Theoretical Computer Science 118 (1993) 67–79
Elsevier

Gentzen-type axiomatization for PAL

Igor Walukiewicz

Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warszawa, Poland

Abstract

Walukiewicz, I., Gentzen-type axiomatization for PAL, Theoretical Computer Science 118 (1993) 67–79.

The aim of propositional algorithmic logic (PAL) is to investigate the properties of simple nondeterministic while-program schemes on propositional level. We present finite, cut-free, Gentzen-type axiomatization of PAL. As a corollary from completeness theorem, we obtain the small-model theorem and algorithm for checking the validity of PAL formulas.

1. Introduction

Propositional algorithmic logic (PAL) was constructed by Mirkowska [4] as a propositional counterpart of algorithmic logic (AL) [7]. The aim of PAL is to investigate the properties of simple nondeterministic while-programs built upon the set of atomic actions on a propositional level abstracting from values of variables, functions, etc. As opposed to PDL, there are two nondual modalities. Modality $\diamond$ means (as in PDL) "there exists successful execution" and modality $\Box$ means "all executions are successful". With these two, we are able to construct formulas expressing important properties of programs, like termination, looping, partial and total correctness.

Lifting theorem brings another reason to investigate such logic. It is a theorem which says that tautologies of PAL become tautologies of AL after replacing program variables by programs and propositional variables by formulas [4]. The existence of an easy-to-implement decision procedure for PAL can help in attempts to construct an automated prover based on AL.

Correspondence to: I. Walukiewicz, Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warszawa, Poland. Email: igw@mimuw.edu.pl.

0304-3975/93/$06.00

Because of the existence of strong modality $\Box$, it was not obvious whether the small-model theorem holds or whether there exists finite axiomatization for PAL. It is a well-known fact that PAL, like PDL with loop construct, does not enjoy the collapsed-model property, i.e. the property which says that if a formula α is unsatisfied in some structure then it is unsatisfied in a structure which is constructed by merging states satisfying the same subformulas of α. The small-model theorem for PDL with loop construct was proved by Street [8], but there still remained a question whether the same theorem holds for PAL.

Infinite axiomatization of PAL was presented in Mirkowska [4]. Attempts to give finite Gentzen-type axiomatization of PDL were made by Nishimura [5, 6]. In this paper we present finite, cut-free Gentzen-type axiomatization for PAL. As a corollary, we obtain the small-model theorem and decision procedure for checking the validity of PAL formulas.

2. Syntax and semantics

The syntax of PAL is based on two sets of symbols: V_0, the set of propositional variables, and Π_0, the set of atomic programs. We will use $p, q, \ldots$ for the elements of V_0 and $A, B, \ldots$ for the elements of Π_0.

From V_0 we construct the set of open formulas F_0 as usual propositional formulas, i.e. $V_0 \in F_0$ and if $\alpha, \beta \in F_0$ then $\alpha \wedge \beta, \alpha \vee \beta, \neg \alpha \in F$.

Given sets F_0 and Π_0, the set Π of programs in generated by the following grammar:

$$\Pi ::= \Pi_0 \mid \Pi;\Pi \mid \textbf{if } F_0 \textbf{ then } \Pi \textbf{ else } \Pi \textbf{ fi} \mid \textbf{either } \Pi \textbf{ or } \Pi \textbf{ ro}$$

$$\mid \textbf{while } F_0 \textbf{ do } \Pi \textbf{ od}.$$

Finally, we define the set of formulas F:

$$F ::= F_0 \mid F \vee F \mid F \wedge F \mid \neg F \mid \Diamond \Pi F \mid \Box \Pi F.$$

We will often write o which will mean any modality, i.e. $\Box$ or $\Diamond$.

Formulas of PAL will be interpreted in the so-called Kripke structures, which have form $\langle S, R, \rho \rangle$, where

- S is a nonempty set of the so-called states,
- R is a function assigning to each atomic program $A \in \Pi_0$ a binary relation $R(A)$ on S,
- ρ assigns to each state $s \in S$ the subset of V_0.

Intuitively, R gives interpretation of atomic programs as actions changing states, ρ determines which propositional variables are true in a given state. Mapping ρ is easily extended to all members of F_0 in a usual propositional sense. In order to extend relation R to all programs in Π, we will provide the notion of execution.

