ExpTime Tableaux with Global Caching for Graded Propositional Dynamic Logic

Linh Anh Nguyen

Institute of Informatics, University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
nguyen@mimuw.edu.pl

Abstract. We present the first direct tableau decision procedure for graded PDL, which uses global caching and has ExpTime (optimal) complexity when numbers are encoded in unary. It shows how to combine integer linear feasibility checking with checking fulfillment of existential star modalities for tableaux with global caching.

1 Introduction

Propositional dynamic logic (PDL) is a well-known modal logic [5, 8]. Originally, it was developed as a logic for reasoning about programs. However, its extensions are also used for other purposes. For example, converse-PDL with regular inclusion axioms (CPDL$_{reg}$) can be used as a framework for multiagent logics [4]. As a variant of PDL, $\mathcal{ALC}_{reg}$ is a description logic for representing and reasoning about terminological knowledge. Several extensions of $\mathcal{ALC}_{reg}$ have been studied by the description logic community [6].

The satisfiability problem in PDL is EXPTIME-complete [5]. In [15], Pratt gave a tableau decision procedure with global caching for deciding PDL. In [14], Nguyen and Szałas reformulated that procedure and extended it for dealing with checking consistency of an ABox w.r.t. a TBox in PDL (where PDL is treated as a description logic). The work [1] by Abate et al. gives another tableau decision procedure with global caching for PDL, which propagates unfulfillment of existential star modalities on-the-fly. There are also tableau decision procedures with global caching or state global caching for CPDL (PDL with converse) [13, 7] and CPDL$_{reg}$ [4, 10].

Graded modal logics allow graded modalities for reasoning about the number of successor states with a certain property. They have attracted attention from many researchers.[1] In description logic, the counterpart of graded modalities is qualified number restrictions. Some well-known EXPTIME description logics with qualified number restrictions are $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$. The description logic corresponding to graded CPDL is $\mathcal{CIQ}$ [3]. De Giacomo and Lenzerini [3] proved that the satisfiability problem in $\mathcal{CIQ}$ is EXPTIME-complete when numbers are encoded in unary. Tobies [16] proved that the satisfiability problem in $\mathcal{SHIQ}$ is EXPTIME-complete even when numbers are encoded in binary.

[1] See http://www.cs.man.ac.uk/~ezolin/ml/ for a list of related publications.

In this paper, we present the first direct tableau decision procedure for GPDL (graded PDL). Our procedure uses global caching and has EXPTIME (optimal) complexity when numbers are encoded in unary. It shows how to combine integer linear feasibility checking [9, 12] with checking fulfillment of existential star modalities for tableaux with global caching.

As related work on automated reasoning in GPDL and its extensions, De Giacomo and Lenzerini gave methods for translating the satisfiability problem in $\mathcal{CIQ}$ into $\mathcal{CIF}$ (a variant of CPDL with functionalities) [3], and in $\mathcal{CIF}$ into CPDL [2]. This established the complexity EXPTIME-complete for $\mathcal{CIQ}$ (and GPDL) when numbers are encoded in unary. However, this indirect method cannot give an efficient decision procedure for GPDL because it is not scalable w.r.t. numbers in graded modalities (i.e., qualified number restrictions). In particular, the translation from $\mathcal{CIQ}$ to $\mathcal{CIF}$ [3] may result in a formula with a quadratic length, and similarly for the translation from $\mathcal{CIF}$ to CPDL [3].

The rest of this paper is structured as follows. In Section 2, we present the notation and semantics of GPDL and recall automaton-modal operators [8, 4], which are used for our tableaux. We omit the feature of "global assumptions" as they can be expressed in PDL (by "local assumptions"). In Section 3, we present a tableau calculus for GPDL, starting with the data structure, the tableau rules and ending with the corresponding tableau decision procedure and its properties. In Section 4, we give an example for illustrating our procedure. Concluding remarks are given in Section 5.

2 Preliminaries

2.1 Graded Propositional Dynamic Logic

We use Σ to denote the set of *atomic programs*, and $\mathcal{PROP}$ to denote the set of *propositions* (i.e., atomic formulas). We denote elements of Σ by letters like σ and ϱ, and elements of $\mathcal{PROP}$ by letters like p and q.

A *Kripke model* is a pair $\mathcal{M} = \langle \Delta^{\mathcal{M}}, \cdot^{\mathcal{M}} \rangle$, where $\Delta^{\mathcal{M}}$ is a set of *states* and $\cdot^{\mathcal{M}}$ is an interpretation function that maps each proposition $p \in \mathcal{PROP}$ to a subset $p^{\mathcal{M}}$ of $\Delta^{\mathcal{M}}$ and each atomic program $\sigma \in \Sigma$ to a binary relation $\sigma^{\mathcal{M}}$ on $\Delta^{\mathcal{M}}$. Intuitively, $p^{\mathcal{M}}$ is the set of states in which p is true and $\sigma^{\mathcal{M}}$ is the binary relation consisting of pairs (input_state, output_state) of the program σ.

Formulas and *programs* of the *base language* of GPDL are defined respectively by the following grammar, where $p \in \mathcal{PROP}$, $\sigma \in \Sigma$ and n is a natural number:

$$\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi \mid \geq n\, \sigma.\varphi \mid \leq n\, \sigma.\varphi$$
$$\alpha ::= \sigma \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha^* \mid \varphi?$$

Notice that we use the notation $\geq n\, \sigma.\varphi$ and $\leq n\, \sigma.\varphi$ as in description logic instead of $\langle\sigma\rangle_{\geq n}\varphi$ and $\langle\sigma\rangle_{\leq n}\varphi$ (as the latter do not look "dual" to each other).

We use letters like α, β to denote programs, and φ, ψ, ξ to denote formulas.

