DESCRIPTION LOGICS - THE LOTREC EXPERIENCE

Kris Oosting

Vrije Universiteit Amsterdam - Technical Artificial Intelligence, The Netherlands; kris.oosting@dfsxpertsys.com

ABSTRACT

LoTREC is a tool to explore tableau reasoning of many different logics. This paper describes the application of LoTREC to Description Logic $\mathcal{ALC}$. First the LoTREC strategies, rules, and connectors are described. Then the results of applying these strategies to given reasoning problems. Followed by a discussion about efficient ordering and blocking strategies. Finally, transitive role ($\mathcal{S}$) and role hierarchy ($\mathcal{H}$) extensions to $\mathcal{ALC}$ are described.

Key words: Description Logic, Tableau, Reasoning, LoTREC.

1. INTRODUCTION

The Logical Tableaux Research Engineering Companion, or LoTREC for short, is generic tableau prover built in Java. It does not only test satisfiability, but it also builds pre-models for the logic formulas. This paper describes the experiments with LoTREC as part of the Description Logics assignment of the Automated Reasoning in AI course. Section **??** describes the tableau algorithm for the LoTREC toolkit. Section **??** describes given reasoning problems and section **??** elaborates on strategies. Section **??** describes two extensions of $\mathcal{ALC}$ and finally section **??** gives the conclusion of this experiment. The appendix of this paper gives the descriptions of the LoTREC connectors, rules, strategies used in the experiments, and premodels of the reasoning problems.

2. TABLEAU ALGORITHM

To make LoTREC work, strategies had to be defined. These strategies were used to solve the logic formulas. Strategies consist of rules and rules use connectors to parse the input formula. Strategies can be compared with small programs which are described in more detail in section **??**. The first step in setting up the LoTREC tableau algorithm is to define the connectors. The second step is to define the rules, and the final step is to define the strategies.
The mapping of the Description Logics syntax to LoTREC syntax is shown in table **??**.

$\mathcal{ALC}$ syntax	LoTREC syntax	LoTREX Display
$\neg C$	not C	`not C`
$C \sqcap D$	and C D	`C and D`
$C \sqcup D$	or C D	`C or D`
$\exists r.C$	some r C	`R some C`
$\forall r.C$	only r C	`R only C`
$\top \equiv D$	tbox D	`⊤ = D`
$\top$	TOP	`TOP`
$\bot$	BOT	`BOT`

Table 1. Syntax mapping of DL to LoTREC.

The first word in the LoTREC syntax are called *connectors* and were defined in the tool. Table **??** shows the auxiliary connectors defined in LoTREC for handling the formula input. For example: the LoTREC input using the `input` connector `input T C` would display `INPUT: TBox = T; Concept = C`.

Name	Arity	Display
add	2	_ & _
input	2	`INPUT: Tbox = _; Concept = _`

Table 2. LoTREC connectors.

Three categories of rules were defined. Table **??** shows the rules of the first category that handle input. Table **??** shows the rules for handling classical propositional logic, and table **??** shows the rules for handling the more complex functions. These rules were the building blocks of the strategies.

The rules described in table **??** and table **??** represent the transformation rules of the tableau satisfiability algorithm. Table **??** shows the mapping of LoTREC rules to the tableau transformation rules as defined in Baader et. al. [?].

Rule Name	Description
InputRule	Expands the `input`, creates a new node, adds all formulas to this new node, and creates a link with label StartTableau.
AddRule	When an `add` connector is encountered, it adds the two variables that follow to the current node.
CopyT	Checks if the root node contains a `tbox` connector, checks if a node is linked through the StartTableau relation, and copies the variables after the `tbox` connector to that node.

Table 3. LoTREC rules regarding input handling.

Rule Name	Description
And	Add the two variables after connector `and` to the current node. If the node is marked CLOSED, then it will not perform this function.
Or	Creates two nodes for each variable after connector `or`. If the node is marked CLOSED, then it will not perform this function. This rule also checks if one of the variables of the or was already placed in the node. If this is the case, then it is not necessary to perform the Or rule.
TestClash	If a node contains A and not A ($A \sqcap \neg A$), then adds CLASH to the node and marks it CLOSED.
BottomRule	If a node contains the BOT ($\bot$) symbol, then marks the node CLOSED and adds STOPPED.

Table 4. LoTREC rules regarding simple operators.

3. REASONING PROBLEMS

Tableau algorithms use negation to reduce subsumption to (un)satisfiability of concept descriptions: $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable [?]. For example, to test if $(\exists R.A) \sqcap (\exists R.B)$ is subsumed by $\exists R.(A \sqcap B)$, check whether the concept description $C = (\exists R.A) \sqcap (\exists R.B) \sqcap \neg(\exists R.(A \sqcap B))$ is unsatisfiable. In *negation normal form* (NNF) this is $C_0 = (\exists R.A) \sqcap (\exists R.B) \sqcap \forall R.(\neg A \sqcup \neg B)$. In negation normal form the negation occurs only in front of concept names. Formulas had to be written in NNF before presented as input to LoTREC.

Three reasoning problems were given to test the in section **??** defined LoTREC strategies. These reasoning problems are described below.
The first problem was described as:
is $\exists r.D$ satisfiable w.r.t. $\mathcal{T} = \{\top \equiv \exists s.C, \top \equiv \forall r.(\bot \sqcup E)\}$?

Rule Name	Description
TestBlock	If a node x has an ancestor y such that x $\subseteq$ y, then y blocks x. Adds BLOCK to the node x and creates a link from node y to node y with label BLOCKS. It also marks node x with BLOCKED.
Only	Rule representing $\forall r.C$. If relation r exists, then adds C to the node linked by r. This rule propagates expressions. This rule checks is a node is marked CLOSED or BLOCKED. If this is the case, it will not perform this function.
Some	Rule representing $\exists r.C$. Creates a new node, links it with the parent node and adds label r to the link. Adds C to the new node. This is a structural rule, because it creates new nodes. This rule checks is a node is marked CLOSED or BLOCKED. If this is the case, it will not perform this function.
PropagateMarked	To propagate marked axioms to the next node in the tableau. In this case the TBox axioms, which are marked [TBox].

Table 5. LoTREC rules regarding complex operators.

LoTREC rule	Transformation Rule
And	$\Rightarrow_{\sqcap}$ IF $(a : C \sqcap D) \in S$ THEN $S' := S \cup \{a : C, a : D\}$
Or	$\Rightarrow_{\sqcup}$ IF $(a : C \sqcup D) \in S$ THEN $S' := S \cup \{a : C\}$ or $S' := S \cup \{a : D\}$
TestClash	$\Rightarrow_{\times}$ IF $\{a : A, a : \neg A\} \subseteq S$ or $(a : \bot) \in S$ THEN mark branch as CLOSED
Only	$\Rightarrow_{\forall}$ IF $(a : \forall r.C) \in S$ and $(a, b) : r \in S$ THEN $S' := S \cup \{b : C\}$
Some	$\Rightarrow_{\exists}$ IF $(a : \exists r.C) \in S$ THEN $S' := S \cup \{(a, i) : r, i : C\}$, where i is a 'fresh' variable in S
TestBlock	$\Rightarrow_{B}$ IF b is a successor of a in S and $\{C \mid b : C \in S\} \subseteq \{D \mid a : D \in S\}$ THEN mark b as BLOCKED by a in S

Table 6. Mapping LoTREC rules to tableau transformation rules.

