppose that $M \in \mathcal{M}_S$ is a marking, $P \in \mathit{Proc}_S$ is a process term and $(\Sigma_0, \mathrm{val}_{pre})$ is a pre-interpretation.

The process $\mathrm{val}(P)$ is semantically blocked for the marking M, if the process term P is syntactically blocked (the converse does not hold)

$$\mathit{blocked}_M(P) \quad \Longrightarrow \quad \mathit{init}(\mathrm{val}(P), M) = \mathit{stop} \qquad (6.3)$$

*

The right side of Impl. (6.3) is equivalent to the statement that Eq. (6.2) holds for all events $e \in \Sigma \cup \Sigma_\tau$, which can directly be derived from the characteristic equation of the operation $\mathit{init}(\cdot,\cdot)$.

6.8.5 Example (Example 6.8.2 continued): The irreducible subterm of the running example

$$\mathit{stop}_t \,|[\, \{{}^{\{p_1\}}a\,\}[X]\{{}^{\{p_3\}}b\,{}^{\{q\}}\}\,]|_t\, {}^{\{q\}}(c \rightarrow_t P),$$

which remains after rewriting the original process term, is syntactically blocked for each of the markings given in Example 6.8.2. Because of $M_i(q) = \mathrm{E}$ for $i \in \{0,\ldots,6\}$, we have $\mathit{blocked}_{M_i}({}^{\{q\}}(c \rightarrow_t P))$ (processes represented by the subterm have to consume a token from place q, which is empty). Together with $\mathit{blocked}_{M_i}(\mathit{stop}_t)$ then also the complete term is blocked. *

Initial Execution Steps of a Process

If the result of rewriting is a term $P' = T \,|||_t\, R$ (up to leading operators that alter the marking, as discussed above), in which the subterm R can syntactically identified as blocked, then possible first execution steps of processes represented by P' are determined by T:

$$\mathit{init}(\mathrm{val}(T \,|||_t\, R), M)\Big\downarrow \begin{smallmatrix}\varnothing\\ e\\ \varnothing\end{smallmatrix}\Big\downarrow \quad = \bigcup_{(m',n')\in\mathrm{En}(M)} \mathit{init}(\mathrm{val}(T)\Big\downarrow \begin{smallmatrix}m'\\ e\\ n'\end{smallmatrix}\Big\downarrow \,|||\, \mathrm{val}(R), M + (m',n'))$$

T only consists of transitions and interleaving, and the behaviour of the process $\mathrm{val}(T)$ thus essentially coincides with the behaviour of a P/T net that contains the same transitions. If we have $T = {}^{o_1}a_1\,{}^{r_1}_{t} \,|||_t \cdots |||_t\, {}^{o_k}a_k\,{}^{r_k}_{t}$, and if the transitions are duplicable,

i.e. have the property $o_i \ominus r_i \neq \emptyset$, then firing of a transition can be represented as a modification of the marking (which corresponds to firing of transitions in P/T nets). The relation (*) in the derivation follows from Lem. 5.4.3, 6 and Lem. 4.1.6, 6.

$$
\begin{aligned}
\mathrm{val}(T)\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow \quad &= \quad \mathrm{val}({}^{o_1}a_1{}^{r_1}_{t} \;|||_t \cdots |||_t\; {}^{o_k}a_k{}^{r_k}_{t})\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow \\
&= \quad \Big({}^{o_1}(\mathrm{val}_{pre}(a_1))^{r_1} \;|||\cdots|||\; {}^{o_k}(\mathrm{val}_{pre}(a_k))^{r_k} \Big)\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow \\
&= \quad \bigcup_{i=1,\ldots,k} \Big({}^{o_i}(\mathrm{val}_{pre}(a_i))^{r_i}\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow \;|||\; |||_{\substack{j=1,\ldots,k \\ i\neq j}} {}^{o_j}(\mathrm{val}_{pre}(a_j))^{r_j} \Big) \\
&= \quad \bigcup_{\substack{i=1,\ldots,k \\ (m',n')=(o_i,r_i) \\ e=\mathrm{val}_{pre}(a_i)}} \Big(\{stop\} \;|||\; |||_{\substack{j=1,\ldots,k \\ i\neq j}} {}^{o_j}(\mathrm{val}_{pre}(a_j))^{r_j} \Big) \\
&\overset{(*)}{\sim}_{M+(m',n')} \quad \bigcup_{\substack{i=1,\ldots,k \\ (m',n')=(o_i,r_i) \\ e=\mathrm{val}_{pre}(a_i)}} \Big(\{ {}^{o_i}(\mathrm{val}_{pre}(a_i))^{r_i}\} \;|||\; |||_{\substack{j=1,\ldots,k \\ i\neq j}} {}^{o_j}(\mathrm{val}_{pre}(a_j))^{r_j} \Big) \\
&= \quad \bigcup_{\substack{i=1,\ldots,k \\ (m',n')=(o_i,r_i) \\ e=\mathrm{val}_{pre}(a_i)}} \{\mathrm{val}(T)\}
\end{aligned}
$$

The properties of relation $\sim_{M+(m',n')}$ given in Lem. 5.4.3 are inherited by the function $init(\cdot,\cdot)$, in particular for any marking M we have

$$P \sim_M Q \quad \Longrightarrow \quad init(P,M) = init(Q,M)$$

Because the relation is also a congruence relation regarding interleaving, the pieces can be put together:

$$
\begin{aligned}
init(\mathrm{val}(P'),M)\Big\downarrow \begin{smallmatrix} \emptyset \\ e \\ \emptyset \end{smallmatrix} \Big\downarrow \quad &= \quad init(\mathrm{val}(T \;|||_t\; R),M)\Big\downarrow \begin{smallmatrix} \emptyset \\ e \\ \emptyset \end{smallmatrix} \Big\downarrow \\
&= \quad \bigcup_{(m',n')\in \mathrm{En}(M)} init(\mathrm{val}(T)\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow \;|||\; \mathrm{val}(R), M+(m',n')) \\
&= \quad \bigcup_{(m',n')\in \mathrm{En}(M)} \quad \bigcup_{\substack{i=1,\ldots,k \\ (m',n')=(o_i,r_i) \\ e=\mathrm{val}_{pre}(a_i)}} \{init(\mathrm{val}(T) \;|||\; \mathrm{val}(R), M+(m',n'))\} \\
&= \quad \bigcup_{\substack{i=1,\ldots,k \\ (o_i,r_i)\in \mathrm{En}(M) \\ e=\mathrm{val}_{pre}(a_i)}} \{init(\mathrm{val}(P'), M+(o_i,r_i))\} \qquad (6.4)
\end{aligned}
$$

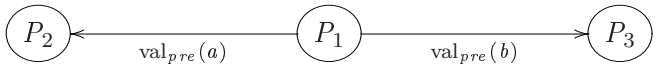
This means that the process described by P' can—as its first step—exactly raise those events that are represented by enabled transitions (regarding the marking M) of subterm T, and that the effect of firing transition ${}^{o_i}a_i{}^{r_i}_{t}$ is completely captured by updating the context marking M to $M+(o_i,r_i)$. It is not necessary to alter the process term P' itself to perform an execution step (but possibly parts of the term become irrelevant

and can be removed, for instance transitions that are not potentially enabled for the marking $M + (o_i, r_i)$, and that will never become enabled in reachable markings M' with $M + (o_i, r_i) \leqslant M'$ thus).

6.8.6 Example (Example 6.8.2 continued): For the first marking $M_1 = (\mathrm{E}, \mathrm{E}, \mathrm{M}, \mathrm{E}, \mathrm{M}, \mathrm{E})$ in the table of the example, we determine the initial steps of processes described by

$$P' \quad := \quad {}^{\{p_2\}}a_{\,t} \;|||_t\; {}^{\{p_4\}}b\,{}^{\{q\}}_{t} \;|||_t\; {}^{\{p_1,p_3,p_5\}}b\,{}^{\{q\}}_{t} \;|||_t\; \Big(stop_t \;|[\,\{{}^{\{p_1\}}a\,\}[X]\{{}^{\{p_3\}}b\,{}^{\{q\}}\}\,]|_t\; {}^{\{q\}}(c \rightarrow_t P)\Big)$$

The second part of the term (that does not contain transitions) is blocked, and all leading transitions are duplicable, hence Eq. (6.4) can be applied. For M_1, two of the transitions are enabled, namely those describing the interleaving case: ${}^{\{p_2\}}a_{\,t}$ and ${}^{\{p_4\}}b\,{}^{\{q\}}_{t}$. Thus, an initial part of (one possible) labelled transition system of processes $init(\mathrm{val}(P'), M_1)$ is:[6]



$$\begin{array}{lcllcl}
P_1 & = & init(\mathrm{val}(P'), M_1) & & = & init(\mathrm{val}(P'), (\mathrm{E}, \mathrm{E}, \mathrm{M}, \mathrm{E}, \mathrm{M}, \mathrm{E})) \\
P_2 & = & init(\mathrm{val}(P'), M_1 + (\{p_2\}, \varnothing)) & & = & init(\mathrm{val}(P'), (\mathrm{E}, \mathrm{E}, \mathrm{D}, \mathrm{E}, \mathrm{M}, \mathrm{E})) \\
P_3 & = & init(\mathrm{val}(P'), M_1 + (\{p_4\}, \{q\})) & & = & init(\mathrm{val}(P'), (\mathrm{M}, \mathrm{E}, \mathrm{M}, \mathrm{E}, \mathrm{D}, \mathrm{M})) \qquad *
\end{array}$$

In the reasoning from above, we have made the following requirements on the process term $P' = T \;|||_t\; R$:

- The transitions of the first part T of the term are duplicable. For an arbitrary term, we can enforce this assumption by employing Impl. (5.29), which states that it is always possible to guard transitions (or other subterms) without altering the meaning of a term:

$$p \notin FP(P) \quad \Longrightarrow \quad p[\mathrm{M}] : {}^{\{p\}}P = P$$

- The second part R of the process term is blocked, which can be ensured by using the syntactic criterion of Def. 6.8.3. For terms that are not blocked, a simple (and effective) method establishes blocking:

  1. If R is (syntactically) blocked then stop the procedure. Otherwise continue with step 2.
  2. Search for a minimum subterm of R that is not blocked, and that is not in the scope of a prefixing operator, and decompose such a subterm by separating a leading step (as a transition) from the remaining part using rewriting rule (6.1). The modified version of term R is called R'.

[6] Strictly speaking, the labels of transitions in the net are triples $(\varnothing, \mathrm{val}_{pre}(a), \varnothing)$ and $(\varnothing, \mathrm{val}_{pre}(b), \varnothing)$, but because results of the operation $init(\cdot, \cdot)$ do never contain free places we only show the event part to make the diagram more readable.

3. Apply the restricted rewriting system without rule (6.1) to the complete term $P' = T \;|||_t\; R'$, until the rewriting terminates. Continue with step 1.

The termination of the loop can be shown, but this result in omitted in this report. The simple algorithm makes it possible to explore the behaviour of a process described by a process term stepwise, and to construct an LTS of the process that is an acyclic graph (and in general not a tree). Based on this LTS, modal or temporal properties of a process can be checked (Sect. 6.10).

6.9. Simplification of CCSP Terms

The explicit notion of causality in CCSP makes it very simply to formulate process terms that contain irrelevant parts. A constructed example is a situation like

$$p[\mathit{false}]_t : {}^{\{p\}}_{t}P$$

which is a term that—independent from the chosen interpretation—represents the process *stop*. Moreover, the rewriting system described so far does consistently *not* remove such superfluous parts of terms, but only creates terms in which the corresponding information is encoded in causal relationships. In practice, it is thus necessary to complement the rewriting rules responsible for analysing process terms with rules that analyse the resulting relationships, and that are able to identify parts of a term that can be removed.

Situations that make such "garbage collection" necessary arise from:

1. Case distinctions, which by the rewriting are realised using state operators, and through which (in most cases) after the evaluation of conditions certain parts are disabled. In the example in Sect. 6.5, we have for instance ((S) is a background procedure that evaluates conditions)

$$\begin{array}{ll} \left(a \,{}^{\{p,p_2\}}_{t} \right) \setminus_t X & \\ \quad \rightsquigarrow & c_1[a \in_t X]_t : \left({}^{\{c_1\}}(\tau_t(a)) \,{}^{\{p,p_2\}}_{t} \right) \;|||_t\; c_1[\neg(a \in_t X)]_t : \left({}^{\{c_1\}} a \,{}^{\{p,p_2\}}_{t} \right) \\ \overset{(\mathrm{S})}{\rightsquigarrow} & c_1[\mathit{false}]_t : \left({}^{\{c_1\}}(\tau_t(a)) \,{}^{\{p,p_2\}}_{t} \right) \;|||_t\; c_1[\mathit{true}]_t : \left({}^{\{c_1\}} a \,{}^{\{p,p_2\}}_{t} \right) \end{array}$$

The first part of the obtained term is unnecessary, because its transition is necessarily disabled.

2. The buffers of the parallel composition operator, in which synchronised events are accumulated, can usually be emptied to a certain degree after performing execution steps (by adapting the values of places accordingly). Rewriting can for instance produce the following term:

$$\begin{array}{ll} a \rightarrow_t P \;|[\, \varnothing[\{a\}]\varnothing \,]|_t\; a \rightarrow_t Q & \\ \quad \overset{*}{\rightsquigarrow} & p_1[\mathit{false}]_t : p_2[\mathit{false}]_t : \underline{a \,{}^{\{p_1,p_2\}}_{t}} \;|||_t \left({}^{\{p_1\}}_{t}P \;|[\, \{a \,{}^{\{p_1\}}\}[\{a\}]\{a \,{}^{\{p_2\}}\} \,]|_t \,{}^{\{p_2\}}_{t}Q \right) \end{array}$$

By a syntactic analysis of the result, as in Sect. 6.8.2, we can determine that the only execution step represented processes can perform is the firing of the underlined transition. A term that describes the resulting new state of the process is obtained by adjusting the values of p_1, p_2:

$$p_1[\mathit{true}]_t : p_2[\mathit{true}]_t : a\,{}_t^{\{p_1,p_2\}} \;|||_t \left({}_t^{\{p_1\}}P \;|[\,\{a\,{}^{\{p_1\}}\}[\{a\}]\{a\,{}^{\{p_2\}}\}\,]|_t\; {}_t^{\{p_2\}}Q \right)$$

Both the transition $a\,{}_t^{\{p_1,p_2\}}$ and all elements of the buffers can be eliminated, because it is not possible to produce tokens for the marked places p_1 and p_2.

3. Case distinctions that are already encoded in plain CSP terms, as for instance in

$$\Big(a \to P \,\Box\, b \to Q \Big) \;|[\,\{a,b\}\,]|\; \Big(a \to R \Big)$$

The subterm $b \to Q$ treats a case that does not occur (provided that $a \neq b$), and that can be removed as soon as the situation is recognised when analysing the term.

A systematic approach to perform simplification of CCSP terms is the employment of reachability analysis for Petri nets. Three very simple methods, which in our experience with an implementation of the rewriting system are already sufficient for all situations we have found to occur, are

- Use the current values of places as approximations (Sect. 5.4) to detect removable parts of a term. Because of the congruence properties of the imposition of approximations, this detection can be applied to subterms that are in the scope of arbitrary CCSP operators. For instance, we can derive for the example of item 2 from above:

$$\begin{aligned} &\mathrm{val}\Big(a\,{}_t^{\{p_1,p_2\}} \;|||_t \left({}_t^{\{p_1\}}P \;|[\,\{a\,{}^{\{p_1\}}\}[\{a\}]\{a\,{}^{\{p_2\}}\}\,]|_t\; {}_t^{\{p_2\}}Q \right)\Big) \\ \sim_M \quad &\mathrm{val}\Big(\mathit{stop}_t \;|||_t \left({}_t^{\{p_1\}}P \;|[\,\varnothing[\{a\}]\varnothing\,]|_t\; {}_t^{\{p_2\}}Q \right)\Big) \end{aligned}$$

for an arbitrary marking $M \in \mathcal{M}_S$ with $M(p_1) = M(p_2) = \mathrm{M}$. This relation entails (if P, Q do not contain free places)

$$\begin{aligned} &\mathrm{val}\Big(p_1[\mathit{true}]_t : p_2[\mathit{true}]_t : a\,{}_t^{\{p_1,p_2\}} \;|||_t \left({}_t^{\{p_1\}}P \;|[\,\{a\,{}^{\{p_1\}}\}[\{a\}]\{a\,{}^{\{p_2\}}\}\,]|_t\; {}_t^{\{p_2\}}Q \right)\Big) \\ = \quad &\mathrm{val}\Big(p_1[\mathit{true}]_t : p_2[\mathit{true}]_t : \mathit{stop}_t \;|||_t \left({}_t^{\{p_1\}}P \;|[\,\varnothing[\{a\}]\varnothing\,]|_t\; {}_t^{\{p_2\}}Q \right)\Big) \\ = \quad &\mathrm{val}\Big(p_1[\mathit{true}]_t : p_2[\mathit{true}]_t : {}_t^{\{p_1\}}P \;|[\,\varnothing[\{a\}]\varnothing\,]|_t\; {}_t^{\{p_2\}}Q \Big) \end{aligned}$$

- To support the idea of the first item, it is in certain situations possible to strengthen the value of a place. Namely, if in the situation

$$p[\mathit{false}]_t : P$$

it can be derived that P never produces tokens for place p, then the value 'empty' of p can be increased to 'dead' (this value cannot be expressed by using the

syntactic causal state operator, but can still be used for simplification as in the first item). A syntactic criterion for the absence of token production is that p does not turn up as output place in transitions of P, i.e. that there are no transitions ${}^{o}a\,{}_{t}^{r\cup\{p\}}$ with $p \notin o \cup r$ (besides that, it is also necessary to ensure that p does not turn up as index of symbols $D^t_{p'+q'}$, $C^t_{p'+q'}$).

The criterion applies to the example of item 1, which can hence be recognised as a candidate for removal:

$$c_1[\mathit{false}]_t : \left({}^{\{c_1\}}(\tau_t(a))\,{}_{t}^{\{p,p_2\}} \right) \quad \rightsquigarrow \quad c_1[\mathit{false}]_t : \mathit{stop}_t$$

- It is always possible to remove causal state operators if the concerned place does not turn up in the scope of the operator. Furthermore, in many situations it is possible to remove state operators whose argument place does only occur in a single position within a process term. This simplification is supported by the laws that describe the commutation of the state operator with the other CCSP operators (Lem. 5.2.2), which can be applied to push down state operators, provided that no occurrences of the treated place leave the scope of the operator. For instance we have

$$\begin{aligned} \mathrm{val}\left(p[\mathit{true}]_t : \left({}^{\{p\}}a\,{}_t \;|||_t\; b\,{}_t \right)\right) &= \mathrm{val}\left(\left(p[\mathit{true}]_t : {}^{\{p\}}a\,{}_t\right) |||_t\; b\,{}_t\right) \\ &= \mathrm{val}\left(a\,{}_t \;|||_t\; b\,{}_t \right) \end{aligned}$$

6.10. Calculi for CCSP and Hennessy-Milner-Logic Using Rewriting

In the same way as it is done in Sect. 3, temporal correctness assertions can be used to state that processes—in this section described by using CCSP terms—adhere to certain modal propositions expressed as HML formulas. Though it is also possible to directly use assertions like

$$P : \Phi,$$

in order to exploit the features of open CCSP process terms, it is beneficial to modify the shape of the statements. Because terms that contain free places can be used to encode whole sets of processes (by considering several values of places), calculi for CCSP can be made more efficient if the correctness assertions also represent markings explicitly.

For the next pages, we assume that a set S of places is fixed.

6.10.1. Direct Evaluation of Hennessy-Milner-Logic Formulas

A first (and not particularly efficient) approach for a calculus more or less adapts the rewriting calculus of Sect. 3.4 to CCSP terms. Therefore, we consider correctness assertions of the kind

$$P : M : \Phi,$$

in which $P \in \mathit{Proc}_S$ is a CCSP process term, $M \in \mathcal{M}_S$ is a marking and $\Phi \in \mathit{For}_{\mathrm{HML}}$ is a HML formula. Because the term P is allowed to contain free places, represented processes $\mathrm{val}(P)$ can as in Sect. 6.8 be regarded as templates that describe sets of concrete processes. M is used as a context marking selecting a particular instance $\mathit{init}(\mathrm{val}(P), M)$ of such a set. Given a pre-interpretation $(\Sigma_0, \mathrm{val}_{pre})$, and provided that the evaluation of HML formulas is performed as in Sect. 3.2 by a binary function $\mathrm{val}_{\mathrm{HML}}$, the meaning of a ternary correctness assertion is defined to be

$$\mathrm{val}(P : M : \Phi) \quad := \quad \mathrm{val}_{\mathrm{HML}}(cc^{-1}_{\Sigma_0,S}(\mathit{init}(\mathrm{val}(P), M)), \Phi) \tag{6.5}$$

The part $\mathit{init}(\mathrm{val}(P), M)$ of this definition denotes the process that is described by term P and the initial marking M (the value $\mathrm{val}(P) \in \mathit{CProc}_{\Sigma_0,S}$ of a term P is defined in Sect. 6.3, and is uniquely determined by the pre-interpretation $(\Sigma_0, \mathrm{val}_{pre})$). Because the process $\mathit{init}(\mathrm{val}(P), M)$ does not contain free places, we can observe that the value of the expression $cc^{-1}_{\Sigma_0,S}(\mathit{init}(\mathrm{val}(P), M))$ is a well-defined equivalent process of the CSP process space, i.e. a process with label format $\mathcal{L}_{CSP} = \Sigma \cup \Sigma_\tau$. For this process, the HML formula Φ can be evaluated as in Sect. 3.2.

In Sect. 3.2, the domain of expressions denoting events in HML formulas—as argument of the modal operator $\langle a \rangle \Phi$—is left unspecified. If HML formulas are used in combination with CCSP process terms (the syntax of which is defined in Sect. 6.2), then it is reasonable to employ the same event expressions for formulas as within process terms. Accordingly, the syntax of HML formulas can in this setting be chosen as

$$\begin{aligned} \mathit{For}_{\mathrm{HML}} \quad ::= \quad & \mathit{true} \mid \neg \mathit{For}_{\mathrm{HML}} \mid \mathit{For}_{\mathrm{HML}} \wedge \mathit{For}_{\mathrm{HML}} \\ & \mid \langle \mathit{EventVis} \rangle \mathit{For}_{\mathrm{HML}} \mid \langle \mathit{EventInt} \rangle \mathit{For}_{\mathrm{HML}} \end{aligned}$$

i.e. events are denoted with elements of the set $\mathit{EventVis} \cup \mathit{EventInt}$. The evaluation mapping $\mathrm{val}_{\mathrm{HML}}$ for HML formulas can analogously be defined referring to the pre-interpretation $(\Sigma_0, \mathrm{val}_{pre})$, which in particular means (only the case that differs from Sect. 3.2 is shown; we use the notation $\alpha_{CSP} := \mathit{Dynamics}(\mathrm{FinLTS}_{\mathcal{L}_{CSP}})$ as in Sect. 2.2.2):

$$\mathrm{val}_{\mathrm{HML}}(P, \langle a \rangle \Phi) \quad = \quad \begin{cases} \mathrm{tt} & \exists\, Q \in \alpha_{CSP}(P)(\mathrm{val}_{pre}(a)).\ \mathrm{val}_{\mathrm{HML}}(Q, \Phi) = \mathrm{tt} \\ \mathrm{ff} & \text{otherwise} \end{cases}$$

It is not completely obvious that an assertion $P : M : \Phi$ can be effectively represented at all, because formally M is a mapping of a countably infinite set S. Actually, the value of M is irrelevant for all but finitely many places $p \in S$, however, because the term P only contains finitely many places, and because $\mathit{init}(\mathrm{val}(P), M)$ does not depend on values $M(p)$ of places p that do not turn up in P.

6.10.1 Example: We consider the CCSP process term

$$P \quad = \quad a \,{}^{\{p\}}_{t}\, |||_t \,{}^{\{p\}}\, b_{\,t}$$

from Example 6.8.1. Because P only contains place p, the relevant information of markings $M \in \mathcal{M}_S$ can be described as a one-tuple $(M(p))$. Correctness assertions

about P and different markings are

$$a\,{}^{\{p\}}_{t}\;|||_t\;{}^{\{p\}}b_{\,t} : (\mathrm{E}) : \langle a\rangle\langle b\rangle true$$
$$a\,{}^{\{p\}}_{t}\;|||_t\;{}^{\{p\}}b_{\,t} : (\mathrm{M}) : \langle b\rangle true$$
$$a\,{}^{\{p\}}_{t}\;|||_t\;{}^{\{p\}}b_{\,t} : (\mathrm{D}) : \langle a\rangle true$$

The evaluation of P for the different initial markings yields the processes

$$\begin{aligned} init(\mathrm{val}(a\,{}^{\{p\}}_{t}\;|||_t\;{}^{\{p\}}b_{\,t}), (\mathrm{E})) &= \mathrm{val}_{pre}(a) \rightarrow \mathrm{val}_{pre}(b) \rightarrow stop \\ init(\mathrm{val}(a\,{}^{\{p\}}_{t}\;|||_t\;{}^{\{p\}}b_{\,t}), (\mathrm{M})) &= \mathrm{val}_{pre}(b) \rightarrow stop \\ init(\mathrm{val}(a\,{}^{\{p\}}_{t}\;|||_t\;{}^{\{p\}}b_{\,t}), (\mathrm{D})) &= stop \end{aligned}$$

and the complete correctness assertions can be evaluated as in Sect. 3.2 by considering the HML formula. The transformation (*) below (application of $cc^{-1}_{\Sigma_0,S}$) replaces the CCSP process with a corresponding CSP process; this does not cause any visible modification of the process term, because the prefixing and *stop* operators of CSP and CCSP are called alike.

$$\begin{aligned} &\mathrm{val}(P : (\mathrm{E}) : \langle a\rangle\langle b\rangle true) \\ &\quad = \quad \mathrm{val}_{\mathrm{HML}}(cc^{-1}_{\Sigma_0,S}(\mathrm{val}_{pre}(a) \rightarrow \mathrm{val}_{pre}(b) \rightarrow stop),\ \langle a\rangle\langle b\rangle true) \\ &\quad \overset{(*)}{=} \quad \mathrm{val}_{\mathrm{HML}}(\mathrm{val}_{pre}(a) \rightarrow \mathrm{val}_{pre}(b) \rightarrow stop,\ \langle a\rangle\langle b\rangle true) \\ &\quad = \quad \mathrm{val}_{\mathrm{HML}}(\mathrm{val}_{pre}(b) \rightarrow stop,\ \langle b\rangle true) \quad = \quad \mathrm{val}_{\mathrm{HML}}(stop,\ true) \quad = \quad \mathrm{tt} \\ &\mathrm{val}(P : (\mathrm{M}) : \langle b\rangle true) \quad = \quad \mathrm{tt} \\ &\mathrm{val}(P : (\mathrm{D}) : \langle a\rangle true) \quad = \quad \mathrm{ff} \end{aligned}$$

*

Evaluation Using a Sequent Calculus

By using the rewriting system for CCSP terms of Sect. 6.4, a sequent calculus for correctness assertions can be constructed with little effort and essentially in the same way as in Sect. 3.4. The rules of the calculus are:

- All rewriting rules of Sect. 6.4 for the normalisation of CCSP process terms.
- Propositional rules for HML formulas: Basically, the first six rules of Table 3.1 can be used (of the CSP calculus that is based on transition assertions), it is only necessary to insert the different kind of correctness assertions:

$$\frac{\Gamma \vdash \Delta}{\Gamma,\ P : M : true \vdash \Delta} \qquad \frac{\vdash P : M : true,\ \Delta}{\Gamma \vdash P : M : true,\ \Delta}$$

$$\frac{\Gamma \vdash P : M : \Phi,\ \Delta}{\Gamma,\ P : M : \neg\Phi \vdash \Delta} \qquad \frac{\Gamma,\ P : M : \Phi \vdash \Delta}{\Gamma \vdash P : M : \neg\Phi,\ \Delta}$$

$$\frac{\Gamma,\ P : M : \Phi_1,\ P : M : \Phi_2 \vdash \Delta}{\Gamma,\ P : M : \Phi_1 \wedge \Phi_2 \vdash \Delta} \qquad \frac{\Gamma \vdash P : M : \Phi_1,\ \Delta \qquad \Gamma \vdash P : M : \Phi_2,\ \Delta}{\Gamma \vdash P : M : \Phi_1 \wedge \Phi_2,\ \Delta}$$

- Rules for updating the context marking according to the prefix of a process term. These rules directly reflect the identities given in Sect. 6.8.1 (only the rules for the antecedent are shown):

$$\frac{\Gamma,\ P : M_q^{M(p)} : \Phi\ \vdash\ \Delta}{\Gamma,\ C^t_{p+q}(P) : M : \Phi\ \vdash\ \Delta}$$

$$\frac{\Gamma,\ \neg\Psi,\ P : M_p^{\mathrm{E}} : \Phi\ \vdash\ \Delta \qquad \Gamma,\ \Psi,\ P : M_p^{\mathrm{M}} : \Phi\ \vdash\ \Delta}{\Gamma,\ \Big(p[\Psi]_t : P\Big) : M : \Phi\ \vdash\ \Delta}$$

- Rules for the modal operator of HML. These rules can be designed referring to Sect. 6.8.2, in which criteria are given for deriving labelled transition systems from process terms. Again, we only show the rules for the antecedent, which must only be applied if the transitions ${}^{o_1}a_1\,{}^{r_1}_{t} \;|||_t \cdots |||_t\; {}^{o_k}a_k\,{}^{r_k}_{t}$ are duplicable, and if the remainder R is blocked: $\mathit{blocked}_M(R)$. In practice, it is also necessary to formulate similar rules for the special case of a missing remainder R.

