Generating Tableau Provers Using \textsc{Mettel}²

\textit{Tutorial}

Dmitry Tishkovsky and Renate Schmidt

School of Computer Science
The University of Manchester

\textsc{The 22nd International Conference}
on \textsc{Automated Reasoning with Analytic Tableaux and Related Methods}

16 – 19 September 2013
Outline

1. Introduction
2. Overview and motivation
3. Distribution kit and installation
4. Demo
5. Specification
   - Syntax specification
   - Tableau specification
   - Options
6. Optimisations and controlling derivations
   - Search strategies
   - Rule priorities
7. Rewriting
8. Blocking
   - Unrestricted blocking rule
   - Specifying the blocking mechanism
9. Further tricks
   - Labelled tableaux
   - Signed tableaux
   - Embedding sorts
   - Eliminating expressions
   - Reducing branching factor
   - Tabular logics
10. Testing and benchmarking utilities
11. Conclusion
Outline

1. Introduction
2. Overview and motivation
3. Distribution kit and installation
4. Demo
5. Specification
   - Syntax specification
   - Tableau specification
   - Options
6. Optimisations and controlling derivations
   - Search strategies
   - Rule priorities
7. Rewriting
8. Blocking
   - Unrestricted blocking rule
   - Specifying the blocking mechanism
9. Further tricks
   - Labelled tableaux
   - Signed tableaux
   - Embedding sorts
   - Eliminating expressions
   - Reducing branching factor
   - Tabular logics
10. Testing and benchmarking utilities
11. Conclusion
MetTeL 2 online demo

Reading the system description for MetTeL may help you in using the following tool.

Show quick help

1. Logic’s syntax

You may select one of the predefined syntax for following list:

User Defined syntax ...

/** every syntax starts with 'specification'
 * keyword followed by the name of logic */

specification myLogic;

/** immediately after there should be
 * a block for introducing the syntax */

syntax myLogic{

/** syntax definition should be initiated with definition of sort */

}
Outline

1. Introduction
2. Overview and motivation
3. Distribution kit and installation
4. Demo
5. Specification
   - Syntax specification
   - Tableau specification
   - Options
6. Optimisations and controlling derivations
   - Search strategies
   - Rule priorities
7. Rewriting
8. Blocking
   - Unrestricted blocking rule
   - Specifying the blocking mechanism
9. Further tricks
   - Labelled tableaux
   - Signed tableaux
   - Embedding sorts
   - Eliminating expressions
   - Reducing branching factor
   - Tabular logics
10. Testing and benchmarking utilities
11. Conclusion
Different applications require different (logical) formalisms
Logics need reasoning tools
Implementation of provers is expensive
Altering existing provers is hard
Translational approach requires additional knowledge and skills for the user

Solution
Generation of a prover code from a specification of a logic.
Goals and Objectives of Implementation of MetTeL²

- An easy to use prover generator
- Modularity of generated code
- Hierarchy of public JAVA classes and interfaces that can be easily extended and integrated with other systems
Outline

1. Introduction
2. Overview and motivation
3. Distribution kit and installation
4. Demo
5. Specification
   - Syntax specification
   - Tableau specification
   - Options
6. Optimisations and controlling derivations
   - Search strategies
   - Rule priorities
7. Rewriting
8. Blocking
   - Unrestricted blocking rule
   - Specifying the blocking mechanism
9. Further tricks
   - Labelled tableaux
   - Signed tableaux
   - Embedding sorts
   - Eliminating expressions
   - Reducing branching factor
   - Tabular logics
10. Testing and benchmarking utilities
11. Conclusion
Requirements and Installation

Requires JAVA. Tested on JRE 1.6 and 1.7.

Installation

- Place all the files from mettel-2.0-XXX.zip into some directory, e.g. ~/mettel2.
**mettel-2.0-XXX.zip**

- **mettel2.jar** — main jar-file for the generator, includes all the packages, depends on the ANTLR libraries:
  - antlr.jar,
  - antlr3.jar,
  - antlr3-runtime.jar,
  - stringtemplate4.jar,

and the CSV formatting library:

- opencsv.jar — required for running benchmark suites.