This problem was reduced to the corresponding concept satisfiability problem as shown in table ??. The appendix shows the generated premodels.

Problem	$(\{\top \equiv \exists s.C, \top \equiv \forall r.(\bot \sqcup E)\}, \exists r.D)$
Reduction	this is reduced to checking ABox consistency w.r.t. TBox; i.e. check whether $\mathcal{A} = \{a : \exists r.D\}$ is inconsistent and unsatisfiable w.r.t. $\mathcal{T} = \{\exists s.C \sqcap \forall r.(\bot \sqcup E)\}$.
LoTREC input	input add tbox some S C tbox only R or BOT E some R D.
LoTREC display	`INPUT: TBox = ((T = S some C) & (T = R only (BOT or E))); Concept = R some D.`
Output	$\exists r.D$ satisfiable w.r.t. $\mathcal{T}$, because there was an open branch in the tableau. See figure A-1 and A-2 in the appendix for the premodels.
# premodels	2

Table 7. First reasoning problem.

The second problem was described as:
is $D \sqcap E$ subsumed by $\exists r.B$ in $\mathcal{T} = \{C \sqsubseteq \neg A, D \sqsubseteq \forall r.(A \sqcup B), E \sqsubseteq \exists r.C\}$?
This problem was reduced to the corresponding concept satisfiability problem as shown in table ??. The appendix shows the generated premodels.

Problem	$(\{C \sqsubseteq \neg A, D \sqsubseteq \forall r.(A \sqcup B), E \sqsubseteq \exists r.C\}, (D \sqcap E \sqsubseteq \exists r.B))$
Reduction	Is $\mathcal{A} = (D \sqcap E) \sqcap (\neg\exists r.B)$ unsatisfiable w.r.t. $\mathcal{T} = \{\neg C \sqcup \neg A, \neg D \sqcup \forall r.(A \sqcup B), \neg E \sqcup \exists r.C\}$? Is $\mathcal{A} = \{a : (D \sqcap E) \sqcap (\neg\exists r.B)\}$ inconsistent w.r.t. $\mathcal{T} = \{\neg C \sqcup \neg A, \neg D \sqcup \forall r.(A \sqcup B), \neg E \sqcup \exists r.C\}$?
LoTREC input	input add add tbox or not C not A tbox or not D only R or A B tbox or not E some R C and and D E only R not B.
LoTREC display	`INPUT: TBox = (((T = ¬C or ¬A ) & (T = ¬D or R only (A or B) )) & (T = ¬E or R some C )); Concept = (D and E) and R only (¬B)`
Output	a closed tableau, therefore the result would be unsatisfiable and inconsistent. $D \sqcap E$ is subsumed by $\exists r.B$ in $\mathcal{T}$. See figure A-3 and A-4 in the appendix for the premodels.
# premodels	13

Table 8. Second reasoning problem.

And the third reasoning problem was described as:
is the ABox $\{C(a)\}$ consistent w.r.t. $\mathcal{T} = \{\top \equiv \forall r.B \sqcap \forall s.C, \top \equiv \neg\forall r.(\neg C \sqcap B), \top \equiv \exists s.\top\}$?

This problem was reduced to the corresponding concept satisfiability problem as shown in table ??. The appendix shows the generated premodels.

Problem	$(\{\top \equiv \forall r.B \sqcap \forall s.C, \top \equiv \neg\forall r.(\neg C \sqcap B), \top \equiv \exists s.\top\}, C)$
Reduction	Is $\mathcal{A} = \{a : C\}$ inconsistent w.r.t. $\mathcal{T} = \{(\forall r.B \sqcap \forall s.C) \sqcap (\neg\forall r.(\neg C \sqcap B)) \sqcap (\exists s.\top)\}$?
LoTREC input	input add add tbox and only R B only S C tbox some R or C not B tbox some S TOP C
LoTREC display	`INPUT: Tbox = (((T = R only B and S only C) & (T = R some (C or ¬B)) & (T = S some TOP)); Concept = C`
Output	all branches contain CLOSED and BLOCKED. The Abox is consistent w.r.t. TBox. See figure A-5 and A-6 in the appendix for the premodels.
# premodels	15

Table 9. Third reasoning problem.

4. STRATEGIES

In LoTREC saturation is achieved by `repeat...end`, and priority by `firstRule...end`. In LoTREC the graphs are processed depth-first. It first processes premodel 1 and then move to the next until finished. The `CPLStrategy` was used by all the other strategies. Figure ?? shows this strategy which uses the rules `TestClash`, `And`, and `Or`. Section ?? will describe why the `Or` rule is applied once. The firstRule operator selects the first rule that is applicable and then starts all over again (`repeat...end`).

The `TestClash` rule was positioned at the beginning of the strategy to verify a clash as quickly as possible. The blocking and efficient ordering strategies are elaborated on in subsequent sections.

```
repeat
    firstRule
       TestClash
       And
       applyOnce Or
    end
end
```

Figure 1. The CPLStrategy.

4.1. Efficient Ordering

The ordering of rules in a LoTREC strategy can have an effect on the efficiency, for instance generating smaller tableau trees for a given problem.
In general, the ordering of rules can be in principle arbitrary, provided they are applied in a fair manner. This *fair strategy* repeats applying all rules in the strategy sequentially. For example: `repeat rule1; rule2; ... ; ruleN end.` To apply a rule means to apply the rule simultaneously to every possible formula of every node in the tableau.
Every time an `Or` rule was encountered, the pre-model was duplicated in LoTREC. Therefore, to demonstrate the efficiency of the order of rules in a strategy, the following problem was defined:
Is $C \sqcup D$ satisfiable w.r.t. $\mathcal{T} = \{\top \equiv \neg C \sqcup D, \top \equiv \neg C \sqcup \neg D\}$.
For LoTREC the input for this problem was `input add tbox or not C D tbox or not C not D or C D`. Two different strategies were used to demonstrate different efficiency. Figure **??** shows the Less Efficient Ordering strategy and figure **??** shows the ALC_Strategy which was used as the more efficient ordering during the experiment. Table **??** shows the results of applying these two strategies to the above formula. For both strategies the number of generated premodels and the number of steps needed to create these premodels are given.

```
LessEfficientOrdering =
ProcessInput
repeat
   firstRule
     repeat
        TestClash
        Or
        And
     end
     TestBottom
     Only
     PropagateMarked
     TestBlock
     Some
   end
end
```

Figure 2. The Less Efficient Ordering.

