Tableaux for Type PDL

Agathoklis Kritsimallis
agiskr@gmail.com

Chrysafis Hartonas
Department of Computer Science and Telecommunications
TEI of Larissa, Greece
hortonas@teilar.gr

ABSTRACT
We give a deterministic exponential, sound and complete tableau-based satisfiability algorithm for the system of Type PDL (τPDL) [9, 8], by extending the algorithm of Goré and Widmann [6] that has been given for the case of the CPDL.

The system of τPDL has been introduced to reason about types of actions and while the type semantics have been adopted in [9], here, we follow the standard relational semantics as they have been presented in [8]. We introduce an appropriate tableau calculus for the satisfiability algorithm and in relation with the case of the CPDL, the algorithm that we present for τPDL handles the backwards possibility operator instead of the converse operator, as well as capabilities statements and abstract processes which are defined as pairs of preconditions and effects, written as \( \varphi \Rightarrow \psi \).

Categories and Subject Descriptors
F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic—Mechanical Theorem Proving, Computational Logic, Proof Theory, Modal Logic; I.2.3 [Artificial Intelligence]: Deduction and Theorem Proving—Deduction

General Terms
Algorithms

Keywords
Type PDL, tableaux, decision procedure, automated reasoning.

1. INTRODUCTION

The system of Type PDL (τPDL) has been introduced in [9], as an extension of Propositional Dynamic Logic (PDL) [2], with the main motivation to reason about types of actions and with potential applications in web service composition, as well as in agent technology. It has been inspired by the semantic web services, where concrete actions of services are hidden from public view and what is publicly known is only the type of actions a service can perform, modelled by their preconditions and effects. Syntactically, the language of τPDL extends the language of PDL precisely by the capabilities statements, the preconditions-effects constructs and the backwards possibility operator (a weak form of converse). The system has two very natural interpretations, one based on the familiar relational semantics and the other based on type semantics, where action terms are interpreted as types of actions (sets of binary relations). In [8], it is shown that there is equivalence between standard relational and type semantics. Additionally, it is shown that the satisfiability problem of τPDL is decidable and a NExpTime complexity upper bound has been established. To the best of our knowledge, the system of τPDL is the first decidable system on a logic for reasoning about agent action capabilities (missing in the KARO Framework [16], [17], [18]).

What has been left open is the investigation of a deterministic exponential decision procedure. Now, we turn our attention on the system of PDL and we shortly review the tableau-based decision procedures that have been given for its satisfiability problem.

The system of Propositional Dynamic Logic was introduced by Fischer and Ladner in [2] (see also [7]) as an abstraction of the logical system of [13], in the sense that atomic programs do not have internal structure (tests, assignments) and only propositional variables are permitted. The satisfiability problem of PDL is decidable, as it enjoys the finite model property. Additionally, in [2] a lower bound is established by showing that the satisfiability problem of PDL is ExpTime-Complete.

One of the first algorithms for PDL is given in [14]. This algorithm is, in the best case, exponential and as is commented by the author himself (in [15]), it is the analogue of the truth tables for Classical Propositional Logic (CPL). The specific algorithm is essentially the same with the one that is described in [7].

The first tableau-based algorithm for PDL was given by Pratt in [15]. The main disadvantage of Pratt’s algorithm is that the construction of the tableau and the examination of the graph for inconsistent nodes and unfulfilled eventualities are carried out in two distinct steps.

In [3], the idea of an “on-the-fly” algorithm seems to appear for the first time, meaning that the distinct stages that exist in Pratt’s algorithm should be merged and all the required checks should take place during the construction of the tableau. Additionally, the tableau system that is in-
introduced is a labelled one where the accessibility relation is explicitly denoted (see also [4]). In the given calculus, there are rules which handle the converse operator and the use of a labelled system simplifies its manipulation. The combination of the iteration operator with the converse one leads to the necessity to use an analytic cut rule as one of the tableau rules. Finally, an informal description of an exponential algorithm is given.

The presented algorithm in [12] is the natural evolution of the one that Pratt proposed. It uses classical unlabelled rules and for the case of the converse operator, as in the previous approach, an analytic cut rule is used. Although this analytic cut rule is more restricted in relation with the one of the previous approach, in the sense that the choice of the formula in which the rule will be applied is more restricted, there are doubts about how possible it is for the specific algorithm to lead to an efficient theorem prover.

In [1], there are the usual tableau rules along with histories and variables. Histories are used to deploy a form of partial branch caching, while variables are used to detect unfulfilled eventualities. Additionally, histories are used to detect “at a world” cycles caused by nested stars. The result of a double exponential algorithm comes naturally.

In [5] and in its extension for the converse operator [6], the way that the global caching plays a crucial role to the achievement of an optimal algorithm becomes obvious (see also [10]). Besides the prevention of the creation of infinite branches due to the iteration operator, global caching doesn’t permit the deployment of nodes for which the satisfiability status has already been determined. Both algorithms are exponential, despite the fact that in the second one, caching is restricted to state nodes and also, there is the need of “restarting” a node (special nodes) in specific cases, due to the converse operator.

Here, we take advantage of the satisfiability algorithm of Goré and Widmann [6] in order to give an exponential decision procedure for the satisfiability problem of \( \tau \text{PDL} \). We adopt the standard relational semantics as they have been presented in [8]. The given satisfiability algorithm, in relation with the CPDL, handles the backwards possibility operator instead of the converse operator, as well as capabilities statements and abstract processes which are defined as pairs of preconditions and effects, written as \( \varphi \Rightarrow \psi \).

