

METTEL

A Tableau Prover with
Logic-Independent Inference Engine

User Manual v.1

M. Khodadadi, R. A. Schmidt, D. Tishkovsky

www.mettel-prover.org

School of Computer Science
The University of Manchester



The University of Manchester

Contents

Quick Start	1
1 System Overview	1
2 METTEL Language	3
2.1 Formulae	3
2.2 Nominals	3
2.3 Attributes	3
2.4 Parameters	5
2.5 Role or Relational terms	5
3 Problem Set Definition	7
4 Tableau Calculus Specification	9
4.1 User-defined tableau calculus	9
4.2 Predefined Tableau Calculi	11
5 Running METTEL	19
6 METTEL Output	21
7 METTEL Errors	23
7.1 Unexpected token	23
7.2 Unexpected character	23
7.3 Missing argument	23
8 METTEL Examples	25
8.1 Priority of operators	25
8.2 Unsatisfiable example for classical propositional logic	25
8.3 Satisfiable example for classical propositional logic	25
8.4 Unsatisfiable example for hybrid logic $HL(@, \vee, \circ)$	26
8.5 Satisfiable example for hybrid logic $HL(@, \vee, \circ)$	26
8.6 Unsatisfiable example for hybrid logic $HL(@, \vee, \circ, u)$	27
8.7 Satisfiable example for hybrid logic $HL(@, \vee, \circ, u)$	27
8.8 Unsatisfiable example for metric logic	27
8.9 Satisfiable example for metric logic	28
8.10 Unsatisfiable example for metric logic with topology operator	28

8.11	Satisfiable example for metric logic with topology operator	28
8.12	Unsatisfiable example for description logic \mathcal{ALBO}	29
8.13	Satisfiable example for description logic \mathcal{ALBO}	29
8.14	Unsatisfiable example for description logic \mathcal{ALBO}^{id}	30
8.15	Satisfiable example for description logic \mathcal{ALBO}^{id} with identity	30
8.16	Example of incomplete tableau calculus	31
8.17	Example of using constant role	31
9	More Tips	33
	Bibliography	37

Quick Start

A tableau calculus and a problem set are the inputs required to run METTEL. The tableau calculus can be either selected from one of general predefined calculi mentioned in Section 4.2 on Page 11; or can be defined as explained in Section 4.1 and then given to METTEL via an input file using the `-tbl` option.

The problem set can be either given to the system in an input file using the option `-i` or through the standard input stream. Input from standard input stream needs to be terminated by `Ctrl + D`.

The command for running METTEL has the following form:

```
java -jar mettel.jar
    [-tbl <calculus-file-name> | <predefined tableau option> ]
    [-o <output-file-name>]
    [-e <error-file-name>]
    [-i <problem-set-file-name>]
    problem set in standard input if no input given
```

Table 5.1 on Page 19 demonstrates different ways of running METTEL via the command line. The following example uses the predefined tableau calculus for *Classical propositional logic* (option `-bool`) for testing the satisfiability of $P \ \& \ Q \ \& \ \sim P$.

```
java -jar mettel.jar -bool
P & Q
~ P
```

(CTRL +D)

The output for this example is **Unsatisfiable**. This means the given formula was found to be unsatisfiable using the in-built tableau calculus of classical propositional logic. The output is going to be **Satisfiable** when the problem set is *satisfiable* with respect to the calculus. Chapter 8, gives more examples of running METTEL.

There are errors that can terminate the execution of METTEL such as insufficient argument. As an example, if we use `-i` option when running METTEL, it should *immediately* follows by the file-name that contains the formulae for the problem set. Otherwise, METTEL prompts the user with the following message:

```
Input file name required
```

Chapter 7 on Page 23 describes the different kinds of errors that might occur when running METTEL.

One

System Overview

METTEL is a tableau prover for various modal, intuitionistic, hybrid, description and metric logics. The user has the option of using one of the several predefined tableau calculi or defines their own tableau calculus as input. METTEL implements generic loop-checking mechanisms and the *unrestricted blocking* mechanism [5] to enforce termination. This makes METTEL useful for experimenting and testing tableau calculi for non-classical logics for which no sound and complete tableau calculi are known or for which no implementation exists.

METTEL decides the following logics:

- Classical propositional logic [7]
- Intuitionistic propositional logic
- Hybrid logic HL(@, u) with the universal modality [8]
- The logic \mathcal{MT} of metric and topology
- All sublogics of the description logic $\mathcal{ALBO}^{\text{id}}$ [5]

At the moment, METTEL is the only tableau prover that can decide logics of metric and topology and description logics with role negation [5]. Using the facility to specify tableau rules METTEL can be used as a prover for many other more expressive or even undecidable logics.

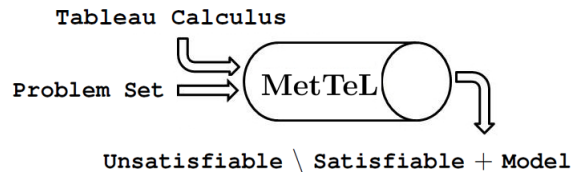


Figure 1.1: How METTEL works

As shown in Figure 1.1, METTEL takes a tableau calculus and a problem set as input. The tableau calculus is given by a set of tableau rules and the problem set is given by a set of formulae. The prover outputs **Unsatisfiable** if the set of formulae is not satisfiable with respect to the tableau calculus. The prover outputs **Satisfiable** and a model found for the set of formulae satisfiable with respect to the tableau calculus.

Two

METTEL Language

The language of METTEL is a many-sorted language. Namely, It consists of five sorts: formulae, nominal terms (including Skolem terms), attributes, rational parameters from metric logic and relational formulae (or roles).

2.1 Formulae

The primitives and operators that can be used to form formulae from other formulae, roles, attributes and nominals are shown in Table 2.1. The first column gives the names of the primitives and operators. The second column shows how these primitives and operators can be used to build formulae syntax in METTEL. The third and fourth columns give examples of both formulae in METTEL syntax and standard syntax from literature.

The priority of the operators is as follows "**forall, exists, @, false, true, ~, <<, &, |, ->**" and "**<->**". Within METTEL formula, the first argument of box and diamond which is role, always appear within a pair of curly brackets, e.g. **{Role}**.

2.2 Nominals

In METTEL, nominals have one of the two forms as indicated in Table 2.2. The second column specifies their syntax and the third column gives examples.

By default, nominal symbols denote constants in the problem set and variables in the tableau calculus definition. They can be declared as being constant at the beginning of the tableau calculus definition, using the **constant** keyword.

```
constant: i, j ;
```

Nominal symbols can be declared as being variables at the beginning of the problem set, using **variable** keyword.

```
variable: i, j ;
```

We use lower-case identifiers for nominals to distinguish them from propositional variables, which should be specified by upper-case identifiers.

