Note

Determinism and non-determinism in PDL

Andreas Wilm

Philosophisches Seminar (Abteilung Logik), Christian-Albrechts-Universität zu Kiel, Germany

Communicated by E. Jäger
Received April 1990
Revised December 1990

Abstract


The main purpose of this paper is to consider a variant of Propositional Dynamic Logic (PDL) in which the natural program concepts \texttt{while do od} and \texttt{if then else fi} are basic and in which \texttt{u} is the only non-deterministic program concept. Dropping \texttt{u} leads to the known system Strict Propositional Dynamic Logic (SPDL). The main result of this paper is that SPDL+\texttt{u} is weaker than PDL. It is pointed out, however, that by extending SPDL+\texttt{u} by propositional assignments a system called Propositional Program Logic (PPL) is obtained which has the same expressive power as PDL.

1. Introduction

\textit{Dynamic Logic} is a special family of languages for reasoning about computer programs. A formal apparatus for this logic was introduced as “modal logic of programs” in the fundamental work of Pratt [10] and developed further by Harel in [5] where he presented some first-order versions. First-order Dynamic Logic has high expressive power and is highly undecidable. For this reason Fischer and Ladner introduced a \textit{propositional} version in [2], the system PDL (“Propositional Dynamic Logic”). PDL is a combination of propositional \textit{modal logic} with elements of \textit{regular languages}, a class of formal languages.

PDL may be understood as a \textit{program logic}, because elements of sequential program languages can be formalized and interpreted by PDL’s syntactical elements for programs \texttt{;} *, \texttt{u}, and \texttt{?}; (program) concepts for short. The concept \texttt{;} stands for
concatenation of programs and is basic for sequential programming. The concepts *, ∪, and ? are non-deterministic and do not correspond to elements of current program languages, containing rather deterministic program concepts like while do od and if then else fi allowing structured programming. However, it is the relational semantics of PDL that makes it possible to simulate while do od and if then else fi in PDL by ;;, *, ∪, and ?. It is not possible to simulate the latter concepts with the former ones because the system SPDL ("Strict PDL") containing the program concepts ;;, while do od, if then else fi, and the program constants skip and abort instead of *, ∪, and ? has less expressive power than PDL (see [7]).

The well-known simulation of while do od and if then else fi by ;;, *, ∪. and ? is presented in Lemma 1. This simulation is correct for a relational semantics having a relation between initial and final states as value for a program. But it becomes unsatisfactory in a semantics that emphasizes the run of a program. In such a path semantics, the value of a program is a set of sequences of states including aborting and infinite program runs without a proper final states. There are examples for path semantics in [5] ("computation trees") and in [11] ("trajectories"). Simulation of while do od and if then else fi by Lemma 1 in these semantics leads to additional aborting program runs which are not intended and which are therefore senseless. So the simulation of while do od and if then else fi in PDL is unnatural from the procedural point of view. For this reason, in [5, 11] the procedural evaluation of programs is extended by "correction rules" to avoid senseless program runs. But the "correction rules" are intuitively less understandable than the evaluation of while do od and if then else fi because they rest upon the principle "first decide, then test" instead of the realistic principle "first test, then decide" that is more usual in structured programming.

A program logic should rest upon this realistic principle. Hence, SPDL would be a better candidate for a program logic than PDL without the problem of less expressive power. To solve this problem, SPDL has to be extended by at least one non-deterministic concept. Of course, SPDL+*+∪ would then be a trivial solution. But the question remains whether both * and ∪ are really necessary to get the expressive power of PDL. In this paper, another extension called PPL ("Propositional Program Logic") following [3] is presented which gives an answer to this additional question. PPL only contains ∪ as non-deterministic program concept and has (in principle) the expressive power of PDL by the means of propositional assignments.

Before PPL is presented in this paper, two intermediate systems between SPDL and PDL, SPDL+* and SPDL+∪, are considered. It is rather easy to show that SPDL+* has the same expressive power as PDL (Theorem 1). However, SPDL+* is less appealing since it contains the highly non-deterministic *. The main result of the paper is that the more natural system SPDL+∪ has less expressive power than PDL (Theorem 2). Expressive power is usually defined with respect to formulas and differs from expressive power which is defined with respect to programs. In Theorems 1 and 2 assertions are made about both kinds of expressive power. Finally,
Theorem 3 says that the syntactic and semantic extension of SPDL + θ with propositional assignments, PPL, leads to a system having (in principle) the expressive power of PDL.

Value assignments \( v := t \) (\( v \) is a variable, \( t \) is a term) are basic in most programming languages. They have been incorporated into first-order Dynamic Logic in [5] and [3]. Assignments can be used with any type of variable. Propositional assignments use propositional variables and Boolean values as terms. Whereas assignments are typical for a first-order situation, propositional assignments keep the propositional character. Thus it is obvious to include them in propositional systems of Dynamic Logic.

The extension of SPDL by \( \cup \) and propositional assignments enables the simulation of the program concept \( \ast \) by \texttt{while do od}. This possibility has been used in [9] to show that loop may be expressed in first-order systems of Dynamic Logic. In spite of this early work there are only a few attempts in the literature to integrate propositional assignments within propositional versions of Dynamic Logic, for example in [13]. Propositional assignments in the sense of this paper are called \textit{global assignments} in [13]. Supplying an important part for the proof of Theorem 3, it is shown there that global assignments can be eliminated in PDL (see Lemma 3 in this paper). However, the main result of [13] is that \textit{local assignments} extend PDL into a system of high expressive power. With these program concepts, marking of states is possible. Therefore, the extension leads to a system already possessing many of the attributes of first-order Dynamic Logic.

Further, the difference between [13] and this paper is the starting point: Choosing SPDL as basic system in this paper enables to avoid the difficulties resulting from a path semantics for PDL.