In the following of this paper, in Section 2 we present the syntax of \( \tau \text{PDL} \) as well as its interpretation in the standard relational semantics, in Section 3 we show how we determine when a process is of type of another process, in Section 4 we give the tableau calculus \( \tau \text{PDL} \), in Section 5 we describe the satisfiability algorithm, in Section 6 we give the complexity results and we outline the proofs of soundness and completeness, and finally, we conclude this paper in Section 7. Due to the lack of space, complete proofs are presented only in the long version [11] of this paper.

2. TYPE PDL

To define the language of \( \tau \text{PDL} \), assume countable, nonempty and disjoint sets of atomic sentences \( At \) and atomic program terms \( AtP \), and let \( I \) be a set of agent names, disjoint from both \( At \) and \( AtP \). The language \( L \) is two-sorted, where the sub-language \( L_s \) is a language of properties of states of a system, whereas the sub-language \( L_a \) is a language of actions (programs, or processes transforming the states of the system to other states).

\[
L_s \ni \varphi := p (p \in At) \mid \neg \varphi \mid \forall A.\varphi \mid \exists A.\varphi \mid L_s (\tau \in I)
\]

\[
L_a \ni \varphi := \pi (\pi \in AtP) \mid \varphi \mid \varphi \Rightarrow \psi \mid AA \mid A + A \mid A^+
\]

**Definition 1.** We define the sets \( A_{\Rightarrow} = \{ \varphi \Rightarrow \psi \in L_a \mid \varphi, \psi \in L_s \} \)

\[
\Sigma = L_s \cup AtP \cup A_{\Rightarrow} \ni \Sigma = \Sigma \setminus L_s
\]

We denote as \( \Box \) a modality of the form \( \forall A_1, \ldots, A_k \) and as \( \Diamond \) a modality of the form \( \exists A_1, \ldots, A_k \), with \( k > 0 \) and for \( i = 1, \ldots, k \), we have that \( A_i \in L_a \). For example, when we write \( \neg \Box \varphi \) and \( \neg \Diamond \varphi \), we denote formulas of the form \( \neg \forall A_1, \ldots, A_k \varphi \) and \( \neg (\varphi. \exists A_k, \ldots, \exists A_1) \) respectively.

**Definition 2.** The set \( Ev \) of eventualities is defined as the set of formulas of the form:

- \( \neg A_1 \Rightarrow \neg A_2 \Rightarrow \cdots \Rightarrow \neg A_n \Rightarrow \xi \in L_s \) with \( n = 2k + 1 \) and \( k \geq 0 \) and \( (\xi = \forall A^+ \chi \lor \xi = \neg (\chi. A^+) \).
- \( \neg A_1 \Rightarrow \neg A_2 \Rightarrow \cdots \Rightarrow \neg A_n \Rightarrow \xi \in L_s \) with \( n = 2k + 1 \) and \( k \geq 0 \) and \( (\xi = \forall A^+ \chi \lor \xi = \neg (\chi. A^+) \).
- \( \forall A_1 \Rightarrow \neg A_2 \Rightarrow \cdots \Rightarrow \neg A_n \Rightarrow \xi \in L_s \) with \( n = 2k + 1 \) and \( k \geq 0 \) and \( (\xi = \forall A^+ \chi \lor \xi = \neg (\chi. A^+) \).

In this paper, we adopt the relational semantics, in the way that they have been introduced in [8]. For the type semantics, in which \( L_a \) is considered a language of types of actions (sets of binary relations), we refer the reader to [9]. In relational semantics, \( L_a \) is treated as a language of actions. In particular, the precondition-effect operator, as in \( \varphi \Rightarrow \psi \), designates abstract actions, specified by the preconditions \( \varphi \) and effects \( \psi \), in such way that each time that \( \varphi \) holds, after their execution, \( \psi \) holds. Sentential tests \( \varphi \) (dropping the PDL operator ?) and complex actions which are built by using the regular operators of composition, choice and Kleene star (positive iteration \( A^+ \)), are treated in the same way as in PDL. In the language \( L_a \), formulas \( \forall A.\varphi \) can be understood as the property which states that after any execution of action \( A \), \( \varphi \) holds. The construct \( \varphi. A \) can be interpreted as holding at a state, if this state can be reached from a state where \( \varphi \) holds, by an action \( A \). Finally, \( C \) is a capabilities statement, holding at a state \( s \) provided that the agent \( i \) has the capability to perform \( A \) at that state.

**Definition 3.** A frame \( F \) is a structure \( F = \langle S, P, \Rightarrow, I \rangle \), also denoted as \( F = \langle S, \Rightarrow, _{-A}, \Rightarrow, I \rangle \), where

A) \((S, P, \Rightarrow)\) is a labeled transition system, with \( S \) the underlying set of the frame, a nonempty set of states of the system. \( P \) is a set of labels and the map \( \Rightarrow : P \to 2^S \times S \) assigns a labeled binary relation \( \Rightarrow \) to each \( \pi \in P \). The map \( \Rightarrow \) extends to all of \( P^+ \) (finite, non-empty sequences of items in \( P \)), by composition (thus we write, for example, \( _{-A} \Rightarrow _{-B} \) for the composition \( _{-A} \Rightarrow _{-B} \)).