  The rule employs Eq. (6.4), which states that in this case processes described by the term ${}^{o_1}a_1\,{}^{r_1}_{t} \;|||_t \cdots |||_t\; {}^{o_k}a_k\,{}^{r_k}_{t} \;|||_t\; R$ can exactly perform the execution steps that are determined by the transitions.

$$\frac{\Big\{\Gamma,\ a_i \doteq b,\ \Big({}^{o_1}a_1\,{}^{r_1}_{t} \;|||_t \cdots |||_t\; {}^{o_k}a_k\,{}^{r_k}_{t} \;|||_t\; R\Big) : \Big(M + (o_i, r_i)\Big) : \Phi\ \vdash\ \Delta\Big\}_{\substack{i=1,\ldots,k \\ (o_i,r_i)\in \mathrm{En}(M)}}}{\Gamma,\ \Big({}^{o_1}a_1\,{}^{r_1}_{t} \;|||_t \cdots |||_t\; {}^{o_k}a_k\,{}^{r_k}_{t} \;|||_t\; R\Big) : M : \langle b\rangle\Phi\ \vdash\ \Delta}$$

  To establish duplicable transitions, we add two auxiliary rules to the calculus (for antecedent and succedent). An equivalent solution is a modification of the previous rule, because the necessary places can as well be introduced on-the-fly. (p is a fresh place)

$$\frac{\Gamma,\ P[p[\mathrm{M}]_t : {}^{o\cup\{p\}}a\,{}^{r}_{t}] : M : \Phi\ \vdash\ \Delta}{\Gamma,\ P[{}^{o}a\,{}^{r}_{t}] : M : \Phi\ \vdash\ \Delta} \qquad \frac{\Gamma\ \vdash\ P[p[\mathrm{M}]_t : {}^{o\cup\{p\}}a\,{}^{r}_{t}] : M : \Phi,\ \Delta}{\Gamma\ \vdash\ P[{}^{o}a\,{}^{r}_{t}] : M : \Phi,\ \Delta}$$

6.10.2 Example: We prove the following correctness assertion. As the process term exactly contains the places $p, q \in S$, we represent markings M as pairs $(M(p), M(q))$.

$${}^{\{p\}}a_{\ t} \;|||_t\; {}^{\{q\}}a_{\ t} : (\mathrm{M},\mathrm{M}) : \neg\langle a\rangle\neg\langle a\rangle \mathit{true}$$

A closed proof looks as follows. Conditions $a \doteq a$ are left out in the tree.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\vdash\ {}^{\{p\}}a_{\ t} \;|||_t\; {}^{\{q\}}a_{\ t} : (\mathrm{D},\mathrm{D}) : \mathit{true}}}{\vdash\ {}^{\{p\}}a_{\ t} \;|||_t\; {}^{\{q\}}a_{\ t} : (\mathrm{D},\mathrm{M}) : \langle a\rangle \mathit{true}}}{{}^{\{p\}}a_{\ t} \;|||_t\; {}^{\{q\}}a_{\ t} : (\mathrm{D},\mathrm{M}) : \neg\langle a\rangle \mathit{true}\ \vdash} \qquad \dfrac{\dfrac{\dfrac{*}{\vdash\ {}^{\{p\}}a_{\ t} \;|||_t\; {}^{\{q\}}a_{\ t} : (\mathrm{D},\mathrm{D}) : \mathit{true}}}{\vdash\ {}^{\{p\}}a_{\ t} \;|||_t\; {}^{\{q\}}a_{\ t} : (\mathrm{M},\mathrm{D}) : \langle a\rangle \mathit{true}}}{{}^{\{p\}}a_{\ t} \;|||_t\; {}^{\{q\}}a_{\ t} : (\mathrm{M},\mathrm{D}) : \neg\langle a\rangle \mathit{true}\ \vdash}}{{}^{\{p\}}a_{\ t} \;|||_t\; {}^{\{q\}}a_{\ t} : (\mathrm{M},\mathrm{M}) : \langle a\rangle\neg\langle a\rangle \mathit{true}\ \vdash}}{\vdash\ {}^{\{p\}}a_{\ t} \;|||_t\; {}^{\{q\}}a_{\ t} : (\mathrm{M},\mathrm{M}) : \neg\langle a\rangle\neg\langle a\rangle \mathit{true}}$$

*

6.10.2. Evaluation of Hennessy-Milner-Logic Formulas Using Marking Relations

The proof shown in Example 6.10.2 contains a lot of redundancy: In particular, the process term ${}^{\{p\}}a_{\,t}\,|||_t\,{}^{\{q\}}a_{\,t}$ is not modified at all, but is handled independently in two branches of the tree. Modifications of concerned processes are solely accounted for by adapting markings, i.e. the pair (M, M). Another disadvantage of the shown proof is the splitting in two branches in principle, because both branches in the end lead to the same sequents

$$\vdash\ {}^{\{p\}}a_{\,t}\,|||_t\,{}^{\{q\}}a_{\,t} : (\mathrm{D},\mathrm{D}) : \mathit{true}$$

We roughly sketch a way of of coping with these inefficiencies by handling sets of markings within a single correctness assertion. This makes it possible to replace the beginning of the proof tree of Example 6.10.2 with something like

$$\dfrac{\dfrac{{}^{\{p\}}a_{\,t}\,|||_t\,{}^{\{q\}}a_{\,t} : \{(\mathrm{D},\mathrm{M}),(\mathrm{M},\mathrm{D})\} : \neg\langle a\rangle \mathit{true}\ \vdash}{{}^{\{p\}}a_{\,t}\,|||_t\,{}^{\{q\}}a_{\,t} : (\mathrm{M},\mathrm{M}) : \langle a\rangle\neg\langle a\rangle \mathit{true}\ \vdash}}{\vdash\ {}^{\{p\}}a_{\,t}\,|||_t\,{}^{\{q\}}a_{\,t} : (\mathrm{M},\mathrm{M}) : \neg\langle a\rangle\neg\langle a\rangle \mathit{true}}$$

to express that the property $\neg\langle a\rangle \mathit{true}$ has to be proved for two processes

$$\mathit{init}(\mathrm{val}({}^{\{p\}}a_{\,t}\,|||_t\,{}^{\{q\}}a_{\,t}),(\mathrm{D},\mathrm{M})) \qquad \mathit{init}(\mathrm{val}({}^{\{p\}}a_{\,t}\,|||_t\,{}^{\{q\}}a_{\,t}),(\mathrm{M},\mathrm{D})).$$

A direct representation of the set $\{(\mathrm{D},\mathrm{M}),(\mathrm{M},\mathrm{D})\}$—as shown—has the disadvantage of not handling alternating modal operators well, however, and it is even unclear whether the set notation represents a disjunction or a conjunction of the values of the particular correctness assertions (for our definition of HML, alternating modalities are obtained by nesting the modal operator and negation).

A more powerful way of treating sets of markings is the usage of an extended kind of modal formulas, in which additional modal operators are able to explicitly refer to markings. This attempt leads to an analysis procedure for CSP terms that satisfies all of the requirements that are formulated in Sect. 3.5. The procedure has been implemented as part of the thesis, and has be found to be the by far most powerful basis for verifying JCSP programs that was considered.

We define an extension RHML of HML by

$$\mathit{For}_{\mathrm{RHML}} \quad ::= \quad \mathit{For}_{\mathrm{HML}} \mid \neg\mathit{For}_{\mathrm{RHML}} \mid \mathit{For}_{\mathrm{RHML}} \wedge \mathit{For}_{\mathrm{RHML}} \mid \langle R\rangle \mathit{For}_{\mathrm{RHML}}$$

where $R \subseteq \mathcal{M}_S \times \mathcal{M}_S$ ranges over the set of binary relations of markings. The meaning of a formula $\langle R\rangle\Phi$ is that—starting from a certain current marking M—a marking M' with $(M, M') \in R$ can be reached for which Φ holds. In general, the evaluation of a formula $\Phi \in \mathit{For}_{\mathrm{RHML}}$ can be defined similarly as for HML: A ternary mapping $\mathrm{val}_{\mathrm{RHML}}$ is introduced that yields the truth value of a formula depending on a CCSP process

and a marking. $\mathrm{val}_{\mathrm{RHML}}$ is defined inductively as follows:

$$\begin{aligned}
\mathrm{val}_{\mathrm{RHML}}(P, M, \neg\Phi) &= \begin{cases} \mathrm{ff} & \mathrm{val}_{\mathrm{RHML}}(P, M, \Phi) = \mathrm{tt} \\ \mathrm{tt} & \text{otherwise} \end{cases} \\
\mathrm{val}_{\mathrm{RHML}}(P, M, \Phi_1 \wedge \Phi_2) &= \begin{cases} \mathrm{tt} & \mathrm{val}_{\mathrm{RHML}}(P, M, \Phi_1) = \mathrm{tt} = \mathrm{val}_{\mathrm{RHML}}(P, M, \Phi_2) \\ \mathrm{ff} & \text{otherwise} \end{cases} \\
\mathrm{val}_{\mathrm{RHML}}(P, M, \langle a \rangle \Phi) &= \begin{cases} \mathrm{tt} & \exists\, M' \in M \cdot R.\ \mathrm{val}_{\mathrm{RHML}}(P, M', \Phi) = \mathrm{tt} \\ \mathrm{ff} & \text{otherwise} \end{cases}
\end{aligned}$$

For $\Phi \in \mathit{For}_{\mathrm{HML}}$:

$$\mathrm{val}_{\mathrm{RHML}}(P, M, \Phi) = \mathrm{val}_{\mathrm{HML}}(cc^{-1}_{\Sigma_0, S}(\mathit{init}(P, M)), \Phi)$$

By referring to the function $\mathrm{val}_{\mathrm{RHML}}$, the evaluation of correctness assertions $P : M : \Phi$ as in Sect. 6.10.1 can be performed also for formulas $\Phi \in \mathit{For}_{\mathrm{RHML}}$:

$$\mathrm{val}(P : M : \Phi) := \mathrm{val}_{\mathrm{RHML}}(\mathrm{val}(P), M, \Phi) \tag{6.6}$$

If Φ is in fact a HML formula, i.e. $\Phi \in \mathit{For}_{\mathrm{HML}}$, this definition coincides with Eq. (6.5).

It is again not obvious that formulas $\Phi \in \mathit{For}_{\mathrm{RHML}}$ can effectively be represented. Because a process term $P \in \mathit{Proc}_S$ only contains finitely many places, however, only a finite subset of the set S of places is relevant for the evaluation of the formula. Relations of this kind can be encoded and manipulated efficiently using binary decision diagrams (BDDs, [Bry86]).

6.10.3 Example (Example 6.10.2 continued): For the transitions ${}^{\{p\}}a_{\,t}$, ${}^{\{q\}}a_{\,t}$, we can represent the enabledness conditions and the effect of firing as relations on the set of markings:

$$\begin{aligned}
R_1 &:= \left\{ \big(M, M + (\{p\}, \varnothing)\big) \in \mathcal{M}_S \times \mathcal{M}_S \ :\ (\{p\}, \varnothing) \in \mathrm{En}(M) \right\} \\
R_2 &:= \left\{ \big(M, M + (\{q\}, \varnothing)\big) \in \mathcal{M}_S \times \mathcal{M}_S \ :\ (\{q\}, \varnothing) \in \mathrm{En}(M) \right\} \\
R &:= R_1 \cup R_2
\end{aligned}$$

Using this relations, the proof of Example 6.10.2 can be redone by translating the HML modal operator into the relational modal operator:

$$\dfrac{\dfrac{\vdash {}^{\{p\}}a_{\,t} \,|||_t\, {}^{\{q\}}a_{\,t} : (\mathrm{M}, \mathrm{M}) : \neg\langle R\rangle\neg\langle R\rangle \mathit{true}}{\vdash {}^{\{p\}}a_{\,t} \,|||_t\, {}^{\{q\}}a_{\,t} : (\mathrm{M}, \mathrm{M}) : \neg\langle R\rangle\neg\langle a\rangle \mathit{true}}}{\vdash {}^{\{p\}}a_{\,t} \,|||_t\, {}^{\{q\}}a_{\,t} : (\mathrm{M}, \mathrm{M}) : \neg\langle a\rangle\neg\langle a\rangle \mathit{true}}$$

The value of the resulting formula $\neg\langle R\rangle\neg\langle R\rangle \mathit{true} \in \mathit{For}_{\mathrm{RHML}}$ only depends on the initial marking (M, M); the actual process described by the term ${}^{\{p\}}a_{\,t} \,|||_t\, {}^{\{q\}}a_{\,t}$ is irrelevant. To conclude the proof, it is hence only necessary to evaluate the formula, which is a computable problem and can completely be carried out using BDD operations. ∗

7. Conclusion and Future Work

We have sketched a method for verifying modal properties of concurrent systems that are implemented using the Java language and the JCSP library. To efficiently handle the terms that are used to model such systems, which are essentially terms of the CSP process algebra, we have described a method of rewriting to a certain normal form. The method is based on an extension of CSP, called CCSP, which enables an encoding of the processes being analysed that contains less redundancy, and that preserves more information offered by the original process description, than pure CSP. By not destroying information about the independence of execution steps, state graphs of processes are derived that in general are directed acyclic graphs (instead of trees).

All described concepts have been implemented as part of the thesis, and have in most cases been developed as conclusions drawn from observations that were made with earlier realisations. The implementation of a verification system for JCSP systems can still be regarded as work in progress, but does in the present state support the most basic classes for the composition of processes as well as different kinds of JCSP channels completely. The system has been used to symbolically execute systems with more than 20 JCSP processes running in parallel. When further increasing the number of running processes, we primarily met the problem that at the time the KeY prover has performance problems handling that many sequential Java programs simultaneously.

Issues that are left over at this point, and that need further investigation, are among other things:

- The set of supported JCSP classes needs to be rounded off, i.e. it is necessary to construct and verify models of the remaining classes that are not yet treated. In particular, support for JCSP channels that transmit arbitrary kinds of data (as described in Sect. 2.3) has to be added. The current implementation of the proof system does only provide channels for integer values.

- Optimisation of the calculus described in Sect. 6.10.2 by adding a specialised treatment of internal execution steps. For modal formulas that do not keep track of such invisible steps—which is a common situation—it should be possible to apply techniques for partial order reduction, described for instance in [GvLH+96], to make the verification of propositions more efficient.

- A possibility that has not yet been investigated is the usage of Petri net unfolding techniques [McM92] instead or together with the approach of Sect. 6.10.2.

- Completeness aspects when using more powerful modal logics, for instance the modal μ-calculus, instead of Hennessy-Milner-Logic. For the μ-calculus, it is in particular necessary to add rules for induction, possibly along the lines of a discharge rule as in [DG00].

- Support for techniques for compositional verification, which is also related to the support for induction, as for instance described in [DG99, Fre01].

A. Proofs

A.2. Communicating Sequential Processes

2.2.5 Proof (Page 40): All proofs are given in [Rut96]. For item 2, it is necessary to employ the image finite condition of Def. 2.2.1. It is possible to weaken the condition, however, for instance it would be sufficient to require that the image is bounded by an arbitrary ordinal. □

A.4. Extending CSP with P/T Net Causality

4.1.4 Proof (Page 68): 1. Follows directly from the laws of the lattice $\mathcal{P}_{\text{fi}}S$.

2. Let $p \in S$ be a place. By Cond. (4.4) we have

$$(m_1 \cup m_2)\backslash(n_1 \cup n_2) = m_1\backslash(n_1 \cup n_2) \cup m_2\backslash(n_1 \cup n_2) = m_1\backslash n_1 \cup m_2\backslash n_2$$

and the two sides of the conjecture are equal by extensionality:

$$\begin{aligned}
\big(M + ((m_1, n_1) + (m_2, n_2))\big)(p) &= \begin{cases} \mathrm{D} & p \in (m_1 \cup m_2)\backslash(n_1 \cup n_2) \\ \mathrm{M} & p \in n_1 \cup n_2 \\ M(p) & \text{otherwise} \end{cases} \\
&= \begin{cases} \mathrm{D} & p \in m_1\backslash n_1 \\ \mathrm{D} & p \in m_2\backslash n_2 \\ \mathrm{M} & p \in n_1 \\ \mathrm{M} & p \in n_2 \\ M(p) & \text{otherwise} \end{cases} \\
&= \begin{cases} \mathrm{D} & p \in m_2\backslash n_2 \\ \mathrm{M} & p \in n_2 \\ (M + (m_1, n_1))(p) & \text{otherwise} \end{cases} \\
&= \big((M + (m_1, n_1)) + (m_2, n_2)\big)(p)
\end{aligned}$$

3. The relation on I/O pairs is an ordering: Reflexivity and antisymmetry follow directly from set axioms. To show transitivity: Suppose that $(m_2, n_2) = (m_1, n_1) + (m', n')$, $(m_3, n_3) = (m_2, n_2) + (m'', n'')$. By item 1 we have

$$(m_3, n_3) = (m_1, n_1) + ((m', n') + (m'', n''))$$

because the premisses of this item hold, i.e. the additions of the I/O pairs are defined:

$$\begin{aligned}
(m_1 \cup n_1) \cap (m'' \cup n'') \subseteq (m_2 \cup n_2) \cap (m'' \cup n'') = \varnothing \\
(m' \cup n') \cap (m'' \cup n'') \subseteq (m_2 \cup n_2) \cap (m'' \cup n'') = \varnothing.
\end{aligned}$$

To prove the conjecture, we assume the left side of the implication. Similar to the derivation in item 2 we have $m_1 \subseteq m_2$ and $n_1 \backslash m_1 \subseteq n_2 \backslash m_2$ and hence by Def. 4.1.2

$$M(m_1) \subseteq M(m_2) \subseteq \{\mathrm{M}\}, \qquad M(n_1 \backslash m_1) \subseteq M(n_2 \backslash m_2) \subseteq \{\mathrm{E}\}. \qquad \square$$

4.1.6 Proof (Page 69): 1. Holds because each marking is an approximation of itself.

2. The property is directly inherited; we assume the left side of the implication. By definition there is an approximated marking $M' \geqslant M$ with $(m', n') \in \mathrm{En}(M')$, and thus $(m, n) \in \mathrm{En}(M') \subseteq \mathrm{PotEn}(M)$ by Lem. 4.1.4, 3.

3. '$\Longrightarrow$' Let $M' \geqslant M$ be a marking with $(m, n) \in \mathrm{En}(M')$. Then for $p \in m$ we have $M(p) \leqslant M'(p) = \mathrm{M}$, and for $p \in n \backslash m$ analogously $M(p) \leqslant M'(p) = \mathrm{E}$ by Def. 4.1.2.

   '$\Longleftarrow$' We construct a marking $M' \geqslant M$ by

$$M'(p) \quad := \quad \begin{cases} \mathrm{M} & p \in m \\ \mathrm{E} & p \in n \backslash m \\ M(p) & \text{otherwise} \end{cases}$$

   and have $(m, n) \in \mathrm{En}(M')$ by Def. 4.1.2.

4. We assume the left side of the implication and make a case distinction for $p \in S$; (*) is given by item 3, (**) follows by Def. 4.1.2:

   - For $p \in m$: $M(p) \overset{(*)}{\leqslant} \mathrm{M} \overset{(**)}{\leqslant} (M + (m, n))(p)$
   - For $p \in n \backslash m$: $M(p) \overset{(*)}{=} \mathrm{E} \leqslant (M + (m, n))(p)$
   - Otherwise: $M(p) = (M + (m, n))(p)$

5. Follows from items 4, 6.

6. We assume the left side of the implication and refute the right side of item 3: If there is a $p \in m \backslash n$: $(M + (m, n))(p) = \mathrm{D} \not\leqslant \mathrm{M}$; If there is a $p \in n \backslash m$: $(M + (m, n))(p) = \mathrm{M} \not\leqslant \mathrm{E}$. Thus item 3 entails $(m, n) \notin \mathrm{PotEn}(M + (m, n))$.

7. '$\Longrightarrow$' Follows directly from Def. 4.1.5

   '$\Longleftarrow$' We use item 3 as (*) and make a case distinction for $p \in S$:

   - For $M'(p) = \mathrm{D}$: Necessarily $M(p) \leqslant M'(p)$ holds
   - For $M'(p) = \mathrm{M}$: $(\{p\}, \varnothing) \overset{(*)}{\in} \mathrm{PotEn}(M') \subseteq \mathrm{PotEn}(M) \quad \overset{(*)}{\Longrightarrow} \quad M(p) \leqslant \mathrm{M}$
   - For $M'(p) = \mathrm{E}$: $(\varnothing, \{p\}) \overset{(*)}{\in} \mathrm{PotEn}(M') \subseteq \mathrm{PotEn}(M) \quad \overset{(*)}{\Longrightarrow} \quad M(p) \leqslant \mathrm{E}$

8. We assume the left side of the implication. By Eq. (4.2) and Eq. (4.3) we have

$$(M + (m,n))|_{S\setminus m\cup n} = M|_{S\setminus m\cup n}, \qquad (M' + (m,n))|_{S\setminus m\cup n} = M'|_{S\setminus m\cup n},$$
$$(M + (m,n))|_{m\cup n} = (M' + (m,n))|_{m\cup n}.$$

Because the comparison $M + (m,n) \leqslant M' + (m,n)$ is performed point-wise, the conclusion holds.

9. We assume the left side of the implication and have $(m',n') = (m,n) + (m'',n'')$ by definition, i.e. also $(m'',n'') \leqslant (m',n')$. Item 3 entails $(m'',n'') \in \mathrm{PotEn}(M)$, and with Eq. (4.1) this means $(m'',n'') \in \mathrm{PotEn}(M + (m,n))$. The derivation is concluded by item 4:

$$M + (m,n) \leqslant M + (m,n) + (m'',n'') = M + (m',n')$$

10. We make a case distinction for $p \in S$:

- For $p \in m'\setminus n$: $(M + (m',n))(p) = \mathrm{D} \geqslant (M + (m,n))(p)$
- For $p \in n$: $(M + (m',n))(p) = \mathrm{M} = (M + (m,n))(p)$
- Otherwise: $(M + (m',n))(p) = M(p) = (M + (m,n))(p)$ □

4.4.1 Proof (Page 84): Omitted. □

A.5. Algebraic Properties of CCSP

5.1.3 Proof (Page 87): 1. **Impl.** (5.3)**:** By Eq. (5.1) the existence of a process $Q \in P\left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\right\downarrow$ would imply $m \cup n \subseteq FP(P)$.

Impl. (5.4)**:** Follows directly from Eq. (5.1).

2. The proof is performed by coinduction on $\mathrm{FinLTS}_{\mathcal{L}_{CCSP}}$. The considered equivalence relation is

$$Rel \quad := \quad <\{(p[v] : Q,\ Q)\ :\ Q \in CProc_{\Sigma_0,S},\ p \notin FP(Q)\}>$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be an LTS label, and $Q \in CProc_{\Sigma_0,S}$ be a process with $p \notin FP(Q)$.

- For $p \in m \cup n$: The definition of $p[v] : \cdot$ in Table 4.4 entails

$$(p[v] : Q)\left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\right\downarrow \quad = \quad \varnothing \overset{(5.3)}{=} Q\left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\right\downarrow$$

- For $p \notin m \cup n$: By Impl. (5.4) p is not free in adjacent processes, and this means that adjacent processes satisfy the condition in the definition of R:

$$Q' \in Q\left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\right\downarrow \quad \Longrightarrow \quad p \notin FP(Q') \quad \Longrightarrow \quad p[v] : Q' \sim_{Rel} Q' \qquad (**)$$

(*) holds in the following equation because the other parts of the characteristic equation in Table 4.4 disappear by Impl. (5.3):

$$(p[v] : Q)\left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\right\downarrow \overset{(*)}{=} p[v] : Q\left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\right\downarrow \overset{(**)}{\sim_{Rel}} Q\left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\right\downarrow$$

3. We first show an auxiliary proposition that augments item 2 (item 2 gives the direction from left to right):

$$q \notin FP(Q) \quad \Longleftrightarrow \quad q[\mathrm{D}] : Q = Q \tag{A.1}$$

The direction from right to left is proved by showing $q \notin FP(q[\mathrm{D}] : Q)$, which together with the right side entails $q \notin FP(Q)$. For $q \notin FP(q[\mathrm{D}] : Q)$ we construct a fixed-point c_q of the function I from Def. 5.1.2 that fulfils $q \notin c_q(q[\mathrm{D}] : Q)$; as FP is the smallest fixed-point of I this ensures the conjecture.