The intended meaning of program operators is as follows:

- $\alpha;\beta$ stands for the sequential composition of α and β

$$
\begin{array}{rlrl}
(\alpha;\beta)^{\mathcal{M}} &= \alpha^{\mathcal{M}} \circ \beta^{\mathcal{M}} & (\alpha \cup \beta)^{\mathcal{M}} &= \alpha^{\mathcal{M}} \cup \beta^{\mathcal{M}} \\
(\alpha^*)^{\mathcal{M}} &= (\alpha^{\mathcal{M}})^* & (\varphi?)^{\mathcal{M}} &= \{(x,x) \mid \varphi^{\mathcal{M}}(x)\} \\
\top^{\mathcal{M}} &= \Delta^{\mathcal{M}} & \bot^{\mathcal{M}} &= \emptyset \\
(\neg\varphi)^{\mathcal{M}} &= \Delta^{\mathcal{M}} \setminus \varphi^{\mathcal{M}} & (\varphi \to \psi)^{\mathcal{M}} &= (\neg\varphi \vee \psi)^{\mathcal{M}} \\
(\varphi \wedge \psi)^{\mathcal{M}} &= \varphi^{\mathcal{M}} \cap \psi^{\mathcal{M}} & (\varphi \vee \psi)^{\mathcal{M}} &= \varphi^{\mathcal{M}} \cup \psi^{\mathcal{M}}
\end{array}
$$

$$
\begin{array}{rl}
(\langle\alpha\rangle\varphi)^{\mathcal{M}} &= \{x \in \Delta^{\mathcal{M}} \mid \exists y(\alpha^{\mathcal{M}}(x,y) \wedge \varphi^{\mathcal{M}}(y))\} \\
([\alpha]\varphi)^{\mathcal{M}} &= \{x \in \Delta^{\mathcal{M}} \mid \forall y(\alpha^{\mathcal{M}}(x,y) \to \varphi^{\mathcal{M}}(y))\} \\
(\geq n\,\sigma.\varphi)^{\mathcal{M}} &= \{x \in \Delta^{\mathcal{M}} \mid \#\{y \in \Delta^{\mathcal{M}} \mid \sigma^{\mathcal{M}}(x,y) \wedge \varphi^{\mathcal{M}}(y)\} \geq n\} \\
(\leq n\,\sigma.\varphi)^{\mathcal{M}} &= \{x \in \Delta^{\mathcal{M}} \mid \#\{y \in \Delta^{\mathcal{M}} \mid \sigma^{\mathcal{M}}(x,y) \wedge \varphi^{\mathcal{M}}(y)\} \leq n\}
\end{array}
$$

Fig. 1. Interpretation of complex programs and complex formulas.

- $\alpha \cup \beta$ stands for the set-theoretical union of α and β
- α^* stands for the reflexive and transitive closure of α
- $\varphi?$ stands for the test operator.

Informally, a formula $\langle\alpha\rangle\varphi$ represents the set of states x such that the program α has a transition from x to a state y satisfying φ. Dually, a formula $[\alpha]\varphi$ represents the set of states x from which every transition of α leads to a state satisfying φ. A formula $\geq n\,\sigma.\varphi$ (resp. $\leq n\,\sigma.\varphi$) represents the set of states x such that the program σ has transitions from x to at least (resp. at most) n pairwise different states satisfying φ.

Formally, the interpretation function of a Kripke model $\mathcal{M}$ is extended to interpret complex formulas and complex programs as shown in Figure 1.

We write $\mathcal{M}, w \models \varphi$ to denote $w \in \varphi^{\mathcal{M}}$. For a set Γ of formulas, we write $\mathcal{M}, w \models \Gamma$ to denote that $\mathcal{M}, w \models \varphi$ for all $\varphi \in \Gamma$. If $\mathcal{M}, w \models \varphi$ (resp. $\mathcal{M}, w \models \Gamma$), then we say that $\mathcal{M}$ *satisfies* φ (resp. Γ) *at* w, and that φ (resp. Γ) is *satisfied at* w in $\mathcal{M}$.

A formula is in negation normal form (NNF) if it does not use $\to$ and it uses $\neg$ only immediately before propositions, and furthermore, it does not contain subformulas of the form $\geq 0\,\sigma.\varphi$ or $\leq 0\,\sigma.\varphi$. Every formula can be transformed to an equivalent formula in NNF. By $\overline{\varphi}$ we denote the NNF of $\neg\varphi$.

2.2 Automaton-Modal Operators

The *alphabet* $\Sigma(\alpha)$ of a program α and the *regular language* $\mathcal{L}(\alpha)$ generated by α are specified as follows:[2]

[2] Note that $\Sigma(\alpha)$ contains not only atomic programs but also expressions of the form $(\varphi?)$, and a program α is a regular expression over its alphabet $\Sigma(\alpha)$.

$$\begin{array}{ll} \Sigma(\sigma) = \{\sigma\} & \mathcal{L}(\sigma) = \{\sigma\} \\ \Sigma(\varphi?) = \{\varphi?\} & \mathcal{L}(\varphi?) = \{\varphi?\} \\ \Sigma(\beta;\gamma) = \Sigma(\beta) \cup \Sigma(\gamma) & \mathcal{L}(\beta;\gamma) = \mathcal{L}(\beta).\mathcal{L}(\gamma) \\ \Sigma(\beta \cup \gamma) = \Sigma(\beta) \cup \Sigma(\gamma) & \mathcal{L}(\beta \cup \gamma) = \mathcal{L}(\beta) \cup \mathcal{L}(\gamma) \\ \Sigma(\beta^*) = \Sigma(\beta) & \mathcal{L}(\beta^*) = (\mathcal{L}(\beta))^* \end{array}$$

where for sets of words M and N, $M.N = \{\alpha\beta \mid \alpha \in M, \beta \in N\}$, $M^0 = \{\varepsilon\}$ (where ε denotes the empty word), $M^{n+1} = M.M^n$ for $n \geq 0$, and $M^* = \bigcup_{n\geq 0} M^n$.