### 2. Syntax and semantics of PDL and SPDL

The syntax of Dynamic Logic includes a set \( \mathcal{N} \) of formulas and a set \( \mathcal{P} \) of programs. The two sets are defined by simultaneous induction, leading to a typical pattern of proofs. The set \( \mathcal{N} \) of PDL and SPDL is given by a set \( \mathcal{N}_0 \) of propositional constants and by the rules

\[
\begin{align*}
(1) \quad (i) \quad & \mathcal{N}_0 \subseteq \mathcal{N}, \\
& \text{true} \in \mathcal{N}, \quad \text{false} \in \mathcal{N}, \\
& \text{(iii) if } F, G \in \mathcal{N}, \text{ then } \lnot F \in \mathcal{N} \text{ and } F \lor G \in \mathcal{N}, \\
& \text{(iv) if } F \in \mathcal{N} \text{ and } \alpha \in \mathcal{P}, \text{ then } [\alpha]F \in \mathcal{N}.
\end{align*}
\]

The set \( \mathcal{P} \) of PDL and SPDL is given by a set \( \mathcal{P}_0 \) of atomic programs, and for PDL by the rules

\[
\begin{align*}
(2a) \quad (i) \quad & \mathcal{P}_0 = \mathcal{P}, \\
& \text{(ii) if } \alpha, \beta \in \mathcal{P}, \text{ then } \alpha;\beta \in \mathcal{P}, \alpha^* \in \mathcal{P} \text{ and } \alpha \cup \beta \in \mathcal{P}, \\
& \text{(iii) if } F \in \mathcal{N}, \text{ then } F? \in \mathcal{P}.
\end{align*}
\]

and for SPDL by the rules
(2b) (i) $\mathcal{P}_0 \subset \mathcal{P}$.

(ii) $\text{skip} \in \mathcal{P}$, $\text{abort} \in \mathcal{P}$.

(iii) if $\alpha, \beta \in \mathcal{P}$ and $F \in \mathcal{S}$, then $\alpha; \beta \in \mathcal{P}$, while $F \text{ do } \alpha \text{ od } \in \mathcal{P}$ and if $F$ then $\alpha$ else $\beta$ fi $\in \mathcal{P}$.

Additional logical connectives as $\land, \rightarrow$, and $\leftrightarrow$ are abbreviated as usual. The modal operator $()$ being dual to the modal operator $[ ]$ is defined by $(\alpha)F = [\lnot[a]\lnot F]$.

The languages of PDL and SPDL are interpreted in models related to models of modal logical Kripke semantics. A model $M = (S, \rho, \pi)$ for PDL or SPDL is given by a set $S$ of states, where $\varphi : \mathcal{S}_0 \rightarrow 2^S$ assigns to any propositional constant $c$ a set of states in which $c$ is true, and $\pi : \mathcal{S}_0 \rightarrow 2^{S \times S}$ assigns to any atomic program $p$ a relation on $S$ determining the transitions of $p$.

The mappings $\varphi$ and $\pi$ are now extended to define the value of an arbitrary formula in a state $s \in S$ and the transition of an arbitrary program. For that purpose a relation of satisfaction $\models$ for any formula and, simultaneously, a relation of transition $R_a$ for any program $\alpha$ are defined. Every element of $R_a$ is a pair of states, initial and final state, determining a transition of $\alpha$. Since programs are interpreted by relations this semantics is called relational semantics. For formulas of PDL or SPDL, $\models$ is defined by

(3) (i) $M, s \models c$ if and only if $s \in \varphi(c)$, for $c \in \mathcal{S}_0$,

(ii) $M, s \models \text{true}$ and $M, s \not\models \text{false}$,

(iii) $M, s \models [\alpha]F$ if and only for all $s' \in S$, $s R_a s'$ implies $M, s' \models F$.

The relations $R_a$ for PDL-programs $\alpha$ are given by

(4a) (i) $R_p = [\pi(p)]$ for $p \in \mathcal{S}_0$,

(ii) $R_{\alpha; \beta} = [\alpha] R_{\beta}$ and $R_{\beta; \gamma} = [\beta] R_{\gamma}$

$= \{(s, s') \mid \text{there is } s'' \in S \text{ such that } (s, s'') \in R_{\beta} \text{ and } (s'', s') \in R_{\gamma}\}$,

$R_{\alpha} = \{(s, s') \mid \text{there is } n \in \mathbb{N} \text{ and a sequence } s_0, \ldots, s_n \in S \text{ such that } s = s_0$, $s' = s_n$, and $s_j R_{\beta} s_{j+1} \text{ for } 0 \leq j < n\}$,

$R_{\beta; \gamma} = [\beta] R_{\gamma}$,

(iii) $R_{E; \emptyset} = [\emptyset]\{s, s'\} [M, s \models E\}$.

The relations $R_a$ for SPDL-programs $\alpha$ are given by

(4b) (i) $R_p = [\pi(p)]$ for $p \in \mathcal{S}_0$,

(ii) $R_{\text{skip}} = \text{id}_S$,

$R_{\text{abort}} = \emptyset$,

(iii) $R_{\alpha; \beta} = [\beta] R_{\alpha; \beta} \\
R_{\text{while } E \text{ do } \beta \text{ od } = [\beta]\{(s, s') \mid \text{there is } n \in \mathbb{N} \text{ and a sequence } s_0, \ldots, s_n \in S \text{ such that } s = s_0$, $s' = s_n$, $s_j R_{\beta} s_{j+1}$, and $M, s_j \models E$, for $0 \leq j < n$ and $M, s_n \models \lnot E\}$,

$R_{\text{if } E \text{ then } \beta \text{ else } \gamma \text{ fi } = [\beta]\{(s, s') \mid s R_{\beta} s' \text{ and } M, s \models E\}$

$\cup [\gamma]\{(s, s') \mid s R_{\gamma} s' \text{ and } M, s \models \lnot E\}$.