- The image of the function $p[v] : \cdot$ is a closed sub-LTS of $\mathrm{FinLTS}_{\mathcal{L}_{CCSP}}$, i.e. processes of the set

  $$D_q \quad := \quad q[\mathrm{D}] : CProc_{\Sigma_0,S}, \qquad q \in S$$

  have no adjacent processes outside of D_q:

  $$\forall (m, e, n) \in \mathcal{L}_{CCSP}.\ (q[\mathrm{D}] : Q)\Big\downarrow^{m}_{e}_{n}\Big\downarrow \subseteq D_q$$

  This holds because of the characteristic equation describing the causal state operator in Table 4.4:

  $$(q[\mathrm{D}] : Q)\Big\downarrow^{m}_{e}_{n}\Big\downarrow \quad = \quad \left\{ \begin{array}{ll} q[\mathrm{D}] : Q\Big\downarrow^{m}_{e}_{n}\Big\downarrow & q \notin m \cup n \\ \varnothing & \text{otherwise} \end{array} \right\} \quad \subseteq \quad D_q \qquad (*)$$

- We define complete sub-lattices of the lattice C of Eq. 5.2, from which those assignments of place sets are removed that assign processes from D_q sets containing q:

  $$C_q \quad := \quad \{c \in C \ : \ \forall\, Q \in D_q.\ q \notin c(Q)\} \quad \subseteq \quad C, \qquad q \in S$$

  Each of the lattices C_q is invariant under I: For an assignment $c \in C_q$ we show $I(c) \in C_q$. $(**)$ in the following derivation uses the characteristic equation for $q[\mathrm{D}] : Q$ in Table 4.4:

  $$\begin{array}{lll} I(c)(q[\mathrm{D}] : Q) & = & \displaystyle\bigcup_{\substack{(m,e,n)\in\mathcal{L}_{CCSP},\\ Q'\in(q[\mathrm{D}]:Q)\downarrow^{m}_{e}_{n}\downarrow}} m \cup n \cup c(Q') \\ & = & \displaystyle\bigcup_{\substack{(m,e,n)\in\mathcal{L}_{CCSP},\\ q\notin m\cup n,\\ Q'\in(q[\mathrm{D}]:Q)\downarrow^{m}_{e}_{n}\downarrow}} m \cup n \cup c(Q') \qquad \text{by } (**) \\ & \not\ni & q \qquad \text{by } (*) \end{array}$$

- For a lattice C_q we have a least fixed-point $c_q := \mathrm{lfp}\, I|_{C_q}$ which is also a fixed-point of I. Because of the definition of C_q the desired property $q \notin c_q(q[\mathrm{D}] : Q)$ holds.

We new proceed to the real proposition of the lemma: Suppose that a place $q \in S \backslash FP(P) \cup \{p\}$ is not not element of the right side of the conjecture. We refer to Eq. (A.1) and show $q \notin FP(p[v] : P)$ by proving $q[\mathrm{D}] : p[v] : P = p[v] : P$ using coinduction on the equivalence relation

$$Rel \quad = \quad < \left\{ (q[\mathrm{D}] : p[u] : Q,\ p[u] : Q) \ : \ \begin{array}{l} Q \in CProc_{\Sigma_0,S},\ q \notin FP(Q) \backslash \{p\}, \\ u \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\} \end{array} \right\} >$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be an LTS label, $Q \in CProc_{\Sigma_0,S}$ be a process with $q \notin FP(Q) \backslash \{p\}$ and $u \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ be a place value.

- For $q \in m \cup n,\ p = q$:

$$(q[\mathrm{D}] : p[u] : Q)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad = \quad \varnothing \quad = \quad (p[u] : Q)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$$

- For $q \in m \cup n,\ p \neq q$: Then even $q \notin FP(Q)$ and thus $(m \cup n) \backslash FP(Q) \neq \varnothing$. For $m' \supseteq m,\ n' \supseteq n$ we then have $Q \Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow = \varnothing$ by Impl. (5.3), which entails (*) in the next equation according to the equation for $p[u] : Q$ in Table 4.4.

$$(q[\mathrm{D}] : p[u] : Q)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad = \quad \varnothing \quad \overset{(*)}{=} \quad (p[u] : Q)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$$

- For $q \notin m \cup n$: (*) in the next derivation follows from the observation that each element of the set $(p[u] : Q)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$ can be written as $p[u'] : Q'$ with $FP(Q') \subseteq FP(Q)$ according to the equation in Table 4.4 and using Impl. (5.4).

$$(q[\mathrm{D}] : p[u] : Q)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad = \quad q[\mathrm{D}] : (p[u] : Q)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \overset{(*)}{\sim}_{Rel} \quad (p[u] : Q)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$$

□

5.2.1 Proof (Page 88): We only prove the statements of item 2, and omit the proofs of propositions about external choice.

Eq. (5.5)**:** By coinduction on the equivalence relation

$$Rel \quad := \quad < \left\{ \begin{array}{l} (P' \,\|[\, L'[X]R' \,]\|\, Q',\ Q' \,\|[\, R'[X]L' \,]\|\, P') \ : \\ P', Q' \in CProc_{\Sigma_0,S},\ L', R' \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP} \end{array} \right\} >$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and P', Q', L', R' as in the definition of *Rel*.

- For $e \in X$:

$$\begin{aligned}
(P' \,\|[\, L'[X]R']|\, Q')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad = \quad & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,\|[\, L'[X]R']|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
& \cup \quad P' \,\|[\, L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R']|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
& \cup \quad P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,\|[\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow]|\, Q' \\
\sim_{Rel} \quad & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,\|[\, R'[X]L']|\, P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \\
& \cup \quad Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,\|[\, R'[X]L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow]|\, P' \\
& \cup \quad Q' \,\|[\, R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow [X]L']|\, P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \\
= \quad & (Q' \,\|[\, R'[X]L']|\, P')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}$$

- For $e \notin X$:

$$\begin{aligned}
& (P' \,\|[\, L'[X]R']|\, Q')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
& = \quad P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \,\|[\, L'[X]R']|\, Q' \quad \cup \quad P' \,\|[\, L'[X]R']|\, Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
& \sim_{Rel} \quad Q' \,\|[\, R'[X]L']|\, P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \cup \quad Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \,\|[\, R'[X]L']|\, P' \\
& = \quad (Q' \,\|[\, R'[X]L']|\, P')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}$$

Eq. (5.6): By coinduction. To improve the readability of the proof we abbreviate $\|[X]|$ with $\|$. The concerned equivalence relation is

$$Rel \quad := \quad <\{((P' \parallel Q') \parallel T',\ P' \parallel (Q' \parallel T')) \ :\ P', Q', T' \in CProc_{\Sigma_0,S}\}>$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and P', Q', T' as in the definition of *Rel*.

- For $e \in X$: Most parts of the characteristic equation of parallelism disappear immediately because of the empty buffers. Regrouping of place sets (*) is possible because of the law of associativity (Lem. 4.1.4, 1) that holds for the addition of place sets.

$$\begin{aligned}
& ((P' \parallel Q') \parallel T')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
& = \quad \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} (P' \parallel Q')\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \parallel T' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
& = \quad \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} \ \bigcup_{\substack{(m_1,n_1)= \\ (m'_1,n'_1)+ \\ (m'_2,n'_2)}} \left(P' \Big\downarrow \begin{smallmatrix} m'_1 \\ e \\ n'_1 \end{smallmatrix} \Big\downarrow \parallel Q' \Big\downarrow \begin{smallmatrix} m'_2 \\ e \\ n'_2 \end{smallmatrix} \Big\downarrow \right) \parallel T' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow
\end{aligned}$$

$$\overset{(*)}{\sim}_{Rel} \bigcup_{\substack{(m,n)=\\(m_1,n_1)+\\(m_2,n_2)}} \bigcup_{\substack{(m_2,n_2)=\\(m_1',n_1')+\\(m_2',n_2')}} P'\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow \parallel \left(Q'\Big\downarrow \begin{smallmatrix} m_1' \\ e \\ n_1' \end{smallmatrix}\Big\downarrow \parallel T'\Big\downarrow \begin{smallmatrix} m_2' \\ e \\ n_2' \end{smallmatrix}\Big\downarrow \right)$$
$$= \quad (P' \parallel (Q' \parallel T'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow$$

- For $e \notin X$:

$$\begin{aligned}
&((P' \parallel Q') \parallel T')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \\
&\quad = \quad (P' \parallel Q')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \parallel T' \ \cup \ (P' \parallel Q') \parallel T'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \\
&\quad = \quad \left(P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \parallel Q'\right) \parallel T' \ \cup \ \left(P' \parallel Q'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow\right) \parallel T' \ \cup \ (P' \parallel Q') \parallel T'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \\
&\sim_{Rel} \quad P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \parallel (Q' \parallel T') \ \cup \ P' \parallel \left(Q'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \parallel T'\right) \ \cup \ P' \parallel \left(Q' \parallel T'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow\right) \\
&\quad = \quad (P' \parallel (Q' \parallel T'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow
\end{aligned}$$

Eq. (5.7): By coinduction on the equivalence relation

$$Rel \quad := \quad < \left\{ \begin{array}{c} (P' \,|[\, L' \uplus \{(o,a,r)\}[X]R' \,]|\, Q',\ P' \,|[\, L'[X]R' \,]|\, Q') \ : \\ P', Q' \in CProc_{\Sigma_0,S},\ L', R' \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP} \end{array} \right\} >$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label and P', Q', L', R' as in the definition of *Rel*.

- For $e \in X$: (*) holds because of $X \not\ni a \neq e \in X$.

$$\begin{aligned}
&(P' \,|[\, L' \uplus \{(o,a,r)\}[X]R' \,]|\, Q')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \\
&= \bigcup_{\substack{(m,n)=\\(m_1,n_1)+\\(m_2,n_2)}} P'\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow \,|[\, L' \uplus \{(o,a,r)\}[X]R' \,]|\, Q'\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow \\
&\quad \cup \ P' \,|[\, (L' \uplus \{(o,a,r)\})\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow [X]R' \,]|\, Q'\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow \\
&\quad \cup \ P'\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow \,|[\, L' \uplus \{(o,a,r)\}[X]R'\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow \,]|\, Q' \\
&\overset{(*)}{=} \bigcup_{\substack{(m,n)=\\(m_1,n_1)+(m_2,n_2)\\(m_1,e,n_1)\in L'}} P' \,|[\, L'\backslash\{(m_1,e,n_1)\} \cup \{(o,a,r)\}[X]R' \,]|\, Q'\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow \\
&\qquad \cup \bigcup_{\substack{(m,n)=\\(m_1,n_1)+\\(m_2,n_2)}} P'\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow \,|[\, L' \uplus \{(o,a,r)\}[X]R' \,]|\, Q'\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow \\
&\qquad\qquad \cup \ P'\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow \,|[\, L' \uplus \{(o,a,r)\}[X]R'\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow \,]|\, Q' \\
&\sim_{Rel} \bigcup_{\substack{(m,n)=\\(m_1,n_1)+(m_2,n_2)\\(m_1,e,n_1)\in L'}} P' \,|[\, L'\backslash\{(m_1,e,n_1)\}[X]R' \,]|\, Q'\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow
\end{aligned}$$

$$
\begin{array}{rcl}
 & \cup & \bigcup_{\substack{(m,n)=\\(m_1,n_1)+\\(m_2,n_2)}} P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; [\![L'[X]R']\!] \; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
 & \cup & P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; [\![L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow]\!] \; Q' \\
= & (P' \; [\![L'[X]R']\!] \; Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{array}
$$

- For $e \notin X$:

$$
\begin{array}{rcl}
 & & (P' \; [\![L' \uplus \{(o,a,r)\}[X]R']\!] \; Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
 & = & P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \; [\![L' \uplus \{(o,a,r)\}[X]R']\!] \; Q' \\
 & & \quad \cup \quad P' \; [\![L' \uplus \{(o,a,r)\}[X]R']\!] \; Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
 & \sim_{Rel} & P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \; [\![L'[X]R']\!] \; Q' \; \cup \; P' \; [\![L'[X]R']\!] \; Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
 & = & (P' \; [\![L'[X]R']\!] \; Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{array}
$$

Eq. (5.8): By coinduction on the equivalence relation

$$
Rel \quad := \quad < \{ (P' \; ||| \; stop, \; P') \; : P' \in CProc_{\Sigma_0,S} \} >
$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label and P' as in the definition of *Rel*.

$$
\begin{array}{rcl}
(P' \; ||| \; stop) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow & = & P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \; ||| \; stop \; \cup \; P' \; ||| \; stop \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad = \quad P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \; ||| \; stop \\
 & \sim_{Rel} & P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{array}
$$

Eq. (5.9): By coinduction on the equivalence relation

$$
Rel \quad := \quad < \left\{ \begin{array}{c} (P' \; [\![L[X]R']\!] \; stop, \; P' \; [\![\varnothing[X]R']\!] \; stop) \; : \\ P' \in CProc_{\Sigma_0,S}, \; R' \in \mathrm{mult}_{\mathrm{fi}} \, \mathcal{L}_{CCSP} \end{array} \right\} >
$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label and P', R' as in the definition of *Rel*.

- For $e \in X$:

$$
\begin{array}{rcl}
(P' \; [\![L[X]R']\!] \; stop) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow & = & \bigcup_{\substack{(m,n)=\\(m_1,n_1)+\\(m_2,n_2)}} P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; [\![L[X]R']\!] \; stop \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
 & & \cup \quad P' \; [\![L \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R']\!] \; stop \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
 & & \cup \quad P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; [\![L[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow]\!] \; stop \\
 & = & \bigcup_{\substack{(m,n)=\\(m_1,n_1)+\\(m_2,n_2)}} P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; [\![L[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow]\!] \; stop
\end{array}
$$

$$\sim_{Rel} \quad \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; [\![\varnothing[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow]\!] \; stop$$

$$= \quad (P' \; [\![\varnothing[X]R']\!] \; stop) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$$

- For $e \notin X$:

$$\begin{aligned}
&(P' \; [\![L[X]R']\!] \; stop) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= \quad P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \; [\![L[X]R']\!] \; stop \;\cup\; P' \; [\![L[X]R']\!] \; stop \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= \quad P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \; [\![L[X]R']\!] \; stop \\
&\sim_{Rel} \quad P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \; [\![\varnothing[X]R']\!] \; stop \\
&= \quad (P' \; [\![\varnothing[X]R']\!] \; stop) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}$$

Eq. (5.9)**:** By coinduction: We show that the two processes have the same adjacent processes. Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label.

- For $e \in X$:

$$\begin{aligned}
&(stop \; [\![X]\!] \; stop) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= \quad \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} \quad stop \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; [\![X]\!] \; stop \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
&\qquad\qquad \cup \quad stop \; [\![\varnothing \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]\varnothing]\!] \; stop \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
&\qquad\qquad \cup \quad stop \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; [\![\varnothing[X]\varnothing \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow]\!] \; stop \\
&= \quad \varnothing \quad = \quad stop \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}$$

- For $e \notin X$:

$$\begin{aligned}
&(stop \; [\![X]\!] \; stop) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= \quad stop \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \; [\![X]\!] \; stop \;\cup\; stop \; [\![X]\!] \; stop \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= \quad \varnothing \quad = \quad stop \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}$$

□

5.2.2 Proof (Page 89): **Eq.** (5.11)**:** By coinduction on the following equivalence relation:

$$Rel \quad := \quad < \left\{ \begin{array}{c} (p[v'] : q[u'] : P', \; q[u'] : p[v'] : P') \; : \\ P' \in CProc_{\Sigma_0,S}, \; v', u' \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\} \end{array} \right\} >$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and P', v', u' as in the definition of *Rel*. The reasoning essentially consists of the application of

the characteristic equation of the state operator. We first introduce abbreviations for the occurrences of the state operator:

$$
\begin{aligned}
C(m',n') \quad &:= \quad (p[v'] : P') \Big\downarrow {}^{m'}_{n'} \Big\downarrow && \text{for } p \notin m' \cup n' \\
&= \quad p[v'] : P' \Big\downarrow {}^{m'}_{n'} \Big\downarrow \quad \cup \\
&\qquad \begin{cases}
p[\mathrm{M}] : P' \Big\downarrow {}^{m'}_{n' \cup \{p\}} \Big\downarrow & v' = \mathrm{E} \\
p[\mathrm{D}] : P' \Big\downarrow {}^{m' \cup \{p\}}_{n'} \Big\downarrow & \\
\qquad \cup\ p[\mathrm{M}] : P' \Big\downarrow {}^{m' \cup \{p\}}_{n' \cup \{p\}} \Big\downarrow & v' = \mathrm{M} \\
\varnothing & v' = \mathrm{D}
\end{cases} \\
D(m',n') \quad &:= \quad (q[u'] : P') \Big\downarrow {}^{m'}_{n'} \Big\downarrow && \text{for } q \notin m' \cup n' \\
&= \quad q[u'] : P' \Big\downarrow {}^{m'}_{n'} \Big\downarrow \quad \cup \\
&\qquad \begin{cases}
q[\mathrm{M}] : P' \Big\downarrow {}^{m'}_{n' \cup \{q\}} \Big\downarrow & u' = \mathrm{E} \\
q[\mathrm{D}] : P' \Big\downarrow {}^{m' \cup \{q\}}_{n'} \Big\downarrow & \\
\qquad \cup\ q[\mathrm{M}] : P' \Big\downarrow {}^{m' \cup \{q\}}_{n' \cup \{q\}} \Big\downarrow & u' = \mathrm{M} \\
\varnothing & u' = \mathrm{D}
\end{cases}
\end{aligned}
$$

For each place value $v'' \in \{\mathrm{E}, \mathrm{M}, \mathrm{D}\}$ we have the auxiliary proposition (*):

$$
\begin{aligned}
p[v''] : D(m',n') \quad &= \quad p[v''] : q[u'] : P' \Big\downarrow {}^{m'}_{n'} \Big\downarrow \quad \cup \\
&\qquad \begin{cases}
p[v''] : q[\mathrm{M}] : P' \Big\downarrow {}^{m'}_{n' \cup \{q\}} \Big\downarrow & u' = \mathrm{E} \\
p[v''] : q[\mathrm{D}] : P' \Big\downarrow {}^{m' \cup \{q\}}_{n'} \Big\downarrow & \\
\qquad \cup\ p[v''] : q[\mathrm{M}] : P' \Big\downarrow {}^{m' \cup \{q\}}_{n' \cup \{q\}} \Big\downarrow & u' = \mathrm{M} \\
\varnothing & u' = \mathrm{D}
\end{cases} \\
&\sim_{Rel} \quad q[u'] : p[v''] : P' \Big\downarrow {}^{m'}_{n'} \Big\downarrow \quad \cup \\
&\qquad \begin{cases}
q[\mathrm{M}] : p[v''] : P' \Big\downarrow {}^{m'}_{n' \cup \{q\}} \Big\downarrow & u' = \mathrm{E} \\
q[\mathrm{D}] : p[v''] : P' \Big\downarrow {}^{m' \cup \{q\}}_{n'} \Big\downarrow & \\
\qquad \cup\ q[\mathrm{M}] : p[v''] : P' \Big\downarrow {}^{m' \cup \{q\}}_{n' \cup \{q\}} \Big\downarrow & u' = \mathrm{M} \\
\varnothing & u' = \mathrm{D}
\end{cases}
\end{aligned}
\qquad (*)
$$

The remainder of the proof is a case distinction on the relations between p, q and the set $m \cup n$:

- For $p \notin m \cup n$, $q \notin m \cup n$: By regrouping the parts of the case distinctions the two occurrences of the state operators can be commuted. Note that (*)

also uses $p \neq q$.

$$
\begin{aligned}
(p[v'] : q[u'] : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad &= \quad p[v'] : D(m,n) \quad \cup \\
&\quad \begin{cases} p[\mathrm{M}] : D(m, n \cup \{p\}) & v' = \mathrm{E} \\ p[\mathrm{D}] : D(m \cup \{p\}, n) & \\ \qquad \cup\ p[\mathrm{M}] : D(m \cup \{p\}, n \cup \{p\}) & v' = \mathrm{M} \\ \varnothing & v' = \mathrm{D} \end{cases} \\
&\overset{(*)}{\sim}_{Rel} \quad q[u'] : C(m,n) \quad \cup \\
&\quad \begin{cases} q[\mathrm{M}] : C(m, n \cup \{q\}) & u' = \mathrm{E} \\ q[\mathrm{D}] : C(m \cup \{q\}, n) & \\ \qquad \cup\ q[\mathrm{M}] : C(m \cup \{q\}, n \cup \{q\}) & u' = \mathrm{M} \\ \varnothing & u' = \mathrm{D} \end{cases} \\
&= \quad (q[u'] : p[v'] : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

- For $p \in m \cup n$, $q \notin m \cup n$:

$$
\begin{aligned}
(p[v'] : q[u'] : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad &= \quad \varnothing \\
&= \quad q[u'] : (p[v'] : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \cup \\
&\quad \begin{cases} q[\mathrm{M}] : (p[v'] : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{q\} \end{smallmatrix} \Big\downarrow & u' = \mathrm{E} \\ q[\mathrm{D}] : (p[v'] : P') \Big\downarrow \begin{smallmatrix} m \cup \{q\} \\ e \\ n \end{smallmatrix} \Big\downarrow & \\ \qquad \cup\ q[\mathrm{M}] : (p[v'] : P') \Big\downarrow \begin{smallmatrix} m \cup \{q\} \\ e \\ n \cup \{q\} \end{smallmatrix} \Big\downarrow & u' = \mathrm{M} \\ \varnothing & u' = \mathrm{D} \end{cases} \\
&= \quad (q[u'] : p[v'] : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

- For $p \notin m \cup n$, $q \in m \cup n$: Symmetric to the last case.
- For $p \in m \cup n$, $q \in m \cup n$:

$$
(p[v'] : q[u'] : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad = \quad \varnothing \quad = \quad (q[u'] : p[v'] : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
$$

Eq. (5.12)**:** By coinduction: We can even show that the adjacent processes of the two sides are equal. Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label.

- For $p \in m \cup n$: Note that this entails $(m, e, n) \neq (\varnothing, a, \varnothing)$.

$$
\begin{aligned}
(p[v] : a \to P) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad &= \quad \varnothing \\
&= \quad \begin{cases} p[v] : P & (m,e,n) = (\varnothing, a, \varnothing) \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad (a \to p[v] : P) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

- For $p \notin m \cup n$: Similar to the first case, most parts of the characteristic equation of $p[v] : \cdot$ disappear immediately.

$$
\begin{aligned}
(p[v] : a \to P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow &= p[v] : (a \to P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow \quad \cup \\
&\quad \begin{cases} p[\mathrm{M}] : (a \to P)\Big\downarrow^{m}_{e}_{n\cup\{p\}}\Big\downarrow & v = \mathrm{E} \\ p[\mathrm{D}] : (a \to P)\Big\downarrow^{m\cup\{p\}}_{e}_{n}\Big\downarrow & \\ \qquad \cup\, p[\mathrm{M}] : (a \to P)\Big\downarrow^{m\cup\{p\}}_{e}_{n\cup\{p\}}\Big\downarrow & v = \mathrm{M} \\ \varnothing & v = \mathrm{D} \end{cases} \\
&= p[v] : (a \to P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow \\
&= p[v] : \left\{ \begin{array}{ll} \{P\} & (m,e,n) = (\varnothing, a, \varnothing) \\ \varnothing & \text{otherwise} \end{array} \right\} \\
&= \begin{cases} \{p[v] : P\} & (m,e,n) = (\varnothing, a, \varnothing) \\ \varnothing & \text{otherwise} \end{cases} \\
&= (a \to p[v] : P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow
\end{aligned}
$$

Eq. (5.15)**:** By coinduction and as for Eq. (5.12). Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label.

- For $p \in m \cup n$: Note that then also $p \in m \backslash o \cup n$.

$$
\begin{aligned}
(p[v] : {}^{o}P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow &= \varnothing = \begin{cases} (p[v] : P)\Big\downarrow^{m\backslash o}_{e}_{n}\Big\downarrow & o \subseteq m \backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= ({}^{o}p[v] : P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow
\end{aligned}
$$

- For $p \notin m \cup n$, $o \subseteq m \backslash n$:

$$
\begin{aligned}
(p[v] : {}^{o}P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow &= p[v] : ({}^{o}P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow \quad \cup \\
&\quad \begin{cases} p[\mathrm{M}] : ({}^{o}P)\Big\downarrow^{m}_{e}_{n\cup\{p\}}\Big\downarrow & v = \mathrm{E} \\ p[\mathrm{D}] : ({}^{o}P)\Big\downarrow^{m\cup\{p\}}_{e}_{n}\Big\downarrow & \\ \qquad \cup\, p[\mathrm{M}] : ({}^{o}P)\Big\downarrow^{m\cup\{p\}}_{e}_{n\cup\{p\}}\Big\downarrow & v = \mathrm{M} \\ \varnothing & v = \mathrm{D} \end{cases} \\
&= p[v] : P\Big\downarrow^{m\backslash o}_{e}_{n}\Big\downarrow \quad \cup \\
&\quad \begin{cases} p[\mathrm{M}] : P\Big\downarrow^{m\backslash o}_{e}_{n\cup\{p\}}\Big\downarrow & v = \mathrm{E} \\ p[\mathrm{D}] : P\Big\downarrow^{(m\cup\{p\})\backslash o}_{e}_{n}\Big\downarrow & \\ \qquad \cup\, p[\mathrm{M}] : P\Big\downarrow^{(m\cup\{p\})\backslash o}_{e}_{n\cup\{p\}}\Big\downarrow & v = \mathrm{M} \\ \varnothing & v = \mathrm{D} \end{cases}
\end{aligned}
$$

$$= \quad (p[v] : P)\Big\downarrow \begin{smallmatrix} m\backslash o \\ e \\ n \end{smallmatrix} \Big\downarrow$$
$$= \quad ({}^{o}p[v] : P)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$$

- For $p \notin m \cup n$, $o \nsubseteq m \backslash n$: Then also $o \nsubseteq m \backslash n \cup \{p\}$.

$$\begin{aligned}
(p[v] : {}^{o}P)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad &= \quad p[v] : (^{o}P)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \cup \\
&\qquad \begin{cases} p[\mathrm{M}] : (^{o}P)\Big\downarrow \begin{smallmatrix} m \\ e \\ n\cup\{p\} \end{smallmatrix} \Big\downarrow & v = \mathrm{E} \\ p[\mathrm{D}] : (^{o}P)\Big\downarrow \begin{smallmatrix} m\cup\{p\} \\ e \\ n \end{smallmatrix} \Big\downarrow & \\ \qquad \cup\ p[\mathrm{M}] : (^{o}P)\Big\downarrow \begin{smallmatrix} m\cup\{p\} \\ e \\ n\cup\{p\} \end{smallmatrix} \Big\downarrow & v = \mathrm{M} \\ \varnothing & v = \mathrm{D} \end{cases} \\
&= \quad \varnothing \quad = \quad ({}^{o}p[v] : P)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}$$

Eq. (5.16)**:** By coinduction. This equation needs a bit more effort than the last two ones, but we can refer to the proof of Eq. (5.15) for some parts of the derivation. We perform coinduction on the following equivalence relation:

$$\mathit{Rel} \quad := \quad <\{({}^{o*}p[v'] : P',\ p[v'] : {}^{o*}P') \ :\ P' \in \mathit{CProc}_{\Sigma_0,S},\ v' \in \{\mathrm{E},\mathrm{M},\mathrm{D}\}\}>$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and P', v' as in the definition of *Rel*.