- **mettel2-core.jar** — required for running the generated provers, depends on:
  - antlr3-runtime.jar — required for running parsers generated by ANTLR.

- **mettel2-util.jar** — required for running expression generators and benchmark suites.
Generating and Running Prover

- Prepare a specification file \texttt{S4.s}.
- Generate the prover:
  \begin{verbatim}
  > java -jar ~/mettel2/mettel2.jar -i S4.s
  \end{verbatim}
- Place the generated \texttt{S4.jar} into \texttt{~/mettel2}.
- Run the prover:
  \begin{verbatim}
  > java -jar ~/mettel2/S4.jar
  \end{verbatim}
Package Structure of Generated Provers

**S4.jar**

- Syntax related classes: `S4.language.S4`
- Tableau related classes: `S4.tableau.S4`
- Executable classes:
  - Main class for the prover: `S4.tableau.S4.S4TableauProver`
  - Main class for running benchmark suites: `S4.tableau.S4.S4Benchmark`
  - Generator of random expressions for specified syntax: `S4.language.S4.util.S4RandomExpressionGenerator`
- Additional resources and examples: `etc`
**Specification**

**Short syntax**

```plaintext
specification <name>;
options{
 ...
}
syntax <name>{
 ...
}
tableau <name>{
 ...
}
```

**Full syntax**

```plaintext
specification <path>;
options{
 ...
}
syntax <name>
options{
 ...
}
}
syntax <name> in syntax <name>
options{
 ...
}
}
tableau <name> in syntax <name>
options{
 ...
}
}
<other syntax and tableau blocks>
```
Inside Syntax Specification

- Declaration of sorts of a (multi-sorted) propositional language:

  
  sort formula, world;

- Connective specifications as simple BNF statements:

  formula at = ’ @ ’ world formula;
  formula disjunction = formula ’ | ’ formula;

Formula examples

\[
\begin{align*}
& P \lor \neg Q, \ @\ell P, \\
& @\ell (P \lor \neg Q), \\
& @\ell \neg (p \lor (P \lor \neg Q)), \\
& @\ell \neg p \lor P \lor \neg Q \\
\end{align*}
\]
**Example**

\[ \phi \overset{\text{def}}{=} p \mid \bot \mid \neg \phi \mid \Diamond \phi \mid \@ w \phi \mid \phi \lor \phi \mid w \approx w \mid R(w, w) \]

\[ w \overset{\text{def}}{=} i \mid f(w, \phi) \]

**syntax S4**

- **sort** formula, world;
- formula false = 'false';
- formula negation = '~' formula;
- formula diamond = '<>' formula;
- formula at = '@' world formula;
- formula disjunction = formula ' | ' formula;
- formula equality = '[' world '=' world ']'; //Equality
- formula relation = 'R' '(' world ',' world ')'; //Relation
- world f = 'f' '(' world ',' formula ')'; //Skolem function


Inside Tableau Specification

- Tableau rule declaration in a premise-conclusion syntax:

\[
@l (P \lor Q) / @l P $| @l Q $;
\]

- Branch separator $|$ and rule separator $;$

- Rule priorities:

\[
@l (P \lor Q) / @l P \quad \text{priority} \quad 2 \quad @l Q
\]

- Default priority is 0
Options for Tableau Specification

- Redefinable separators $|$ and $;$

```plaintext
options{
...
tableau.rule.delimiter =;
tableau.rule.branch.delimiter =||
}
...

tableau S4{
...
  @| (P | Q) / @| P || @| Q priority 3;
}
```

- The delimiter / between premises and conclusions can be redefined via `tableau.rule.premise.delimiter` property.