The programs $F?$, for $F \in \mathcal{S}$, are called tests. A program $\alpha$ terminates in $s \in S$ if there is $s' \in S$ such that $s R_a s'$. This is equivalent to $M, s \models (\alpha)\text{true}$. Additionally,
for any system of propositional Dynamic Logic and for any model $\mathcal{M}$ the following are defined:

- formula $F$ is true in $\mathcal{M}$ and $s \in S$ iff $\mathcal{M}, s \models F$;
- formula $F$ is valid in $\mathcal{M}$, $\mathcal{M} \models F$, iff $\mathcal{M}, s \models F$ for all $s \in S$;
- formula $F$ is valid, $\models F$, iff $\mathcal{M} \models F$ for all models $\mathcal{M}$.

Since the classes of models for PDL and SPDL are identical the program concepts of both systems and the programs constructed with them are comparable. The following lemma lists some simple PDL- and SPDL-programs that are identical with respect to the relational semantics.

**Lemma 1.**

(a) $R_{(\neg E;\alpha)^*}; \neg E; R_{\text{while } E \text{ do } \alpha \text{ od}}$

(b) $R_{E}; R_{\text{if } E \text{ then skip else abort fi}}$

Proof. Directly from definition (4). □

The following example shows that the simulation of while do od and if then else fi by the means of Lemma 1(a) is not correct if aborting program runs are considered semantically. Such runs are marked with an additional state, usually symbolized by $\Lambda$ (see for instance [11]). In the model

```
\begin{array}{ccc}
  c & \neg c \\
  \bullet & p & \bullet \\
  s & \rightarrow & c \in \Sigma_0, p \in \Psi_0 \\
  s' & & \\
\end{array}
```

the program while cdop od has the only run $(s, s')$ in the state $s$ whereas the simulation $(c?;p)^*; \neg c?$ also generates the runs $(s, \Lambda)$ and $(s, s', \Lambda)$. Accordingly, the program ifthen else pfi has the only run $(s, s')$, and the program $(c?;p) \cup (\neg c?;p)$ has the additional run $(s, \Lambda)$.

PDL and SPDL are complete and decidable. Axioms and rules for PDL were introduced in [12], correctness follows directly from the definitions (3) and (4). The completeness of PDL was proven in various works in different ways, a combined proof of completeness and decidability of PDL is presented for instance in [6]. Axioms and rules for SPDL are found in [3], completeness and decidability especially of SPDL are proved in [4].

3. Expressive power of SPDL+* and SPDL+∪

In [7] it is shown that for $p, q \in \Psi_0$ the PDL-formula

$$[(p \cup q)^*](p)\text{true} \wedge (q)\text{true}$$

cannot be expressed in SPDL, i.e., PDL has more expressive power than SPDL. The advantage of SPDL to PDL regarding path semantics is diminished decisively
by this result. The question arises how the expressive power of SPDL might be increased by extending this system by non-deterministic or deterministic program concepts.

It is easily shown that the following two formulas are equivalent to (5):

\[(6) \quad \text{while } ((p)\text{true} \land (q)\text{true}) \text{ do } (p \cup q) \text{ od} \text{false}\]

\[(7) \quad [(p^*; q^*)][(p)\text{true} \land (q)\text{true}).\]

That means that already the systems SPDL+* and SPDL+∪, the extensions of SPDL by * and ∪, respectively, have more expressive power than SPDL. In [7] this conclusion is not mentioned, nor is there a reference to the question whether even PDL is more expressive than SPDL+* and SPDL+∪. This question is answered by the following two theorems.

**Theorem 1.** (a) The program concept ∪ cannot be simulated in SPDL+*.

(b) SPDL+* is equivalent in its expressive power to PDL.

**Proof.** A detailed proof of this theorem is found in [14]. For part (a) a simple model is presented in which no program of SPDL+* has the same transitions as the PDL-program \( p \cup q \) for \( p, q \in \mathcal{P}_0 \). Part (b) is shown by induction on the structure of formulas and programs by using \( R_{(\alpha \cup \beta)^*} = R_{(\alpha^*; \beta^*)^*} \) and \( \models [\alpha \cup \beta]F \leftrightarrow [\alpha]F \land [\beta]F. \)

Part (a) says that SPDL+* has less expressive power than PDL when the expressiveness of programs is considered only. This assertion is true for SPDL+∪, too, as stated in part (a) of the second theorem:

**Theorem 2.** (a) The program concept * cannot be simulated in SPDL+∪.

(b) SPDL+∪ has less expressive power than PDL.

Part (a) in Theorem 2 directly follows from part (b). To prove part (b), some preliminaries are necessary. First, an infinite class \( \mathcal{M} \) of models is defined. Each model of \( \mathcal{M} \) is composed of an infinity of "segment models" of the same type, thus each model of \( \mathcal{M} \) is an infinite model. This idea has been inspired by a proof technique used in [8]. Now, for all \( d \in \mathbb{N} \), a segment model \( \mathcal{S}_d \) is defined in the way shown in Fig. 1. The transitions of \( p, q \in \mathcal{P}_0 \) are marked by arrows, transitions of

![Fig. 1.](image-url)
other atomic programs do not exist, i.e. \( \pi(r) = \emptyset \) for all \( r \in \mathcal{B}_0 \) such that \( r \neq p, q \). Furthermore, for all \( c \in \mathcal{R}_0 \), \( \varphi(c) = \{s_0, \ldots, s_{d+1}\} \) holds, i.e. all propositional constants are true in all states of \( \Xi_d \). For all \( d \in \mathbb{N} \), infinite concatenation of segment models \( \Xi_d \) with completion by an additional transition of the program \( q \) now leads to an infinite model \( \mathcal{M}_d \), making up the class \( \mathcal{M} = \{ \mathcal{M}_d | d \in \mathbb{N} \} \) (see Fig. 2). Here, for all \( d, t \in \mathbb{N} \), \( s_0^{t+1} = s_{d+1}^t \). For the additional state \( \tilde{s} \), \( \tilde{s} \models [p \cup q]false \). However, from all other states, transitions of \( p \) and \( q \) are possible, i.e. for all \( d, t \in \mathbb{N} \) and \( h \leq d + 1 \), \( \mathcal{M}_d, s_h \models \langle p \rangle true \wedge \langle q \rangle true \). Furthermore, for all \( d, t \in \mathbb{N} \) and \( h \leq d + 1 \) and for each program \( while E do \alpha od \) of \( \text{SPDL}^+ \cup \) we define

\[
S_h^{d}(E, \alpha) = \{ s | s \models R \text{ while } E do \alpha \text{ od } s \text{ in } \mathcal{M}_d \}.
\]

