Automating Prover Development for Non-Classical Logics

Renate A. Schmidt
School of Computer Science
The University of Manchester
6 September 2013

Outline

1. Automated prover generation
2. The MetTeL tool
3. Termination via blocking
4. Rule refinement
Introduction

The toolset of every logician, and everyone using logic, should include at least one prover.

- Too few people use provers and too few provers are available.
- Implementing a prover is not straightforward and time-consuming.

Solution

Use prover engineering platforms or prover generators:

- LoTREC: The Tableau Work Bench

Automated prover generation
The MetTeL project

EPSRC project
Started 2011
Dmitry Tishkovsky, Mohammad Khodadadi

Design objectives

- Easy to use
- Reliable
- General and flexible
- Easily extended and integrated

Semantic labelled tableau

- Aim: construct a model or refute given formulae
- Goal-directed
- Rules break down formulae
- Rules for each logical operator
- Branching rules → derivations are trees
Using MetTeL to generate prover for S4

\[ S4 = \text{modal logic } K \text{ in which the accessibility relation } \mathcal{R} \text{ is a pre-order} \]

// Input file for MetTeL
// to generate prover for modal logic S4
specification S4;

options{
    name.separator =
    // tableau.rule.delimiter =;
    // tableau.rule.branch.delimiter =||
    // tableau.rule.premise.delimiter =/;
    // list.left.delimiter =
    // list.right.delimiter =
    // branch.bound =
}

Input file for MetTeL (cont’d): The tableau language

syntax S4 { // tableau language for logic S4
    sort formula, world; // declare two sorts

    // tableau formulae are labelled formulae
    formula at = '@' world formula;

    // connectives of the logic
    formula true = 'true';
    formula negation = '~' formula;
    formula box = '[ ]' formula;
    formula disjunction = formula ' | ' formula;

}  

- Any connectives of fixed finite arity can be defined
Input file for MetTeL (cont’d): The tableau language

```plaintext
// explicit accessibility relation
formula relation = 'R' '(', 'world', ',', 'world', ')';

// witnesses for diamond formulae
world succ = 'f' '(', 'world', ',', 'formula', ')';

// for blocking
formula equality = '[' 'world' '=' 'world' ']';
```

- **equality** = MetTeL keyword, used to define equality symbol in the tableau language

---

Input file to MetTeL (cont’d): The tableau calculus

```plaintext
tableau S4 { // tableau calculus
@s P @s (¬P)         // Closure rule
/s
@s (¬true)          // Decomposition rules
/priority 1 $;
@s (¬P)
/s @s P
@s (P|Q)
/s @s P @s Q
@s (¬(P|Q))
/s @s (¬P) @s (¬Q)
@s ([]P) R(s,t)
/s @t P
@s (¬([]P))
/s R(s,f(s,P)) @f(s,P) (¬P) // f(s,P) is newly created R-successor
/priority 7 $;
}
```
Input to MetTeL (cont’d): The tableau calculus

```java
@s P  // Reflexivity
 / R(s, s) priority 1 $
R(s, t) R(t, u) // Transitivity
 / R(s, u) priority 2 $

@s P @t Q // Blocking rule
 / [s=t] $ \neg([s=t]) priority 6 $
// R(s, t) // Ancestor blocking
// / [s=t] $ \neg([s=t]) priority 6 $
```

Running MetTeL

```bash
> java -jar mettel2.jar -i ~/work/mettel/S4/spec
```
creates the prover.

Running the generated prover:

```bash
> java -jar S4.jar
```
with

```java
@s p
@s (~p)
```
produces

**Unsatisfiable.**
**Contradiction:** `[( @ s p ), ( @ s (~ p ) )]`
Web interface www.mettel-prover.org

- Modal Logics
- Description logics, incl. $ALBO^{id}$
- Intuitionistic propositional logic
- Three valued Lukasiewicz logic
- Simple properties of lists
- Hybrid modal logics & counting operators
- Multi-agent interrogative-epistemic logics with privacy & sequential queries
- Temporal logic
- Testing rule admissibility
- ...