2.3 Attributes

The attributes sort in METTEL consists of an attribute which declared by a lower-case identifier. Table 2.3 contain the syntax and an example for attributes.

Primitive/Operator	Syntax	Sample	Standard Syntax
TRUE	TRUE	TRUE	\top
FALSE	FALSE	FALSE	\perp
Nominal	see Table 2.2	j	j
Propositional Variable	Upper-case Identifier	P	P
Not	~ formula	~ P -> Q	$\neg P \rightarrow Q$
Or	formula formula	P Q	$P \vee Q$
And	formula & formula	P & Q	$P \wedge Q$
Implication	formula -> formula	P -> Q	$P \rightarrow Q$
Double Implication	formula <-> formula	P&Q <-> C D	$P \wedge Q \Leftrightarrow C \vee D$
Closer	formula << {attribute}formula	Q << {a} P&Q	$Q \leftarrow_a P \wedge Q$
AT	@ nominal formula	@i P ~Q	$@_i P \vee \neg Q$
Diamond or Existential restriction	E{ role } formula	E{R} Q	$\exists R.Q$
	exists { role } formula	exists{R} Q	$\exists R.Q$
	some { role } formula	some{R} Q	$\exists R.Q$
	dia { role } formula	dia{R} Q	$\diamond R.Q$
	diamond { role } formula	diamond{R} Q	$\diamond R.Q$
Existential modality	some formula	some ~P	$\exists \neg P$
	exists formula	exists ~P	$\exists \neg P$
Box or Universal restriction	A{ role } formula	A{R} (P->Q)	$\forall R.(P \rightarrow Q)$
	forall { role } formula	forall{R} (P->Q)	$\forall R.(P \rightarrow Q)$
	all { role } formula	all{R} (P->Q)	$\forall R.(P \rightarrow Q)$
	box { role } formula	box{R} (P->Q)	$\square R.(P \rightarrow Q)$
Universal modality	forall formula	forall (P & ~Q)	$\forall (P \wedge \neg Q)$
	all formula	all (P & ~Q)	$\forall (P \wedge \neg Q)$

Table 2.1: Syntax of formulae

Type	Syntax	Sample
Nominal	lower-case Identifier	j
Skolem term	Function-name[args seprated by comma]	f[i,P,R]

Table 2.2: Syntax of nominals

Primitive	Syntax	Sample
Attribute	lower-case Identifier	a

Table 2.3: Syntax of attributes

2.4 Parameters

Table 2.4 shows what primitives and operators we do allow in METTEL parameter. The first column shows that, parameters in METTEL are either constant or variable or the result of sum operator on two other parameters. The second column shows how these parameters can be written. Finally, the last column demonstrate a sample of these primitives and operators in METTEL syntax.

Primitive/Operator	Syntax	Sample
Contant Parameter	lower-case Identifier	c
Variable Parameter	Rational Number	0.1
Parameter Sum	parameter + parameter	c + 0.1

Table 2.4: Syntax of parameters

2.5 Role or Relational terms

Table 2.5 defines the syntax for role or relational formulae. The first column gives the names of the primitives and operators. The second column shows how these primitives and operators can be used to build role syntax in METTEL. The third and fourth columns give examples of both formulae in METTEL syntax and standard syntax from literature. The last three rows of Table 2.5 contain the roles for metric logic.

The priority of operators is as follows "+, |, &, ||, ., ;, *, ~, - and ^".

Since variable roles and propositional variables have the same syntax, METTEL distinguish them based on the context. Constant roles can be specified by lower-case identifiers.

Primitive/Operator	Syntax	Sample	Standard Syntax
Constant Role	lower-case Identifier	r	r
Variable Role	upper-case Identifier	R	R
Complement	~role	~R	$\neg R$
Reflex-Transitive closure	role*	R*	R^*
Inverse	role- role^	R- R^	$R\sim$ $R\sim$
Union	role + role	R + S	$R \vee S$
	role role	R S	$R \vee S$
Intersection	role role	R S	$R \wedge S$
	role & role	R & S	$R \wedge S$
Composition	role . role	R.S	$R \circ S$
	role ; role	R;S	$R \circ S$
Identity Role	id	id	
Metric Role	lower-case Identifier < parameter	x<a	$\exists(x < a)$
Topology Role	lower-case Identifier <= parameter	x<=a	$\exists(x \leq a)$
Topology Role	topo	topo	

Table 2.5: Syntax of roles

Three

Problem Set Definition

A problem set is a set of formulae. METTEL attempts to determine the satisfiability of the given problem set by applying the rules of a particular tableau calculus. The user can provide the problem set either in the standard input or in a file.

Here are three examples of formulae specifiable in METTEL:

```
box box{~(~R | S)} FALSE  
( @i dia{R} j & @j dia{R} k ) -> @i dia{R} k  
exists{a<x;a<y} P -> exists{a<(x+y)} P
```

The first formula encodes role inclusion $R \sqsubseteq S$ in terms of the box operator, role negation and role union. The second formula defines R as a transitive relation in terms of the diamond operator, formula conjunction and formula implication. The last formula expresses the triangle property of metric relations (if there is a path to a property P consisting of two distances for an attribute a which are strictly less than x and y respectively, then there is an a -distance to P which is strictly less than $x + y$).

By default, nominals are *constants*. For using nominals as variables, they need to be declared as being variable at the beginning of the problem set as follows.

```
variable: i , k ;
```

The BNF specification of problem set is given in Figure 3.1.

```

Problem set := variable: lower-case identifier (, lower-case identifier)? ;)?
              formula formula*

formula :=    formula <-> formula      |  formula -> formula      |
              formula | formula        |  formula & formula      |
              '~' formula               |  @ nominal formula      |
              TRUE | true              |  FASLE | false        |
              E{ role } formula         |  A{ role } formula      |
              nominal                   |  propositionalVariable  |
              << { attribute } formula  |
              (exists | some | dia | diamond) ({ role })? formula |
              (forall | all | box) ({ role })? formula

nominal :=   lower-case identifier      |
              lower-case identifier [ formula formula* ];

role :=      role (+ | |) role          |  role (& | ||) role      |
              role (; | .) role        |  role *                  |
              ~ role                    |  role (^ | -)           |
              Upper-case identifier     |  lower-case identifier  |
              lower-case identifier (< | <=) parameter

```

Figure 3.1: BNF of Problem set

Four

Tableau Calculus Specification

The user can either choose one of the predefined tableau calculi, or define a new calculus in the tableau specification language of METTEL.

4.1 User-defined tableau calculus

The syntax for specifying tableau rules follows the standard premises/conclusions notation found in the literature.

Premises / Conclusion ;

That is, premises and conclusions are separated by / and each rule is terminated by ;.