The models of \( \mathcal{M} \) are constructed in such a way that the PDL-program \( (p^*; q; p^*; q)^* \) has a transition to state \( \tilde{s} \) from all states \( s_t^* \) where \( t \) is odd, while it cannot reach \( \tilde{s} \) in states \( s_t^* \) where \( t \) is even. This property holds for all \( d \in \mathbb{N} \) and is due to the nested \( * \)-programs of that program. It can be characterized by the PDL-formula \( ((p^*; q; p^*; q)^*)[p \cup q]false \) which is true in odd states and false in even states.

![Fig. 2.](image)

The following lemma shows that there is no formula of \( \text{SPDL}^+ \cup \) expressing this property of \( (p^*; q; p^*; q)^* \). The reason is the special definition of transitions of \( \text{while do od} \)-programs. Each transition of a program \( \text{while } E \text{ do } \alpha \text{ od } \) that is not equivalent to a transition of \( \text{skip} \) is characterized by a different evaluation of the test \( E \) in the initial and final state. Hence, there are no proper transitions between states in which all formulas have the same value. To simplify the proof, a normal form of programs is introduced. It is easily shown (see for instance [14]) that all programs of \( \text{SPDL}^+ \cup \) can be transformed to equivalent programs in normal form.

A program \( \alpha \) is in \( \text{(while do od-)normal form} \) iff \( \alpha = \beta_1 \cup \cdots \cup \beta_n \) and \( \beta_j = \gamma_{j_1} \cup \cdots \cup \gamma_{j_m} \), for all \( j, 1 \leq j \leq n \), such that for all \( \gamma_{jk}, 1 \leq k \leq m_j \), one of the following cases holds:

(a) \( \gamma_{jk} \) = atomic program;
(b) \( \gamma_{jk} \) = test;
(c) \( \gamma_{jk} = \text{while } E_{jk} \text{ do } \delta_{jk} \text{ od } \).
Tests can be simulated in SPDL+ ∪ by the means of Lemma 1(b). The normal form of a program α is denoted by αN.

**Lemma 2.** In M the following are valid:

(F) For each formula F of SPDL+ ∪ there are dF, tF ∈ N such that for all d ≥ dF one of the following conditions holds:

(a) for all t ≥ tF and h ≤ d + 1, M_d, s^h |= F,
(b) for all t ≥ tF and h ≤ d + 1, M_d, s^h |= ¬F.

(P) For each program while E do α od of SPDL+ ∪ there are dE,α, tE,α ∈ N such that for all d ≥ dE,α one of the following conditions holds:

(a) for all t ≥ tE,α and h ≤ d + 1, S^h(E, α) = {s^h},
(b) there is a finite set S(E, α) of states in M_d such that for all t ≥ tE,α and h ≤ d + 1, S^h(E, α) = S(E, α).

**Proof.** By induction on the structure of formulas and programs:

(F) (1) F = c, for c ∈ \{0, 1\}. By definition of M, condition (a) holds for d_F = 0, t_F = 0.

(2) F = true, false: trivial

(2') F = ¬G: By assumption for G there are d_G, t_G ∈ N. Choose d_F = d_G and t_F = t_G. Let d ≥ d_F. If G meets condition (a) then ¬G meets condition (b), and if G meets condition (b) then ¬G meets condition (a).

(2'') F = G ∨ H: By assumption for G and H there are d_G, d_H, t_G, t_H ∈ N. Choose d_F = max{d_G, d_H} and t_F = max{t_G, t_H}. Let d ≥ d_F. If condition (a) is met by G or H then by G ∨ H, too, and if condition (b) is met by G and H then by G ∨ H, too.

(3) F = [a]G: By assumption for G there are d_G, t_G ∈ N.

(i) a = r, for r ∈ \{0, 1\}: If r ≠ p, q then by definition of M, [r]G is true in all states of all models of M. Let r = p, q. Choose d_F = d_G and t_F = d_G + 1. Let d ≥ d_F. If G meets condition (a) then by definition of M, [p]G also meets condition (a), and if G meets condition (b) then by definition of M, [p]G also meets condition (b).

(ii) a = skip: By ⊨[skip]G ↔ G, the assertion holds for d_F = d_G and t_F = t_G by assumption for G.

a = abort: By ⊨[abort]G ↔ true.

(iii) a = β; γ: It is ⊨[β; γ]G ↔ [β][γ]G. By assumption for formulas and programs, the assertion holds for [γ]G and hence by assumption for programs for [β][γ]G, too.

a = while E do β od: By (P) there are d_E, β, t_E, β ∈ N. Choose d_F = max{d_E, β, d_G} and t_F = max{t_E, β, t_G}. Let d ≥ d_F. By (P) one of the following conditions holds:

(a) For all t ≥ t_F and h ≤ d + 1: S^h(E, β) = {s^h}. Then in M_d in all states above s^F the formula ¬E holds. By assumption for formulas, [while E do β od]G meets either condition (a) or condition (b).