We will use letters like ω to denote either an atomic program from Σ or a test (of the form $\varphi?$). A word $\omega_1 \ldots \omega_k \in \mathcal{L}(\alpha)$ can be treated as the program $(\omega_1;\ldots;\omega_k)$, especially when it is interpreted in a Kripke model.

Recall that a *finite automaton* A over alphabet $\Sigma(\alpha)$ is a tuple $\langle \Sigma(\alpha), Q, I, \delta, F\rangle$, where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $\delta \subseteq Q \times \Sigma(\alpha) \times Q$ is the transition relation, and $F \subseteq Q$ is the set of accepting states. A *run* of A on a word $\omega_1 \ldots \omega_k$ is a finite sequence of states $q_0, q_1, \ldots, q_k$ such that $q_0 \in I$ and $\delta(q_{i-1}, \omega_i, q_i)$ holds for every $1 \leq i \leq k$. It is an *accepting run* if $q_k \in F$. We say that A *accepts* a word w if there exists an accepting run of A on w. The set of words accepted by A is denoted by $\mathcal{L}(A)$.

We will use the following convention:

- given a finite automaton A, we always assume that $A = (\Sigma_A, Q_A, I_A, \delta_A, F_A)$
- for $q \in Q_A$, we define $\delta_A(q) = \{(\omega, q') \mid (q, \omega, q') \in \delta_A\}$.

As a finite automaton A over alphabet $\Sigma(\alpha)$ corresponds to a program (the regular expression recognizing the same language), it is interpreted in a Kripke model as follows:

$$A^M = \bigcup\{\gamma^M \mid \gamma \in \mathcal{L}(A)\}. \tag{1}$$

For each program α, let $\mathbb{A}_\alpha$ be a finite automaton recognizing the regular language $\mathcal{L}(\alpha)$. The automaton $\mathbb{A}_\alpha$ can be constructed from α in polynomial time. We extend the base language with the auxiliary modal operators $[A, q]$ and $\langle A, q\rangle$, where A is $\mathbb{A}_\alpha$ for some program α and q is a state of A. Here, $[A, q]$ and $\langle A, q\rangle$ stand respectively for $[(A, q)]$ and $\langle(A, q)\rangle$, where (A, q) is the automaton that differs from A only in that q is its only initial state. We call $[A, q]$ (resp. $\langle A, q\rangle$) a *universal* (resp. *existential*) *automaton-modal operator*.

In the *extended language*, if φ is a formula, then $[A, q]\varphi$ and $\langle A, q\rangle\varphi$ are also formulas. The semantics of these formulas are defined as usual, treating (A, q) as a program with semantics specified by (1).

Given a Kripke model M and a state $x \in \Delta^M$, we have that $x \in ([A, q]\varphi)^M$ (resp. $x \in (\langle A, q\rangle\varphi)^M$) iff

> $x_k \in \varphi^M$ for all (resp. some) $x_k \in \Delta^M$ such that there exist a word $\omega_1 \ldots \omega_k$ (with $k \geq 0$) accepted by (A, q) with $(x, x_k) \in (\omega_1;\ldots;\omega_k)^M$.

The condition $(x, x_k) \in (\omega_1;\ldots;\omega_k)^M$ means there exist states $x_0 = x, x_1, \ldots, x_{k-1}$ of M such that, for each $1 \leq i \leq k$, if $\omega_i \in \Sigma$ then $(x_{i-1}, x_i) \in \omega_i^M$, else $\omega_i = (\psi_i?)$ for some ψ_i and $x_{i-1} = x_i$ and $x_i \in \psi_i^M$. Clearly, $\langle A, q\rangle$ is dual to $[A, q]$ in the sense that $\langle A, q\rangle\varphi \equiv \neg[A, q]\neg\varphi$ for any formula φ.

3 A Tableau Calculus for GPDL

In this section we present a tableau calculus for checking whether a given finite set of formulas in NNF is satisfiable. We specify the data structure, the tableau rules, the corresponding tableau decision procedure and state its properties.

3.1 The Data Structure

Let $EdgeLabels = \{\mathsf{testingUnsat}, \mathsf{checkingFeasibility}\} \times \Sigma$. For $e \in EdgeLabels$, let $e = (\pi_T(e), \pi_\Sigma(e))$. Thus, $\pi_T(e)$ is called the type of the edge label e, and $\pi_\Sigma(e)$ is an atomic program.

A *tableau* is a rooted graph $G = (V, E, \nu)$, where V is a set of nodes, $E \subseteq V \times V$ is a set of edges, $\nu \in V$ is the root, each node $v \in V$ has a number of attributes, and each edge (v, w) may be labeled by a set $ELabels(v, w) \subseteq EdgeLabels$. The attributes of a tableau node v are:

- $Type(v) \in \{\mathsf{state}, \mathsf{non\text{-}state}\}$,
- $Status(v) \in \{\mathsf{unexpanded}, \mathsf{p\text{-}expanded}, \mathsf{f\text{-}expanded}, \mathsf{unsat}\}$,
- $Label(v)$, which is a finite set of formulas, called the *label* of v,
- $RFmls(v)$, which is a finite set of so called *reduced formulas* of v,
- $ILConstraints(v)$, which is a finite set of integer linear constraints.

We call v a state if $Type(v) = \mathsf{state}$, and a non-state otherwise. If $(v, w) \in E$ then we call v a *predecessor* of w and w a *successor* of v. An edge outgoing from a node v is labeled iff $Type(v) = \mathsf{state}$. The statuses p-expanded, f-expanded and unsat mean "partially expanded", "fully expanded", and "unsatisfiable", respectively. $Status(v)$ may be p-expanded only when $Type(v) = \mathsf{state}$, and $RFmls(v) \neq \emptyset$ only when $Type(v) = \mathsf{non\text{-}state}$.

