Information and Computation **162**, 117–137 (2000)
doi:10.1006/inco.1999.2852, available online at http://www.idealibrary.com on IDEAL®

Combining Deduction and Model Checking into Tableaux and Algorithms for Converse-PDL[1]

Giuseppe De Giacomo and Fabio Massacci[2]

Dip. di Informatica e Sistemistica, Università di Roma "La Sapienza," Italy

This paper presents a prefixed tableaux calculus for Propositional Dynamic Logic with Converse based on a combination of different techniques such as prefixed tableaux for modal logics and model checkers for μ-calculus. We prove the correctness and completeness of the calculus and illustrate its features. We also discuss the transformation of the tableaux method (naively NEXPTIME) into an EXPTIME algorithm.

1. INTRODUCTION

Propositional dynamic logics (PDLs) are modal logics introduced in [10] to model the evolution of the computation process by describing the properties of states reached by programs during their execution [15, 24, 27]. Over the years, PDLs have been proved to be a valuable formal tool in computer science, logic, computational linguistics, and artificial Intelligence far beyond their original use for program verification (e.g., [4, 12, 14, 15, 24, 23]).

In this paper we focus on *Converse-PDL* (CPDL) [10], obtained from the basic logic PDL by adding the converse operator to programs, which is interpreted as the converse of the (input–output) relation interpreting the program. A possible use of the converse is for formalizing preconditions; e.g., $[\pi^-]\varphi$ can be interpreted as "before running program π, property φ must hold."

There are certain applications of PDLs where the ability of denoting converse programs is essential. For instance, recent research in knowledge representation [7, 5, 23] points out a tight correspondence between PDLs and a family of class-based knowledge representation formalisms, known as the description logics [29]. Such research shows that, by means of suitable polynomial reductions, inference in very expressive description logics can be rephrased as inference in CPDL. Thus, inference procedures for CPDL can be exploited as the reasoning core for such knowledge

[1] A preliminary version of this work appeared in [8]

[2] Communicating author: Fabio Massacci, Dipartimento di Informatica e Sistemistica, via Salaria 113, I-00198 Roma, Italy, E-mail: massacci@dis.uniroma1.it.

0890-5401/00 $35.00

representation formalisms. Indeed, this was one of the main motivations that has led us to look into inference procedures for CPDL.

From the formal point of view, CPDL shares many characteristics with the basic PDL, and many results for PDL extend to CPDL without difficulties. For instance the proofs of finite model property for PDL in [10] are easily extended to CPDL, as well as the proof of EXPTIME-completeness in [21]. However, efficient—in practical cases—inference procedures have been successfully developed for PDL, but their extension to CPDL has proved to be a difficult task and unsuccessful till now (to the best of our knowledge).

To be precise, inference procedures based on model enumeration [10, 21] or on automata on infinite trees [27] have been extended to converse of programs. Yet, these procedures are more suited for proving theoretical results than for being used in applications. Tableaux procedures for PDL [20, 22], which are typically simpler in practice, have never been extended. In [20] Pratt said "We do not have a practical approach to this difficulty with converse, and our practical procedure therefore does not deal with converse."

In this paper we propose a *prefixed tableaux calculus* for PDL and extend it to deal with the converse operator, thus obtaining the first tableaux calculus for CPDL. The tableaux-based technique we propose here combines in a natural way a number of intuitions and techniques that have been developed for validity–satisfiability checking [9, 17, 20, 22, 27, 28] and model checking [3, 17, 25, 24] with prefixed tableaux [11, 13, 18] for modal logics. Indeed, the work in this paper confirms that the combination of model checking and theorem proving techniques, as witnessed also at the recent CAV conference [1], may be very fruitful also for purely deductive techniques.

1.1. Plan of the Paper

In the rest of the paper we present at first the intuitions underlying our work and emphasize the possibility of merging theorem proving and model checking techniques for dynamic and temporal logics (Section 2). We introduce some notions on CPDL (Section 3), present the tableaux calculus (Section 4), give examples (Section 5) and prove its soundness and completeness (Section 6). Finally we sketch the transformation of NEXPTIME tableaux into EXPTIME algorithms (Section 7) and conclude (Section 8).

2. TABLEAUX AND MODEL CHECKING TECHNIQUES

The use of tableaux for modal logics dates back to Kripke (see [13] for a recent overview or [11] for a classical treatment) and efficient procedures based on depth first search are known [14, 16]. Tableaux methods for satisfiability checking have been developed for the basic PDL [20, 22], but not for many extensions, such as CPDL, whereas automata theoretic techniques are available [26, 27].

The problem is that PDLs, temporal logics, or the modal μ-calculus cannot be tamed only by traditional tableaux methods for logics of knowledge and belief:

— Some formulae impose fixpoint properties on paths (in the rest of the paper we call them *iterated eventualities or necessities*) so that the traditional local validation of a model is not enough.

— PDLs, like the modal μ-calculus, have an added difficulty w.r.t. temporal logics (which have only one transition, next, and simpler eventualities) since programs can be extremely complicated and deciding whether two states are reached by a program may have the same complexity of the original problem.

— The converse operator (or past for temporal logics) imposes constraints on previously visited states so that these must be reprocessed.

— The models of a formula can be exponential in its size and therefore direct depth-first search with backtracking at choice points may lead to a double exponential algorithm (although decidability for these logics is in EXPTIME).

Tableaux approaches for PDLs (and expressive temporal logics) [20; 22; 9; or 17, Sect. 5.1–5.3] cope with these problems in two conceptual phases:

1. Construct an AND-OR *graph* based on the Fisher–Ladner closure which satisfies the constraints of a normal modal logic, where nodes with the same formulae are identified and nodes that are locally inconsistent are deleted.

2. model check the pseudo-model thus obtained for iterated eventualities (deleting nodes with unfulfilled eventualities).

The separation into two phases is present also in automata techniques where the automaton corresponding to a formula[3] is the product of two automata: one for checking local modal conditions and one for eventualities [26–28].

The first intuition is to build the AND-OR graph of phase 1 in an *incremental fashion*, as particles tableaux for temporal logics [17],

Intuition 1. A node of the tableau is a set of formulae and new nodes are generated only by reducing already present formulae.

In this way we generate nodes and check for local consistency on-the-fly. In contrast with maximal model techniques [21], tableaux methods generate all subsets of the Fisher–Ladner closure only in the worst cases.

The idea of proceeding in an incremental fashion can be pushed further by executing the two phases, building the AND-OR graph and model checking eventualities, at the same time[4]. In this way we need not wait for the construction of a (large) model just to discard it with the very first eventuality we checked.

Intuition 2. Merge the pseudomodel checking stage for iterated eventualities with the incremental tableaux construction.

Thus an algorithm may close a branch due to an unfulfilled eventuality *before* the construction of the whole model, improving its average time behaviour.

[3] One constructs an automaton that accepts the tree models of the formula and hence a formula is satisfiable iff the automaton accepts some models [9, 27, 28].

[4] There are incremental model checking techniques for temporal logics [3, 2, 17].

For temporal logics this problem has been solved [3, 17]: as soon as the same node is repeated along the path and we find again the same eventuality (i.e., the same formula) we conclude that this is a bad (unfulfilling) loop.

For PDL/CPDL there are more difficulties: to conclude that an eventuality of the form $\langle \rho^* \rangle \varphi$ is not fulfilled, it is not enough to "find again" the same eventuality into a duplicate node. One must verify that the two nodes are connected by some ρ-steps and ρ can be a complex program. This problem does not arise (mutatis mutandis) in temporal logic because ρ can only be "next".

The solution developed by Pratt in [22] has been the introduction of an operator $\Rightarrow$ to link every eventuality of the form $\langle \rho \rangle \varphi$ with the corresponding fulfilling node where φ holds. However this may lead to a cumbersome tableau and to many unnecessary relations between formulae. Moreover it does not completely address the issue of merging the two phases, because one needs to construct the transitive closure of $\Rightarrow$ and this can be done only after the pseudomodel construction is terminated. The automata theoretic approaches [27, 28], instead of building such $\Rightarrow$-links, use suitable eventuality automata, which also require the construction of the pseudomodel first.

There is another way that fully addresses this problem, based on the same intuition of *history variables*, i.e., "auxiliary variables whose values in the current states reflect all the facts we need to know about the past" [17][5]. This idea has been used for model checking techniques for the modal μ-calculus, introducing new propositional variables (names) for denoting fixpoints [24].

If we introduce a name X to denote $\langle \rho^* \rangle \varphi$ and use the name for subsequent reductions then we have a direct way to identify, by just looking at them, when two nodes are connected by some ρ-steps and therefore use the simple check of temporal logics (repeated eventually into duplicated nodes).

Intuition 3. Associate iterated eventualities with new names $X \doteq \langle \rho^* \rangle \varphi$ and define reduction rules which use the name rather than the formula.