Where the definition of ProcessInput was:

```
InputRule
repeat
   AddRule
end
CopyT
```

```
ALC_Strategy =
ProcessInput
repeat
   firstRule
     repeat
        firstRule
            TestClash
            And
            applyOnce Or
        end
     end
     TestBottom
     Only
     PropagateMarked
     firstRule
        TestBlock
        Some
     end
   end
end
```

Figure 3. The More Efficient Ordering, ALC_Strategy.

For a description of the `InputRule`, `AddRule`, and `CopyT` rule see section **??**.
In the `ALC_Strategy` the `Or` rule was called with the `applyOnce` operator. The reason for this was to duplicate a premodel only if necessary. Without the `applyOnce`, LoTREC would just create many duplicates for each encountered `Or` rule. The `applyOnce Or` would keep the strategy more space efficient. Table **??** illustrates this.

Strategy	# premodels	# steps
Less Efficient Ordering	8 (1, 2.1, 3.1, 4, 2.2.1, 2.3, 3.2, 2.2.2)	10
ALC Strategy	5 (1, 2.1, 3.1, 2.2, 3.2)	8

Table 10. Tested strategies for efficiency.

The code snipped `firstRule TestBlock Some end` in the `ALC_Strategy` was positioned at the end. This ensured that when no other rule would apply, this part would be called. First the `TestBlock` rule was offered an opportunity to do its work, then `Some` rule. In this set up the Some rule would not create unnecessary nodes.
Interesting to note was that the `Or` rule without the check if one of the variables already exists in the node, would generate 7 premodels in 11 steps for the `ALC_Strategy`.
A second test was performed to test efficiency. The LoTREC input was: `input add add tbox or not C not A tbox or not D only R or A B tbox or not E some R C and and D E only R not B`. The

ALC_Strategy generated 13 premodels in 33 steps, while the LessEfficientStrategy generated 30 models in at least 80 steps. The exact number of steps could not be determined because LoTREC crashed everytime. However, in both tests the more efficient ALC_Strategy was more space and time efficient.

4.2. Blocking

To prevent infinite expansion of the tableau tree, which might be caused by straightforward application of the rules $\Rightarrow_\exists$ and $\Rightarrow_\equiv$, a blocking rule has to be applied. This blocking rule should detect cycles and prevent further application of the $\Rightarrow_\exists$ rule. A node b is said to be blocked by a node a if there exists a path of nodes from a to b, and the label of b is a subset of the label of a. The tableau transformation blocking rule was described in table **??**.

To ensure completeness the blocking rule can be applied only if no other rule applies, apart from $\Rightarrow_\exists$ which is exactly the rule to be blocked. A block represents a cyclical model. Baader, Horrocks, and Sattler [**?**] explain the meaning of "blocked" in the formulation of the expansions rules as follows: Without the $\sqsubseteq$-rule (i.e., in case the TBox is empty), the tableau algorithm for $\mathcal{ALC}$ would always terminate, even without blocking. Blocking prevents application of expansion rules when the construction becomes repetitive; i.e., when it is obvious that the sub-tree rooted in some node x will be similar to the sub-tree rooted in some predecessor y of x. To be more precise, we say that a node y is an *ancestor* of a node x if they both belong to the same completion tree and either y is a predecessor of x, or there exists a predecessor z of x such that y is an ancestor of z. A node x is *blocked* if there is an ancestor y of x such that $\mathcal{L}(x) \subseteq \mathcal{L}(y)$ (in this case we say that y blocks x), or if there is an ancestor z of x such that z is blocked; if a node x is blocked and none of its ancestors is blocked, then we say that x is *directly blocked*. When the algorithm stops with a clash free completion forest, a branch that contains a directly blocked node x represents an infinite branch in the corresponding model having a regular structure that corresponds to an infinite repetition (or unraveling) of the section of the graph between x and the node that blocks it.
The strategy WrongBlocking gives an example of blocking which damages completeness. Figure **??** shows two examples of the wrong blocking strategy.

The idea reflected in this figure is that finding an opportunity to stop as soon as possible would create a smaller tableau. Rules check if a node was already closed or blocked to prevent looping. However, in this case the TestBlock is called too early in the strategy. In the second strategy example, TestBlock was called at the end. The fairness approach of LoTREC should give each rule an opportunity to do its work. The TestBlock would be applied to the tableau if all other rules had a chance to perform. The resulting output of both strategies were the same. This was demonstrated during the experiment with the following concept satisfiability problem instance: $(\{\top \equiv (\forall r.\exists r.\bot) \sqcap C, \top \equiv \exists r.C\}, C)$.

```
WrongBlocking1:
ProcessInput
repeat
   CPLStrategy
   TestBottom
   TestBlock
   Only
   Some
   PropagateMarked
end

WrongBlocking2:
ProcessInput
repeat
   CPLStrategy
   TestBottom
   Only
   Some
   PropagateMarked
   TestBlock
end
```

Figure 4. Two Wrong Blocking examples.

Completeness means that the algorithm can prove every true condition. If the input is inconsistent, the algorithm closes. If the algorithm does not close, then the input is consistent. Premature blocking will violate this rule and can close an open branch prematurely, or as was the case in in figure **??**, not close at all. The encircled branch of the tableau was left open due to wrong blocking and the same branch closed when correct blocking was applied.
The ALC_Strategy discussed in section **??** and shown in figure **??** uses subset blocking and ensured completeness. To guarantee that nodes are saturated before blocking, use TestBlock rule when no other rules, except for Some rule, could be used. The appendix gives an other example of wrong and correct blocking.

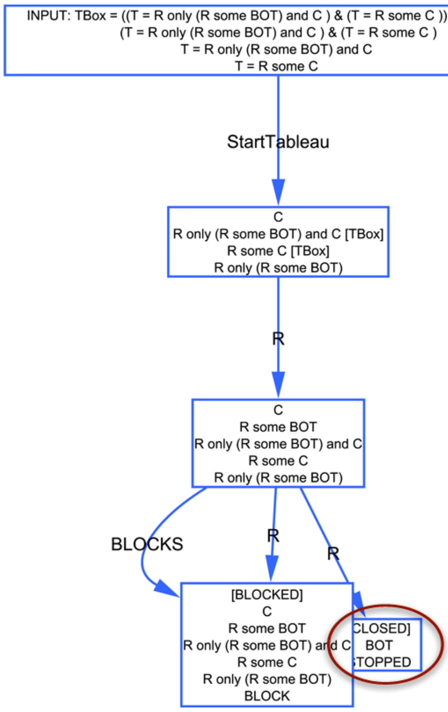


Figure 5. Premodel with correct blocking and completeness.

5. EXTENSIONS

This section shortly describes two extensions to $\mathcal{ALC}$: transitive roles $\mathcal{ALCS}$ and role hierarchy $\mathcal{ALCH}$.