$ILConstraints(v)$ is available only when $Type(v) = \mathsf{state}$ and

$$Status(v) \notin \{\mathsf{unexpanded}, \mathsf{p\text{-}expanded}\}.$$

The constraints use variables $x_{w,e}$ indexed by a pair (w, e) such that $(v, w) \in E$, $e \in ELabels(v, w)$ and $\pi_T(e) = \mathsf{checkingFeasibility}$. Such a variable specifies how many copies of the successor w using the edge label e will be created for v.

We apply global caching in the sense that if v_1 and v_2 are different nodes then either $Type(v_1) \neq Type(v_2)$ or $Label(v_1) \neq Label(v_2)$ or $RFmls(v_1) \neq RFmls(v_2)$.

By $FullLabel(v)$ we denote the set $Label(v) \cup RFmls(v)$.

3.2 Tableau Rules

Our tableau calculus $\mathcal{C}_{\mathrm{GPDL}}$ for the GPDL is specified by a number of static rules, the $(forming\text{-}state)$ rule, two transitional rules and the $(unsat)$ rule for updating unsatisfiability of nodes. The rules except $(unsat)$ are used to expand nodes of tableaux. Static rules are written downwards, with a set of formulas above the line as the *premise*, which represents the label of the node to which the rule is applied, and a number of sets of formulas below the line as the *(possible) conclusions*, which represent the labels of the

Table 1. The static rules of $\mathcal{C}_{\mathrm{GPDL}}$

$(\wedge)\ \dfrac{X, \varphi \wedge \psi}{X, \varphi, \psi}$ $\qquad$ $(\vee)\ \dfrac{X, \varphi \vee \psi}{X, \varphi \mid X, \psi}$
if $\alpha \notin \Sigma,\ \alpha$ is not a test, and $I_{\mathbb{A}_\alpha} = \{q_1, \ldots, q_k\}$: $(aut_\Box)\ \dfrac{X, [\alpha]\varphi}{X, [\mathbb{A}_\alpha, q_1]\varphi, \ldots, [\mathbb{A}_\alpha, q_k]\varphi}$ $(aut_\Diamond)\ \dfrac{X, \langle\alpha\rangle\varphi}{X, \langle\mathbb{A}_\alpha, q_1\rangle\varphi \mid \ldots \mid X, \langle\mathbb{A}_\alpha, q_k\rangle\varphi}$
if $\delta_A(q) = \{(\omega_1, q_1), \ldots, (\omega_k, q_k)\}$ and $q \notin F_A$: $([\mathrm{A}])\ \dfrac{X, [A, q]\varphi}{X, [\omega_1][A, q_1]\varphi, \ldots, [\omega_k][A, q_k]\varphi}$ $(\langle\mathrm{A}\rangle)\ \dfrac{X, \langle A, q\rangle\varphi}{X, \langle\omega_1\rangle\langle A, q_1\rangle\varphi \mid \ldots \mid X, \langle\omega_k\rangle\langle A, q_k\rangle\varphi}$
if $\delta_A(q) = \{(\omega_1, q_1), \ldots, (\omega_k, q_k)\}$ and $q \in F_A$: $([\mathrm{A}]_f)\ \dfrac{X, [A, q]\varphi}{X, [\omega_1][A, q_1]\varphi, \ldots, [\omega_k][A, q_k]\varphi, \varphi}$ $(\langle\mathrm{A}\rangle_f)\ \dfrac{X, \langle A, q\rangle\varphi}{X, \langle\omega_1\rangle\langle A, q_1\rangle\varphi \mid \ldots \mid X, \langle\omega_k\rangle\langle A, q_k\rangle\varphi \mid X, \varphi}$
$(\Box_?)\ \dfrac{X, [\psi?]\varphi}{X, \overline{\psi} \mid X, \varphi}$ $\qquad$ $(\Diamond_?)\ \dfrac{X, \langle\psi?\rangle\varphi}{X, \psi, \varphi}$
$(\geq_\Diamond)\ \dfrac{X, \geq n\, \sigma.\varphi}{X, \geq n\, \sigma.\varphi, \langle\sigma\rangle\varphi}\ (n \geq 1)$ $(\Diamond_\geq)\ \dfrac{X, \langle\sigma\rangle\varphi}{X, \langle\sigma\rangle\varphi, \geq 1\, \sigma.\varphi}$ if X does not contain any $\geq n\, \sigma.\varphi$ with $n \geq 1$

successor nodes resulting from the application of the rule. Possible conclusions of a static rule are separated by |. If a rule is unary (i.e. with only one possible conclusion) then its only conclusion is "firm" and we ignore the word "possible". The meaning of a static rule is that if the premise is satisfiable then some of the possible conclusions are also satisfiable.

We use Γ, X, Y to denote sets of formulas and write Γ, φ to denote $\Gamma \cup \{\varphi\}$. The static rules of $\mathcal{C}_{\mathrm{GPDL}}$ are specified in Table 1. For any among them, the distinguished

Function `NewSucc` $(v, type, label, rFmls, eLabel)$

Global data: a rooted graph (V, E, ν).
Purpose: create a new successor for v.
create a new node w, $V := V \cup \{w\}$, $E := E \cup \{(v, w)\}$;
$Type(w) := type$, $Status(w) :=$ unexpanded;
$Label(w) := label$, $RFmls(w) := rFmls$;
if $Type(v) =$ state **then** $ELabels(v, w) := \{eLabel\}$;
return w;

Function `ConToSucc` $(v, type, label, rFmls, eLabel)$

Global data: a rooted graph (V, E, ν).
Purpose: connect a node v to a successor, which is created if necessary.
if *there exists a node* w *such that* $Type(w) = type$, $Label(w) = label$ *and* $RFmls(w) = rFmls$ **then**
  $E := E \cup \{(v, w)\}$;
  if $Type(v) =$ state **then** $ELabels(v, w) := ELabels(v, w) \cup \{eLabel\}$;
else
  $w :=$ `NewSucc`$(v, type, label, rFmls, eLabel)$;
return w;

formula of the premise is called the *principal formula* of the rule. A static rule ρ is *applicable* to a node v if the following conditions hold:

- $Status(v) =$ unexpanded and $Type(v) =$ non-state,
- the premise of the rule is equal to $Label(v)$,
- the conditions accompanied with ρ are satisfied,
- the principal formula of ρ does not belong to $RFmls(v)$.