Next we focus on the converse operator. One of the intuitions used for automata theoretic techniques [27] is to use *two-way* tree models where a transition can go from the current state to its child (direct) or vice versa (converse). We combine this idea with the notion of prefixed tableaux developed for modal logics of knowledge and belief [11, 18] to get the following:

Intuition 4. Associate to each a formula a prefix formed by the sequence of states and direct–converse transitions starting from the initial state and ending in the current state where the formula is supposed to hold. Devise rules which take into account both formulae and prefixes.

In this framework we simply devise tableaux rules for the converse of an (atomic) program with the techniques for symmetric modal logics B or S5 from [18]:

Intuition 5. Necessity-like formulae with direct programs such as $[A]\varphi$ move formulae forward when the last prefixed transition is direct and backward when the last transition is converse. $[A^-]$ does the opposite.

[5] A related idea is used for automata [27] by introducing state formulae like $cycle(\rho)$ to represent the fact that there is a ρ-loop which passes through that state.

The next step, when we look for duplicated nodes, is to take into account what has been also noted for automata [27]: "we also have to know what program connects this node to its predecessor." With prefixed tableaux this check is straightforward: it is embedded into the label of the formulae of each state.

We still need to guess a small subset of formulae that characterizes past computations, especially when we are trying to discard unfulfilled eventualities[6]. The existence of a set characterizing past symmetric computations was pointed out in [11] and used in [6] to provide a translation of CPDL into PDL.

Intuition 6. To identify two duplicate nodes the formulae which compose them must also completely specify the past (the arriving program), so we use analytic cut on a small subset of subformulae to complete the nodes.

Last, we want to lower the complexity of the tableaux procedure from NEXPTIME to EXPTIME. The key point is that to avoid exponential behavior on average it is not enough to store only the visited nodes as in [22]. We must also track the nodes already proven to be inconsistent as in [3].

Intuition 7. Use a global data structure to store bad sets which have been proven to lead to contradictions in previous expansions of the tableau. Before expanding a node always check that it is not a previously seen bad set.

3. SYNTAX AND SEMANTICS OF CPDL

We sketch some basic notions on CPDL (see [15, 24] for an introduction).

If $\mathscr{A}$ is a set of direct atomic programs (a, b, etc.), $\mathscr{P}$ a set of propositional letters (P, Q, etc.), the formulae φ, ψ and the programs ρ, χ of CPDL are

$$\varphi, \psi ::= P \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle \rho \rangle \varphi \quad \rho, \chi ::= a \mid \rho; \chi \mid \rho \cup \chi \mid \rho^* \mid \rho^- \mid \varphi?$$

Other connectives can be seen as abbreviations—e.g., $[\rho]\varphi \equiv \neg\langle \rho \rangle \neg\varphi$. W.l.o.g. we restrict the converse operator to atomic programs, by using equivalences such as $(\rho; \chi)^- \equiv (\chi^-; \rho^-)$ or $(\rho^*)^- \equiv (\rho^-)^*$. The metavariable A denotes either a direct or a converse atomic program (considering $(a^-)^- \doteq a$) and we refer to it as an *atomic program*, without specifying direct or converse. In the following Φ is the formula to be proved valid or satisfiable.

The semantics is based on Kripke structures [15]: a model is a pair $\langle W, \mathscr{I} \rangle$ where W is a nonempty set of states and $\mathscr{I}$ an interpretation s.t. for every direct atomic program a it is $a^{\mathscr{I}} \subseteq W \times W$ and for every propositional letter P it is $P^{\mathscr{I}} \subseteq W$. The interpretation $\mathscr{I}$ is then extended as follows:

$$(\psi \wedge \varphi)^{\mathscr{I}} = \varphi^{\mathscr{I}} \cap \psi^{\mathscr{I}} \qquad (\neg\varphi)^{\mathscr{I}} = W - \varphi^{\mathscr{I}}$$

$$(\langle \rho \rangle \varphi)^{\mathscr{I}} = \{w \mid \exists v \in W \text{ s.t. } \langle w, v \rangle \in \rho^{\mathscr{I}} \text{ and } v \in \varphi^{\mathscr{I}}\}$$

$$(\rho; \chi)^{\mathscr{I}} = \{\langle w, v \rangle \mid \exists u \langle w, u \rangle \in \rho^{\mathscr{I}} \text{ and } \langle u, v \rangle \in \chi^{\mathscr{I}}\}$$

[6] Remember that in this case we are trying to prove that an eventuality has not been fulfilled and never will be (no matter how long we continue the computation).

$$(\rho \cup \chi)^{\mathscr{I}} = \rho^{\mathscr{I}} \cup \chi^{\mathscr{I}}$$
$$(\rho^*)^{\mathscr{I}} = \text{reflexive and transitive closure of } \rho^{\mathscr{I}}$$
$$(\rho^-)^{\mathscr{I}} = \{\langle v, w\rangle \mid \langle w, v\rangle \in \rho^{\mathscr{I}}\}$$
$$(\varphi?)^{\mathscr{I}} = \{\langle w, w\rangle \mid s \in \varphi^{\mathscr{I}}\}.$$

In the following we write $w \models \varphi$ for $w \in \varphi^{\mathscr{I}}$.

DEFINITION 1. A CPDL formula Φ is *satisfiable* iff there is a model $\langle W, \mathscr{I}\rangle$ where $(\Phi)^{\mathscr{I}}$ is not empty. A formula Φ is *valid* if for every model $\langle W, \mathscr{I}\rangle$ it is $(\Phi)^{\mathscr{I}} = W$.

The *Fisher–Ladner closure* of a formula Φ [10, 15] is defined inductively as:

— $\Phi \in CL(\Phi)$;
— if $\varphi \in CL(\Phi)$ then $\neg\varphi \in CL(\Phi)$, provided φ does not start with $\neg$;
— if $\neg\varphi$, $\varphi \wedge \psi$ or $\langle \rho\rangle \varphi$ are in $CL(\Phi)$ then $\varphi, \psi \in CL(\Phi)$;
— if $\langle \rho; \chi\rangle \varphi \in CL(\Phi)$ then $\langle \rho\rangle\langle \chi\rangle \varphi \in CL(\Phi)$;
— if $\langle \rho \cup \chi\rangle \varphi \in CL(\Phi)$ then both $\langle \rho\rangle \varphi$ and $\langle \chi\rangle \varphi$ are in $CL(\Phi)$;
— if $\langle \psi?\rangle \varphi \in CL(\Phi)$ then $\psi \in CL(\Phi)$;
— if $\langle \rho^*\rangle \varphi \in CL(\Phi)$ then $\langle \rho\rangle\langle \rho^*\rangle \varphi \in CL(\Phi)$.

To establish the truth value of a formula Φ in a model it is sufficient to check the value of the formulae in $CL(\Phi)$ for every state of the model [10, 15]. Both number and size of the formulae in $CL(\Phi)$ are linearly bounded by the size of Φ.

4. PREFIXED TABLEAUX FOR CPDL

Prefixed tableaux are based on *prefixed formulae*, i.e. pairs $\langle \sigma : \varphi\rangle$ where φ is a CPDL formula and σ is an alternating sequence of integers and atomic programs called *prefix* defined as $\sigma ::= 1 \mid \sigma.a.n \mid \sigma.a^-.n$.

Intuitively σ denote the *sequence of two-way transitions* (in the sense of [27]) of states and atomic programs A that starts from the initial state 1 and reaches the state where φ holds. For instance the prefix $1.a^-.2.b.3.a.5.c^-.4$ corresponds to $w_1 \overset{a}{\leftarrow} w_2 \overset{b}{\rightarrow} w_3 \overset{a}{\rightarrow} w_5 \overset{c}{\leftarrow} w_4$. PDL has only one-way transitions.

We use the standard initial subsequence ordering $\sqsubseteq$—i.e., impose $\sigma \sqsubset \sigma.A.n$ for every σ, A, and n, and take the reflexive–transitive closure.

The definition of branch and tableau is standard [11, 13, 18]. A *tableau* $\mathscr{T}$ is a rooted tree where nodes are labeled with formulae, a *branch* $\mathscr{B}$ is a path from the root to a leaf, and a *segment* $\mathscr{S}$ is a path from the root to a node of the tree. If a segment terminates into a branching due to a disjunctive rule $(\beta, [test], \langle choice\rangle, X, LB(A))$ we denote the left-hand extension of $\mathscr{S}$ (the left "branch" par abuse de language) with $\mathscr{S}_l$ and the right-hand extension with $\mathscr{S}_r$. Intuitively a branch is a (tentative) model for the initial formula.