- Must be redefined in the global `options` block or in the `options` block for the corresponding syntax.
Example

**tableau** S4{
  @i ~(~P) / @i P  \textit{priority} 1; //Double—negation removal
  @i(P|Q) / @i P || @i Q \textit{priority} 3; //Disjunction rule
  @i~(P|Q) / @i~P @i~Q \textit{priority} 1; //"Conjunction" rule
  @i<>P / R(i,f(i,P)) @f(i,P)P \textit{priority} 7; //Diamond rule
  @i~(<>P) R(i,j) / @j~P \textit{priority} 2; //"Box" propagation rule
  @i P / R(i,i) \textit{priority} 1; //Reflexivity
  R(i,j) R(j,k) / R(i,k) \textit{priority} 2; //Transitivity
  @i P @i~P / \textit{priority} 0; //Closure rule
  R(i,j) / [i=j] || ~([i=j]) \textit{priority} 6; //Ancestor blocking rule
  ~([i=i])/ \textit{priority} 0; //Closure rule for inequality
}

Notice separators in tableau rules!
options{
    branch.bound=
    branch.selection.strategy=

    equality.keywords={equality}

    tableau.rule.delimiter=$;
    tableau.rule.branch.delimiter=$|
    tableau.rule.premise.delimiter=/
}

Outline

1. Introduction
2. Overview and motivation
3. Distribution kit and installation
4. Demo
5. Specification
   - Syntax specification
   - Tableau specification
   - Options
6. Optimisations and controlling derivations
   - Search strategies
   - Rule priorities
7. Rewriting
8. Blocking
   - Unrestricted blocking rule
   - Specifying the blocking mechanism
9. Further tricks
   - Labelled tableaux
   - Signed tableaux
   - Embedding sorts
   - Eliminating expressions
   - Reducing branching factor
   - Tabular logics
10. Testing and benchmarking utilities
11. Conclusion
Features of Generated Provers

- Dynamic backtracking
- Conflict directed backjumping
- DFLR or BF search strategy
- Rule applications are controlled via rule priorities
- Equality reasoning via backward and forward rewriting
D. Tishkovsky and R. Schmidt

DFLR and BF Search Strategies

- Option for depth-first left-to-right search strategy:

```java
branch.selection.strategy = \n    mettel.core.tableau.MettelSimpleLIFOBranchSelectionStrategy
```

- Some logics, e.g. \( \mathcal{ALBO}^{id} \), require fair branch selection strategy for termination.

- Option for breadth-first search strategy:

```java
branch.selectionstrategy = \n    mettel.core.tableau.MettelSimpleFIFOBranchSelectionStrategy
```

- Other implementations of the interface `MettelBranchSelectionStrategy` are allowed.
A rule can be selected only if the queue of expressions associated with the rule is not empty (rule is *selectable*).

Selection must be fair within each priority group: if a rule is selectable then it will be selected eventually.

A rule can be selected only if all rules with smaller priority values (higher priority) are not selectable.

Note that selectable rule is not necessarily applicable.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Implemented Rule Selection Algorithm

Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
Example

In order to achieve that the closure rule is applied immediately after any new information is added to a branch assign to the closure rule the priority value 0 and to all other rule values higher than 0, e.g., 1.
A Guideline for Choosing Rule Priorities

Given two rules $\rho$ and $\rho'$, set priority value of $\rho$ less than the priority value of $\rho'$ if

- $\rho$ has lower branching factor than $\rho'$, or
- branching factors of $\rho$ and $\rho'$ are equal but $\rho$ has less premises.

Additional optimisation is possible, if each set of expressions has a complexity measure.

Example

tableau S4{
    @i ~(~P) / @i P priority 1; //Double–negation removal
    @i (P|Q) / @i P || @i Q priority 3; //Disjunction rule
    @i ~(P|Q) / @i~P @i~Q priority 1; //"Conjunction" rule
    @i<>P / R(i,f(i,P)) @f(i,P)P priority 7; //Diamond rule
    @i~(<>)P R(i,j) / @j~P priority 2; //"Box" propagation rule
    @i P / R(i,i) priority 1; //Reflexivity
    R(i,j) R(j,k) / R(i,k) priority 2; //Transitivity
    @i P @i~P / priority 0; //Closure rule
    R(i,j) / [i=j] || ~(i=j) priority 6; //Ancestor blocking rule
    ~(i=j) //priority 0; //Closure rule for inequality
}
Why Rewriting?