B) \( I \) is a set of agent names
Table 1: Interpretation in Standard Relational Semantics

<table>
<thead>
<tr>
<th>Formula</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[p]_e^F$</td>
<td>$e(p) \subseteq S$</td>
</tr>
<tr>
<td>$[-\varphi]_e^F$</td>
<td>$S \setminus [\varphi]_e^F$</td>
</tr>
<tr>
<td>$[\forall A. \varphi]_e^F$</td>
<td>${ s \in S</td>
</tr>
<tr>
<td>$[\varphi \lor \exists A]_e^F$</td>
<td>${ s \in S</td>
</tr>
<tr>
<td>$[C, \pi]_e^F$</td>
<td>${ s \in S</td>
</tr>
<tr>
<td>$[C, \varphi]_e^F$</td>
<td>$S$</td>
</tr>
<tr>
<td>$[C, (\varphi \Rightarrow \psi)]_e^F$</td>
<td>${ s \in S</td>
</tr>
<tr>
<td>$[C, (A+B)]_e^F$</td>
<td>$[C, A]_e^F \cap [C, B]_e^F$</td>
</tr>
<tr>
<td>$[C, A^+]_e^F$</td>
<td>$\bigcup {[\varphi]_e^F</td>
</tr>
</tbody>
</table>

The frame is an $L$-frame if the set $P$ of process labels is the set $AtP$ of atomic process terms.

**Definition 4.** An $L$-model is a triple $N = (F, e, (\pi^N)_{i \in I})$:

1. $F$ is an $L$-frame
2. $e : At \rightarrow 2^S$ is an interpretation function of atomic properties of states
3. For each $i \in I$, $\pi^N$ is a map assigning to the agent $i$ capabilities that it has at each state $s \in S$, more specifically $\pi^N(s) \subseteq \bigcup \{ \overset{\varphi}{\rightarrow} \mid \varphi \in \Sigma^+ \}$.

The interpretation function is extended to all formulae (sentences and action types) of the language, as shown in Table 1, where for $\varphi \in L_s$, its interpretation $[\varphi]_e^F \subseteq S$ is the set of states where $\varphi$ holds and for $A \in L_s$, the set $\overset{\varphi}{\rightarrow} \subseteq S \times S$ is a process, i.e. a binary relation on $S$.

The satisfaction relation is defined as usual: For any state $s \in S$, $s \models [\varphi]_e^F$ iff $s \in [\varphi]_e^F$.

**Remark 1.** We will be interested in normal models, where the interpretation and the capabilities assignment function are required to satisfy the normality condition (2), for each $i$.

$$\pi^N(s) = \bigcup \{ \overset{\varphi \Rightarrow \psi}{\rightarrow} \mid s \in [C, (\varphi \Rightarrow \psi)]_e^F \} \cup \bigcup \{ \overset{\varphi}{\rightarrow} \mid s \in [C, \pi]_e^F \} \quad (2)$$

**Lemma 1.** The following semantical equivalences hold:

$$\begin{align*}
C_i (A \lor B) &\equiv C_i A \lor \forall A C_i B \\
C_i (A + B) &\equiv C_i A \land C_i B \\
C_i A^+ &\equiv C_i A \land \forall A^+ C_i A
\end{align*}$$

**Proof.** We refer the reader to [9, 8].

Other operators (e.g. implication, forward existential and backwards universal quantifiers) can be defined in the usual way, as in modal logics on a classical propositional basis.

### 3. Typing Judgement

The question that we answer in this section, always in the context of an $L$-model, is when a process $A \in \Sigma^+$ is considered to be of type of another process $A' \in AtP$. One of the problematic situations of the satisfiability problem of $\tau$PDL is the case of a state which satisfies pairs of formulas $\forall A. \chi$ and $\forall (\varphi \Rightarrow \psi). \vartheta$, with $A \in \Sigma^+$ and the issue that arises is whether process $A$ is of type $\varphi \Rightarrow \psi$ or not. It is clear that formula $\forall A. \chi$ imposes the existence of a transition of process $A$. Consequently, it is critical to know if the specific process is of type $\varphi \Rightarrow \psi$, i.e. $\overset{\varphi \Rightarrow \psi}{\rightarrow} \subseteq \overset{\chi}{\rightarrow}$, as it is always possible that $\vartheta = \chi$. In the following of this section, we establish some interesting properties that concern the accessibility relation of a process of the form $\varphi \Rightarrow \psi$ and we show how we resolve the issue of typing judgement. The essential ideas of this section are summarized as follows:

- We show that for any $L$-model, the relation $\overset{\chi}{\rightarrow}$ consists of transitions $s \overset{\chi}{\rightarrow} s'$ of any $A \in \Sigma^+$, such that state $s$ satisfies formula $\neg \varphi$ or state $s'$ satisfies formula $\psi$.
- The way that we redefine relation $\overset{\varphi \Rightarrow \psi}{\rightarrow}$ allows us to conclude that if formula $\neg \varphi$ holds in a state $s$, then for any transition $(s, s') \in S^2$ that leaves the specific state, we have that $s \overset{\varphi \Rightarrow \psi}{\rightarrow} s'$. Of course, for any transition $(s, s') \in S^2$ that reaches a state $s'$ where $\psi$ holds, we also have that $s \overset{\varphi \Rightarrow \psi}{\rightarrow} s'$.
- Due to the way we approach relation $\overset{\varphi \Rightarrow \psi}{\rightarrow}$, the problem of how we determine if a process is of type of another process of the form $\varphi \Rightarrow \psi$ is simplified considerably. Formula $\forall (\varphi \Rightarrow \psi). \vartheta$ expresses the following property: If we are in a state where $\neg \varphi$ holds, then no matter which process will be executed and no matter which state will be reached, in that state, formula $\vartheta$ holds. Additionally, when we are in a state, no matter which process will be executed and no matter which state will be reached, if in that state formula $\psi$ holds, then formula $\vartheta$ also holds.

We need to express somehow the property of a state, according to which, no matter which process will be executed and no matter which state will be reached after its execution, in that state, a formula $\vartheta \in L_s$ will hold. We notice that formula $\forall (tt \Rightarrow tt). \vartheta \in L_s$ is what we are looking for. We use constant $tt$ to denote any tautology and as a result, it is satisfied by all the states of any $L$-model.