Additional features and functionalities

- Rule applications are controlled via rule priorities
- Equality reasoning via ordered term rewriting
- DFLR and BF search strategies
- Generic blocking mechanism
- Dynamic backtracking
- Random formula generation
- Conflict directed backjumping
- Support for benchmarking
Generic blocking mechanism

General idea of blocking
Use the tableau procedure to find finite models through identifying terms or reusing terms

Unrestricted blocking

(ub)  
\[ s \approx t \mid s \not\approx t \]

\( s \approx t \) is a trigger for ordered rewriting \( s \rightarrow t \), if \( s \succ t \)

Termination condition
Apply (ub) rule eagerly for all distinct pairs of terms

Example using unrestricted blocking

Example: \( \Box \neg \Box p \) is \( S4 \)-satisfiable

\[ b = f(p, a) \]

Assume \( a \approx b \)
Rewrite \( b \rightarrow a \)
Example using unrestricted blocking

\[(\text{ub})\quad s \approx t \mid s \not\approx t\]

Example: \(\square \neg \square p\) is S4-satisfiable

\[\neg \square p\]
\[R(a, b)\]
\[b : \neg p\]
\[a \approx b\]

\[a = f(p, a)\]

\[b = f(p, a)\]

\[a \approx b\]

Soundness, completeness and termination

Theorem
- \(\text{Tab} + (\text{ub})\) is a sound and complete, if \(\text{Tab}\) is sound and complete.
- \(\text{Tab} + (\text{ub})\) is terminating, if the logic has the finite model property.

\(\text{Tab}\) is terminating if for any finite set \(N\), every open tableau \(\text{Tab}(N)\) has a finite open branch.

Semantic labelled tableaux can decide
- Numerous modal, description and hybrid logics, incl. \(ALBO^{id}\)
- Two-variable fragment of first-order logic
Restricting the application of blocking

Adding premises/side-conditions restricts the application of the (ub) rule

**Predecessor blocking**

\[
\frac{R(s, t)}{s \approx t \mid s \not\approx t}
\]  

(ub-pred)

Gives ancestor blocking for S4

**Theorem**

- \( \text{Tab} \) extended with any such restricted blocking rule is sound and complete, if \( \text{Tab} \) is sound and complete.

Exclude a set from blocking

Let \( S \) be a finite set of terms

\[
\frac{R(s, t)}{s \approx t \mid s \not\approx t \mid t \not\in S}
\]  

(ub-noS)

**Theorem**

- \( \text{Tab} + (\text{ub-noS}) \) is sound, complete and terminating, if \( \text{Tab} \) is sound and complete and \( \text{Tab} + (\text{ub}) \) is terminating.
Unrestricted blocking vs. ub-noS

- Experimental results for description logic SHOI
- Manchester corpus: ~4500 OWL ontologies
- $S = \text{ABox elements}$

Rule refinement

Moving formulae from conclusion positions to premise positions

- Reduces branching
- Search space is smaller

$$\begin{align*}
\frac{s : \Box \phi}{\neg R(s, t) \mid t : \phi} \quad \text{refines to} \quad \frac{s : \Box \phi, R(s, t)}{t : \phi} \\
\frac{\neg R(s, t) \mid \neg R(t, u) \mid R(s, u)}{R(s, u)} \quad \text{refines to} \quad \frac{R(s, t), R(t, u)}{R(s, u)} \\
\frac{s : \phi \lor \psi}{s : \phi \mid s : \psi} \quad \text{refines to} \quad \frac{s : \phi \lor \psi, s : \neg \phi}{s : \psi}
\end{align*}$$
Atomic rule refinement

Theorem

- Replacing a rule by its atomic rule refinement preserves soundness and completeness. Holds for any semantic tableau calculus.

Moving negated atoms to premise positions

\[
\frac{s : \neg p \lor \psi}{s : \neg p, s : \psi}
\]

replaced by

\[
\frac{s : \neg p \lor \psi, s : p}{s : \psi}
\]

Separate rules for all TBox statements $A \sqsubseteq \forall R.B$

\[
\frac{s : \neg A \sqcup \forall R.B}{s : A, R(s, t)}
\]

replaced by

\[
\frac{s : A, R(s, t)}{t : B}
\]

More experimental results for SHOIQ

Effect of atomic rule refinement for TBox statements
Ongoing and future work

- Implement tableau calculus synthesis
- Study restricted forms of blocking; decidability; automatic use of appropriate blocking
- Stronger conditions/principles for automated rule refinement
- Incorporate techniques for automatically computing correspondence properties
Thank you!

Dmitry Tishkovsky
Mohammad Khodadadi
Fabrizio Principato
Jacobus Meulen
Tomas Alijevas

Michal Zawidzki
Stefan Minica
Clare Dixon, Boris Konev
John Stell, David Rydeheard