5.1. Transitive Roles

The implementation described in this paper used the tableau rule for transitive roles which was defined as:
$\Rightarrow \forall_+$ IF $(a : \forall r.C) \in S$ and $(a, b) : r \in S$ THEN $S' := S \cup \{b : C\}$.

In LoTREC an extra connector `onlyT` was defined for this purpose. This connector would indicate that the transitive version of the `only` connector was used. Rule `onlyPlus` was added to the strategy to handle `onlyT`. This rule was a copy of the `only` rule with one addition: adding a copy of itself to the next node in the premodel. The ALCS 2 Strategy was tested on reasoning problem 3 described in section **??** of where the premodels can be found in figure A-9 of the appendix.

In LoTREC there is a predefined logic called *K4-implicit-edges*, which contains a similar `onlyT` rule. The axioms of a TBox were copied to each node in the premodel as part of the strategy. Transitivity as defined here would not make a difference when it was part of an TBox axiom, because the axiom would be copied anyway. Another predefined logic in LoTREC is *K4-explicit-edges*, which considers all the roles te be transitive, i.e. if node u was linked via R to node v, and node v was linked to node w via R, then a link R was created between node u and w. In this case, the `onlyT` rule was not applicable.
Figure **??** shows the strategies for both the implicit and explicit transitive additions.

```
ALCS 1 (implicit):
ProcessInput
repeat
    CPLStrategy
    TestBottom
    TestBlock
    Only
    OnlyPlus
    Some
    PropagateMarked
end

ALCS 2 (explicit):
ProcessInput
repeat
    CPLStrategy
    TestBottom
    Only
    Some
    PropagateMarked
    TransitiveEdges
    TestBlock
end
```

Figure 6. Two Transitive Strategies ($\mathcal{ALCS}$).

5.2. Role Hierarchies

TBoxes migt also include simple role inclusions of the form $r \sqsubseteq s$ which state that the role r is a "sub-role" of role s. This was implemented in LoTREC using the `subrole` connector, and `RoleHierarchy` rule. This is a simple implementation. More complex implementations are described in [**?**]. Figure **??** shows the ALCH strategy code and figure A-10 in the appendix shows an example of a generated premodel.

6. CONCLUSION

To summarize the LoTREC experience: an interesting tool for tableau experiments. Unfortunately, the lack of documentation, the memory leaks, and bugs can make it somewhat less friendly to use. However, the step function is very useful in understanding how the strategies work and how to make these strategies

```
ALCH:
ProcessInput
CopySubRole
repeat
   CPLStrategy
   TestBottom
   TestBlock
   Only
   OnlyPlus
   Some
   PropagateMarked
   RoleHierarchy
   firstRule
        TestBlock
        Some
   end
end
```

Figure 7. The $\mathcal{ALCH}$ Strategy.

more efficient. The build-in commands make it possible to write strategies for a wide variety of logics. Description Logics in a vast and interesting subject. In this paper only the surface was scratched. But nevertheless a very interesting surface. More work should be done to explore the transitive roles and role hierarchies. Another future subject could be inverse roles, which need a different blocking mechanism called dynamic blocking.

Appendix

LoTREC code

Connectors

Name	input
Display	INPUT: TBox = _; Concept = _

Name	tbox
Display	T = _

Name	add
Display	_ & _

Name	not
Display	¬_

Name	and
Display	_ and _

Name	or
Display	_ or _

Name	some
Display	_ some _

Name	only
Display	_ only _

Name	onlyT
Display	_ onlyT _

Name	subrole
Display	_ subrole _

The rules are in sequence of which they were used in the strategies.

Name	InputRule
Conditions	hasElement thisnode input variable t variable c
Actions	add thisnode variable t createNewNode nextnode link thisnode nextnode StartTableau add nextnode variable c

Name	AddRule
Conditions	hasElement thisnode add variable x variable y
Actions	add thisnode variable x add thisnode variable y

Name	CopyT
Conditions	hasElement thisnode tbox variable x isLinked thisnode nextnode StartTableau
Actions	add nextnode variable x markExpressions nextnode variable x TBox

Name	TestClash
Conditions	hasElement node variable x hasElement node not variable x
Actions	add node CLASH mark node CLOSED

Name	And
Conditions	hasElement node and variable x variable y isNotMarked node BLOCKED isNotMarked node CLOSED
Actions	add node variable x add node variable y

Name	Or
Conditions	hasElement node or variable x variable y isNotMarked node BLOCKED isNotMarked node CLOSED hasNotElement variable x hasNotElement variable y
Actions	duplicate premodel_copy add node variable x add premodel_copy.node variable y

Name	TestBottom
Conditions	hasElement node BOT
Actions	add node STOPPED mark node CLOSED

Name	Only
Conditions	hasElement thisnode only variable r variable c isLinked thisnode nextnode variable r isNotMarked nextnode BLOCKED isNotMarked nextnode CLOSED
Actions	add nextnode variable c

Name	PropagateMarked
Conditions	isNewNode node’ isAncestor node node’ hasElement node variable x isMarkedExpression node variable x TBox isNotMarked node’ BLOCKED isNotMarked node’ CLOSED
Actions	add node’ variable x

Name	TestBlock
Conditions	isNewNode childnode isAncestor parentnode childnode contains parentnode childnode
Actions	mark childnode BLOCKED link parentnode childnode BLOCKS add childnode BLOCK

Name	Some
Conditions	hasElement thisnode some variable r variable y isNotMarked thisnode BLOCKED isNotMarked thisnode CLOSED
Actions	createNewNode nextnode link thisnode nextnode variable r add nextnode variable y

Name	OnlyPlus
Conditions	hasElement thisnode onlyT variable r variable a isLinked thisnode nextnode variable r isNotMarked nextnode BLOCKED isNotMarked nextnode CLOSED
Actions	add nextnode variable a add nextnode onlyT variable r variable a

Rules

Name	TransitiveEdges
Conditions	isLinked node_u node_v isLinked node_v node_w
Actions	link node_u node_w variable r

Name	RoleHierarchy
Conditions	is NewNode node’ isAncestor node node’ hasElement node subrole variable r variable s isMarkedExpression node subrole variable r variable s Subrole isLinked node node’ variable r
Actions	link node node’ variable s

Name	CopySubRole
Conditions	hasElement node subrole variable r variable s isLinked node node’ StartTableau
Actions	add node’ subrole variable r variable s markExpressions node’ subrole variable r variable s Subrole

Strategies

The following strategies were defined in LoTREC for this project.