Definition 2.1. Given structure $\mathfrak{I}=\langle S, R, \rho\rangle$, we define one step of execution relation $\triangleright$ between configurations, i.e. pairs (state, program) as follows:

(1) $(s, A)\triangleright(s', \varepsilon)$, where $A\in\Pi_0$ and $(s, s')\in R(A)$,

(2) $(s, K; L)\triangleright(s', K'; L)$ if $(s, K)\triangleright(s', K')$,

(3) $(s, K; L)\triangleright(s', L)$ if $(s, K)\triangleright(s', \varepsilon)$,

(4) $(s,$ **if** γ **then** K **else** L **fi**$)\triangleright(s, K)$ if $\gamma\in\rho(s)$ and (s, L) otherwise,

(5) $(s,$ **either** K **or** L **ro**$)\triangleright(s, K)$ and $(s,$ **either** K **or** L **ro**$)\triangleright(s, L)$,

(6) $(s,$ **while** γ **do** K **od**$)\triangleright(s, K;$ **while** γ **do** K **od**$)$ if $\gamma\in\rho(s)$ and (s, ε) otherwise, where ε is a special endmarker.

Execution of program K from state $s_0\in S$ is a sequence of configurations $(s_i, L_i)_{i\in I}$ such that $s_0=s$, $L_0=K$ and $(s_i, L_i)\triangleright(s_{i+1}, L_{i+1})$. We will call execution *successful* if it is finite and its last element is of the form (s, ε); state s will be called the *final state* of execution.

Now we are ready to formulate the notion of satisfiability of a PAL formula α in a state s of a structure $\mathfrak{I}$ ($\mathfrak{I}, s\models\alpha$):

(1) $\mathfrak{I}, s\models\alpha$ iff $\alpha\in\rho(s)$, for $\alpha\in F_0$;

(2) if $\alpha=\Diamond K\beta$ then $\mathfrak{I}, s\models\alpha$ iff there exists successful execution of program K from state s and in its final state $t, \mathfrak{I}, t\models\beta$;

(3) if $\alpha=\Box K\beta$ then $\mathfrak{I}, s\models\alpha$ iff every execution of program K is successful and in every final state $t, \mathfrak{I}, t\models\beta$;

(4) in the case of propositional connectives, as usual.

We will say that the formula α is $\mathfrak{I}$-*valid* ($\mathfrak{I}\models\alpha$) iff, for all states $s\in\mathfrak{I}, \mathfrak{I}, s\models\alpha$. Finally, we will say that α is valid ($\models\alpha$) iff for all structures $\mathfrak{I}, \mathfrak{I}\models\alpha$.

The main difference between PAL and logics derived from PDL lies in the interpretation of programs. In PAL, programs are interpreted as a set of possible sequences of executions and not as a binary relation on states. This gives the possibility to distinguish naturally infinite and unsuccessful executions. In PDL this problem was solved by adding repeat construct [8] and observation that one can code requirement that no execution of program is unsuccessful. PAL can be coded in μ-calculus in the same way as PDL with repeat. There is also the same as with PDL problem with possible exponential blowup, which disappears when one uses multiple fixpoints [9].

3. Sequents and Gentzen-type calculus

In this section we will present rules of our system and prove some simple facts about it. But first we introduce some basic notions of Gentzen-type calculi.

Sequent will be an ordered pair of finite sets of formulas. We will write a sequent as $\Gamma\to\Delta$ and call set Γ the *predecessor* of sequent and set Δ the *successor* of sequent $\Gamma\to\Delta$. We will say that a sequent $\Gamma\to\Delta$ is true in a state s of a structure $\mathfrak{I}$ ($\mathfrak{I}, s\models\Gamma\to\Delta$) iff in this state conjunction of formulas in Γ implies disjunction of formulas in Δ.

Our Gentzen-type system for PAL, GPAL, consists of the following rules:

- *Propositional*

(1) $$\frac{\Gamma\to\Delta,\alpha}{\neg\alpha,\Gamma\to\Delta} \qquad \frac{\alpha,\Gamma\to\Delta}{\Gamma\to\Delta,\neg\alpha}$$

(2) $$\frac{\alpha,\beta,\Gamma\to\Delta}{\alpha\wedge\beta,\Gamma\to\Delta} \qquad \frac{\Gamma\to\Delta,\alpha \mid \Gamma\to\Delta,\beta}{\Gamma\to\Delta,\alpha\wedge\beta}$$

(3) $$\frac{\alpha,\Gamma\to\Delta \mid \beta,\Gamma\to\Delta}{\alpha\vee\beta,\Gamma\to\Delta} \qquad \frac{\Gamma\to\Delta,\alpha,\beta}{\Gamma\to\Delta,\alpha\vee\beta}$$

- *Program*

(4) $$\frac{\circ K(\circ L\alpha),\Gamma\to\Delta}{\circ(K;L)\alpha,\Gamma\to\Delta} \qquad \frac{\Gamma\to\Delta,\circ K(\circ L\alpha)}{\Gamma\to\Delta,\circ(K;L)\alpha}$$