A prefix is *present* in a segment if there is a prefixed formula with that prefix already in the segment, and it is *new* if it is not already present. In the following

$$\alpha : \frac{\sigma : \varphi \wedge \psi}{\begin{array}{c}\sigma : \varphi \\ \sigma : \psi\end{array}} \qquad \beta : \frac{\sigma : \neg(\varphi \wedge \psi)}{\sigma : \neg\varphi \mid \sigma : \neg\psi} \qquad dneg : \frac{\sigma : \neg\neg\varphi}{\sigma : \varphi}$$

$$[seq] : \frac{\sigma : \neg\langle\chi;\rho\rangle\varphi}{\sigma : \neg\langle\chi\rangle\langle\rho\rangle\varphi} \qquad \langle seq\rangle : \frac{\sigma : \langle\chi;\rho\rangle\varphi}{\sigma : \langle\chi\rangle\langle\rho\rangle\varphi}$$

$$[test] : \frac{\sigma : \neg\langle\psi?\rangle\varphi}{\sigma : \neg\psi \mid \sigma : \neg\varphi} \qquad \langle test\rangle : \frac{\sigma : \langle\psi?\rangle\varphi}{\begin{array}{c}\sigma : \psi \\ \sigma : \varphi\end{array}}$$

$$[choice] : \frac{\sigma : \neg\langle\chi \cup \rho\rangle\varphi}{\begin{array}{c}\sigma : \neg\langle\chi\rangle\varphi \\ \sigma : \neg\langle\rho\rangle\varphi\end{array}} \qquad \langle choice\rangle : \frac{\sigma : \langle\chi \cup \rho\rangle\varphi}{\sigma : \langle\chi\rangle\varphi \mid \sigma : \langle\rho\rangle\varphi}$$

FIG. 1. Propositional and program tableau rules.

we say that the pair $\langle \sigma_0, \mathscr{S}_0 \rangle$ is *shorter* than $\langle \sigma, \mathscr{S} \rangle$ if either (i) σ_0 is a proper initial subsequence of σ and $\mathscr{S}_0$ is an initial subsegment of $\mathscr{S}$ or (ii) $\sigma = \sigma_0$ and $\mathscr{S}_0$ is a proper initial subsegment of $\mathscr{S}$.

Figure 1 shows the rules for and, not, sequence, choice and test.

The rules for CPDL formulae starting with an atomic program A are shown in Fig. 2 and are the usual π-rule for $\langle A \rangle \varphi$-formulae and the ν-rule for $[A]\varphi$-formulae. The converse requires a rule which is analogous of the B rule for symmetric modal logics [18]: we use *both forward* (F) *and backward* (B) *rules* for necessities.

The rules for iteration combines prefixed tableaux with the ideas of [25, 24] for model checking fixpoints in the modal μ-calculus[7] (Fig. 3). In practice when an iterated eventuality $\langle \rho^* \rangle \varphi$ is found, we introduce a *new* propositional variable X (possibly with indices), set a side condition $X \doteq \langle \rho^* \rangle \varphi$, and use the X-rule for further reductions. The set $\mathscr{X}$ of propositional variables introduced in this way is distinct from the set $\mathscr{P}$ of propositional letters which are used for formulae.

The use of $\sigma : \neg\varphi$ in the right part of the X rule is motivated by the definition of $\langle \rho^* \rangle \varphi$ as a *least* fixpoint: ρ-steps are performed while $\neg\varphi$ is true, stopping as soon as φ becomes true. Indeed $\langle \rho^* \rangle \varphi \equiv \langle (\neg\varphi?; \rho)^* \rangle \varphi \equiv \langle$`while` $\neg\varphi$ `do` $\rho \rangle \top$ is valid in CPDL [10].

As mentioned in Section 2, these variables are introduced to detect the presence of ρ loops which do not fulfill $\langle \rho^* \rangle \varphi$. Using such variables, we can eliminate the $\Rightarrow$ (and its transitive closure) introduced by Pratt [22].

Remark 2. The combination of converse $\cdot^-$ with iteration $\cdot^*$ is harder than both of them in isolation: although $[A]^B$ is enough for CPDL without iteration and X rules are enough for PDL, their combination is not enough for full CPDL.

Iteration $\cdot^*$ constructs properties with *unbounded delays*, such as $\langle a^* \rangle P$, while converse $\cdot^-$ can be used for *late discoveries*, such as $\langle a \rangle [a^-] \neg P$, which impose a property on the current state *after* the execution of a program. Their combination

[7] A formula $\langle \rho^* \rangle \varphi$ can be expressed in the modal μ-calculus as $\mu X.\varphi \vee \langle \rho \rangle X$ where $\mu X.\Psi(X)$ denote the least fixpoint of the open formula $\Psi(X)$.

$$\langle A\rangle : \quad \frac{\sigma : \langle A\rangle\varphi}{\sigma.A.n : \varphi} \quad \text{with } \sigma.A.n \text{ new in the branch}$$

$$[A]^F : \quad \frac{\sigma : \neg\langle A\rangle\varphi}{\sigma.A.n : \neg\varphi} \quad \text{with } \sigma.A.n \text{ already present in the branch}$$

$$[A]^B : \quad \frac{\sigma.A^-.n : \neg\langle A\rangle\varphi}{\sigma : \neg\varphi} \quad \text{with } \sigma \text{ already present in the branch}$$

FIG. 2. Transitional rules for CPDL.

creates *bombs*, such as $P \wedge \langle a^*\rangle[(a^-)^*]\neg P$, which after an unbounded number of iterations, make the initial state inconsistent.

So, we use a restricted *analytic cut* LB (look behind), presented in Fig. 4, where Φ is the formula to be proved valid or satisfiable.

Since the cut is analytic and its application strongly restricted, its introduction does not destroy the decidability of the calculus (although its unnecessary application may choke the proof search).

In the following, if $\mathscr{S}$ is a segment (possibly a branch) of a tableau we indicate with $\mathscr{S}/\sigma$ the set of prefixed formulae in $\mathscr{S}$ labeled with the prefix σ; i.e.,

$$\mathscr{S}/\sigma = \{\varphi \mid \langle \sigma : \varphi\rangle \in \mathscr{S}\}.$$

DEFINITION 3. A prefix σ is *reduced* in $\mathscr{S}$ if $\langle A\rangle$-rules are the only rules not yet applied to formulae of $\mathscr{S}/\sigma$. It is *fully reduced* if all rules have been applied.

Two prefixes are "a different name for the same state" in CPDL (where the past matters) if they (i) have the same properties (dynamic formulae) and (ii) are reachable by the same atomic program. An identical requirement is used for automata [27]. Point (ii) is not necessary for PDL.

DEFINITION 4. A prefix σ in the segment $\mathscr{S}$ is a *copy* of a prefix σ_0 in $\mathscr{S}_0$ if (i) $\mathscr{S}/\sigma = \mathscr{S}_0/\sigma_0$, and (ii) both have the form $\sigma'.A.n$ and $\sigma'_0.A.m$ for the same atomic program A. If distinct X_i, X_j are present in σ_0 and σ we consider them equal if they represent the same iterated eventuality, i.e., $X_i \doteq \langle \rho^*\rangle \varphi \doteq X_j$.

DEFINITION 5. A branch $\mathscr{B}$ is *π-completed* if (i) all prefixes are reduced, and (ii) for every σ which is not fully reduced there is a pair $\langle \sigma_0, \mathscr{S}_0\rangle$ shorter than $\langle \sigma, \mathscr{B}\rangle$ s.t. σ_0 is fully reduced in the segment $\mathscr{S}_0$ and σ is a copy of σ_0 in $\mathscr{B}$.

$$[*] : \frac{\sigma : \neg\langle\rho^*\rangle\varphi}{\begin{array}{l}\sigma : \neg\varphi \\ \sigma : \neg\langle\rho\rangle\langle\rho^*\rangle\varphi\end{array}} \qquad \langle * \rangle : \frac{\sigma : \langle\rho^*\rangle\varphi}{\begin{array}{l}\sigma : X \\ X \doteq \langle\rho^*\rangle\varphi\end{array}} \; X \text{ new on the branch}$$

$$X : \frac{\sigma : X}{\begin{array}{l}\sigma : \varphi \mid \sigma : \neg\varphi \\ \mid \sigma : \langle\rho\rangle X\end{array}}$$

FIG. 3. Rules for $*$-iteration operator.

$$LB(A): \quad \frac{\vdots}{\sigma.A.n : \langle A^- \rangle \varphi \mid \sigma.A.n : \neg\langle A^- \rangle \varphi}$$

$\sigma.A.n$ is already present and φ is a formula of $CL(\Phi)$

FIG. 4. Look behind analytic cut.

If $\sigma : X$ has been considered equal to a $\sigma_0 : X_0$ we say that X *collapses* into X_0. By π-completenes, collapsed X *cannot* generate new $\sigma' : X$ with longer σ'. The intuition for π-completeness is that $\langle A \rangle$-rules are not applied to formulae belonging to copies.

DEFINITION 6. An eventuality $X \doteq \langle \rho^* \rangle \varphi$ is *fulfilled* in segment $\mathscr{S}$ iff there is either a $\sigma : X$ in $\mathscr{S}$ such that $\sigma : \varphi$ is also in $\mathscr{S}$ or X has been collapsed into an X_0 which is fulfilled.

DEFINITION 7. A branch $\mathscr{B}$ is *contradictory* iff it contains both $\sigma : P$ and $\sigma : \neg P$, for some P and some σ. A π-completed branch $\mathscr{B}$ is *ignorable* iff there is an eventuality X which is not fulfilled.