**Proposition 1.** Let $N = (F, e, (\pi^N)_{i \in I})$ be an $L$-model, with $F = \langle S, (\overset{\varphi}{\rightarrow})_{e \in P}, I \rangle$, and $s \in S$ a state.
For the relation $\sigma \mathbin{\rightarrow\rightarrow} \tau$ is reflexive and transitive:

$$\forall \rho, \tau \in \Sigma^+, \rho \mathbin{\rightarrow\rightarrow} \tau \iff \rho = \tau$$

For each $a \in \Sigma^+$, $\sigma \mathbin{\rightarrow\rightarrow} \tau$.

**Proposition 2.** Let $\mathcal{N} = (S, (\mathcal{N}^a)_{a \in \mathcal{P}}, I, e, (\mathcal{N}^N)_{e \in I})$ be an $\mathcal{L}$-model and $\varphi \Rightarrow \psi \in \mathcal{L}_a$ a process. Then, for processes $\neg \varphi a, a \psi \in \Sigma^+$, with $a \in \Sigma^+$, the following properties hold:

$$\neg \varphi a \subseteq \varphi \Rightarrow \psi$$

$$\varphi \psi \subseteq \varphi \Rightarrow \psi$$

$$(4)$$

In order to clarify the importance of the previous proposition, notice Figure 1. There is an $\mathcal{L}$-model which consists of four states and they satisfy the formulas with which they are labelled. The question that we need to answer is whether state $s_2$ satisfies formula $\vartheta$ or not, as $s_1$ satisfies formula $\forall (\varphi \Rightarrow \psi) \vartheta$. Someone could argue that we are not in position to give an answer, as $\pi$ is not of type $\varphi \Rightarrow \psi$, i.e. $\pi \not\subseteq \varphi \Rightarrow \psi$, and as a result the structure of the specific $\mathcal{L}$-model does not require $s_2$ to satisfy formula $\vartheta$. Unfortunately, this is not correct. According to Proposition 2, $\neg \varphi a \subseteq \varphi \Rightarrow \psi$ and $\varphi \psi \subseteq \varphi \Rightarrow \psi$. So, it is immediate that states $s_1$ and $s_2$ have to satisfy formula $\vartheta$.

**Figure 1: An example of an $\mathcal{L}$-model.**

**Lemma 2.** Let $\mathcal{N} = (S, (\mathcal{N}^a)_{a \in \mathcal{P}}, I, e, (\mathcal{N}^N)_{e \in I})$ be an $\mathcal{L}$-model and $\varphi \Rightarrow \psi \in \mathcal{L}_a$ a process. We consider relations $T_1, T_2 \subseteq S \times S$ which are defined as follows:

$$T_1 = \{(s, s') \in S^2 \mid \exists a \in \Sigma^+ (s \xrightarrow{\mathcal{N}^a} s' \wedge (s \parallel \neg \varphi \Rightarrow s' \parallel \neg \mathcal{N}^a \vartheta))\}$$

$$T_2 = \varphi \Rightarrow \psi \cup \{(s, s') \in S^2 \mid \exists a \in \Sigma^+ (s \xrightarrow{\mathcal{N}^a} s' \wedge (s \parallel \neg \varphi \Rightarrow s' \parallel \neg \mathcal{N}^a \vartheta))\}$$

$$(5)$$

$$(6)$$

For the relation $\varphi \Rightarrow \psi$, we have that $\varphi \Rightarrow \psi = T_1 = T_2$.

**Proof.** It is immediate that $\varphi \Rightarrow \psi = T_1$. Then, by using Proposition 2, we get that $T_1 \subseteq T_2$. It is also immediate that $T_2 \subseteq T_1$. In order to show that $T_1 \subseteq T_2$, it is enough to show that an arbitrary transition $s \xrightarrow{\mathcal{N}^a} s'$ that exists in $T_1$ with $a \in \Sigma^+$, also exists in $T_2$. If $a \in \Sigma^+$ is not a pure sentential test (i.e. $a \not\in \mathcal{L}_a^+$), we have to show that there is $b \in \Sigma^+$, such that $s \xrightarrow{\mathcal{N}^b} s'$. Intuitively, since execution of tests leads to the same state, the existence of such $b \in \Sigma^+$ is immediate. We leave details to the interested reader.

Now, we are in position to approach formulas of the form $\forall (\varphi \Rightarrow \psi) \vartheta$ in such way that the issue of typing judgement is considerably simplified.

**Lemma 3.** The following semantical equivalences hold:

$$\forall (\varphi \Rightarrow \psi) \vartheta \equiv \left(\neg \varphi \Rightarrow \forall (tt \Rightarrow t) \vartheta \right) \wedge \forall (tt \Rightarrow t) \vartheta$$

$$(7)$$

$$\equiv \left(\varphi \Rightarrow \forall (tt \Rightarrow t) \vartheta \right) \wedge \forall (tt \Rightarrow t) \vartheta$$

$$\forall (\varphi \Rightarrow \psi) \vartheta \equiv \left(\vartheta \Rightarrow \forall (tt \Rightarrow t) \vartheta \right) \wedge \forall (tt \Rightarrow t) \vartheta$$

$$(8)$$

**Proof.** The key element of this proof is the way that we approach relation $\varphi \Rightarrow \psi$ in Lemma 2, as a result of Proposition 2. For more details, we refer the reader to [11].