(5) $$\frac{\gamma,\circ K\alpha,\Gamma\to\Delta \mid \circ L\alpha,\Gamma\to\Delta,\gamma}{\circ\,\textbf{if}\ \gamma\ \textbf{then}\ K\ \textbf{else}\ L\ \textbf{fi}\ \alpha,\Gamma\to\Delta} \qquad \frac{\gamma,\Gamma\to\Delta,\circ K\alpha \mid \Gamma\to\Delta,\circ L\alpha,\gamma}{\Gamma\to\Delta,\circ\,\textbf{if}\ \gamma\ \textbf{then}\ K\ \textbf{else}\ L\ \textbf{fi}\ \alpha}$$

$$\frac{\alpha,\Gamma\to\Delta,\gamma \mid \gamma,\circ K(\circ\,\textbf{while}\ \gamma\ \textbf{do}\ K\ \textbf{od}\ \alpha),\Gamma\to\Delta}{\circ\,\textbf{while}\ \gamma\ \textbf{do}\ K\ \textbf{od}\ \alpha,\Gamma\to\Delta}$$

(6) $$\frac{\Gamma\to\Delta,\gamma,\alpha \mid \gamma,\Gamma\to\Delta,\circ K(\circ\,\textbf{while}\ \gamma\ \textbf{do}\ K\ \textbf{od}\ \alpha)}{\Gamma\to\Delta,\ \circ\,\textbf{while}\ \gamma\ \textbf{do}\ K\ \textbf{od}\ \alpha}$$

(7) $$\frac{\Diamond K\alpha,\Gamma\to\Delta \mid \Diamond L\alpha,\Gamma\to\Delta}{\Diamond\,\textbf{either}\ K\ \textbf{or}\ L\ \textbf{ro}\ \alpha,\Gamma\to\Delta} \qquad \frac{\Gamma\to\Delta,\Diamond K\alpha,\ \Diamond L\alpha}{\Gamma\to\Delta,\Diamond\,\textbf{either}\ K\ \textbf{or}\ L\ \textbf{ro}\ \alpha}$$

(8) $$\frac{\Box K\alpha,\Box L\alpha,\Gamma\to\Delta}{\Box\,\textbf{either}\ K\ \textbf{or}\ L\ \textbf{ro}\ \alpha,\Gamma\to\Delta} \qquad \frac{\Gamma\to\Delta,\Box K\alpha \mid \Gamma\to\Delta,\Box L\alpha}{\Gamma\to\Delta,\Box\,\textbf{either}\ K\ \textbf{or}\ L\ \textbf{ro}\ \alpha}$$

- Specific (in what follows, A is an atomic program)

(9) $$\frac{\{\alpha:\ \Box A\alpha\in\Gamma\},\beta\to\{\alpha:\ \Diamond A\alpha\in\Delta\}}{\circ A\beta,\Gamma\to\Delta},$$

(10) $$\frac{\{\alpha:\ \Box A\alpha\in\Gamma\}\to\{\alpha:\ \Diamond A\alpha\in\Delta\},\beta}{\circ A\gamma,\Gamma\to\Delta,\Box A\beta}.$$

Upper sequents of rules will be called *assumptions*, lower will be called *conclusion*. We will use the above rules in top–down fashion, i.e. given a sequent we are to prove, we will pick a rule which could have given such a conclusion and take its assumptions as next sequents to prove. We will call such a process an application of the rule.

Definition 3.1. Sequent $\Gamma \to \Delta$ will be called
unreducible iff there is no rule of GPAL applicable to $\Gamma \to \Delta$,
strongly unreducible iff there is no propositional or program rule applicable to $\Gamma \to \Delta$,
axiom iff $\Gamma \cap \Delta \neq \emptyset$,
simply refutable iff $\Gamma \to \Delta$ is an unreducible sequent but not an axiom.

Let us look at some simple properties of GPAL which will be of some use later.

Proposition 3.2. *Propositional and program rules are strong, i.e. for an arbitrary structure $\mathfrak{I}$ and state s of $\mathfrak{I}$, the conclusion of a rule is true in s iff all assumptions are true in s.*

Lemma 3.3 *Specific rules are sound, i.e. for an arbitrary structure $\mathfrak{I}$, if an assumption of a rule is $\mathfrak{I}$-valid then the conclusion is $\mathfrak{I}$-valid. Additionally, if $\Sigma \to \Omega$ is obtained from $\Gamma \to \Delta$ by reducing an atomic program A with some specific rule and if $\mathfrak{I}, s \not\models \Gamma \to \Delta$, then there exists a state t such that $(s,t) \in R(A)$ and $\mathfrak{I}, t \not\models \Sigma \to \Omega$.*