- Considerably simplifies derivations:
  - Exhaustive rewriting:
    
    \[ f(i, P) \xrightarrow{R} i \text{ then } f(f(i, P), P) \xrightarrow{R} i. \]
  
  - On-the-fly simplification — forward rewriting.

- Rewrite system can be efficiently maintained.

- Equality reasoning is required for the generic blocking mechanism.
A lexicographic path ordering \( \prec \) on all expressions of current tableau branch.

- Rewrite relation \( \mathcal{R} \).
- Forward rewriting: every new expression in a branch is rewritten wrt \( \mathcal{R} \).
- Backward rewriting is triggered by equality expressions:
  - every equality expression \( E(\alpha, \beta) \) in current branch is oriented wrt \( \prec \), and
  - either \( \alpha \xrightarrow{\mathcal{R}} \beta \) or \( \beta \xrightarrow{\mathcal{R}} \alpha \) is added to \( \mathcal{R} \), and
  - all the expressions in the branch are rewritten wrt \( \mathcal{R} \).
Option `equality.keywords`:
Default: `equality.keywords = {equality}`

```plaintext
options{
  equality.keywords = {equality, equivalence}
}
```

Corresponding name in a binary connective specification:

```plaintext
syntax SomeLogic{
...
  formula equality = ' [ nominal '=' nominal ] ';
  formula equivalence = formula '<->' formula;
}
```
There are periodicities in tableau derivations. They are usually detected with some form of loop checking mechanisms:
- subset or equality blocking,
- ancestor or anywhere blocking,
- static or dynamic blocking,
- pattern-based blocking, etc.

The standard loop-checking mechanisms are tied to particular logics.

In METTE\textsc{l}², a generic blocking mechanism, called unrestricted blocking mechanism, can be specified.
Unrestricted Blocking Rule

$\text{(ub): } x \approx y \mid x \not\approx y$

Termination Condition

In every open branch there is some node from which point onwards, all possible applications of the (ub) rule have been performed before any application of any term-generating rule.

- Early blocking:
  
  Standard loop checking:

  - Relies on tableau backtracking
  - Provides termination for logics with FMP
Unrestricted Blocking Rule

(ub): \( x \approx y \ | \ x \not\approx y \)

Termination Condition

In every open branch there is some node from which point onwards, all possible applications of the (ub) rule have been performed before any application of any term-generating rule.

- Early blocking:

Standard loop checking:

- Relies on tableau backtracking
- Provides termination for logics with FMP
Unrestricted Blocking Rule

\[
(\text{ub}): \frac{x \approx y}{x \not\approx y}
\]

Termination Condition
In every open branch there is some node from which point onwards, all possible applications of the (ub) rule have been performed before any application of any term-generating rule.

- Early blocking:

  Standard loop checking:

- Relies on tableau backtracking
- Provides termination for logics with FMP
Unrestricted Blocking Rule

(ub): \[ x \approx y \mid x \not\equiv y \]

Termination Condition
In every open branch there is some node from which point onwards, all possible applications of the (ub) rule have been performed before any application of any term-generating rule.

- Early blocking:

Standard loop checking:

- Relies on tableau backtracking
- Provides termination for logics with FMP
Unrestricted Blocking Rule

\[(ub): \frac{x \approx y \mid x \not\approx y}{\text{Termination Condition}}\]

Termination Condition

In every open branch there is some node from which point onwards, all possible applications of the (ub) rule have been performed before any application of any term-generating rule.

- Early blocking:
- Standard loop checking:
- Relies on tableau backtracking
- Provides termination for logics with FMP
Unrestricted Blocking Rule

\[(ub): \frac{x \approx y}{x \not\approx y}\]

Termination Condition

In every open branch there is some node from which point onwards, all possible applications of the (ub) rule have been performed before any application of any term-generating rule.

- Early blocking:

  Standard loop checking:

  ![Diagram of a standard loop checking](Image)

- Relies on tableau backtracking
- Provides termination for logics with FMP
Unrestricted Blocking Rule

\[(\text{ub}): \quad x \approx y \mid x \not\approx y\]

Termination Condition

In every open branch there is some node from which point onwards, all possible applications of the (ub) rule have been performed before any application of any term-generating rule.