**4. TABLEAU CALCULUS $\mathcal{T} \tau PD$L**

**4.1 The Static Rules**

In this subsection, we present the static rules of $\mathcal{T} \tau PD$L calculus as $\alpha$- and $\beta$-formulas, in the way that they appear in Table 2. The $\alpha$-formulas ($\beta$-formulas) are the principal formulas, whereas the $\alpha_1, \alpha_2$ ($\beta_1, \beta_2$) are the side formulas of the corresponding rules. As we are going to see in the next section, we adopt the satisfiability algorithm of Goré and Widmann [6], in which a tableau node is not just a set of formulas. Nevertheless, we choose to present the rules in this way, here, independently of the way that the other attributes of a tableau node evolve, in order to abstract them from the corresponding details. We draw the reader’s attention to the rules that concern formulas with a process of the form $\varphi \Rightarrow \psi$, capabilities statements and formulas with the backwards possibility operator.

**Remark 2.** The convention for $\alpha$- and $\beta$-formulas is that formulas $\alpha \leftrightarrow \alpha_1 \land \alpha_2$ and $\beta \leftrightarrow \beta_1 \lor \beta_2$ are logically valid. Unfortunately, in our case, this is not always true. In rules $(\forall a)$ and $(N \exists t)$, which concern formulas $\forall (tt \Rightarrow t) \vartheta$ and $\neg (\vartheta \exists t)$, respectively, only the left-to-right implication holds. We point out that formula $\forall (tt \Rightarrow t) \vartheta \Rightarrow \vartheta \Rightarrow \vartheta \forall (tt \Rightarrow t)$ is logically valid, but we notice that formulas $\forall (tt \Rightarrow t) \vartheta \Rightarrow \vartheta \forall (tt \Rightarrow t)$ and $\forall (tt \Rightarrow t) \vartheta \exists t$ are equivalent (see Proposition 1). Similarly for the case of the $(N \exists a)$ rule. Again, in rules $(N \forall a)$ and $(N \exists a)$ which concern formulas $\neg (\forall (\varphi \Rightarrow \psi)) \chi$ and $\chi \exists (\varphi \Rightarrow \psi)$, respectively, also only the left-to-right implication holds. We notice that formulas $\varphi \lor \neg \varphi$ and $\varphi \lor \neg \varphi$ are logically valid. By some abuse and for uniformity of notation, we also treat them as $\alpha$- and $\beta$-formulas. Whenever there is reason to argue separately for these rules, we do it.

**Lemma 4.** In light of Remark 2, formulas $\alpha \leftrightarrow \alpha_1 \land \alpha_2$ and $\beta \leftrightarrow \beta_1 \lor \beta_2$ are logically valid.

**Proof.** Most of the cases follow immediately by the results that are reported in [9] and [8]. The $\tau PD$L language can be interpreted by two kinds of semantics, the type semantics [9] and the standard relational semantics [8]. We know that the two systems are equivalent, in the sense that a formula is satisfiable in type semantics iff it is satisfiable in the standard relational semantics. For the capabilities statements, see Lemma 1 (see also [9, 8]). The validity of the formulas which correspond to the rules $(\forall a)$ and $(N \exists a)$ is immediate consequence of Lemma 3 of this paper. We leave details to the interested reader.

**Corollary 1.** The static rules of $\mathcal{T} \tau PD$L calculus are sound.
4.2 The Transitional Rule

The last rule of the calculus is the transitional one. Let \( \Gamma \subseteq L_\alpha \) be a set of formulas:

\[
(\text{trans}) \quad \Gamma \quad \Delta_1 \& \Delta_2 \& \cdots \& \Delta_n
\]

where if \( \Theta \) is the set of formulas of the form \( \neg \forall A \chi \) and \( \chi \exists A \) with \( A \in \Sigma \) that appear in \( \Gamma \), then \( n = 2^\Theta \geq 0 \) and for \( i = 1, \ldots, n \), by distinguishing cases for \( \forall i \in \Theta \), set \( \Delta_i \) is defined as follows:

- \( \forall i = \neg \forall \forall \chi \):  
  \[
  \Delta_i = \{ \neg \chi \} \cup \{ \neg \forall \forall \varphi \in \Gamma \} \cup \\
  \{ \forall (tt \Rightarrow tt). \varphi \mid \forall (tt \Rightarrow tt). \varphi \in \Gamma \}
\]

- \( \exists i = \exists \forall \varphi \chi \) and \( \varphi \in \Gamma \):  
  \[
  \Delta_i = \{ \neg \chi \} \cup \{ \exists \forall (tt \Rightarrow tt) \varphi \mid \forall (tt \Rightarrow tt) \varphi \in \Gamma \}
\]

- \( \forall i = \neg \forall (\varphi \Rightarrow \varphi_2) \chi \) and \( \varphi \in \Gamma \):  
  \[
  \Delta_i = \{ \chi \} \cup \{ \neg \exists (tt \Rightarrow tt) \varphi \mid \forall (tt \Rightarrow tt) \varphi \in \Gamma \}
\]

- \( \exists i = \exists \forall \varphi \varphi \) and \( \varphi_2 \in \Gamma \):  
  \[
  \Delta_i = \{ \chi \} \cup \{ \neg \exists (tt \Rightarrow tt) \varphi \mid \forall (tt \Rightarrow tt) \varphi \in \Gamma \}
\]

- \( \forall i = \exists \exists \varphi \varphi \) and \( \varphi_2 \in \Gamma \):  
  \[
  \Delta_i = \{ \chi \} \cup \{ \neg \exists (tt \Rightarrow tt) \varphi \mid \forall (tt \Rightarrow tt) \varphi \in \Gamma \}
\]

