E-Matching and the Hypertableau Rule in the Theorem Prover Princess

Philipp Rümmer philipp@chalmers.se

Wintermeeting of the SET Division January 13th 2009

- The theorem prover Princess
- E-Matching
- Hypertableaux
- How to simulate e-matching using hypertableaux

Prover for first-order logic with linear integer arithmetic:

- Tailored to program verification
- Complete for first-order logic, Presburger arithmetic, etc.
- Classical sequent calculus, non-clausal
- No uninterpreted functions \rightarrow relational encoding
- Free variables + unification + constraints

More information, implementation, paper: http://www.cse.chalmers.se/~philipp/princess/ Prover for first-order logic with linear integer arithmetic:

- Tailored to program verification
- Complete for first-order logic, Presburger arithmetic, etc.
- Classical sequent calculus, non-clausal
- No uninterpreted functions \rightarrow relational encoding
- Free variables + unification + constraints

$$\frac{\Gamma, \forall \bar{x}. \phi, [\bar{x}/\bar{X}]\phi \vdash \Delta}{\Gamma, \forall \bar{x}. \phi \vdash \Delta}$$

More information, implementation, paper: http://www.cse.chalmers.se/~philipp/princess/

2. E-Matching

Standard quantifier handling in SMT solvers:

• Matching of "triggers" (modulo equations):

 $\frac{\Gamma, \forall \bar{x}. \phi[t[\bar{x}]], [\bar{x}/\bar{s}]\phi[t[\bar{x}]] \vdash \psi[t[\bar{s}]], \Delta}{\Gamma, \forall \bar{x}. \phi[t[\bar{x}]] \vdash \psi[t[\bar{s}]], \Delta}$

• Triggers $t[\bar{x}]$ are often provided by user

2. E-Matching

Standard quantifier handling in SMT solvers:

• Matching of "triggers" (modulo equations):

 $\frac{\Gamma, \forall \bar{x}. \phi[t[\bar{x}]], [\bar{x}/\bar{s}]\phi[t[\bar{x}]] \vdash \psi[t[\bar{s}]], \Delta}{\Gamma, \forall \bar{x}. \phi[t[\bar{x}]] \vdash \psi[t[\bar{s}]], \Delta}$

• Triggers $t[\bar{x}]$ are often provided by user

```
\forall int a, i, v;
   select(store(a, i, v), i) = v
\forall int a, i1, i2, v;
   (i1 != i2 ->
    select(store(a, i1, v), i2) = select(a, i2))
```

2. E-Matching

Standard quantifier handling in SMT solvers:

• Matching of "triggers" (modulo equations):

 $\frac{\Gamma, \forall \bar{x}. \phi[t[\bar{x}]], [\bar{x}/\bar{s}]\phi[t[\bar{x}]] \vdash \psi[t[\bar{s}]], \Delta}{\Gamma, \forall \bar{x}. \phi[t[\bar{x}]] \vdash \psi[t[\bar{s}]], \Delta}$

• Triggers $t[\bar{x}]$ are often provided by user

```
\forall int a, i, v;
    select(store(a, i, v), i) = v
\forall int a, i1, i2, v;
    (i1 != i2 ->
    select(store(a, i1, v), i2) = select(a, i2))
```

_

E-Matching	Free variables + unification
$\text{Heuristic} \rightarrow \text{incomplete}$	Systematic
Good for "simple" instances	Can find "difficult" instances
Quite cheap	Quite expensive \rightarrow Very nondeterministic

E-Matching	Free variables + unification
Heuristic \rightarrow incomplete	Systematic
Good for "simple" instances	Can find "difficult" instances
Quite cheap	Quite expensive \rightarrow Very nondeterministic

 $\Rightarrow \textbf{Combination?}$

Derive models of clause sets by fixed-point iteration:

- Clauses without negative literals:
 - \Rightarrow Instantiate with free variables
- Clauses with negative literals:
 - \Rightarrow Discharge negative literals with unit resolution

Derive models of clause sets by fixed-point iteration:

- Clauses without negative literals:
 - \Rightarrow Instantiate with free variables
- Clauses with negative literals:
 - \Rightarrow Discharge negative literals with unit resolution

 $\forall x.p(x), \forall x.(p(x) \rightarrow q(x) \lor r(x+1)), \forall x.\neg r(x) \vdash$

Derive models of clause sets by fixed-point iteration:

- Clauses without negative literals:
 - \Rightarrow Instantiate with free variables
- Clauses with negative literals:
 - \Rightarrow Discharge negative literals with unit resolution

 $\forall x.p(x), \forall x.(p(x) \rightarrow q(x) \lor r(x+1)), \forall x.\neg r(x) \vdash$

Derive models of clause sets by fixed-point iteration:

- Clauses without negative literals:
 - \Rightarrow Instantiate with free variables
- Clauses with negative literals:

$$\frac{\ldots, p(X) \vdash}{\forall x. p(x), \forall x. (p(x) \rightarrow q(x) \lor r(x+1)), \forall x. \neg r(x) \vdash}$$

Derive models of clause sets by fixed-point iteration:

- Clauses without negative literals:
 - \Rightarrow Instantiate with free variables
- Clauses with negative literals:

$$\frac{\overline{(x,p(x),\forall x.(p(x)\rightarrow q(x)\lor r(x+1))},\forall x.\neg r(x)\vdash (x,y))}$$

Derive models of clause sets by fixed-point iteration:

- Clauses without negative literals:
 - \Rightarrow Instantiate with free variables
- Clauses with negative literals:

$$\frac{\overline{q(X) \lor r(X+1) \vdash}}{ \forall x.p(x), \forall x.(p(x) \rightarrow q(x) \lor r(x+1)), \forall x.\neg r(x) \vdash}$$

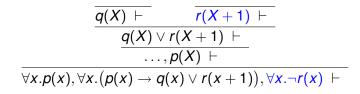
Derive models of clause sets by fixed-point iteration:

- Clauses without negative literals:
 - \Rightarrow Instantiate with free variables
- Clauses with negative literals:
 - \Rightarrow Discharge negative literals with unit resolution

$$\frac{\overline{q(X) \vdash r(X+1) \vdash}}{\underline{q(X) \lor r(X+1) \vdash}}_{(X,p(X),\forall x.(p(x) \rightarrow q(x) \lor r(x+1)),\forall x.\neg r(x) \vdash}$$

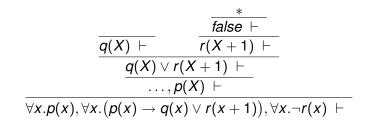
Derive models of clause sets by fixed-point iteration:

- Clauses without negative literals:
 - \Rightarrow Instantiate with free variables
- Clauses with negative literals:
 - \Rightarrow Discharge negative literals with unit resolution



Derive models of clause sets by fixed-point iteration:

- Clauses without negative literals:
 - \Rightarrow Instantiate with free variables
- Clauses with negative literals:



Derive models of clause sets by fixed-point iteration:

- Clauses without negative literals:
 - \Rightarrow Instantiate with free variables
- Clauses with negative literals:

 \Rightarrow Discharge negative literals with unit resolution

$$\frac{\overline{q(X) \vdash}}{\frac{q(X) \lor r(X+1) \vdash}{\frac{q(X) \lor r(X+1) \vdash}{\dots, p(X) \vdash}}}{\frac{q(X) \lor r(X+1) \vdash}{\dots, p(X) \vdash}}$$

Completeness (Conjecture)

If $\Gamma \vdash \Delta$ is provable in the ordinary Princess calculus, then it is also provable with the Hypertableau rule.

n-ary function *f* becomes (n + 1)-ary predicate f_p :

Axioms: Totality + Functionality

 $\forall \overline{x}. \exists y. f_{\rho}(\overline{x}, y) \\ \forall \overline{x}, y_1, y_2. (f_{\rho}(\overline{x}, y_1) \rightarrow f_{\rho}(\overline{x}, y_2) \rightarrow y_1 \doteq y_2)$

• Two equivalent ways to encode function applications:

$$\begin{aligned} \phi[f(\bar{t})] & \rightsquigarrow & \forall y.(f_p(\bar{t},y) \to \phi[y]) & \text{(negative)} \\ & \rightsquigarrow & \exists y.(f_p(\bar{t},y) \land \phi[y]) & \text{(positive)} \end{aligned}$$

All function applications become literals

E-Matching using the Hypertableau rule

 $\forall \bar{\mathbf{x}}.\phi[t[\bar{\mathbf{x}}]]$

3

£

negative function encoding for $t[\bar{x}]$

positive encoding for other function appl.

- E-Matching (almost) like in SMT-solvers
- But: Choice of triggers has no effect on completeness!

E-Matching using the Hypertableau rule

 $\forall \bar{\mathbf{x}}.\phi[\mathbf{t}[\bar{\mathbf{x}}]]$

3

Ł

negative function encoding for $t[\bar{x}]$

positive encoding for other function appl.

- E-Matching (almost) like in SMT-solvers
- But: Choice of triggers has no effect on completeness!

Example:

- Relational function encoding + hypertableau = E-Matching
- Implementation in progress . . .
- E-Matching made respectable?

Future work, open questions:

- Formal completeness proof for Princess hypertableau rule
- When to use e-matching, when to use free variables?
- Relational encoding vs. native functions
- Partial functions vs. total functions

Thanks for your attention!