Name	ProcessInput
Code	InputRule repeat AddRule end CopyT

Name	CPLStrategy

Code	repeat
	    firstRule
	        TestClash
	        And
	        applyOnce Or
	    end
	end

Name	SomeBlock

Code	firstRule
	    TestBlock
	    Some
	end

Name	LessEfficientStrategy

Code	ProcessInput
	repeat
	    SimplyHopeless
	    TestBottom
	    Only
	    Some
	    PropagateMarked
	    TestBlock
	end

Name	WrongBlocking1

Code	ProcessInput
	repeat
	    CPLStrategy
	    TestBottom
	    TestBlock
	    Only
	    Some
	    PropagateMarked
	end

Name	WrongBlocking2

Code	ProcessInput
	repeat
	    CPLStrategy
	    TestBottom
	    Only
	    Some
	    PropagateMarked
	    TestBlock
	end

Name	ALC_Strategy (EfficientOrdering)

Code	ProcessInput
	repeat
	    firstRule
	        CPLStrategy
	        TestBottom
	        Only
	        PropagateMarked
	        SomeBlock
	    end
	end

Name	ALCS_Strategy1

Code	ProcessInput
	repeat
	    firstRule
	        CPLStrategy
	        TestBottom
	        Only
	        OnlyPlus
	        PropagateMarked
	        firstRule
	            TestBlock
	            Some
	        end
	    end
	end

Name	ALCS_Strategy2

Code	ProcessInput
	repeat
	    firstRule
	        CPLStrategy
	        TestBottom
	        Only
	        PropagateMarked
	        TransitiveEdges
	        firstRule
	            TestBlock
	            Some
	        end
	    end
	end

Name	ALCH_Strategy

Code	ProcessInput
	CopySubRole
	repeat
	    firstRule
	        CPLStrategy
	        TestBottom
	        Only
	        PropagateMarked
	        RoleHierarchy
	        SomeBlock
	    end
	end

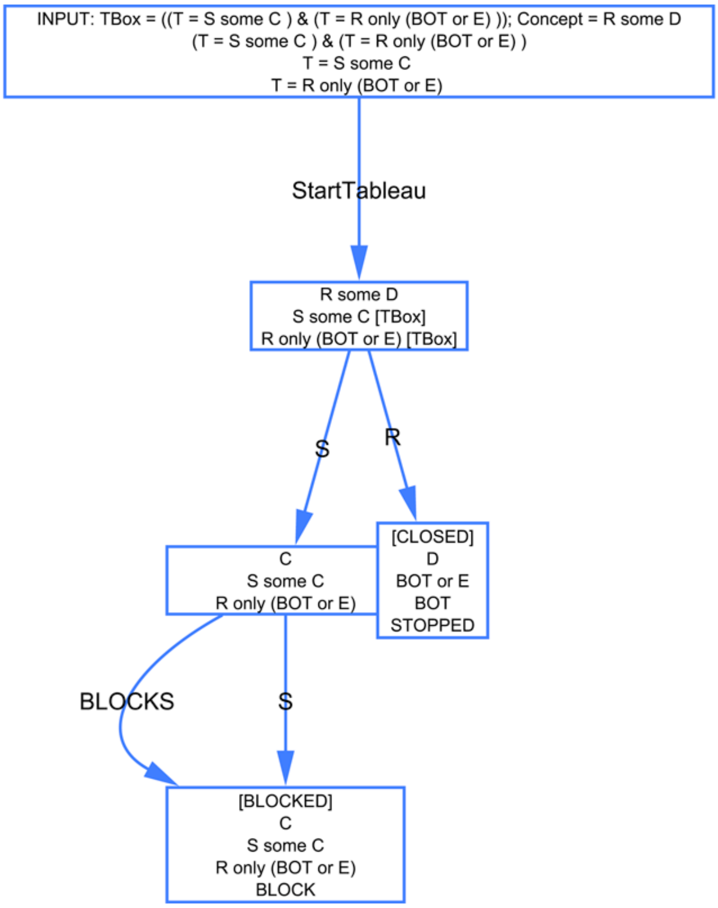


Figure A-1.

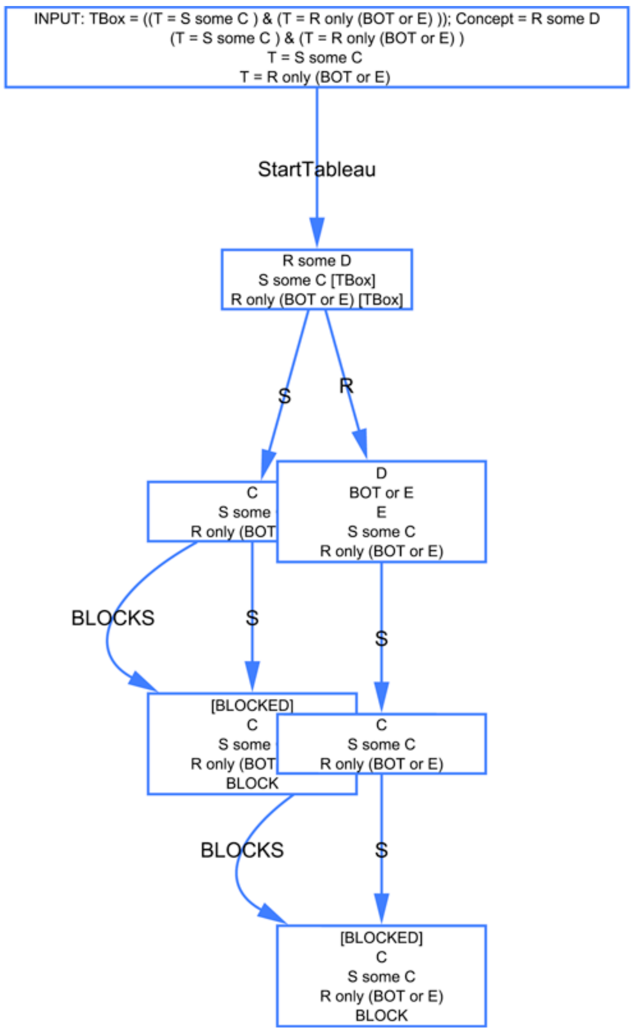


Figure A-2.

For the individual premodels, please see the .pdf files with prefix rp-2.