The last condition means that if the principal formula belongs to $RFmls(v)$ then ρ has been applied to an ancestor node of v that corresponds to the same state in the intended Kripke model as v, and therefore should not be applied again.

If ρ is a static rule applicable to v, then the application is as follows:

- Let φ be the principal formula and $X_1, \ldots, X_k$ the possible conclusions of ρ.
- For each $1 \leq i \leq k$, do `ConToSucc`$(v,$ non-state$, X_i, RFmls(v) \cup \{\varphi\},$ null$)$, which is specified on page 50.
- $Status(v) :=$ f-expanded.

The $(forming\text{-}state)$ rule is applicable to a node v if $Type(v) =$ non-state and no static rule is applicable to v. Application of this rule to such a node v is done by calling `ConToSucc`$(v,$ state$, Label(v), \emptyset,$ null$)$.

The transitional partial-expansion rule (TP) is applicable to a node v if $Type(v) =$ state and $Status(v) =$ unexpanded. Application of this rule to such a node v is done as follows:

1. for each $\langle\sigma\rangle\varphi \in Label(v)$ (where $\sigma \in \Sigma$), do

   (a) $label := \{\varphi\} \cup \{\psi \mid [\sigma]\psi \in Label(v)\}$
   (b) $eLabel := (\mathsf{testingUnsat}, \sigma)$
   (c) $\mathtt{ConToSucc}(v, \mathsf{non\text{-}state}, label, \emptyset, eLabel)$
2. $Status(v) := \mathsf{p\text{-}expanded}$.

The transitional full-expansion rule (TF) is applicable to a node v if $Type(v) = \mathsf{state}$ and $Status(v) = \mathsf{p\text{-}expanded}$. Application of this rule to such a node v is done as follows:

1. $\mathcal{E} := \emptyset$, $\mathcal{E}' := \emptyset$
2. for each $(\geq n\, \sigma.\varphi) \in Label(v)$ do
   $\mathcal{E} := \mathcal{E} \cup \{(\sigma, X)\}$, where $X = \{\varphi\} \cup \{\psi \mid [\sigma]\psi \in Label(v)\}$
3. for each $(\leq n\, \sigma.\varphi) \in Label(v)$ do
   (a) for each $(\sigma, X) \in \mathcal{E}$ do
      i. if $\{\varphi, \overline{\varphi}\} \cap X = \emptyset$ then $\mathcal{E}' := \mathcal{E}' \cup \{(\sigma, X \cup \{\varphi\}), (\sigma, X \cup \{\overline{\varphi}\})\}$
      (i.e., (σ, X) is replaced by $(\sigma, X \cup \{\varphi\})$ and $(\sigma, X \cup \{\overline{\varphi}\})$)
      ii. else $\mathcal{E}' := \mathcal{E}' \cup \{(\sigma, X)\}$
   (b) $\mathcal{E} := \mathcal{E}'$, $\mathcal{E}' := \emptyset$
4. repeat
   for each $(\leq n\, \sigma.\varphi) \in Label(v)$, $(\sigma, X) \in \mathcal{E}$ and $(\sigma, X') \in \mathcal{E}$ such that $\varphi \in X \cap X'$, $(\sigma, X \cup X') \notin \mathcal{E}$ and $X \cup X'$ does not contain any pair of the form ψ, $\overline{\psi}$, add $(\sigma, X \cup X')$ to $\mathcal{E}$ (i.e., the merger of (σ, X) and (σ, X') is added to $\mathcal{E}$)
   until no tuples were added to $\mathcal{E}$ during the last iteration
5. for each $(\sigma, X) \in \mathcal{E}$ do
   $\mathtt{ConToSucc}(v, \mathsf{non\text{-}state}, X, \emptyset, (\mathsf{checkingFeasibility}, \sigma))$
6. $ILConstraints(v) := \{x_{w,e} \geq 0 \mid (v, w) \in E, e \in ELabels(v, w)$ and $\pi_T(e) = \mathsf{checkingFeasibility}\}$
7. for each $\varphi \in Label(v)$ do
   (a) if φ is of the form $\geq n\, \sigma.\psi$ then add to $ILConstraints(v)$ the constraint $\sum\{x_{w,e} \mid (v, w) \in E,\ e \in ELabels(v, w),\ e = (\mathsf{checkingFeasibility}, \sigma),\ \psi \in Label(w)\} \geq n$
   (b) if φ is of the form $\leq n\, \sigma.\psi$ then add to $ILConstraints(v)$ the constraint $\sum\{x_{w,e} \mid (v, w) \in E,\ e \in ELabels(v, w),\ e = (\mathsf{checkingFeasibility}, \sigma),\ \psi \in Label(w)\} \leq n$
8. $Status(v) := \mathsf{f\text{-}expanded}$.

