Transforming Predicates or Updating States? Total Correctness in Dynamic Logic and Structured Programming

Marc Pauly

September 11, 1997

1 Introduction

1.1 How to model actions

Programs can be viewed as specifications of actions, and these actions can be modeled in a variety of ways. Depending on whether one is interested in cognitive actions such as speech acts or reasoning, economic actions of e.g. consumers and companies, or social actions such as group behavior, different theories and models of actions have been proposed.

If the goal is to model computational actions (i.e. computer programs), there are a number of models which have been developed. Take for example the extremely simple program $x := 0;y := 1$ which first assigns 0 to the variable x and then 1 to the variable y, where the states of the computer are assignments of natural numbers to variables. Among the well-known approaches to modeling such a program are

- *State Transformer Semantics.* Programs are functions from initial states to (sets of) possible final states, the state(s) which can be reached through executing the program. For the example, the function would map any state s to the state s' where $s'(x) = 0, s'(y) = 1$ and $s'(z) = s(z)$ for any z other than x, y.

- *Trace Semantics.* Programs are sets of sequences of states, where the sequences faithfully model the possible executions step by step. Here we are not just interested in what the result of a computation is, but also in how it is obtained. The above program would be modeled by the set of all sequences of states $\langle s, s', s'' \rangle$, where s' is the same as s except that $s'(x) = 0$ and s'' is the same as s' except that $s''(y) = 1$.

- *Arrow Logic.* Programs are abstract entities; we only model the connection between these entities, but not any kind of input/output relation. In our example, the program would be modeled by an action c such that c is the composition of two actions a and b which model $x := 0$ and $y := 1$ respectively.

- *Predicate Transformer Semantics.* Programs are functions from sets of states to sets of states. Given a set of final states, the function specifies the initial states from which an execution of the program is guaranteed to terminate in one of the final states. (This set of initial states is called the weakest precondition.) For our example, a set of states Y would be mapped to the set of all states s such that there is a state s' in Y such that s is the same as s' except that $s'(x) = 0$ and $s'(y) = 1$.

Since we are interested in models which allow us to talk about the total correctness of programs, the last approach will turn out to be quite natural, since its notion of a program is already based on the program's correctness conditions. But also a model based e.g. on the first approach can be used to reason about total correctness.

Among the first three approaches, there is a clear order of abstraction. If we take $x \succ y$ to mean "x is a richer model (i.e. contains more information, is less abstract) than y", we get

$$\text{Trace Semantics} \succ \text{State Transformer Semantics} \succ \text{Arrow Logic}$$

The first and last approach however appear to be on the same level of abstraction and the precise connections between them will be investigated in this paper. This investigation will start with an abstract comparison between predicate transformers and state transformers as set-theoretic objects and will then deal more concretely with the transformers associated with particular classes of programs.

1.2 Program classes, sample programs and types of failure

The actions/programs under investigation will be taken from well-known programming languages, such as the language of $WHILE$ programs or the guarded-command language. To get a feeling for the concrete predicate transformers of the programming constructs in these languages, the reader can take a quick look at sections 3.1 and 3.3. Once we turn to reasoning systems for total correctness, we will distinguish three increasingly complex program classes. Our reasoning systems will have to account for the termination properties of the programs in these classes.

For the simplest class, the class of **deterministic** programs, $WHILE$ programs such as

$$\mathsf{if}\ x \geq 0\ \mathsf{then}\ y := 4\ \mathsf{else\ while}\ true\ \mathsf{do\ skip}$$

will be at the center of our attention, and in general programs which under certain conditions do not terminate. In this class, every conditional choice is of the form $\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2$ so that for every input state there is not more than one output state.

For **bounded nondeterministic** programs, we allow for more than one output state through use of the nondeterministic guards in the guarded-command language, such as in

$$\mathsf{if}\ x \geq 0 \rightarrow S_1 \| x \leq 0 \rightarrow S_2\ \mathsf{fi}$$

where if $x = 0$, S_1 or S_2 can be executed nondeterministically. Thus, the guards do not have to be mutually exclusive. In programs from this class, all choices are made from a finite number of alternatives.

The next natural class are the **unbounded nondeterministic** programs, i.e. programs where there can be infinitely many output states for a given input state. A celebrated example is

$$\begin{array}{l} \mathsf{while}\ x \neq 0\ \mathsf{do}\ \mathsf{if}\ x > 0\ \mathsf{then}\ x := x - 1 \\ \qquad\qquad\qquad\qquad\quad \mathsf{else}\ x := y.y > 0 \end{array}$$

Here, $x := y.y > 0$ means that an arbitrary positive integer is assigned to x, thus allowing for infinitely many possibilities. If the goal is to correctly capture the termination properties of this program, quite expressive reasoning systems turn out to be required. This also holds for programs such as

$$p^*$$

known from Dynamic Logic, where some program p is iterated an arbitrary finite number of times.

In most programming languages, there are two ways in which a program can fail to terminate: Either the program gets stuck at a certain state, e.g. if a division by 0 has been attempted, or the program runs in an infinite loop without reaching any final state at all; this latter type of **failure** is called **divergence**. These two types of failures are exemplified by the following two programs:

$$\mathsf{abort}\ \text{vs.}\ \mathsf{while}\ \mathit{true}\ \mathsf{do}\ \mathsf{skip}$$

In this paper, we will not distinguish these two types of failure. Naturally, this means deciding to ignore information as to the kind of failure we are dealing with. But while it is certainly true that in "the real world" these two kinds of failure do not amount to the same thing, a model which identifies these two failures is still quite adequate for many purposes. Especially when we are interested in proving the total correctness of our program, we may only be interested in the states from which the program is guaranteed to terminate successfully. In this perspective, we do not care about the reasons for failure, since all kinds of failure are equally undesired. It should be noted that from the perspective of weakest-precondition semantics, it is actually quite common not to distinguish these two kinds of failure (see e.g. [DS90], [Bac81], [BES95]).

1.3 Purpose and overview of the paper

On a very general level, the goal of this paper is a comparison between two semantic traditions, the predicate transformer tradition in the style of Dijkstra and the more standard state transformer tradition in the style of e.g. Dynamic Logic. Since state-transformer (ST) semantics seem to be much more well-known and more frequently applied than predicate-transformer (PT) semantics, this paper will concentrate on the PT-approach and connecting it to the ST-approach rather than vice versa. Hence, part of this paper can also serve as an introduction to PT-semantics for ST-semanticists, especially sections 2 and 3.

The two traditions will be compared on three increasingly concrete levels: (1) on the abstract set-theoretic level where predicate transformers and state transformers are viewed as abstract functions, (2) on the level of the classes of transformers which are associated with particular programming languages, such as the $WHILE$ language or the guarded-command language, (3) on the level of logical implementation, i.e. by investigating logics in which we can formalize (for a given class of programs) predicate transformers on the one hand and state transformers on the other hand. While most of the general results used in this comparison are not new (the most important ones can be found in [AP86], [Bac81] and [HP78]), we think that the comparison has not yet been done in such a systematic fashion, especially regarding the three program classes just mentioned. Furthermore, as a result of this comparison we propose a new logic called PDL^{pt} which is a propositional dynamic logic with PT-semantics. It will be argued that this logic is a rather general framework for reasoning about the total correctness of programs and actions more generally.

As for a more detailed overview, section 2 compares the expressiveness of predicate transformers and state transformers on an abstract set-theoretic level. The main general result here is due to [AP86], while the extensions of this result regarding the different subclasses of transformers (deterministic, bounded nondeterministic, etc.) seem to be new.

Section 3 then turns to our three program classes and presents the state transformers and predicate transformers of the given programming constructs. These transformer semantics are quite well-known and can be found e.g. in [NN92] and [Bac81], [BES95] respectively. The section closes with an example (from [Hes90]) of a "programming construct" which cannot be modeled by state transformers.

The formalization of different predicate transformer classes in first-order and infinitary logic is discussed in section 4. The results here are all due to [Bac81]. Since it turns out that the distinction between bounded and unbounded nondeterminism is crucial in this context, the whole section also serves to set the stage for trying to recover similar distinctions in Dynamic Logic.

Section 5 runs parallel to the previous section in trying to formalize different state transformer classes in propositional dynamic logic (PDL). For the nondeterministic case, extensions of PDL are considered which have been proposed already in [HP78]. One of the extensions, PDL^+ adds an error state to PDL

and a corresponding new modality $[p]^+$; the other extension, $PDL + loop$ adds a new predicate *loop* which checks for infinite chains.

Parallel to section 3.2 but in much more detail, section 6 demonstrates that we can rewrite the PDL^+ definition of $[p^*]q$ (partial correctness) as well as the definition of $[p^*]^+q$ (total correctness) in terms of predicate transformers which use fixpoints. In section 7.2, we demonstrate that this new fixpoint approach allows one to recover a complexity difference between bounded and unbounded nondeterminism by placing a bound on the fixpoint iteration, thus mirroring the results from section 4. This in turn makes it possible to investigate finitary total correctness logics. We do not think that predicate transformers in the semantics of Dynamic Logic, definitions of $[p^*]^+q$ in terms of fixpoints and finitary total correctness logics have been investigated before.

Section 7 also introduces a new logic PDL^{pt}. A model for this logic associates a predicate transformer rather than a state transformer with every atomic program, and for any program p defines the semantics of $[p]^+q$ and $[p]q$ completely in terms of predicate transformers, thus carrying the discussion of the previous section to its natural conclusion. It is argued that this logic provides a general framework for reasoning about the total correctness of programs.

2 State Transformers & Predicate Transformers: A Set-Theoretic Comparison

Since our goal is to reason about total program correctness, we intend our predicate transformers to be **weakest-precondition** transformers, wp-transformers for short. For any program P, wp_P is the weakest-precondition transformer associated with it. Thus, for any postcondition Y, $wp_P(Y)$ is the weakest precondition of P under postcondition Y, i.e. the set of states X such that if P is started in $s \in X$, P is guaranteed to terminate in a state $s' \in Y$. The weakest-precondition transformers are distinguished from the **weakest-liberal-precondition** transformers, wlp-transformers for short. For a given program P and a given postcondition Y, $wlp_P(Y)$ is the set of states X such that if P is started in $s \in X$, P either fails to terminate or terminates in a state $s' \in Y$.

Taking a step back from the intended modeling application of predicate transformers, given a set of states S, predicate transformers are simply operations on $\mathcal{P}(S)$.

On the other hand, state transformers are intended to model the program's input-output behavior directly. This can be done in a number of ways. Most simply, if we restrict ourselves to deterministic programs, our state transformers could be partial operations on the set of states, being undefined in case the program does not successfully terminate. Once we want to talk about nondeterministic programs however, deterministic state transformers are not sufficient anymore as a model.

A natural way to increase the complexity of the model would be to introduce the nondeterministic state transformers as functions from S into $\mathcal{P}(S)$, thus moving to the next natural stage of set-theoretic complexity. It turns out however that this model is insufficient for the purpose of total correctness. Consider a program with two possible executions from a given state s, one leading to a final state s', and the other failing to terminate. In this model, the set of output states would have to be $\{s'\}$, thus making it impossible to distinguish this program from a program which when started in s always terminates in s'. The problem in this model turns out to be the fact that failure of termination can only be successfully modeled when, given a starting state s, all executions fail to terminate, in which case the set of output states would be empty. As soon as some executions terminate and some do not, we run into problems. Nonetheless, we will discuss this model in what follows because it is the natural intermediate step to a more elaborate model and because it turns out that it is still useful for reasoning about weakest liberal preconditions.

Returning to the problem we ran into when talking about total correctness of nondeterministic programs, the solution which is most congenial to state-transformer semantics is to raise the set-theoretic complexity yet a bit more by adding a special state $\perp$ to S which functions as a failure state. In this new model, the program mentioned above would map the input state s to the set of output states $\{s', \perp\}$. Hence, by introducing a failure state, we recover the distinction between this program and the one which when started in s always terminates in s'. The new state transformers will thus be functions from S into $\mathcal{P}(S \cup \{\perp\})$. This model will be adopted for most of the rest of this paper, since it is the most general of the state-transformer models mentioned.

After presenting the basic definitions and some set-theoretic facts, results will be given which relate predicate transformers to the three state transformer models just mentioned. The main results of this part are based on [AP86] and [DS90]. The section ends with some further distinctions among nondeterministic state transformers, linking these to corresponding distinctions among predicate transformers.

2.1 The basic transformer classes and central definitions

Definition 1 (Transformer Classes) *Given the set of states S, we define the following classes of functions:*

- *DST: all partial functions $f : S \hookrightarrow S$.*
- *NST: all total functions $f : S \to \mathcal{P}(\mathcal{S})$.*
- *$NST_\perp$: all total functions $f : S \to \mathcal{P}(\mathcal{S} \cup \{\perp\})$, where $\perp \notin S$.*
- *PT: all total functions $F : \mathcal{P}(\mathcal{S}) \to \mathcal{P}(\mathcal{S})$.*

We call these transformers deterministic state transformers, nondeterministic state transformers and predicate transformers respectively. It will be clear from the context whether by "nondeterministic predicate transformer" we mean a function of NST or of $NST_\perp$. We shall often denote $S \cup \{\perp\}$ as $S_\perp$.

Regarding predicate transformers, we adopt the following terminology from [DS90] and [Dij76].

Definition 2 (Junctivity) *A predicate transformer F is called* universally conjunctive *iff $F(\bigcap_{X\in V} X) = \bigcap_{X\in V} F(X)$ for all $V \subseteq \mathcal{P}(\mathcal{S})$. Similarly, F is called* universally disjunctive *iff $F(\bigcup_{X\in V} X) = \bigcup_{X\in V} F(X)$ for all $V \subseteq \mathcal{P}(\mathcal{S})$. The notions of* positively conjunctive *and* positively disjunctive *are defined analogously for all nonempty V. Observe that for $V = \emptyset$, we define universal disjunctivity as $F(\emptyset) = \emptyset$ and universal conjunctivity as $F(S) = S$. The first of these, $F(\emptyset) = \emptyset$ is called the* law of excluded miracle, *LEM for short. Finally, F is called* or-continuous *iff for all nonempty denumerable sequences of predicates $C_0 \subseteq C_1 \subseteq C_2 \subseteq \cdots$ we have $F(\bigcup_{i<\omega} C_i) = \bigcup_{i<\omega} F(C_i)$.*

Definition 3 (Equivalence) *Two transformer classes T_1 and T_2 are called f-*equivalent, *written $T_1 \cong_f T_2$, iff f is a total bijection from T_1 to T_2. For a weaker notion, $T_1 \succeq_f T_2$, iff f is a total surjective function from T_1 to T_2, not necessarily injective.*

We are of course only interested in f-equivalence for particular f, namely functions which map a state transformer to its corresponding weakest (or weakest liberal) precondition transformer. The functions used in the following theorems fulfill this purpose, as will be argued. Some set-theoretic facts we will use in the following theorems:

Fact 1 *For all partial functions $f : A \hookrightarrow B$ and nonempty $V \subseteq \mathcal{P}(B)$, we have*

$$\begin{array}{lll} (a) & f^{-1}[\bigcup_{X\in V} X] = \bigcup_{X\in V} f^{-1}[X] & \text{and } f^{-1}[\emptyset] = \emptyset \\ (b) & f^{-1}[\bigcap_{X\in V} X] = \bigcap_{X\in V} f^{-1}[X] & \text{and } f^{-1}[B] = B \quad \text{if } f \text{ is total} \\ (c) & \mathcal{P}(\bigcap_{X\in V} X) = \bigcap_{X\in V} \mathcal{P}(X) & \end{array}$$

2.2 Deterministic state transformers

We will start our investigation with the class of deterministic state transformers and their corresponding weakest-precondition predicate transformers:

Theorem 2 *$DST \cong_{\lambda f.\lambda Q.f^{-1}[Q]}$ universally disjunctive positively conjunctive PT*

Proof. Let $\phi(f) = \lambda Q.f^{-1}[Q]$, with $f \in DST$. From the preceding fact we know that $\phi(f)$ is indeed universally disjunctive and positively conjunctive; clearly, ϕ is total. Furthermore, to see that ϕ is injective, suppose $f_0 \neq f_1$, i.e. for some s, $f_0(s) \neq f_1(s)$. Then, assuming that $f_0(s)$ is defined, $s \in \phi(f_0)(\{f_0(s)\})$, while $s \notin \phi(f_1)(\{f_0(s)\})$, so $\phi(f_0) \neq \phi(f_1)$.