(b) There is a finite set S(E, β) of states in M_d such that for all t ≥ t_F and h ≤ d + 1, S^h(E, β) = S(E, β). Hence, for all t ≥ t_F and h ≤ d + 1, M_d, s^h |= [while E do β od]G iff for all s ∈ S(E, β), M_d, s |= G. In M_d this is either
true or false, i.e. \([\text{while } E \text{ do } \beta \text{ od}]G\) meets condition (a) or condition (b), respectively.

\[ \alpha = \text{if } E \text{ then } \beta \text{ else } \gamma G \text{ if } \beta \text{ meets condition (a) or condition (b), respectively.} \]

\[ \alpha = \beta \cup \gamma G \text{ if } \beta \text{ meets condition (a) or condition (b), respectively.} \]

(P) By (F) there are \(d_E, t_E \in \mathbb{N}\). Let \(\alpha_N\) be the normal form of \(\alpha\) such that \(\alpha_N = \beta_1 \cup \cdots \cup \beta_n\) and \(\beta_j = \gamma_{j_1}; \cdots ; \gamma_{j_m}, 1 \leq j \leq n\), where \(\gamma_{jk}, 1 \leq k \leq m_j\), is an atomic program, a test, or a while do od-program. Because there are not transitions for all \(r \in \mathbb{Q}_0\) such that \(r \neq p, q\), the assumption holds that in all \(\beta_j, 1 \leq j \leq n\), only the atomic programs \(p\) and \(q\) occur. The assertion is proved now by induction on the number of nested while do od-programs in the program while E do \(\alpha_N\) od.

Induction start: In \(\alpha_N\) no program \(\gamma_{j_k}\) is a while do od-program. For all \(j, 1 \leq j \leq n\), let \(T_j = \{F \mid \text{there is } k \leq m_j \text{ such that } \gamma_{jk} = F? \text{ or } \neg F?\}\). By definition of \(\alpha_N\), all \(F \in T_j\) occur in \(\alpha\), i.e. by (F) there are \(d_F, t_F \in \mathbb{N}\) for each \(F \in T_j\). Hence let \(d_{T_j} = \max\{d_F \mid F \in T_j\}\) and \(t_{T_j} = \max\{t_F \mid F \in T_j\}\). Further, for all \(j, 1 \leq j \leq n\), let \(f_p\) be the maximal number of programs \(\gamma_{jk} = p\) that can be run through in \(\beta_j\) without running through a program \(\gamma_{j_k} = q\). With these definitions let

\[ d_T = \max\{d_{T_j} \mid 1 \leq j \leq n\}, \]

\[ t_T = \max\{t_{T_j} \mid 1 \leq j \leq n\}, \]

\[ f_p = \max\{f_p \mid 1 \leq j \leq n\}. \]

Choose \(d_{E, \alpha} = \max\{d_E, d_T, 2f_p + 1\}\) and \(t_{E, \alpha} = \max\{t_E, t_T\} + 1\). Let \(d \geq d_{E, \alpha}\). Then by (F) the following two cases result:

(1) For all \(t \geq t_{E, \alpha}\) and \(h \leq d + 1\), \(s_h^T \models \neg E\). Then, for all \(t \geq t_{E, \alpha}\) and \(h \leq d + 1\), \(S_h^T(R_{\text{while } E \text{ do } \alpha \text{ od}} s_h^T) = \{s_h^T\}\).

(2) For all \(t \geq t_{E, \alpha}\) and \(h \leq d + 1\), \(s_h^T \models E\). Let \(t' \geq t_{E, \alpha}\) and \(h' \leq d + 1\) such that \(S_h^T(E, \alpha) \neq \emptyset\), i.e. let \(s'\) be in \(s_{h'}^T\) in \(s_h^T\) such that \(S_h^T(R_{\text{while } E \text{ do } \alpha \text{ od}} s')\). Then for all \(t \geq t_{E, \alpha}\) and \(h \leq d + 1\), \(S_h^T(R_{\text{while } E \text{ do } \alpha \text{ od}} s')\).

For proving this assertion examine the segment model \(\tilde{\mathcal{S}}_{d}\) in \(W_d\) lying directly below the state \(s_0^{E, \alpha}\), (see Fig. 3). Here, \(s_{d+1} = s_0^{E, \alpha} = s_{d+1}^{E, \alpha-1}\) and \(s_0 = s_0^{E, \alpha-1}\). By definition of \(t_{E, \alpha}\), \(E\) holds in all states of \(\tilde{\mathcal{S}}_{d}\). Because of \(W_d, s' \models \neg E\) there is, by definition of \(\alpha_N\), a sequence \(\beta_{j_1}, \ldots, \beta_{j_m}, 1 \leq j_0, \ldots, j_m \leq n\), such that the concatenation \((E?; \beta_{j_m}); \cdots; (E?; \beta_{j_1})\) leads from \(s_h^T\) to \(s'\) throughout \(\tilde{\mathcal{S}}_{d}\). By definition of \(t_{E, \alpha}\), in \(W_d\) each program \(\beta_j\) is equivalent to a sequence of programs \(p\) and \(q\) in all states above \(s_0\) since each test in \(\beta_j\) is either true in all states or false in all states above

![Fig. 3.](image-url)
So. By definition of \(d_{E, \alpha} \), in the sequence \(\beta_{j_0}, \ldots, \beta_{j_m}, 1 \leq j_0, \ldots, j_m \leq n \), a program \(\beta_{j_i}'\) must occur that is equivalent to a sequence of \(p\)'s. The reason is that if a program \(\beta_{j_i}, 0 \leq i \leq m\), is run through in \(\Xi_d\) each occurrence of \(q\) in \(\beta_{j_i}\) makes \(\beta_{j_i}\) return to the state \(\xi_{d+1}\) at the states \(\xi_d\) to \(\xi_1\). Hence, there is even a state \(\xi_h\) in \(\Xi_d\) such that \(\xi_h \stackrel{R_{\text{while} E \text{do} \alpha \text{od} s'}}{\longrightarrow} \xi_i\). In addition, in the sequence \(\beta_{j_0}, \ldots, \beta_{j_m}, 1 \leq j_0, \ldots, j_m \leq n\), a program \(\beta_{j_i}'\) must occur that is equivalent to a sequence of \(p\)'s and \(q\)'s with an odd number of \(q\)'s. Otherwise, no transition from a state above \(s_0^{t_{E, \alpha}}\) to a state below \(s_d^{t_{E, \alpha}}\) is possible. For each state \(s_h'\) above \(s_0^{t_{E, \alpha}}\), now by construction of \(\mathcal{W}_d\) a finite concatenation may be constructed by the two programs \((E ?; \beta_{j_i}')\) and \((E ?; \beta_{j_i}'\) which leads from \(s_h'\) to \(s_h\). As a result, \(s_h \stackrel{R_{\text{while} E \text{do} \alpha \text{od} s'}}{\longrightarrow} \xi_i\) holds for all \(t \geq t_{E, \alpha}\) and \(h \leq d + 1\).