In a π-completed branch $\mathscr{B}$, an eventuality X cannot collapse (directly or transitively) into itself and be fulfilled at the same time, since we have either $\sigma : \varphi$ or $\sigma : \neg\varphi$ in the X-rule. Indeed suppose we tried to collapse an X from $\langle \sigma, \mathscr{B} \rangle$ into a shorter $\langle \sigma_0, \mathscr{S}_0 \rangle$ also with $\sigma_0 : X$. Obviously σ_0 cannot fulfill X, otherwise the branch would only have had $\sigma_0 : \varphi$ but no other X and therefore a longer $\langle \sigma, \mathscr{B} \rangle$ with X could not have been generated at all. On the contrary the right-hand "branch" $\mathscr{S}_{0r}$ has another instance of X, i.e., $\sigma_0 : \langle \rho \rangle X$ but also $\sigma_0 : \neg\varphi$. It can generate the longer $\langle \sigma, \mathscr{B} \rangle$ by reducing $\langle \rho \rangle X$. However if σ must be a copy of σ_0 this means that also σ: $\neg\varphi$ must be there and σ would not fulfill X.

In a nutshell we found an unsuccessful loop [22]: we try to fulfill an eventuality $\langle \rho^* \rangle \varphi$ with X-rules; the left-hand "branches" with $\sigma : \varphi$ are always discarded; finally we meet a prefix σ_ω with the same formulae of an already seen prefix; so we conclude that we could never fulfill it and give up. We can express this fact with the following sufficient condition to close ignorable branches:

PROPOSITION 8. *A branch $\mathscr{B}$ is ignorable if there is an X which collapses* (*directly or transitively*) *into itself provided the corresponding shorter $\langle \sigma_0, \mathscr{S}_0 \rangle$ is fully reduced.*

Fully reduced means also that cut has been applied in all possible ways and this is the only place where we need it: we must be sure that two prefixes are identical also with respect to the past, at least w.r.t. the Fisher–Ladner closure.

Criterion 1. Apply $LB(A)$ only if the prefixed formulae $\sigma.A.n : X$ and $\sigma.A.n : \neg\varphi$ occurs in the branch and we are trying to collapse X into an X_0 (also itself).

DEFINITION 9. A tableau is *closed* if all branches are either contradictory or ignorable. A tableau is *open* if at least one branch is open (π-completed and neither contradictory nor ignorable).

DEFINITION 10. A *tableau validity proof* for the formula Φ in the logic CPDL is the closed tableau starting with $\langle 1 : \neg\Phi \rangle$.

For satisfiable formulae a model can be built from an open branch, as is done in the completeness proof.

5. EXAMPLES AND QUESTIONS

We assume that we have rules for the necessity operator which can be easily derived from the corresponding [*rule*] described in Section 4.

A simple example where converse forces us to "move" back and forth among states is given in Fig. 5. An example of an ignorable branch is given in Fig. 6.

There are also interesting questions which can be better clarified by examples.

QUESTION 11. *Why X different are introduced each time the same* $\langle \rho^* \rangle \varphi$ *is met with a different prefix if later on we identify them in the loop checking*?

The propositional variables $X \doteq \langle \rho^* \rangle \varphi$ are an automatic bookkeeping system: if we introduce $\sigma_0 : X$ at a certain stage and, later on, we find a longer σ with $\sigma : X$, then we can infer, *without further checks* that there are some $\langle \rho \rangle$-steps from σ_0 to σ, no matter how complicated ρ is. Hence, if $\sigma : \varphi$ is present in the branch, we can conclude that the initial occurrence of $\langle \rho^* \rangle \varphi$ and all subsequent occurrences (represented by X) are fulfilled.

If we reused the same X for a different prefix $\sigma_1 : \langle \rho^* \rangle \varphi$ in the branch, then we could not know anymore whether σ follows from σ_0 or from σ_1. We would have to wait for the completion of the (pseudo)model and then model check it, which is exactly what we are trying to avoid.

Yet X_i are just names for the corresponding $\langle \rho^* \rangle \varphi$ and we want to fulfill the formula and not a name. If we find out that $X_i \doteq \langle \rho^* \rangle \varphi \doteq X_j$ this means that they

(1) $1 : \neg P \wedge \langle a \rangle \left[a^-\right] \langle b^- \rangle Q \wedge \left[b^-; b^*\right] P$ initial formula
(2) $1 : \neg P$ ..... $\alpha -$ rule to (1)
(3) $1 : \langle a \rangle \left[a^-\right] \langle b^- \rangle Q$
(4) $1 : \left[b^-; b^*\right] P$
(5) $1 : \left[b^-\right] [b^*] P$ ..... $[seq] -$ rule to (4)
(6) $1.a.2 : \left[a^-\right] \langle b^- \rangle Q$ ..... $\langle a \rangle -$ rule to (3)
(7) $1 : \langle b^- \rangle Q$ ..... $\left[a^-\right]^B -$ rule to (6)
(8) $1.b^-.3 : Q$ ..... $\langle b^- \rangle -$ rule to (7)
(9) $1.b^-.3 : [b^*] P$ ..... $\left[b^-\right]^F -$ rule to (5)
(10) $1.b^-.3 : P$ ..... $[*] -$ rule to (9)
(11) $1.b^-.3 : [b][b^*] P$
(12) $1 : [b^*] P$ ..... $[b]^B -$ rule to (11)
(13) $1 : P$ ..... $[*] -$ rule to (12)
(14) $1 : [b][b^*] P$
(15) $\perp -$ contradiction in 1

FIG. 5. Tableau proof.

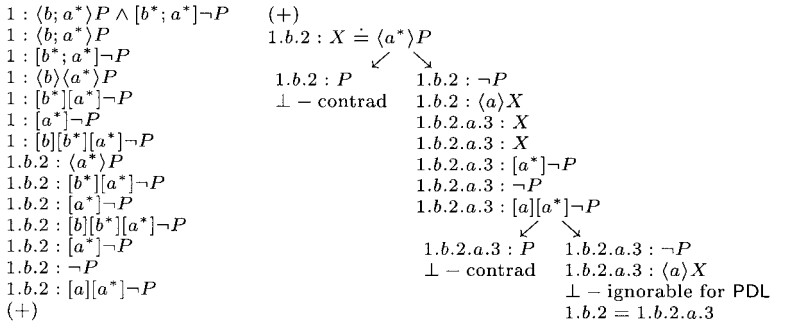


FIG. 6. Another tableau proof.

are just different names for the same property: if a state σ_0 fulfills the same formulae of σ plus X_i then it clearly fulfills the X_j occurring in σ and thus we can identify the two formulae and hence the states.

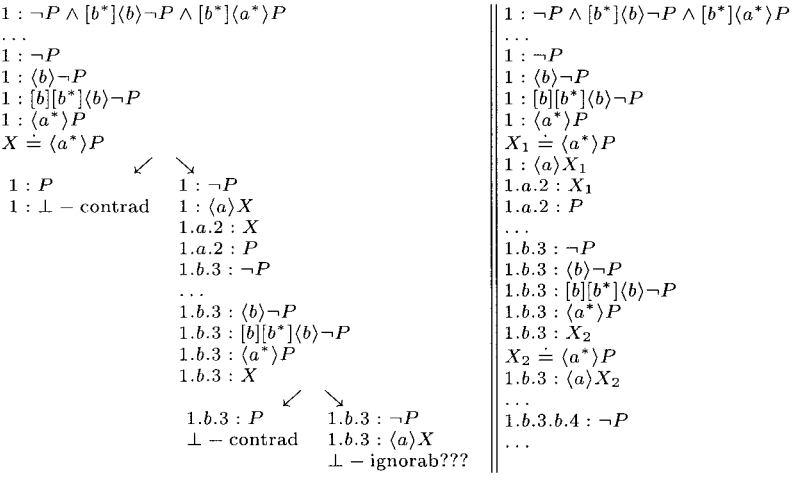
For instance try the following (without $LB(A)$, since there is no converse):

$$\Phi_{SAT} \doteq P \wedge [b^*](\langle b \rangle P \wedge \langle a^* \rangle \neg P) \qquad \Phi_{UNSAT} \doteq \Phi_{SAT} \wedge [b^*; a^*]P.$$

The consequences of the wrong usage of X are shown in Fig. 7 for Φ_{SAT}.

QUESTION 12. *Is cut really necessary*?

Cut is eliminable if formulae do not contain both converse and iteration operators since the soundness proof does not require its use. The difficult formula is $[(\rho^-)^*]$ which imposes unbounded constraints on the past. Check the following formulae without cut and with a (modified) X without the $\sigma : \neg\varphi$ on the right.



The left tableaux is wrongly closed by the ignorability condition because the same X is recycled. The right one never terminates because different X_i for the same eventualities are not identified.