Proof. Let $\Sigma \to \Omega$ be an assumption and $\circ A\beta, \Gamma \to \Delta$ a conclusion of rule (9). Let us assume that, for some structure $\mathfrak{I} = \langle S, R, \rho \rangle$ and state s, $\mathfrak{I}, s \not\models \circ A\beta, \Gamma \to \Delta$. This means that $\mathfrak{I}, s \models \alpha$ for every formula α in predecessor and $\mathfrak{I}, s \not\models \alpha$ for $\alpha \in \Delta$. Hence, $\mathfrak{I}, s \models \circ A\beta$ and there exists a state t such that $(s,t) \in R(A)$ and $\mathfrak{I}, t \models \beta$. What is more, for every formula $\Box A\alpha \in \Gamma$, $\mathfrak{I}, t \models \alpha$ and for every $\Diamond A\alpha \in \Delta$, $A, t \not\models \alpha$. So, finally, we conclude that $\mathfrak{I}, t \not\models \Sigma \to \Omega$.

Proof for rule (10) is similar. □

We end this section by emphasizing the fact that GPAL rules are cut-free, which can be formulated more precisely as follows.

Definition 3.4. *Fisher–Ladner* closure of a formula α, $FL(\alpha)$ is the smallest set of formulas such that
(1) $\alpha \in FL(\alpha)$,
(2) if $\beta \in FL(\alpha)$ and the sequent $\beta \to$ can be reduced to $\Sigma \to \Omega$ then $\Sigma \cup \Omega \subseteq FL(\alpha)$.

Proposition 3.5. *For any formula α, $|FL(\alpha)| = \mathcal{O}(|\alpha|)$, where $|\alpha|$ is the number of symbols in α. For any sequent $\Gamma \to \Delta$, the number of all sequents which can be obtained from it by application of GPAL rules is less than $\mathcal{O}(2^{c|\Gamma \to \Delta|})$ for some constant c.*

4. Completeness

In this section we will prove the completeness theorem for GPAL. The first step will be to define what we mean by proof in our system. Essentially we will follow the

standard definitions, with one exception that the property of being final sequent will not only depend on the given sequent but also on its ancestors in a diagram. We could make this definition local by putting some information about ancestors to the sequent, but it would only make our system more obscure and hide intuitions behind it.

Definition 4.1. *Proof* of a sequent $\Gamma \to \Delta$ is a pair $\langle T, L \rangle$, where T is a finite tree and L a function assigning to each node of T a label (sequent) satisfying the following conditions:

(1) if n_0 is root of T then $L(n_0) = \Gamma \to \Delta$;

(2) for any internal node n, labels of its sons are assumptions and label of n is a conclusion in some rule of GPAL;

(3) node n is a leaf iff it is either a redundant node or is labeled with axiom. The notion of redundant node is defined below.

Hence, we are left of define what a redundant node is. To explain intuitions behind its definition, let us look at the sequent calculus for classical propositional logic. In this system we can treat proof construction as a process of checking that there is no valuation which does not satisfy a given sequent. For example, left-hand side rule (3) of our system, which is also a rule of system for propositional logic, can be explained as: a valuation which does not satisfy $\alpha \vee \beta, \Gamma \to \Delta$ must also not satisfy $\alpha, \Gamma \to \Delta$ or $\beta, \Gamma \to \Delta$. When we reach unreducible sequents, we know that we have checked all possibilities and if all leafs are axioms, we know that there is no such valuation. On the other hand, when we reach an unreducible sequent but not an axiom (called here a simply refutable sequent), we can directly construct a counterexample valuation.

Our system can be seen in a similar way, but with this complication that we are not only dealing with valuations but also with executions of programs. That is, we have to make sure that in our search we have looked also at any possible execution pattern. The same way as sequents served to describe valuations, trace relations we define below will describe execution patterns. So, when we can be sure that we have checked all execution patterns, i.e. "What is the concept which plays the same role as axiom but on the level of traces?" we will say that node n is *redundant* iff there are two ancestors of it, m_1, m_2 such that they are all labeled by the same sequent (represent the same valuation) and all execution patterns in interval (m_1, n) are the same as in (m_1, m_2). When such a situation occurs, everything we can do from node n we could have done from m_2; hence, there is no point of checking any further. We will also have a notion corresponding to simply refutable sequent, which will be *loop node*. As we will see, loop nodes play a crucial role in showing counterexample executions in case a sequent is not valid.

Let us try to formalize these definitions below.

Definition 4.2. Let a pair $\langle T, L \rangle$ satisfying conditions (1) and (2) of Definition 4.1 be given. We define the following:

– *specific node* is a node n such that a specific rule was applied to $L(n)$.