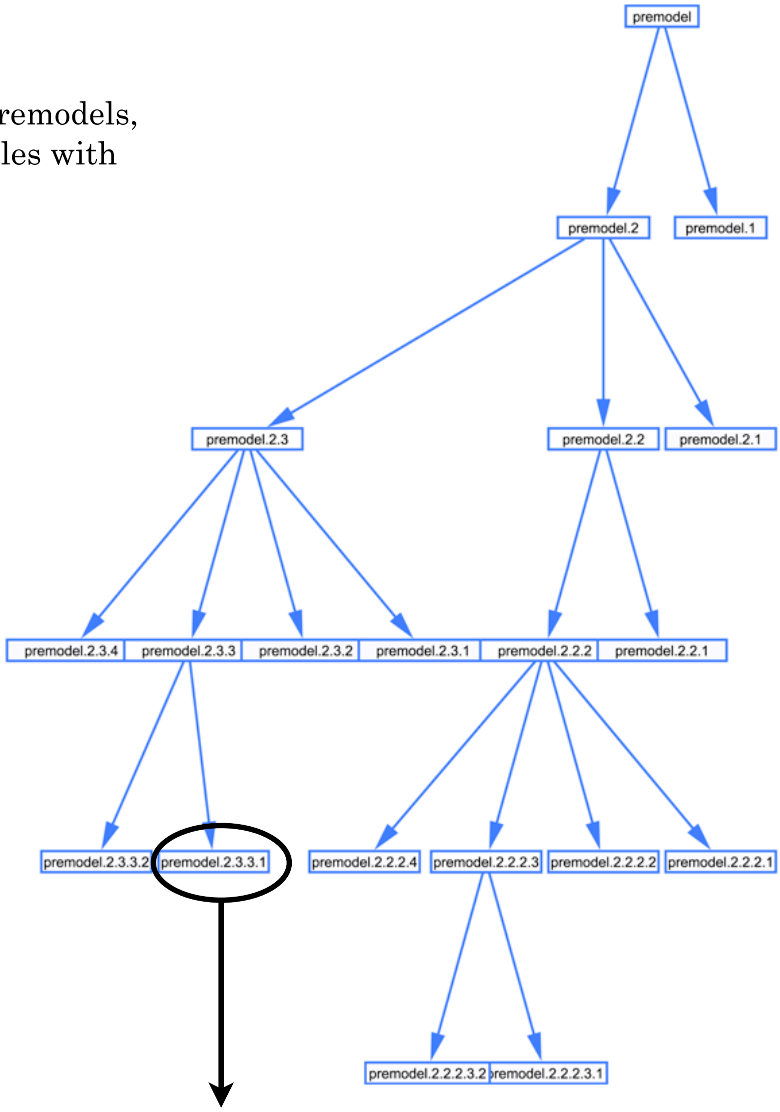


Figure A-3.

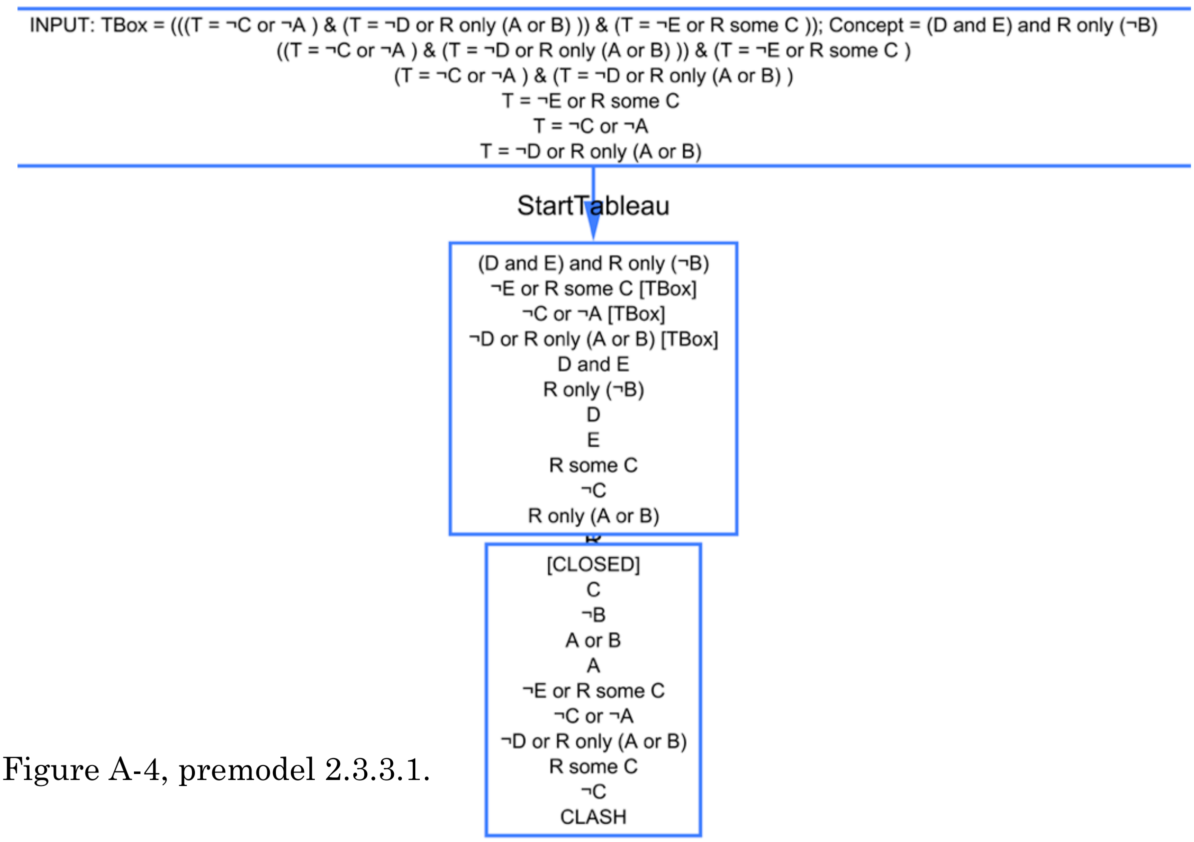


Figure A-4, premodel 2.3.3.1.

Premodels of the third Reasoning Problem

For the individual premodels, please see the .pdf files with prefix rp-3.