- For $p \in m \cup n$:

$$\begin{aligned}
(p[v'] : {}^{o*}P')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad &= \quad \varnothing \\
&= \quad \begin{cases} {}^{o*}\left((p[v'] : P')\Big\downarrow \begin{smallmatrix} m\backslash o \\ e \\ n\backslash o \end{smallmatrix} \Big\downarrow\right) & e \in \Sigma_\tau,\ o \subseteq m \cap n \\ (p[v'] : P')\Big\downarrow \begin{smallmatrix} m\backslash o \\ e \\ n \end{smallmatrix} \Big\downarrow & e \in \Sigma,\ o \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad ({}^{o*}p[v'] : P')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}$$

- For $p \notin m \cup n$, $e \in \Sigma_\tau$, $o \subseteq m \cap n$:

$$\begin{aligned}
(p[v'] : {}^{o*}P')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad &= \quad p[v'] : (^{o*}P')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \cup \\
&\qquad \begin{cases} p[\mathrm{M}] : (^{o*}P')\Big\downarrow \begin{smallmatrix} m \\ e \\ n\cup\{p\} \end{smallmatrix} \Big\downarrow & v' = \mathrm{E} \\ p[\mathrm{D}] : (^{o*}P')\Big\downarrow \begin{smallmatrix} m\cup\{p\} \\ e \\ n \end{smallmatrix} \Big\downarrow & \\ \qquad \cup\ p[\mathrm{M}] : (^{o*}P')\Big\downarrow \begin{smallmatrix} m\cup\{p\} \\ e \\ n\cup\{p\} \end{smallmatrix} \Big\downarrow & v' = \mathrm{M} \\ \varnothing & v' = \mathrm{D} \end{cases} \\
&= \quad p[v'] : {}^{o*}\left(P'\Big\downarrow \begin{smallmatrix} m\backslash o \\ e \\ n\backslash o \end{smallmatrix} \Big\downarrow\right) \quad \cup
\end{aligned}$$

$$
\begin{array}{rl}
 & \begin{cases}
p[\mathrm{M}] : {}^{o*}\left(P' \Big\downarrow {}^{m\setminus o}_{(n\cup\{p\})\setminus o}\Big\downarrow\right) & v' = \mathrm{E} \\
p[\mathrm{D}] : {}^{o*}\left(P' \Big\downarrow {}^{(m\cup\{p\})\setminus o}_{n\setminus o}\Big\downarrow\right) & \\
\qquad \cup\, p[\mathrm{M}] : {}^{o*}\left(P' \Big\downarrow {}^{(m\cup\{p\})\setminus o}_{(n\cup\{p\})\setminus o}\Big\downarrow\right) & v' = \mathrm{M} \\
\varnothing & v' = \mathrm{D}
\end{cases} \\
\sim_{Rel} \quad {}^{o*}\left(p[v'] : P' \Big\downarrow {}^{m\setminus o}_{n\setminus o}\Big\downarrow\right) \quad \cup & \\
 & \begin{cases}
{}^{o*}\left(p[\mathrm{M}] : P' \Big\downarrow {}^{m\setminus o}_{(n\cup\{p\})\setminus o}\Big\downarrow\right) & v' = \mathrm{E} \\
{}^{o*}\left(p[\mathrm{D}] : P' \Big\downarrow {}^{(m\cup\{p\})\setminus o}_{n\setminus o}\Big\downarrow\right) & \\
\qquad \cup\ {}^{o*}\left(p[\mathrm{M}] : P' \Big\downarrow {}^{(m\cup\{p\})\setminus o}_{(n\cup\{p\})\setminus o}\Big\downarrow\right) & v' = \mathrm{M} \\
\varnothing & v' = \mathrm{D}
\end{cases} \\
= \quad {}^{o*}\left((p[v'] : P') \Big\downarrow {}^{m\setminus o}_{n\setminus o}\Big\downarrow\right) & \\
= \quad ({}^{o*}p[v'] : P') \Big\downarrow {}^{m}_{n}\Big\downarrow &
\end{array}
$$

- For $p \notin m \cup n$: The other cases, i.e. $e \in \Sigma_\tau \wedge o \nsubseteq m \cap n$ and $e \in \Sigma$ are conducted exactly as for Eq. (5.15), as the starred consumption operator behaves like normal consumption in these situations.

Eq. (5.17)**:** Omitted.

Eq. (5.18)**:** By coinduction on the following equivalence relation:

$$
Rel \quad := \quad < \left\{ \begin{array}{c} ((p[v'] : P') \,|\![\, L'[X]R' \,]\!|\, Q',\ p[v'] : (P' \,|\![\, L'[X]R' \,]\!|\, Q')) \ : \\ P', Q' \in CProc_{\Sigma_0,S},\ L', R' \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP},\ v' \in \{\mathrm{E},\mathrm{M},\mathrm{D}\}, \\ p \notin FP(Q'),\ p \notin FP(L'),\ p \notin FP(R') \end{array} \right\} >
$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and P', Q', L', R', v' as in the definition of *Rel*. We need the definition of the causal state operator and use the following abbreviations for $p \notin m' \cup n'$:

$$
\begin{array}{rcl}
C(m', n') & := & (p[v'] : P') \Big\downarrow {}^{m'}_{n'}\Big\downarrow \quad = \quad p[v'] : P' \Big\downarrow {}^{m'}_{n'}\Big\downarrow \cup\, D(m', n') \\
D(m', n') & := & \begin{cases}
p[\mathrm{M}] : P' \Big\downarrow {}^{m'}_{n'\cup\{p\}}\Big\downarrow & v' = \mathrm{E} \\
p[\mathrm{D}] : P' \Big\downarrow {}^{m'\cup\{p\}}_{n'}\Big\downarrow \ \cup\ p[\mathrm{M}] : P' \Big\downarrow {}^{m'\cup\{p\}}_{n'\cup\{p\}}\Big\downarrow & v' = \mathrm{M} \\
\varnothing & v' = \mathrm{D}
\end{cases}
\end{array}
$$

- For $p \notin m \cup n$, $e \in X$:

$$
\begin{array}{lll}
((p[v'] : P') \,|[\, L'[X]R' \,]|\, Q')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow & & \\
= & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & C(m_1,n_1) \,|[\, L'[X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
& \cup & (p[v'] : P') \,|[\, L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
& \cup & C(m_1,n_1) \,|[\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]|\, Q' \\
= & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & \left.\begin{array}{l} \left(p[v'] : P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow\right) \,|[\, L'[X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\ \cup\ (p[v'] : P') \,|[\, L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\ \cup\ \left(p[v'] : P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow\right) \,|[\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]|\, Q' \end{array}\right\} =: U_1 \\
\cup & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & \left.\begin{array}{l} D(m_1,n_1) \,|[\, L'[X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\ \cup\ D(m_1,n_1) \,|[\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]|\, Q' \end{array}\right\} =: U_2
\end{array}
$$

$$
\begin{array}{lll}
U_1 \quad \sim_{Rel} & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & p[v'] : \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,|[\, L'[X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow\right) \\
& \cup & p[v'] : \left(P' \,|[\, L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow\right) \\
& \cup & p[v'] : \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,|[\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]|\, Q'\right) \\
= & & p[v'] : (P' \,|[\, L'[X]R' \,]|\, Q')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{array}
$$

For the remaining part U_2 we make a further case distinction on the value of v':

– For $v' = \mathrm{E}$: (*) employs Impl. (5.4) to show that relation R can be applied, (**) holds because of $p \notin FP(L') \cup FP(R') \cup FP(Q')$ and Impl. (5.3).

$$
\begin{array}{lll}
U_2 \quad = & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & \left(p[\mathrm{M}] : P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \cup \{p\} \end{smallmatrix} \Big\downarrow\right) \,|[\, L'[X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
& \cup & \left(p[\mathrm{M}] : P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \cup \{p\} \end{smallmatrix} \Big\downarrow\right) \,|[\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]|\, Q' \\
\overset{(*)}{\sim}_{Rel} & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & p[\mathrm{M}] : \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \cup \{p\} \end{smallmatrix} \Big\downarrow \,|[\, L'[X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow\right) \\
& \cup & p[\mathrm{M}] : \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \cup \{p\} \end{smallmatrix} \Big\downarrow \,|[\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]|\, Q'\right) \\
\overset{(**)}{=} & p[\mathrm{M}] : \bigcup_{\substack{(m,n\cup\{p\})= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,|[\, L'[X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
& & \cup\ P' \,|[\, L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R' \,]|\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
& & \cup\ P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,|[\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]|\, Q'
\end{array}
$$

$$= \quad p[\mathrm{M}] : (P' \,[\![\, L'[X]R' \,]\!]\, Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow$$

– For $v' = \mathrm{M}$: (*) and (**) have the same meaning as for the first case.

$$\begin{aligned}
U_2 \quad = \quad & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} \Big(p[\mathrm{D}] : P' \Big\downarrow \begin{smallmatrix} m_1 \cup \{p\} \\ e \\ n_1 \end{smallmatrix} \Big\downarrow\Big) \,[\![\, L'[X]R' \,]\!]\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
& \cup \Big(p[\mathrm{M}] : P' \Big\downarrow \begin{smallmatrix} m_1 \cup \{p\} \\ e \\ n_1 \cup \{p\} \end{smallmatrix} \Big\downarrow\Big) \,[\![\, L'[X]R' \,]\!]\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\
& \cup \Big(p[\mathrm{D}] : P' \Big\downarrow \begin{smallmatrix} m_1 \cup \{p\} \\ e \\ n_1 \end{smallmatrix} \Big\downarrow\Big) \,[\![\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]\!]\, Q' \\
& \cup \Big(p[\mathrm{M}] : P' \Big\downarrow \begin{smallmatrix} m_1 \cup \{p\} \\ e \\ n_1 \cup \{p\} \end{smallmatrix} \Big\downarrow\Big) \,[\![\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]\!]\, Q' \\
\overset{(*)}{\sim}_{Rel} \quad & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} p[\mathrm{D}] : \Big(P' \Big\downarrow \begin{smallmatrix} m_1 \cup \{p\} \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]R' \,]\!]\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow\Big) \\
& \cup \; p[\mathrm{M}] : \Big(P' \Big\downarrow \begin{smallmatrix} m_1 \cup \{p\} \\ e \\ n_1 \cup \{p\} \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]R' \,]\!]\, Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow\Big) \\
& \cup \; p[\mathrm{D}] : \Big(P' \Big\downarrow \begin{smallmatrix} m_1 \cup \{p\} \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]\!]\, Q'\Big) \\
& \cup \; p[\mathrm{M}] : \Big(P' \Big\downarrow \begin{smallmatrix} m_1 \cup \{p\} \\ e \\ n_1 \cup \{p\} \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]\!]\, Q'\Big) \\
\overset{(**)}{=} \quad & p[\mathrm{D}] : (P' \,[\![\, L'[X]R' \,]\!]\, Q') \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \end{smallmatrix} \Big\downarrow \\
& \cup \; p[\mathrm{M}] : (P' \,[\![\, L'[X]R' \,]\!]\, Q') \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow
\end{aligned}$$

– For $v' = \mathrm{D}$: $U_2 = \varnothing$

The derivation is concluded by putting the pieces together and using the characteristic equation of the state operator:

$$\begin{aligned}
U_1 \cup U_2 \quad \sim_{Rel} \quad & p[v'] : (P' \,[\![\, L'[X]R' \,]\!]\, Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \cup \\
& \begin{cases}
p[\mathrm{M}] : (P' \,[\![\, L'[X]R' \,]\!]\, Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow & v' = \mathrm{E} \\
p[\mathrm{D}] : (P' \,[\![\, L'[X]R' \,]\!]\, Q') \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \end{smallmatrix} \Big\downarrow & \\
\qquad \cup \; p[\mathrm{M}] : (P' \,[\![\, L'[X]R' \,]\!]\, Q') \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow & v' = \mathrm{M} \\
\varnothing & v' = \mathrm{D}
\end{cases} \\
= \quad & (p[v'] : (P' \,[\![\, L'[X]R' \,]\!]\, Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}$$

- For $p \notin m \cup n$, $e \notin X$: (*) employs Impl. (5.4) to show that relation R can be applied, (**) holds because $p \notin FP(Q')$ and Impl. (5.3) entail that the

second part of the characteristic equation of interleaving disappears.

$$
\begin{aligned}
&((p[v'] : P') \,[\![L'[X]R']\!]\, Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&\quad = \quad C(m,n) \,[\![L'[X]R']\!]\, Q' \;\cup\; (p[v'] : P') \,[\![L'[X]R']\!]\, Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&\quad = \quad \left(p[v'] : P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right) [\![L'[X]R']\!]\, Q' \\
&\qquad\qquad \cup\; (p[v'] : P') \,[\![L'[X]R']\!]\, Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \cup \\
&\qquad\qquad \begin{cases} p[\mathrm{M}] : P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow & v' = \mathrm{E} \\ p[\mathrm{D}] : P' \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \end{smallmatrix} \Big\downarrow \;\cup\; p[\mathrm{M}] : P' \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow & v' = \mathrm{M} \\ \varnothing & v' = \mathrm{D} \end{cases} [\![L'[X]R']\!]\, Q' \\
&\overset{(*)}{\sim}_{Rel} \quad p[v'] : \left(P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow [\![L'[X]R']\!]\, Q' \;\cup\; P' \,[\![L'[X]R']\!]\, Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right) \quad \cup \\
&\qquad\qquad \begin{cases} p[\mathrm{M}] : \left(P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow [\![L'[X]R']\!]\, Q'\right) & v' = \mathrm{E} \\ p[\mathrm{D}] : \left(P' \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \end{smallmatrix} \Big\downarrow [\![L'[X]R']\!]\, Q'\right) & \\ \qquad \cup\, p[\mathrm{M}] : \left(P' \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow [\![L'[X]R']\!]\, Q'\right) & v' = \mathrm{M} \\ \varnothing & v' = \mathrm{D} \end{cases} \\
&\overset{(**)}{=} \quad p[v'] : (P' \,[\![L'[X]R']\!]\, Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \cup \\
&\qquad\qquad \begin{cases} p[\mathrm{M}] : (P' \,[\![L'[X]R']\!]\, Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow & v' = \mathrm{E} \\ p[\mathrm{D}] : (P' \,[\![L'[X]R']\!]\, Q') \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \end{smallmatrix} \Big\downarrow & \\ \qquad \cup\, p[\mathrm{M}] : (P' \,[\![L'[X]R']\!]\, Q') \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow & v' = \mathrm{M} \\ \varnothing & v' = \mathrm{D} \end{cases} \\
&\quad = \quad (p[v'] : (P' \,[\![L'[X]R']\!]\, Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

- For $p \in m \cup n$: (*) in the final equation follows by checking that each part of the right side of the characteristic equation of parallelism is the empty set: For $p \in m' \cup n'$ we have (because of the declarations of Q', L', R')

$$
(p[v'] : P') \Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow = \varnothing \qquad Q' \Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow = \varnothing \qquad L' \Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow = \varnothing \qquad R' \Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow = \varnothing
$$

$$
\begin{aligned}
(p[v'] : (P' \,[\![L'[X]R']\!]\, Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad &= \quad \varnothing \\
&\overset{(*)}{=} \quad ((p[v'] : P') \,[\![L'[X]R']\!]\, Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

Eq. (5.13)**:** Omitted.

Eq. (5.14)**:** Omitted. □

5.2.3 Proof (Page 90): **Eq.** (5.19)**:** By coinduction: The dynamics of the two processes coincide: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label.

$$
\begin{aligned}
\left({}^{o}({}^{r}P)\right)\Big\downarrow_{n}^{e}{}^{m}\Big\downarrow &= \begin{cases} \left({}^{r}P\right)\Big\downarrow_{n}^{e}{}^{m\backslash o}\Big\downarrow & o \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} P\Big\downarrow_{n}^{e}{}^{m\backslash(o\cup r)}\Big\downarrow & o \subseteq m\backslash n, r \subseteq m\backslash o\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} \left({}^{o\cup r}P\right)\Big\downarrow_{n}^{e}{}^{m}\Big\downarrow & r \cap o = \varnothing \\ \varnothing = stop\Big\downarrow_{n}^{e}{}^{m}\Big\downarrow & r \cap o \neq \varnothing \end{cases}
\end{aligned}
$$

Eq. (5.20)**:** By coinduction. The two cases are treated in slightly different ways:

- For $r \cap o = \varnothing$: The coinduction is performed on the equivalence relation

$$Rel \quad := \quad <\{({}^{o*}({}^{r*}P'),\ {}^{o\cup r*}P')\ :\ P' \in CProc_{\Sigma_0,S}\}>$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and P' as in the definition of *Rel*. In the derivation we use the following auxiliary equivalences that are entailed by $r \cap o = \varnothing$:

$$r \subseteq m\backslash o \iff r \subseteq m \qquad r \subseteq (m\backslash o) \cap (n\backslash o) \iff r \subseteq m \cap n \qquad (*)$$

$$
\begin{aligned}
\left({}^{o*}({}^{r*}P')\right)\Big\downarrow_{n}^{e}{}^{m}\Big\downarrow &= \begin{cases} {}^{o*}\left(\left({}^{r*}P'\right)\Big\downarrow_{n\backslash o}^{e}{}^{m\backslash o}\Big\downarrow\right) & e \in \Sigma_\tau, o \subseteq m \cap n \\ \left({}^{r*}P'\right)\Big\downarrow_{n}^{e}{}^{m\backslash o}\Big\downarrow & e \in \Sigma, o \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&\overset{(*)}{=} \begin{cases} {}^{o*}\left({}^{r*}\left(P'\Big\downarrow_{n\backslash(o\cup r)}^{e}{}^{m\backslash(o\cup r)}\Big\downarrow\right)\right) & e \in \Sigma_\tau, o \cup r \subseteq m \cap n \\ P'\Big\downarrow_{n}^{e}{}^{m\backslash(o\cup r)}\Big\downarrow & e \in \Sigma, o \cup r \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&\sim_{Rel} \begin{cases} {}^{o\cup r*}\left(P'\Big\downarrow_{n\backslash(o\cup r)}^{e}{}^{m\backslash(o\cup r)}\Big\downarrow\right) & e \in \Sigma_\tau, o \cup r \subseteq m \cap n \\ P'\Big\downarrow_{n}^{e}{}^{m\backslash(o\cup r)}\Big\downarrow & e \in \Sigma, o \cup r \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \left({}^{o\cup r*}P'\right)\Big\downarrow_{n}^{e}{}^{m}\Big\downarrow
\end{aligned}
$$

- For $r \cap o \neq \varnothing$: We show that the dynamics of the two processes coincide:

Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label. (*) uses $r \nsubseteq m \backslash o$.

$$
\begin{aligned}
\left({}^{o}*\left({}^{r}*P'\right)\right)\Big\downarrow^{m}_{e}_{n}\Big\downarrow \quad &= \quad \begin{cases} {}^{o}*\left(\left({}^{r}*P'\right)\Big\downarrow^{m\backslash o}_{e}_{n\backslash o}\Big\downarrow\right) & e \in \Sigma_\tau, o \subseteq m \cap n \\ \left({}^{r}*P'\right)\Big\downarrow^{m\backslash o}_{e}_{n}\Big\downarrow & e \in \Sigma, o \subseteq m \backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&\overset{(*)}{=} \quad \varnothing \quad = \quad \mathit{stop}\Big\downarrow^{m}_{e}_{n}\Big\downarrow
\end{aligned}
$$

Eq. (5.21)**:** By coinduction: We show that the dynamics of the two processes coincide: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label.

$$
\begin{aligned}
\left({}^{o}\left({}^{r}*P'\right)\right)\Big\downarrow^{m}_{e}_{n}\Big\downarrow \quad &= \quad \begin{cases} \left({}^{r}*P'\right)\Big\downarrow^{m\backslash o}_{e}_{n}\Big\downarrow & o \subseteq m \backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad \begin{cases} {}^{r}*\left(P'\Big\downarrow^{m\backslash(o\cup r)}_{e}_{n\backslash r}\Big\downarrow\right) & e \in \Sigma_\tau, o \subseteq m\backslash n, r \subseteq (m\backslash o) \cap n \\ P'\Big\downarrow^{m\backslash(o\cup r)}_{e}_{n}\Big\downarrow & e \in \Sigma, o \subseteq m\backslash n, r \subseteq m\backslash o\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&\overset{(*)}{=} \quad \begin{cases} {}^{r}*\left(P'\Big\downarrow^{m\backslash(o\cup r)}_{e}_{n\backslash r}\Big\downarrow\right) & e \in \Sigma_\tau, r \subseteq m \cap n, o \subseteq m\backslash r\backslash n \\ P'\Big\downarrow^{m\backslash(o\cup r)}_{e}_{n}\Big\downarrow & e \in \Sigma, r \subseteq m\backslash n, o \subseteq m\backslash r\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad \begin{cases} {}^{r}*\left(\left({}^{o}P'\right)\Big\downarrow^{m\backslash r}_{e}_{n\backslash r}\Big\downarrow\right) & e \in \Sigma_\tau, r \subseteq m \cap n \\ \left({}^{o}P'\right)\Big\downarrow^{m\backslash r}_{e}_{n}\Big\downarrow & e \in \Sigma, r \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad \left({}^{r}*\left({}^{o}P'\right)\right)\Big\downarrow^{m}_{e}_{n}\Big\downarrow
\end{aligned}
$$

Equation (*) holds because of the equivalences

$$
\begin{aligned}
o \subseteq m\backslash n \wedge r \subseteq (m\backslash o) \cap n \quad &\Longleftrightarrow \quad o \subseteq m\backslash n \wedge r \subseteq m \cap n \wedge o \cap r = \varnothing \\
&\Longleftrightarrow \quad r \subseteq m \cap n \wedge o \subseteq m\backslash n\backslash r \\
o \subseteq m\backslash n \wedge r \subseteq m\backslash o\backslash n \quad &\Longleftrightarrow \quad o \subseteq m\backslash n \wedge r \subseteq m\backslash n \wedge o \cap r = \varnothing \\
&\Longleftrightarrow \quad r \subseteq m\backslash n \wedge o \subseteq m\backslash r\backslash n
\end{aligned}
$$

Eq. (5.22)**:** By coinduction: We show that the dynamics of the processes ${}^{o}({}^{r}*P)$ and *stop* coincide, the second part of the conjecture follows from Eq. (5.21). The derivation is similar to the one of Eq. (5.21). Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label.

(*) uses $r \not\subseteq m\backslash o$.

$$
\begin{aligned}
({}^{o}({}^{r*}P))\Big\downarrow^{m}_{e}_{n}\Big\downarrow \quad &= \quad \begin{cases} ({}^{r*}P)\Big\downarrow^{m\backslash o}_{e}_{n}\Big\downarrow & o \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad \begin{cases} {}^{r*}\left(P\Big\downarrow^{m\backslash(o\cup r)}_{e}_{n\backslash r}\Big\downarrow\right) & e \in \Sigma_\tau, o \subseteq m\backslash n, r \subseteq (m\backslash o) \cap n \\ P\Big\downarrow^{m\backslash(o\cup r)}_{e}_{n}\Big\downarrow & e \in \Sigma, o \subseteq m\backslash n, r \subseteq m\backslash o\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&\overset{(*)}{=} \quad \varnothing \quad = \quad stop\Big\downarrow^{m}_{e}_{n}\Big\downarrow
\end{aligned}
$$

Eq. (5.23): Both parts of the proposition are proved using coinduction. For ${}^{\varnothing}P = P$ we only need to show that the dynamics of the two processes are equal. Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label.

$$
({}^{\varnothing}P)\Big\downarrow^{m}_{e}_{n}\Big\downarrow \quad = \quad \left\{\begin{array}{ll} P\Big\downarrow^{m\backslash\varnothing}_{e}_{n}\Big\downarrow & \varnothing \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{array}\right\} \quad = \quad P\Big\downarrow^{m}_{e}_{n}\Big\downarrow
$$

The second part ${}^{\varnothing*}P = P$ is proved by coinduction on the equivalence relation

$$
Rel \quad := \quad <\{({}^{\varnothing*}P',\ P')\ :\ P' \in CProc_{\Sigma_0,S}\}>
$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and P' as in the definition of *Rel*.

$$
({}^{\varnothing*}P')\Big\downarrow^{m}_{e}_{n}\Big\downarrow \quad = \quad \left\{\begin{array}{ll} {}^{\varnothing*}\left(P'\Big\downarrow^{m\backslash\varnothing}_{e}_{n\backslash\varnothing}\Big\downarrow\right) & e \in \Sigma_\tau, \varnothing \subseteq m \cap n \\ P'\Big\downarrow^{m\backslash\varnothing}_{e}_{n}\Big\downarrow & e \in \Sigma, \varnothing \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{array}\right\} \quad \sim_{Rel} \quad P'\Big\downarrow^{m}_{e}_{n}\Big\downarrow
$$

Eq. (5.24): Omitted. □

5.2.4 Proof (Page 91): All proof are conducted by coinduction and by showing that the dynamics of the two sides of the equations coincide. This simple proof pattern can be applied because transitions necessarily break down after a single execution step. For all items suppose that $(m, e, n) \in \mathcal{L}_{CCSP}$ is an LTS label.