FIG. 7. Erroneous tableaux proofs.

$$\Psi_{UNSAT} \doteq P \wedge \langle a^* \rangle [(a^-)^*] \neg P$$

$$\Psi_{SAT} \doteq P \wedge \langle a^* \rangle \left(\neg P \wedge \bigwedge_{i=1}^{n} [a^{-i}] \neg P \right),$$

where we abbreviate $a^-; ...; a^-$ for i times with a^{-i}. The second formula is satisfiable, while the first is not. Without cut and without the $\neg\varphi$-branch in the X rule, after n applications of the X rule both formulae will have only contradictory or ignorable branches. After $n+1$ steps, the tableau for Ψ_{SAT} has one nonignorable branch whereas the one for Ψ_{UNSAT} remains ignorable.

6. SOUNDNESS AND COMPLETENESS

The intuitions behind soundness and completeness can be explained by examining the concepts of copied prefixes and ignorable branches from a model theoretic perspective, in particular by comparing tableau rules to visiting steps of a model and prefixes to booking devices (names for states).

Whenever we find two prefixes σ_0 and σ_ω which have the same properties we may conclude that they are essentially identical (model $\mathcal{M}$ in Fig. 8). Thus, there is no need to expand the formulae of σ_ω: we have already done it for σ_0 (by Definition 5) and if we did not find a contradiction before we will not find it now. We can avoid the visit of the infinite path from σ_ω by changing the model. If a branch is open then we introduce a loop back to σ_0, thus dropping the in-principle-infinite path starting from σ_ω (model $\mathcal{M}_{\text{good}}$ in Fig. 8). This is the completeness theorem. When a branch is ignorable there is an eventuality $\langle \rho^* \rangle \phi$ on σ that, after some $\langle \rho \rangle$-steps where $\neg\phi$ always holds, arrives to an "identical" state σ_ω. So we change the model to $\mathcal{M}_{\text{bad}}$ (Fig. 8) and conclude that we cannot fulfill the eventuality in any number of ρ-steps. This is the soundness theorem.

In the tableaux of [20, 22] they were called successful and unsuccessful loops.

The formal soundness proof follows an established path [11, 18]:

1. devise a mapping between prefixes in a tableau and states in a model;
2. prove a safe extension lemma (any tableaux rule applied to a satisfiable formula preserves satisfiability with this mapping);
3. prove a safe closure lemma (no satisfiable branch is ignored).

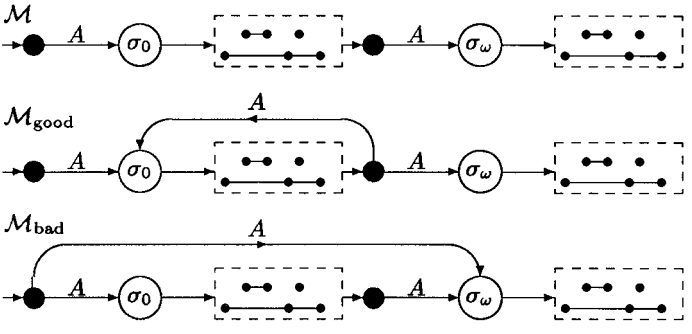


FIG. 8. Bad and good models.

Remark 13. For modal logics safe closure is immediate (a branch must only be noncontradictory), whereas it is the hardest part for PDL/CPDL: we have to verify, with a *finite* computation, that an eventuality will *never* be fulfilled.

DEFINITION 14. Let $\mathscr{B}$ be a branch and $\langle W, \mathscr{I} \rangle$ a model, a *mapping* is a function $\iota(\ )$ from prefixes to states such that for all σ and $\sigma.A.n$ present in $\mathscr{B}$ it is $\langle \iota(\sigma), \iota(\sigma.A.n) \rangle \in A^{\mathscr{I}}$.

DEFINITION 15. A tableaux branch $\mathscr{B}$ is *satisfiable* (SAT for short) in the model $\langle W, \mathscr{I} \rangle$ if there is a mapping $\iota(\ )$ such that for every $\langle \sigma : \varphi \rangle$ present in $\mathscr{B}$ it is $\iota(\sigma) \models \varphi$. A tableau is SAT if one branch is SAT in some model $\langle W, \mathscr{I} \rangle$.

THEOREM 16. *If $\mathscr{T}$ is a SAT tableau, then the tableau $\mathscr{T}'$ obtained by an application of a tableau rule is also SAT.*

Proof. The proof is by induction on the applied rules as in [11, 13, 18]. ∎

The difficult part is proving that ignorable branches can be discarded. We need a preliminary result telling that either (i) if an eventuality X is not collapsed then it terminates into a fulfilling $\sigma : \varphi$, i.e., no more X are generated, or (ii) if we follow the path from an eventuality that is collapsed into itself (directly or indirectly) then we always meet $\langle \sigma : X \rangle$ with $\langle \sigma : \neg\varphi \rangle$.

To this extent we introduce the following relation $\prec$ between prefixed formulae for each $X \doteq \langle \rho^* \rangle \varphi$ in a branch $\mathscr{B}$:

— $\langle \sigma_0 : X \rangle \prec \langle \sigma : X \rangle$ iff $\sigma_0 : X$ appears in the segment $\mathscr{S}_0$, $\sigma : X$ properly[8] appears in $\mathscr{S}$, and $\langle \sigma_0, \mathscr{S}_0 \rangle$ is shorter than $\langle \sigma, \mathscr{S} \rangle$;

— $\langle \sigma : X \rangle \prec \langle \sigma_0 : X_0 \rangle$ iff X has been collapsed in X_0 and σ_0 is the fully reduced copy of σ.

The transitive (but not reflexive) closure of $\prec$ is $\prec\!\prec$. We need to use segments to take tests into accounts. In a nutshell *we are reconstructing* (*only for the proof and only for iterated eventualities*) *the* $\Rightarrow$ *relation of Pratt* [22].

Then we can prove the following result whose intuition is that either an eventuality is fulfilled (and $\prec$ forms a well-ordering with maximum) or after a certain stage all X fall into a final cluster (and $\prec$ is not well-founded).

LEMMA 17. *Let $\mathscr{B}$ be a π-completed branch, $\sigma : X$ be a prefixed eventuality, with $X \doteq \langle \rho^* \rangle \varphi$, and* $\mathtt{path}(\sigma : X)$ *be the set of $\langle \sigma_i : X_i \rangle$ present in $\mathscr{B}$, such that $\langle \sigma : X \rangle \prec\!\prec \langle \sigma_i : X_i \rangle$. Then there is a $\langle \sigma_\omega, X_\omega \rangle$ in* $\mathtt{path}(\sigma : X)$ *such that*:

— *either $\langle \sigma_\omega : \varphi \rangle$ is in $\mathscr{B}$ and then* $\mathtt{path}(\sigma : X)$ *is well-ordered by $\prec$ and $\langle \sigma_\omega, X_\omega \rangle$ is the maximum element.*

— *or $\langle \sigma_\omega : X_\omega \rangle \prec\!\prec \langle \sigma_\omega : X_\omega \rangle$ and then for all $\langle \sigma_i : X_i \rangle \in$* $\mathtt{path}(\sigma : X)$ *the prefixed formula $\langle \sigma_i : \neg\varphi \rangle$ is in $\mathscr{B}$ and $\langle \sigma_i : X_i \rangle \prec\!\prec \langle \sigma_\omega : X_\omega \rangle$.*

[8] When $\sigma = \sigma_0$ this means that $\sigma_0 : X$ must have been introduced again in the part of $\mathscr{S}$ properly extending $\mathscr{S}_0$. Otherwise we would always have $\sigma_0 : X \prec \sigma_0 : X$.

Again the proof is by induction on the applied rules using the argument for Proposition 8 and the fact that a collapsed eventuality is not further reduced. Notice that the assignment of a different X for each occurrence of the eventuality $\langle \rho^* \rangle \varphi$ with a different prefix is necessary for the proof. With this provision the only way to obtain some $\sigma' : X$ is by reducing some previously generated $\sigma : \langle \rho \rangle X$.

THEOREM 18. *If $\mathscr{T}$ is a SAT tableau, then one SAT branch is not ignorable.*

Proof. Suppose the contrary: $\mathscr{T}$ is SAT with all SAT branches ignorable (clearly SAT branches cannot be contradictory). It is worth noting that each branch can be ignorable due to a different unfulfilled $X \doteq \langle \rho^* \rangle \varphi$ (or even more than one). It is easy to prove the following proposition.

PROPOSITION 19. *Let $\mathscr{B}$ be an ignorable branch for some $\sigma : X$. For every model $\langle W, \mathscr{I} \rangle$ and for every mapping $\iota(\ )$ such that $\mathscr{B}$ is SAT for them, if $\langle \sigma_i, X_i \rangle$ is in* $\mathtt{path}(\sigma : X)$ *then $\iota(\sigma_i) \not\models \varphi$.*

This is immediate because of the presence of $\neg\varphi$ on the right-hand branch of the X-rule: it boils down to the definition of SAT branches.