Since by assumption \(\neg E\) holds only in a finite number of states below \(s_0^{t_{E, \alpha}}\), there is a finite set \(S(E, \alpha)\) of states in \(\mathcal{W}_d\) such that \(S_h'(E, \alpha) = S(E, \alpha)\) for all \(t \geq t_{E, \alpha}\) and \(h \leq d + 1\).

Case (1) yields condition (a), case (2) condition (b) of assertion (P).

**Induction step:** In \(\alpha_N\) a program \(\gamma_{j_i}\) occurs that is a \(\text{while do od}\)-program. For all \(j, 1 \leq j \leq n\), let \(W_j = \{((F, \delta) | \text{there is } k \leq m_j \text{ such that } \gamma_{jk} = \text{while } F\text{ do } \delta \text{ od}\}\}. \) Since by definition of \(\alpha_N\) the set of \(\text{while do od}\)-programs occurring in \(\alpha_N\) is identical to the set of \(\text{while do od}\)-programs occurring in \(\alpha\), by assumption of the induction on nested \(\text{while do od}\)-programs there are \(d_{F, \delta}, t_{F, \delta} \in \mathbb{N}\) for each \((F, \delta) \in W_j\). Let \(d_w = \max\{d_{F, \delta} | (F, \delta) \in W_j\}\) and \(t_w = \max\{t_{F, \delta} | (F, \delta) \in W_j\}\). By that we have also defined

\[
\begin{align*}
\text{Let } d = d_{E, \alpha}. & \quad \text{Then by (F) the following two cases result:} \\
& \quad (1) \text{ For all } t \geq t_{E, \alpha} \text{ and } h \leq d + 1: \mathcal{W}_d, s_h \equiv \neg E. \text{ Then, for all } t \geq t_{E, \alpha} \text{ and } h \leq d + 1, \\
& \quad S_h'(E, \alpha) = \{s_h'\}. \\
& \quad (2) \text{ For all } t \geq t_{E, \alpha} \text{ and } h \leq d + 1: \mathcal{W}_d, s_h \equiv E. \text{ Let } t' \geq t_{E, \alpha} \text{ and } h' \leq d + 1 \text{ such that } \\
& \quad S_h'(E, \alpha) \neq \emptyset, \text{ i.e. let } s' \text{ be in } \mathcal{W}_d \text{ such that } s_h \equiv \text{while do od} \text{ s}'. \text{ Then for all } \\
& \quad t \geq t_{E, \alpha} \text{ and } h \leq d + 1, s_h \equiv \text{while do od} \text{ s}'. \\
& \quad \text{To prove this assertion, look again at the segment model } \Xi_d \text{ in } \mathcal{W}_d \text{ lying directly below the state } s_0^{t_{E, \alpha}}. \text{ By definition of } t_{E, \alpha}, E \text{ holds in all states of } \Xi_d. \text{ Because of } \\
& \quad \mathcal{W}_d, s_0 \equiv \neg E, \text{ there is, by definition of } \alpha_N, \text{ a sequence } \beta_{j_0}, \ldots, \beta_{j_m}, 1 \leq j_0, \ldots, j_m \leq n, \\
& \quad \text{such that the concatenation } (E ?; \beta_{j_0}), \ldots, (E ?; \beta_{j_m}) \text{ leads from } s_0^{t_{E, \alpha}} \text{ to } s' \text{ throughout } \Xi_d. \text{ By assumption of the induction and by definition of } t_{E, \alpha}, \text{ each } \text{while do od}\text{-program occurring in the programs } \beta_{j_i} \text{ meets condition (a) or (b) in } \mathcal{W}_d \text{ above } \\
& \quad s_0^{t_{E, \alpha}}. \text{ Let } \gamma_{j_i} = \text{while } F\text{ do } \delta \text{ od} \text{ be the first while do od}\text{-program occurring in the sequence } \beta_{j_0}, \ldots, \beta_{j_m}, 1 \leq j_0, \ldots, j_m \leq n, \text{ as a subprocess that meets condition (P)(b).} \\
& \quad \text{Let } \beta_{j_i}' \text{ be the program in which } \gamma_{j_i} \text{ occurs, and let } \gamma' \text{ be the initial part of } \beta_{j_i}' \text{ preceding } \gamma_{j_i}'. \text{ Then all } \text{while do od}\text{-programs that are run through above } s_0^{t_{E, \alpha}} \text{ before } \\
& \quad \gamma_{j_i} \text{ are equivalent there to skip by assumption of the induction and by definition of } t_{E, \alpha}. \text{ There are two cases now:} \\
\end{align*}
\]
Determinism and non-determinism in PDL

(i) The program $\gamma'_{i}$ is not run through above $s^{t_{E,\alpha-1}}_{0}$. By $d_{E,\alpha} \geq 2f_{p} + 1$, $\gamma'$ cannot lead throughout $\Xi_{d}$. Hence, there are programs $\beta_{j_{i}}$ of the sequence $\beta_{j_{0}}, \ldots, \beta_{j_{m}}$, $1 \leq j_{0}, \ldots, j_{m} \leq n$, which are run through before $\beta'_{j_{i}}$ and which lead into and even throughout $\Xi_{d}$ because of the definition of $d_{E,\alpha}$. The proof of the assertion then follows the proof of the induction start, i.e. for all $t \geq t_{E,\alpha}$ and $h \leq d + 1$, $s'_{h} R_{\text{while} c \in d \alpha \text{and} s'}$. (This case is also true if there is no program $\gamma'_{i}$ that meets condition (P)(b).)