- Early blocking:

Standard loop checking:

- Relies on tableau backtracking
- Provides termination for logics with FMP
Unrestricted Blocking Rule

(ub): \[ x \approx y \mid x \not\approx y \]

Termination Condition

In every open branch there is some node from which point onwards, all possible applications of the (ub) rule have been performed before any application of any term-generating rule.

- Early blocking:

Standard loop checking:

- Relies on tableau backtracking
- Provides termination for logics with FMP
Unrestricted Blocking Rule

\[(\text{ub}): \ x \approx y \mid x \not\approx y\]

Termination Condition

In every open branch there is some node from which point onwards, all possible applications of the (ub) rule have been performed before any application of any term-generating rule.

- Early blocking:

Standard loop checking:

- Relies on tableau backtracking
- Provides termination for logics with FMP
Unrestricted Blocking Rule

\[(ub): x \approx y \mid x \not\approx y\]

Termination Condition

In every open branch there is some node from which point onwards, all possible applications of the \((ub)\) rule have been performed before any application of any term-generating rule.

- Early blocking:

  Standard loop checking:

  \[i \neq j\]

  \[\square \diamond p \not\rightarrow p\]

- Relies on tableau backtracking
- Provides termination for logics with FMP
Specifying the Unrestricted Blocking Mechanism

- Ensure that equality reasoning is turned on.
- Append equality expressions to the syntax specification.
- Specify the unrestricted blocking rule:

```plaintext
tableau S4{
  ...
  @i P @j Q / [i=j] || ~[i=j];
}
```

- Satisfy the termination condition by using rule priorities:

```plaintext
tableau S4{
  ...
  @i<>P / R(i,f(i,P)) @f(i,P)P priority 7; //Diamond rule
  ...
  @i P @j Q / [i=j] || ~(i=j)) priority 6; //Blocking rule
}
```
Example

syntax S4{
  sort formula, world;
  formula false = 'false';
  formula negation = '~' formula;
  formula diamond = '<>' formula;
  formula at = '@' world formula;
  formula disjunction = formula ' | ' formula;
  formula equality = '[ ' world '=' world ' ] '; //Equality
  formula relation = 'R' '(' world ', ' world ') '; //Relation
  world f = 'f' '(' world ', ' formula ') '; //Skolem function
}

tableau S4{
  @i (~P) / @i P priority 1; //Double—negation removal
  @i(P|Q) / @i P || @i Q priority 3; //Disjunction rule
  @i~(P|Q) / @i~P @i~Q priority 1; //"Conjunction" rule
  @i<>P / R(i,f(i,P)) @f(i,P)P priority 7; //Diamond rule
  @i~(<>P) R(i,j) / @j~P priority 2; //"Box" propagation rule
  @i P / R(i,i) priority 1; //Reflexivity
  R(i,j) R(j,k) / R(i,k) priority 2; //Transitivity
  @i P @i~P / priority 0; //Closure rule
  R(i,j) / [i=j] || ~(i=j) priority 6; //Ancestor blocking rule
  ~(i=i) / priority 0; //Closure rule for inequality
}
Outline

1. Introduction
2. Overview and motivation
3. Distribution kit and installation
4. Demo
5. Specification
   - Syntax specification
   - Tableau specification
   - Options
6. Optimisations and controlling derivations
   - Search strategies
   - Rule priorities
7. Rewriting
8. Blocking
   - Unrestricted blocking rule
   -Specifying the blocking mechanism
9. Further tricks
   - Labelled tableaux
   - Signed tableaux
   - Embedding sorts
   - Eliminating expressions
   - Reducing branching factor
   - Tabular logics
10. Testing and benchmarking utilities
11. Conclusion
Labelled Tableaux

sort labelledFormula, formula, label;
...
labelledFormula at = ' @' label formula;
...