We give here an explanation for the rule (TF). To satisfy a requirement $(\geq n\, \sigma.\varphi) \in Label(v)$, one can first create a successor w of v specified by the pair (σ, X) computed at the step 2, where X presents $Label(w)$, and then clone w to create n successors for v (or only record the intention somehow). The label of w contains only formulas necessary for realizing the requirement $\geq n\, \sigma.\varphi$ and the related ones of the form $[\sigma]\psi$ at v. To satisfy requirements of the form $\leq n'\, \sigma.\varphi'$ at v, we tend to use only copies of such nodes like w extended with either φ' or $\overline{\varphi'}$ (for easy counting) as well as the mergers of such extended nodes. So, we first start with the set $\mathcal{E}$ constructed at the step 2, which consists of pairs with information about successors to be created for v. We then modify $\mathcal{E}$ by taking into account necessary extensions for the nodes (see the step 3). After that we continue modifying $\mathcal{E}$ by taking into account also appropriate mergers of nodes (see

the step 4). Successors for v are created at the step 5. The number of copies of a node w that are intended to be used as successors of v with an edge label e is represented by the variable $x_{w,e}$ (we will not actually create such copies). The set $ILConstraints(v)$ consisting of appropriate constraints about such variables are set at the steps 6-7.

Definition 1. Suppose $Status(v) \neq \mathsf{unsat}$ and $\langle A, q\rangle\varphi \in Label(v)$. A *trace* of $\langle A, q\rangle\varphi$ starting from v is a sequence $(v_0, \varphi_0), \ldots, (v_k, \varphi_k)$ such that:

- $v_0 = v$ and $\varphi_0 = \langle A, q\rangle\varphi$,
- for every $1 \leq i \leq k$, v_i is a successor of v_{i-1}, $Status(v_i) \neq \mathsf{unsat}$, and φ_i is a formula of $FullLabel(v_i)$ such that
  - if the tableau rule expanding v_{i-1} is a static rule and φ_{i-1} is not its principal formula then $\varphi_i = \varphi_{i-1}$,
  - else if the rule is $(\langle \mathrm{A}\rangle)$ or $(\langle \mathrm{A}\rangle_f)$ then φ_{i-1} is the principal formula of the form $\langle A, q'\rangle\varphi$ and φ_i is the formula obtained from φ_{i-1},
  - else if the rule is $(\Diamond_?)$ then φ_{i-1} is the principal formula of the form $\langle\psi?\rangle\langle A, q'\rangle\varphi$ and $\varphi_i = \langle A, q'\rangle\varphi$,
  - else $Type(v_{i-1}) = \mathsf{state}$, φ_{i-1} is of the form $\langle\sigma\rangle\langle A, q'\rangle\varphi$, v_i is a successor of v_{i-1} resulting from the application of the tableau rule (TP) to v_{i-1}, $(\mathsf{testingUnsat}, \sigma) \in ELabels(v_{i-1}, v_i)$, and $\varphi_i = \langle A, q'\rangle\varphi$.

The trace is called a $\Diamond$-*realization for* $\langle A, q\rangle\varphi$ *at* v_0 if $\varphi_k = \varphi$.

The $(unsat)$ rule is specified as follows: set $Status(v) := \mathsf{unsat}$ if $Status(v) \neq \mathsf{unsat}$ and one of the following holds:

1. $\bot \in Label(v)$ or there exists $\{\varphi, \neg\varphi\} \subseteq Label(v)$;
2. $Type(v) = \mathsf{non\text{-}state}$ and, for every $(v, w) \in E$, $Status(w) = \mathsf{unsat}$;
3. $Type(v) = \mathsf{state}$ and there exist $(v, w) \in E$ and $e \in ELabels(v, w)$ such that $\pi_T(e) = \mathsf{testingUnsat}$ and $Status(w) = \mathsf{unsat}$;
4. $Type(v) = \mathsf{state}$, $Status(v) = \mathsf{f\text{-}expanded}$ and $ILConstraints(v) \cup \{x_{w,e} = 0 \mid (v, w) \in E, e \in ELabels(v, w), \pi_T(e) = \mathsf{checkingFeasibility}$ and $Status(w) = \mathsf{unsat}\}$ is infeasible;
5. there does not exist any $\Diamond$-realization for some $\langle A, q\rangle\varphi \in Label(v)$ at v when all paths starting from v do not contain any node that can be modified by some tableau rule.

3.3 Checking Unsatisfiability

Let Γ be a finite set of formulas in NNF. A $\mathcal{C}_{\mathrm{GPDL}}$-*tableau* for Γ is a tableau $G = (V, E, \nu)$ constructed as follows. At the beginning, $V = \{\nu\}$, $E = \emptyset$ and the attributes of the root ν are specified as follows: $Type(\nu) = \mathsf{non\text{-}state}$, $Status(\nu) = \mathsf{unexpanded}$, $Label(\nu) = \Gamma$ and $RFmls(\nu) = \emptyset$. Then, while $Status(\nu) \neq \mathsf{unsat}$ and there is a tableau rule applicable to some node v, apply that rule to v.[3] Observe that the set of all formulas that may appear in the labels of the nodes of G is finite. Due to global caching, G is finite and can be effectively constructed.

[3] As an optimization, it makes sense to expand v only when there may exist a path from the root to v that does not contain any node with status unsat.

Theorem 1. *Let Γ be a finite set of formulas in NNF and $G = (V, E, \nu)$ an arbitrary $\mathcal{C}_{\mathrm{GPDL}}$-tableau for Γ. Then, Γ is unsatisfiable iff $Status(\nu) = \mathsf{unsat}$.*

To check satisfiability of a finite set Γ of formulas in NNF, one can construct a $\mathcal{C}_{\mathrm{GPDL}}$-tableau $G = (V, E, \nu)$ for Γ and return "no" when $Status(\nu) = \mathsf{unsat}$, or "yes" otherwise. We call this the $\mathcal{C}_{\mathrm{GPDL}}$*-tableau decision procedure*. Various optimization techniques [11] can be applied to this procedure.