(ii) The program $\gamma'_{i}$ is run through above $s^{t_{E,\alpha-1}}_{0}$. By assumption of the induction for $\gamma'_{i}$ and by definition of $t_{E,\alpha}$, there is a finite set $S(F, \delta)$ of states in $\mathcal{M}_{d}$ such that, for all $t \geq t_{E,\alpha}$ and $h \leq d + 1$, $S'_{h}(F, \delta) = S(F, \delta)$. Then also, for all $t \geq t_{E,\alpha}$, $h \leq d + 1$, and $s \in S(F, \delta)$, $s'_{h} R_{\gamma^{'}, \text{while} c \in d \alpha \text{and} s}$, since $\gamma'$ is equivalent to a sequence of $p$'s and $q$'s above $s^{t_{E,\alpha-1}}_{0}$. Hence, from each state above $s^{t_{E,\alpha}}_{0}$ a sequence of programs $(E ?; \beta_{j})$ may be concatenated starting with the program $(E ?; \beta'_{i})$ and leading to the state $s'$, i.e. for all $t \geq t_{E,\alpha}$ and $h \leq d + 1$, $s'_{h} R_{\text{while} c \in d \alpha \text{and} s'}$.

Since, by assumption, the formula $\neg E$ holds only in a finite number of states in $\mathcal{M}_{d}$ below $s^{t_{E,\alpha}}_{0}$, there is a finite set $S(E, \alpha)$ of states in $\mathcal{M}_{d}$ such that for all $t \geq t_{E,\alpha}$ and $h \leq d + 1$, $S'_{h}(E, \alpha) = S(E, \alpha)$.

In summary, as in the proof of the induction start, case (1) yields condition (a) and case (2) yields condition (b) of the assertion (P). □

The proof of Theorem 2(b) now follows from Lemma 2 yielding as a corollary the minimal star height of the PDL-program $(p^{*};q;p^{*};q)^{*}$. Star height is a notion of the theory of regular languages specifying the greatest number of nested stars in a regular expression. Hence, the star height of $(p^{*};q;p^{*};q)^{*}$ is 2. The minimal star height of an expression is the minimal number of nested stars that is needed to construct an expression equivalent to that expression.

Proof of Theorem 2(b). The PDL-formula $\langle(p^{*};q;p^{*};q)^{*}\rangle[ p \cup q ] \text{false}$ cannot be expressed in SPDL $+ \cup$ because in all models $\mathcal{M}_{d} \in M$, $d \in \mathbb{N}$, it is true in an infinite number of states and false in an infinite number of states. □

Corollary 1. The minimal star height of the regular expression $(p^{*};q;p^{*};q)^{*}$ is 2.

Proof. Suppose that the minimal star height of $(p^{*};q;p^{*};q)^{*}$ is 1. Then there is a regular expression $\alpha$ on $p$ and $q$ such that in no $^{*}$-expression occurs another $^{*}$-expression and such that

$\vdash \langle \alpha \rangle[p \cup q] \text{false} \leftrightarrow \langle(p^{*};q;p^{*};q)^{*}\rangle[ p \cup q ] \text{false}.$

By the same equivalences transforming (5) into (6), $\langle \alpha \rangle[p \cup q] \text{false}$ can be changed equivalently into a formula of SPDL $+ \cup$. But this is a contradiction to Lemma 2. □

Lemma 2 is therefore a special way to prove the existence of regular expressions on two symbols having minimal star height 2. It was first proved in [1] by means of the theory of regular languages that there are regular expressions on two symbols having arbitrarily high minimal star height.
4. The System PPL

Theorems 1 and 2 show how the non-deterministic concepts * and \( \cup \) increase the expressive power of SPDL. The concept * turns out to be stronger than \( \cup \) because SPDL+* has the expressive power of PDL. But looking at systems of Dynamic Logic as program logics means that the expressive power concerning programs plays an important role, too. In this sense SPDL+* is a weaker system than PDL. In comparing the non-deterministic concepts * and \( \cup \), the latter one seems to be intuitively more accessible. A choice between two alternatives is more intelligible than a choice between an infinite number of alternatives. For this reason, in the following a system is introduced which is based on SPDL+\( \cup \) and extends it by a deterministic program concept. By this the expressive power of PDL—concerning both formulas and programs—is obtained.

The syntax of the system PPL ("Propositional Program Logic") is defined as the syntax of SPDL. A set \( \mathcal{P} \) of propositional variables is added requiring the new rule

\[
(1)(v) \quad \mathcal{P} \subseteq \mathcal{F}.
\]

The set \( \mathcal{P} \) of PPL-programs is defined as in SPDL. The following rules for non-deterministic choice and propositional assignments are added:

\[
(2b)(iv) \quad \text{if } a, b \in \mathcal{P}, \text{ then } a \cup b \in \mathcal{P},
\]

\[
(v) \quad \text{if } v \in \mathcal{P}, \text{ then } v := \text{true} \in \mathcal{P} \text{ and } v := \text{false} \in \mathcal{P}.
\]

A Boolean random assignment is defined by \( v := ? =_{df} v := \text{true} \cup v := \text{false} \). This extension preserves SPDL+\( \cup \) as a special case with \( \mathcal{P} = \emptyset \).

The models of PPL are extensions of SPDL-models. But the two classes of models differ in the way how states are defined. While in a model of SPDL a state cannot be analyzed further, in a model of PPL a state is composed of a world and a valuation denoting the truth value of the propositional variables in the state. The valuation can be changed by the propositional assignments, thus each possible pair of world and valuation is a state.

To define a PPL-model, an SPDL-model \( \mathcal{M} = (W, \varphi, \pi) \) is chosen as a basis called model structure of a PPL-model. A PPL-model \( \mathcal{M} = (W, H, \varphi, \pi) \) consists of a model structure \( (W, \varphi, \pi) \) and the set \( H \) of all mappings \( h : \mathcal{P} \rightarrow \{ \text{"true"}, \text{"false"} \} \). The elements of \( W \) are called worlds, the elements of \( H \) valuations. \( S =_{df} W \times H \) defines the set of states.