- *strong interval* is a part of path of T between two specific nodes not containing a specific node.
- If $L(n)=\Sigma\to\Omega$ then we will refer to Σ as $\Gamma(n)$ and to Ω as $\Delta(n)$.
- The *trace relation* $S_{m,n}$ for any node m and its descendant n, is a smallest relation satisfying the following properties:
  (1) If n is a son of m then
    (a) if formula $\alpha\in\Gamma(m)$ is not reduced by the rule applied to $L(m)$ then $(\alpha,\alpha)\in S_{m,n}$,
    (b) if formula $\circ M\beta\in\Gamma(n)$ is obtained from $\circ K\alpha\in\Gamma(m)$ then $(\circ K\alpha,\circ M\beta)\in S_{m,n}$.
  (2) In the other case, $S_{m,n}=S_{m,m'}\circ S_{m',n}$, where m' is a son of m belonging to the path from m to n.
- the *trace* of a formula $\alpha_m\in\Gamma(m)$ on a path P from m is a sequence $(\alpha_n)_{n\in P}$ such that $(\alpha_m,\alpha_n)\in S_{m,n}$.
- *Loop node* is a node $n\in T$ such that there exists a corresponding ancestor m, $L(m)=L(n)$, and $S_{m,n}=\emptyset$.
- *Redundant node* is a node m such that there exist ancestors m_1,m_2 such that $L(m_1)=L(m_2)=L(n)$, $S_{m_1,m_2}=S_{m_1,n}$, $\{L(k)\colon m_2<k<n\}\subseteq\{L(k)\colon k \text{ ancestor of } m_1\}$ and there is no loop node between m_1 and n.

The proofs of both soundness and completeness theorems for our system are somehow indirect. In both cases, rather than use the notion of proof, we will use a somewhat dual notion.

Definition 4.3. *Refutation tree* for a sequent $\Gamma\to\Delta$ will be a pair $\langle T,L\rangle$, where T is a finite tree and L a labeling of nodes of T with sequents such that
(1) the root of T is labeled with $\Gamma\to\Delta$,
(2) if n is an internal node then
  (a) if some propositional or program left-hand side rule R is applicable to $L(n)$ then n has exactly one son n' and $L(n')$ is one of the assumptions of rule R,
  (b) if some propositional or program right-hand side rule R is applicable to $L(n)$ and none of the assumptions is a label of a node in this strong interval then n has exactly one son n' and $L(n')$ is this assumption,
  (c) if none of the above conditions applies then there is one son for each possible application of specific rule to $L(n)$ labeled with the resulting assumption,
(3) if n is a leaf then either n is a loop node or $L(n)$ is a simply refutable sequent,
(4) none of the labels is an axiom.

The duality of the concepts of proof and refutation tree is expressed in the following lemma.

Lemma 4.4. *Sequent $\Gamma\to\Delta$ is unprovable iff there exists a refutation tree for $\Gamma\to\Delta$.*

Proof. An essential difference between these two concepts is that there are different rules for constructing the sons of an internal node and different conditions for a node to be a leaf. Let us construct a more general structure $\mathscr{D}$, which in some sense will encapsulate both the previous ones. The idea is that the sons of an internal node will be both those which we get by applying proof rules and those from refutation tree rules. We also allow both kinds of leafs in $\mathscr{D}$. To be more precise, structure $\mathscr{D}$ is defined to be a finite labeled tree satisfying the following conditions:

(1) the root of T is labeled with $\Gamma \to \Delta$;

(2) if n is an internal node then

(a) if some propositional or program left-hand side rule R is applicable to $L(n)$ then, for each assumption of R, there is a son of n labeled with it,

(b) if some propositional or program right-hand side rule R is applicable to $L(n)$ and none of the assumptions is a label of a node in this strong interval then, for each assumption of R, there is a son of n labeled with it,

(c) if none of the above conditions applies then there is one son for each possible application of a specific rule to $L(n)$ labeled with the resulting assumption;

(3) if n is a leaf then it is labeled either with an axiom or a simply refutable sequent or it is redundant or loop node.

It follows easily from Lemma 3.5 that such a structure exists for any sequent $\Gamma \to \Delta$. Given such a diagram $\mathscr{D}$, we proceed as follows:

- we mark every axiom or redundant node by 1 and other leaves by 0;
- for any internal node to which a strong rule was applied, we label it with 1 iff all sons are labeled with 1's; otherwise, it is labeled by 0;
- specific node is labeled by 1 iff there exists a son labeled by 1; otherwise, it is labeled by 0.

It is easy to see that if the root of $\mathscr{D}$ is labeled with 1 then we can find a proof diagram in $\mathscr{D}$; otherwise, we can find a refutation tree in $\mathscr{D}$. To make this proof complete, we have to show the following lemma.