**Lemma 5.** The (trans)-rule is sound.

**Proof.** We assume that \( \Gamma \) is satisfiable by a state \( s \in S \) of an \( \mathcal{L} \)-model \( \mathcal{N} = (\mathcal{S}, (\mathcal{S}_i), \mathcal{E}, \mathcal{I}, (\mathcal{I}_n): \in \cal{I}) \), and we show that each \( \Delta_i \) is also satisfiable. We examine only the case of a formula \( \neg \forall (\varphi_1 \Rightarrow \varphi_2) \chi \in \Gamma \), while the remaining cases follow in a similar fashion. By semantic definitions (see Table 1), since \( s \models \neg \forall (\varphi_1 \Rightarrow \varphi_2) \chi \) there is a state \( s' \models \neg \forall \neg (\varphi_1 \Rightarrow \varphi_2) \chi \). If \( \varphi_1 \in \Gamma \), then \( s \models \neg \forall \varphi_1 \); since \( s \models \neg \forall \varphi_1 \) and \( s \models \neg \forall \varphi_2 \) can we conclude that \( s \models \neg \forall \varphi_2 \). Furthermore, for each formula \( \forall (tt \Rightarrow tt). \varphi \in \Gamma \), since \( s \models \forall (tt \Rightarrow tt) \varphi \) by Proposition 1, we can conclude that \( s \models \forall (tt \Rightarrow tt) \varphi \) and \( s \models \neg \forall (tt \Rightarrow tt) \varphi \). We also know that \( \varphi \models \varphi \Rightarrow \varphi \) and thus, \( s \models \neg \forall (tt \Rightarrow tt) \varphi \). \( \square \)

Notice that formulas of the form \( \neg \forall (\varphi \Rightarrow \varphi) \chi \) and \( \neg \exists (\varphi \Rightarrow \varphi) \chi \) are used not only as principal formulas of the (trans)-rule, but also as principal formulas of the static rules \( (N\forall) \) and \( (\exists \Rightarrow) \), respectively.

5. THE SATISFIABILITY ALGORITHM

In this section, we present the way that the satisfiability algorithm of [6] can be used for the system of \( \tau \mathcal{PDL} \). We assume that the reader is familiar with the algorithm of [6] and in the following we focus on specific details and we point out the required adaptations that must be made.

In the sequel, as a matter of notation, the undefined value is denoted as \( \bot \) and if \( X \) is a set, then \( X^+=X \cup \{ \bot \} \). If we consider a function \( f: X \rightarrow Y^+ \), then we denote as \( f^\bot \) the function which is undefined for all the values of its domain.

**Definition 5.** A tableau proof (or just tableau) for a formula \( \varphi \in \mathcal{L}_\alpha \) is a directed graph \( G_\varphi = (V, E) \) with \( V \) a set of nodes and \( E \subseteq V^2 \) a set of directed edges, such that:
A tableau node \( x \in V \) is a setuple \( (\Gamma_x, ann_x, pst_x, ppr_x, drc_x, id_x, sts_x) \), such that:

- \( \Gamma_x \subseteq \mathcal{L} \) is a set of formulas,
- the annotation function \( ann_x : \text{Ev} \rightarrow \Gamma^+_x \), with \( \text{Ev} \), to be the set of eventualities of \( \Gamma_x \),
- the parent state \( pst_x \in (V \cup \{self\})^+ \), where \( self \) is a constant value,
- the parent process \( ppr_x \in \Sigma^+ \),
- the direction attribute \( drc_x \in \{+, -, \}^+ \),
- the attribute \( id_x \in \mathbb{N}^+ \) indicates when the status of a node is defined for the first time,
- the status \( sts_x \in \{\text{unexp, undef}\} \cup \{(\text{closed, alt}_x) | alt_x \subseteq 2^{\mathcal{L}^+} \} \cup \{(\text{open, prs}_x, alt_x) | prs_x : \text{Ev} \rightarrow (^{\mathcal{L}^+ \times \text{Ev})^+} \} \) and \( alt_x \subseteq 2^{\mathcal{L}^+} \) with \( prs_x(\varphi) \subseteq \{(z, \psi) | (x, z) \in E^+ \} \) and \( \psi \in \Gamma_x \).

- The root node \( V \ni \tau = \{\varphi\} \), \( ann \ni \perp, \perp, \perp, \perp, \perp, \text{unexp} \).
- The graph \( G \) is constructed as follows: While one of the special-purpose rules is applicable, then apply it.

For the attributes of a node (besides the direction), as well as for the special-purpose rules, we refer the reader to [6].

There are four special-purpose rules and we shortly mention that the first expands nodes, in our case according to the rules of \( \mathcal{T} \mathcal{P} \mathcal{D} \mathcal{L} \) calculus, the second defines the status of a node, the third keeps the status of nodes up-to-date and the last one closes nodes which have an unfinished eventuality.

We call a node \( x \) state iff its set of formulas is fully saturated as far as the static rules are concerned and additionally, \( pst_x = \text{self} \). On the other hand, we call a node \( x \) special iff its set of formulas is saturated, but this time \( pst_x \neq \text{self} \). The question that arises is why there is this distinction between these two types of nodes as both of them are fully saturated. According to the algorithm given in [6], the reason of the existence of the special nodes is to “restart” them, in the case that their set of formulas will be proven to be “too small”. More specifically, a special node has always at most one state node as its child with the corresponding edge to be labelled with the marker “cs” (corresponding state). Both of them have the same set of formulas. The special node monitors its corresponding state and if it is proven to be “too small”, then alternative nodes are created as its children, with the required additional formulas.

In a tableau, it is impossible to exist two states with the same set of formulas. Before defining a state as part of the tableau, the algorithm examines if there is already one with the same set of formulas and if this is the case, then it doesn’t create the same node again. A special node has always at most one state as its child. Of course, it may have additional children nodes (the alternative ones). On the other hand, due to the state caching, a state may have more than one special node, as parent-nodes.