Example

syntax S4{
sort tableauFormula, formula, world;
formula false = 'false';
formula negation = '~' formula;
formula diamond = '<>' formula;
tableauFormula at = '@' world formula;
tableauFormula equality = '[' world '=' world ']' ; //Equality
tableauFormula inequality = '~' '[' world '=' world ']' ; //Equality
tableauFormula relation = 'R' '(' world ',' world ')' ; //Relation
world f = 'f' '(' world ',' formula ')' ; //Skolem function
}
Signed Tableaux

```plaintext
sort signedFormula, formula;
...
signedFormula trueValue = 'T' formula;
signedFormula falseValue = 'F' formula;
...
```

Example

```plaintext
syntax Int{
  sort formula, individual;
  formula false = 'false';
  formula trueValue = 'T' formula;
  formula falseValue = 'F' formula;
  formula at = '@' individual formula;
  formula negation = '~' formula;
  formula conjunction = formula ' & ' formula;
  formula disjunction = formula ' | ' formula;
  formula implication = formula ' -> ' formula;
  individual successorTmp = 'f' '(' individual ', ' formula ', ' formula ')';
  formula relation = 'R' '(' individual ', ' individual ')';
  formula equality = '[' individual ' = ' individual ']';
}
```
sort formula, proposition;
...
formula proposition = ' # ' proposition;
...

syntax IEL{
  sort formula, individual, prop, agent;
  formula true = ' true ';
  formula false = ' false ';
  formula singleton = '{ ' individual ' } ';
  formula atom = ' # ' prop;
  formula negation = ' ~ ' formula;
  formula diamondQ = '<q>' agent formula;
  formula diamondK = '<k>' agent formula;
  formula diamondX = '<x>' agent formula;
  formula at = '@' individual formula;
  formula query = '[ ? ' formula ' ]' agent formula;
  formula resol = '[ ! ]' agent formula;
  formula disjunction = formula ' | ' formula;
  formula equality = '[ ' individual ' = ' individual ' ] ';
  individual fq = 'fq' '(' individual ',' agent ',' formula ') ';
  individual fk = 'fk' '(' individual ',' agent ',' formula ') ';
  individual fx = 'fx' '(' individual ',' agent ',' formula ') ';
}
Example

```plaintext
options{
    ...
    equality.keywords={equality,equivalence}
}

syntax LTL{
    ...
    formula equivalence = formula ' <-> ' formula;
    ...
    formula eventuality = ' E ' ( ' world ' , ' formula ' ) ' ;
}

tableau LTL{
    ...
    @i (<>P) / @i (E(i,P)) priority 1;
    @i (E(j,P)) / @i P ((E(j,P)) <-> (<>P)) || @f(i) (E(j,P)) priority 7;
    ...
    @i E(j,P) / priority 8; // "Bad" loop check
}
```
Move negated conclusion to premises if it cannot be instantiated to become “non-atomic”.

\[ \neg R(i, j) \mid \neg R(j, k) \mid R(i, k) \]

\[ \neg R(i, j) \mid @i \neg p \]

\[ \neg R(i, j) \mid @j \neg p \]
Move negated conclusion to premises if it cannot be instantiated to become “non-atomic”.

\[
\frac{R(i, j), \ R(j, k)}{R(i, k)}
\]

\[
\frac{\Diamond i \neg p}{\neg R(i, j) \mid \neg j \neg p}
\]
Move negated conclusion to premises if it cannot be instantiated to become “non-atomic”.

\[
\frac{R(i,j), \ R(j,k)}{R(i,k)}
\]

\[
\frac{\@_i \neg \diamond p}{\neg R(i,j) \mid \@_j \neg p}
\]
Move negated conclusion to premises if it cannot be instantiated to become “non-atomic”.