Lemma 4.5. *If there exists a refutation tree then there exists a refutation tree without redundant nodes and, dually, if there exists a proof then there exists a proof without loop nodes.*

Proof. Suppose that there exists a refutation tree and there is a redundant node n on a path P of this tree, i.e. there are two ancestors of n, $m_1 < m_2$, such that $L(n) = L(m_1) = L(m_2)$, $S_{m_1,m_2} = S_{m_1,n}$ and $\{L(k)\colon m_2 < k < n\} \subseteq \{L(k)\colon k < m_1\}$. Then, as the name suggests some of this path is redundant, namely, we are able to connect the father of m_2 directly to n. Because of the last condition imposed on a set of labels, if some node was loop node then we can assume that the node corresponding to it is above m_1. From the condition imposed on traces, we know that, after such a cut, nodes which were loop nodes remain to be so. Hence, after this operation, what is left is still a refutation tree.

Proof of the second part of the lemma is analogous.

The above lemma makes it possible to formulate the soundness and completeness properties using the notion of refutation tree rather than proof. First, let us try it on the soundness theorem.

Lemma 4.6. *For any sequent $\Gamma\to\Delta$, if $\not\models \Gamma\to\Delta$ then there exists a refutation tree for $\Gamma\to\Delta$.*

Proof. For this proof, we introduce a little more general concept than that of execution of a program.

Definition 4.7. If $\alpha = \circ K_1(\ldots(\circ K_n\beta)\ldots)$ and β is not of the form $\circ L\gamma$ then *realization of program in formula* α from state s ($Rel(\alpha, s)$) is the shortest successful execution of program $K_1;\cdots;K_n$ from state s.

From the assumption $\not\models \Gamma\to\Delta$ it follows that there exists a structure $\mathfrak{I}=\langle S, R, \rho\rangle$ and state s_0 such that $\mathfrak{I}, s_0 \not\models \Gamma\to\Delta$. We will construct a refutation tree $\mathfrak{R}$ for $\Gamma\to\Delta$, assigning to each node n of it state s_n of $\mathfrak{I}$ such that $\mathfrak{I}, s_n \not\models L(n)$. With the root n_0 of $\mathfrak{R}$, we associate a state $s_{n_0}=s_0$. For any internal node of $\mathfrak{R}$ with associated state s_n, we proceed in the following way:

- If we are to apply some strong rule R to $L(n)$ then from Proposition 3.2 it follows that one of the assumptions of R, say $\Sigma\to\Omega$, is unsatisfiable in state s_n. Hence, we can take as a label of the only son of n, m sequent $\Sigma\to\Omega$ and set $s_m=s_n$. This strategy is fine for all cases except when we are to reduce a formula of the form $\diamond$ **either** K **or** L **ro** α in the predecessor of the sequent. In this case, if both the obtained assumptions are unsatisfiable, we choose the assumption with $\diamond K\alpha$ if $Rel(\diamond K\alpha, s_n)$ is shorter than $Rel(\diamond L\alpha, s_n)$ and another assumption otherwise.
- if we are to apply strong rules then from Lemma 3.3 it follows that, for every possible assumption $\Sigma\to\Omega$, there exists a state t such that $(s_n, t)\in R(A)$ for an appropriate program A and $\mathfrak{I}, t\models\Sigma\to\Omega$. Of course, we can take t as a state associated with a son of n labeled by $\Sigma\to\Omega$, but again we face the problem of nondeterminism. Only in one case, when we reduce a fomula of the form $\diamond A\alpha\in\Gamma(n)$, does it matter what state we choose. In this case we choose state t to be the one which is next in $Rel(\diamond A\alpha, s_n)$.

Structure $\mathfrak{R}$ constructed in such a way clearly satisfies all conditions imposed on a refutation, except for the one which requires $\mathfrak{R}$ to be finite. Our $\mathfrak{R}$ may have infinite paths, but the following lemma will show that we can cut $\mathfrak{R}$ to a refutation tree.

Lemma 4.8. *For every node n, formula $\alpha\in\Gamma(n)$ and path P from node n, every trace of α is finite.*

Proof. First let us observe that if $\circ K\beta\in\Gamma(n)$ and $(\circ K\beta, \circ M_i\beta)\in S_{n,m_i}$, for $i\in I$, is a part of a trace of formula $\circ K\beta$ then it is finite. Indeed, looking at the corresponding sequences $(s_{m_i}, M_i)_{i\in I}$, it is easy to see that it constitutes a part of an execution of

a program K from state s_n. Hence, we are done if $\circ \equiv \Box$. When $\circ \equiv \Diamond$, one more observation that we need is that this distinguished execution is a part of the shortest execution of successive programs in $\Diamond K\beta$ from s_n; hence, since $s_n \models \Diamond K\beta$, it must be finite. With this observation, we prove the lemma by induction on the structure of α.
– In case of propositional variables or boolean connectives, from definition of trace relation it follows that the occurrence of such a formula is traced as long as it is not reduced. Hence, it is contained in one strong interval. Suppose that such an interval is infinite; then it follows immediately that there must exist a formula of the form $\circ$ **while** γ **do** K **od** α in the predecessor of some sequent which is reduced infinitely often. But, from what was stated above, this is impossible.
– If $\alpha = \Diamond K\beta \in \Gamma(n)$ then let us assume conversely that the trace of α from node n is infinite. By induction hypothesis, every trace of formula β is finite; hence, infinite trace must have occurred because of infinite reductions of program K. That means that there must be an infinite sequence of formulas $\Diamond M_i\beta$, $i \in I$, such that $(\circ K\beta, \circ M_i\beta) \in S_{n,m_i}$, which contradicts our first observation.
– If $\alpha = \Box K\beta \in L(n)$ then the proof is as above.