There are three types of tableau rules: non-branching rules, branching rules and closure rules. Non-branching rules have only one branch in their conclusion. All the conclusion formulae used to be placed within a pair of brackets, as follows:

RulePremises / { RuleConclusion } ;

Branching rules are the rules that split the current branch into more than one branch. A separated pair of curly brackets is required for each branch. The user can also optionally put a | between each branch for better readability. In general branching rules look like this:

RulePremises / { FirstConclusion } | ... | { LastConclusion } ;

Finally, closure rules have **FALSE** in their conclusion. They have the following form:

RulePremises / { FALSE } ;

Formally, the tableau specification input is defined by the BNF grammar given in Figure 4.1. Here, *formula* refers to any formula in METTEL syntax as defined in Table 2.1. Examples are shown in Table 4.1. On the left, the rules are written in METTEL syntax and on the right they are written in standard syntax found in the literature.

Tableau Calculus := (constant: Lower-case identifier (, lower-case identifier)? ;)?
Tableau Rule ; (Tableau Rule;)*

Tableau Rule := formula (,formula)* / { formula (,formula)* }
(|? { formula (,formula)* })* ;

Figure 4.1: BNF of Tableau Calculus

METTEL Syntax	Standard Syntax
$@i (P Q) / \{ @i P \} \mid \{ @i Q \};$	$\frac{@i(P \vee Q)}{@i P \mid @i Q}$
$@i (\sim P \ \& \ \sim Q) / \{ @i \sim P, @i \sim Q \};$	$\frac{@i(\neg P \wedge \neg Q)}{@i \neg P, @i \neg Q}$
$@i \sim(\text{dia}\{R\} P), @i \text{dia}\{R\} j / \{ @j \sim P \};$	$\frac{@i \neg \diamond RP, @i \diamond Rj}{@j \neg P}$
$@i \text{dia}\{R\} P / \{ @f[i, R, P] P, @i \text{dia}\{R\} f[i, R, P] \};$	$\frac{@i \diamond RP}{@_{f[i, R, P]} P, @i \diamond R f[i, R, P]}$
$@i P, @i \sim P / \{ \text{FALSE} \};$	$\frac{@i P \ @i \neg P}{\perp}$

Table 4.1: Comparing METTEL syntax with the syntax use in literature for defining tableau rules

The first rule in Table 4.1 is an *or* rule, defined as a branching rule which creates a splitting point adding $@i P$ to the left branch and $@i Q$ to the right branch. The other rules are non-branching rules except the last rule which is a closure rule and has **FALSE** in the conclusion. The second rule is an instance of the standard *and* rule. It extends the current branch by adding both $@i \sim P$ and $@i \sim Q$. We use comma to separate the formulae in one branch. The third rule is the standard *box* rule rewritten in terms of the negated diamond. The fourth rule is a standard diamond rule which generate a new nominal in the conclusion. The last rule, as we said earlier, is a closure rule.

Note the use of the Skolem term $f[i, R, P]$ in the fourth rule. f is a *function name* which is applied to a set of arguments. Here, $f[i, R, P]$ represents a nominal newly created as witness for the premise $@i \text{dia}\{R\} P$. The arguments i, r, p given uniquely link the term to the premise (as i, r, p are the symbols occurring in the premise). Using Skolem terms in this way means that it is possible to see how nominals created. For example it can be read off from the term $f[f[a, r1, p], r2, q]$ that it represents a world reachable from world a via a $r1$ transition followed by a $r2$ transition. It also used, since it is easy to track the ancestors of any newly generated nominal.

Since role and propositional variable have similar syntax, we require the user to specify the sorts of arguments when a Skolem function is used for the first time. In the Skolem term, sort specification has the form of **SORT:ARGUMENT** for each argument, e.g., $f[\text{role: } R, \text{prop: } P, \text{nominal: } i]$. The sort can be **param** for parameter, **attr** for attribute, **nom** for nominal, **prop** for propositional and **role** for relation. So, the fourth rule of Table 4.1 would actually have to be written as $f[\text{nom: } i, \text{role: } R, \text{prop: } P]$.

METTEL assumes the following conventions. In the user defined tableau, all the nominals are variable nominals unless they have been declared as constant nominal. Any constant nominals declaration should precise the rules.

constant: i , k ;

By default, constant roles are denoted by identifiers which starting with a lower-case letter, e.g. r . Constant roles cannot be substituted with any other role. Therefore, in rules, any

Option	Tableau calculus for
-bool	Classical propositional logic.
-hl	Hybrid logic HL(@) with relational union and composition (Default).
-hlu	Like -hl plus the universal modality.
-met	Metric logic without the 'closer' operator.
-topo	Metric logic with the topology operator but without the 'closer' operator [4].
-albo	\mathcal{ALBO} with the unrestricted blocking rule mechanism [5].
-alboid	Like -albo plus identity

Table 4.2: Predefined tableau calculi

constant role can only be matched to itself.

Rule application priority

METTEL group the provided rules in two three groups: 1.Non-branching, 2.Non-Branching and generating new nominal and 3.branching rules. Rules are applied in this order. The order in which rules inside a group are applied is determined by the order in which the rules are defined in the input file.

Switching off preprocessing

By default, METTEL does some preprocessing on the user-defined tableau calculus. This may sometimes result in unexpected behaviour. The preprocessing stage can be omitted using the `-tbl-npp` option instead of `-tbl` for inputting the tableau calculus. Therefore, the command for running METTEL will be as follows:

```
> java -jar mettel.jar [-tbl-npp <t-fname>] [-i <in-fname>]
```

4.2 Predefined Tableau Calculi

There are several predefined tableau calculi in METTEL. All these tableau calculi are sound, complete and decision procedure for their targeted logic. For using these calculi, METTEL needs to be run with the appropriate option as listed in Table 4.2.

Command line execution is via:

```
> java -jar mettel.jar [<tableau option>] [-i <in-fname>]
```

More information about running METTEL can be found in Section 5. In the remainder of this section, we provide the preprocessing that applies on problem sets beside the tableau rules for each calculus.