Lastly, to see that ϕ is onto, assume we have some universally disjunctive predicate transformer F. Define the state transformer f as $f(s) = t$ iff $s \in F(\{t\})$. To see that f is indeed well-defined, suppose by reductio that $s \in F(\{t\}) \cap F(\{t'\})$ with $t \neq t'$. By positive conjunctivity, $s \in F(\{t\} \cap \{t'\}) = F(\emptyset)$, contradicting the universal disjunctivity of F. So f is indeed well-defined.

Now take an arbitrary postcondition $Q \subseteq S$, and let V_Q be the set of all singleton subsets of Q. Because of universal disjunctivity of F, we then have

$$\begin{aligned} F(Q) &= F(\bigcup_{\{t\}\in V_Q}\{t\}) &&= \bigcup_{\{t\}\in V_Q} F(\{t\}) = \bigcup_{\{t\}\in V_Q}\{s \in S | f(s) = t\} \\ & &&= \{s \in S | f(s)\downarrow \wedge f(s) \in Q\} = f^{-1}[Q] = \phi(f)(Q) \end{aligned}$$

□

This result says that for any deterministic state transformer f, we have a univ. disj. pos. conj. predicate transformer F such that $F(Q) = f^{-1}[Q] = \{s \in S | f(s)\downarrow\ \wedge\ f(s) \in Q\}$, and vice versa. Thus, the function used in the equivalence links a state transformer with its corresponding weakest-precondition transformer. Hence, if we want to identify these two transformers, the result shows that DST and univ. disj. pos. conj. PT are really just "the same" classes, they are equally expressive.

2.3 Nondeterministic state transformers

For the next lemma which is central to the following results, observe that if a predicate transformer F is positively conjunctive it is also monotonic, i.e. $A \subseteq B$ implies $F(A) \subseteq F(B)$. For since $A \subseteq B$ is equivalent to $A \cap B = A$, we have $F(A \cap B) = F(A)$. By positive conjunctivity of F, we obtain $F(A) \cap F(B) = F(A)$, and hence $F(A) \subseteq F(B)$.

We can prove the main result using the following lemma from [AP86]. For any predicate transformer F satisfying certain conditions and for any $a \in F(S)$, it guarantees the existence of a set M_a which can be thought of as the intersection of all sets B such that $a \in F(B)$. When given a predicate transformer for which we want to define the corresponding state transformer, we will map state a to this set M_a.

Lemma 3 (Stability) *For every positively conjunctive predicate transformer F and for every state $a \in F(S)$ there is a set M_a with the property:*

$$\forall B \subseteq S(a \in F(B) \leftrightarrow M_a \subseteq B)$$

If F obeys the law of excluded miracle, M_a is nonempty.

Proof. Let $b_0, b_1, \cdots$ be an enumeration of the states $b \in S$ for which there is a set $B \subseteq S$ with $b \notin B$ and $a \in F(B)$. (If such a b does not exist, let $M_a := S$, and the claim follows. In case of an uncountable S, we enumerate using ordinals). Now let $B_0, B_1, \cdots$ be an enumeration of these subsets of S with $b_i \notin B_i$ and $a \in F(B_i)$, and define $M_a := \bigcap B_i$.

Because there is at least one such a B_i and because for all such i, $a \in F(B_i)$, positive conjunctivity gives us that $a \in F(M_a)$. If F obeys the law of excluded miracle, M_a will be nonempty. Since F is positively conjunctive, F is also monotonic. Thus, if $M_a \subseteq B$, $F(M_a) \subseteq F(B)$, and so $a \in F(B)$.

For the other implication, suppose by reductio that there is a set B such that $a \in F(B)$ while $M_a \not\subseteq B$. This means that there has to be some c

such that $c \in M_a$ and $c \notin B$. This c thus has to be one of the b_i, and so $c \notin B_i \supseteq \bigcap B_i = M_a$, contradicting the earlier assumption that $c \in M_a$. $\square$

We will now turn to the second model mentioned in the introduction, nondeterministic state transformers. As already mentioned, this model is only sufficient to model partial correctness, and so we obtain the following result regarding mapping state transformers to their weakest- liberal-precondition predicate transformers:

Theorem 4 $NST \cong_{\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]}$ *universally conjunctive PT*

Proof. Let $\phi(f) = \lambda Q.f^{-1}[\mathcal{P}(Q)]$, with $f \in NST$. Again, ϕ is total. Furthermore, $\phi(f)$ is guaranteed to be positively conjunctive by the previous fact, and $\phi(f)(S) = S$. For injectivity, suppose $f_0 \neq f_1$, i.e. for some s there is an x such that $x \in f_0(s)$ while $x \notin f_1(s)$. Then $s \notin \phi(f_0)(f_1(s))$, while $s \in \phi(f_1)(f_1(s))$, so $\phi(f_0) \neq \phi(f_1)$.

Lastly, to show that ϕ is onto, let F be some universally conjunctive predicate transformer. Then for $a \in S$, define $f(a) = M_a$, where M_a is the (possibly empty) set provided by the stability lemma. Then for any $B \subseteq S$,

$$\begin{aligned} \phi(f)(B) &= \{a \in S | f(a) \subseteq B\} &&= \{a \in S | M_a \subseteq B\} \\ & &&= \{a \in S | a \in F(B)\} = F(B) \end{aligned}$$

$\square$

The theorem establishes that for any nondeterministic state transformer in NST there exists a universally conjunctive predicate transformer F such that $F(Q) = f^{-1}[\mathcal{P}(Q)] = \{s \in S | f(s) \subseteq Q\}$, and vice versa. While this looks the same as before, F can only be seen as the associated weakest-liberal-precondition transformer. Since we mentioned before that NST is not rich enough to model failure of termination for programs with multiple execution paths, we can see the output set as the final states of terminating executions. The above equation also shows that if $f(s) = \emptyset$, i.e. if all executions from s fail, $s \in F(Q)$ for all Q which is what we want for the weakest liberal precondition.

2.4 Nondeterministic state transformers with failure state

Turning towards the third model, nondeterministic state transformers plus failure state, we will now have the necessary equipment to talk about the total correctness of nondeterministic programs, i.e. about the correspondence between state transformers and their weakest-precondition counterparts:

Theorem 5 $NST_\perp \succeq_{\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]}$ *positively conjunctive PT*

Proof. Let $\phi(f) = \lambda Q.f^{-1}[\mathcal{P}(Q)]$, with $f \in NST_\perp$. Again, ϕ is total and, $\phi(f)$ is guaranteed to be positively conjunctive by the previous fact.

To show that ϕ is onto, let F be some positively conjunctive predicate transformer. Then for $a \in S$, define f by

$$f(a) = \begin{cases} M_a & \text{if } a \in F(S) \\ S_\perp & \text{if } a \notin F(S) \end{cases}$$

where M_a is the (possibly empty) set provided by the stability lemma. Then $\phi(f) = F$, by the same argument as for NST. □

Note that within the model $NST_\perp$, the intuitive notion of the weakest liberal precondition turns out to be somewhat ambiguous: It is unclear how we interpret a state transformer which maps a state s to the empty set. (This possibility was left open to accommodate the transformers used in Dynamic Logic later on.) Since we used $\perp$ as a failure state, this cannot be taken to mean that the program fails to terminate successfully. Furthermore, we already described in the introduction of this section when discussing the inadequacy of NST that modelling failure by the empty set leads to problems. As to the weakest precondition, this seems to create two possible ways to formalize the weakest precondition of a program with state transformer f under postcondition Q:

$$\{s \in S | f(s) \subseteq Q\} \tag{1}$$

$$\{s \in S | f(s) \subseteq Q \wedge f(s) \neq \emptyset\} \tag{2}$$

Like in Dynamic Logic, we will interpret $f(s) = \emptyset$ as saying that the program is not applicable in state s. We leave this possibility open on purpose since we may want to introduce programs/actions whose application depends on certain presuppositions (think of e.g. speech acts in natural language and their presuppositions), where we do not want to say that the program/action fails if these presuppositions do not hold. Hence, we will adopt (1) as a formalization of the concept of a weakest precondition, i.e. we will say that if a program is not applicable in a given state, all of its executions from that state (of which there are none) will successfully terminate satisfying any postcondition. A consequence of this is that we cannot assume that $wp_P(false) = false$ for any program p, i.e. we cannot assume that the predicate transformers corresponding to $NST_\perp$ all satisfy the law of excluded miracle. Theorem 5 in fact demonstrates this. Lastly, note that if we assume that all programs are applicable in every state (which is what Dijkstra assumes in [Dij76]), the two definitions coincide and there is no more ambiguity. If we wanted to make this assumption, we could modify the definition of $NST_\perp$ as follows:

Definition 4 *Given the set of states S, define $NST'_\perp$ as the class of all total functions $f : S \to (\mathcal{P}(\mathcal{S} \cup \{\perp\}) - \{\emptyset\})$, where $\perp \notin S$.*

The result corresponding to the equivalence theorem for $NST_\perp$ (where "positively conjunctive LEM PT" denotes the class of predicate transformers which are positively conjunctive and obey the law of excluded miracle) would then be

Theorem 6 $NST'_\perp \succeq_{\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]}$ *positively conjunctive LEM PT*

Proof. The proof is virtually identical to the one for $NST_\perp$. We use again the stability lemma, but since the predicate transformers now satisfy the stronger requirement, we indeed obtain a nonempty set M_a by the lemma. □

Returning to the interpretation of theorem 5, it states that for any nondeterministic state transformer in $NST_\perp$ there is some positively conjunctive predicate transformer F such that $F(Q) = f^{-1}[\mathcal{P}(Q)] = \{s \in S | f(s) \subseteq Q\}$, and vice versa. Since $Q \subseteq S$, $\perp \notin Q$ for any Q, and so if $\perp \in f(s)$, we have $s \notin F(Q)$. So this is again the right function to link a nondeterministic state transformer to its weakest-precondition transformer; however, we cannot say that the two classes contain the same amount of information, for the mapping used in the result is actually not injective. Take for example two state transformers f_0 and f_1 such that $f_0(s) = \{s, \perp\}$, $f_1(s) = \{\perp\}$ and $f_0(s') = f_1(s')$ for all $s' \neq s$. Both of these will be mapped to the same predicate transformer. From the perspective of wp-transformers, all that matters is that for both state transformers, not all computations from state s will terminate. The information that for f_0, there still is one computation that does terminate is irrelevant if we are only interested in the states from which *all* computations terminate. Hence, it is now clear why the theorem cannot provide an injective mapping.

If we want to capture all of the information present in the state transformers, we need to map each state transformer to two predicate transformers, the wp- and wlp-transformer. Note that the two state transformers f_0, f_1 mentioned above, while being mapped to the same wp-transformer, are not mapped to the same wlp-transformer: For a given state transformer f, $\lambda Q.f^{-1}[\mathcal{P}(Q \cup \{\perp\})]$ maps f to its corresponding wlp-transformer, since it allows the output set of a state transformer to contain $\perp$. The relation between wlp- and wp-transformer is expressed by the following equation:

$$\begin{aligned} wp(X) &= f^{-1}[\mathcal{P}(X)] = \{s \in S | f(s) \subseteq X\} \\ &= \{s \in S | f(s) \subseteq X \cup \{\perp\}\} \cap \{s \in S | \perp \notin f(s)\} \\ &= \{s \in S | f(s) \subseteq X \cup \{\perp\}\} \cap \{s \in S | f(s) \subseteq S\} = wlp(X) \cap wp(S) \end{aligned}$$

Hence, it is sufficient to associate with every state transformer in $NST_\perp$ its wlp-transformer together with the set $wp(S)$, since the wp-transformer can be defined using these. Now if we extend our notion of equivalence from transformer classes to classes in general, we can show that wlp-transformer plus the set $wp(S)$ do indeed capture all the information of a state transformer:

Theorem 7 $NST_\perp \cong_{\lambda f.(\lambda Q.f^{-1}[\mathcal{P}(Q \cup \{\perp\})], f^{-1}[\mathcal{P}(S)])}$ *universally conjunctive* $PT \times \mathcal{P}(S)$

Proof. Let $\phi(f) = (\lambda Q.f^{-1}[\mathcal{P}(Q \cup \{\perp\})], f^{-1}[\mathcal{P}(S)])$ and let $\delta(f)$ be the left element of $\phi(f)$. Clearly, ϕ is total. Also, $\delta(f)$ will be universally conjunctive: using the earlier set-theoretic facts, we obtain positive conjunctivity, and since $f^{-1}[\mathcal{P}(S \cup \{\perp\})] = S$, we have universal conjunctivity.

For injectivity, suppose that $f_0 \neq f_1$, i.e. for some s, $f_0(s) \neq f_1(s)$. Suppose without loss of generality that $x \in f_0(s)$ while $x \notin f_1(s)$. In the first case, $x = \perp$, in which case, $s \notin f_0^{-1}[\mathcal{P}(S)]$ while $s \in f_1^{-1}[\mathcal{P}(S)]$. In the other case, $x \neq \perp$, in which case $s \notin \delta(f_0)(f_1(s) - \{\perp\})$ while $s \in \delta(f_1)(f_1(s) - \{\perp\})$. In either case, we have $\phi(f_0) \neq \phi(f_1)$.

Lastly, we show that ϕ is onto. Take any universally conjunctive predicate transformer F and any $Q \subseteq S$. Then for $a \in S$, define f by

$$f(a) = \begin{cases} M_a & \text{if } a \in Q \\ M_a \cup \{\perp\} & \text{if } a \notin Q \end{cases}$$

where M_a is the (possibly empty) set provided by the stability lemma. Then for any $B \subseteq S$ we have

$$\begin{aligned} \delta(f)(B) &= \{a \in S | f(a) \subseteq (B \cup \{\perp\})\} &&= \{a \in S | M_a \subseteq B\} \\ & &&= \{a \in S | a \in F(B)\} = F(B) \end{aligned}$$

Furthermore,

$$f^{-1}[\mathcal{P}(S)] = \{a \in S | f(a) \subseteq S\} = \{a \in S | a \in Q\} = Q$$

This completes the proof that ϕ is onto. □

The results regarding the basic transformer classes are summarized in the following table:

ST Class	Correspond.	Function	Interpret.	PT Class
DST	$\cong$	$\lambda f.\lambda Q.f^{-1}[Q]$	wp	univ. disj. & pos. conj.
NST	$\cong$	$\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]$	wlp	universally conjunctive
$NST_\perp$	$\succeq$	$\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]$	wp	positively conjunctive

2.5 Fine structure: varieties of nondeterminism

Within $NST_\perp$, one can make a rather natural division of state transformers into three classes: deterministic, bounded nondeterministic, and unbounded nondeterministic. The distinction between the last two subclasses will become important later on.

Definition 5 (Subclasses of $NST_\perp$) *Given the set of states S, we define the following classes of functions:*

- *deterministic* $NST_\perp := \{f \in NST_\perp \mid \forall s \in S : |f(s)| = 1\}$
- *bounded* $NST_\perp := \{f \in NST_\perp \mid \forall s \in S : |f(s)| < \omega \vee \perp \in f(s)\}$
- *unbounded* $NST_\perp := \{f \in NST_\perp \mid \exists s \in S : |f(s)| \geq \omega \wedge \perp \notin f(s)\}$

Thus, we have

$$\text{deterministic } NST_\perp \quad \subseteq \quad \text{bounded } NST_\perp \quad \subseteq \quad NST_\perp$$

and bounded $NST_\perp$ as the complement of unbounded $NST_\perp$ under $NST_\perp$. Note that deterministic state transformers are applicable in all states, i.e. $f(s) = \emptyset$ is not allowed.