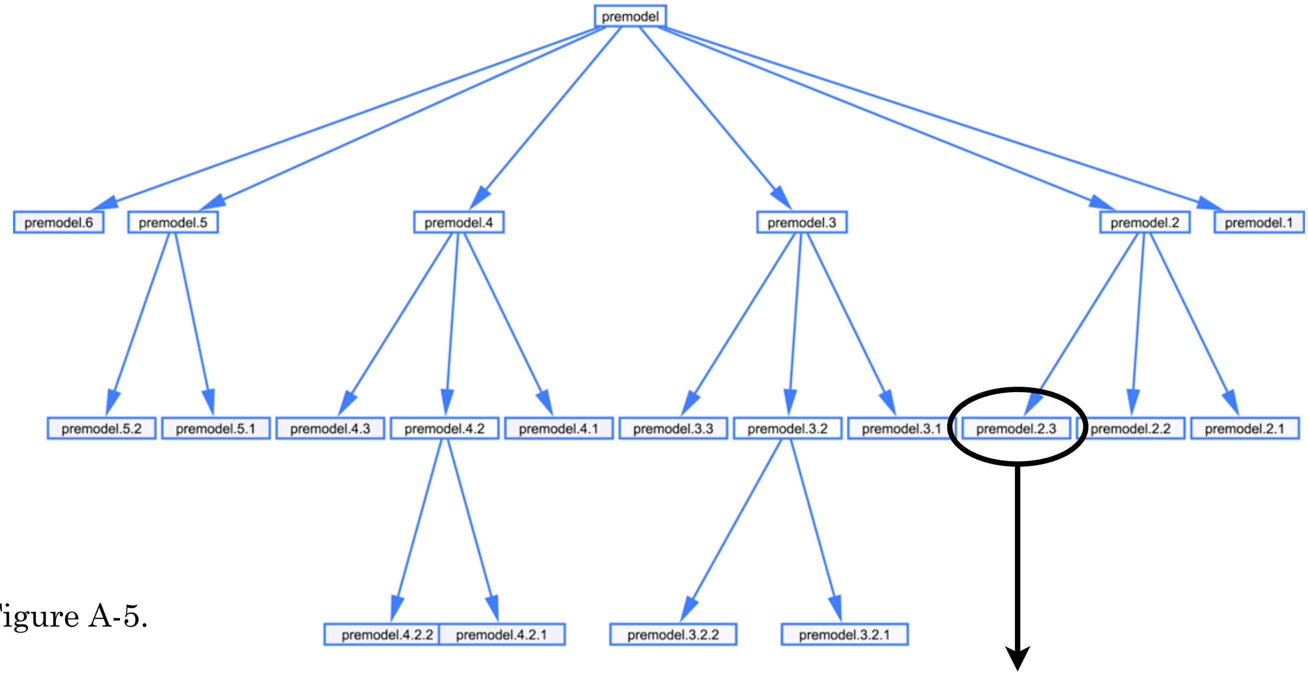


Figure A-5.

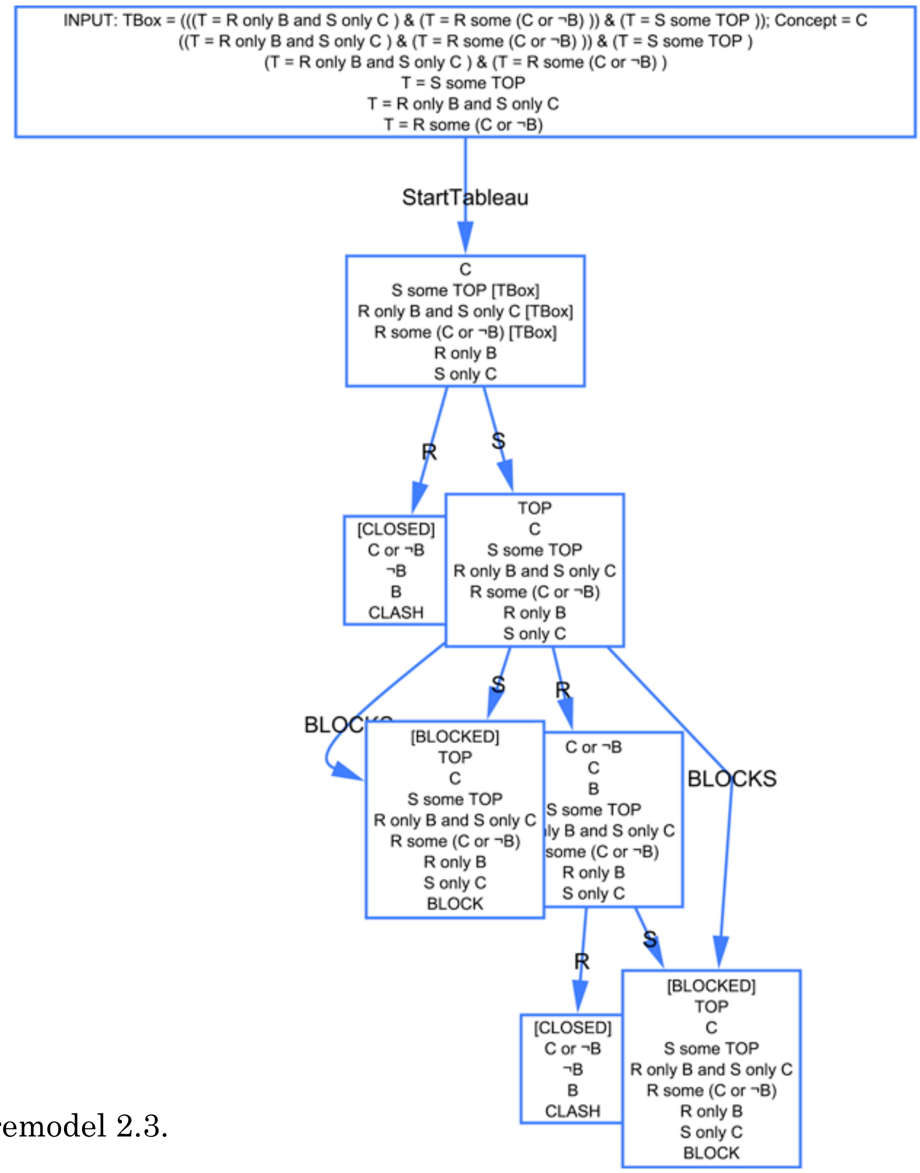


Figure A-6, premodel 2.3.