Corollary 1. *The $\mathcal{C}_{\mathrm{GPDL}}$-tableau decision procedure has* ExpTime *complexity when numbers are encoded in unary.*

Theorem 1 and Corollary 1 can be proved in a similar way as done for the tableau decision procedures for CPDL$_{reg}$ [10], $\mathcal{SHIQ}$ [9] and $\mathcal{SHOQ}$ [12]. Proofs for them will be provided later for the full version of the current paper.

4 An Illustrative Example

Consider

$$\Gamma = \{\langle\sigma^*\rangle p,\ \neg p,\ [\sigma;\sigma;\sigma^*]\neg p,\ [\sigma](\neg p \vee \neg q),\ \geq 1000\, \sigma.q,\ \leq 1000\, \sigma.(p \vee q)\},$$

and let

$$\begin{array}{l} A_1 = \mathbb{A}_{\sigma^*} \quad\ = (\{\sigma\}, \{0\}, \{0\}, \{(0, \sigma, 0)\}, \{0\}) \\ A_2 = \mathbb{A}_{\sigma;\sigma;\sigma^*} = (\{\sigma\}, \{0, 1, 2\}, \{0\}, \{(0, \sigma, 1), (1, \sigma, 2), (2, \sigma, 2)\}, \{2\}). \end{array}$$

A $\mathcal{C}_{\mathrm{GPDL}}$-tableau $G = (V, E, \nu)$ for Γ is constructed as follows:

- At the beginning, G contains only the non-state ν with $Label(\nu) = \Gamma$.
- Applying $(aut_\diamond)$ to ν, this node is connected to a new non-state v_1 with

$$Label(v_1) = \Gamma - \{\langle\sigma^*\rangle p\} \cup \{\langle A_1, 0\rangle p\}.$$

- Applying $(aut_\Box)$ to v_1, this node is connected to a new non-state v_2 with

$$Label(v_2) = Label(v_1) - \{[\sigma;\sigma;\sigma^*]\neg p\} \cup \{[A_2, 0]\neg p\}.$$

- Applying ([A]) to v_2, this node is connected to a new non-state v_3 with

$$Label(v_3) = Label(v_2) - \{[A_2, 0]\neg p\} \cup \{[\sigma][A_2, 1]\neg p\}.$$

- Applying $(\geq_\diamond)$ to v_3, this node is connected to a new non-state v_4 with

$$Label(v_4) = Label(v_3) \cup \{\langle\sigma\rangle q\}.$$

- Applying $(\langle\mathrm{A}\rangle_f)$ to v_4 using the principal formula $\langle A_1, 0\rangle p$, this node is connected to two new non-states v_5 and v_6 with

$$\begin{array}{l} Label(v_5) = Label(v_4) - \{\langle A_1, 0\rangle p\} \cup \{p\} \\ Label(v_6) = Label(v_4) - \{\langle A_1, 0\rangle p\} \cup \{\langle\sigma\rangle\langle A_1, 0\rangle p\}. \end{array}$$

- Since $\{p, \neg p\} \subseteq Label(v_5)$, applying the $(unsat)$ rule to v_5, this node gets status unsat.
- Applying $(\Diamond_{\geq})$ to v_6, this node is connected to a new non-state v_7 with

$$\begin{aligned} Label(v_7) &= Label(v_6) \cup \{\geq 1\, \sigma.\langle A_1, 0\rangle p\} \\ &= \{\langle\sigma\rangle\langle A_1, 0\rangle p,\ \geq 1\, \sigma.\langle A_1, 0\rangle p,\ \neg p,\ [\sigma][A_2, 1]\neg p,\ [\sigma](\neg p \vee \neg q), \\ &\quad \geq 1000\, \sigma.q,\ \langle\sigma\rangle q,\ \leq 1000\, \sigma.(p \vee q)\}. \end{aligned}$$

- Applying the $(forming\text{-}state)$ rule to v_7, this node is connected to a new state v_8 with $Label(v_8) = Label(v_7)$.
- Applying (TP) to v_8, this state is connected to two new non-states v_9 and v_{10}, with $ELabels(v_8, v_9) = ELabels(v_8, v_{10}) = \{(\mathsf{testingUnsat}, \sigma)\}$ and

$$\begin{aligned} Label(v_9) &= \{\langle A_1, 0\rangle p,\ [A_2, 1]\neg p,\ \neg p \vee \neg q\} \\ Label(v_{10}) &= \{q,\ [A_2, 1]\neg p,\ \neg p \vee \neg q\}. \end{aligned}$$

- Applying (TF) to v_8, this state is connected to additional new non-states $v_{11} - v_{16}$, with $ELabels(v_8, v_i) = \{e\}$, where $e = (\mathsf{checkingFeasibility}, \sigma)$ and $11 \leq i \leq 16$, and

$$\begin{aligned} Label(v_{11}) &= Label(v_9) \cup \{p \vee q\} \\ Label(v_{12}) &= Label(v_9) \cup \{\neg p \wedge \neg q\} \\ Label(v_{13}) &= Label(v_{10}) \cup \{p \vee q\} \\ Label(v_{14}) &= Label(v_{10}) \cup \{\neg p \wedge \neg q\} \\ Label(v_{15}) &= Label(v_{11}) \cup Label(v_{13}) \\ Label(v_{16}) &= Label(v_{12}) \cup Label(v_{14}). \end{aligned}$$

  $ILConstraints(v_8)$ consists of $x_{v_i,e} \geq 0$, for $11 \leq i \leq 16$, and the following:

$$\begin{aligned} x_{v_{11},e} + x_{v_{12},e} + x_{v_{15},e} + x_{v_{16},e} &\geq 1 \\ x_{v_{13},e} + x_{v_{14},e} + x_{v_{15},e} + x_{v_{16},e} &\geq 1000 \\ x_{v_{11},e} + x_{v_{13},e} + x_{v_{15},e} &\leq 1000. \end{aligned}$$