The reader who is familiar with the satisfiability algorithm of [6] may have noticed minor differences. We remind that, in [6], only states do not have a parent state (i.e. \( pst = \perp \)), whereas the rest types of nodes always have one. In order to manage to equip all non state nodes with a parent state, in the case that \( \varphi \) is the input formula, they use as input formula, formula \( \langle d \rangle \varphi \), where \( d \) is an atomic program such that it doesn’t appear in \( \varphi \). Thus, it is legitimate to say that formula \( \varphi \) is satisfiable iff formula \( \langle d \rangle \varphi \) is satisfiable.

In \( \mathcal{T} \mathcal{P} \mathcal{D} \mathcal{L} \), this is not true, as it is always possible to appear a formula \( \neg \langle \chi \rangle \langle tt \Rightarrow tt \rangle \) and as a result no matter which process \( \pi \in \text{AtP} \) we are going to choose, we know that in any \( \mathcal{L} \)-model, it is true that \( \pi \ni \langle tt \Rightarrow tt \rangle \). We treat state nodes in a slightly different way, as we identify them by assigning to their parent state attribute the constant value \( \text{self} \). Additionally, in contrast with [6], we allow to exist non states in which their parent state attribute is undefined (e.g. the root node), which simply indicates that they do not have a parent state. We just need a technical distinction between special and state nodes with the same set of formulas. In the case of a special node, we need to know the actual parent state (if any exists), while in the case of a state, we don’t.

When static rules (\( \mathcal{N}_{\text{trs}} \) and \( \mathcal{E}_{\text{trs}} \)) are applied, although the corresponding principal formula might be an eventualty, there isn’t the need for the annotation function to record the reduction. The specific rules do not lead to “at a world” cycles or fulfill eventualties and we treat them as exceptions of the general form of a \( \beta \)-rule. Furthermore, for the same reasons, the specific rules are applied only in the case that none of the corresponding \( \beta_1, \beta_2 \) formulas exists in the expanded node and not in the case that \( \beta \in \text{Ev} \) and \( ann(\beta) = \perp \), where \( ann \) is the annotation function of the expanded node.

The backwards possibility operator has led us to introduce a new attribute, the direction one \( \text{drc} \). Suppose that \( z \) is the defined child node after the application of the \((\text{trans})\)-rule. If the corresponding principal formula was of the form \( \neg \forall A \chi \), with \( A \subseteq \Sigma \), then \( \text{drc}_z = + \). In the different case, i.e. it was of the form \( \exists A \chi \), with \( A \subseteq \Sigma \), then \( \text{drc}_z = - \). The intuition behind our choice should be clear. The attribute \( \text{drc} \) denotes the direction of the transitions of the accessibility relation. During the application of the other rules, it is passed from node to node unchanged, except for the case of a state in which it becomes undefined.

The direction attribute affects the way that we define the status of a special node. In particular, it guides us how we should calculate the alternative set of formulas in order to determine whether its parent state has the required formulas or not. In the following, let \( x \) be a special node (\( pst_x \neq \perp \)):

\[
sts_x = \begin{cases} 
\text{det-sts-spl}(x) & \text{if } pst_x = \perp \text{ or } \langle ppr_x \rangle \perp \text{ and } \Delta \subseteq \Gamma \text{ of } pst_x \ni \perp \ni \perp \ni \perp \ni \perp \ni \perp \\
\text{(closed, } \Delta \text{)} & \text{if } pst_x \neq \perp \text{ and } \Delta \not\subseteq \Gamma \text{ of } pst_x 
\end{cases}
\]  

In Eq. (16), for the procedure \( \text{det-sta-spl}(x) \), we refer the reader to [6], whereas the set \( \Delta \subseteq \mathcal{L} \) is defined as follows:

- In the case that \( \text{drc}_x = + \) and \( ppr_x \in \text{AtP} \):
  \[
  \Delta = \{\neg \varphi \mid \neg \langle \varphi \rangle \in \Gamma_x \} \cup \{\neg \langle \varphi \rangle \in \Gamma_x \} \cup \{\varphi \in \Gamma_x \}
  \]
- In the case that \( \text{drc}_x = + \) and \( ppr_x \in A_{\neg \text{ss}} \):
  \[
  \Delta = \{\varphi \mid \varphi \in \Gamma_x \}
  \]
- In the case that \( \text{drc}_x = - \) and \( ppr_x \in \text{AtP} \):
  \[
  \Delta = \{\varphi \mid \varphi \in \Gamma_x \}
  \]
- In the case that \( \text{drc}_x = - \) and \( ppr_x \in A_{\neg \text{ss}} \):
  \[
  \Delta = \{\varphi \mid \varphi \in \Gamma_x \}
  \]

Notice that \( pst_x \neq \perp \) iff \( \text{drc}_x \neq \perp \) iff \( ppr_x \neq \perp \).
We refer to a node as open when its status is open and as closed, when its status is closed. Furthermore, we call a tableau open (respectively closed) iff its root node is open (resp. closed). Finally, we say that a formula $\varphi \in \mathcal{L}_\ast$ is satisfiable iff there is an open tableau $G_\varphi$.

In Figure 2, we see an example of a tableau. We point out that what we see in the figure is the final form of the tableau. We have omitted an attribute $ann$, as there aren’t any eventuations. Additionally, we use operators which do not belong to the native $\tau PDL$ language. Nevertheless, we consider the corresponding rules of the $T \tau PDL$ calculus with the appropriate changes.