Since $\mathscr{B}$ is SAT, there is a model $\langle W, \mathscr{I} \rangle$ and a mapping $\iota(\ )$ on which $\mathscr{B}$ is SAT with a certain mapping $\iota(\ )$. However, since $\mathscr{B}$ is ignorable, by Lemma 17, we can consider the final cluster generated by any prefixed formula $\sigma : X$ for the X which is unfulfilled.

We can show that there are at least two ρ-connected prefixes in the final cluster on $\mathtt{path}(\sigma : X)$. Indeed the only way to make a loop, is to have some X_j along the path collapsing into an X_i (maybe itself) and therefore the corresponding prefixes must be different. Let us denote the fully reduced prefix in the loop by $\sigma_0 \equiv \sigma'_0.A.n_0$ and the copy with $\sigma_\omega \equiv \sigma'_\omega.A.n_\omega$.

Since $\mathscr{B}$ is SAT, $\iota(\sigma_0) \models \langle \rho \rangle X$, and therefore an integer N and $N+1$ states $w_0, ..., w_N$ in $\langle W, \mathscr{I} \rangle$ exist such that $\iota(\sigma_0) = w_0$, $\langle w_i, w_{i+1} \rangle \in \rho^{\mathscr{I}}$, and $w_N \models \varphi$.

Since the prefix σ_0 has been fully reduced $\mathscr{B}$ must also contain $\langle \sigma_0 : \langle \rho \rangle X \rangle$ and this must be reduced. Thus the only way to generate the next $\langle \sigma_1 : X \rangle \succ \langle \sigma_0 : X \rangle$ to go to σ_ω is to perform at least one ρ-step. Hence, in general, there are R ρ-steps from σ_0 to σ_1 to σ_2 ... to σ_ω for some integer $R \geqslant 1$

Let first assume that $N \leqslant R$. Then we can remap the prefixes $\sigma_0, ..., \sigma_N$ so that $\iota(\sigma_i) = w_i$. By Proposition 19 the formula φ cannot be fulfilled by any mapping $\iota(\ )$ of the σ_i in $\mathtt{path}(\sigma : X)$ on the states of $\langle W, \mathscr{I} \rangle$. Hence we get a contradiction.

Let assume that $N > R$. Then there must be $N - R$ ρ-steps from σ_ω to fulfill φ in the model $\langle W, \mathscr{I} \rangle$ under $\iota(\ )$.

Now we construct a new model $\langle W, \mathscr{J} \rangle$ as in Fig. 8 by copying the original model: $W' = \{w_c \mid w \in W\}$, $P^{\mathscr{J}} = \{w_c \mid w \in P^{\mathscr{I}}\}$ and $a^{\mathscr{J}} = \{\langle w_c, v_c \rangle \mid \langle w, v \rangle \in a^{\mathscr{I}}\}$. Then we add a new A-arc, where A is the atomic (direct or converse) program occurring last in σ_0 and σ_ω, i.e.:

$$A^{\mathscr{J}} = \{\langle w_c, v_c \rangle \mid \langle w, v \rangle \in A^{\mathscr{I}}\} \cup \{\langle \iota(\sigma_0)_c, \iota(\sigma_\omega)_c \rangle\}.$$

The key point is to prove that this new A-arc can be safely added.

Since $\mathscr{B}$ is π-completed, all possible instances of $LB(A)$ have been applied and therefore for every $\psi \in CL(\Phi)$ either $\langle \sigma_0 : \neg\langle A^- \rangle \psi \rangle$ or $\langle \sigma_0 : \langle A^- \rangle \psi \rangle$ is present on the branch. The prefix σ_ω is a copy of σ_0 by hypothesis, so $\langle \sigma_\omega : \neg\langle A^- \rangle \psi \rangle$ is present in the branch iff $\langle \sigma_0 : \neg\langle A^- \rangle \psi \rangle$ is present. Since the branch is SAT on the original model $\langle W, \mathscr{I} \rangle$, it is $\iota(\sigma_0) \models \neg\langle A^- \rangle) \psi$ iff $\iota(\sigma_\omega) \models \neg\langle A^- \rangle \psi$ for every $\psi \in CL(\Phi)$.

Consider now the state $\iota(\sigma_\omega)_c$. The only difference with the original state $\iota(\sigma_\omega)$ is the incoming A-arc. But, as we have seen above, the two states see exactly the same formulae of $CL(\Phi)$ going back through A. By the filtration lemma [10, 15], these are the only formulae necessary for establishing the truth value of Φ. Hence, by induction, we have that $\iota(\sigma_\omega)_c$ satisfies $\langle \rho^* \rangle \varphi$ in $N - R$ ρ-steps in the new model (as in the old one).

Then we construct a new mapping $\jmath(\ )$ on the new model as follows: map every prefix shorter than or unrelated to σ_0 in the same way as $\iota(\ )$ does, and map $\jmath(\sigma_0)$ on $\iota(\sigma_\omega)_c$. This make the branch still satisfiable: the formulae are the same for both σ_0 and σ_ω and the incoming arc does not affect them. By Theorem 16 we can expand the tableau and still preserve SAT.

In the new model the state $\jmath(\sigma_0)$ fulfills the eventuality $\langle \rho^* \rangle \varphi$ in $N - R < N$ ρ-steps. We can repeat the process until we reach an $N' \leqslant R$, getting again a contradiction.

The *correctness theorem* follows with a standard argument [11]:

THEOREM 20. *If Φ has a validity proof then Φ is valid for* CPDL.

For completeness, we also have an established path [11, 13, 18]: apply a systematic and fair procedure to the tableau and if it does not close, choose an open branch to build a model for the initial formula $\neg\Phi$, i.e., a counter-model for Φ. The key is the following *model existence theorem*.

THEOREM 21. *If $\mathscr{B}$ is an open branch then there is $\langle W, \mathscr{I} \rangle$ where it is SAT.*

Proof. Construct the model as follows:

$$
\begin{aligned}
W &\doteq \{\sigma \mid \sigma \text{ is present in } \mathscr{B}\} \\
a^{\mathscr{I}} &\doteq \{\langle \sigma, \sigma.a.n \rangle \mid \sigma \text{ and } \sigma.a.n \text{ are present in } \mathscr{B}\} \\
&\quad \cup \{\langle \sigma.a^-.n, \sigma \rangle \mid \sigma \text{ and } \sigma.a^-.n \text{ are present in } \mathscr{B}\} \\
P^{\mathscr{I}} &\doteq \{\sigma \mid \sigma : P \in \mathscr{B}\}.
\end{aligned}
$$

To take loops and repetitions into account, we modify slightly the above definition: if σ is a copy of some fully reduced prefix σ_0 then we delete σ from W, replace σ with σ_0 in all transitions $a^{\mathscr{I}}$, and construct the mapping $\iota(\ )$.

$$
\iota(\sigma) = \begin{cases} \sigma_0 & \text{if } \sigma \text{ is a copy of a fully reduced } \sigma_0 \\ \sigma & \text{otherwise.} \end{cases}
$$

Next we prove that if $\langle \sigma : \varphi \rangle \in \mathscr{B}$ then $\iota(\sigma) \models \varphi$ by induction on the construction of φ. We focus on modal connectives and iteration operators.

Suppose that $\langle \sigma : \langle a \rangle \varphi \rangle \in \mathscr{B}$. If σ is fully reduced then we apply exactly the argument of [11, 13, 18]. If σ is a copy then the mapping $\iota(\ )$ maps σ on σ_0, which is fully reduced, and the above reasoning applies. Similarly for a^-.

For the necessity operator we show the case for a^-. Suppose that $\langle \sigma : \neg\langle a^- \rangle \varphi$ is in $\mathscr{B}$ and that σ is fully reduced (the argument for not fully reduced prefixes is analogous to the previous case). By construction the only possible pairs such that $\langle \sigma, \sigma' \rangle \in (a^-)^{\mathscr{I}}$ are:

1. the pairs $\langle \sigma, \sigma.a^-.n \rangle$ for some n;
2. the pair $\langle \sigma_0.a.n, \sigma_0 \rangle$ if σ has the form $\sigma_0.a.m$;
3. $\langle \sigma, \sigma'_0 \rangle$ where σ'_0 is fully reduced copy of a prefix σ' which satisfies one the two previous conditions.

For case (1) then for every $\sigma.a^-.n$ present in $\mathscr{B}$ it is $\langle \sigma.a^-.n : \neg\varphi \rangle \in \mathscr{B}$ by π-completion w.r.t. $[a^-]^F$. Hence, $\iota(\sigma.a^-.n) \models \neg\varphi$ by inductive hypothesis. For case (2) consider π-completion w.r.t. the rule $[a^-]^B$: the prefixed formula $\langle \sigma_0 : \neg\varphi \rangle$ occurs in $\mathscr{B}$. Case (3) is a repetition of the previous two since σ is fully reduced and the relevant rule to σ' has been applied already. Since σ'_0 is a copy then we have $\sigma'_0 : \neg\varphi$ and we are done. Thus, for all $\iota(\sigma')$ it is $\iota(\sigma') \models \neg\varphi$ by inductive hypothesis and therefore, by definition of $\models$, it is $\iota(\sigma) \models \neg\langle a^- \rangle \varphi$.