- Consider the node v_{12}. To shorten the presentation, we ignore details about expansions for v_{12} and its descendants. We have $\{\langle A_1, 0\rangle p,\ [A_2, 1]\neg p,\ \neg p \wedge \neg q\} \subset Label(v_{12})$. It can be seen that there will not be any $\Diamond$-realization for $\langle A_1, 0\rangle p$ at v_{12} (there will be a cycle going through nodes with status different from unsat). As a consequence, $Status(v_{12})$ will be changed at some step to unsat by the $(unsat)$ rule.
- Consider the node v_{15}. We have $\{\langle A_1, 0\rangle p,\ [A_2, 1]\neg p,\ q,\ \neg p \vee \neg q\} \subset Label(v_{15})$. Similarly as for v_{12}, it can be seen that there will not be any $\Diamond$-realization for $\langle A_1, 0\rangle p$ at v_{15}. As a consequence, $Status(v_{15})$ will be changed at some step to unsat by the $(unsat)$ rule.
- Observe that $\{q, \neg p \wedge \neg q\} \subset Label(v_{14}) \subset Label(v_{16})$. Clearly, $Status(v_{14})$ and $Status(v_{16})$ will be changed at some steps to unsat.
- Consider the moment when the statuses of the nodes v_{12}, v_{14}, v_{15} and v_{16} have been changed to unsat and consider the set that extends $ILConstraints(v_8)$ with

$x_{v_i,e} = 0$ for $i \in \{12, 14, 15, 16\}$. This set is reduced to the following one w.r.t. feasibility:

$$\begin{aligned} x_{v_{11},e} &\geq 1 \\ x_{v_{13},e} &\geq 1000 \\ x_{v_{11},e} + x_{v_{13},e} &\leq 1000. \end{aligned}$$

Clearly, it is infeasible. As a consequence, $Status(v_8)$ is changed to unsat by the $(unsat)$ rule. By applying this rule in the propagation manner, the statuses of the nodes v_7, v_6, $v_4 - v_1$, ν are changed to unsat one after the other. According to Theorem 1, we claim that the set Γ is unsatisfiable.

5 Conclusions

We have given the first direct tableau decision procedure for GPDL, which has EXPTIME (optimal) complexity when numbers are encoded in unary. It uses global caching and exploits our technique of integer linear feasibility checking [9].

We have implemented our procedure in the scope of the reasoner TGC2.[4] This reasoner also allows converse modalities [13, 9] and ABoxes [14, 9]. As far as we know, it is the first reasoner that can decide GPDL.

Preliminary experiments with TGC2 showed that our method deals with number restrictions (graded modalities) much better than the well-known reasoners Racer, FaCT++, HermiT and Pellet for description logics.

Acknowledgments

This work was done in cooperation with the project 2011/02/A/HS1/00395, which is granted by the Polish National Science Centre (NCN).

References

1. Abate, P., Goré, R., Widmann, F.: An on-the-fly tableau-based decision procedure for PDL-satisfiability. Electr. Notes Theor. Comput. Sci. 231, 191–209 (2009)
2. De Giacomo, G., Lenzerini, M.: Boosting the correspondence between description logics and propositional dynamic logics. In: Proceedings of AAAI 1994. pp. 205–212. AAAI Press / The MIT Press (1994)
3. De Giacomo, G., Lenzerini, M.: What's in an aggregate: Foundations for description logics with tuples and sets. In: Proceedings of IJCAI 95. pp. 801–807. Morgan Kaufmann (1995)
4. Dunin-Kęplicz, B., Nguyen, L., Szałas, A.: Converse-PDL with regular inclusion axioms: a framework for MAS logics. Journal of Applied Non-Classical Logics 21(1), 61–91 (2011)
5. Fischer, M., Ladner, R.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. 18(2), 194–211 (1979)
6. Giacomo, G.D., Lenzerini, M.: TBox and ABox reasoning in expressive description logics. In: Proceedings of KR'1996. pp. 316–327 (1996)

[4] See http://www.mimuw.edu.pl/~nguyen/TGC2.

7. Goré, R., Widmann, F.: Optimal and cut-free tableaux for propositional dynamic logic with converse. In: Proceedings of IJCAR 2010. LNCS, vol. 6173, pp. 225–239. Springer (2010)
8. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press (2000)
9. Nguyen, L.: ExpTime tableaux for the description logic $\mathcal{SHIQ}$ based on global state caching and integer linear feasibility checking. arXiv:1205.5838 (2012)
10. Nguyen, L.: Cut-free ExpTime tableaux for Converse-PDL extended with regular inclusion axioms. In: Proc. of KES-AMSTA'2013. Frontiers in Artificial Intelligence and Applications, vol. 252, pp. 235–244. IOS Press (2013)
11. Nguyen, L.: Designing a tableau reasoner for description logics. In: Proc. of ICC-SAMA'2015. Advances in Intelligent Systems and Computing, vol. 358, pp. 321–333. Springer (2015)
12. Nguyen, L., Golińska-Pilarek, J.: An ExpTime tableau method for dealing with nominals and qualified number restrictions in deciding the description logic SHOQ. Fundam. Inform. 135(4), 433–449 (2014)
13. Nguyen, L., Szałas, A.: An optimal tableau decision procedure for Converse-PDL. In: Nguyen, N.T., Bui, T.D., Szczerbicki, E., Nguyen, N.B. (eds.) Proceedings of KSE'2009. pp. 207–214. IEEE Computer Society (2009)
14. Nguyen, L., Szałas, A.: Checking consistency of an ABox w.r.t. global assumptions in PDL. Fundamenta Informaticae 102(1), 97–113 (2010)
15. Pratt, V.: A near-optimal method for reasoning about action. J. Comput. Syst. Sci. 20(2), 231–254 (1980)
16. Tobies, S.: Complexity results and practical algorithms for logics in knowledge representation. Ph.D. thesis, RWTH-Aachen (2001)