We now have two ways to model deterministic programs, either by DST or by deterministic $NST_\perp$. The connection between these two classes is easy to establish:

Lemma 8 *deterministic* $NST_\perp \cong_{\lambda f.\lambda s.\tau(f,s)} DST$*, where* $\tau(f,s) = x$ *s.t.* $\{x\} = f(s)$ *if* $f(s) \neq \{\perp\}$*, and undefined otherwise.*

Proof. Let $\phi(f) = \lambda s.\tau(f,s)$ with $f \in$ deterministic $NST_\perp$. Clearly, ϕ is into DST, injective and total. To show that it is also onto, take any $g \in DST$, and let $f \in NST_\perp$ be defined by $f(s) = \{g(s)\}$ if $g(s)$ is defined, and $\{\perp\}$ otherwise. Then $\phi(f) = g$. □

Now we can extend the *wp*-correspondence results of the previous section to the subclasses just introduced.

Theorem 9 *deterministic* $NST_\perp \cong_{\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]}$ *universally disjunctive positively conjunctive* PT

Proof. Since $DST \cong_{\lambda f.\lambda Q.f^{-1}[Q]}$ universally disjunctive PT, functional concatenation with ϕ of the previous lemma gives us that deterministic $NST_\perp$ $\cong_{\lambda f.\lambda Q.f^{-1}[Q]\circ\phi}$ universally disjunctive positively conjunctive PT. We can simplify the equivalence function to the desired one as follows:

$$\begin{aligned}
\lambda f.\lambda Q.f^{-1}[Q] \circ \phi &\quad \text{with } f \in DST \\
= \lambda f.\lambda Q.(\phi(f))^{-1}[Q] &\quad \text{with } f \in \text{deterministic } NST_\perp \\
= \lambda f.\lambda Q.\{s \in S | \phi(f)(s) \in Q\} & \\
= \lambda f.\lambda Q.\{s \in S | f(s) \subseteq Q\} &\quad = \lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]
\end{aligned}$$

□

Turning to bounded nondeterminism, we have the following equivalence:

Theorem 10 *bounded* $NST_\perp \succeq_{\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]}$ *positively conjunctive or-continuous* PT

Proof. Let $\phi(f) = \lambda Q.f^{-1}[\mathcal{P}(Q)]$, with $f \in$ bounded $NST_\perp$. Like in the previous results of this kind, it is easily checked that ϕ is total, and that $\phi(f)$ is positively conjunctive. To see that $\phi(f)$ is always or-continuous, assume that f is bounded nondeterministic. Then if $C_0 \subseteq C_1 \subseteq C_2 \subseteq \cdots$, we have $\phi(f)(\bigcup_{i<\omega} C_i) = \{s \in S | f(s) \subseteq \bigcup_{i<\omega} C_i\}$. Because $f(s)$, provided it does not contain $\perp$, contains only finitely many elements, there must be a C_n with $f(s) \subseteq C_n$. Hence, the last set equals $\bigcup_{i<\omega}\{s \in S | f(s) \subseteq C_i\} = \bigcup_{i<\omega} \phi(f)(C_i)$.

Lastly, we show that ϕ is onto. We know already by theorem 5 that for any positively conjunctive or-continuous F there is a $f \in NST_\perp$ such that $\phi(f) = F$. All we need to check is that f is bounded. So suppose by reductio that f in unbounded, i.e for some s_0 we have $f(s_0) = \{t_0, t_1, t_2, \cdots\}$. Then define $C_0 = \{t_0\}$ and $C_{i+1} = C_i \cup \{t_{i+1}\}$. So $C_0 \subseteq C_1 \subseteq C_2 \subseteq \cdots$, and $s_0 \in F(\bigcup_{i<\omega} C_i)$, while $s_0 \notin \bigcup_{i<\omega} F(C_i)$, contradicting the fact that F was supposed to be or-continuous. □

Again, the function used in the equivalences links a state transformer with its corresponding weakest-precondition transformer, so we obtain the following table for weakest-precondition correspondence under the equivalence function $\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]$:

ST Class	Correspondence	PT Class
$NST_\perp$	$\succeq$	positively conjunctive
bounded $NST_\perp$	$\succeq$	pos. conj. & or-continuous
deterministic $NST_\perp$	$\cong$	pos. conj. & universally disjunctive

Note that the inclusion relation which holds between the state transformers top-down also holds between the predicate transformers.

3 PT- and ST-Semantics for Representative Program Classes

After this abstract discussion of the relation between predicate transformers and state transformers in general, we now turn to concrete examples of such transformers, namely the transformers associated with well-known programming constructs. We will introduce these programming constructs in order of increasing complexity, i.e. the transformers associated with these constructs will be less and less restricted.

Starting with the deterministic $WHILE$ language and some comments about the fixpoint definitions used for its semantics, we will move to the bounded nondeterministic $GUARD$ language and then to two unbounded nondeterministic languages, giving PT-semantics (based on [BES95] and [Bac81]) and ST-semantics (see e.g. [NN92]) for these languages and pointing out correspondences by applying the abstract results obtained in the previous section. Finally, an example of a highly expressive programming construct will be given (based on [Hes90]) which cannot be modeled by a state transformer anymore.

3.1 A deterministic program class

Definition 6 ($WHILE$) *Given some language L with constant-, function-, and predicate-symbols, we define*

$$\begin{aligned} WHILE := \ & \mathsf{skip} \mid \mathsf{abort} \mid x := t \mid S_1; S_2 \mid \\ & \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2 \mid \mathsf{while}\ b\ \mathsf{do}\ S \end{aligned}$$

where S, S_1, S_2 range over $WHILE$, t is a term of L and b is a boolean expression of L containing no quantifiers, unless stated otherwise. If we want to make L explicit, we denote the program class as $WHILE_L$.

Note that since we decided to identify abort with while $true$ do skip, there is actually no need to have abort in the language since it is definable. Nonetheless, since it is usually considered to be part of the language and since we may in the future want to distinguish it from while $true$ do skip, we included it.

For a given language L, a given interpretation $\mathcal{I}$ for L and the set of states S containing all variable assignments for $\mathcal{I}$, the standard state transformer semantics given in terms of DST (i.e. partial maps on S) e.g. in [NN92] is as follows for $s \in S$:

$$\begin{array}{lcl}
f_{\mathsf{skip}}(s) & := & s \\
f_{\mathsf{abort}}(s) & := & \mathit{undefined} \\
f_{x:=t}(s) & := & s[x \mapsto t^{s,\mathcal{I}}] \\
f_{P_1;P_2}(s) & := & f_{P_2} \circ f_{P_1}(s) \\
f_{\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2}(s) & := & \mathit{cond}(b^{\mathcal{I}}, f_{P_1}, f_{P_2})(s) \\
f_{\mathsf{while}\ b\ \mathsf{do}\ P} & := & \mu m \in DST.\ \mathit{cond}(b^{\mathcal{I}}, m \circ f_P, id)
\end{array}$$

where $\mathit{cond}(B, f_1, f_2)(s) = f_1(s)$ if $s \in B$ and $f_2(s)$ otherwise. $\mu m.F(m)$ designates the least fixpoint of the operator F, i.e. the state transformer $m \in DST$ such that $F(m) = m$, and for all $r \in DST$ such that $F(r) = r$, $m \sqsubseteq r$. Regarding the ordering, we use the standard ordering of partial functions based on information content, i.e. for any $f, g \in DST$, $f \sqsubseteq g$ iff $\forall s, s' \in S : f(s) = s' \rightarrow g(s) = s'$. So the least element of this ordering is actually f_{abort}.

Definition 7 (ST_{WHILE}) *Given an interpretation $\mathcal{I}$ for the language L, and the set S of states (i.e. variable assignments), define the class of deterministic state transformers $ST_{WHILE^{\mathcal{I}}_L}$ (or ST_{WHILE} if L and $\mathcal{I}$ are clear) inductively as follows: It includes f_{skip}, f_{abort} and $f_{x:=t}$; if f_P is in it, $f_{\mathsf{while}\ b\ \mathsf{do}\ P}$ is also included; if f_{P_1} and f_{P_2} are members, $f_{P_1;P_2}$ and $f_{\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2}$ are also members.*

As for the corresponding PT-semantics (see e.g. [BES95]), we have for $N \subseteq S$:

$$\begin{array}{lcl}
wp_{\mathsf{skip}}(N) & := & N \\
wp_{\mathsf{abort}}(N) & := & \emptyset \\
wp_{x:=t}(N) & := & \{s \in S | s[x \mapsto t^{s,\mathcal{I}}] \in N\} \\
wp_{P_1;P_2}N & := & wp_{P_1}(wp_{P_2}(N)) \\
wp_{\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2}(N) & := & (b^{\mathcal{I}} \cap wp_{P_1}(N)) \cup ((\neg b)^{\mathcal{I}} \cap wp_{P_2}(N)) \\
wp_{\mathsf{while}\ b\ \mathsf{do}\ P}(N) & := & \mu Q \subseteq S.\ \big(((\neg b)^{\mathcal{I}} \cap N) \cup (b^{\mathcal{I}} \cap wp_P(Q)) \big)
\end{array}$$

Definition 8 (PT_{WHILE}) *Given an interpretation $\mathcal{I}$ for the language L, and the set S of states (i.e. variable assignments), define the class of predicate transformers $PT_{WHILE^{\mathcal{I}}_L}$ (or PT_{WHILE} if L and $\mathcal{I}$ are clear) inductively to be the smallest class including all the predicate transformers just defined: It includes wp_{skip}, wp_{abort} and $wp_{x:=t}$; if wp_P is in it, $wp_{\mathsf{while}\ b\ \mathsf{do}\ P}$ is also included; if wp_{P_1} and wp_{P_2} are members, $wp_{P_1;P_2}$ and $wp_{\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2}$ are also members.*

The reader is invited to verify the following result which demonstrates that the two semantics do indeed correspond the way we intend them to:

Fact 11 $ST_{WHILE^{\mathcal{I}}_L} \cong_{\lambda f.\lambda Q.f^{-1}[Q]} PT_{WHILE^{\mathcal{I}}_L}$

Since ST_{WHILE} is a subclass of DST, theorem 2 from the previous section yields the following (which can also be verified directly):

Theorem 12 *All predicate transformers in PT_{WHILE} are universally disjunctive and positively conjunctive.*

3.2 Complexity of $WHILE$-fixpoints

For the while-construct, both semantics given in the last subsection made use of a fixpoint construction. The development of the fixpoint theory associated with these semantics will not be given in detail (the relevant facts can be found e.g. in [AP86] and [DS90]), but for the less common PT-semantics, results regarding the existence of an upper bound shall be given which will be used later:

Fact 13 *Let $F_X(Y) := ((\neg b)^{\mathcal{I}} \cap X) \cup (b^{\mathcal{I}} \cap wp_P(Y))$. If wp_P is or-continuous, then*

$$wp_{\textsf{while } b \textsf{ do } P}(X) = \bigcup_{0 \leq i < \omega} F_X^i(\emptyset)$$

where $F_X^0(Y) = Y$ and $F_X^{i+1}(Y) = F_X(F_X^i(Y))$.

Corollary 14 *For $P \in WHILE$,*

$$wp_{\textsf{while } b \textsf{ do } P}(X) = \bigcup_{0 \leq i < \omega} F_X^i(\emptyset)$$

Comparing the complexity of the fixpoint definitions used in the two semantics, the PT-approach turns out to be somewhat simpler. In the predicate transformer approach, we define an ordering on predicates, namely set inclusion, and we calculate the fixpoint over predicate transformer outputs, namely preconditions. In the state transformer approach on the other hand, we define an ordering on state transformers themselves, and calculate the fixpoint over the state transformers, not over the state transformer outputs. Formally, the difference is

$$\lambda N.\mu Q.\delta(N,Q) \quad \text{versus} \quad \mu f.\lambda s.\gamma(f,s)$$

for predicate transformers versus state transformers respectively, where $Q \subseteq S$ and $f \in DST$. A unary fixpoint operator is sufficient for predicate transformers. A predicate is simply mapped to the least predicate satisfying a certain condition. The binary fixpoint operator needed for state transformers operates on more complex functional objects, namely state transformers. As a consequence, the necessary partial ordering, the subset relation, is much simpler and more well behaved then the ordering needed for state transformers. The difference becomes even more apparent when we turn towards nondeterministic state transformers and their ordering.

3.3 A bounded nondeterministic program class

Definition 9 ($GUARD$) *Given some language L with constant-, function-, and predicate-symbols, we define*

$$\begin{array}{rl} GUARD := & \mathsf{skip} \mid \mathsf{abort} \mid x := t \mid S_1; S_2 \mid \\ & \mathsf{if}\ b_1 \to S_1 \| \cdots \| b_n \to S_n\ \mathsf{fi} \mid \\ & \mathsf{while}\ b\ \mathsf{do}\ S \end{array}$$

Again, $S, S_1, \cdots, S_n$ range over $GUARD$, t is a term of L and unless noted otherwise, b and the b_i are pure boolean expression of L, not necessarily mutually exclusive. If we want to make L explicit, we denote the program class as $GUARD_L$.

Note that the nondeterministic loop-construction $\mathsf{do}\ b_1 \to P_1 \| \cdots \| b_n \to P_n\ \mathsf{od}$ is now definable by $\mathsf{while}\ b_1 \vee \cdots \vee b_n\ \mathsf{do}\ \mathsf{if}\ b_1 \to P_1 \| \cdots \| b_n \to P_n\ \mathsf{fi}$, so that we will not include this nondeterministic loop construction as basic in $GUARD$.

The ST-semantics for this language would use $NST_\perp$ (in fact $NST'_\perp$) rather than DST in order to handle the nondeterminism, and can be adapted from the semantics for $WHILE$ (see e.g. [AP86] or [Bac80] for the technical details regarding ordering of nondeterministic state transformers etc.). Note e.g. that now $f_{\mathsf{abort}}(s) = \{\perp\}$. The state transformer for the nondeterministic conditional statement is

$$f_{\text{if } b_1 \to P_1 \| \cdots \| b_n \to P_n \text{ fi}}(s) := \begin{cases} \bigcup\{f_{P_i}(s) \mid s \in b_i^{\mathcal{I}}\} & \text{if } s \in \bigcup_{i=1}^n b_i^{\mathcal{I}} \\ \{\perp\} & \text{otherwise} \end{cases}$$

The definition of ST_{GUARD} can now be adapted from the definition of ST_{WHILE} in the obvious way.

The PT-semantics of this language is the same as for $WHILE$ except that the predicate transformer for the deterministic conditional construct is replaced by the following:

$$wp_{\text{if } b_1 \to P_1 \| \cdots \| b_n \to P_n \text{ fi}}(N) := (\bigcup_{i=1}^{n} b_i^{\mathcal{I}}) \cap \bigcap_{i=1}^{n}((\neg b_i)^{\mathcal{I}} \cup wp_{P_i}(N))$$

Again, the definition of PT_{GUARD} is easily adapted from the definition of PT_{WHILE}.

As with the $WHILE$ language, we again have a correspondence between the transformer classes. Given fact 11, one only needs to demonstrate the correspondence for the nondeterministic conditional. Note that we can only obtain a weaker correspondence now. For example the state transformers of abort and $\mathsf{if}\ true \to \mathsf{abort}\ \| true \to \mathsf{skip}\ \mathsf{fi}$ will both correspond to the same wp-transformer.

Fact 15 $ST_{GUARD_L^{\mathcal{I}}} \succeq_{\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]} PT_{GUARD_L^{\mathcal{I}}}$

It can be verified that ST_{GUARD} is bounded nondeterministic. We then have as a corollary to theorem 10 from the previous section:

Theorem 16 *PT_{GUARD} is or-continuous and positively conjunctive, i.e. PT_{GUARD} is a subclass of or-continuous positively conjunctive PT.*

3.4 Two unbounded nondeterministic program classes

3.4.1 Adding nondeterministic assignment to $GUARD$

One well-known way to extend $GUARD$ is to introduce a nondeterministic assignment statement $x := y.Q$ which is interpreted as "set x to some value y for which $Q(y)$ holds". In this full generality, nondeterministic assignment has been studied e.g. in [Bac80] and [Bac81]. An instance of this statement, $x := y.y > 0$ (i.e. "set x to any positive integer"), is already mentioned in [Dij76], but also the instance $x :=?$, i.e. "set x to any integer" has been studied, e.g. in the context of Dynamic Logic ([HKT95]). Note that the standard assignment statement $x := t$ is an instance of this statement, namely $x := y.(y = t)$.