6. Complexity, Soundness and Completeness

In this section, we argue about the complexity of the algorithm and we shortly discuss its soundness and completeness. The corresponding proofs which are presented in [11] follow those that have been given in [6] and also in [12].

Let the closure $cl(\varphi)$ of the input formula $\varphi$ be the smallest set of formulas that can appear in a tableau proof, according to the rules of the calculus $T \tau PDL$. The following proposition is an immediate consequence of the way that the tableau rules have been defined and of the results that are reported in [9]. Additionally, we refer the reader to see in [9] the definitions of the Fischer-Ladner closure, as well as of the size of a formula, denoted as $|\varphi|$.

**Proposition 3.** Let $G_\varphi = (V, E)$ be a tableau proof.

1. For the cardinality of the closure $cl(\varphi)$, we have that $|cl(\varphi)| \in \mathcal{O}(|\tau FL(\varphi)|)$ and as a result $|\tau cl(\varphi)| \in \mathcal{O}(|\varphi|^2)$.

2. For each node $x \in V$, we have that $\Gamma_x \subseteq cl(\varphi)$ and as a result $|\Gamma_x| \in \mathcal{O}(|\varphi|^2)$.

**Theorem 1.** In the worst case, the algorithm runs in exponential time in the size of the input formula.

**Proof.** The proof proceeds exactly as in [6]. The only difference is in the cardinality of the closure of the input formula, see Proposition 3.

**Theorem 2 (Soundness).** If there is an open tableau proof $G_\varphi = (V, E)$ for formula $\varphi \in \mathcal{L}_\ast$, then $\varphi$ is satisfiable.

**Proof sketch.** When we say that the algorithm is sound, we mean that the formula for which the algorithm has decided that is satisfiable, is indeed satisfiable, i.e. there is an $\mathcal{L}$-model which satisfies it. In order to show soundness, there are two distinct steps that must be carried out. First, we have to show that a Hintikka structure can be defined from an open tableau. A Hintikka structure is a transition system in which the states are labelled with formulas. The transition relation as well as the labelling function of its states fulfill specific conditions. Secondly, relying on the specific Hintikka structure, we define an $\mathcal{L}$-model and we show that it satisfies each formula with which a state of the Hintikka structure is labelled. Besides the cases of the new constructs that we need to examine, there is a peculiarity in the way that the accessibility relation of the $\mathcal{L}$-model is defined by the transition relation of the Hintikka structure. Suppose
that $(\mathcal{A})_{AC\Sigma^\psi}$ is the accessibility relation of an $L$-model $\mathcal{N}$ which corresponds to a Hintikka structure $\mathcal{H}$ of a tableau proof. Moreover, let $(A)_{AC\Sigma^\psi}$ be the transition relation of $\mathcal{H}$. The relation $\mathcal{A}^\psi\mathcal{A}^\psi$ is not restricted to the transitions of $\mathcal{H}$ which are labelled with process $\varphi \Rightarrow \psi$, but is defined as follows: 

$\mathcal{A}^\psi\mathcal{A}^\psi = \mathcal{A}^\psi \cup \{(s, s') \in S^2 | \exists \alpha \in \Sigma^\psi (s \rightarrow^\alpha s' \land (s \models \neg \psi \Rightarrow s' \models \neg \psi))\}$. For more details, we refer the reader to [11].

**Theorem 3 (Completeness).** If formula $\varphi \in L_\omega$ is satisfiable, then all tableau proofs $G_\varphi = (V, E)$ are open.

**Proof Sketch.** In order to show completeness, we must prove that whenever the input formula is satisfiable, then the constructed tableau is open, i.e., the root node is open. There are four reasons for which a node can be defined as closed, as direct consequence of its own attributes and not as a result of its corresponding closed children nodes. These reasons are: a contradiction, an “at a world” cycle, if the specific node is a special one and its parent state has not the required formulas (see the last case of Eq. (16)), and finally an unfulfilled eventuality. We must show that there is an appropriate subgraph of a tableau proof, called fat path (the specific term has been used by Pratt in [15]), in which none of the previous cases occurs and hence, we obtain an open tableau. A fat path of a tableau has the same root node with the specific tableau and for each or-node ($\exists$- or special-node) which belongs to the fat path, at least one of its children also belongs to it, while for each and-node (state node), all of its children belong to the fat path. Notice that an $\alpha$-node has always only one child. We remind the reader that tableau rules are sound (see Corollary 1 and Lemma 5). For more details, we refer the reader to [11].

The following corollary is an immediate consequence of Theorems 2 and 3.

**Corollary 2.** If there is a closed tableau proof $G_\varphi = (V, E)$, then formula $\varphi \in L_\omega$ is unsatisfiable and all tableau proofs for formula $\varphi \in L_\omega$ are closed.

### 7. Conclusions

In this paper, we have given a satisfiability algorithm for the system of rPDL, by extending the algorithm of [6]. More specifically, the tableau calculus $T_{\tau PDL}$ includes the appropriate rules which handle formulas with processes of the form $\varphi \Rightarrow \psi$, capabilities statements, as well as formulas with the backwards possibility operator. In the case that we need to determine whether a process is of type $\varphi \Rightarrow \psi$ or not, we have redefined the accessibility relation of a process of the form $\varphi \Rightarrow \psi$ in such way that this issue is considerably simplified. Furthermore, we have made minor changes in the algorithm of [6], for example the addition of another attribute in the definition of a tableau node (i.e. the drc attribute) in order to be possible for the algorithm to handle the backwards possibility operator in the appropriate way. We have shown that the calculus $T_{\tau PDL}$ in the context of the satisfiability algorithm is sound and complete and finally, we have shown that the given decision procedure is exponential.

### 8. References