Another Blocking Example

Figure A-7, Wrong Blocking

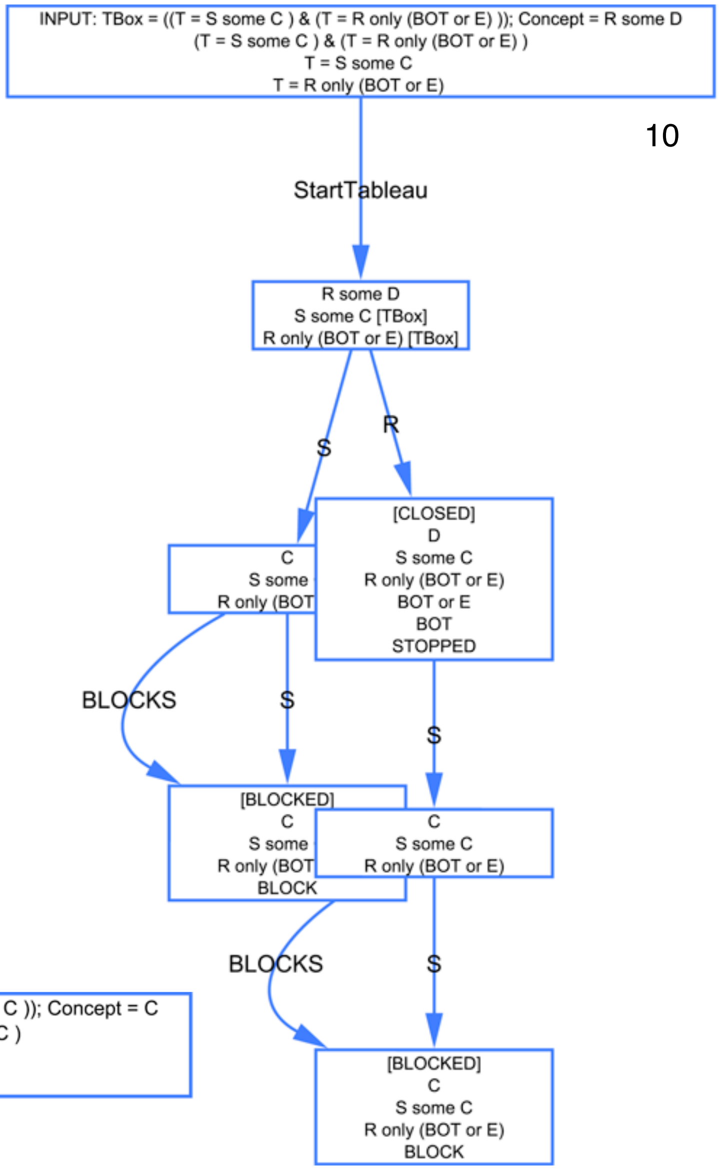
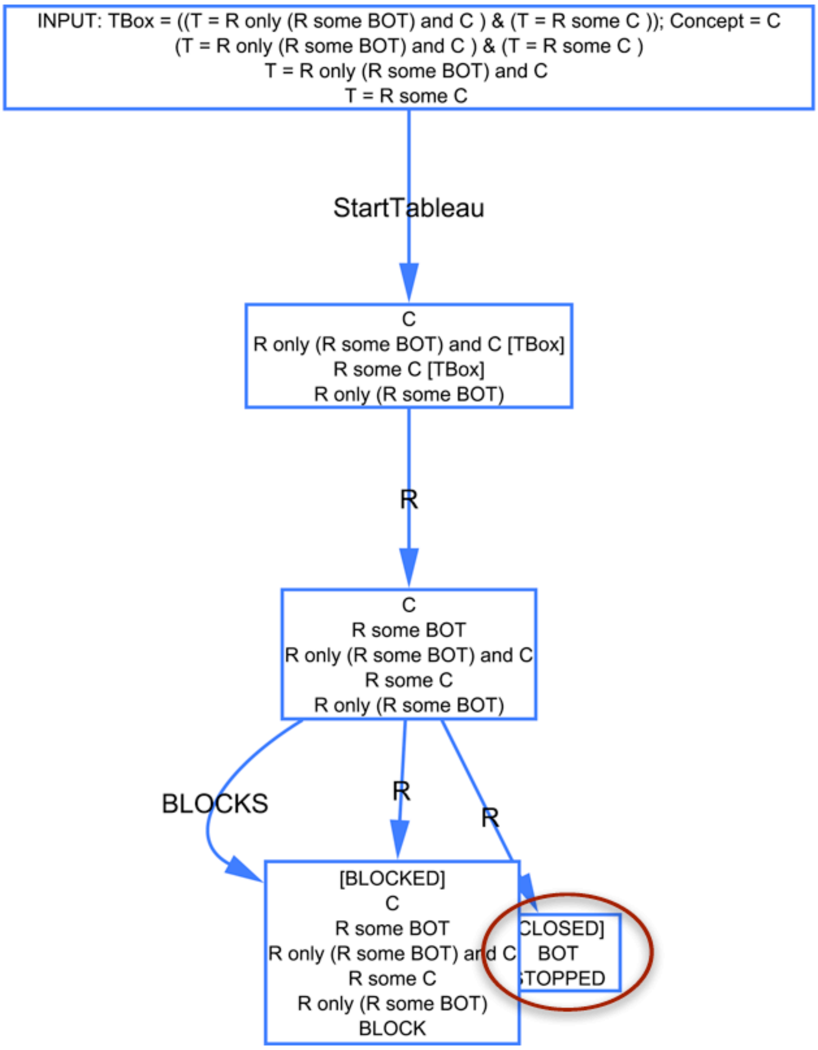
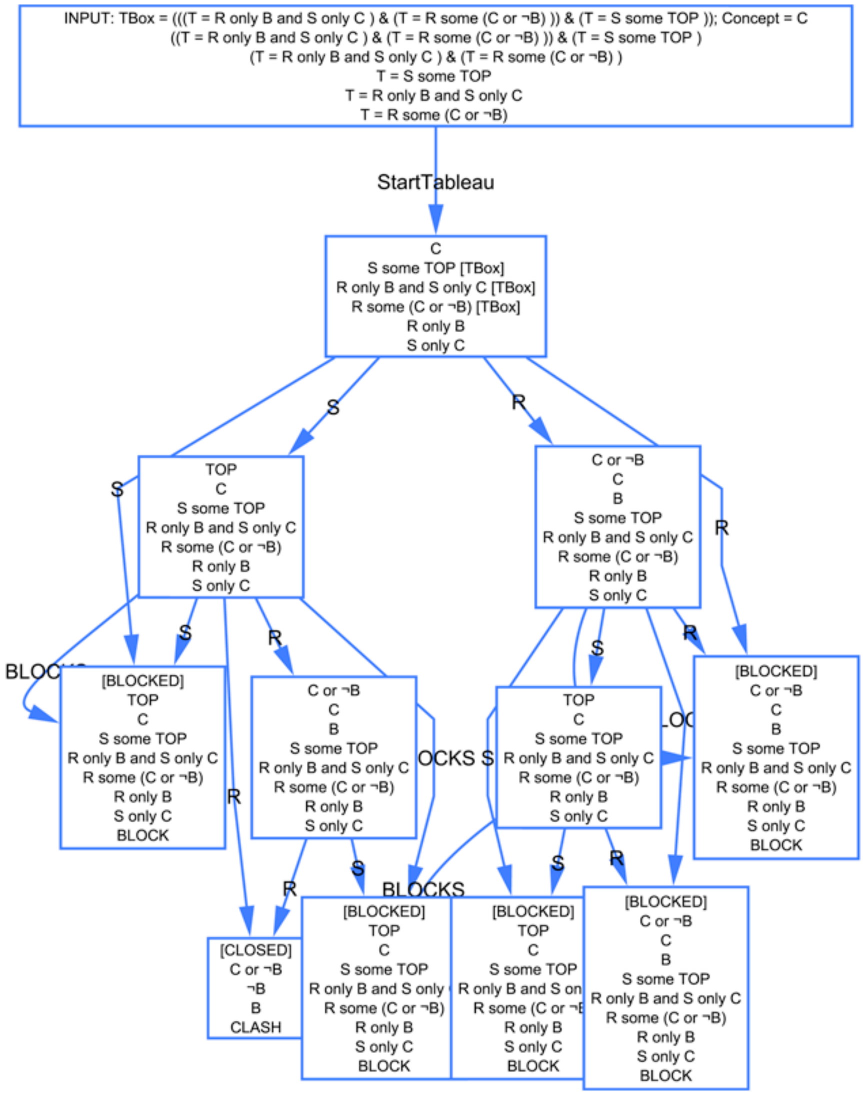


Figure A-8, Correct Blocking



Transitive Roles

Figure A-9,
Premodel 5.1



Role Hierarchie

Figure A-10.