Definition 10 **($GUARD^{\times}$)** *$GUARD_L^{\times}$ extends $GUARD_L$ with the nondeterministic assignment statement, $x := y.Q$, where Q is a formula of L which may contain free variables and quantifiers.*

The state transformer for nondeterministic assignment is

$$f_{x:=y.Q}(s) := \begin{cases} \{s[x \mapsto a] \in S | s[y \mapsto a] \in Q^{\mathcal{I}}\} & \text{if } \exists a : s[y \mapsto a] \in Q^{\mathcal{I}} \\ \{\bot\} & \text{otherwise} \end{cases}$$

Definition 11 **($ST_{GUARD^{\times}}$)** $ST_{GUARD^{\times}} := ST_{GUARD} \cup \{f_{x:=y.Q}\}$

Given the weakest precondition correspondence we would like, we will in this case derive the PT-semantics from the ST-semantics:

$$\begin{aligned} wp_{x:=y.Q}(N) &= f^{-1}_{x:=y.Q}[\mathcal{P}(N)] = \{s \in S | f_{x:=y.Q}(s) \subseteq N\} \\ &= \{s \in S | \exists a : s[y \mapsto a] \in Q^{\mathcal{I}} \wedge \forall a : s[y \mapsto a] \in Q^{\mathcal{I}} \rightarrow s[x \mapsto a] \in N\} \end{aligned}$$

Definition 12 **($PT_{GUARD^{\times}}$)** $PT_{GUARD^{\times}} := PT_{GUARD} \cup \{wp_{x:=y.Q}\}$

If we take the correspondence between ST_{GUARD} and PT_{GUARD} from the previous subsection as given, our derivation of $wp_{x:-y.Q}$ from $f_{x:=y.Q}$ establishes the following:

Fact 17 $ST_{GUARD_L^{\times},\mathcal{I}} \succeq_{\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]} PT_{GUARD_L^{\times},\mathcal{I}}$

It can be easily checked that $wp_{x:=y.(y>0)}$ yields a not or-continuous predicate transformer in the language of arithmetic under the standard interpretation, take for instance the sequence of sets $C_0 \subseteq C_1 \subseteq \cdots$ with $C_i := \{s \in S | 0 \leq s(x) \leq i\}$. Thus we have by theorem 10:

Theorem 18 *$ST_{GUARD_L^{\times},\mathcal{I}}$ can contain unbounded nondeterministic transformers depending on the choice of L and $\mathcal{I}$.*

3.4.2 Adding the Kleene star to $GUARD$

Another way to allow for unbounded nondeterminism is to add the Kleene star * to $GUARD$. Well known e.g. from dynamic logic, P^* means "execute P an arbitrary finite number of times".

Definition 13 **($GUARD^*$)** *$GUARD^*_L$ properly includes $GUARD_L$ and extends it in the obvious way: If P is a statement of $GUARD^*_L$ then P^* is a statement of $GUARD^*_L$ as well.*

The state transformer associated with this operation is:

$$f_{P^*}(s) := \bigcup_{i=0}^{\omega} f^i_P(s)$$

where $f^0_P = f_{\mathsf{skip}}$ and $f^{n+1}_P = f^n_P \circ f_P$, with functional composition adapted in the obvious way for sets of outputs.

Definition 14 **(ST_{GUARD^*})** *ST_{GUARD^*} includes ST_{GUARD} and extends it in the natural way: If f_P is in ST_{GUARD^*}, then f_{P^*} is in it as well.*

As for the predicate transformer, we have

$$wp_{P^*}(N) := \bigcap_{i=0}^{\omega} wp^i_P(N)$$

where $wp^0_P = wp_{\mathsf{skip}}$ and $wp^{n+1}_P = wp^n_P \circ wp_P$.

Definition 15 **(PT_{GUARD^*})** *PT_{GUARD^*} includes PT_{GUARD} and extends it in the natural way: If wp_P is in PT_{GUARD^*}, then wp_{P^*} is in it as well.*

The correspondence between f_{P^*} and wp_{P^*} is verified easily, and so we again obtain

Fact 19 $ST_{GUARD^{*,\mathcal{I}}_L} \succeq_{\lambda f.\lambda Q.f^{-1}[\mathcal{P}(Q)]} PT_{GUARD^{*,\mathcal{I}}_L}$

Again using the language of arithmetic with its standard model, $wp_{(x:=x+1)^*}$ yields a not or-continuous predicate transformer. So like with $GUARD^\times$ we have

Theorem 20 *$ST_{GUARD^{*,\mathcal{I}}_L}$ can contain unbounded nondeterministic transformers depending on the choice of L and $\mathcal{I}$.*

3.5 Beyond unbounded nondeterminism: reaching the limits of $NST_\perp$

The program constructions just introduced for unbounded nondeterminism have already been very powerful in that there is in general no continuous predicate transformer which can model them. But we can go yet one step further by looking at program constructions whose associated *wp*-transformer violates positive conjunctivity. By theorem 5, a program P cannot be modeled within $NST_\perp$ if wp_P violates positive conjunctivity.

An example of such a programming construct is called *angelic choice*; it is described e.g. in [Hes90]. Given two programs P_1 and P_2, we define

$$wp_{P_1 \diamond P_2}(N) := wp_{P_1}(N) \cup wp_{P_2}(N)$$

This indeed implements some kind of angelic choice: We choose the right program execution depending on the postcondition. As Hesselink points out in [Hes90], angelic choice is in some way the natural counterpart of the usual nondeterministic choice $P_1 \vee P_2$ which he calls *demonic* choice, where the predicate transformer is defined as $wp_{P_1 \vee P_2}(N) := wp_{P_1}(N) \cap wp_{P_2}(N)$. But while demonic choice can be modeled by a state transformer (it is a special case of the nondeterministic conditional statement), angelic choice seems indeed "too heavenly to be true" (the symmetry on the level of predicate transformers is thus misleading):

Theorem 21 *$wp_{P_1 \diamond P_2}$ is not positively conjunctive; consequently, angelic choice cannot be modeled by an element of $NST_\perp$ (by theorem 5).*

Proof. Let $P = (x := 0 \diamond x := 1)$, with states being variable assignments for the variable x, and $wp_{x:=n}$ defined in the obvious way and in the standard model, so that $wp_{x:=n}(x = n) = S$ and $wp_{x:=n}(x = m) = \emptyset$ for $n \neq m$, where $x = n$ denotes the set of states where $x = n$.

Then $wp_P(x = 0) = wp_P(x = 1) = S$, and so $wp_P(x = 0) \cap wp_P(x = 1) = S$, while $wp_P(x = 0 \cap x = 1) = wp_P(\emptyset) = \emptyset$. □

This result can be either seen as demonstrating that state transformers are not rich enough to model all programs, or as demonstrating that programs involving angelic choice do not exist, i.e. are not really programs after all. (The latter view must be attributed to Dijkstra.) We believe that while angelic choice may not correspond to any realistic programming construct, it may very well turn out to be a useful concept within the broad perspective of modeling actions. In this broad perspective, we can regard angelic choice as deliberate or goal-sensitive action. Consider e.g. an agent who has knowledge about the effects of two actions P_1 and P_2 and the goal G which she wants to bring about. In this case, we may want to say that the situations in which the agent can successfully accomplish the goal are all those from which either P_1 or P_2 (or both) can accomplish the goal. So from this broader perspective, constructs such as angelic choice whose *wp*-transformer violates positive conjunctivity are very reasonable to consider.

4 Formalizing PT-Semantics: Weakest Preconditions in First-Order and Infinitary Logic

So far, we have discussed two semantic conceptualizations of programs, emphasizing the connections between the two. Since we introduced these two approaches so that we could reason about the (total) correctness of programs, our next step must be to tie the semantic concept of a weakest precondition to a logical syntax. Furthermore, we are also interested in obtaining some information about the complexity of reasoning about a given program class by looking at the complexity of the logic needed to handle this class.

Dijkstra himself proposes a reasoning system for weakest preconditions. The syntactic framework used in [DS90] is basically a higher-order logic which includes a special bracket-operator $[X]$ which means something like "X holds in all states". While this formal framework may be perfectly adequate for the working computer scientist, it is not of much use if one wants to learn about the complexity of the weakest preconditions of a given class of programs. For this, one must look to weaker logical systems and see how far one can formalize weakest preconditions in them. Nonetheless, one could try to prove Dijkstra's axiomatization complete for the semantics given in the previous section. We do not know whether this has been done yet.

In this section, we will first look at how far the predicate transformer approach for deterministic and bounded nondeterministic programs can be formalized within standard first-order logic. The limitations of this logic naturally lead to the consideration of infinitary logic which overcomes some of the shortcomings of first-order logic, but also runs into problems for unbounded nondeterministic programs. Most of the results in this section are from [Bac81].

The central notion of this section is that of a logic's expressive power, i.e. the degree to which it can represent weakest preconditions.

Definition 16 (Global Expressiveness) *Let P be a statement in $STAT_L$, and let W and Q be formulas of the logic under consideration. Then W* globally expresses the weakest precondition *of P under Q iff for all models $\mathcal{I}$ of the logic, $W^{\mathcal{I}} = wp_P(Q^{\mathcal{I}})$.*

We will say that a logic is globally expressive for $STAT_L$ *iff for all programs $P \in STAT_L$, and all formulas Q there is a formula W such W globally expresses the weakest precondition of P under Q. Finally, a logic is* globally expressive for $STAT$ *iff for all languages L, the logic is globally expressive for $STAT_L$.*

Note. We can define an analogous notion of *local expressiveness* with respect to a given model of the language. One result regarding this weaker notion of expressiveness will be given later on. **End of Note.**

4.1 Deterministic and bounded nondeterministic programs

4.1.1 First-Order Logic

For all $WHILE$ constructs except the while-loop, the weakest preconditions can be (globally) expressed in first-order logic. For any $P \in WHILE$, $WP_P(\phi)$ globally expresses the weakest precondition of P under ϕ:

$$
\begin{array}{lcl}
WP_{\mathsf{skip}}(\phi) & := & \phi \\
WP_{\mathsf{abort}}(\phi) & := & false \\
WP_{\mathrm{x:=t}}(\phi) & := & \phi^t_x \\
WP_{P_1;P_2}\phi & := & wp_{P_1}(wp_{P_2}(\phi)) \\
WP_{\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2}(\phi) & := & (b \to wp_{P_1}(\phi)) \wedge (\neg b \to wp_{P_2}(\phi))
\end{array}
$$

Nonetheless, Back (in [Bac81]) shows the following negative result:

Theorem 22 *$L_{\omega\omega}$ is not globally expressive for $WHILE$.*

Proof. Let L be the language of groups with constant 1 for the unit element, unary function $^{-1}$ for inverse, and binary function $\times$ for multiplication. Let P be the program:

$$
\begin{array}{l}
x := y \\
\text{while } x \neq 1 \text{ do } x := x \times y
\end{array}
$$

Let Φ be the finite set of first-order axioms for Abelian groups. Suppose now by reductio that $L_{\omega\omega}$ is globally expressive for $WHILE_L$. Then there is some first-order formula $W(y)$ which in any model holds exactly in those states where this program is guaranteed to terminate. Hence $\Phi \cup \{\forall y W(y)\}$ would then characterize the Abelian groups where for any element a there is some n such that $a^n = 1$, the class of torsion Abelian groups. It has been shown however ([Bar77]) that this class is not axiomatizable in $L_{\omega\omega}$. Hence, $L_{\omega\omega}$ is not globally expressive for $WHILE_L$, and thus not globally expressive for $WHILE$. □

Excursion on Local Expressiveness
After this negative result, one can ask whether we might achieve a weaker positive result in terms of local expressiveness w.r.t. a particular language and model.

Definition 17 (Local Expressiveness) *Let P be a statement in $STAT_L$, and let W and Q be formulas of the logic under consideration. Then, W* locally expresses the weakest precondition *of P under Q relative to model $\mathcal{I}$ iff $W^{\mathcal{I}} = wp_P(Q^{\mathcal{I}})$.*

We will say that given $STAT_L$, a model $\mathcal{I}$ is locally expressive iff for all programs $P \in STAT_L$, and all formulas Q there is a formula W such W locally expresses the weakest precondition of P under Q relative to $\mathcal{I}$.

Regarding weak second-order logic, a positive expressiveness result has been shown in [BES95]. This result can be turned into a result on local expressiveness w.r.t. the standard first-order model of arithmetic.

As to the while-loop, fix the language as the language of arithmetic, and let $\mathcal{A}$ be the standard model of arithmetic. Following [BES95], we now define:

$$WP_{\text{while } b \text{ do } P}(\phi) := \exists f E_{b,P,\phi}(f, \bar{x})$$

where

$$\begin{aligned} E_{b,P,\phi} := \quad & (\neg b \wedge \phi)^{f(|f|-1)}_{\bar{x}} \wedge f(0) = \bar{x} \\ & \wedge \forall\, 0 \leq i < |f| - 1 \left(b \wedge WP_P(\bar{x} = f(i+1)) \right)^{f(i)}_{\bar{x}} \end{aligned}$$

and f ranges over finite sequences $\langle f(0), f(1), \cdots, f(n)\rangle$ of natural numbers, $|f| = n + 1$ is the length of the sequence, and $\bar{x}$ is the list of programming variables in the loop. The idea is that f codes the computation sequence of the loop execution by storing the values of the m variables occurring in the loop as m-tuples for each stage of the execution. In the last stage $f(n)$, $\neg b$ must hold as well as the postcondition ϕ. For each previous stage i we demand that b and $WP_P(\bar{x} = f(i+1))$ hold of the variable values at stage i, the latter meaning that execution of P terminates with the variable values of stage $i + 1$. Finally, the variable values at the first stage $f(0)$ must correspond to the initial precondition variable values.

We know from the proof of Gödel's First Incompleteness Theorems that notions such as "finite sequence", "length", etc. can be expressed through coding in purely number theoretic terms. Thus, the above sentence is indeed a first-order sentence in the language of arithmetic. The proof that this sentence indeed locally expresses the weakest precondition relative to the standard model of arithmetic is rather technical and can be adapted from the proof in [BES95]. We thus have

Theorem 23 *$L_{\omega\omega}$ is locally expressive for $WHILE_{arithmetic}$ relative to the standard model of arithmetic.*

The result can be extended to $GUARD_{arithmetic}$, by encoding computations as finite trees rather than sequences to account for the nondeterminism.

Fact 24 *$L_{\omega\omega}$ is locally expressive for $GUARD_{arithmetic}$ relative to the standard model of arithmetic.*

Gödel's incompleteness result shows of course that we cannot hope to obtain a complete axiomatization of arithmetic, so that while proving total correctness now has become a matter of proving formulas say in Peano Arithmetic, there will be total true correctness assertions we cannot prove, e.g. $true \rightarrow WP_{\mathsf{skip}}(G)$, where G is the true (in the standard model) but unprovable sentence from Gödel's First Incompleteness Theorem. **End of Excursion.**

4.1.2 Infinitary Logic

The preceding results show that if it is global expressiveness we are after, we need to turn to a logic more powerful than first-order logic. One option is to turn to $L_{\omega_1\omega}$, which allows countably infinite conjunctions and disjunctions.

As mentioned before, first-order logic is already globally expressive if we exclude while-statements, so we only need to treat this case. As stated in corollary 14, the predicate transformers of $WHILE$ and $GUARD$ are or-continuous, and hence we know that $wp_{\textsf{while } b \textsf{ do } P}(N) = \bigcup_{0 \leq i < \omega} F_N^i(\emptyset)$, with $F_N^0(\emptyset) = \emptyset$ and $F_N^{i+1}(X) = F_N(F_N^i(X))$ with $F_N(X) = ((\neg b)^{\mathcal{I}} \cap N) \cup (b^{\mathcal{I}} \cap wp_P(X))$. But this can be easily expressed by infinitary logic, namely by the formula

$$WP_{\textsf{while } b \textsf{ do } P}(\phi) \quad := \quad \bigvee_{0 \leq i < \omega} H^i \tag{3}$$

with $H^0 = false$ and $H^{i+1} = (\neg b \wedge \phi) \vee (b \wedge WP_P(H^i))$. Thus, we have

Theorem 25 *$L_{\omega_1 \omega}$ is globally expressive for $WHILE$ and $GUARD$.*

Since $L_{\omega_1 \omega}$ has a complete axiomatization (of course only by allowing proofs with countably infinite length), this result also provides us with a complete proof system for total correctness assertions $R[P]Q$: For formulas R, Q and program P, define $\Phi \models R[P]Q$ to mean that for all models $\mathcal{I}$, if $\mathcal{I} \models \Phi$ then $R^{\mathcal{I}} \subseteq wp_P(Q^{\mathcal{I}})$. Then we have

Theorem 26 *For any formulas R, Q of $L_{\omega_1 \omega}$ and any program $P \in GUARD$, we have for all countable sets Φ of $L_{\omega_1 \omega}$ sentences*

$$\Phi \models R[P]Q \Leftrightarrow \Phi \vdash R \rightarrow W$$

where W globally expresses the weakest precondition of P under Q.