Preprocessing on the problem set for predefined tableau calculi

When User select one of the predefined tableau calculi other than `-bool`, formulae in the problem set preprocessed before tableau derivation. If we assume P and Q are formula,

preprocessing apply the following rules recursively:

$$\begin{array}{lll}
TRUE \Rightarrow \neg FALSE & \Box P \Rightarrow \neg(\Diamond \neg P) & \forall^u P \Rightarrow \neg(\exists^u \neg P) \\
\neg\neg P \Rightarrow P & P \leftrightarrow Q \Rightarrow (P \rightarrow Q) \wedge (Q \rightarrow P) & P \rightarrow Q \Rightarrow \neg P \vee Q \\
P \wedge Q \Rightarrow \neg(\neg P \vee \neg Q) & (R^*)^* \Rightarrow R^* & \neg\neg R \Rightarrow R \\
(R^\sim)^\sim \Rightarrow R & (R \vee S)^\sim \Rightarrow (R^\sim \vee S^\sim) & (R \circ S)^\sim \Rightarrow (R^\sim \circ S^\sim) \\
(R \wedge S)^\sim \Rightarrow (R^\sim \wedge S^\sim) & (R^*)^\sim \Rightarrow (R^\sim)^* & (\neg R)^\sim \Rightarrow \neg(R^\sim)
\end{array}$$

Classical propositional logic

Formulae in *classical propositional logic* are built using the Boolean operators \wedge , \vee , \rightarrow , \leftrightarrow and \neg . For using the predefined tableau calculus for *classical propositional logic*, option `-bool` should be used when running METTEL.

```
> java -jar mettel.jar -bool
```

By selecting this calculus, METTEL applies the rules from Figure 4.2 for the tableau derivation.

$$\begin{array}{cccc}
\frac{P, \neg P}{\perp} & \frac{\neg\neg P}{P} & \frac{P \wedge Q}{P, Q} & \frac{\neg(P \wedge Q)}{\neg P \mid \neg Q} \\
\frac{P \rightarrow Q}{\neg P \mid Q} & \frac{\neg(P \rightarrow Q)}{P, \neg Q} & \frac{P \vee Q}{P \mid \neg P, Q} & \frac{\neg(P \vee Q)}{\neg P, \neg Q} \\
\frac{P \leftrightarrow Q}{\neg P \vee Q, P \vee \neg Q} & & \frac{\neg(P \leftrightarrow Q)}{\neg P \wedge Q \mid P \wedge \neg Q} &
\end{array}$$

Figure 4.2: Predefined tableau rules for classical propositional logic

Hybrid logic with relational union and composition, HL(@, \vee , \circ)

The hybrid logic HL(@, \vee , \circ) is the extension of modal logic with the satisfaction operator @, relation union \vee and relational composition \circ . The satisfaction operator is used to assert that a formula is true at a world, and it has the form @_{*i*} where *i* is a nominal. Option `-hl` in METTEL selects the predefined tableau calculus for HL(@, \vee , \circ).

```
> java -jar mettel.jar -hl
```

When using this option, METTEL uses the rules from Figure 4.3 for the tableau derivation.

This tableau calculus is the default tableau calculus for METTEL, this means if user runs METTEL without providing any option for tableau calculus, it automatically select Hybrid logic HL(@) with relational union and composition tableau calculus.

Hybrid logic with relational union, composition and universal modality, HL(@, \vee , \circ , *u*)

For modal logic and hybrid logic problems with the universal modality the option `-hlu` should be used. In this case the predefined tableau calculus for logics with relational composition and universal modality us used.

Metric logic without the ‘closer’ operator

Metric logic extends the hybrid logic $HL(@, \vee, \circ, u)$ with the *distance operators* $\exists^{<a}$ and $\exists^{\leq a}$, for $a \in \mathbb{Q}^{>0}$. For using the predefined tableau calculus for this logic, option `-met` should be used when running METTEL.

`> java -jar mettel.jar -met`

When using this option, METTEL uses the rules from Figure 4.5 for the tableau derivation. The rules for this tableau calculus are the one shown in Figure 4.4 plus the rules for

$$\begin{array}{c}
\frac{}{\textcircled{i}\perp} \quad \frac{\textcircled{i}j}{\textcircled{j}i} \quad \frac{\textcircled{i}\neg j}{\textcircled{j}\neg i} \\
\frac{\textcircled{i}P}{\textcircled{i}i} \quad \frac{\textcircled{i}P, \textcircled{i}\neg P}{\perp} \quad \frac{\textcircled{i}(\textcircled{j}P)}{\textcircled{j}P} \quad \frac{\textcircled{i}\neg(\textcircled{j}P)}{\textcircled{j}\neg P} \\
\frac{\textcircled{i}(P \vee Q)}{\textcircled{i}P \mid \textcircled{i}\neg P, \textcircled{i}Q} \quad \frac{\textcircled{i}\neg(P \vee Q)}{\textcircled{i}\neg P, \textcircled{i}\neg Q} \\
\frac{\textcircled{i}\exists R.P}{\textcircled{i}\exists R.P} \quad \frac{\textcircled{i}\neg(\exists R.P), \textcircled{i}\exists R.j}{\textcircled{j}\neg P} \\
\frac{\textcircled{i}\exists R.\sim.j}{\textcircled{j}\exists R.i} \quad \frac{\textcircled{i}\neg(\exists R.\sim.P), \textcircled{j}\exists R.i}{\textcircled{j}\neg P} \\
\frac{\textcircled{i}\exists(R \vee S).P}{\textcircled{i}\exists R.P \mid \textcircled{i}\exists S.P} \quad \frac{\textcircled{i}\neg\exists(R \vee S).P}{\textcircled{i}\neg\exists R.P, \textcircled{i}\neg\exists S.P} \\
\frac{\textcircled{i}\exists(R \circ S).j}{\textcircled{i}\exists R.f[i, R, S, j], \textcircled{i}\exists S.j} \quad \frac{\textcircled{i}\neg\exists(R \circ S).P}{\textcircled{i}\neg(\exists R.\exists S.P)} \\
\frac{\textcircled{i}\exists^u P}{\textcircled{i}f[i, P]P} \quad \frac{\textcircled{i}\neg\exists^u P, \textcircled{j}j}{\textcircled{j}\neg P} \\
\frac{\textcircled{i}\exists^{a \leq q} j}{\textcircled{j}\exists^{a \leq q} i} \quad \frac{\textcircled{i}\exists^{a < q} j}{\textcircled{j}\exists^{a < q} i} \quad \frac{\textcircled{i}\exists^{a \leq q} j}{\textcircled{i}\exists^{a \leq q} j} \quad \frac{\textcircled{i}\neg(\exists^{a < q} P)}{\textcircled{i}\neg(\exists^{a \leq q} P)} \\
\frac{\textcircled{i}i, \textcircled{j}\exists^{a < q} P}{\textcircled{i}\exists^{a < q} i} \quad \frac{\textcircled{i}i, \textcircled{j}\exists^{a \leq q} P}{\textcircled{i}\exists^{a \leq q} i} \\
\frac{\textcircled{i}i, \textcircled{j}\forall^{a < q} P}{\textcircled{i}\exists^{a < q} i} \quad \frac{\textcircled{i}i, \textcircled{j}\forall^{a \leq q} P}{\textcircled{i}\exists^{a \leq q} i} \\
\frac{\textcircled{i}\exists^{a < q} j, \textcircled{j}\exists^{a < r} k}{\textcircled{i}\exists^{a < (q \vee r)} k} \quad \frac{\textcircled{i}\exists^{a \leq q} j, \textcircled{j}\exists^{a < r} k}{\textcircled{i}\exists^{a < (q \vee r)} k} \\
\frac{\textcircled{i}\exists^{a < q} j, \textcircled{j}\exists^{a \leq r} k}{\textcircled{i}\exists^{a < (q \vee r)} k} \quad \frac{\textcircled{i}\exists^{a \leq q} j, \textcircled{j}\exists^{a \leq r} k}{\textcircled{i}\exists^{a \leq (q \vee r)} k} \\
\frac{\textcircled{i}\neg(\exists^{a < q} P), \textcircled{i}\exists^{a < r} j}{\textcircled{j}\neg P} \quad \frac{\textcircled{i}\neg(\exists^{a \leq q} P), \textcircled{i}\exists^{a < r} j}{\textcircled{j}\neg P} \\
\frac{\textcircled{i}\neg(\exists^{a < q} P), \textcircled{i}\exists^{a \leq r} j}{\textcircled{j}\neg P} \quad \frac{\textcircled{i}\neg(\exists^{a \leq q} P), \textcircled{i}\exists^{a \leq r} j}{\textcircled{j}\neg P}
\end{array}$$