For $\langle \rho^* \rangle \varphi$ we have to prove that whenever $\sigma : X$ appears the corresponding $\langle \rho^* \rangle \varphi$ is satisfied. The proof is by double induction: on the formula size and on the number of $\prec$ steps in $\mathtt{path}(\sigma : X)$, using Lemma 17. One chooses as a base for the latter induction the top prefix σ_ω such that $\langle \sigma_\omega : \varphi \rangle$ is present. By induction hypothesis it is $\iota(\sigma_\omega) \models \varphi$ and by definition it is $\iota(\sigma_\omega) \models \langle \rho^* \rangle \varphi$. For the induction step consider a pair $\langle \sigma_j : X_j \rangle \prec \langle \sigma_{j+1} : X_{j+1} \rangle$ in $\mathtt{path}(\sigma : X)$ so $\iota(\sigma_{j+1}) \models \langle \rho^* \rangle \varphi$ by inductive hypothesis. If X_j collapsed into X_{j+i} then σ_j is a copy of σ_{j+1} and the result follows trivially: $\iota(\ )$ maps them on the same world. For the other case this means that $X_{j+i} \equiv X_j \equiv X$. Hence the only way to introduce $\sigma_{j+1} : X$ is to reduce completely $\sigma_j : \langle \rho \rangle X$. By induction on the construction of ρ (with a technique similar to those used in [6]) it is possible to verify that $\langle \sigma_j, \sigma_{j+1} \rangle$ is in $\rho^{\mathscr{I}}$ and therefore the claim follows by definition of $\models$. For instance if $\rho \equiv \chi;\ \tau$ then by π-completion $\sigma_j : \langle \chi \rangle\langle \tau \rangle X$ is on the branch and therefore there must be σ' such that $\langle \sigma_j, \sigma' \rangle \in \chi^{\mathscr{I}}$ and $\sigma' : \langle \tau \rangle X$ is present. This also yields $\langle \sigma', \sigma_{j+i} \rangle \in \tau^{\mathscr{I}}$ and the claim follows by induction. ▌

A *completeness theorem* follows with standard argument [11, 13, 18]

THEOREM 22. *If Φ is valid then Φ has a validity proof for CPDL.*

7. FROM NEXPTIME TABLEAUX TO EXPTIME ALGORITHMS

Tableaux lead to the following naive algorithm:

— select a formula from the branch and reduce it;

— if the reduction requires branching, then choose one branch and add the other to the stack;

— repeat until the branch is contradictory, ignorable, or open;

— in the first two cases discard the branch and backtrack.

This algorithm computes each time from scratch without keeping track of discarded branches; i.e., *it does not learn from failures*. This makes sense for logics in PSPACE [14] but not for PDLs. In fact the naive algorithm works in NEXPTIME, while CPDL is EXPTIME complete [10, 21].

A worst case EXPTIME algorithm can be developed as in [22] or [17, Sections 5.1–5.4]: use a suitable data structure where all possible subsets of the formulae that may appear in the tableau are listed. As soon as our expansion procedure introduces a new formula with a certain prefix, we collect the formulae with the same prefix and look in our database: if this set is already marked then we do not expand it further, otherwise we mark it. We use the pairs—set of formulae, arriving program—as elements of the database.

At the end, we start a marking algorithm which deletes bad pairs as in [22]. A key difference is that we discard at once all prefixes with an X making the branch ignorable. This is more effective than [22] also for PDL since we do not compute the transitive closure of $\Rightarrow$ but just look for Xs locally.

Marking each set with the "arriving programs" and "using cut" implies that, for each atomic program A, our database could contain all propositionally consistent subsets of $\{\psi, \langle A\rangle\psi, \neg\langle A\rangle\psi \mid \psi \in CL(\Phi)\}$. This gives an upper bound for the database size exponential in $O(|Act(\Phi)| \times |\Phi|^2)$, where $Act(\Phi)$ are the direct or converse atomic programs in Φ and hence the desired EXPTIME bound.

We can transform this algorithm into an *on-the-fly method* by creating the data structure dynamically as in [3, 17, Section 5.5] or in [11, 18] for modal logics rather than starting from an already fully developed one. Each time we find a new set of formulae $\mathscr{S}/\sigma$ we add the pair $\langle \mathscr{S}/\sigma, last(\sigma)\rangle$ to the database. Before expanding new formulae we check whether the corresponding pair is not already among the *visited* ones.

Again, the use of the bookkeeping system anticipates the closure of branches due to unfulfilled eventualities. By using X variables, the check for closure is substantially identical to the methods used for temporal logics where the duplication of an eventuality in a visited set is enough to close the branch.

Still the flavor of the algorithm is *breadth first*. Its transformation into a *depth-first* algorithm is similar to the techniques of [3]. We first present the algorithm for PDL to clarify the intuitions and afterward we extend it to CPDL.

For a segment $\mathscr{S}$ we say that $\mathscr{S}/\sigma.A.n$ is a *successor* [17] of $\mathscr{S}/\sigma$ if the prefix $\sigma.A.n$ has been introduced by an application of an $\langle A\rangle$-rule to a prefixed formula $\sigma : \langle A\rangle\varphi$ in $\mathscr{S}$. Next we can define a "bad" set as follows:

DEFINITION 23. Let $\mathscr{T}$ be a tableau, $\mathscr{S}$ a segment and σ a prefix occurring in $\mathscr{T}$, then the set $\mathscr{S}/\sigma$ is an *inconsistent set* ($\perp$-set) for $\mathscr{T}$ iff either

1. $\varphi, \neg\varphi \in \mathscr{S}/\sigma$ for some formula φ (contradiction);

2. $X, \neg\varphi \in \mathscr{S}/\sigma$ for some $X \doteq \langle \rho^* \rangle \varphi$ and $\mathscr{S}/\sigma$ is a copy of a fully reduced σ_0 for a $\langle \sigma_0, \mathscr{S}_0\rangle$ shorter than $\langle \sigma, \mathscr{S}\rangle$ (unfulfilled eventuality);

3. $S \subseteq \mathscr{S}/\sigma$ for some $\perp$-set S;
4. $\mathscr{S}$ terminates into a disjunction and both $\mathscr{S}_l/\sigma$ and $\mathscr{S}_r/\sigma$ are $\perp$-sets;
5. $\mathscr{S}/\sigma$ has a $\perp$-set successor $\mathscr{S}/\sigma.a.n$.

The depth-first algorithm is then immediate:

— select a segment $\mathscr{S}$ and a prefix σ which have some unreduced formulae

— if $\mathscr{S}/\sigma$ is a $\perp$-set (already seen or by using the two local conditions) then mark all its prefixed formulae as reduced *and* apply Definition 9 to generate new $\perp$-sets;

— else if $\mathscr{S}/\sigma$ is a copy (among the visited nodes) then mark all its prefixed formulae as reduced;

— else select a formula and reduce it; if any of the resulting segment $\mathscr{S}'$ is such that $\mathscr{S}'/\sigma$ is fully reduced then add $\mathscr{S}'/\sigma$ among the visited nodes;

— loop until either the root has become a $\perp$-set (UNSAT) or an open branch can be found (SAT).

Not all selection rules may work properly with this algorithm. It suffices that they are *copy preserving*: if, at the certain stage of the computation, $\mathscr{S}/\sigma$ has been detected as a copy of another $\mathscr{S}_0/\sigma_0$ it should remain so for the rest of the computation. For instance applying α, ν-like rules first (which do not introduce branching or new prefixes) preserves copies for PDL.

The correctness of the algorithm can be proved by a "cut and paste" argument: whenever we closed a branch due to a bad set we just replace the previously generated subtree for the bad set, and so on inductively.

For its complexity observe that the number of prefixes of an open branch is bounded by the number of visited sets (less $\perp$-sets) and these are exponentially bounded by the size of the Fisher–Ladner closure. Each closed branch introduces at least one new bad set and those are also bounded by the Fisher–Ladner closure.

For instance suppose we want to close $\mathscr{S}_l/\sigma$ and then $\mathscr{S}_r/\sigma$. These two may be equal or both already seen but $\mathscr{S}/\sigma$ is surely added as a new $\perp$-set (otherwise we would not have applied the β-rule but directly closed the branch).

For CPDL things are more involved. The first modification, already mentioned, is to use pairs $\langle \mathscr{S}/\sigma, last(\sigma) \rangle$ for visited nodes. The second step is to introduce the notion of *temporary copy* in a similar fashion of [13, 18] for symmetric and Euclidean modal logics. When $\mathscr{S}/\sigma$ is found to be a copy its unreduced formulae are simply frozen and eventually reactivated when a new prefixed formula with prefix σ or σ_0 is introduced (note that a set may be reactivated at most $|\Phi|$-times). Moreover, in the $\perp$-set condition (2), cut-formulae must have been applied in all possible ways.