Let us now take any infinite path P of $\mathfrak{R}$. It follows easily from Proposition 3.5 that there are only finitely many different sequents which can occur as a label in $\mathfrak{R}$. Hence, there must be a sequent $\Gamma \to \Delta$ which occurs infinitely many times on P. From Lemma 4.8 it follows that there must be two occurrences of $\Gamma \to \Delta$, m and n, such that $S_{m,n} = \emptyset$; hence, n is a loop node and we can cut path P at this point. □

Now we will turn to the completeness of GPAL system. Our goal will be to show that, given a refutation tree for $\Gamma \to \Delta$, we can construct a structure in which it is unsatisfiable.

Definition 4.9. Given a refutation tree $\mathfrak{R} = \langle T, L \rangle$, we define a canonical structure $\mathfrak{I} = \langle S, R, \rho \rangle$ in the following way:
- $S = \{s \in T: s \text{ a specific node}\}$.
- $(s,t) \in R(A)$ iff there exists a son of s, r, such that $L(r)$ is obtained by reducing program A and r is either in the same strong interval as t or in the same strong interval as a leaf n whose corresponding node m is in the same strong interval as t.
- $p \in \rho(s)$ iff $p \in \Gamma(s)$.

We will show that the root of refutation tree $\mathfrak{R}$ is unsatisfied in the root of a canonical structure built in the way described above.

Lemma 4.10. *If a sequent $\Gamma \to \Delta$ has a refutation tree $\mathfrak{R} = \langle T, L \rangle$ then it is unsatisfied in the canonical structure $\mathfrak{I} = \langle S, R, \rho \rangle$ built from $\mathfrak{R}$.*

Proof. First of all, let us introduce a new definition which will be very helpful in this proof. In what follows we will write $\Gamma^*(i)$ to denote the set $\bigcup\{\Gamma(s): s \text{ in the same strong interval as } i\}$ and $\Delta^*(i)$ for $\bigcup\{\Delta(s): s \text{ in the same strong interval as } i\}$.

Definition 4.11. Let us take a state s of $\mathfrak{I}$ and a formula $\circ K\alpha \in \Gamma^*(s)$ ($\circ K\alpha \in \Delta^*(s)$). A traced execution of program K from s will be such an execution $(s_i, M_i)_{i\in I}$ of K from s that every formula $\circ M_i\alpha \in \Gamma^*(s_i)$ ($\circ M_i\alpha \in \Delta^*(s_i)$, respectively).

The usefulness of this notion is shown by the following lemma.

Lemma 4.12. *For a state s of $\mathfrak{I}$ and formula $\circ K\alpha \in \Gamma^*(s)$, any traced execution of program K is successful and in its final state t formula $\alpha \in \Gamma^*(t)$.*

Proof. Suppose, conversely, that there exists an infinite traced execution of K. Then, since $\mathfrak{I}$ is finite, there must exist a state s such that it occurs at least twice in this execution. What is more, there must exist a loop node n of tree $\mathfrak{R}$ such that node m corresponding to n is in the same strong interval as s and s appeared at least twice in the execution. Then, because this execution was a traced one, we obtain that $S_{s,n} \neq \emptyset$, contradicting the definition of a loop node. □

Lemma 4.10 follows from the next lemma.

Lemma 4.13. *If $\alpha \in \Gamma^*(s)$ then $\mathfrak{I}, s \models \alpha$ and if $\alpha \in \Delta^*(s)$ then $\mathfrak{I}, s \not\models \alpha$.*