Figure 4.5: Predefined tableau rules for Metric logic without the ‘closer’ operator

metric reflexivity, metric symmetry, metric inclusion, metric triangle equality and metric propagation.

Metric logic with the topology operator but without the ‘closer’ operator

Formulae in this logic are built using the Boolean operators \wedge , \vee , \rightarrow and \neg , the *distance operators* $\exists^{<a}$ and $\exists^{\leq a}$, for $a \in \mathbb{Q}^{>0}$, and the topological *interior* and *closure operators*. For using the predefined tableau calculus for *metric logic with the topology operator but without the ‘closer’*, option `-topo` should be used when running METTEL.

```
> java -jar mettel.jar -topo
```

When using this option, METTEL uses the rules from Figure 4.6 for the tableau derivation. The rules for this tableau calculus are the one shown in Figure 4.5 plus the rules for topology reflexivity, topology transitivity, topology inclusion, topology triangle equality and topology propagation.

The first versions of METTEL were designed for this logic. METTEL is still the only tableau prover that can decide it [4]. METTEL has been compared on a class of metric and topology problems with the first-order resolution provers SPASS and VAMPIRE, where it performed better [4].

Description logic \mathcal{ALBO}

\mathcal{ALBO} is an expressive description logic extending the basic description logic \mathcal{ALC} with the Boolean role operators, inverse of roles, domain and range restriction operators and nominals (individuals) [5]. The reasoning complexity for \mathcal{ALBO} is NEXPTIME-COMplete. It subsumes Boolean modal logic and the two-variable fragment of first-order logic. For using the predefined tableau calculus for *description logic \mathcal{ALBO}* , option `-albo` should be used when running METTEL.

```
> java -jar mettel.jar -albo
```

When using this option, METTEL uses the rules from Figure 4.7 for the tableau derivation.

The rules for this tableau calculus are the one shown in Figure 4.3 plus the rules for complement role and the unrestricted blocking rule. This option use *unrestricted blocking rule mechanism* as a replacement for loop checking mechanism to ensure termination [5]. This predefined calculus was introduced in [5] where it was shown that it provides a decision procedure for \mathcal{ALBO} . The only other decision procedure currently known for \mathcal{ALBO} is an approach based on resolution by [11].

Description logic \mathcal{ALBO}^{id}

The logic for this option is an extension of \mathcal{ALBO} that we have introduced in the last section with *role identity*. For using the predefined tableau calculus for *description logic \mathcal{ALBO}^{id}* , option `-alboid` should be used when running METTEL.

```
> java -jar mettel.jar -alboid
```

When using this option, METTEL uses the rules from Figure 4.8 for the tableau derivation. The rules for this tableau calculus are the one shown in Figure 4.4 plus the rules for id role, role complement, role cut and nominal cut.

Five

Running METTEL

The input to METTEL is a tableau calculus specification and a problem set. The tableau calculus can be either selected from one of the predefined calculi, mentioned in Section 4.2, or can be defined as explained in Section 4.1, and then gives to METTEL in an input file. The problem set can be either provided to the system in an input file or through the standard input. The user needs to specify the end of the input from standard input stream by **Ctrl + D**.

Table 5.1 shows the different possibilities of running METTEL.

		Syntax and example
✓	✓	<code>java -jar mettel.jar [<tableau option>] [-i <in-fname>]</code> e.g. <code>java -jar mettel.jar -bool -i problemset1.p.mtl</code>
✓	✓	<code>java -jar mettel.jar [<tableau option>]</code> e.g. <code>java -jar mettel.jar -bool</code> <code>P & Q</code> <code>~ P</code> <code>(Ctrl + D)</code>
✓	✓	<code>java -jar mettel.jar [-tbl <t-fname>] [-i <in-fname>]</code> e.g. <code>java -jar mettel.jar -tbl tableau1.t.mtl -i probset1.p.mtl</code>
✓	✓	<code>java -jar mettel.jar [-tbl <t-fname>]</code> e.g. <code>java -jar mettel.jar -tbl tableau1.t.mtl</code> <code>P & Q</code> <code>~ P</code> <code>(Ctrl + D)</code>

Table 5.1: METTEL runtime options

Six

METTEL Output

METTEL is a refutation prover. This means when it outputs the **unsatisfiable**, the input problem is unsatisfiable with respect to the tableau calculus used. More precisely, when METTEL has constructed a closed tableau derivation, that is, in each branch a contradiction was found, METTEL outputs **unsatisfiable**. As an example, assume we have stored the sample tableau rules from Section 4.1 in the file `sampleTableau.t.mt1` and execute the following command:

```
> java -jar mettel.jar -tbl sampleTableau.t.mt1
   box{R} (Q&P) & dia{R}~Q
```

METTEL produces the output: **Unsatisfiable**.

On the other hand, when the input problem is satisfiable with respect to the tableau calculus, it outputs **satisfiable**. More precisely, when METTEL has constructed an open tableau derivation, that is, one complete branch without contradiction, METTEL outputs **satisfiable**. If we run METTEL again with the same calculus but input `box{R} (Q&P) & dia{R}P` as problem set the output is:

Satisfiable

```
MODEL: [ (@{f(n0,R,P)}P), (@{n0}(exists{R}f(n0,R,P))), (@{f(n0,R,P)}(~((~Q)
| (~P))))), (@{n0} (~((exists{R}((~Q)| (~P))) | (~(exists{R}P))))), (@{n0
} (~(exists{R}((~Q)| (~P))))), (@{n0}(exists{R}P)), (@{f(n0,R,P)}Q) ]
```

```
Compact Representation: [ (@{f(n0,R,P)}P), (@{n0}(exists{R}f(n0,R,P))), (@{
f(n0,R,P)}Q) ]
```

As can be seen in this example beside **Satisfiable** message, METTEL presents a model both in long and compact format. The provided *model* is the set of formulae which are produced by the application of the rules on the given problem set branch and did not result in closure.