Eq. (5.25):

$$
\begin{aligned}
({}^{s}({}^{o}\,a\,{}^{r}))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad &= \quad \begin{cases} {}^{o}\,a\,{}^{r}\Big\downarrow \begin{smallmatrix} m\backslash s \\ e \\ n \end{smallmatrix} \Big\downarrow & s \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad \begin{cases} \{stop\} & s \subseteq m\backslash n, (o,a,r) = (m\backslash s, e, n) \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad \begin{cases} \{stop\} & s \cap (o \cup r) = \varnothing, (o \cup s, a, r) = (m, e, n) \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad \begin{cases} {}^{o\cup s}\,a\,{}^{r}\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow & s \cap (o \cup r) = \varnothing \\ stop\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow & \text{otherwise} \end{cases}
\end{aligned}
$$

Eq. (5.26):

$$
\begin{aligned}
&({}^{s*}({}^{o}\,a\,{}^{r}))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= \quad \begin{cases} {}^{s*}\left({}^{o}\,a\,{}^{r}\Big\downarrow \begin{smallmatrix} m\backslash s \\ e \\ n\backslash s \end{smallmatrix} \Big\downarrow\right) & e \in \Sigma_\tau, s \subseteq m \cap n \\ {}^{o}\,a\,{}^{r}\Big\downarrow \begin{smallmatrix} m\backslash s \\ e \\ n \end{smallmatrix} \Big\downarrow & e \in \Sigma, s \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad \begin{cases} {}^{s*}\{stop\} & e \in \Sigma_\tau, s \subseteq m \cap n, (o,a,r) = (m\backslash s, e, n\backslash s) \\ \{stop\} & e \in \Sigma, s \subseteq m\backslash n, (o,a,r) = (m\backslash s, e, n) \\ \varnothing & \text{otherwise} \end{cases} \\
&\overset{(5.24)}{=} \quad \begin{cases} \{stop\} & a \in \Sigma_\tau, s \cap (o \cup r) = \varnothing, (o \cup s, a, r \cup s) = (m, e, n) \\ \{stop\} & a \in \Sigma, s \cap (o \cup r) = \varnothing, (o \cup s, a, r) = (m, e, n) \\ \varnothing & \text{otherwise} \end{cases} \\
&= \quad \begin{cases} {}^{o\cup s}\,a\,{}^{r\cup s}\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow & a \in \Sigma_\tau, s \cap (o \cup r) = \varnothing \\ {}^{o\cup s}\,a\,{}^{r}\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow & a \in \Sigma, s \cap (o \cup r) = \varnothing \\ stop\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow & \text{otherwise} \end{cases}
\end{aligned}
$$

Eq. (5.27): (*) uses $l : stop = stop$.

$$
\begin{aligned}
(l : {}^{o}\,a\,{}^{r})\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad &= \quad \bigcup_{e \in \Sigma_\tau} l : {}^{o}\,a\,{}^{r}\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \cup \quad \bigcup_{\substack{c \in \Sigma \\ e = l.c}} l : {}^{o}\,a\,{}^{r}\Big\downarrow \begin{smallmatrix} m \\ c \\ n \end{smallmatrix} \Big\downarrow \\
&= \quad \bigcup_{\substack{e \in \Sigma_\tau \\ (o,a,r) = (m,e,n)}} l : \{stop\} \quad \cup \quad \bigcup_{\substack{c \in \Sigma \\ e = l.c \\ (o,a,r) = (m,c,n)}} l : \{stop\}
\end{aligned}
$$

$$\stackrel{(*)}{=} \bigcup_{\substack{a\in\Sigma_\tau \\ (o,a,r)=(m,e,n)}} \{stop\} \;\cup \bigcup_{\substack{a\in\Sigma \\ (o,l.a,r)=(m,e,n)}} \{stop\}$$

$$= \begin{cases} {}^{o}(l.a)^{\,r}\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow & a \in \Sigma \\ {}^{o}a^{\,r}\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow & a \in \Sigma_\tau \end{cases}$$

Eq. (5.28): (*) uses $stop \setminus X = stop$.

$$({}^{o}a^{\,r} \setminus X)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow = \bigcup_{e\notin X} {}^{o}a^{\,r}\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \setminus X \;\cup \bigcup_{\substack{c\in X \\ e=\tau(c)}} {}^{o}a^{\,r}\Big\downarrow \begin{smallmatrix} m \\ c \\ n \end{smallmatrix}\Big\downarrow \setminus X$$

$$= \bigcup_{\substack{e\notin X \\ (o,a,r)=(m,e,n)}} \{stop\} \setminus X \;\cup \bigcup_{\substack{c\in X \\ e=\tau(c) \\ (o,a,r)=(m,c,n)}} \{stop\} \setminus X$$

$$\stackrel{(*)}{=} \bigcup_{\substack{a\notin X \\ (o,a,r)=(m,e,n)}} \{stop\} \;\cup \bigcup_{\substack{a\in X \\ (o,\tau(a),r)=(m,e,n)}} \{stop\}$$

$$= \begin{cases} {}^{o}(\tau(a))^{\,r}\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow & a \in X \\ {}^{o}a^{\,r}\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow & a \notin X \end{cases}$$

□

5.3.1 Proof (Page 93): 1. By coinduction. For the first equation $p[\mathrm{M}] : {}^{\{p\}}P = P$ we show that the two processes have the same adjacent processes. Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label.

- For $p \notin m \cup n$: (*) holds because of $p \notin m$ and the characteristic equation of token consumption, (**) follows from Lem. 5.1.3.

$$(p[\mathrm{M}] : {}^{\{p\}}P)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow = p[\mathrm{M}] : ({}^{\{p\}}P)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \;\cup\; p[\mathrm{D}] : ({}^{\{p\}}P)\Big\downarrow \begin{smallmatrix} m\cup\{p\} \\ e \\ n \end{smallmatrix}\Big\downarrow$$
$$\cup\; p[\mathrm{M}] : ({}^{\{p\}}P)\Big\downarrow \begin{smallmatrix} m\cup\{p\} \\ e \\ n\cup\{p\} \end{smallmatrix}\Big\downarrow$$
$$\stackrel{(*)}{=} p[\mathrm{D}] : P\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \quad \stackrel{(**)}{=} \quad P\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow$$

- For $p \in m \cup n$: The characteristic equation of token consumption and Impl. (5.3) entail

$$(p[\mathrm{M}] : {}^{\{p\}}P)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow = \varnothing = P\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow$$

The second equation $p[\mathrm{M}] : {}^{\{p\}*}P = P$ is proved by coinduction on the equivalence relation

$$Rel := <\{(p[\mathrm{M}] : {}^{\{p\}*}P',\ P') \ :\ P' \in CProc_{\Sigma_0,S},\ p \notin FP(P')\}>$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be an LTS label and P' as in the definition of *Rel*.

- For $p \notin m \cup n$: (*) and (**) have the same meanings as for the first equation.

$$
\begin{aligned}
(p[\mathrm{M}] : {}^{\{p\}*}P')\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow \quad &= \quad p[\mathrm{M}] : ({}^{\{p\}*}P')\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow \;\cup\; p[\mathrm{D}] : ({}^{\{p\}*}P')\Big\downarrow^{m\cup\{p\}}_{\substack{e\\n}}\Big\downarrow \\
&\qquad \cup\; p[\mathrm{M}] : ({}^{\{p\}*}P')\Big\downarrow^{m\cup\{p\}}_{\substack{e\\n\cup\{p\}}}\Big\downarrow \\
&\overset{(*)}{=} \begin{cases} p[\mathrm{M}] : {}^{\{p\}*}P'\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow & e \in \Sigma_\tau \\ p[\mathrm{D}] : P'\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow & e \in \Sigma \end{cases} \\
&\overset{(**)}{\sim_{Rel}} \left\{ \begin{matrix} P'\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow & e \in \Sigma_\tau \\ P'\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow & e \in \Sigma \end{matrix} \right\} \quad = \quad P'\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow
\end{aligned}
$$

- For $p \in m \cup n$: Exactly as for the first equation.

2. By coinduction: We show that the dynamics of the processes coincide. Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label.

   - For $p \notin m \cup n$:

$$
\begin{aligned}
(p[\mathrm{D}] : {}^{\{p\}}P)\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow \quad &= \quad p[\mathrm{D}] : ({}^{\{p\}}P)\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow \quad = \quad \varnothing \\
&= \quad p[\mathrm{D}] : ({}^{\{p\}*}P)\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow \quad = \quad stop\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow
\end{aligned}
$$

   - For $p \in m \cup n$:

$$
(p[\mathrm{D}] : {}^{\{p\}}P)\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow \quad = \quad (p[\mathrm{D}] : {}^{\{p\}*}P)\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow \quad = \quad \varnothing \quad = \quad stop\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow
$$

3. By coinduction: We show that the dynamics of the processes coincide. Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label. (*) holds because of $a \notin \Sigma_\tau$.

$$
\begin{aligned}
&({}^{o*}({}^{r}(a \to P)))\Big\downarrow^{m}_{\substack{e\\n}}\Big\downarrow \\
&= \begin{cases} {}^{o*}\left(({}^{r}(a \to P))\Big\downarrow^{m\backslash o}_{\substack{e\\n\backslash o}}\Big\downarrow\right) & e \in \Sigma_\tau,\ o \subseteq m \cap n \\ ({}^{r}(a \to P))\Big\downarrow^{m\backslash o}_{\substack{e\\n}}\Big\downarrow & e \in \Sigma,\ o \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} {}^{o*}\left((a \to P)\Big\downarrow^{m\backslash(o\cup r)}_{\substack{e\\n\backslash o}}\Big\downarrow\right) & e \in \Sigma_\tau,\ o \subseteq m \cap n,\ r \subseteq m\backslash o\backslash n \\ (a \to P)\Big\downarrow^{m\backslash(o\cup r)}_{\substack{e\\n}}\Big\downarrow & e \in \Sigma,\ o \cup r \subseteq m\backslash n \\ \varnothing & \text{otherwise} \end{cases} \\
&\overset{(*)}{=} \begin{cases} \{P\} & o \cup r \subseteq m\backslash n,\ (m\backslash(o \cup r), e, n) = (\varnothing, a, \varnothing) \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} \{P\} & e = a,\ m = o \cup r,\ n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&=: \quad U
\end{aligned}
$$

For the remaining parts of the derivation we make a case distinction:

- For $p \notin m \cup n$: We use auxiliary statements (**) that hold for $p \notin m \cup n$ because of the characteristic equations for transitions and token consumption:

$$ {}^{o\cup r}a\,{}^{\{p\}}\Big\downarrow {}^{m}_{n}\Big\downarrow = \varnothing, \qquad ({}^{\{p\}}P)\Big\downarrow {}^{m}_{n}\Big\downarrow = \varnothing, \qquad ({}^{\{p\}}P)\Big\downarrow {}^{m}_{n\cup\{p\}}\Big\downarrow = \varnothing \quad (**) $$

  The derivation is continued starting with the right side:

$$
\begin{aligned}
& (p[\mathrm{E}] : ({}^{o\cup r}a\,{}^{\{p\}} \;|||\; {}^{\{p\}}P))\Big\downarrow {}^{m}_{n}\Big\downarrow \\
= \;& p[\mathrm{E}] : ({}^{o\cup r}a\,{}^{\{p\}} \;|||\; {}^{\{p\}}P)\Big\downarrow {}^{m}_{n}\Big\downarrow \\
& \quad \cup \quad p[\mathrm{M}] : ({}^{o\cup r}a\,{}^{\{p\}} \;|||\; {}^{\{p\}}P)\Big\downarrow {}^{m}_{n\cup\{p\}}\Big\downarrow \\
= \;& p[\mathrm{E}] : \Big({}^{o\cup r}a\,{}^{\{p\}}\Big\downarrow {}^{m}_{n}\Big\downarrow \;|||\; {}^{\{p\}}P \;\cup\; {}^{o\cup r}a\,{}^{\{p\}} \;|||\; ({}^{\{p\}}P)\Big\downarrow {}^{m}_{n}\Big\downarrow\Big) \\
& \quad \cup \quad p[\mathrm{M}] : \Big({}^{o\cup r}a\,{}^{\{p\}}\Big\downarrow {}^{m}_{n\cup\{p\}}\Big\downarrow \;|||\; {}^{\{p\}}P\Big) \\
& \quad \cup \quad p[\mathrm{M}] : \Big({}^{o\cup r}a\,{}^{\{p\}} \;|||\; ({}^{\{p\}}P)\Big\downarrow {}^{m}_{n\cup\{p\}}\Big\downarrow\Big) \\
\overset{(**)}{=} \;& p[\mathrm{M}] : \Big({}^{o\cup r}a\,{}^{\{p\}}\Big\downarrow {}^{m}_{n\cup\{p\}}\Big\downarrow \;|||\; {}^{\{p\}}P\Big) \\
= \;& \begin{cases} p[\mathrm{M}] : (\{stop\} \;|||\; {}^{\{p\}}P) & (m,e,n\cup\{p\}) = (o\cup r, a, \{p\}) \\ \varnothing & \text{otherwise} \end{cases} \\
\overset{(5.8)}{=} \;& \begin{cases} \{p[\mathrm{M}] : {}^{\{p\}}P\} & e = a,\ m = o\cup r,\ n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
\overset{(5.29)}{=} \;& U
\end{aligned}
$$

- For $p \in m \cup n$: We have $U = \varnothing$ because of $p \in m \cup n$ and $p \notin o \cup r \cup s$, and hence

$$ U \quad = \quad \varnothing \quad = \quad (p[\mathrm{E}] : ({}^{o\cup r}a\,{}^{s\cup\{p\}} \;|||\; {}^{\{p\}}P))\Big\downarrow {}^{m}_{n}\Big\downarrow $$

4. By coinduction on the equivalence relation

$$ Rel \quad := \quad < \left\{ \begin{array}{c} ({}^{o*}(P' \,\square\, Q'),\ p[\mathrm{M}] : ({}^{o\cup\{p\}*}P' \;|||\; {}^{o\cup\{p\}*}Q')) \;: \\ P', Q' \in CProc_{\Sigma_0,S},\ p \notin FP(P') \cup FP(Q') \end{array} \right\} > $$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be an LTS label and P', Q' as in the definition of *Rel*. To improve readability we use the abbreviations $\tilde{o} := o \cup \{p\}$, $\tilde{m} := m \cup \{p\}$, $\tilde{n} := n \cup \{p\}$ and for arbitrary

sets $m', n' \in \mathcal{P}_{\text{fi}} S$ we define

$$
\begin{aligned}
& C(m', n') \\
& \quad := \quad (\tilde{o}{*}\, P' \;|||\; \tilde{o}{*}\, Q') \Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow \\
& \quad = \quad (\tilde{o}{*}\, P') \Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow \;|||\; \tilde{o}{*}\, Q' \;\cup\; \tilde{o}{*}\, P' \;|||\; (\tilde{o}{*}\, Q') \Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow \\
& \quad = \quad \begin{cases}
\tilde{o}{*} \left(P' \Big\downarrow \begin{smallmatrix} m' \backslash \tilde{o} \\ e \\ n' \backslash \tilde{o} \end{smallmatrix} \Big\downarrow \right) |||\; \tilde{o}{*}\, Q' \;\cup\; \tilde{o}{*}\, P' \;|||\; \tilde{o}{*} \left(Q' \Big\downarrow \begin{smallmatrix} m' \backslash \tilde{o} \\ e \\ n' \backslash \tilde{o} \end{smallmatrix} \Big\downarrow \right) & e \in \Sigma_\tau,\ \tilde{o} \subseteq m' \cap n' \\
P' \Big\downarrow \begin{smallmatrix} m' \backslash \tilde{o} \\ e \\ n' \end{smallmatrix} \Big\downarrow \;|||\; \tilde{o}{*}\, Q' \;\cup\; \tilde{o}{*}\, P' \;|||\; Q' \Big\downarrow \begin{smallmatrix} m' \backslash \tilde{o} \\ e \\ n' \end{smallmatrix} \Big\downarrow & e \in \Sigma,\ \tilde{o} \subseteq m' \backslash n' \\
\varnothing & \text{otherwise}
\end{cases}
\end{aligned}
$$

For the main derivation we make a case distinction:

- For $p \notin m \cup n$: The evaluation of $C(m', n')$ for different arguments yields

$$
\begin{aligned}
C(m, n) \quad &= \quad \varnothing \\
C(\tilde{m}, n) \quad &= \quad \begin{cases}
P' \Big\downarrow \begin{smallmatrix} m \backslash o \\ e \\ n \end{smallmatrix} \Big\downarrow \;|||\; \tilde{o}{*}\, Q' \;\cup\; \tilde{o}{*}\, P' \;|||\; Q' \Big\downarrow \begin{smallmatrix} m \backslash o \\ e \\ n \end{smallmatrix} \Big\downarrow & e \in \Sigma,\ o \subseteq m \backslash n \\
\varnothing & \text{otherwise}
\end{cases} \\
C(\tilde{m}, \tilde{n}) \quad &= \quad \begin{cases}
\tilde{o}{*} \left(P' \Big\downarrow \begin{smallmatrix} m \backslash o \\ e \\ n \backslash o \end{smallmatrix} \Big\downarrow \right) |||\; \tilde{o}{*}\, Q' & \\
\qquad \cup\; \tilde{o}{*}\, P' \;|||\; \tilde{o}{*} \left(Q' \Big\downarrow \begin{smallmatrix} m \backslash o \\ e \\ n \backslash o \end{smallmatrix} \Big\downarrow \right) & e \in \Sigma_\tau, o \subseteq m \cap n \\
\varnothing & \text{otherwise}
\end{cases}
\end{aligned}
$$

  For the verification of the bisimulation condition we start with the right side of the relation generators. (*) uses Eq. (5.18) to push down the occurrences of the state operator, which can be applied because of Impl. (5.4), (**) refers to Eq. (5.30) and Eq. (5.8) to eliminate blocked parts of the interleaving and to show that *Rel* applies.

$$
\begin{aligned}
& (p[\mathrm{M}] : (\tilde{o}{*}\, P' \;|||\; \tilde{o}{*}\, Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
& \quad = \quad p[\mathrm{M}] : C(m, n) \;\cup\; p[\mathrm{D}] : C(\tilde{m}, n) \;\cup\; p[\mathrm{M}] : C(\tilde{m}, \tilde{n}) \\
& \quad = \quad \begin{cases}
p[\mathrm{M}] : \left(\tilde{o}{*} \left(P' \Big\downarrow \begin{smallmatrix} m \backslash o \\ e \\ n \backslash o \end{smallmatrix} \Big\downarrow \right) |||\; \tilde{o}{*}\, Q' \right) & \\
\qquad \cup\, p[\mathrm{M}] : \left(\tilde{o}{*}\, P' \;|||\; \tilde{o}{*} \left(Q' \Big\downarrow \begin{smallmatrix} m \backslash o \\ e \\ n \backslash o \end{smallmatrix} \Big\downarrow \right) \right) & e \in \Sigma_\tau,\ o \subseteq m \cap n \\
p[\mathrm{D}] : \left(P' \Big\downarrow \begin{smallmatrix} m \backslash o \\ e \\ n \end{smallmatrix} \Big\downarrow \;|||\; \tilde{o}{*}\, Q' \right) & \\
\qquad \cup\, p[\mathrm{D}] : \left(\tilde{o}{*}\, P' \;|||\; Q' \Big\downarrow \begin{smallmatrix} m \backslash o \\ e \\ n \end{smallmatrix} \Big\downarrow \right) & e \in \Sigma,\ o \subseteq m \backslash n \\
\varnothing & \text{otherwise}
\end{cases}
\end{aligned}
$$

$$
\begin{array}{ll}
 & \begin{cases}
p[\mathrm{M}] : \left({}^{\tilde{o}*}\left(P' \Big\downarrow^{m\backslash o}_{n\backslash o}{}_{e} \right) \;|||\; {}^{\tilde{o}*} Q' \right) & \\
\qquad \cup\; p[\mathrm{M}] : \left({}^{\tilde{o}*} P' \;|||\; {}^{\tilde{o}*}\left(Q' \Big\downarrow^{m\backslash o}_{n\backslash o}{}_{e} \right) \right) & e \in \Sigma_\tau,\ o \subseteq m \cap n \\
P' \Big\downarrow^{m\backslash o}_{n}{}_{e} \;|||\; (p[\mathrm{D}] : {}^{\tilde{o}*} Q') & \\
\qquad \cup\; (p[\mathrm{D}] : {}^{\tilde{o}*} P') \;|||\; Q' \Big\downarrow^{m\backslash o}_{n}{}_{e} & e \in \Sigma,\ o \subseteq m\backslash n \\
\varnothing & \text{otherwise}
\end{cases} \\
\overset{(*)}{=} & \\
\overset{(**)}{\sim_{Rel}} & \begin{cases}
{}^{o*}\left(P' \Big\downarrow^{m\backslash o}_{n\backslash o}{}_{e} \;\Box\; Q' \right) & \\
\qquad \cup\; {}^{o*}\left(P' \;\Box\; Q' \Big\downarrow^{m\backslash o}_{n\backslash o}{}_{e} \right) & e \in \Sigma_\tau,\ o \subseteq m \cap n \\
P' \Big\downarrow^{m\backslash o}_{n}{}_{e} \;\cup\; Q' \Big\downarrow^{m\backslash o}_{n}{}_{e} & e \in \Sigma,\ o \subseteq m\backslash n \\
\varnothing & \text{otherwise}
\end{cases} \\
= & \begin{cases}
{}^{o*}\left((P' \;\Box\; Q') \Big\downarrow^{m\backslash o}_{n\backslash o}{}_{e} \right) & e \in \Sigma_\tau,\ o \subseteq m \cap n \\
(P' \;\Box\; Q') \Big\downarrow^{m\backslash o}_{n}{}_{e} & e \in \Sigma,\ o \subseteq m\backslash n \\
\varnothing & \text{otherwise}
\end{cases} \\
= & ({}^{o*}(P' \;\Box\; Q')) \Big\downarrow^{m}_{n}{}_{e}
\end{array}
$$

- For $p \in m \cup n$: The condition is directly entailed by the characteristic equation of the state operator and Impl. (5.3), together with the fact that

$$FP(P' \;\Box\; Q') \quad = \quad FP(P') \cup FP(Q') \qquad \Box$$

5.4.3 Proof (Page 98): 1. We prove the conjecture by referring to the last formulation of the monotonicity property. The proof consists of a coinduction on the equivalence relation

$$Rel \quad := \quad < \left\{ \begin{array}{c} (M' : P',\ M' : Q') \ : \\ M' \in \mathcal{M}_S,\ P', Q' \in CProc_{\Sigma_0,S}, \\ \exists\, M'' \in \mathcal{M}_S.\ (M'' \leqslant M' \wedge P' \sim_{M''} Q') \end{array} \right\} >$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be an LTS label, $M' \in \mathcal{M}_S,\ P', Q' \in CProc_{\Sigma_0,S}$ as in the definition of *Rel* and let $M'' \in \mathcal{M}_S$ be a second marking with $M'' \leqslant M'$ and $P' \sim_{M''} Q'$.

- For $(m, n) \in \mathrm{PotEn}(M')$: We first show an auxiliary statement that justifies the applicability of *Rel* in the final reasoning. From Lem. 4.1.6, 7 we can derive $(m, n) \in \mathrm{PotEn}(M'')$ which is used as (*). At (**) the condition of

Rel is satisfied because of Lem. 4.1.6, 8.

$$
\begin{array}{ll}
& M'' : P' = M'' : Q' \\
\Longrightarrow & (M'' : P') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow = (M'' : Q') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \\
\overset{(*)}{\Longleftrightarrow} & (M'' + (m,n)) : P' \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow = (M'' + (m,n)) : Q' \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \\
\overset{(**)}{\Longrightarrow} & (M' + (m,n)) : P' \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \sim_{Rel} (M' + (m,n)) : Q' \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \qquad (***)
\end{array}
$$

Finally the bisimulation condition can be verified as follows:

$$
\begin{array}{rll}
(M' : P') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow & = & (M' + (m,n)) : P' \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \\
& \overset{(***)}{\sim_{Rel}} & (M' + (m,n)) : Q' \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \\
& = & (M' : Q') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow
\end{array}
$$

- For $(m,n) \notin \mathrm{PotEn}(M')$:

$$
(M' : P') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \quad = \quad \varnothing \quad = \quad (M' : Q') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow
$$

2. By conduction on the equivalence relation

$$
Rel \quad := \quad < \{ (M' : M' : P',\ M' : P') \ :\ M' \in \mathcal{M}_S,\ P' \in CProc_{\Sigma_0,S} \} >
$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label and M', P' as in the definition of *Rel*.

- For $(m,n) \in \mathrm{PotEn}(M')$:

$$
\begin{array}{rll}
(M' : M' : P') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow & = & (M' + (m,n)) : (M' : P') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \\
& = & (M' + (m,n)) : (M' + (m,n)) : P \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \\
& \sim_{Rel} & (M' + (m,n)) : P \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \\
& = & (M' : P') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow
\end{array}
$$

- For $(m,n) \notin \mathrm{PotEn}(M')$:

$$
(M' : M' : P') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow \quad = \quad \varnothing \quad = \quad (M' : P') \left\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \right\downarrow
$$

3. By coinduction on the equivalence relation

$$
Rel \quad := \quad < \{ (M' : P',\ P') \ :\ M' \in \mathcal{M}_S,\ P' \in CProc_{\Sigma_0,S},\ FP(P') = \varnothing \} >
$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label and M', P' as in the definition of *Rel*.

- For $(m,n) \in \mathrm{PotEn}(M')$:

$$(M' : P')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad = \quad (M' + (m,n)) : P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \overset{(5.4)}{\sim_{Rel}} \quad P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$$

- For $(m,n) \notin \mathrm{PotEn}(M')$: By Lem. 4.1.6, 3 then we have $m \cup n \neq \varnothing$, which together with (5.3) and $FP(P') = \varnothing$ entails (*).

$$(M' : P')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad = \quad \varnothing \quad \overset{(*)}{=} \quad P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$$

4. We only show two of the implications, the other ones can be proved similarly.

Impl. (5.40): By coinduction on the equivalence relation

$$Rel \quad := \quad < \left\{ \begin{array}{c} (M' : (P' \,[\![\, L'[X] R' \,]\!]\, Q'),\ M' : (\tilde{P}' \,[\![\, L'[X] R' \,]\!]\, \tilde{Q}')) \ : \\ M' \in \mathcal{M}_S,\ P', Q', \tilde{P}', \tilde{Q}' \in CProc_{\Sigma_0,S}, \\ L', R' \in \mathrm{mult_{fi}}\, \mathcal{L}_{CCSP},\ P' \sim_{M'} \tilde{P}',\ Q' \sim_{M'} \tilde{Q}' \end{array} \right\} >$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label and M', P', Q', $\tilde{P}'$, $\tilde{Q}'$ as in the definition of *Rel*.