Proof.

$$\begin{aligned} \Phi \models R[P]Q \quad & \Longleftrightarrow \forall \mathcal{I} : \mathcal{I} \models \Phi \Rightarrow R^{\mathcal{I}} \subseteq wp_P(Q^{\mathcal{I}}) \\ & \Longleftrightarrow \forall \mathcal{I} : \mathcal{I} \models \Phi \Rightarrow \mathcal{I} \models R \rightarrow W \\ & \Longleftrightarrow \Phi \models R \rightarrow W \\ & \Longleftrightarrow \Phi \vdash R \rightarrow W \end{aligned}$$

where W globally expresses the weakest precondition of P under Q. □

Questions & Conjectures 1 *It is remarkable that no difference between deterministic and bounded nondeterministic programs could be discovered, i.e. $L_{\omega\omega}$ is not expressive for either while $L_{\omega_1 \omega}$ is expressive for both. Considering that semantically in terms of their ST-models these two classes seem rather different in complexity, is there an interesting fragment of $L_{\omega_1 \omega}$ which is expressive for deterministic but not for bounded nondeterministic programs? More generally, which fragments of $L_{\omega_1 \omega}$ are sufficient to express the weakest preconditions of the different program classes?*

4.2 Unbounded nondeterministic programs

Turning to our two unbounded nondeterministic languages, $GUARD^{\times}$ and $GUARD^{*}$, we would like to define

$$\begin{array}{l} WP_{x:=y.Q}(\phi) := \exists y.Q \wedge \forall y.(Q \rightarrow \phi^y_x) \\ WP_{P^*}(\phi) := \bigwedge_{0 \leq i < \omega} WP_{P^i}(\phi) \end{array}$$

where $P^0 := skip$ and $P^{i+1} := P; P^i$. Both of these are formulas of $L_{\omega_1\omega}$, and so one might first think that the addition of these constructs will be conservative in the sense that global expressiveness of $L_{\omega_1\omega}$ is preserved. However, this is not the case. As we said earlier in section 3.4, the two wp-transformers need not be or-continuous. This lack of or-continuity means that adding them to our language will prevent us from applying fact 13 which guarantees reaching the least fixpoint of the while-loop by ω; thus, our $L_{\omega_1\omega}$ formula (3) will no longer globally express the weakest precondition of the while loop, since it only contains ω disjuncts. The following program P is a concrete example of this expressiveness failure:

$$\begin{array}{l} \mathsf{while}\ x \neq 0\ \mathsf{do}\ \mathsf{if}\ x > 0\ \mathsf{then}\ x := x - 1 \\ \qquad\qquad\qquad\qquad\quad \mathsf{else}\ x := y.y > 0 \end{array}$$

As for $WP_P(true)$, we calculate using (3) (in the standard model):

$$\begin{array}{ll} H^0 & \equiv false \\ H^1 & \equiv x = 0 \\ H^2 & \equiv (x = 0 \wedge true) \vee (x \neq 0 \wedge (x > 0 \rightarrow WP_{x:=x-1}(x = 0)) \\ & \quad \wedge (x \not> 0 \rightarrow WP_{x:=y.y>0}(x = 0))) \\ & \equiv x = 0 \vee (x \neq 0 \wedge (x > 0 \rightarrow x = 1) \wedge (x \not> 0 \rightarrow false) \\ & \equiv x = 0 \vee x = 1 \\ \vdots & \\ H^n & \equiv x = 0 \vee \cdots \vee x = n - 1 \\ \vdots & \end{array} \tag{4}$$

and hence $WP_P(true) \equiv x \geq 0$. However, we would expect program P to always terminate, even for negative values of x. Semantically, where we iterate to the least fixpoint even if it lies beyond ω, we can see that $wp_P(true) = true$ with the fixpoint being reached at ω_1: Let P_1 be the program $\mathsf{if}\ x > 0\ \mathsf{then}\ x := x - 1$ $\mathsf{else}\ x := y.y > 0$. Recall that the operator used in the fixpoint approximation (see section 3.1) with respect to our program P relative to postcondition $true$ was

$$wp_P(X) = (x = 0^{\mathcal{I}} \cap true) \cup (x \neq 0^{\mathcal{I}} \cap wp_{P_1}(X))$$

Then we have the following semantic calculation of the fixpoint in the standard model $\mathcal{I}$:

$$\begin{array}{ll}
wp_P^0(true) & = \emptyset \\
wp_P^1(true) & = (x = 0^{\mathcal{I}} \cap true) \cup (x \neq 0^{\mathcal{I}} \cap wp_{P_1}(\emptyset)) \\
 & = x = 0^{\mathcal{I}} \\
wp_P^2(true) & = (x = 0^{\mathcal{I}} \cap true) \cup (x \neq 0^{\mathcal{I}} \cap wp_{P_1}(x = 0^{\mathcal{I}})) \\
 & = x = 0^{\mathcal{I}} \cup (x \neq 0^{\mathcal{I}} \cap ((x > 0^{\mathcal{I}} \cap wp_{x:=x-1}(x = 0^{\mathcal{I}})) \\
 & \quad \cup (x \not> 0^{\mathcal{I}} \cap wp_{x:=y.y>0}(x = 0^{\mathcal{I}})))) \\
 & = x = 0^{\mathcal{I}} \cup x = 1^{\mathcal{I}} \\
\vdots & \\
wp_P^n(true) & = x = 0^{\mathcal{I}} \cup \cdots \cup x = n - 1^{\mathcal{I}} \\
\vdots & \\
wp_P^{\omega}(true) & = x \geq 0^{\mathcal{I}} \\
wp_P^{\omega_1}(true) & = wp_P(wp_P^{\omega}(true)) \\
 & = x = 0^{\mathcal{I}} \cup (x \neq 0^{\mathcal{I}} \cap ((x > 0^{\mathcal{I}} \cap wp_{x:=x-1}(x \geq 0^{\mathcal{I}})) \\
 & \quad \cup (x \not> 0^{\mathcal{I}} \cap wp_{x:=y.y>0}(x \geq 0^{\mathcal{I}})))) \\
 & = true
\end{array}$$

This example of an unbounded nondeterministic program is often used to distinguish **weak termination** from **strong termination**. A loop is said to terminate strongly if for every state s there is a positive integer n_s such that the loop is guaranteed to terminate in less than n_s iterations. (3) formalizes strong termination, and provided we have only bounded nondeterminism, loop termination is indeed always strong, so no problems arise. With an unbounded nondeterministic program such as $x := y.x > 0$ however, the above example illustrates that there are cases where loops terminate only weakly, so that (3) is not an appropriate formalization of termination anymore.

The fact that (3) does not globally express the weakest precondition of the while-loop cannot be remedied by choosing another formula, as the following theorem asserts.

Theorem 27 *$L_{\omega_1\omega}$ is not globally expressive for $GUARD^{\times}$.*

Proof. Suppose the contrary, and let the language consist of only the binary predicate symbol A. Let P be the following program

$$while\ \exists y.A(x,y)\ do\ x := y.A(x,y)$$

Let $W(x)$ be the formula which globally expresses the weakest precondition of P under *true*. Then for an arbitrary structure, $W(x)$ holds precisely of those states s where there is no infinite A sequence starting in $s(x)$. Thus, $\forall x W(x)$ expresses that relation A is well-founded. But since well-foundedness cannot be expressed in $L_{\omega_1\omega}$ (see e.g. [Kei71]), we have a contradiction. Hence, such a W cannot exist. □

Back mentions that the use of quantification in the test of the while-loop is not essential in the proof. For we could still make the same argument with the program

$$\begin{array}{l} set\ b; \\ \text{while } b \text{ do } x := y.A(x,y);\ set\ b \end{array}$$

where *set b* is the statement

$$b := c.\ \big(\ (\exists y.A(x,y) \wedge c = true) \vee (\neg\exists y.A(x,y) \wedge c = false)\ \big)$$

Since well-foundedness cannot be expressed in $L_{\alpha\omega}$ for any infinite cardinal α, the same proof carries over to these more expressive logics as well. To obtain global expressiveness in the presence of nondeterministic assignment, Back shows that we need to go to a logic which allows infinite sequences of quantifiers as well:

Fact 28 *$L_{\omega_1\omega_1}$ is globally expressive for $GUARD^{\times}$.*

5 Formalizing ST-Semantics: Weakest Preconditions in Dynamic Logic

While in the previous section the attempt was to directly express weakest preconditions in some logic, we now return to the more standard approach of formalizing weakest preconditions using state-transformer semantics. In Dynamic Logic, the state transformers which model programs are simply relations on states, i.e. subsets of $S \times S$ or equivalently state transformers of NST. For a given program p, one can define correctness statements in terms of the $[p]$ and $\langle p \rangle$ modalities. While a more adequate comparison between the previous formalization in infinitary logic and a formalization in Dynamic Logic would require us to consider full QDL, we will still restrict ourselves to PDL. It is of course simpler than QDL, but it still suffices to provide some interesting points of comparison.

For better comparison with the previous results, this section will use the same notion of expressiveness which was used in the previous section. The programming constructs used earlier have their well-known analogues in Dynamic Logic. To make use of this analogy, we shall abbreviate certain PDL formulas in the language of the earlier constructs:

$$\begin{array}{rl} \mathsf{skip} & := \top? \\ \mathsf{abort} & := \bot? \\ \mathsf{if}\ b\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2 & := b?; p_1 \cup \neg b?; p_2 \\ \mathsf{if}\ b_1 \rightarrow p_1 \| \cdots \| b_n \rightarrow p_n\ \mathsf{fi} & := b_1?; p_1 \cup \cdots \cup b_n?; p_n \\ \mathsf{while}\ b\ \mathsf{do}\ p & := (b?; p)^*; \neg b? \end{array}$$

Note that again abort is definable as while $true$ do skip. In what follows, we will allow for the possibility of non-propositional test operators $b?$, i.e. occurrences of $\langle\rangle$ or $[]$ are allowed in b.

This section starts by formalizing weakest preconditions in standard PDL for deterministic programs. Next, in order to handle (bounded) nondeterminism, two extensions of PDL will be discussed and proved equivalent, namely PDL^+ and $PDL + loop$. In the last part, it will be shown that these extensions are sufficient for unbounded nondeterminism as well. $PDL + loop/repeat$ grew out of PDL^+ which was introduced in [HP78]; another primary source on this issue is [Har79]. The equivalence between the two systems is intuitively clear and mentioned in [Har79], though not actually proved.

5.1 Deterministic PDL-programs

We will define the class of $WHILE$-**programs** inductively like the class of PDL-programs in general, except that we restrict the clause for program iteration to programs of the form while b do p, and that we restrict the clause for program disjunction to if b then p_1 else p_2. The details of this definition should be clear. Note that this class does not exactly correspond to the $WHILE$ programs defined earlier, since we allow e.g. non-propositional tests. All we want is to restrict the class of PDL-programs just enough to obtain a deterministic fragment.

Definition 18 **($PDL(WHILE)$)** *To obtain $PDL(WHILE)$, we restrict PDL by (1) only allowing semantically deterministic atomic programs (where every input state is mapped to at most one output state) and by (2) restricting the class of programs to $WHILE$-programs.*

(In the terminology of [HKT95] where several deterministic restrictions of PDL are mentioned, $PDL(WHILE)$ corresponds to Structured Deterministic PDL, or $SDPDL$ for short.)

It can be shown by induction that all programs of $PDL(WHILE)$ are semantically deterministic, i.e. the state transformers associated with programs from $PDL(WHILE)$ can all be modeled in DST. Thus, the appropriate notion of weakest precondition can be taken from section 2: Taking any deterministic program p with its deterministic state transformer $f_p \in DST$, we defined the weakest precondition of p under the set of states Q as $wp_p(Q) := f_p^{-1}[Q]$. Now we can appeal to the definition of expressiveness, and prove that

Theorem 29 *$PDL(WHILE)$ is globally expressive for $WHILE$.*

Proof. So let p be an arbitrary $WHILE$ program, and q an arbitrary $PDL(WHILE)$ formula. Then for any model $\mathcal{I}$, $(\langle p\rangle q)^{\mathcal{I}} = wp_p(q^{\mathcal{I}})$. □

In contrast to the case of infinitary logic, the proof of this fact was trivial since PDL has an operator in the language which is well suited for total correctness assertions for all deterministic programs.

Regarding axiomatization, it is well known that PDL can be completely axiomatized (see e.g. [Har84]). Deterministic PDL (i.e. $DPDL$) is PDL restricted to structures where all atomic programs are partial functions. If we add the axiom $\langle p\rangle q \rightarrow [p]q$ for all atomic programs p, [BAHP82] shows that the resulting axiom system is sound and complete for $DPDL$. As a consequence, the resulting system is also sound and complete for $PDL(WHILE)$, provided that we restrict the theorems to those formulas which only contain $WHILE$-programs.

5.2 Bounded nondeterministic PDL-programs

PDL's state transformers are general relations on states, i.e. functions in NST; hence they can also represent nondeterministic programs. However, when reasoning about total correctness, this model is not sufficient, as has been discussed in section 2. Consider as an example of the problem mentioned there the state transformers of

$$\textsf{if } true \rightarrow \textsf{while } true \textsf{ do skip } \| true \rightarrow \textsf{skip fi}$$

or equivalently $\textsf{skip} \cup \textsf{while } true \textsf{ do skip}$. This state transformer turns out to be the same as the one associated with $\textsf{skip}$. Thus, only for partial correctness are PDL-state transformers an appropriate model of programs. For modeling total correctness, we therefore need to enrich PDL in some way.

5.2.1 PDL^+

In line with the earlier discussion, we can upgrade the usual PDL state transformers to $NST_\perp$ state transformers. This approach has been taken in [HP78], where Harel and Pratt introduce a logic called DL^+.

Definition 19 **(PDL^+)** *The syntax of PDL^+ augments the syntax of PDL by also allowing formulas of the form $[\alpha]^+p$ and $\langle\alpha\rangle^+p$, where α is a program and p a formula of PDL^+.*

A PDL^+ structure is a pair $\mathcal{I} = (S^{\mathcal{I}}_\perp, m^{\mathcal{I}})$, where $S^{\mathcal{I}}_\perp = S^{\mathcal{I}} \cup \{\perp\}$ is the set of states or the universe. $\perp \notin S^{\mathcal{I}}$ is a special divergence state where no proposition holds. For all atomic programs a,

$$m^{\mathcal{I}}(a) = R^{\mathcal{I}}_a = \subseteq S^{\mathcal{I}} \times S^{\mathcal{I}}$$

and for all atomic propositions p,

$$m^{\mathcal{I}}(p) = p^{\mathcal{I}} \subseteq S^{\mathcal{I}}$$

In general, for any program p, we will freely change between relational and functional perspective, i.e. we will identify $R^{\mathcal{I}}_p$ with $f_p \in NST_\perp$.

Validity in PDL^+ is defined without taking $\perp$ into account (so that e.g. $\models p \rightarrow p$ still holds), i.e. $\mathcal{I} \models \varphi$ iff for all $s \in S^{\mathcal{I}}$, $\mathcal{I}, s \models \varphi$.