Both the model and its compact representation are specified within a pair of square brackets. In the METTEL present nominals nominals are presented within a pairs of curly brackets in the output syntax.

In this example, in order to satisfy `dia{R}P`, METTEL creates a new nominal, namely the Skolem term `{f(n0,R,P)}`, which belong to `P` that is `(@{f(n0,R,P)}P)`. This nominal then connects to the root nominal which is captured by `@{n0}(exists{R}f(n0,R,P))`. Then box rule is applied and `Q & P` is added to the newly generated nominal, that is `@{f(n0,R,P)}(~((~Q)| (~P)))`. Since `P` is already true at this nominal, METTEL only adds `Q` to the set of formula true at this nominal, this is captured by `(@{f(n0,R,P)}Q)`.

Since every model can be represented only by nominals, relations and propositions that

are correct on them, METTEL also generates a *Compact Representation* of the model. This is done by selecting all the grounded literals.

Both output models follow the standard notation of prover outputs, which is responsible for the large number of parentheses and brackets.

By default, METTEL prints the output on the standard output. The output can be sent to a file by using option `-o` followed by the desired output file name when running METTEL.

```
java -jar mettel.jar -bool -i probl.p.mtl -o result-probset1.p.mtl
```

The execution of METTEL can be terminated by pressing CTRL+C.

Seven

METTEL Errors

When the execution of METTEL aborts, this may be due to the following errors: unexpected tokens, unexpected character and insufficient argument.

7.1 Unexpected token

When parsing a user-defined tableau file or problem set, if METTEL parse a word that is unexpected according to the BNF grammars provided in Figure 3.1 and Figure 4.1, it will throw an *Unexpected token* exception. The error message printed to standard output has the form:

```
== Unexpected token <source> =====  
(<line>:<column>) unexpected token: <token>  
<extra information>  
=====
```

Here, *<source>* specifies where the error happened which can be either in the user defined *tableau calculus* or in the *problem set*. The second line specifies the position of the unexpected token and the token itself. In some cases, METTEL provides some extra information to help the user find and rectify the error. For example, the following error message caused by attempting to specify a constant nominal in the problem set.

```
== Unexpected token Problem set =====  
(1:1) unexpected token: CONSTANT  
CONSTANT can be only used at the beginning of tableau definition.  
=====
```

7.2 Unexpected character

7.3 Missing argument

There are options that need to be followed by an argument. As an example, if we use `-tbl` option when running METTEL, it should *immediately* followed by the name of a file that contains the specification of the tableau calculus. Otherwise, METTEL gives the following error message:

```
Tableau file name required
```

The user should then run METTEL again and supply all the required arguments. Options such as `-i`, `-tbl-npp`, `-o` and `-e` similarly need to be followed by an argument. They should be followed by file name for the *problem set*, file name for the *user specified tableau*

calculus, desired file names for keeping the *output* and *errors*. If no argument is given, an error message is displayed.

Eight

METTEL Examples

In this section, we give examples to demonstrate the use of METTEL. The examples in Section 8.1-8.14, use predefined tableau calculus and the remaining examples use user-defined tableau calculus. Each example contains the running command, problem set as input stream and the output obtained. Some examples are accompanied by related notes and tips.

8.1 Priority of operators

User should be very careful when he is not sure about the priority of the operators. For example user may input $@n0 P \& Q$ for $@n0 (P \& Q)$, while it is actually interpreted as $@n1 (@n0 P) \& Q$ by METTEL where $n1$ is a new nominal.

 Unary operators have higher priority than binary operators.

8.2 Unsatisfiable example for classical propositional logic


At first we load the MetTel with the predefined *Classical propositional logic* tableau calculus by using the `-bool` option

```
> java -jar mettel.jar -bool
```

Then we type the formula $(P1 \mid P2) \& (\sim P1 \& \sim P3)$ followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

```
Unsatisfiable
```

This means the given formula was found to be unsatisfiable with respect to the tableau calculus of classical propositional logic.

 Problem sets in the input stream should be terminated with `Ctrl + D`.

8.3 Satisfiable example for classical propositional logic

At first we load the MetTel with the predefined *Classical propositional logic* tableau calculus by using the `-bool` option


```
> java -jar mettel.jar -bool
```

Then we type the formula $(P1 \mid P2) \& (\sim P1 \& \sim P3)$ followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

Satisfiable

MODEL: [((P1|P2) & ((~P1) & (~P3))), ((~P1) & (~P3)), (P1|P2), (~P1), (~P3), P2]

This means the given formula was found to be satisfiable with respect to the tableau calculus of classical propositional logic.

 METTEL does not produce a compact representation of the generated model for classical propositional logic.

8.4 Unsatisfiable example for hybrid logic HL(@, ∨, ◦)


At first we load the MetTeL with the predefined *Hybrid logic* HL(@, ∨, ◦) tableau calculus by using the `-hl` option

```
> java -jar mettel.jar -hl
```

Then we type the formula `@i ~ (forall{R} (P->Q) -> (forall{R} P->forall{R} Q))` followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

Unsatisfiable

This means the given formula was found to be unsatisfiable with respect to the tableau calculus of hybrid logic HL(@, ∨, ◦).

 METTEL ignores extra new lines and extra spaces. So the user can even have an extra line inside one formula or put more than one formula in one line

8.5 Satisfiable example for hybrid logic HL(@, ∨, ◦)

At first we load the MetTeL with the predefined *Hybrid logic* HL(@, ∨, ◦) tableau calculus by using the `-hl` option

```
> java -jar mettel.jar -hl
```


Then we type the formula `@i (forall{R} (P->Q) -> (forall{R} P->forall{R} Q)) // Comment on hybrid example` followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

Satisfiable

```
MODEL: [ (@{i} ((exists{R} (~((~P) | Q))) | ((exists{R} (~P)) | (~ (exists{R} (~Q)))))
, (@{i} (exists{R} (~((~P) | Q)))) , (@{i} (exists{R} n0)) ,
(@{i} i) , (@{n0} (~((~P) | Q))) , (@{n0} (~Q)) , (@{n0} P) , (@{n0} n0) ]
```

```
Compact representation: [ (@{n0} n0) , (@{i} i) , (@{n0} P) , (@{i} (exists{R} n0)) ,
(@{n0} (~Q)) ]
```

This means the given formula was found to be satisfiable with respect to the tableau calculus of hybrid logic HL(@, ∨, ◦).

 Both in problem set and tableau calculus input files and input stream, lines starting with `\\` are assumed to be comments

8.6 Unsatisfiable example for hybrid logic $HL(@, \vee, \circ, u)$


At first we load the MetTeL with the predefined *Hybrid logic* $HL(@, \vee, \circ, u)$ tableau calculus by using the `-hlu` option

```
> java -jar mettel.jar -hlu
```

Then we type the formula `forall A{R+S+T}~P & exists E{S+R}P` followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

```
Unsatisfiable
```

This means the given formula was found to be unsatisfiable with respect to the tableau calculus of hybrid logic $HL(@, \vee, \circ, u)$.