- For $e \in X$, $(m,n) \in \mathrm{PotEn}(M')$: We again start with proving an auxiliary statement that shows the applicability of the relation *Rel*. The I/O pair (m,n) is decomposed as in the characteristic equation of parallelism by $(m,n) = (m_1,n_1) + (m_2,n_2)$, and we use $(m_1,n_1) \in \mathrm{PotEn}(M')$ which follows from Lem. 4.1.6, 2 as (*). (**) holds because of Lem. 4.1.6, 9 and Impl. (5.33).

$$\begin{array}{ll} & M' : P' = M' : \tilde{P}' \\ \Longrightarrow & (M' : P')\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow = (M' : \tilde{P}')\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \\ \overset{(*)}{\Longleftrightarrow} & (M' + (m_1,n_1)) : P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow = (M' + (m_1,n_1)) : \tilde{P}' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \\ \overset{(**)}{\Longrightarrow} & (M' + (m,n)) : P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow = (M' + (m,n)) : \tilde{P}' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \qquad (***) \end{array}$$

The same derivation can be performed for Q', $\tilde{Q}'$ and the pair (m_2,n_2). Furthermore we have $P' \sim_{M'+(m,n)} \tilde{P}'$ and $Q' \sim_{M'+(m,n)} \tilde{Q}'$ by (**). Altogether this justifies the applicability of *Rel* to the expressions the

parallel composition breaks down to, and is used as (***):

$$
\begin{aligned}
&(M' : (P' \,|[\, L'[X]R' \,]|\, Q'))\Big\downarrow^{m}_{n}_{e}\Big\downarrow \\
&\quad = \quad (M' + (m,n)) : (P' \,|[\, L'[X]R' \,]|\, Q')\Big\downarrow^{m}_{n}_{e}\Big\downarrow \\
&\quad = \quad (M' + (m,n)) : \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} \begin{array}{ll} & P'\Big\downarrow^{m_1}_{n_1}_{e}\Big\downarrow \,|[\, L'[X]R' \,]|\, Q'\Big\downarrow^{m_2}_{n_2}_{e}\Big\downarrow \\ \cup & P' \,|[\, L'\Big\downarrow^{m_1}_{n_1}_{e}\Big\downarrow [X]R' \,]|\, Q'\Big\downarrow^{m_2}_{n_2}_{e}\Big\downarrow \\ \cup & P'\Big\downarrow^{m_1}_{n_1}_{e}\Big\downarrow \,|[\, L'[X]R'\Big\downarrow^{m_2}_{n_2}_{e}\Big\downarrow \,]|\, Q' \end{array} \\
&\overset{(***)}{\sim_{Rel}} \quad (M' + (m,n)) : \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} \begin{array}{ll} & \tilde{P}'\Big\downarrow^{m_1}_{n_1}_{e}\Big\downarrow \,|[\, L'[X]R' \,]|\, \tilde{Q}'\Big\downarrow^{m_2}_{n_2}_{e}\Big\downarrow \\ \cup & \tilde{P}' \,|[\, L'\Big\downarrow^{m_1}_{n_1}_{e}\Big\downarrow [X]R' \,]|\, \tilde{Q}'\Big\downarrow^{m_2}_{n_2}_{e}\Big\downarrow \\ \cup & \tilde{P}'\Big\downarrow^{m_1}_{n_1}_{e}\Big\downarrow \,|[\, L'[X]R'\Big\downarrow^{m_2}_{n_2}_{e}\Big\downarrow \,]|\, \tilde{Q}' \end{array} \\
&\quad = \quad (M' : (\tilde{P}' \,|[\, L'[X]R' \,]|\, \tilde{Q}'))\Big\downarrow^{m}_{n}_{e}\Big\downarrow
\end{aligned}
$$

- For $e \notin X$, $(m,n) \in \mathrm{PotEn}(M')$: The transformation (***) can be justified as for the first case $e \in X$.

$$
\begin{aligned}
&(M' : (P' \,|[\, L'[X]R' \,]|\, Q'))\Big\downarrow^{m}_{n}_{e}\Big\downarrow \\
&\quad = \quad (M' + (m,n)) : (P' \,|[\, L'[X]R' \,]|\, Q')\Big\downarrow^{m}_{n}_{e}\Big\downarrow \\
&\quad = \quad (M' + (m,n)) : (P'\Big\downarrow^{m}_{n}_{e}\Big\downarrow \,|[\, L'[X]R' \,]|\, Q') \\
&\qquad\qquad \cup \quad (M' + (m,n)) : (P' \,|[\, L'[X]R' \,]|\, Q'\Big\downarrow^{m}_{n}_{e}\Big\downarrow) \\
&\overset{(***)}{\sim_{Rel}} \quad (M' + (m,n)) : (\tilde{P}'\Big\downarrow^{m}_{n}_{e}\Big\downarrow \,|[\, L'[X]R' \,]|\, \tilde{Q}') \\
&\qquad\qquad \cup \quad (M' + (m,n)) : (\tilde{P}' \,|[\, L'[X]R' \,]|\, \tilde{Q}'\Big\downarrow^{m}_{n}_{e}\Big\downarrow) \\
&\quad = \quad (M' : (\tilde{P}' \,|[\, L'[X]R' \,]|\, \tilde{Q}'))\Big\downarrow^{m}_{n}_{e}\Big\downarrow
\end{aligned}
$$

- For $(m,n) \in \mathrm{PotEn}(M')$:

$$
\begin{aligned}
(M' : (P' \,|[\, L'[X]R' \,]|\, Q'))\Big\downarrow^{m}_{n}_{e}\Big\downarrow \quad &= \quad \varnothing \\
&= \quad (M' : (\tilde{P}' \,|[\, L'[X]R' \,]|\, \tilde{Q}'))\Big\downarrow^{m}_{n}_{e}\Big\downarrow
\end{aligned}
$$

Impl. (5.43): By coinduction on the equivalence relation

$$
Rel \quad := \quad < \left\{ \begin{array}{c} (\tilde{M}' : p[v'] : P',\ \tilde{M}' : p[v'] : Q') \ : \\ M', \tilde{M}' \in \mathcal{M}_S,\ P', Q' \in \mathit{CProc}_{\Sigma_0,S},\ v' \in \{\mathrm{E},\mathrm{M},\mathrm{D}\}, \\ M'|_{S\setminus\{p\}} = \tilde{M}'|_{S\setminus\{p\}},\ M'(p) = v',\ P' \sim_{M'} Q' \end{array} \right\} >
$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label and M', $\tilde{M}'$, P', Q', v' as in the definition of *Rel*. We abbreviate $\tilde{M}' + (m,n)$ with M^+ in the sequel.

- For $p \notin m \cup n$, $(m,n) \in \mathrm{PotEn}(\tilde{M}')$:

$$
\begin{aligned}
&(\tilde{M}' : p[v'] : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= \quad M^+ : (p[v'] : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= \quad M^+ : p[v'] : P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad \cup \\
&\qquad\qquad M^+ : \begin{cases} p[\mathrm{M}] : P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow & v' = \mathrm{E} \\ p[\mathrm{D}] : P' \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \end{smallmatrix} \Big\downarrow & \\ \qquad \cup\, p[\mathrm{M}] : P' \Big\downarrow \begin{smallmatrix} m \cup \{p\} \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow & v' = \mathrm{M} \\ \varnothing & v' = \mathrm{D} \end{cases} \\
&=: \quad U
\end{aligned}
$$

To show that *Rel* can be applied at this point, we have to check its conditions for the different parts of U. The first expression is independent of v'. (*) holds because M' and $\tilde{M}'$ only differ in $p \notin m \cup n$ and because of Eq. (4.1). By Eq. (4.2) and Eq. (4.3) likewise the markings $M' + (m,n)$ and M^+ coincide up to the value at p, and we have $(M' + (m,n))(p) = M'(p) = v'$. In total the conditions of *Rel* are satisfied at (**).

$$
\begin{aligned}
& M' : P' = M' : Q' \\
& \quad \Longrightarrow \quad (M' : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow = (M' : Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
& \quad \overset{(*)}{\Longleftrightarrow} \quad (M' + (m,n)) : P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow = (M' + (m,n)) : \tilde{P}' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
& \quad \overset{(**)}{\Longrightarrow} \quad M^+ : p[v'] : P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \sim_{Rel} M^+ : p[v'] : Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

The other parts are treated in similar manner by a case distinction on the value v':

- For $v' = \mathrm{E}$: $(m, n \cup \{p\}) \in \mathrm{PotEn}(M')$ holds at (*) because of $M'(p) = \mathrm{E}$, and we have $(M' + (m, n \cup \{p\}))(p) = \mathrm{M}$ at (**) by Def. 4.1.2.

$$
\begin{aligned}
& M' : P' = M' : Q' \\
& \quad \Longrightarrow \quad (M' : P') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow = (M' : Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow \\
& \quad \overset{(*)}{\Longleftrightarrow} \quad (M' + (m, n \cup \{p\})) : P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow \\
& \qquad\qquad = (M' + (m, n \cup \{p\})) : \tilde{P}' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow \\
& \quad \overset{(**)}{\Longrightarrow} \quad M^+ : p[\mathrm{M}] : P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow \sim_{Rel} M^+ : p[\mathrm{M}] : Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \cup \{p\} \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

- For $v' = \mathrm{M}$: The two remaining expressions of U belong to this case. For the penultimate one we again have $(m \cup \{p\}, n) \in \mathrm{PotEn}(M')$

because of $M'(p) = \mathrm{M}$ and $(M' + (m \cup \{p\}, n))(p) = \mathrm{D}$.

$$
\begin{aligned}
& M' : P' = M' : Q' \\
\Longrightarrow\quad & (M' : P')\Big\downarrow^{m\cup\{p\}}_{e\ n}\Big\downarrow = (M' : Q')\Big\downarrow^{m\cup\{p\}}_{e\ n}\Big\downarrow \\
\Longleftrightarrow\quad & (M' + (m \cup \{p\}, n)) : P'\Big\downarrow^{m\cup\{p\}}_{e\ n}\Big\downarrow \\
& \qquad = (M' + (m \cup \{p\}, n)) : \tilde{P}'\Big\downarrow^{m\cup\{p\}}_{e\ n}\Big\downarrow \\
\Longrightarrow\quad & M^+ : p[\mathrm{D}] : P'\Big\downarrow^{m\cup\{p\}}_{e\ n}\Big\downarrow \sim_{Rel} M^+ : p[\mathrm{D}] : Q'\Big\downarrow^{m\cup\{p\}}_{e\ n}\Big\downarrow
\end{aligned}
$$

The last part is completed with $(m \cup \{p\}, n \cup \{p\}) \in \mathrm{PotEn}(M')$ because of $M'(p) = \mathrm{M}$ and $(M' + (m \cup \{p\}, n \cup \{p\}))(p) = \mathrm{M}$.

$$
\begin{aligned}
& M' : P' = M' : Q' \\
\Longrightarrow\quad & (M' : P')\Big\downarrow^{m\cup\{p\}}_{e\ n\cup\{p\}}\Big\downarrow = (M' : Q')\Big\downarrow^{m\cup\{p\}}_{e\ n\cup\{p\}}\Big\downarrow \\
\Longleftrightarrow\quad & (M' + (m \cup \{p\}, n \cup \{p\})) : P'\Big\downarrow^{m\cup\{p\}}_{e\ n\cup\{p\}}\Big\downarrow \\
& \qquad = (M' + (m \cup \{p\}, n \cup \{p\})) : \tilde{P}'\Big\downarrow^{m\cup\{p\}}_{e\ n\cup\{p\}}\Big\downarrow \\
\Longrightarrow\quad & M^+ : p[\mathrm{M}] : P'\Big\downarrow^{m\cup\{p\}}_{e\ n\cup\{p\}}\Big\downarrow \sim_{Rel} M^+ : p[\mathrm{M}] : Q'\Big\downarrow^{m\cup\{p\}}_{e\ n\cup\{p\}}\Big\downarrow
\end{aligned}
$$

The derivation is concluded as follows:

$$
\begin{aligned}
U \quad \sim_{Rel} \quad & M^+ : p[v'] : Q'\Big\downarrow^{m}_{e\ n}\Big\downarrow \quad \cup \\
& M^+ : \begin{cases} p[\mathrm{M}] : Q'\Big\downarrow^{m}_{e\ n\cup\{p\}}\Big\downarrow & v' = \mathrm{E} \\ p[\mathrm{D}] : Q'\Big\downarrow^{m\cup\{p\}}_{e\ n}\Big\downarrow & \\ \qquad \cup\ p[\mathrm{M}] : Q'\Big\downarrow^{m\cup\{p\}}_{e\ n\cup\{p\}}\Big\downarrow & v' = \mathrm{M} \\ \varnothing & v' = \mathrm{D} \end{cases} \\
= \quad & (\tilde{M}' : p[v'] : Q')\Big\downarrow^{m}_{e\ n}\Big\downarrow
\end{aligned}
$$

- For $p \in m \cup n$, $(m, n) \in \mathrm{PotEn}(\tilde{M}')$:

$$
\begin{aligned}
(\tilde{M}' : p[v'] : P')\Big\downarrow^{m}_{e\ n}\Big\downarrow \quad &= \quad M^+ : (p[v'] : P')\Big\downarrow^{m}_{e\ n}\Big\downarrow \\
&= \quad \varnothing \quad = \quad (\tilde{M}' : p[v'] : Q')\Big\downarrow^{m}_{e\ n}\Big\downarrow
\end{aligned}
$$

- For $(m, n) \notin \mathrm{PotEn}(\tilde{M}')$:

$$
(\tilde{M}' : p[v'] : P')\Big\downarrow^{m}_{e\ n}\Big\downarrow \quad = \quad \varnothing \quad = \quad (\tilde{M}' : p[v'] : Q')\Big\downarrow^{m}_{e\ n}\Big\downarrow
$$

5. Omitted.

6. Omitted. □

5.5.2 Proof (Page 103): We follow the case distinction of the lemma:

- For $a \in X$: By coinduction on the equivalence relation

$$Rel \quad := \quad < \left\{ \begin{array}{c} (M' : (P' \,[\![\, L'[X] R' \,]\!]\, ({}^{o}\, a\, {}^{r} \;|||\; Q')), \\ M' : (mrg\big((o,a,r), L'\big) \;|||\; (P' \,[\![\, L'[X] R' \uplus \{(o,a,r)\} \,]\!]\, Q'))) \;: \\ M' \in \mathcal{M}_S,\; P', Q' \in CProc_{\Sigma_0,S},\; L', R' \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP}, \\ \text{each element of } L' \text{ duplicable} \end{array} \right\} >$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and M', P', Q', L', R' as in the definition of *Rel*. In the sequel we use the following abbreviations to make the terms fit the lines:

$$\begin{array}{ll} M^+ := M' + (m,n) & \tilde{R} := R' \uplus \{(o,a,r)\} \\ \tilde{R}^- := \tilde{R} \backslash \{(m_2, e, n_2)\} & R'^- := R' \backslash \{(m_2, e, n_2)\} \\ R'^{-+} := R'^- \uplus \{(o,a,r)\} & MR := mrg\big((o,a,r), L'\big) \end{array}$$

Furthermore we assume $(m,n) \in \mathrm{PotEn}(M')$, as the other case can be proved immediately and as always.

  - For $e \in X$: An argument that is used frequently in the reasoning as (***) is that whenever we have a duplicable I/O pair $(m', n') \leqslant (m, n)$ (the ordering on I/O pairs is defined in Lem. 4.1.4) the property $(m', n') \notin \mathrm{PotEn}(M^+)$ holds by Lem. 4.1.6:

$$\begin{array}{rcl} (m',n') \leqslant (m,n) & \Longrightarrow & \mathrm{PotEn}(M^+) \subseteq \mathrm{PotEn}(M' + (m',n')) \\ m' \ominus n' \neq \varnothing & \Longrightarrow & (m',n') \notin \mathrm{PotEn}(M' + (m',n')) \\ & \Longrightarrow & (m',n') \notin \mathrm{PotEn}(M^+) \end{array}$$

    In particular we use (***) for the pair (o, r) and the I/O of elements of L', which are duplicable by premiss. A further implication is that for every I/O pair $(m'', n'') \geqslant (m', n')$ we have $(m'', n'') \notin \mathrm{PotEn}(M^+)$ in this case.

    We start by analysing the left component of an arbitrary generator of *Rel*: (*) in the derivation holds because of the characteristic equation of transi-

tions, and because of the properties of interleaving Lem. 5.2.1.

$$
\begin{aligned}
&(M' : (P' \,|[\, L'[X]R' \,]|\, ({}^{o}a^{\,r} \;|||\; Q')))\Big\downarrow^{m}_{n}\Big\downarrow^{e}\\
&\quad = \quad M^{+} : (P' \,|[\, L'[X]R' \,]|\, ({}^{o}a^{\,r} \;|||\; Q'))\Big\downarrow^{m}_{n}\Big\downarrow^{e}\\
&\quad = \quad M^{+} : \bigcup_{\substack{(m,n)=\\(m_1,n_1)+\\(m_2,n_2)}} P'\Big\downarrow^{m_1}_{n_1}\Big\downarrow^{e} |[\, L'[X]R' \,]|\, ({}^{o}a^{\,r} \;|||\; Q')\Big\downarrow^{m_2}_{n_2}\Big\downarrow^{e}\\
&\qquad\qquad\qquad \cup \quad P' \,|[\, L'\Big\downarrow^{m_1}_{n_1}\Big\downarrow^{e} [X]R' \,]|\, ({}^{o}a^{\,r} \;|||\; Q')\Big\downarrow^{m_2}_{n_2}\Big\downarrow^{e}\\
&\qquad\qquad\qquad \cup \quad P'\Big\downarrow^{m_1}_{n_1}\Big\downarrow^{e} |[\, L'[X]R'\Big\downarrow^{m_2}_{n_2}\Big\downarrow^{e} \,]|\, ({}^{o}a^{\,r} \;|||\; Q')\\
&\overset{(*)}{=} \quad M^{+} : \bigcup_{\substack{(m,n)=\\(m_1,n_1)+\\(o,r)\\a=e}} \Big(\underbrace{P'\Big\downarrow^{m_1}_{n_1}\Big\downarrow^{e} |[\, L'[X]R' \,]|\, Q'}_{=:U_2} \;\cup\; \underbrace{P' \,|[\, L'\Big\downarrow^{m_1}_{n_1}\Big\downarrow^{e} [X]R' \,]|\, Q'}_{=:U_1} \Big)\\
&\qquad\qquad \cup \quad M^{+} : \bigcup_{\substack{(m,n)=\\(m_1,n_1)+\\(m_2,n_2)}} P'\Big\downarrow^{m_1}_{n_1}\Big\downarrow^{e} |[\, L'[X]R' \,]|\, \Big({}^{o}a^{\,r} \;|||\; Q'\Big\downarrow^{m_2}_{n_2}\Big\downarrow^{e}\Big)\\
&\qquad\qquad\qquad\qquad\qquad \cup \quad P' \,|[\, L'\Big\downarrow^{m_1}_{n_1}\Big\downarrow^{e} [X]R' \,]|\, \Big({}^{o}a^{\,r} \;|||\; Q'\Big\downarrow^{m_2}_{n_2}\Big\downarrow^{e}\Big)\\
&\qquad\qquad\qquad\qquad\qquad \cup \quad \underbrace{P'\Big\downarrow^{m_1}_{n_1}\Big\downarrow^{e} |[\, L'[X]R'\Big\downarrow^{m_2}_{n_2}\Big\downarrow^{e} \,]|\, ({}^{o}a^{\,r} \;|||\; Q')}_{=:U_3}
\end{aligned}
$$

∗ The part U_1 of the expression derived so far corresponds to the merged transitions $mrg\big((o,a,r),L'\big)$, and as an intermediate statement we first show that

$$
\bigcup_{\substack{(m,n)=\\(m_1,n_1)+(o,r)\\a=e}} U_1 \quad \sim_{M^{+}} \quad mrg\big((o,a,r),L'\big)\Big\downarrow^{m}_{n}\Big\downarrow^{e} \;|||\; (P' \,|[\, L'[X]\tilde{R} \,]|\, Q') \tag{A.2}
$$

Therefore, we concentrate on the merge expression. In the transformations the properties of interleaving from Lem. 5.2.1 are used implicitly. We use (***) in the scope of a union enforcing the condition $(m,n) = (o,r) + (o',r')$, i.e. $(o,r) \leqslant (m,n)$ together with Eq. (5.45): By this equation we have ${}^{\tilde{o}''}a^{\,\tilde{r}''} \sim_{M^{+}} stop$.

$$
\begin{aligned}
&mrg\big((o,a,r),L'\big)\Big\downarrow^{m}_{n}\Big\downarrow^{e}\\
&\quad = \quad \Big(\;|||_{\substack{(o',a,r')\in L'\\(o'',r'')=(o,r)+(o',r')}} {}^{o''}a^{\,r''} \Big)\Big\downarrow^{m}_{n}\Big\downarrow^{e}\\
&\quad = \quad \bigcup_{\substack{(o',a,r')\in L'\\(o'',r'')=(o,r)+(o',r')}} \Big({}^{o''}a^{\,r''}\Big\downarrow^{m}_{n}\Big\downarrow^{e} \;|||\; |||_{\substack{(\tilde{o}',a,\tilde{r}')\in L'\\(\tilde{o}'',\tilde{r}'')=(o,r)+(\tilde{o}',\tilde{r}')\\(\tilde{o}',\tilde{r}')\neq(o',r')}} {}^{\tilde{o}''}a^{\,\tilde{r}''} \Big)\\
&\quad = \quad \bigcup_{\substack{(o',a,r')\in L'\\(m,n)=(o,r)+(o',r')\\a=e}} \Big(\{stop\} \;|||\; |||_{\substack{(\tilde{o}',a,\tilde{r}')\in L'\\(\tilde{o}'',\tilde{r}'')=(o,r)+(\tilde{o}',\tilde{r}')\\(\tilde{o}',\tilde{r}')\neq(o',r')}} {}^{\tilde{o}''}a^{\,\tilde{r}''} \Big)
\end{aligned}
$$

$$\overset{(***)}{\sim_{M^+}} \bigcup_{\substack{(o',a,r')\in L' \\ (m,n)=(o,r)+(o',r') \\ a=e}} \{stop\}$$

We then use this expression to prove Eq. (A.2), again using (***) together with Impl. (5.44):

$$\begin{aligned}
& mrg\big((o,a,r),L'\big)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix}\Big\downarrow \;|||\; (P' \;|[\, L'[X]\tilde{R}\,]|\; Q') \\
& \quad \sim_{M^+} \bigcup_{\substack{(o',a,r')\in L' \\ (m,n)=(o,r)+(o',r') \\ a=e}} \{P' \;|[\, L'[X]\tilde{R}\,]|\; Q'\} \\
& \quad \overset{(5.44)}{\sim_{M^+}} \bigcup_{\substack{(o',a,r')\in L' \\ (m,n)=(o,r)+(o',r') \\ a=e}} \{P' \;|[\, L'\backslash\{(o',a,r')\}[X]R'\,]|\; Q'\} \\
& \quad = \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(o,r) \\ a=e}} P' \;|[\, L'\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow [X]R'\,]|\; Q' \quad = \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(o,r) \\ a=e}} U_1
\end{aligned}$$

$*$ The next sub-problem we tackle is the combination of the parts U_2 and U_3, which in the end leads us to the following relation:

$$\begin{aligned}
& M^+ : \Big(\bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(o,r) \\ a=e}} U_2 \;\cup \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(m_2,n_2)}} U_3 \Big) \\
& \quad \sim_{Rel} \quad M^+ : \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(m_2,n_2)}} MR \;|||\; \Big(P'\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow \;|[\, L'[X]\tilde{R}\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow \,]|\; Q' \Big)
\end{aligned} \tag{A.3}$$

To show Eq. (A.3) we start with the right side. As (*) we use the observation

$$\begin{aligned}
\tilde{R}\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow \quad &= \quad (R' \uplus \{(o,a,r)\})\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow \\
&= \bigcup_{(o,a,r)=(m_2,e,n_2)} \{R'\} \;\cup \bigcup_{(m_2,e,n_2)\in R'} \{\tilde{R}\backslash\{(m_2,e,n_2)\}\} \quad (*)
\end{aligned}$$

The derivation is performed as follows:

$$\begin{aligned}
& M^+ : \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(m_2,n_2)}} MR \;|||\; \Big(P'\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow \;|[\, L'[X]\tilde{R}\Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix}\Big\downarrow \,]|\; Q' \Big) \\
& \overset{(*)}{=} \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(o,r) \\ a=e}} M^+ : \Big(MR \;|||\; \Big(P'\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow \;|[\, L'[X]R'\,]|\; Q' \Big)\Big) \\
& \qquad \cup \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(m_2,n_2) \\ (m_2,e,n_2)\in R'}} M^+ : \Big(MR \;|||\; \Big(P'\Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix}\Big\downarrow \;|[\, L'[X]\tilde{R}^-\,]|\; Q' \Big)\Big)
\end{aligned}$$

For the first part of the expression the union ensures $(o,r) \leqslant (m,n)$ and thus the applicability of (***): We have $M^+ : MR = \mathit{stop}$ by Eq. (5.45).

$$
\begin{aligned}
= \quad & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(o,r) \\ a=e}} M^+ : \Big(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; \| [\![L'[X]R']\!] \| \; Q' \Big) \\
& \cup \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(m_2,n_2) \\ (m_2,e,n_2)\in R'}} M^+ : \Big(MR \;|||\; \Big(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; \| [\![L'[X]\tilde{R}^-]\!] \| \; Q' \Big) \Big)
\end{aligned}
$$

At this point the buffer $\tilde{R}^- = (R' \uplus \{o,a,r\}) \backslash \{(m_2,e,n_2)\}$ can be replaced with $R'^{-+} = R' \backslash \{(m_2,e,n_2)\} \uplus \{(o,a,r)\}$, because for the only case in which this is potentially a different multiset, namely for $(o,a,r) = (m_2,e,n_2)$, we have $(o,r) \leqslant (m,n)$ and can apply (***) and Impl. (5.44) to justify the modification.

$$
\begin{aligned}
= \quad & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(o,r) \\ a=e}} M^+ : U_2 \\
& \cup \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(m_2,n_2) \\ (m_2,e,n_2)\in R'}} M^+ : \Big(MR \;|||\; \Big(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; \| [\![L'[X]R'^{-+}]\!] \| \; Q' \Big) \Big) \\
\sim_{\mathit{Rel}} \quad & \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(o,r) \\ a=e}} M^+ : U_2 \\
& \cup \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(m_2,n_2) \\ (m_2,e,n_2)\in R'}} M^+ : \Big(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; \| [\![L'[X]R'^-]\!] \| \, ({}^o a^{\,r} \;|||\; Q') \Big) \\
= \quad & M^+ : \Big(\bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(o,r) \\ a=e}} U_2 \\
& \cup \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(m_2,n_2)}} P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \; \| [\![L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow]\!] \| \, ({}^o a^{\,r} \;|||\; Q') \Big) \\
= \quad & M^+ : \Big(\bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(o,r) \\ a=e}} U_2 \;\cup \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+(m_2,n_2)}} U_3 \Big)
\end{aligned}
$$

$*$ For the remaining two parts of the main expression we can directly apply the relation Rel. At (*) the merge process $\mathit{mrg}\big((o,a,r), L' \big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \big\downarrow\big)$ can be replaced with $\mathit{mrg}\big((o,a,r), L'\big)$, because we have $(m_1,n_1) \notin \mathrm{PotEn}(M^+)$ by (***), and in the only situation in which the merge term is relevant

$((m_1, e, n_1) \in L')$ each I/O pair $(o'', r'') \geqslant (m_1, n_1)$ is not potentially enabled either: $(o'', r'') \notin \mathrm{PotEn}(M^+)$. This entails $M^+ : {}^{o''}a^{\,r''} = \mathit{stop}$.

$$
\begin{array}{rrl}
M^+ : & \bigcup\limits_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [\![L'[X]R']\!] \left({}^{o}a^{\,r} \;|||\; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \right) \\
& \cup & P' \,[\![L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R']\!] \left({}^{o}a^{\,r} \;|||\; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \right) \\
\sim_{\mathit{Rel}} \quad M^+ : & \bigcup\limits_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & \mathit{mrg}\big((o,a,r), L'\big) \\
& & \quad ||| \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [\![L'[X]\tilde{R}]\!] \; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \right) \\
& \cup & \mathit{mrg}\big((o,a,r), L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \big) \\
& & \quad ||| \left(P' \,[\![L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]\tilde{R}]\!] \; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \right) \\
\overset{(*)}{=} \quad M^+ : & \bigcup\limits_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & MR ||| \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [\![L'[X]\tilde{R}]\!] \; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \right) \\
& \cup & MR ||| \left(P' \,[\![L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]\tilde{R}]\!] \; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \right)
\end{array}
$$

The pieces are put together to conclude the reasoning:

$$
\begin{array}{rrl}
M^+ : & \bigcup\limits_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (o,r) \\ a=e}} & \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [\![L'[X]R']\!] \; Q' \;\cup\; P' \,[\![L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R']\!] \; Q' \right) \\
\cup \quad M^+ : & \bigcup\limits_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [\![L'[X]R']\!] \left({}^{o}a^{\,r} \;|||\; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \right) \\
& \cup & P' \,[\![L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R']\!] \left({}^{o}a^{\,r} \;|||\; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \right) \\
& \cup & P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [\![L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow]\!] \; ({}^{o}a^{\,r} \;|||\; Q') \\
\sim_{\mathit{Rel}} \quad M^+ : & & \left(MR \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \;|||\; (P' \,[\![L'[X]\tilde{R}]\!] \; Q') \right) \\
\cup \quad M^+ : & \bigcup\limits_{\substack{(m,n)= \\ (m_1,n_1)+(m_2,n_2)}} & MR ||| \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [\![L'[X]\tilde{R} \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow]\!] \; Q' \right) \\
\cup \quad M^+ : & \bigcup\limits_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & MR ||| \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [\![L'[X]\tilde{R}]\!] \; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \right) \\
& \cup & MR ||| \left(P' \,[\![L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]\tilde{R}]\!] \; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \right) \\
= \quad M^+ : & & \left(MR \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \;|||\; (P' \,[\![L'[X]\tilde{R}]\!] \; Q') \right) \\
\cup \quad M^+ : & & \left(MR \;|||\; (P' \,[\![L'[X]\tilde{R}]\!] \; Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \right) \\
= \quad M^+ : & & (MR \;|||\; (P' \,[\![L'[X]\tilde{R}]\!] \; Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
= \quad (M' : & & (MR \;|||\; (P' \,[\![L'[X]\tilde{R}]\!] \; Q'))) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{array}
$$

– For $e \notin X$: Note that this entails $a \neq e$ which is used as (*).

$$
\begin{array}{rl}
& (M' : (P' \,[\![\, L'[X]R' \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q'))) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
= & M^+ : (P' \,[\![\, L'[X]R' \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
= & M^+ : \Big(P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]R' \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q') \Big) \\
& \quad \cup \quad M^+ : \Big(P' \,[\![\, L'[X]R' \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \Big) \\
\overset{(*)}{=} & M^+ : \Big(P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]R' \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q') \Big) \\
& \quad \cup \quad M^+ : \Big(P' \,[\![\, L'[X]R' \,]\!]\, \Big({}^{o}a^{\,r} \;|||\; Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \Big) \Big) \\
\sim_{Rel} & M^+ : \Big(MR \;|||\; \Big(P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]\tilde{R} \,]\!]\, Q' \Big) \Big) \\
& \quad \cup \quad M^+ : \Big(MR \;|||\; \Big(P' \,[\![\, L'[X]\tilde{R} \,]\!]\, Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \Big) \Big) \\
= & M^+ : \Big(MR \;|||\; (P' \,[\![\, L'[X]\tilde{R} \,]\!]\, Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \Big) \\
\overset{(*)}{=} & M^+ : (MR \;|||\; (P' \,[\![\, L'[X]\tilde{R} \,]\!]\, Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
= & (M' : (MR \;|||\; (P' \,[\![\, L'[X]\tilde{R} \,]\!]\, Q'))) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{array}
$$

• For $a \notin X$: By coinduction on the equivalence relation

$$
Rel \quad := \quad < \left\{ \begin{array}{c} (M' : (P' \,[\![\, L'[X]R' \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q')), \\ M' : ({}^{o}a^{\,r} \;|||\; (P' \,[\![\, L'[X]R' \,]\!]\, Q'))) \;: \\ M' \in \mathcal{M}_S,\; P', Q' \in \mathit{CProc}_{\Sigma_0,S},\; L', R' \in \mathrm{mult}_{\mathrm{fi}}\, \mathcal{L}_{CCSP} \end{array} \right\} >
$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and M', P', Q', L', R' as in the definition of *Rel*. Again we only consider the interesting case $(m, n) \in \mathrm{PotEn}(M')$, and we use the abbreviation $M^+ := M' + (m, n)$.