As to the semantics of non-atomic programs, only a few changes need to be made: First, we define $p_1; p_2^{\mathcal{I}} := p_1^{\mathcal{I}} \circ (p_2^{\mathcal{I}} \cup \{(\perp, \perp)\})$. Second, we redefine

$$R^{\mathcal{I}}_{\alpha^*} = \bigcup_{0 \leq n < \omega} (R^{\mathcal{I}}_\alpha)^n \cup \{(s_0, \perp) | \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R^{\mathcal{I}}_\alpha\}$$

with $(R_\alpha^{\mathcal{I}})^0 = \mathsf{skip}$ *and* $(R_\alpha^{\mathcal{I}})^{n+1} = R_\alpha^{\mathcal{I}} \circ ((R_\alpha^{\mathcal{I}})^n \cup \{\bot, \bot\})$. *The semantics of the formulas is still as usual, except for the modalities:*

$$\mathcal{I}, s \models [\alpha]p \;\text{ iff }\; \forall t : R_\alpha^{\mathcal{I}}(s,t) \Rightarrow \mathcal{I}, t \not\models \neg p$$

$$\mathcal{I}, s \models [\alpha]^+ p \;\text{ iff }\; \forall t : R_\alpha^{\mathcal{I}}(s,t) \Rightarrow \mathcal{I}, t \models p$$

The operators $\langle\alpha\rangle$ *and* $\langle\alpha\rangle^+$ *are the duals of these respectively, i.e.*

$$\mathcal{I}, s \models \langle\alpha\rangle p \;\text{ iff }\; \exists t : R_\alpha^{\mathcal{I}}(s,t) \wedge \mathcal{I}, t \models p$$

$$\mathcal{I}, s \models \langle\alpha\rangle^+ p \;\text{ iff }\; \exists t : R_\alpha^{\mathcal{I}}(s,t) \wedge \mathcal{I}, t \not\models \neg p$$

Note. The differences to the logic proposed by Harel & Pratt are as follows: First, they let the state transformers be subsets of $S_\bot^{\mathcal{I}} \times S_\bot^{\mathcal{I}}$ and let $(\bot, \bot) \in R_a$ for every basic program a. This has some small technical advantages, since it allows to leave the definition of program composition unchanged. Nonetheless, it is somewhat unintuitive considering the interpretation we give to $\bot$, and it does not link up as nicely with our previous work. Second, they allow atomic programs to fail, i.e. $(s, \bot) \in R_a^{\mathcal{I}}$ is possible. **End of Note.**

Some further comments regarding abort need to be made. In PDL^+, we cannot define abort as $\bot$ anymore since then we would run into the same problem we encountered with infinite loops: The programs skip and abort $\cup$ skip would have the same state transformer. Hence, unless we want to distinguish abortion from divergence (a distinction we do not make in this paper), we would want an atomic program a which maps any state to $\bot$. We could add such a program to PDL^+, but we can of course equally well decide to just define abort as while *true* do skip. Our stipulation that atomic programs never fail is driven by the connection we want to establish between PDL^+ and the well-known $PDL{+}loop$, a logic which is only concerned with divergence. While this stipulation may not seem natural if we want to model failure other than looping, it does make sense to stipulate that atomic programs never diverge. Hence, PDL^+ can be viewed in the first place as a divergence logic rather than a failure logic, i.e. its purpose is to allow one to reason about divergence more than about failure in general.

Definition 20 ($PDL^+(GUARD)$) *To obtain* $PDL^+(GUARD)$, *we restrict* PDL^+ *by only allowing bounded nondeterministic atomic programs.*

It can be shown by induction that all programs of $PDL(GUARD)$ are bounded nondeterministic. Note that in PDL^+, p^* does not produce unbounded nondeterminism provided that p is bounded nondeterministic. A proof of this essentially comes down to König's Lemma, i.e. the fact that every infinite but finitely branching tree must have an infinite branch.

Again, the appropriate notion of weakest precondition can be taken from section 2: Taking any program p with its state transformer $f_p \in NST_\bot$, we defined the weakest precondition of p under the set of states $Q \subseteq S$ as $wp_p(Q) := f_p^{-1}[\mathcal{P}(Q)] = \{s \in S^{\mathcal{I}} | \forall t : R_p^{\mathcal{I}}(s,t) \to t \in Q\}$.

Theorem 30 *$PDL^+(GUARD)$ is globally expressive for the class of all PDL^+ programs.*

Proof. Take any program α and any postcondition $q \in PDL^+(GUARD)$. Then for any model $\mathcal{I}$ with universe $S_\perp^{\mathcal{I}}$, we have

$$[\alpha]^+ q^{\mathcal{I}} = \{s \in S^{\mathcal{I}} | \forall t : R_\alpha^{\mathcal{I}}(s,t) \Rightarrow \mathcal{I}, t \models q\} = wp_\alpha(q^{\mathcal{I}})$$

□

Note that the expressiveness of PDL^+ is not exhausted by the theorem. For we cannot only talk about the fact that there are failing/diverging α-computations (something which the weakest precondition approach allows us to do), we can also express by the formula $[\alpha]false \wedge \langle\alpha\rangle^+ false$ that there is exactly one α-computation and that this computation fails.

As to axiomatization, an axiomatization for Harel & Pratt's first-order DL^+ is discussed in [HP78] for which arithmetical completeness is demonstrated. For PDL^+ we will establish an equivalence with $PDL+loop$ for which the existence of a complete axiomatization is stated in [HS82].

5.2.2 *PDL* plus *loop* or *repeat*

Another extension of PDL which has been proposed to handle total correctness is the addition of one of two extra predicates to PDL, namely $loop(\alpha)$ or $repeat(\alpha)$, resulting in $LPDL$ and $RPDL$ respectively. The semantics of these expression is as follows:

$$repeat(\alpha)^{\mathcal{I}} = \{s_0 \in S | \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_\alpha^{\mathcal{I}}\}$$

$$\begin{aligned}
loop(a)^{\mathcal{I}} &= \emptyset \text{ for atomic programs } a \\
loop(p?)^{\mathcal{I}} &= \emptyset \\
loop(\alpha;\beta)^{\mathcal{I}} &= loop(\alpha)^{\mathcal{I}} \cup \langle\alpha\rangle loop(\beta)^{\mathcal{I}} \\
loop(\alpha \cup \beta)^{\mathcal{I}} &= loop(\alpha)^{\mathcal{I}} \cup loop(\beta)^{\mathcal{I}} \\
loop(\alpha^*)^{\mathcal{I}} &= \langle\alpha^*\rangle loop(\alpha)^{\mathcal{I}} \cup \{s_0 \in S | \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_\alpha^{\mathcal{I}}\}
\end{aligned}$$

Clearly, the clause for $loop(\alpha^*)$ shows that $loop(\alpha^*) \equiv \langle\alpha^*\rangle loop(\alpha) \vee repeat(\alpha)$, and hence given $repeat$, we can define $loop(\alpha)$ for every α. On the other hand, it is shown in [HS82] that $RPDL$ is strictly more expressive than $LPDL$. We will adopt $LPDL$, since it is weaker and yet sufficient for total correctness reasoning.

5.2.3 Equivalence of *LPDL* and *PDL*$^+$

The following lemma is stated without proof in [HP78]. Since it is as easy to verify in PDL^+ as in Harel & Pratt's DL^+, we will similarly omit the proof.

Lemma 31 *The following are valid in PDL^+:*

$$\begin{array}{lrl}
(a) & \langle\alpha\rangle^+ p & \equiv \langle\alpha\rangle p \vee \langle\alpha\rangle^+ false \\
(b) & \langle\alpha;\beta\rangle^+ p & \equiv \langle\alpha\rangle^+ false \vee \langle\alpha\rangle\langle\beta\rangle^+ false \vee \langle\alpha\rangle\langle\beta\rangle p \\
(c) & \langle\alpha \cup \beta\rangle^+ p & \equiv \langle\alpha\rangle^+ p \vee \langle\beta\rangle^+ p
\end{array}$$

Note that by the duality $[\alpha]^+p \equiv \neg\langle\alpha\rangle^+\neg p$ and by part (a) of the above lemma, we can from now on restrict the syntax of PDL^+ by excluding formulas which contain $[\alpha]$ or $[\alpha]^+$, and by restricting the use of $\langle\alpha\rangle^+$ to $\langle\alpha\rangle^+ false$. We can now define a translation function τ from PDL^+ to $LPDL$:

Definition 21 (Translation Function τ from PDL^+ into $LPDL$) *The translation function τ maps PDL^+ models to $LPDL$ models and PDL^+ formulas to $LPDL$ formulas. $\tau(\mathcal{I}) := \mathcal{I}'$, where $\mathcal{I}'$ is the same as $\mathcal{I}$ except that $S_{\mathcal{I}'} = S_{\mathcal{I}} - \{\perp\}$. On formulas, $\tau(\phi) = \phi'$, where ϕ' is the same as ϕ, except that occurrences of $\langle\alpha\rangle^+ false$ have been replaced by $loop(\alpha)$. (By the previous lemma, it is easy to see that this translation maps $[\alpha]^+\varphi$ to $[\alpha]\varphi \wedge \neg loop(\alpha)$.)*

Note that τ is a total bijection for formulas as well as for models. Let τ^{-1} denote the inverse of τ.

Lemma 32 *For all programs α and PDL^+ models $\mathcal{I}$,*

$$R_\alpha^{\mathcal{I}} \cap \{(s,t) \in S^{\mathcal{I}} \times S^{\mathcal{I}} | t \neq \perp\} = R_\alpha^{\tau(\mathcal{I})}$$

and for all $s \in S^{\mathcal{I}}$,

$$(s,\perp) \in \bigcup_{n\geq 0} (R_\alpha^{\mathcal{I}})^n \quad \textit{iff} \ \exists t : (s,t) \in \bigcup_{n\geq 0} (R_\alpha^{\mathcal{I}})^n \wedge R_\alpha^{\mathcal{I}}(t,\perp)$$

Proof. The proof of the first part is by induction on α and will be skipped. For the second part, the implication from right to left is immediate. From left to right, we use the fact that $(s,\perp) \notin (R_\alpha^{\mathcal{I}})^0$ always holds for any s. □

Theorem 33 *For all PDL^+ models $\mathcal{I}$ and all PDL^+ formulas ϕ,*

$$\phi^{\mathcal{I}} = \tau(\phi)^{\tau(\mathcal{I})}$$

Proof. The proof is by induction on ϕ. Since for all atomic p, $\perp \notin p^{\mathcal{I}}$, the base case for atomic propositions and the inductive step for the boolean connectives is immediate. As for $\phi = \langle\alpha\rangle\gamma$, we have by induction hypothesis and by the previous lemma,

$$\begin{aligned} (\langle\alpha\rangle\gamma)^{\mathcal{I}} &= \{s \in S^{\mathcal{I}} | \exists t : R_\alpha^{\mathcal{I}}(s,t) \wedge \gamma^{\mathcal{I}}(t)\} \\ &= \{s \in S^{\mathcal{I}} | \exists t : R_\alpha^{\tau(\mathcal{I})}(s,t) \wedge t \in \tau(\gamma)^{\tau(\mathcal{I})}\} \\ &= (\langle\alpha\rangle\tau(\gamma))^{\tau(\mathcal{I})} = \tau(\langle\alpha\rangle\gamma)^{\tau(\mathcal{I})} \end{aligned}$$

The main part of the proof is showing that $(\langle\alpha\rangle^+ false)^{\mathcal{I}} = loop(\alpha)^{\tau(\mathcal{I})}$. This is proved by induction on α. The base case for atomic α follows immediately from the definitions and the fact that atomic programs do not diverge. For the inductive step, we have by the equivalences (b) and (c) from the earlier lemma 31 and by lemma 32,

$$
\begin{aligned}
(\langle\alpha\cup\beta\rangle^{+} false)^{\mathcal{I}} &= (\langle\alpha\rangle^{+} false \vee \langle\beta\rangle^{+} false)^{\mathcal{I}} \\
&= loop(\alpha)^{\tau(\mathcal{I})} \cup loop(\beta)^{\tau(\mathcal{I})} \\
&= loop(\alpha\cup\beta)^{\tau(\mathcal{I})}
\end{aligned}
$$

$$
\begin{aligned}
(\langle\alpha;\beta\rangle^{+} false)^{\mathcal{I}} &= (\langle\alpha\rangle^{+} false \vee \langle\alpha\rangle\langle\beta\rangle^{+} false \vee \langle\alpha\rangle\langle\beta\rangle false)^{\mathcal{I}} \\
&= (loop(\alpha) \vee \langle\alpha\rangle loop(\beta) \vee \langle\alpha\rangle\langle\beta\rangle false)^{\tau(\mathcal{I})} \\
&= loop(\alpha;\beta)^{\tau(\mathcal{I})}
\end{aligned}
$$

$$
\begin{aligned}
(\langle p?\rangle^{+} false)^{\mathcal{I}} &= \{s \in S^{\mathcal{I}} | p^{\mathcal{I}}(s) \wedge s \not\models true\} \\
&= \emptyset \;=\; loop(p?)^{\tau(\mathcal{I})}
\end{aligned}
$$

$$
\begin{aligned}
(\langle\alpha^{*}\rangle^{+} false)^{\mathcal{I}} &= \{s \in S^{\mathcal{I}} | (s,\perp) \in \textstyle\bigcup_{n\geq 0}(R_{\alpha}^{\mathcal{I}})^{n}\} \\
&\quad \cup\{s_0 \in S^{\mathcal{I}} | \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_{\alpha}^{\mathcal{I}}\} \\
&= \{s \in S^{\mathcal{I}} | \exists t : (s,t) \in \textstyle\bigcup_{n\geq 0}(R_{\alpha}^{\mathcal{I}})^{n} \wedge R_{\alpha}^{\mathcal{I}}(t,\perp)\} \\
&\quad \cup\{s_0 \in S^{\mathcal{I}} | \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_{\alpha}^{\mathcal{I}}\} \\
&= (\langle\alpha^{*}\rangle\langle\alpha\rangle^{+} false)^{\mathcal{I}} \\
&\quad \cup\{s_0 \in S^{\mathcal{I}} | \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_{\alpha}^{\mathcal{I}}\} \\
&= \langle\alpha^{*}\rangle loop(\alpha)^{\tau(\mathcal{I})} \\
&\quad \cup\{s_0 \in S^{\mathcal{I}} | \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_{\alpha}^{\tau(\mathcal{I})}\} \\
&= loop(\alpha^{*})^{\tau(\mathcal{I})}
\end{aligned}
$$

□

Corollary 34 *For all PDL^{+} formulas ϕ, ϕ is valid in PDL^{+} if and only if $\tau(\phi)$ is valid in $LPDL$.*

5.3 Unbounded nondeterministic PDL-programs

In the case of infinitary logic, we encountered problems as soon as we extended our language with unbounded nondeterministic programming constructs such as nondeterministic assignment: Global expressiveness was not preserved under these extensions.

In the case of PDL^{+} and $LPDL$, the case lies somewhat different. The problems with infinitary logic arose because we used a fixpoint construction which only approximated the fixpoint up to ω, a construction which turned out not to be sufficient for unbounded nondeterministic programs. In PDL^{+} and $LPDL$ however, the definition of α^{*} does not use fixpoints but checks for infinite α-sequences directly. Section 7.2 will show that if we use fixpoints to define the semantics of $[\alpha^{*}]^{+}q$ in PDL^{+}, we get results analogous to infinitary logic.

Like in our discussion of infinitary logic, we consider two extensions, namely allowing unbounded nondeterministic atomic programs (in which case we arrive at full PDL^{+}) and adding a $^{\bullet}$-operator for reflexive transitive closure.