Proceeding from a PPL-model \( \mathcal{M} \), now analogous to a SPDL-model the mappings \( \varphi \) and \( \pi \) are extended to define the value of an arbitrary formula in a state \( s = (w, h) \) of \( \mathcal{M} \) and the transitions of an arbitrary program. For that purpose, again the relations \( \models \) and \( R_\alpha \) are defined. The definition of \( \models \) in the system SPDL is completed by

\[
(3)(iv) \quad \mathcal{M}, (w, h) \models v \iff_{df} h(v) = \text{"true"} \quad \text{for } v \in \mathcal{P}.
\]

The definition of the relations \( R_\alpha \) is corrected and extended by
Determinism and non-determinism in PDL

(4b) (i) \((w, h) R_p (w', h') \text{ iff} df \ (w, w') \in \pi(p) \text{ and } h \equiv h' \text{ for } p \in \mathbb{P}_0.\)

(iii) \(R_{a \cup \beta} = df R_a \cup R_{\beta} \).

(iv) \((w, h) R_{v = \text{true}} (w', h') \text{ iff} df \ w = w', h \equiv /_{v} h' \text{ and } h'(v) = \text{"true"}, \)

for \(v \in \mathbb{V} \),

\((w, h) R_{v = \text{false}} (w', h') \text{ iff} df \ w = w', h \equiv /_{v} h' \text{ and } h'(v) = \text{"false"}, \)

for \(v \in \mathbb{V} \).

Here, for \(h, h' \in H \text{ and } v \in \mathbb{V} \) the following definition holds:

\[ h \equiv /_{v} h' \text{ iff} df h(v') = h'(v') \text{ for all } v' \in H \text{ such that } v' \neq v. \]

For \(\mathbb{V} = \emptyset, H \) consists of the empty mapping, i.e. \(W \times H = W\). Then a PPL-model is equivalent to its model structure, and the relations \(\models \) and \(R_a\) on \(W \times H\) are identical to the corresponding relations on \(W\). The definition of \(R_{v = \text{true}}, R_{v = \text{false}}\),

and \(S\) yields the property that the propositional assignments are deterministic and terminate in all states, i.e. \(R_{v = \text{true}} \text{ and } R_{v = \text{false}}\) are totally defined functions on \(S\).

By propositional assignments it is possible to simulate the program concept \(*\) in the system PPL. This means that PPL has the expressive power of PDL. For each PPL-program \(\alpha\), a PPL-program \(\alpha^*\) can be found that is equivalent to \(\alpha^*\), i.e. \(R_{\alpha^*} = R_{\alpha^*}\) where the latter relation is defined in (4a). Let \(v \in \mathbb{V}\), and let \(\alpha\) be a PPL-program in which \(v\) does not occur. Then the following holds:

\[
\alpha^* = \text{if } v \text{ then } v := ? ; \text{while } v \text{ do } \alpha ; v := ? ; v := \text{true}
\]

\[\text{else } v := ? ; \text{while } \neg v \text{ do } \alpha ; v := ? ; v := \text{false } fi,\]

i.e., these two programs have the same transitions in each model. The complicated form caused by the outer conditional program brings about that the variable \(v\) maintains its initial value during and after the execution of \(\alpha^*\). Hence, side effects are avoided. In particular, this means that nested \(*\)-programs can be expressed in PPL using a single variable—provided this variable does not occur in the innermost \(*\)-program.

A simple example would show that the simulation (8) does not hold generally, i.e., in case the value of \(v\) changes during the execution of \(\alpha\). But, as the next lemma shows, each \(*\)-program is equivalent to the union of four partial programs in which no variable occurs in \(*\)-programs. These partial programs can be simulated by (8). This lemma is found already in [13], but it is formulated there in another way. Additionally, the proof in [13] is only sketched and is incomplete in the case of \(*\)-programs. A detailed proof is given in [14].

Lemma 3. Let \(F\) be a formula of the system PPL+*, let \(\alpha\) be a program of this system, and let \(v \in \mathbb{V}\) be a propositional variable. Then the following holds:

(F) There are formulas \(F \setminus v\) and \(F \setminus \neg v\) of PPL+* in which \(v\) does not occur, such that

\[ \models F \leftrightarrow (v \to F \setminus v) \land (\neg v \to F \setminus \neg v), \]

(P) There are programs \(\alpha_1, \alpha_2, \alpha_3\) and \(\alpha_4\) of PPL+* in which \(v\) does not occur, such that

\[ R_{\alpha} = R_{(v ?; \alpha_1 ; v = \text{true}) \cup (v ?; \alpha_2 ; v = \text{false}) \cup (\neg v ?; \alpha_3 ; v = \text{true}) \cup (\neg v ?; \alpha_4 ; v = \text{false})}. \]
Proof. By induction on the structure of formulas and programs. □

Theorem 3. PPL has the expressive power of PDL.

Proof. Since in PPL the program concept * can be simulated, each PDL-formula can be expressed in PPL if \( \mathfrak{B} \neq \emptyset \). Let \( F \) be a PPL-formula. By Lemma 1, all programs in \( F \) can equivalently be transformed into PDL-programs. By Lemma 3', for all \( v \in \mathfrak{B} \) there are PDL-formulas \( F \setminus v \) and \( F \setminus v \) such that

\[
\models F \leftrightarrow (v \rightarrow F \setminus v) \land (\neg v \rightarrow F \setminus v).
\]

Thus, except the additional information of valuations which in PDL may be expressed by propositional constants, the extension of SPDL+ \( \cup \) by propositional assignments does not yield any really new expressions. □

In [14] axioms and rules for the system PPL are introduced. They lean upon axioms and rules from [3, 4]. It is shown that these axioms and rules are correct and that PPL is complete and decidable. The proof follows the combination technique presented in [6] extending it according to the new semantical conditions.

References