– For $e \in X$: Note that this entails $a \neq e$ which is used as (*).

$$
\begin{array}{rl}
& (M' : (P' \,[\![\, L'[X]R' \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q'))) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
= & M^+ : (P' \,[\![\, L'[X]R' \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
= & M^+ : \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} \begin{array}{ll} & P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]R' \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q') \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\ \cup & P' \,[\![\, L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R' \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q') \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \\ \cup & P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q') \end{array} \\
\overset{(*)}{=} & M^+ : \bigcup_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} \begin{array}{ll} & P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]R' \,]\!]\, \Big({}^{o}a^{\,r} \;|||\; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \Big) \\ \cup & P' \,[\![\, L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R' \,]\!]\, \Big({}^{o}a^{\,r} \;|||\; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \Big) \\ \cup & P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \,[\![\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]\!]\, ({}^{o}a^{\,r} \;|||\; Q') \end{array}
\end{array}
$$

$$
\begin{array}{rll}
\sim_{Rel} & M^+ : \bigcup\limits_{\substack{(m,n)= \\ (m_1,n_1)+ \\ (m_2,n_2)}} & {}^{o}a^{r} \;|||\; \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \;|[\, L'[X]R' \,]|\; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow\right) \\
& \cup & {}^{o}a^{r} \;|||\; \left(P' \;|[\, L' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow [X]R' \,]|\; Q' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow\right) \\
& \cup & {}^{o}a^{r} \;|||\; \left(P' \Big\downarrow \begin{smallmatrix} m_1 \\ e \\ n_1 \end{smallmatrix} \Big\downarrow \;|[\, L'[X]R' \Big\downarrow \begin{smallmatrix} m_2 \\ e \\ n_2 \end{smallmatrix} \Big\downarrow \,]|\; Q'\right) \\
= & M^+ : \left({}^{o}a^{r} \;|||\; (P' \;|[\, L'[X]R' \,]|\; Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right) \\
\overset{(*)}{=} & M^+ : ({}^{o}a^{r} \;|||\; (P' \;|[\, L'[X]R' \,]|\; Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
= & (M' : ({}^{o}a^{r} \;|||\; (P' \;|[\, L'[X]R' \,]|\; Q'))) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{array}
$$

– For $e \notin X$:

$$
\begin{array}{rll}
& (M' : (P' \;|[\, L'[X]R' \,]|\; ({}^{o}a^{r} \;|||\; Q'))) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
= & M^+ : (P' \;|[\, L'[X]R' \,]|\; ({}^{o}a^{r} \;|||\; Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
= & M^+ : \left(P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \;|[\, L'[X]R' \,]|\; ({}^{o}a^{r} \;|||\; Q')\right) \\
& \cup \quad M^+ : \left(P' \;|[\, L'[X]R' \,]|\; ({}^{o}a^{r} \;|||\; Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right) \\
= & M^+ : \left(P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \;|[\, L'[X]R' \,]|\; ({}^{o}a^{r} \;|||\; Q')\right) \\
& \cup \quad M^+ : \left(P' \;|[\, L'[X]R' \,]|\; \left({}^{o}a^{r} \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \;|||\; Q'\right)\right) \\
& \cup \quad M^+ : \left(P' \;|[\, L'[X]R' \,]|\; \left({}^{o}a^{r} \;|||\; Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right)\right) \\
\sim_{Rel} & M^+ : \left({}^{o}a^{r} \;|||\; \left(P' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \;|[\, L'[X]R' \,]|\; Q'\right)\right) \\
& \cup \quad M^+ : \left({}^{o}a^{r} \;|||\; \left(P' \;|[\, L'[X]R' \,]|\; Q' \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right)\right) \\
& \cup \quad \bigcup\limits_{(o,a,r)=(m,e,n)} \{M^+ : (P' \;|[\, L'[X]R' \,]|\; Q')\} \\
= & M^+ : \left({}^{o}a^{r} \;|||\; (P' \;|[\, L'[X]R' \,]|\; Q') \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right) \\
& \cup \quad M^+ : \left({}^{o}a^{r} \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \;|||\; (P' \;|[\, L'[X]R' \,]|\; Q')\right) \\
= & M^+ : ({}^{o}a^{r} \;|||\; (P' \;|[\, L'[X]R' \,]|\; Q')) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
= & (M' : ({}^{o}a^{r} \;|||\; (P' \;|[\, L'[X]R' \,]|\; Q'))) \Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{array}
$$

□

5.6.1 Proof (Page 104): **Eq.** (5.47): By coinduction on the equivalence relation

$$Rel \quad := \quad <\{(l : (P' \;|||\; Q'),\ (l : P') \;|||\; (l : Q')) \ :\ P', Q' \in CProc_{\Sigma_0,S}\}>$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and P', Q' as

in the definition of Rel.

$$
\begin{aligned}
&(l:(P' \mid\mid\mid Q'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= \bigcup_{e\in\Sigma_\tau} l:(P' \mid\mid\mid Q')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \;\cup \bigcup_{\substack{a\in\Sigma \\ e=l.a}} l:(P' \mid\mid\mid Q')\Big\downarrow \begin{smallmatrix} m \\ a \\ n \end{smallmatrix} \Big\downarrow \\
&= l:\bigcup_{e\in\Sigma_\tau}\left(P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \mid\mid\mid Q' \;\cup\; P' \mid\mid\mid Q'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right) \\
&\qquad \cup \quad l:\bigcup_{\substack{a\in\Sigma \\ e=l.a}}\left(P'\Big\downarrow \begin{smallmatrix} m \\ a \\ n \end{smallmatrix} \Big\downarrow \mid\mid\mid Q' \;\cup\; P' \mid\mid\mid Q'\Big\downarrow \begin{smallmatrix} m \\ a \\ n \end{smallmatrix} \Big\downarrow\right) \\
\sim_{Rel} \quad &\bigcup_{e\in\Sigma_\tau}\left(\left(l:P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right) \mid\mid\mid (l:Q') \;\cup\; (l:P') \mid\mid\mid \left(l:Q'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right)\right) \\
&\qquad \cup \quad \bigcup_{\substack{a\in\Sigma \\ e=l.a}}\left(\left(l:P'\Big\downarrow \begin{smallmatrix} m \\ a \\ n \end{smallmatrix} \Big\downarrow\right) \mid\mid\mid (l:Q') \;\cup\; (l:P') \mid\mid\mid \left(l:Q'\Big\downarrow \begin{smallmatrix} m \\ a \\ n \end{smallmatrix} \Big\downarrow\right)\right) \\
&= (l:P')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \mid\mid\mid (l:Q') \;\cup\; (l:P') \mid\mid\mid (l:Q')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= ((l:P') \mid\mid\mid (l:Q'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

Eq. (5.48): By coinduction on the equivalence relation

$$Rel \quad := \quad <\{((P' \mid\mid\mid Q') \setminus X,\ (P' \setminus X) \mid\mid\mid (Q' \setminus X)) \ :\ P', Q' \in CProc_{\Sigma_0,S}\}>$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label and P', Q' as in the definition of Rel.

$$
\begin{aligned}
&((P' \mid\mid\mid Q') \setminus X)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= \bigcup_{e\notin X}(P' \mid\mid\mid Q')\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \setminus X \;\cup \bigcup_{\substack{a\in X \\ e=\tau(a)}}(P' \mid\mid\mid Q')\Big\downarrow \begin{smallmatrix} m \\ a \\ n \end{smallmatrix} \Big\downarrow \setminus X \\
&= \bigcup_{e\notin X}\left(P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \mid\mid\mid Q' \;\cup\; P' \mid\mid\mid Q'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\right) \setminus X \\
&\qquad \cup \quad \bigcup_{\substack{a\in X \\ e=\tau(a)}}\left(P'\Big\downarrow \begin{smallmatrix} m \\ a \\ n \end{smallmatrix} \Big\downarrow \mid\mid\mid Q' \;\cup\; P' \mid\mid\mid Q'\Big\downarrow \begin{smallmatrix} m \\ a \\ n \end{smallmatrix} \Big\downarrow\right) \setminus X \\
\sim_{Rel} \quad &\bigcup_{e\notin X}\left(\left(P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \setminus X\right) \mid\mid\mid (Q' \setminus X) \;\cup\; (P' \setminus X) \mid\mid\mid \left(Q'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \setminus X\right)\right) \\
&\qquad \cup \quad \bigcup_{\substack{a\in X \\ e=\tau(a)}}\left(\left(P'\Big\downarrow \begin{smallmatrix} m \\ a \\ n \end{smallmatrix} \Big\downarrow \setminus X\right) \mid\mid\mid (Q' \setminus X) \;\cup\; (P' \setminus X) \mid\mid\mid \left(Q'\Big\downarrow \begin{smallmatrix} m \\ a \\ n \end{smallmatrix} \Big\downarrow \setminus X\right)\right) \\
&= (P' \setminus X)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \mid\mid\mid (Q' \setminus X) \;\cup\; (P' \setminus X) \mid\mid\mid (Q' \setminus X)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&= ((P' \setminus X) \mid\mid\mid (Q' \setminus X))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

□

5.7.3 Proof (Page 111): 1. (*) holds because m is an element of the image of A_{p+q} iff $m = A_{p+q}(m)$, i.e. iff $m \cap \{p,q\} \in \{\varnothing, \{p,q\}\}$. For the first case the only possible value of m' satisfying the conditions is $m = m\backslash\{q\}$ in any case, for the second one because of $m \cap \{p,q\} = \varnothing \iff m' \cap \{p,q\} = \varnothing$ and $q \notin m'$ the solution has to be $m\backslash\{q\}$. The same holds for n'.

$$
\begin{aligned}
&(D_{p+q}(P))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&\quad = \bigcup_{\substack{m = A_{p+q}(m') \\ n = A_{p+q}(n')}} D_{p+q}(P\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow) \overset{(5.3)}{=} \bigcup_{\substack{m = A_{p+q}(m') \\ n = A_{p+q}(n') \\ q \notin m' \cup n'}} D_{p+q}(P\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow) \\
&\overset{(*)}{=} \begin{cases} D_{p+q}(P\Big\downarrow \begin{smallmatrix} m\backslash\{q\} \\ e \\ n\backslash\{q\} \end{smallmatrix} \Big\downarrow) & m = A_{p+q}(m), n = A_{p+q}(n) \\ \varnothing & \text{otherwise} \end{cases}
\end{aligned}
$$

2. By coinduction on the equivalence relation

$$Rel \quad := \quad <\{(D_{p+q}(P),\ P') \ :\ P' \in CProc_{\Sigma_0,S}\}>$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label and P' as in the definition of *Rel*.

- For $\{p,q\} \cap (m \cup n) = \varnothing$:

$$
\begin{aligned}
(D_{p+q}(P'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow &\overset{(5.49)}{=} \bigcup_{\substack{m = A_{p+q}(m) \\ n = A_{p+q}(n)}} D_{p+q}(P'\Big\downarrow \begin{smallmatrix} m\backslash\{q\} \\ e \\ n\backslash\{q\} \end{smallmatrix} \Big\downarrow) \\
&= \quad D_{p+q}(P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow) \quad \sim_{Rel} \quad P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

- For $\{p,q\} \cap (m \cup n) \neq \varnothing$: (*) holds because the conditions of the union imply $\{p,q\} \subseteq m \cup n$, and because of Impl. (5.3).

$$
(D_{p+q}(P'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \overset{(5.49)}{=} \bigcup_{\substack{m = A_{p+q}(m) \\ n = A_{p+q}(n)}} D_{p+q}(P'\Big\downarrow \begin{smallmatrix} m\backslash\{q\} \\ e \\ n\backslash\{q\} \end{smallmatrix} \Big\downarrow) \overset{(*)}{=} \varnothing \overset{(*)}{=} P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
$$

3. We only prove one of the equations as an example; the other proofs can be conducted similarly.

Eq. (5.56)**:** By coinduction: We show that the dynamics of the two processes coincide. Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label. We use the abbreviation

$o' := A_{p+q}(o)$.

$$(D_{p+q}({}^{o}P))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$$

$$= \bigcup_{\substack{m=A_{p+q}(m') \\ n=A_{p+q}(n')}} D_{p+q}(({}^{o}P)\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow) = \bigcup_{\substack{m=A_{p+q}(m') \\ n=A_{p+q}(n') \\ o\subseteq m'\setminus n'}} D_{p+q}(P\Big\downarrow \begin{smallmatrix} m'\setminus o \\ e \\ n' \end{smallmatrix} \Big\downarrow)$$

$$= \bigcup_{\substack{m=A_{p+q}(m'\cup o) \\ n=A_{p+q}(n') \\ o\subseteq (m'\cup o)\setminus n' \\ o\cap m'=\varnothing}} D_{p+q}(P\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow) = \bigcup_{\substack{m=A_{p+q}(m'\cup o) \\ n=A_{p+q}(n') \\ o\cap n'=\varnothing \\ o\cap m'=\varnothing}} D_{p+q}(P\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow) =: U_1$$

$$({}^{o'}D_{p+q}(P))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$$

$$= \bigcup_{o'\subseteq m\setminus n} (D_{p+q}(P))\Big\downarrow \begin{smallmatrix} m\setminus o' \\ e \\ n \end{smallmatrix} \Big\downarrow = \bigcup_{\substack{m\setminus o'=A_{p+q}(m') \\ n=A_{p+q}(n') \\ o'\subseteq m\setminus n}} D_{p+q}(P\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow) =: U_2$$

The conditions of the unions of U_1 and U_2 are equivalent: In any case we can assume $q \notin m' \cup n'$ because of $q \notin FP(P)$.

- For $p \in o$: Then $o' = o \cup \{q\}$. We show the two directions of the equivalence:

  '$U_1 \implies U_2$':

  $m\setminus o' = A_{p+q}(m')$: The premiss $o \cap m' = \varnothing$ entails $p \notin m'$ and thus

  $$A_{p+q}(m') = m' = m'\setminus o' = (m' \cup o \cup \{q\})\setminus o' = A_{p+q}(m' \cup o)\setminus o' = m\setminus o'$$

  $o' \subseteq m\setminus n$: Analogously we have $o \cap n' = \varnothing$ and thus $p \notin n'$, which by the definition of A_{p+q} means $A_{p+q}(n') = n'$:

  $$m\setminus n = A_{p+q}(m' \cup o)\setminus A_{p+q}(n') = (m' \cup o \cup \{q\})\setminus n' \supseteq o$$

  '$U_2 \implies U_1$': Because of $\{p, q\} \subseteq o'$ we have $p, q \notin A_{p+q}(m') = m\setminus o'$, and hence $A_{p+q}(m') = m'$. In the same manner from $o' \subseteq m\setminus n$ the property $o' \cap n = \varnothing$ follows, and thus $A_{p+q}(n') = n' = n$.

  $o \cap m' = \varnothing$: From $m' = A_{p+q}(m') = m\setminus o'$ follows

  $$o \cap m' \subseteq o' \cap m' = \varnothing$$

  $o \cap n' = \varnothing$:

  $$o \cap n' = o \cap n \subseteq o' \cap n = \varnothing$$

  $m = A_{p+q}(m' \cup o)$:

  $$A_{p+q}(m' \cup o) = m' \cup o \cup \{q\} = (m\setminus o') \cup o' = m$$

- For $p \notin o$: Together with $q \notin o$ this means $o' = A_{p+q}(o) = o$ and

$$A_{p+q}(m' \cup o) = A_{p+q}(m') \cup o$$

Again we separate the two implication directions:

'$U_1 \implies U_2$':

$m \backslash o' = A_{p+q}(m')$: By premiss we have $A_{p+q}(m') \cup o = m$ and thus $A_{p+q}(m') \backslash o = m \backslash o$. The remaining part to prove is

$$A_{p+q}(m') \cap o \subseteq (m' \cup \{p, q\}) \cap o = \varnothing$$

$o' \subseteq m \backslash n$: We have $o \cap n' = \varnothing = o \cap n$ and hence

$$\begin{aligned} m \backslash n = (A_{p+q}(m') \cup o) \backslash n &= (A_{p+q}(m') \backslash n) \cup (o \backslash n) \\ &= (A_{p+q}(m') \backslash n) \cup o \supseteq o \end{aligned}$$

'$U_2 \implies U_1$':

$o \cap m' = \varnothing$:

$$o \cap m' \subseteq o \cap A_{p+q}(m') = o \cap (m \backslash o') = \varnothing$$

$o \cap n' = \varnothing$: Because of $o' \subseteq m \backslash n$ we have

$$o \cap n' = o \cap n = \varnothing$$

$m = A_{p+q}(m' \cup o)$:

$$A_{p+q}(m' \cup o) = A_{p+q}(m') \cup o = (m \backslash o') \cup o = m$$

4. Omitted. □

5.7.5 Proof (Page 114): 1. By coinduction on the equivalence relation

$$Rel \quad := \quad < \{ (\, C_{p+q}(D_{p+q}(P')),\ P') \ :\ P' \in CProc_{\Sigma_0, S},\ q \notin FP(P') \} >$$

Rel is a bisimulation equivalence: Let $(m, e, n) \in \mathcal{L}_{CCSP}$ be a label and P' as in the definition of *Rel*.

- For $q \notin m \cup n$: For (*) it can be observed that the conditions of the union have the unique solution $(m', n') = (A_{p+q}(m), A_{p+q}(n))$. (**) uses that the conditions of the definition of Cl_{p+q} always evaluates to false because of $m' \cap \{p, q\} \backslash n' \in \{\varnothing, \{p, q\}\}$, which means that $Cl_{p+q}(m', n', \cdot)$ is the iden-

tity.

$$
\begin{aligned}
&(C_{p+q}(D_{p+q}(P')))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&\quad = \bigcup_{(m,n)\lhd_{p+q}(m',n')} C_{p+q}(Cl_{p+q}(m',n',(D_{p+q}(P'))\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow)) \\
&\quad \overset{(5.49)}{=} \bigcup_{\substack{(m,n)\lhd_{p+q}(m',n') \\ m'=A_{p+q}(m') \\ n'=A_{p+q}(n')}} C_{p+q}(Cl_{p+q}(m',n',D_{p+q}(P'\Big\downarrow \begin{smallmatrix} m'\setminus\{q\} \\ e \\ n'\setminus\{q\} \end{smallmatrix} \Big\downarrow))) \\
&\quad \overset{(*)}{=} \bigcup_{\substack{(m,n)\lhd_{p+q}(m',n') \\ m'=A_{p+q}(m') \\ n'=A_{p+q}(n')}} C_{p+q}(Cl_{p+q}(m',n',D_{p+q}(P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow))) \\
&\quad \overset{(**)}{=} C_{p+q}(D_{p+q}(P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow)) \quad \sim_{Rel} \quad P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow
\end{aligned}
$$

- For $q \in m \cup n$: Because of $q \notin FP(P')$ we have

$$(C_{p+q}(D_{p+q}(P')))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad = \quad \varnothing \quad = \quad P'\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow$$

2. Omitted.

3. By coinduction on the equivalence relation

$$Rel \quad := \quad < \left\{ \begin{array}{c} (\tilde{M}' : C_{p+q}(P'),\ \tilde{M}' : C_{p+q}(Q')) \ : \\ M', \tilde{M}' \in \mathcal{M}_S,\ P', Q' \in CProc_{\Sigma_0,S}, \\ M'|_{S\setminus\{q\}} = \tilde{M}'|_{S\setminus\{q\}},\ M'(p) = M'(q),\ P' \sim_{M'} Q' \end{array} \right\} >$$

Rel is a bisimulation equivalence: Let $(m,e,n) \in \mathcal{L}_{CCSP}$ be a label and M', $\tilde{M}'$, P', Q' as in the definition of *Rel*. We abbreviate $M' + (m',n')$ with M^+ and $\tilde{M}' + (m,n)$ with $\tilde{M}^+$ in the sequel, and we only consider the interesting case $(m,n) \in \mathrm{PotEn}(\tilde{M}')$.

- For $q \notin m \cup n$:

$$
\begin{aligned}
&(\tilde{M}' : C_{p+q}(P'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&\quad = \quad \tilde{M}^+ : (C_{p+q}(P'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\
&\quad = \quad \tilde{M}^+ : \bigcup_{(m,n)\lhd_{p+q}(m',n')} C_{p+q}(Cl_{p+q}(m',n',P'\Big\downarrow \begin{smallmatrix} m' \\ e \\ n' \end{smallmatrix} \Big\downarrow)) \quad =: \quad U
\end{aligned}
$$

To be able to continue the derivation, at this point we first show the applicability of *Rel* as an auxiliary statement for each pair (m',n') with $(m,n) \lhd_{p+q} (m',n')$. The property $(m',n') \in \mathrm{PotEn}(M')$ at (*) follows from $(m,n) \in \mathrm{PotEn}(\tilde{M}')$: By Lem. 4.1.6, 3 we have to show (we use $m \cap \{p,q\} = \varnothing \iff m' \cap \{p,q\} = \varnothing$ and $M'(p) = M'(q) = \tilde{M}'(p)$ for the

first inequation, and $n'\backslash m' = A_{p+q}(n\backslash m)$ for the second one, which follows from Table 5.6)

$$\begin{array}{rcll} M'(m') & = & M'(m'\backslash\{p,q\}) \cup M'(m' \cap \{p,q\}) & \\ & = & \tilde{M}'(m\backslash\{p,q\}) \cup \tilde{M}'(m \cap \{p,q\}) & \leqslant \quad \mathrm{M} \\ M'(n'\backslash m') & = & \begin{cases} M'((n\backslash m) \cup \{q\}) = \tilde{M}'(n\backslash m) \leqslant \mathrm{E} & p \in n\backslash m \\ M'(n\backslash m) = \tilde{M}'(n\backslash m) \leqslant \mathrm{E} & p \notin n\backslash m \end{cases} & \end{array}$$

For (**) we employ Impl. (5.43) to treat the application $p[\mathrm{M}] : \cdot$ or $q[\mathrm{M}] : \cdot$ of the causal state operator potentially introduced by Cl_{p+q}. Therefore, we have to show the premiss $M^+(p) \leqslant \mathrm{M}$ (resp. $M^+(q) \leqslant \mathrm{M}$, also using Lem. 4.1.6, 7) of the implication for concerned situations. If Cl_{p+q} introduces $p[\mathrm{M}] : \cdot$ we have $q \in m'\backslash n'$ and $p \notin m'\backslash n'$ and thus $M'(q) = M'(p) \leqslant \mathrm{M}$ (because of $(m', n') \in \mathrm{PotEn}(M')$). By definition this entails $M^+(p) = (M' + (m', n'))(p) \leqslant \mathrm{M}$. Analogous reasoning applies to the case $q[\mathrm{M}] : \cdot$.