 There are five alternatives names for diamond the operator, and four alternatives name for the box operator, which all cause identical behaviour. This gives users the flexibility to use the names they prefer.

8.7 Satisfiable example for hybrid logic $HL(@, \vee, \circ, u)$

At first we load the MetTeL with the predefined *Hybrid logic* $HL(@, \vee, \circ, u)$ tableau calculus by using the `-hlu` option

```
> java -jar mettel.jar -hlu
```


Then we type the formula `~ forall (P & ~P)` followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

```
Satisfiable
```

```
MODEL:[(@{n0}n0), (@{n0}(exists ((~P)|P))),
(@{n1}(~P)), (@{n1}n1), (@{n1}((~P)|P))]
```

```
Compact representation:[(@{n0}n0),
(@{n1}(~P)), (@{n1}n1)]
```

This means the given formula was found to be satisfiable with respect to the tableau calculus of hybrid logic $HL(@, \vee, \circ, u)$.

 If there is a role after the keyword `exists` (`forall`), they form a diamond (box) operator. But if there is not any role there, it is an existential modality (universal modality) operator.

8.8 Unsatisfiable example for metric logic

At first we load the MetTeL with the predefined *Metric logic without the 'closer' operator* tableau calculus by using the `-met` option

```
> java -jar mettel.jar -met
```

Then we type the formula `~ ((forall {a<.3}P->forall{a<.2}forall{a<.1}P))` followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

```
Unsatisfiable
```


This means the given formula was found to be unsatisfiable with respect to the tableau calculus of metric logic without the 'closer' operator.

Satisfiable

```
MODEL: [ (@{n0}n0), (@{n0} (~P)), (@{n0} (exists{topo}n0)),
  (@{n0} ( (~ (exists{ a<=1/10 }P) ) | (exists{topo} (exists{ a<1/10 }P) ) ) ) ),
  (@{n0} (~ (exists{ a<=1/10 }P) ) ) ]
```

```
Compact representation: [ (@{n0}n0), (@{n0} (~P)) ]
```

This means the given formula was found to be satisfiable with respect to the tableau calculus of metric logic with the topology operator.

 In the model for metric logic problem sets, METTEL shows rational number in fraction format instead of float format.

8.12 Unsatisfiable example for description logic \mathcal{ALBO}


At first we load the MetTeL with the predefined *description logic \mathcal{ALBO}* tableau calculus by using the `-albo` option

```
> java -jar mettel.jar -albo
```

Then we type the formula `~(exists{R}i | exists{~R}i)` followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

Unsatisfiable

This means the given formula was found to be unsatisfiable with respect to the tableau calculus of description logic \mathcal{ALBO} .

 In METTEL `albo` and `alboid` use *unrestricted blocking* to ensure termination. The unrestricted blocking is based on the following rule:

$$(ub) : \frac{@l \{l\}, @l' \{l'\}}{@l \{l'\} | @l \neg \{l'\}}$$

8.13 Satisfiable example for description logic \mathcal{ALBO}

At first we load the MetTeL with the predefined *description logic \mathcal{ALBO}* tableau calculus by using the `-albo` option

```
> java -jar mettel.jar -albo
```


Then we type the formula `E{R}p , E{S}p , E{T}~E{T}E{S^}(p|~p), ~E{T}E{~T}~E{S^}(p|~p)` followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

Satisfiable

```
MODEL: [ (@{n0} (exists{R}p)), (@{n0} (exists{S}p)),
  (@{n0} (exists{T} ( ~ (exists{T} (exists{ (S-) } (p | (~p) ) ) ) ) ) ),
  (@{n0} (exists{T}n0)), (@{n0} (~ (exists{ (S-) } (p | (~p) ) ) ) ),
  (@{n0} (~ (exists{ (~T) } (~ (exists{ (S-) } (p | (~p) ) ) ) ) ) ),
  (@{n0} (~ (exists{T} (exists{ (S-) } (p | (~p) ) ) ) ) ),
  (@{n0} (~ (exists{T} (exists{ (~T) } (~ (exists{ (S-) } (p | (~p) ) ) ) ) ) ) ),
  (@{n0}n0) ]
```

Compact representation: $[(@\{n0\}(\text{exists}\{T\}n0)), (@\{n0\}n0), (@\{n0\}(\text{exists}\{R\}p)), (@\{n0\}(\text{exists}\{S\}p))]$

This means the given formula was found to be satisfiable with respect to the tableau calculus of description logic \mathcal{ALBO} .

 Formulae can be optionally separated with a comma.

8.14 Unsatisfiable example for description logic \mathcal{ALBO}^{id}


At first we load the MetTeL with the predefined *description logic* \mathcal{ALBO}^{id} tableau calculus by using the `-alboid` option

```
> java -jar mettel.jar -alboid
```

Then we type the formula `@i exists{id} P @i ~P` followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

Unsatisfiable

This means the given formula was found to be unsatisfiable with respect to the tableau calculus of description logic \mathcal{ALBO} with identity.

 In METTEL `id` is the predefined identity role.

8.15 Satisfiable example for description logic \mathcal{ALBO}^{id} with identity

At first we load the MetTeL with the predefined *description logic* \mathcal{ALBO}^{id} tableau calculus by using the `-alboid` option

```
> java -jar mettel.jar -alboid
```


Then we type the formula `~ (E{S + ~S} ~E{R} p | E{S + ~S} ~E{R} ~p)` followed by `Ctrl + D` (in Linux) to test whether it is satisfiable or not. The output is:

Satisfiable

```
MODEL: [(@{n0}(exists{R}(~p))), (@{n0}(exists{R}n0)),
(@{n0}(exists{R}p)), (@{n0}(exists{S}n0)), (@{n0}(exists{S}p)),
(@{n0} (~ ( (exists{ (S+ (~S)) } (~ (exists{R}p)) | (exists{ (S+ (~S)) }
(~ (exists{R} (~p)) )) )) ),
(@{n0} (~ (exists{ (S+ (~S)) } (~ (exists{R} (~p)) )))),
(@{n0} (~ (exists{ (S+ (~S)) } (~ (exists{R} p)) ))),
(@{n0} (~ (exists{ (~S) } (~ (exists{R} (~p)) )))),
(@{n0} (~ (exists{ (~S) } (~ (exists{R} p)) ))),
(@{n0} (~ (exists{S} (~ (exists{R} (~p)) )))),
(@{n0} (~ (exists{S} (~ (exists{R} p)) ))), (@{n0} (~p)), (@{n0}n0),
(@{p}(exists{R} (~p))), (@{p}(exists{R}n0)), (@{p}(exists{R}p)),
(@{p} (~n0)), (@{p}p)]
```

Compact representation: $[(@\{n0\}n0), (@\{n0\}(\text{exists}\{R\}p)), (@\{n0\}(\text{exists}\{S\}n0)), (@\{p\}p), (@\{p\}(\text{exists}\{R\}p)), (@\{n0\}(\sim p)), (@\{n0\}(\text{exists}\{S\}p)), (@\{n0\}(\text{exists}\{R\}n0)), (@\{p\}(\sim n0)), (@\{p\}(\text{exists}\{R\}n0))]$

This means the given formula was found to be satisfiable with respect to the tableau calculus of description logic \mathcal{ALBO} with identity.