Regarding the first extension to full PDL^+, the result that $PDL^+(GUARD)$ is globally expressive for all PDL^+ programs already covers the possibility of unbounded nondeterministic atomic programs:

Theorem 35 *PDL^+ is globally expressive for the class of all PDL^+ programs.*

Proof. Same as for $PDL^+(GUARD)$. □

Regarding the second extension, note that while we have the Kleene star in the language of PDL^+, p^* is actually not the reflexive transitive closure of p anymore, like it used to be in PDL. For we modified the definition of p^* to include a transition to $\perp$ whenever there is an infinite p-sequence. Hence, we could say that the meaning of α^* has been modified from $\bigcup_{i<\omega} \alpha^i$ to $\bigcup_{i\leq\omega} \alpha^i$. As a result, we e.g. obtain

$$wp_{\mathsf{skip}^*}(true) = wp_{\mathsf{while\ true\ do\ skip}}(true) = false$$

The reason for this result is that skip^* can loop in every state, i.e. contains $(s, \perp)$ for any state s. This shows that skip^* indeed gets the interpretation "do skip an arbitrary finite number of times or infinitely often". This intensional and intentional abuse of the Kleene star does not cause any problem, but if we now also want to have the real reflexive transitive closure, i.e. $\bigcup_{i<\omega} \alpha^i$, we need to introduce a new operator (say $^\bullet$) so that $\alpha^\bullet$ will mean "do α an arbitrary (finite) number of times". For this new operator, we would thus define $\alpha^\bullet$ as $\bigcup_{i<\omega} \alpha^i$ In $LPDL$, the only semantic difference between $\alpha^\bullet$ and α^* will show up in the definition of $loop$.

$$\begin{array}{ll} loop(\alpha^\bullet)^{\mathcal{I}} & = \langle\alpha^*\rangle loop(\alpha)^{\mathcal{I}} \\ loop(\alpha^*)^{\mathcal{I}} & = \langle\alpha^*\rangle loop(\alpha)^{\mathcal{I}} \cup \{s_0 \in S | \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_\alpha^{\mathcal{I}}\} \end{array}$$

It should be easy to see that the new $^\bullet$ operator does indeed make unbounded nondeterminism possible. Take for example a deterministic atomic program such as $x := x + 1$ where states assign integer values to x. Then $x := x + 1^*$ will still be bounded nondeterministic since $x := x + 1$ can be applied infinitely often, while $x := x + 1^\bullet$ will be unbounded nondeterministic.

6 Fixpoint Semantics for Iteration in PDL^+

In section 3.2, two fixpoint definitions for the while-loop have been discussed, one using a unary fixpoint operator on predicates, the other using a binary fixpoint operator on state-transformers. In PDL^+, the semantic definition of iteration does not use any fixpoint construction, but rather checks for infinite chains directly:

$$R_{\alpha^*}^{\mathcal{I}} = \bigcup_{0\leq n<\omega} (R_\alpha^{\mathcal{I}})^n \cup \{(s_0, \perp) | \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_\alpha^{\mathcal{I}}\} \qquad (5)$$

The lack of a fixpoint definition turns out to be the reason why in the previous section no complexity difference between $PDL^+(GUARD)$ and $PDL^+(GUARD^\times)$ could be discovered in terms of global expressiveness. We can however restate the semantic definitions of iteration in terms of fixpoints. Doing so will allow for a more fine-grained analysis of the complexity of different program classes.

We will start our investigation with partial correctness in PDL^+, i.e. with providing a direct fixpoint definition of the predicate transformer of $[p^*]q$, also demonstrating that we can guarantee ω as an upper bound. Next, we provide a similar fixpoint definition for the predicate transformer of $[p^*]^+q$, and we show that an upper bound can only be guaranteed under certain conditions. Besides showing the complexity difference between partial and total correctness reasoning, these fixpoint definitions will then allow us to get more precise results regarding the complexity difference between bounded and unbounded nondeterminism in section 7.

6.1 Partial correctness: fixpoint semantics for $[p^*]q$

Using definition (5), recall the standard definition of $[p^*]q$ in PDL^+:

$$\begin{aligned}([\alpha^*]q)^{\mathcal{I}} &= \{s \in S^{\mathcal{I}} | \forall t \in S^{\mathcal{I}}_{\perp} : (s,t) \in R^{\mathcal{I}}_{\alpha^*} \Rightarrow t \notin \neg q^{\mathcal{I}}\} \\ &= \{s \in S^{\mathcal{I}} | \forall t \in S^{\mathcal{I}}_{\perp} : (s,t) \in R^{\mathcal{I}}_{\alpha^*} \Rightarrow t \in (q^{\mathcal{I}} \cup \{\perp\})\} \\ &= \{s_0 \in S^{\mathcal{I}} | \forall t \in S^{\mathcal{I}}_{\perp} : ((s_0,t) \in \textstyle\bigcup_{0 \leq n < \omega} (R^{\mathcal{I}}_{\alpha})^n \rightarrow (q^{\mathcal{I}}(t) \vee t = \perp))\}\end{aligned} \tag{6}$$

Defining the semantics of $[p^*]q$ directly by means of a predicate transformer, we define

$$\begin{aligned}([\alpha^*]q)^{\mathcal{I}} &= \nu X.\, \sigma^{\mathcal{I}}_{\alpha,q^{\mathcal{I}}}(X) \\ &\quad \text{where } \sigma^{\mathcal{I}}_{\alpha,Y}(X) = X \cap \{s \in S^{\mathcal{I}} | Y(s) \wedge \forall t : R^{\mathcal{I}}_{\alpha}(s,t) \rightarrow (X(t) \vee t = \perp)\}\end{aligned} \tag{7}$$

where $\nu X.F(X)$ denotes the greatest fixpoint of operator F. Since σ is monotone, such a greatest fixpoint exists. We can show that the two definitions of $[\alpha^*]q$ coincide:

Theorem 36 *The two semantic definitions of $[\alpha^*]q$ coincide, i.e. (6) = (7).*

Proof. We need to show that

$$\nu X.\, \sigma^{\mathcal{I}}_{\alpha,Y}(X) = \{s_0 \in S^{\mathcal{I}} | \forall t \in S^{\mathcal{I}}_{\perp} : ((s_0,t) \in \bigcup_{0 \leq n < \omega} (R^{\mathcal{I}}_{\alpha})^n \rightarrow (Y(t) \vee t = \perp))\}$$

Let Φ be the set on the right-hand side of the equation. First, we show that Φ is a fixpoint of σ, i.e. $\sigma^{\mathcal{I}}_{\alpha,Y}(\Phi) = \Phi$. Clearly, we have $\sigma^{\mathcal{I}}_{\alpha,Y}(\Phi) \subseteq \Phi$. For the other inclusion, suppose that $s_0 \in \Phi$. Since $(s_0,s_0) \in (R^{\mathcal{I}}_{\alpha})^0$, we know that $Y(s_0)$. Now take any t such that $R^{\mathcal{I}}_{\alpha}(s_0,t)$ holds, and suppose that $t \neq \perp$. We must show that $t \in \Phi$. But since any α^n-successor of t is a α^{n+1}-successor of s_0, we know that $t \in \Phi$ since $s_0 \in \Phi$. Hence, we showed that $s_0 \in \sigma^{\mathcal{I}}_{\alpha,Y}(\Phi)$.

Second, we show that for any $Z \subseteq S^{\mathcal{I}}$ with $\sigma^{\mathcal{I}}_{\alpha,Y}(Z) = Z$, $\Phi \supseteq Z$ must hold. We shall demonstrate this by contraposition: We show the following by induction on n:

$$\forall s \in S : \big(\exists t : (s,t) \in (R^{\mathcal{I}}_{\alpha})^n \wedge t \notin Y \cup \{\bot\} \big) \rightarrow s \notin \sigma^{\mathcal{I}}_{\alpha,Y}(Z)$$

For $n = 0$ and any s, we know that if the antecedent is true, $s = t$ and hence $s \notin Y \cup \{\bot\}$, so clearly $s \notin \sigma^{\mathcal{I}}_{\alpha,Y}(Z)$. For $n = m + 1$, suppose $(s,t) \in (R^{\mathcal{I}}_{\alpha})^{m+1}$, then $(s,t') \in R^{\mathcal{I}}_{\alpha}$ and $(t',t) \in (R^{\mathcal{I}}_{\alpha})^m$ with $t \notin Y \cup \{\bot\}$. By induction hypothesis, $t' \notin \sigma^{\mathcal{I}}_{\alpha,Y}(Z) = Z$, so s cannot be in $\sigma^{\mathcal{I}}_{\alpha,Y}(Z)$. □

Regarding definition (7), we can say more about the complexity (i.e. upper bound) of the fixpoint:

Theorem 37 *The bound on the fixpoint in (7) is always ω, i.e.*

$$([\alpha^*]q)^{\mathcal{I}} = \bigcap_{0 \leq n < \omega} (\sigma^{\mathcal{I}}_{\alpha,q^{\mathcal{I}}})^n (S^{\mathcal{I}})$$

with σ as defined in (7).

Proof. Since it is clear that $\sigma(\bigcap_{0 \leq n < \omega} \sigma^n(S)) \subseteq \bigcap_{0 \leq n < \omega} \sigma^n(S)$, we only need to show the opposite inclusion. So suppose $x \in \bigcap_{0 \leq n < \omega} \sigma^n(S)$. Since x is also in $\sigma(S)$, $q^{\mathcal{I}}(x)$ holds. Furthermore, suppose $R^{\mathcal{I}}_{\alpha}(x,t)$, then for any n, since $x \in \sigma^{n+1}(S)$, $t \in \sigma^n(S) \cup \{\bot\}$, and hence $t \in \bigcap_{0 \leq n < \omega} \sigma^n(S) \cup \{\bot\}$. Thus, $x \in \sigma(\bigcap_{0 \leq n < \omega} \sigma^n(S))$.

To see that it is the largest fixpoint, suppose $\sigma(Z) = Z$. By induction, we can see that for all n, $\sigma^n(S) \supseteq Z$, and hence $\bigcap_{0 \leq n < \omega} \sigma^n(S) \supseteq Z$ as well. □

Hence, for weakest liberal preconditions, we can always reach the fixpoint after at most ω approximations.

Note on fixpoint semantics for PDL. The previous results carry over to the definition of $[p^*]q$ in PDL, where the corresponding fixpoint operator is

$$\begin{aligned} ([\alpha^*]q)^{\mathcal{I}} = \;& \nu X.\, \sigma^{\mathcal{I}}_{\alpha,q^{\mathcal{I}}}(X) \\ & \text{where } \sigma^{\mathcal{I}}_{\alpha,Y}(X) = X \cap \{s \in S^{\mathcal{I}} | Y(s) \wedge \forall t : R^{\mathcal{I}}_{\alpha}(s,t) \rightarrow X(t)\} \end{aligned} \tag{8}$$

Furthermore, in PDL we can also define the state transformer of p^* using a binary fixpoint operator. The standard definition for the semantics of iteration is

$$R^{\mathcal{I}}_{\alpha^*} = \bigcup_{0 \leq n < \omega} (R^{\mathcal{I}}_{\alpha})^n \tag{9}$$

We can rewrite this definition as

$$R^{\mathcal{I}}_{\alpha^*} = \bigcup_{0 \leq n < \omega} \pi^n(\emptyset) \text{ where } \pi^{\mathcal{I}}_{\alpha}(R) = id^{\mathcal{I}} \cup R^{\mathcal{I}}_{\alpha} \circ R$$

where id is the identity relation. It is easily observed that this is the least fixpoint of the operator π, so that we can provide a fixpoint definition for the semantics of α^* by using a binary least-fixpoint operator:

$$R^{\mathcal{I}}_{\alpha^*} = \mu R.\, \pi^{\mathcal{I}}_{\alpha}(R) = \bigcup_{0 \leq n < \omega} \pi^n(\emptyset) \text{ where } \pi^{\mathcal{I}}_{\alpha}(R) = id^{\mathcal{I}} \cup R^{\mathcal{I}}_{\alpha} \circ R \tag{10}$$

So in PDL, we can always reach the fixpoint after at most ω approximations, no matter whether we define the weakest liberal precondition by a predicate transformer which uses a unary fixpoint operator or by a state transformer which uses a binary fixpoint operator. **End of Note.**

6.2 Total correctness: fixpoint semantics for $[p^*]^+q$

Turning now towards total correctness, recall the standard PDL^+ definition of $[p^*]^+q$ which uses (5):

$$\begin{aligned}([\alpha^*]^+q)^{\mathcal{I}} &= \{s_0 \in S^{\mathcal{I}} | \forall t \in S^{\mathcal{I}}_{\perp} : (s_0, t) \in R^{\mathcal{I}}_{\alpha^*} \Rightarrow t \in q^{\mathcal{I}}\} \\ &= \{s_0 \in S^{\mathcal{I}} | \forall t \in S^{\mathcal{I}}_{\perp} : ((s_0, t) \in \textstyle\bigcup_{0 \leq n < \omega} (R^{\mathcal{I}}_{\alpha})^n \rightarrow q^{\mathcal{I}}(t)) \\ &\quad \wedge \neg \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R^{\mathcal{I}}_{\alpha}\}\end{aligned} \tag{11}$$

And like in the case of the weakest liberal precondition, we can again use a unary fixpoint operator on predicates in defining $[\alpha^*]^+q$ directly:

$$\begin{aligned}([\alpha^*]^+q)^{\mathcal{I}} = \quad & \mu X.\, \rho^{\mathcal{I}}_{\alpha, q^{\mathcal{I}}}(X) \\ & \text{where } \rho^{\mathcal{I}}_{\alpha, Y}(X) = X \cup \{s \in S^{\mathcal{I}} | Y(s) \wedge \forall t : R^{\mathcal{I}}_{\alpha}(s,t) \rightarrow X(t)\}\end{aligned} \tag{12}$$

Like in the case of the weakest precondition predicate transformer of the while-loop, (12) approximates the least fixpoint starting at the final states where the program cannot be iterated anymore.

Again, like σ, ρ is monotone, so the least fixpoint exists, and again we can show that the two resulting definitions of $[\alpha^*]^+q$ coincide:

Theorem 38 *The two semantic definitions of* $[\alpha^*]^+q$ *coincide, i.e. (11) = (12).*

Proof. We must show that

$$\begin{aligned}\mu X.\, \rho^{\mathcal{I}}_{\alpha, Y}(X) = \{s_0 \in S^{\mathcal{I}} | \quad & \forall t \in S^{\mathcal{I}} : ((s_0, t) \in \textstyle\bigcup_{0 \leq n < \omega} (R^{\mathcal{I}}_{\alpha})^n \rightarrow Y(t)) \\ & \wedge \neg \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R^{\mathcal{I}}_{\alpha}\}\end{aligned}$$

So let Φ be the set on the right-hand side of the equation. First, we show that Φ is a fixpoint of ρ, i.e. $\rho^{\mathcal{I}}_{\alpha,Y}(\Phi) = \Phi$. Clearly, we have $\Phi \subseteq \rho^{\mathcal{I}}_{\alpha,Y}(\Phi)$. For the other inclusion, suppose by reductio that $s_0 \in \rho^{\mathcal{I}}_{\alpha,Y}(\Phi)$ while $s_0 \notin \Phi$. Then $Y(s_0) \wedge \forall t : (R^{\mathcal{I}}_{\alpha}(s_0, t) \rightarrow \Phi(t))$ holds by definition of ρ. We know that there can be no infinite α-sequences $s_0, s_1, s_2, \cdots$ starting at s_0, for then s_1 would have to be in Φ, and so the sequence $s_1, s_2, \cdots$ would be an infinite sequence starting with an element in Φ, a contradiction. On the other hand, suppose that

$(s_0, t) \in \bigcup_{0 \leq n < \omega} (R_\alpha^\mathcal{I})^n$, i.e. for some $n \geq 0$, $(s_0, t) \in (R_\alpha^\mathcal{I})^n$. For $n = 0$, $t = s_0$, and so $Y(t)$ holds by assumption. Otherwise, $n = m + 1$, and $(s_0, t) \in (R_\alpha^\mathcal{I})^{m+1}$ means that $(s_0, t') \in R_\alpha^\mathcal{I}$ and $(t', t) \in (R_\alpha^\mathcal{I})^m$. Thus, t' must be in Φ, and so $Y(t)$ must hold as well. Hence, both conditions for membership in Φ are satisfied, i.e. $s_0 \in \Phi$, a contradiction.