\[
\frac{R(i, j), \ R(j, k)}{R(i, k)}
\]

\[
\frac{@i \neg \Box p, \ R(i, j)}{@j \neg p}
\]
specification Lukasiewicz3;
syntax Lukasiewicz{
    sort valuation, formula;
    valuation true = 'T' formula | unknown = 'U' formula | false = 'F' formula;
    formula true = 'true' | false = 'false';
    formula negation = '~' formula;
    formula conjunction = formula ' & ' formula;
    formula disjunction = formula ' | ' formula;
    formula implication = formula ' -> ' formula;
}
tableau Lukasiewicz{
    T P F P / priority 0 $; T P U P / priority 0 $;
    U P F P / priority 0 $; U P F P / priority 0 $;
    T ~P / F P priority 1 $; U ~P / U P priority 1 $; F ~P / T P priority 1 $;
    T (P & Q) / T P T Q priority 2 $; F (P & Q) / F P $ | F Q priority 1 $;
    U (P & Q) / T P U Q $ | U P T Q $ | U P U Q priority 3 $;
    T (P | Q) / T P $ | T Q priority 2 $; F (P | Q) / F P F Q priority 1 $;
    U (P | Q) / F P U Q $ | U P F Q $ | U P U Q priority 3 $;
    F (P -> Q) / T P F Q priority 1 $; U (P -> Q) / U P F Q $ | T P U Q priority 2 $;
    T (P -> Q) / T Q $ | F P $ | U P U Q priority 3 $;
    T false / priority 0 $; U false / priority 0 $;
    U true / priority 0 $; F true / priority 0 $;
}
Outline

1. Introduction
2. Overview and motivation
3. Distribution kit and installation
4. Demo
5. Specification
   - Syntax specification
   - Tableau specification
   - Options
6. Optimisations and controlling derivations
   - Search strategies
   - Rule priorities
7. Rewriting
8. Blocking
   - Unrestricted blocking rule
   - Specifying the blocking mechanism
9. Further tricks
   - Labelled tableaux
   - Signed tableaux
   - Embedding sorts
   - Eliminating expressions
   - Reducing branching factor
   - Tabular logics
10. Testing and benchmarking utilities
11. Conclusion
Copy `S4RandomExpressionGenerator.properties` from `output/S4/etc` and edit it.

```properties
world.f.frequency = 1
world.variable.frequency = 1
world.depth = 1
world.variables = i, j, k
world.variables.number = 3

world.generate = 0

world.top.connectives =

formula.false.frequency = 1
formula.negation.frequency = 1
formula.diamond.frequency = 1
formula.at.frequency = 1
formula.disjunction.frequency = 1
formula.worldEquality.frequency = 1
formula.relation.frequency = 1

formula.variable.frequency = 1
formula.depth = 10
formula.variables = p, q, r
formula.variables.number = 3

formula.generate = 1000

formula.top.connectives = at
```
Generating Random Problems and Benchmarking

- Generate problems

```bash
```

- Run a benchmark

```bash
```
Outline

1. Introduction
2. Overview and motivation
3. Distribution kit and installation
4. Demo
5. Specification
   - Syntax specification
   - Tableau specification
   - Options
6. Optimisations and controlling derivations
   - Search strategies
   - Rule priorities
7. Rewriting
8. Blocking
   - Unrestricted blocking rule
   - Specifying the blocking mechanism
9. Further tricks
   - Labelled tableaux
   - Signed tableaux
   - Embedding sorts
   - Eliminating expressions
   - Reducing branching factor
   - Tabular logics
10. Testing and benchmarking utilities
11. Conclusion
**Summary**

- MetTeL\(^2\) is easy-to-use and flexible tool
- MetTeL\(^2\) allows specification of various types of tableau calculi
- Case studies:
  - Boolean logic
  - S4
  - IPC
  - \(ALCO\)
  - \(ALBO^{\text{id}}\)
  - \(SHOI\)
  - LTL and temporal logic with cardinality constraints
  - Tree-valued Łukasiewicz logic
  - K with global counting operators
  - Interrogative epistemic logics
  - etc
Local aims:

- Further optimisations of generated provers.
- More options to allow the user to control the generation process.
- Extending specification languages.
- Increase level of abstraction of the tableau core.

An ultimate goal is to enable automatic generation of provers from other definitions of logics. In particular:

- Implement the tableau synthesis framework for synthesis of tableau calculi from semantics of logics.
- Investigate a possibility to generate tableau provers from Hilbert axiomatisations.
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!
Thank You!