Finally we must also change the notion of successor and add the notion of a *predecessor*. For instance the formula $\neg P \wedge \langle a \rangle [a^-] P$ has the tableau

$$\{1 : \neg P \wedge \langle a \rangle [a^-] P \sharp 1 : \neg P \sharp 1 : \langle a \rangle [a^-] P \sharp 1.a.2 : [a^-] P \sharp 1 : P\}$$

With the PDL-definition the $\perp$-set would only have been $\{P, \neg P\}$ while also $\{\neg P, \langle a \rangle [a^-] P\}$ is clearly a $\perp$-set.

In practice we cannot only rewind downward from $\mathscr{S}/\sigma.A.n$ to $\mathscr{S}/\sigma$ through the application of the $\langle A \rangle$-rule. So, if $\mathscr{S}/\sigma$ is found to be a $\perp$-set as in the example above then we must rewind upward from $\mathscr{S}/\sigma$ to $\mathscr{S}/\sigma.A.n$ through the $[A^-]^B$-rules until we have rewinded them all and then downward again with the $\langle A \rangle$-rules. At the end of the process we arrive at the same prefix σ in a shorter segment $\mathscr{S}_0$ and $\mathscr{S}_0/\sigma$ is the new $\perp$-set.

8. CONCLUSION

One characterizing feature of PDLs is the presence of *fixpoint* operators. In comparison with tableaux for modal logics [11, 13, 14, 18], the tableaux for modal fixpoint logics are conceptually divided in two: (1) build a (pseudo)model expanding the modal part; (2) check this model for the satisfiability of fixpoint formulae. The notion of ignorable branches stems out from the idea of merging the second step into the first one.

Such a merging requires one to keep track, during the expansions phase, of iterated eventualities and of their fulfillment, and to this extent we adopted the techniques used for modal μ-calculus in [24, 25]. The necessity of (successful and unsuccessful) loop checking for eventualities has been pointed out in [20, 22] for PDL and is even stronger for the modal μ-calculus [25].

We think that the combination of model checking and deductive techniques, improves the efficiency and readability of the calculus. In this setting our tableaux calculus is a first step[9] toward effective decision procedures for CPDL and the corresponding description logics.

ACKNOWLEDGMENTS

We thank F. Donini and M. Vardi for useful discussions and suggestions. We gratefully acknowledge the financial support of the Agenzia Spaziale Italiana (ASI), Consiglio Nazionale delle Ricerche (CNR), and Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST). Part of this work was done while F. Massacci was visiting the Computer Laboratory at the University of Cambridge (UK).

Received July 27, 1998; final manuscript received September 29, 1998; published online August 10, 2000

REFERENCES

1. Alur, R., and Henzinger, T., Eds. (1996), "Proc. of the Int. Conf. on Comp. Aided Verification (CAV-96)," Lecture Notes in Computer Science, Vol. 1102, Springer-Verlag, Berlin/New York.
2. Bhat, G., and Cleaveland, R. (1996), Efficient local model-checking for fragments of the modal μ-calculus, *in* "Proc. of the Int. Conf. on Rools and Algorithms for the Construction and Analysis

[9] For PDLs without the finite model property, the direct search for a model of tableaux techniques is infeasible. It may be possible to extend them by searching for finite structures *representing* models, i.e., that contain enough information to be expanded into full (infinite) models. This is also well known from first order tableaus.

of Systems (TACAS-96)," Lecture Notes in Computer Science, Vol. 1055, pp. 107–126, Springer-Verlag, Berlin/New York.

3. Bhat, G., Cleaveland, R., and Grumberg, O. (1995), Efficient on-the-fly model checking for CTL*, *in* "Proc. of the 10th Int. Conf. of Logic in Comp. Sci. (LICS-95)," pp. 388–397, IEEE Comput. Soc. Press, Los Alamitos, CA.
4. Blackburn, P., and Spaan, E. (1993), A modal perspective on computational complexity of attribute value grammar, *J. Logic*, *Language Inform.* **2**, 129–169.
5. Calvanese, D., De Giacomo, G., and Lenzerini, M. (1995), Structured objects: Modeling and reasoning, *in* "Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD-95)," Lecture Notes in Computer Science, Vol. 1013, pp. 229–246, Springer-Verlag, Berlin/New York.
6. De Giacomo, G. (1996), Eliminating "converse" from Converse PDL, *J. Logic*, *Language Inform.* **5**, 193–208.
7. De Giacomo, G., and Lenzerini, M. (1994), Boosting the correspondence between description logics and propositional dynamic logics, *in* "Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)," pp. 205–212, AAAI Press/The MIT PRESS, Cambridge, MA.
8. De Giacomo, G., and Massacci, F. (1996), Tableaux and algorithms for propositional dynamic logic with converse, *in* "Proc. of the 13th Int. Conf. on Automated Deduction (CADE-96)," Lecture Notes in Artificial Intelligence, Vol. 1104, pp. 613–628, Springer-Verlag, Berlin/New York.
9. Emerson, A. (1996), Automated temporal reasoning about reactive systems, *in* "Logics for Concurrency (Structure versus Automata)" (F. Moller and G. Birtwistle, Eds.), Lecture Notes in Computer Science, Vol. 1043, pp. 41–101, Springer-Verlag, Berlin/New York.
10. Fisher, N., and Ladner, R. (1979), Propositional dynamic logic of regular programs, *J. Comput. System Sci.* **18**, 194–211.
11. Fitting, M. (1983), "Proof Methods for Modal and Intuitionistic Logics," Reidel, Dordrecht.
12. Friedman, N., and Halper, J. (1994), On the complexity of conditional logics, *in* "Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)," pp. 202–213, Morgan Kaufman, San Mateo, CA.
13. Goré, R. (1995), "Tableaux Method for Modal and Temporal Logics," Tech. Rep. TR-ARP-15-5, Australian National University. [To appear as chapter in the Handbook of Tableau Methods]
14. Halpern, J., and Moses, Y. (1992), A guide to completeness and complexity for modal logics of knowledge and belief, *Artificial Intelligence* **54**, 319–379.
15. Kozen, D., and Tiuryn, J. (1990), Logic of programs, *in* "Handbook of Theoretical Computer Science" (J. van Leeuwen, Ed.), Vol. II, Chap. 14, pp. 789–840, Elsevier Science, Amsterdam.
16. Ladner, R. (1977), The computational complexity of provability in systems of modal propositional logic, *SIAM J. Comput.* **6** 467–480.
17. Manna, Z., and Pnueli, A. (1995), "Temporal Verification of Reactive Systems (Safety)," Vol. 2, Springer-Verlag, Berlin/New York.
18. Massacci, F. (1994), Strongly analytic tableaux for normal modal logics, *in* "Proc. of the 12th Int. Conf. on Automated Deduction (CADE-94)," Lecture Notes in Artificial Intelligence, Vol. 814, pp. 723–737, Springer-Verlag, Berlin.
19. Moller, F., and Birtwistle, G., Eds. (1996), "Logics for Concurrency (Structure versus Automata)," Lecture Notes in Computing Science, Vol. 1043, Springer-Verlag, Berlin/New York.
20. Pratt, V. (1978), A practical decision method for propositional dynamic logic, *in* "Proc. of the 10th ACM Symp. on Theory of Computing (STOC-78)," pp. 326–337.
21. Pratt, V. (1979), Models of program logics, *in* "Proc. of the 20th Annual Symp. on the Found. of Comp. Sci. (FOCS-79)," pp. 115–122, IEEE Comput. Soc. Press, Los Alamitos, CA.
22. Pratt, V. (11980), A near-optimal method for reasoning about action, *J. Comput. System Sci.* **20**, 231–255.
23. Schild, K. (1991), A correspondence theory for terminological logics: Preliminary report, *in* "Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)," pp. 466–471, Morgan Kaufmann, Los Altos, CA.

24. Stirling, C. (1996), Modal and temporal logics for processes, *in* "Logics for Concurrency (Structure versus Automata)" (F. Moller and G. Birtwistle, Eds.), Lecture Notes in Computer Science, Vol. 1043, pp. 149–237, Springer-Verlag, Berlin/New York.

25. Stirling, C., and Walker, D. (1991), Local model checking in modal mu-calculus, *Theoret. Comput. Sci.* **89**, 161–177.

26. Streett, R., and Emerson, A. (1989), An automata theoretic decision procedure for the propositional mu-calculus, *Inform. and Control* **81**, 249–264.

27. Vardi, M., and Wolper, P. (1986), Automata-theoretic techniques for modal logics of programs, *J. Comput. System Sci.* **32**, 183–221.

28. Vardi, M., and Wolper, P. (1994), Reasoning about infinite computations, *Inform. and Comput.* **115**, 1–37.

29. Woods, W., and Schmolze, J. (1992), The KL-ONE family, *in* "Semantic Networks in Artificial Intelligence," pp. 133–178, Pergamon Press, New York.