For (***) we finally apply *Rel*, and have to show that $M^+|_{S\backslash\{q\}} \leqslant \tilde{M}^+|_{S\backslash\{q\}}$ (using Lem. 4.1.6, 7). Because of $m' \ominus m \subseteq \{p,q\}$, $n' \ominus n \subseteq \{p,q\}$ the only interesting point is p. The different possible cases (by the definition of $M' + (m', n')$) are checked referring to Table 5.6:

$$\begin{array}{rcl} p \in m'\backslash n' & \Longrightarrow & p \in m\backslash n \quad \Longrightarrow \quad \tilde{M}^+(p) = \mathrm{D} \\ p \in n' & \Longrightarrow & p \in m \cup n \quad \Longrightarrow \quad \tilde{M}^+(p) = \mathrm{M} = M^+(p) \\ p \notin m' \cup n' & \Longrightarrow & M^+(p) = M'(p) = \tilde{M}'(p) \leqslant \tilde{M}^+(p) \end{array}$$

Eventually we derive the auxiliary statement

$$\begin{array}{ll} & M' : P' = M' : Q' \\ \Longrightarrow & (M' : P')\Big\downarrow_{n'}^{m'e}\Big\downarrow = (M' : Q')\Big\downarrow_{n'}^{m'e}\Big\downarrow \\ \overset{(*)}{\Longleftrightarrow} & M^+ : P'\Big\downarrow_{n'}^{m'e}\Big\downarrow = M^+ : Q'\Big\downarrow_{n'}^{m'e}\Big\downarrow \\ \overset{(**)}{\Longrightarrow} & M^+ : Cl_{p+q}(m', n', P'\Big\downarrow_{n'}^{m'e}\Big\downarrow) = M^+ : Cl_{p+q}(m', n', Q'\Big\downarrow_{n'}^{m'e}\Big\downarrow) \\ \overset{(***)}{\Longrightarrow} & \tilde{M}^+ : C_{p+q}(\,Cl_{p+q}(m', n', P'\Big\downarrow_{n'}^{m'e}\Big\downarrow)) \\ & \qquad \sim_{Rel} \tilde{M}^+ : C_{p+q}(Cl_{p+q}(m', n', Q'\Big\downarrow_{n'}^{m'e}\Big\downarrow)) \end{array}$$

The reasoning is concluded with

$$\begin{array}{rcl} U & \sim_{Rel} & \tilde{M}^+ : \bigcup_{(m,n) \lhd_{p+q} (m',n')} C_{p+q}(Cl_{p+q}(m', n', Q'\Big\downarrow_{n'}^{m'e}\Big\downarrow)) \\ & = & (\tilde{M}' : C_{p+q}(Q'))\Big\downarrow_{n}^{me}\Big\downarrow \end{array}$$

- For $q \in m \cup n$:

$$\begin{aligned}(\tilde{M}' : C_{p+q}(P'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad &= \quad \tilde{M}^+ : (C_{p+q}(P'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \\ &= \quad \varnothing \quad = \quad (\tilde{M}' : C_{p+q}(Q'))\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow\end{aligned}$$

4. Omitted.

5. Omitted. □

5.7.6 Proof (Page 118): Omitted. □

5.7.7 Proof (Page 119): Omitted. □

A.6. Transformation of CCSP Terms to Head Normal Form

6.6.2 Proof (Page 142): We conduct the proof in three steps: Of the two parts that need to be shown for item 1 (the two conditions of well-formedness), we start with the first one; successively we show item 2 of the lemma while referring to the first part of item 1, and the proof is concluded with the second part of item 1. Essentially, each proof step is an investigation of the concerned rewriting rules. We repeatedly use the fact that—given a certain position within the process term P—each rewriting step can either be carried out

- At the position, i.e. the operator at the position directly turns up on the left side of the rule
- Below the position, i.e. within the scope of the symbol at the position
- Above the position, whereby the position turns up within a variable instantiation of the left side of the rule
- Unrelated to the position, i.e. the rewritten subterm is disjoint from the subterm below the position.

Item 1, first part: The first condition of well-formedness, namely that C^t_{p+q} turns up at most once in P', follows from the fact that no rewriting rules contain variables $\tilde{P}$, $\tilde{P}$ for process terms twice on the right side, i.e. subterms of the process term are not duplicated. Those rules that directly contain a symbol $C^t_{p'+q'}$ on their right side either contain the symbol also on the left side, or have the premiss that q' is a fresh place.

For the second condition (if C^t_{p+q} turns up in P', then all occurrences of q are in the scope of C^t_{p+q}) we assume that C^t_{p+q} turns up in P'.

- If C^t_{p+q} also occurs in P: If the rewriting takes place in the scope of C^t_{p+q}, then parts of P that are not below C^t_{p+q} are not modified, and thus q cannot be introduced in these parts. If the rewriting is above or unrelated to the position of C^t_{p+q}, which is at most matched by a variable $\tilde{P}$ for process

terms, then by premiss (and by checking the rules) no other variable of the rule is instantiated with an expression containing q. As (the instantiation of) $\tilde{P}$ is not altered by the application of the rule, the rewriting does not introduce new occurrences of q.

Rules that directly match on C^t_{p+q} only enlarge the scope of the symbol by moving it towards the root of the process term, and thus cannot violate the second condition (a special case is the rule for eliminating C^t_{p+q}, which is not important at this point, because it is inconsistent with the assumption that C^t_{p+q} turns up in P').

- If C^t_{p+q} does not occur in P: There are only three rules (namely for the distribution of token consumption, and for the commutation of place duplication and place merging) that introduce new occurrences of C^t_{p+q}; all of them have the premiss that q is a fresh place.

For the third condition of well-formedness (q does not turn up in scopes of symbols D^t_{p+q}), it can immediately be checked that all rules only introduce places that are already part of the original subterm, or that do not occur in the whole process term P at all (fresh places). Rewriting in scopes of D^t_{p+q} does therefore not introduce new occurrences of q. Rewriting outside of the scope of D^t_{p+q} does not alter concerned subterms $D^t_{p+q}(\ldots)$ of P. The rules that contain D^t_{p+q} on the right side are:

- Rules for moving D^t_{p+q} downwards to the leafs of the process term P: The arguments of occurrences of D^t_{p+q} are in this case subterms of arguments of D^t_{p+q} on the left side, which by premiss do not contain q.
- Rules for moving C^t_{p+q} upwards to the root: This can apply D^t_{p+q} to subterms that in P are not in the scope of C^t_{p+q}, and which by premiss do not contain q.
- The rule for the commutation of place duplication and place merging that introduces a new symbol D^t_{p+q}: By premiss of the rule, q is a fresh place.
- Distribution of token consumption: By premiss of the rule, q is a fresh place.

Item 2: For each of the rewriting rules it can be checked that the proposition holds if the rule is applied on the top level of a process term, which is backed by the lemmas that are given for each group of rules. For the rule eliminating the symbol C^t_{p+q}, Lem. 5.7.6, 1 can be applied because of the well-formedness of P.

For applications of a rule within a process term, i.e. possibly in the scope of operators from which the elements of the term algebra $Proc_S$ are constructed, we employ the congruence property of $\sim_{M_{\mathrm{E}}}$ that is stated in Lem. 5.4.3 and Lem. 5.7.5, Lem. 5.7.3. For the latter lemma, we need the statement that place q does not turn up in scopes of D^t_{p+q} also in the process term P', which follows from the first part of the proof.

Item 1, second part: Again we assume that C^t_{p+q} turns up in P'.

- If C^t_{p+q} also occurs in P: If the rewriting takes places in the scope of C^t_{p+q}, then we can observe that the well-formedness of P entails the well-formedness of subterms of P. By item 2, this means that the argument Q of C^t_{p+q} in P is rewritten to Q' in P' with $Q \sim_{M_\mathrm{E}} Q'$, and thus

$$T^{fix}_{p+q}(M_\mathrm{E} : \mathrm{val}(Q)) \Longleftrightarrow T^{fix}_{p+q}(M_\mathrm{E} : \mathrm{val}(Q'))$$

  Rewriting outside of the scope of C^t_{p+q} does not alter the subterm $C^t_{p+q}(Q)$. For the rules that move C^t_{p+q} towards the root of the process term, we can apply Lem. 5.7.6, 4.
- If C^t_{p+q} does not occur in P: The two rules for token consumption, and the rule for commuting place duplication and place merging (which introduce symbols C^t_{p+q}) can be treated using Lem. 5.7.6, 2 and Lem. 5.7.6, 4. □

6.7.1 Proof (Page 143): 1. Omitted.

2. The particular features of irreducible process terms hold because of:
   - There are rules that commute $C^t_{p'+q'}$ with all other operators, and thus $C^t_{p'+q'}$ can in irreducible terms only turn up on top level.
   - For all situations of the shape $D^t_{p'+q'}(Q)$ there are applicable rules, namely rules that commute $D^t_{p'+q'}$ with the top-level operator of Q. Hence $D^t_{p'+q'}$ cannot turn up in irreducible terms.
   - There are rules that commute the causal state operator with all other kinds of symbols apart from $C^t_{p'+q'}$, and thus state operators can in P' only turn up in the scope of $C^t_{p'+q'}$ (and not below other operators).
   - We decompose the subterm Q below the state operators as $T \,|||_t\, R$ by collecting in T all leading transitions of Q that are only below interleaving $|||_t$. Therefore, we show that for all irreducible process terms Q' that do not contain the causal state operator or $C^t_{p'+q'}$, $D^t_{p'+q'}$ one of the following statements holds:
     - Q' does not contain transitions at all
     - Q' is a transition
     - Q' has the shape ${}^{o}a^{r}_{t} \,|||_t\, Q''$.

     This is proved by structural induction on the elements of $Proc_S$, and by using the observation that subterms of an irreducible process term are also irreducible.
     - Primary processes and $stop_t$ do not contain transitions
     - Terms $a \rightarrow_t Q''$ or $Q'' \,\Box_t\, Q'''$ are not irreducible
     - A transition directly satisfies the hypothesis
     - If for terms $\left({}^{o}_{t}Q''\right)$, $\left({}^{o*}_{t}Q''\right)$, $\left(l :_t Q''\right)$, $\left(Q'' \setminus_t X\right)$ the subterm Q'' is irreducible, then one of the three cases applies. If Q'' does not contain transitions, then Q' does not either. If Q'' is a transition or its top-level operator is interleaving, then a rewriting rule can be applied to Q'.

– If a term $Q'' \,[\![\, L[X]R \,]\!]_t\, Q'''$ is irreducible, then also Q'' and Q''' are irreducible. Provided that one of the subterms contains a transition, it is possible to apply a rule for propagation of transitions through parallelism by the induction hypothesis. Note that formally the elements of L, R are no transitions.
– If a term $Q'' \,|||_t\, Q'''$ is irreducible: If Q'' or Q''' contains a transition, then either Q' has the shape ${}^{o}a\,{}^{r}_{t} \,|||_t\, Q''$, or a rewriting rule for reordering of interleaving can be applied (by the induction hypothesis).

This entails that the subterm Q below state operators is of the kind

$$ {}^{o_1}a_1\,{}^{r_1}_{t} \,|||_t \left({}^{o_2}a_2\,{}^{r_2}_{t} \,|||_t \left(\cdots \left({}^{o_n}a_n\,{}^{r_n}_{t} \,|||_t\, Q' \right) \right) \right) $$

where Q' does not contain transitions. We choose $T = {}^{o_1}a_1\,{}^{r_1}_{t} \,|||_t \cdots |||_t\, {}^{o_n}a_n\,{}^{r_n}_{t}$ (or $T = stop_t$ for $n = 0$) and $R = Q'$, which concludes the proof.

3. The proof is conducted similarly as for item 2. Additionally, we can first observe that rewriting does not introduce transitions in the scope of the prefixing operator, i.e. the premiss of the item is preserved. This follows from the property of the rewriting system that those rules that contain transitions on the right side also contain transitions on the left side, and can thus not be applied in the scope of prefixing. The only exception is the rule that matches on prefixing, which is not applied by premiss.

   The structural induction of item 2 can the be adapted, as the case of an irreducible term $a \to_t Q''$ is handled by the fact that Q'' does not contain transitions. □

6.8.4 Proof (Page 148): By structural inductions on the set $Proc_S$ of process terms. To treat the particular cases, we also use coinduction and show that the process $init(\mathrm{val}(P), M)$ is not able to perform any execution step (like the process $stop$). Only a selection of the cases is shown below.

- For $blocked_M(stop_t)$:

$$ init(\mathrm{val}(stop_t), M) \quad = \quad init(stop, M) \quad = \quad stop $$

- For $blocked_M({}^{o}a\,{}^{r}_{t})$: Then necessarily $(o, r) \notin \mathrm{En}(M)$, which is used as (*).

$$ init(\mathrm{val}({}^{o}a\,{}^{r}_{t}), M)\Big\downarrow {}^{m}_{e}_{n}\Big\downarrow \quad = \quad init({}^{o}(\mathrm{val}_{pre}(a))\,{}^{r}, M)\Big\downarrow {}^{m}_{e}_{n}\Big\downarrow $$

$$ = \begin{cases} \bigcup_{(m',n')\in \mathrm{En}(M)} init({}^{o}(\mathrm{val}_{pre}(a))\,{}^{r}\Big\downarrow {}^{m'}_{e}_{n'}\Big\downarrow, M + (m', n')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} $$

$$ = \begin{cases} \bigcup_{\substack{(m',n')\in \mathrm{En}(M) \\ (m',n')=(o,r) \\ \mathrm{val}_{pre}(a)=e}} \{init(stop, M + (m', n'))\} & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} $$

$$ \overset{(*)}{=} \quad \varnothing $$

- For $\mathit{blocked}_M({}^{o}_{t}P)$ or $\mathit{blocked}_M({}^{o*}_{t}P)$: There are two cases:

  – For $(o,\varnothing) \notin \mathrm{En}(M)$: This means $M(o) \not\subseteq \{\mathrm{M}\}$, which in particular entails $(m',n') \notin \mathrm{En}(M)$ whenever $o \subset m'$. As for the last case we have

$$
\begin{aligned}
\mathit{init}(\mathrm{val}({}^{o}_{t}P), M)\Big\downarrow {}^{m}_{n}{}_{e}\Big\downarrow \quad &= \quad \mathit{init}({}^{o}(\mathrm{val}(P)), M)\Big\downarrow {}^{m}_{n}{}_{e}\Big\downarrow \\
&= \begin{cases} \bigcup_{(m',n')\in\mathrm{En}(M)} \mathit{init}(({}^{o}(\mathrm{val}(P)))\Big\downarrow {}^{m'}_{n'}{}_{e}\Big\downarrow, M+(m',n')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} \bigcup_{\substack{(m',n')\in\mathrm{En}(M) \\ o\subseteq m'\setminus n'}} \mathit{init}(\mathrm{val}(P)\Big\downarrow {}^{m'\setminus o}_{n'}{}_{e}\Big\downarrow, M+(m',n')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&= \varnothing
\end{aligned}
$$

    For the term ${}^{o*}_{t}P$ the case $e \in \Sigma$ is handled as for ${}^{o}P$, for $e \in \Sigma_\tau$ the derivation becomes slightly different:

$$
\begin{aligned}
\mathit{init}(\mathrm{val}({}^{o*}_{t}P), M)\Big\downarrow {}^{m}_{n}{}_{e}\Big\downarrow \quad &= \quad \mathit{init}({}^{o*}(\mathrm{val}(P)), M)\Big\downarrow {}^{m}_{n}{}_{e}\Big\downarrow \\
&= \begin{cases} \bigcup_{(m',n')\in\mathrm{En}(M)} \mathit{init}(({}^{o*}(\mathrm{val}(P)))\Big\downarrow {}^{m'}_{n'}{}_{e}\Big\downarrow, M+(m',n')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} \bigcup_{\substack{(m',n')\in\mathrm{En}(M) \\ o\subseteq m'\cap n'}} \mathit{init}(\mathrm{val}(P)\Big\downarrow {}^{m'\setminus o}_{n'\setminus o}{}_{e}\Big\downarrow, M+(m',n')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&= \varnothing
\end{aligned}
$$

  – For $\mathit{blocked}_M(P)$: We assume $(o,\varnothing) \in \mathrm{En}(M)$, because otherwise the first case applies. $o \subseteq m'\setminus n'$ means that the pair (m',n') can be decomposed in $(m',n') = (o,\varnothing) + (m'',n'')$, and because of $(o,\varnothing) \in \mathrm{En}(M)$ we have

$$
(o,\varnothing) + (m'',n'') \in \mathrm{En}(M) \quad \Longleftrightarrow \quad (m'',n'') \in \mathrm{En}(M + (o,\varnothing)) \qquad (*)
$$

    which is used in the following transformations. A second auxiliary statement can be derived from $\mathrm{En}(M + (o,\varnothing)) \subseteq \mathrm{En}(M)$ (again provided that $(o,\varnothing) \in \mathrm{En}(M)$):

$$
\mathit{init}(P, M) = \mathit{stop} \quad \Longrightarrow \quad \mathit{init}(P, M + (o,\varnothing)) = \mathit{stop} \qquad (**)
$$

$$
\begin{aligned}
&\mathit{init}(\mathrm{val}({}^{o}_{t}P), M)\Big\downarrow {}^{m}_{n}{}_{e}\Big\downarrow \quad = \quad \mathit{init}({}^{o}(\mathrm{val}(P)), M)\Big\downarrow {}^{m}_{n}{}_{e}\Big\downarrow \\
&= \begin{cases} \bigcup_{(m',n')\in \mathrm{En}(M)} \mathit{init}(({}^{o}(\mathrm{val}(P)))\Big\downarrow {}^{m'}_{n'}{}_{e}\Big\downarrow, M + (m',n')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} \bigcup_{\substack{(m',n')\in \mathrm{En}(M) \\ o\subseteq m'\setminus n'}} \mathit{init}(\mathrm{val}(P)\Big\downarrow {}^{m'\setminus o}_{n'}{}_{e}\Big\downarrow, M + (m',n')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} \bigcup_{\substack{(o,\varnothing)+(m'',n'') \\ \in \mathrm{En}(M)}} \mathit{init}(\mathrm{val}(P)\Big\downarrow {}^{m''}_{n''}{}_{e}\Big\downarrow, M + (o,\varnothing) + (m'',n'')) & \\ & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&\overset{(*)}{=} \begin{cases} \bigcup_{\substack{(m'',n'')\in \\ \mathrm{En}(M+(o,\varnothing))}} \mathit{init}(\mathrm{val}(P)\Big\downarrow {}^{m''}_{n''}{}_{e}\Big\downarrow, M + (o,\varnothing) + (m'',n'')) & \\ & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&= \mathit{init}(\mathrm{val}(P), M + (o,\varnothing))\Big\downarrow {}^{m}_{n}{}_{e}\Big\downarrow \overset{(**)}{=} \varnothing
\end{aligned}
$$

Starred token consumption can be treated similarly: We only consider the case $e \in \Sigma_\tau$. By assumption we have

$$
\begin{aligned}
(o,o) \in \mathrm{En}(M) \quad &\iff \quad (o,\varnothing) \in \mathrm{En}(M) \\
&\iff \quad M + (o,o) = M \qquad (*)
\end{aligned}
$$

and from $o \subseteq m' \cap n'$ the decomposition $(m',n') = (o,o) + (m'',n'')$ follows.

$$
\begin{aligned}
&\mathit{init}(\mathrm{val}({}^{o*}_{t}P), M)\Big\downarrow {}^{m}_{n}{}_{e}\Big\downarrow \quad = \quad \mathit{init}({}^{o*}(\mathrm{val}(P)), M)\Big\downarrow {}^{m}_{n}{}_{e}\Big\downarrow \\
&= \begin{cases} \bigcup_{(m',n')\in \mathrm{En}(M)} \mathit{init}(({}^{o*}(\mathrm{val}(P)))\Big\downarrow {}^{m'}_{n'}{}_{e}\Big\downarrow, M + (m',n')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} \bigcup_{\substack{(m',n')\in \mathrm{En}(M) \\ o\subseteq m'\cap n'}} \mathit{init}(\mathrm{val}(P)\Big\downarrow {}^{m'\setminus o}_{n'\setminus o}{}_{e}\Big\downarrow, M + (m',n')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} \bigcup_{\substack{(o,o)+(m'',n'') \\ \in \mathrm{En}(M)}} \mathit{init}(\mathrm{val}(P)\Big\downarrow {}^{m''}_{n''}{}_{e}\Big\downarrow, M + (o,o) + (m'',n'')) & \\ & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&= \begin{cases} \bigcup_{\substack{(m'',n'')\in \mathrm{En}(M) \\ o\cap(m''\cup n'')=\varnothing}} \mathit{init}(\mathrm{val}(P)\Big\downarrow {}^{m''}_{n''}{}_{e}\Big\downarrow, M + (m'',n'')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \\
&\subseteq \begin{cases} \bigcup_{(m'',n'')\in \mathrm{En}(M)} \mathit{init}(\mathrm{val}(P)\Big\downarrow {}^{m''}_{n''}{}_{e}\Big\downarrow, M + (m'',n'')) & m \cup n = \varnothing \\ \varnothing & \text{otherwise} \end{cases}
\end{aligned}
$$

$$= \quad init(\mathrm{val}(P), M)\Big\downarrow \begin{smallmatrix} m \\ e \\ n \end{smallmatrix} \Big\downarrow \quad = \quad \varnothing \qquad \square$$

Bibliography

[ABB+04] Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, and Peter H. Schmitt. The KeY tool. *Software and System Modeling (SoSyM), Springer*, 2004.

[ACD+03] T. Arts, G. Chugunov, M. Dam, L. å. Fredlund, D. Gurov, and T. Noll. A tool for verifying software written in erlang. *Int. Journal of Software Tools for Technology Transfer*, 4(4):405–420, August 2003.

[AFV01] L. Aceto, W.J. Fokkink, and C. Verhoef. Structural operational semantics. In A. Ponse J. Bergstra and S. Smolka, editors, *Handbook of Process Algebra*. Elsevier, North-Holland, 2001.

[AVWW96] Joe Armstrong, Robert Virding, Claes Wikström, and Mike Williams. *Concurrent Programming in Erlang*. Prentice-Hall, second edition, 1996.

[Bar92] Geoff Barret. *occam3 reference manual*. Inmos, 1992.

[BB01] J.C.M. Baeten and T. Basten. Partial-order process algebra (and its relation to petri nets). In A. Ponse J. Bergstra and S. Smolka, editors, *Handbook of Process Algebra*. Elsevier, North-Holland, 2001.

[BDK01] Eike Best, Raymond Devillers, and Maciej Koutny. *Petri net algebra*. Springer-Verlag New York, Inc., 2001.

[BDK02] Eike Best, Raymond Devillers, and Maciej Koutny. The box algebra = petri nets + process expressions. *Inf. Comput.*, 178(1):44–100, 2002.

[Bec01] Bernhard Beckert. A dynamic logic for the formal verification of Java Card programs. In I. Attali and T. Jensen, editors, *Java on Smart Cards: Programming and Security. Revised Papers, Java Card 2000, International Workshop, Cannes, France*, volume 2041 of *LNCS*, pages 6–24. Springer, 2001.

[BK84] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1–3):109–137, 1984.

[BPS01] J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

[Bry86] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[BS01] Julian Bradfield and Colin Stirling. Modal logics and mu-calculi: an introduction. In A. Ponse J. Bergstra and S. Smolka, editors, *Handbook of Process Algebra*. Elsevier, North-Holland, 2001.

[CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71. Springer-Verlag, 1982.

[CGRH00] Andrea Corradini, Martin Große-Rhode, and Reiko Heckel. Structured transition systems as lax coalgebras. In Bart Jacobs, Larry Moss, Horst Reichel, and Jan Rutten, editors, *Electronic Notes in Theoretical Computer Science*, volume 11. Elsevier, 2000.

[Der87] Nachum Dershowitz. Termination of rewriting. In J.-P. Jouannaud, editor, *Rewriting Techniques and Applications*, pages 69–115. Academic Press, 1987. Reprinted from *Journal of Symbolic Computation*.

[DG99] M. Dam and D. Gurov. Compositional verification of CCS processes. In *Proc. PSI'99*, volume 1755 of *Springer LNCS*, pages 247–256, 1999.

[DG00] M. Dam and D. Gurov. μ-calculus with explicit points and approximations. In: *Proc. FICS'2000*, 2000.

[DP01] N. Dershowitz and D.A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001.

[EHD86] M Elizabeth, C Hull, and G Donnan. Contextually communicating sequential processes: a software engineering environment. *Softw. Pract. Exper.*, 16(9):845–864, 1986.

[For97] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement: FDR2 Manual*, 1997.

[Fre01] Lars-Åke Fredlund. A framework for reasoning about ERLANG, 2001.

[Gen35] Gerhard Gentzen. Untersuchungen über das Logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation [Sza69].

[GJSB00] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. Addison Wesley, 2nd edition, 2000.

[Gla01] R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001. Available at `http://boole.stanford.edu/pub/-spectrum1.ps.gz`.

[Gol87] U. Goltz. *On Representing CCS Programs by Finite Petri Nets.* Number 290. Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augustin, Arbeitspapiere der GMD, 1987. NewsletterInfo: 30.

[GvLH+96] Patrice Godefroid, J. van Leeuwen, J. Hartmanis, G. Goos, and Pierre Wolper. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem.* Springer-Verlag New York, Inc., 1996.

[Har84] David Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II: Extensions of Classical Logic, chapter 10, pages 497–604. Reidel, 1984.

[HM80] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 299–309. Springer-Verlag, 1980.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, NJ, 1985. & 0-13-153289-8.

[McM92] K. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. CAV '92, Fourth Workshop on Computer-Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 164–174, 1992.

[NPW80] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theor. Computer Science*, 13(1):85–108, January 1980.

[OMG01] Object Modeling Group. *Unified Modelling Language Specification, version 1.4*, September 2001.

[PJ00] P.H.Welch and J.M.R.Martin. A CSP Model for Java Multithreading. In P.Nixon and I.Ritchie, editors, *Software Engineering for Parallel and Distributed Systems*, pages 114–122. ICSE 2000, IEEE Computer Society Press, June 2000.

[Pow03] John Power. Towards a theory of mathematical operational semantics. In H. Peter Gumm, editor, *Electronic Notes in Theoretical Computer Science*, volume 82. Elsevier, 2003.

[Ros98] A. W. Roscoe. *The theory and practice of concurrency.* Prentice-Hall, 1998.

[RT94] Jan J. M. M. Rutten and Daniele Turi. Initial algebra and final coalgebra semantics for concurrency. In *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium*, pages 530–582. Springer-Verlag, 1994.

[Rut95] J.J.M.M. Rutten. A calculus of transition systems (towards universal coalgebra). *Technical Report CS-R9503, CWI, Amsterdam*, 1995.

[Rut96] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 1996.

[Sim95] Alex K. Simpson. Compositionality via cut-elimination: Hennessy-milner logic for an arbitrary GSOS. In *Logic in Computer Science*, pages 420–430, 1995.

[Sti92] Colin Stirling. Modal and temporal logics. In S. Abramsky, Dov M. Gabbay, and T. S.Ẽ. Maibaum, editors, *Handbook of Logic in Computer Science. Volume 2. Background: Computational Structures*, pages 477–563. Oxford University Press, 1992.

[Sun02] Sun Microsystems, Inc., Palo Alto/CA, USA. *Java Card 2.2 Platform Specification*, September 2002.

[Sza69] M. E. Szabo, editor. *The Collected Papers of Gerhard Gentzen.* North-Holland, Amsterdam, 1969.

[Tar55] A. Tarski. A lattice-theoretical fixedpoint theorem and its application. *Pacific Journal of Mathematics*, (83):157–167, 1955.

[TP97] Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *Proceedings 12th Ann. IEEE Symp. on Logic in Computer Science, LICS'97, Warsaw, Poland, 29 June – 2 July 1997*, pages 280–291. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[WA] P.H. Welch and P.D. Austin. *Java Communicating Sequential Processes home page.* http://www.cs.ukc.ac.uk/projects/ofa/jcsp/.