Proof. Proof will proceed by induction on formula α:
- If $\alpha \in \Gamma^*(s)$ is a propositional variable then it is easy to observe that $\alpha \in \Gamma(s)$; hence, from definition of ρ, $\mathfrak{I}, s \models \alpha$.
- Other cases with propositional connectives and dual one for $\alpha \in \Delta^*(s)$ follow immediately from the form of rules.
- If $\Box K\alpha \in \Gamma^*(s)$ then it follows easily from the definition of a canonical structure and form of the rules that every execution of program K from state s is traced. Then $\mathfrak{I}, s \models \Box K\alpha$ follows from Lemma 4.12 and the induction hypothesis.
- If $\Diamond K\alpha \in \Gamma^*(s)$ then we observe that there exists a traced execution of K from s. Then, as in case above, we obtain that $\mathfrak{I}, s \models \Diamond K\alpha$.
- If $\Box K\alpha \in \Delta^*(s)$ then we show that there exists a traced execution of K from s. If it is successful then in its final state t, $\beta \in \Delta^*(t)$, but from the induction hypothesis $\mathfrak{I}, s \not\models \alpha$; hence, $\mathfrak{I}, s \not\models \Box K\alpha$.
- If $\Diamond K\alpha \in \Delta^*(s)$ then we observe that every execution of K from state s is traced. Hence, in the final state of every successful execution formula $\beta \in \Delta^*(t)$.

From Lemma 4.13 it follows immediately that in state s, which belongs to the same strong interval as the root of $\mathfrak{R}$, the label of the root $\Gamma \to \Delta$ is unsatisfied. □

Summarizing the results of Lemmas 4.5 and 4.7, we obtain the following theorem.

Theorem 4.14 (Soundness and completeness). *For any sequent $\Gamma \to \Delta$, $\Gamma \to \Delta$ is valid iff $\Gamma \to \Delta$ is provable.*

5. Algorithm

Having proved the completeness theorem, we can use Lemma 4.4 to get an algorithm for checking the validity of PAL sequents. It can be described simply as:

given a sequent, construct a structure $\mathfrak{D}$ as described in Lemma 4.4.

Note that this algorithm always stops because, by Lemma 3.5, the degree of every node as well as the length of every path is bounded. This procedure gives as the result either a proof of the validity or a refutation tree which can be transformed to a counterexample structure using the construction from Definition 4.9.

It is easily seen that the complexity of the above algorithm is the same as the size of a constructed structure. The degree of each node in the structure is bounded by the number of different formulas which a sequent can contain, and, hence, by $FL(\Gamma\to\Delta)=\mathcal{O}(n)$. Because we are not expanding further than redundant node, we get an upper bound on the path length. Namely, each sequent can occur on one path no more times than there are different trace relations. Since there are $\mathcal{O}(n)$ formulas which can occur in construction, there are at most $\mathcal{O}(2^n)$ different sequents and $\mathcal{O}(2^{n^2})$ trace relations. Hence, the length of the path is bound by $\mathcal{O}(2^{cn^2})$ and the whole structure by $\mathcal{O}(n^{2^{cn^2}})$ for some constant c. This leads to the following theorem.

Theorem 5.1 (Small model). *For every not-valid PAL formula α, there exists a model $\mathfrak{I}$ such that $\mathfrak{I}\not\models\alpha$ and the size of $\mathfrak{I}$ is less than $\mathcal{O}(n^{2^{cn^2}})$, where $n=|\alpha|$.*

6. Conclusions

We have presented a Gentzen-type system for propositional algorithmic logic and proved its completeness. Basing on this system, we have obtained decision procedure which outputs either a proof or a counterexample structure. Finally, we have obtained the small-model theorem which assures that, for every not-valid sequent $\Gamma\to\Delta$, there exists a counterexample model of size bounded by a double exponential function of size of $\Gamma\to\Delta$.

Acknowledgment

I thank Prof. Grażyna Mirkowska for inspiration, patience and very valuable help.

References

[1] B. Chlebus, On the decidability of propositional algorithmic logic, *Z. Math. Logik* **28** (1982) 247–261.
[2] M.J. Fisher and R.E. Ladner, Propositional dynamic logic of regular programs, *J. Comput. System Sci.* **18** (1979) 194–211.

[3] D. Harel, Dynamic Logic, *Handbook of Philosophical Logic*, Vol. 11 (1984) pp. 197–604.
[4] G. Mirkowska, PAL – propositional algorithmic logic, *Fund. Inform.* **4** (1981) 675–760. Also in Lecture Notes in Computer Science Vol. 125 (1981) 23–101.
[5] H. Nishimura, Sequential method in propositional dynamic logic, *Acta Inform.* **2** (1979) 377–400.
[6] H. Nishimura, Semantical analysis of constructive PDL, *Publ. Res. Inst. Math. Sci.* (Kyoto Univ.) (1981).
[7] A. Salwicki, Formalized algorithmic languages, *Bull. Acad. Polon Sci. Ser Sci. Math. Astron. Phys.* **18** (1970) 227–232.
[8] R.S. Street, Propositional dynamic logic of looping and converse is elementary decidable, *Inform. and Control* **54** (1982) 121–141.
[9] M. Vardi and P. Wolper, Automata theoretic techniques for modal logics of programs, in: *Proc. 16th ACM Symp. on the Theory of Computing* (ACM, 1984) 445–446.