 METTEL follows the priorities we have mentioned in Chapter 2 unless the user have specified it otherwise using parenthesis

8.16 Example of incomplete tableau calculus


This example shows how METTEL can be used for experimenting with a new tableau calculus. Assume that we have proposed the following incomplete calculus for description logic \mathcal{ALC} :

```
@i (P|Q) / { @i P } | { @i Q };
@i ~(dia{R} P), @i dia{R} j / { @j ~P };
@i dia{R} P / { @f[i,R,P] P, @i dia{R} f[i,R,P] };
@i P, @i ~P / { FALSE };
```

If we save this calculus into mytableau.tmtl, and run it on `box{R} (Q&P) & dia{R} ~Q` the output is:

```
Satisfiable
MODEL: [ (@{n0} (~((exists{R} ((~Q) | (~P))) | (~(exists{R} (~Q))))))]
Compact representation: []
```

The answer METTEL gives is incorrect since this problem set is actually unsatisfiable in description logic \mathcal{ALC} . This wrong answer is the result of specifying a tableau calculus which is not complete. One can see this invalidity more clearly by running METTEL on the same problem set using one of the applicable predefined calculi, e.g. using `-h1` option. By adding the rule `@i (~P & ~Q) / { @i ~P, @i ~Q }`; to the specification we can make the tableau calculi complete.

 For a complete tableau calculus, there should generally be rules for positive and negative occurrences of each operators in the logic.

8.17 Example of using constant role

As we have already mentioned, in a tableau calculus specification by default constant roles are denoted by identifiers which start with a lower-case letter, e.g. `r`. A constant role `r` occurring in a rules only matches with `r` itself. A variable role on the other hand matches with all formulae of sort role.

For example, assume that we want to use a specific role in our calculus which is functional. We can have a tableau rule specified just for this role using a constant role symbol as follows:

```
@i exists{r} j, @i exists{r} k /{@j k};
@i P, @i ~P / { FALSE };
```

In this calculus, the first rule ensures that `r` is a functional role. We test this calculus using following problem set.


```
@a exists {r} b
@a exists {r} c
```

```
@b Q
@c ~Q
```

The output for this problem set is **Unsatisfiable**. The nominal **b** is going to be equal to **c** as the result of the first rule application and then the closure rule is applied on **@b Q** and **@b ~Q**. If we rerun with the same calculus but with following problem set, the output will be **Satisfiable**, since the first rule is not applicable anymore.

```
@a exists {r2} b
@a exists {r2} c
@b Q
@c ~Q
```

If we want all the roles to be functional we should change the first rule to **@i exists{R} j, @i exists{R} k /{@j k};**

-  Constant roles should be used to specify properties for specific roles (e.g. hierarchy, inclusion between two roles).

Nine

More Tips

Maybe table of error messages + FAQ

List of Figures

1.1	How METTEL works	1
3.1	BNF of Problem set	8
4.1	BNF of Tableau Calculus	9
4.2	Predefined tableau rules for classical propositional logic	12
4.3	Predefined tableau rules for hybrid logic $HL(@, \vee, \circ)$	13
4.4	Predefined tableau rules for hybrid logic $HL(@, \vee, \circ, u)$	13
4.5	Predefined tableau rules for Metric logic without the 'closer' operator	14
4.6	Predefined tableau rules for Metric logic with the topology operator but without the 'closer' operator	16
4.7	Predefined tableau rules for description logic \mathcal{ALBO}	17
4.8	Predefined tableau rules for description logic \mathcal{ALBO}^{id}	17

List of Tables

2.1	Syntax of formulae	4
2.2	Syntax of nominals	4
2.3	Syntax of attributes	4
2.4	Syntax of parameters	5
2.5	Syntax of roles	6
4.1	Comparing METTEL syntax with the syntax use in literature for defining tableau rules	10
4.2	Predefined tableau calculi	11
5.1	METTEL runtime options	19

Bibliography

- [1] P. Abate and R. Goré. The Tableaux Work Bench. In *TABLEAUX'03*, vol. 2796 of *LNCS*, pp. 230–236. Springer, 2003.
- [2] S. Babenyshev, V. Rybakov, R. A. Schmidt, and D. Tishkovsky. A tableau method for checking rule admissibility in S4. *Electron. Notes Theor. Comput. Sci.*, 262:17–32, 2010.
- [3] O. Gasquet, A. Herzig, D. Longin, and M. Sahade. LoTREC: Logical tableaux research engineering companion. In *TABLEAUX'05*, vol. 3702 of *LNCS*, pp. 318–322. Springer, 2005.
- [4] U. Hustadt, D. Tishkovsky, F. Wolter, and M. Zakharyashev. Automated reasoning about metric and topology (System description). In *JELIA06*, vol. 4160 of *LNAI*, pp. 490–493. Springer, 2006.
- [5] R. A. Schmidt and D. Tishkovsky. Using tableau to decide expressive description logics with role negation. In *ISWC07*, vol. 4825 of *LNCS*, pp. 438–451. Springer, 2007.
- [6] R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. In *TABLEAUX'09*, vol. 5607 of *LNAI*, pp. 310–324. Springer, 2009.
- [7] M. Fitting. Basic modal logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming: Logical Foundations*, vol. 1, pp. 365–448. 1993.
- [8] P. Blackburn. Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto. In *Logic Journal of IGPL*, vol. 8, 2000.
- [9] R. A. Schmidt and D. Tishkovsky. Deciding *ALBO* with tableau. In *Proceedings of the 20th International Workshop on Description Logics (DL-2007)*, pp. 135–146., 2007.
- [10] M. Sheremet, D. Tishkovsky, F. Wolter, and M. Zakharyashev. From topology to metric: modal logic and quantification in metric spaces. In *Advances in Modal Logic*, vol. 6, pp. 429–448. 2006.
- [11] H. de Nivelle and I. Pratt-Hartmann. A Resolution-Based Decision Procedure for the Two-Variable Fragment with Equality. In *Automated Reasoning*, ed. GorÅl, Leitsch, and Nipkow, vol. 2083, pp. 2011–225. Springer, 2001.