Second, we show that for any $Z \subseteq S^\mathcal{I}$ with $\rho_{\alpha,Y}^\mathcal{I}(Z) = Z$, $\Phi \subseteq Z$ must hold. Define the α-*height* of a state s as follows: s has α-height n iff n is the smallest number such that for all states t, if $(s, t) \in (R_\alpha^\mathcal{I})^m$ then $m \leq n$. Thus, the α-height is the least upper bound on the length of α-sequences. We show by induction on n that for all $s_0 \in \Phi$, if s_0 has α-height n, then $s_0 \in \rho_{\alpha,Y}^\mathcal{I}(Z)$. So take any $s_0 \in \Phi$. Clearly, $Y(s_0)$ holds for any α-height. If $n = 0$, $s_0 \in \rho_{\alpha,Y}^\mathcal{I}(Z)$ holds automatically, since no α- successors exist. For the inductive step, suppose s_0 has α-height n. Take any $(s_0, t) \in R_\alpha^\mathcal{I}$. Then t has an α-height smaller than n. Since t must also be in Φ, by inductive hypothesis, $t \in Z$. Hence, since Φ contains no infinite α-sequences and every $s \in \Phi$ thus has an α-height, we are done. □

Like in the case of $[p^*]q$, we are able to improve definition 12 by giving a bound on the fixpoint. In this case however, this improvement is only possible under suitable conditions. For this, we will introduce the following notion:

Definition 22 (finitary operator) *An operator $\tau : \mathcal{P}(A) \to \mathcal{P}(A)$ is called* finitary *iff for all $X \subseteq A$ and $a \in A$ such that $a \in \tau(X)$ there is a finite $Y \subseteq X$ such that $a \in \tau(Y)$*

Clearly, if $R_\alpha^\mathcal{I}$ is bounded nondeterministic, $\rho_{\alpha,q^\mathcal{I}}^\mathcal{I}$ is finitary for any choice of q. A proof of the following result can be found e.g. in [Doe94]:

Theorem 39 *For finitary operators τ on $\mathcal{P}(S)$, we have*

$$\mu X.\, \tau(X) = \bigcup_{0 \leq n < \omega} \tau^n(\emptyset)$$

Corollary 40 *If $R_\alpha^\mathcal{I}$ is only bounded nondeterministic,*

$$([\alpha^*]^+ q)^\mathcal{I} = \bigcup_{0 \leq n < \omega} (\rho_{\alpha,q^\mathcal{I}}^\mathcal{I})^n(\emptyset)$$

with ρ as defined in (12).

Questions & Conjectures 2 *Can we give a binary fixpoint definition for $R_{\alpha^*}^\mathcal{I}$ in PDL$^+$, like we did in PDL?*

7 A Dynamic Logic of Predicate Transformers

In the previous discussion, we provided a predicate transformer definition of $[p^*]^+ q$ in terms of least fixpoints. Hence, it is natural to investigate predicate transformer versions of PDL^+, where all semantic definitions are in terms of predicate transformers. Such a logic (PDL^{pt}) will be introduced and we show that a suitable restriction of it is equivalent to PDL^+. Next, we investigate finitary versions of this logic and show that these logics are sensitive to the complexity difference between bounded and unbounded nondeterminism. Finally, it is argued that PDL^{pt} provides a very general framework for reasoning about partial and total correctness.

7.1 PDL^{pt}

Definition 23 (PDL^{pt}) *The syntax of PDL^{pt} is the same as that of PDL^+. A model for PDL^{pt} is a pair $\mathcal{I} = (S^{\mathcal{I}}, m^{\mathcal{I}})$, where $S^{\mathcal{I}}$ is the set of states or the universe, and for all atomic programs a,*

$$m^{\mathcal{I}}(a) = F_a^{\mathcal{I}} : \mathcal{P}(S^{\mathcal{I}}) \to \mathcal{P}(S^{\mathcal{I}})$$

Since we assume that atomic programs always terminate, F_a is the wlp- as well as wp-transformer of atomic program a. For all atomic propositions p,

$$m^{\mathcal{I}}(p) = p^{\mathcal{I}} \subseteq S^{\mathcal{I}}$$

We extend $F_\alpha^{\mathcal{I}}$ to non-atomic programs α inductively as follows:

$$\begin{aligned}
F^{\mathcal{I}}_{\alpha;\beta}(Y) &= F^{\mathcal{I}}_{\alpha}(F^{\mathcal{I}}_{\beta}(Y)) \\
F^{\mathcal{I}}_{\alpha\cup\beta}(Y) &= F^{\mathcal{I}}_{\alpha}(Y) \cap F^{\mathcal{I}}_{\beta}(Y) \\
F^{\mathcal{I}}_{\varphi?}(Y) &= \neg\varphi^{\mathcal{I}} \cup Y \\
F^{\mathcal{I}}_{\alpha^*}(Y) &= \mu X.\, \rho^{\mathcal{I}}_{\alpha,Y}(X) \\
&\quad \textit{where } \rho^{\mathcal{I}}_{\alpha,Y}(X) = X \cup \{s \in S^{\mathcal{I}} | Y(s) \wedge s \in F^{\mathcal{I}}_{\alpha}(X)\}
\end{aligned}$$

Thus, for non-atomic programs, F represents the wp-transformer. Analogously, we define another transformer G for all programs which represents the wlp-transformer. For atomic programs a, $G_a^{\mathcal{I}} = F_a^{\mathcal{I}}$. For non-atomic programs α, the clauses for $G_\alpha^{\mathcal{I}}$ are exactly analogous to those of $F_\alpha^{\mathcal{I}}$, except that we have

$$\begin{aligned}
G^{\mathcal{I}}_{\alpha^*}(Y) &= \nu X.\, \sigma^{\mathcal{I}}_{\alpha,Y}(X) \\
&\quad \textit{where } \sigma^{\mathcal{I}}_{\alpha,Y}(X) = X \cap \{s \in S^{\mathcal{I}} | Y(s) \wedge s \in G^{\mathcal{I}}_{\alpha}(X)\}
\end{aligned}$$

The semantic definitions for pure boolean statements is as usual; for $[\alpha]^+\varphi$, we define $([\alpha]^+\varphi)^{\mathcal{I}} = F_\alpha^{\mathcal{I}}(\varphi^{\mathcal{I}})$ and $([\alpha]\varphi)^{\mathcal{I}} = G_\alpha^{\mathcal{I}}(\varphi^{\mathcal{I}})$. We obtain various restrictions of PDL^{pt} by restricting the atomic predicate transformer, so that e.g. universally conjunctive PDL^{pt} refers to PDL^{pt} with the F-transformers of all atomic programs being universally conjunctive.

Definition 24 (Translation Function τ from PDL^+ into PDL^{pt}) *The translation function τ maps a PDL^+ model $\mathcal{I}$ to a PDL^{pt} model $\mathcal{I}'$ by removing the failure state $\perp$ and by assigning to any atomic state transformer f_a the corresponding wlp-transformer using theorem 4. This result demonstrates that τ is injective and that for universally conjunctive PDL^{pt}, τ is also surjective.*

Recall that in $NST_\perp$ and hence in PDL^+, weakest precondition and weakest liberal precondition were defined as follows:

$$\begin{array}{ll} wp_\alpha^{\mathcal{I}}(X) & = f_\alpha^{-1}[\mathcal{P}(X)] = \{s \in S^{\mathcal{I}} | \forall t : R_\alpha^{\mathcal{I}}(s,t) \rightarrow X(t)\} \\ wlp_\alpha^{\mathcal{I}}(X) & = f_\alpha^{-1}[\mathcal{P}(X \cup \{\perp\})] = \{s \in S^{\mathcal{I}} | \forall t : R_\alpha^{\mathcal{I}}(s,t) \rightarrow (X(t) \vee t = \perp)\} \end{array}$$

The following lemma is easy to demonstrate:

Lemma 41 *The following hold in PDL^+:*

$$\begin{array}{lll} (a) & wp_{\varphi?}^{\mathcal{I}}(X) & = \neg\varphi^{\mathcal{I}} \cup X \\ (b) & wp_{\alpha;\beta}^{\mathcal{I}}(X) & = wp_\alpha^{\mathcal{I}}(wp_\beta^{\mathcal{I}}(X)) \\ (c) & wp_{\alpha\cup\beta}^{\mathcal{I}}(X) & = wp_\alpha^{\mathcal{I}}(X) \cap wp_\beta^{\mathcal{I}}(X) \end{array}$$

The analogous results hold for wlp as well.

The main lemma for the equivalence states that the correspondence between state transformers in PDL^+ and predicate transformers in PDL^{pt} holds not only on the level of atomic programs, but extends to all programs.

Lemma 42 *For all PDL^+ models $\mathcal{I}$,*

$$\begin{array}{ll} \forall\alpha\forall X : wp_\alpha^{\mathcal{I}}(X) & = F_\alpha^{\tau(\mathcal{I})}(X) \\ \forall\alpha\forall X : wlp_\alpha^{\mathcal{I}}(X) & = G_\alpha^{\tau(\mathcal{I})}(X) \end{array}$$

Proof. The proof is by induction on α. For atomic programs, the lemma holds by the definition of τ. For $\varphi?$, $\alpha;\beta$ and $\alpha \cup \beta$, the equality follows from the preceding lemma. Regarding α^*, we can utilize the correspondences from the previous section on fixpoints, namely theorems 38 and 36.

$$\begin{array}{ll} wp_{\alpha^*}^{\mathcal{I}}(X) & = \{s_0 \in S^{\mathcal{I}} | \forall t : R_{\alpha^*}^{\mathcal{I}}(s_0,t) \rightarrow X(t)\} \\ & = \{s_0 \in S^{\mathcal{I}} | \forall t : ((s_0,t) \in \bigcup_{0\le n<\omega} (R_\alpha^{\mathcal{I}})^n \rightarrow X(t)) \\ & \quad \wedge \neg\exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_\alpha^{\mathcal{I}}\} \\ & = \mu Z.\ \rho_{\alpha,X}^{\mathcal{I}}(Z) \\ & \quad \text{where } \rho_{\alpha,X}^{\mathcal{I}}(Z) = Z \cup \{s \in S^{\mathcal{I}} | X(s) \wedge \forall t : R_\alpha^{\mathcal{I}}(s,t) \rightarrow Z(t)\} \\ & = \mu Z.\ \rho_{\alpha,X}^{\mathcal{I}}(Z) \\ & \quad \text{where } \rho_{\alpha,X}^{\mathcal{I}}(Z) = Z \cup \{s \in S^{\mathcal{I}} | X(s) \wedge s \in wp_\alpha^{\mathcal{I}}(Z)\} \\ & = \mu Z.\ \rho_{\alpha,X}^{\mathcal{I}}(Z) \\ & \quad \text{where } \rho_{\alpha,X}^{\mathcal{I}}(Z) = Z \cup \{s \in S^{\mathcal{I}} | X(s) \wedge s \in F_\alpha^{\tau(\mathcal{I})}(Z)\} \\ & = F_{\alpha^*}^{\tau(\mathcal{I})}(X) \end{array}$$

Similarly for wlp, we have

$$\begin{aligned}
wlp^{\mathcal{I}}_{\alpha^*}(X) &= \{s_0 \in S^{\mathcal{I}} | \forall t : R^{\mathcal{I}}_{\alpha^*}(s_0, t) \to (X(t) \vee t = \perp)\} \\
&= \{s_0 \in S^{\mathcal{I}} | \forall t : ((s_0, t) \in \bigcup_{0 \leq n < \omega} (R^{\mathcal{I}}_{\alpha})^n \to (X(t) \vee t = \perp))\} \\
&= \nu Z.\ \sigma^{\mathcal{I}}_{\alpha,X}(Z) \\
&\quad\ \text{where } \sigma^{\mathcal{I}}_{\alpha,X}(Z) = Z \cap \{s \in S^{\mathcal{I}} | X(s) \wedge \forall t : R^{\mathcal{I}}_{\alpha}(s,t) \to (Z(t) \vee t = \perp)\} \\
&= \nu Z.\ \sigma^{\mathcal{I}}_{\alpha,X}(Z) \\
&\quad\ \text{where } \sigma^{\mathcal{I}}_{\alpha,X}(Z) = Z \cap \{s \in S^{\mathcal{I}} | X(s) \wedge s \in wlp^{\mathcal{I}}_{\alpha}(Z)\} \\
&= \nu Z.\ \sigma^{\mathcal{I}}_{\alpha,X}(Z) \\
&\quad\ \text{where } \sigma^{\mathcal{I}}_{\alpha,X}(Z) = Z \cap \{s \in S^{\mathcal{I}} | X(s) \wedge s \in G^{\tau(\mathcal{I})}_{\alpha}(Z)\} \\
&= G^{\tau(\mathcal{I})}_{\alpha^*}(X)
\end{aligned}$$

□

Theorem 43 *For all PDL^+ models $\mathcal{I}$ and all PDL^+ formulas ϕ*

$$\phi^{\mathcal{I}} = \phi^{\tau(\mathcal{I})}$$

Proof. An easy induction using the previous lemma. □

Corollary 44 *For all PDL^+ formulas ϕ, ϕ is valid in PDL^+ if and only if ϕ is valid in universally conjunctive PDL^{pt}.*

Given the equivalence between $LPDL$ and PDL^+ from corollary 34, we also obtain an equivalence between universally conjunctive PDL^{pt} and $LPDL$ under translation.

7.2 Finitary versions of PDL^{pt}: investigating expressiveness

We discovered earlier that for unbounded nondeterministic programs, the calculation of the weakest precondition may take more than ω iterations, while for bounded nondeterminism we could guarantee that fixpoint approximation up to ω is sufficient. This suggests weakening PDL^{pt} in the following way:

Definition 25 (PDL^{pt}_{ω}) *PDL^{pt}_{ω} is defined exactly like PDL^{pt}, except that $F^{\mathcal{I}}_{\alpha^*}$ is now defined as*

$$\begin{aligned}
F^{\mathcal{I}}_{\alpha^*}(Y) &= \bigcup_{0 \leq n < \omega} (\rho^{\mathcal{I}}_{\alpha,Y})^n(\emptyset) \\
&\quad\ \textit{where } \rho^{\mathcal{I}}_{\alpha,Y}(X) = X \cup \{s \in S^{\mathcal{I}} | Y(s) \wedge s \in F^{\mathcal{I}}_{\alpha}(X)\}
\end{aligned}$$

Using such a finitary total correctness logic, we can now obtain results analogous to those for infinitary logic: The complexity difference between bounded and unbounded nondeterminism can now be recaptured in Dynamic Logic as well.

The first result establishes that PDL^{pt} is expressive enough to pick out models with no infinite p-branches, where p is an atomic program. Thus, PDL^{pt} is strong enough so that this class of structures is definable in the language. This can be seen as establishing that PDL^{pt} is expressive enough for total correctness reasoning.

Theorem 45 *In universally conjunctive PDL^{pt}, we have for any atomic program p*

$$\mathcal{I} \models [p^*]^+ true \quad \textit{iff} \quad \mathcal{I} \textit{ contains no infinite } p\textit{-branch}$$

Proof. Let $f_p = R_p^{\mathcal{I}}$ be the state transformer corresponding to $F_p^{\mathcal{I}}$. Then using theorem 38 we have

$$\begin{aligned}
[[p^*]^+ true^{\mathcal{I}} &= \mu X.\ \rho^{\mathcal{I}}_{p,true}(X) \\
&\quad \text{where } \rho^{\mathcal{I}}_{p,true}(X) = X \cup \{s \in S^{\mathcal{I}} | s \in F_p^{\mathcal{I}}(X)\} \\
&= \mu X.\ \rho^{\mathcal{I}}_{p,true}(X) \\
&\quad \text{where } \rho^{\mathcal{I}}_{p,true}(X) = X \cup \{s \in S^{\mathcal{I}} | f_p(s) \subseteq X\} \\
&= \{s_0 \in S^{\mathcal{I}} | \forall t \in S^{\mathcal{I}} : ((s_0, t) \in \textstyle\bigcup_{0 \leq n < \omega} (R_p^{\mathcal{I}})^n \rightarrow t \neq \perp) \\
&\quad \wedge \neg \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_p^{\mathcal{I}}\} \\
&= \{s_0 \in S^{\mathcal{I}} | \neg \exists s_1 \exists s_2 \cdots \forall i \geq 0 (s_i, s_{i+1}) \in R_p^{\mathcal{I}}\}
\end{aligned}$$

□

The second result can be seen as demonstrating that PDL^{pt}_{ω} fails to be globally expressive:

Theorem 46 *In universally conjunctive PDL^{pt}_{ω}, we have for any atomic program p:*

$$\neg \exists \phi : \quad \mathcal{I} \models \phi \quad \textit{iff} \quad \mathcal{I} \textit{ contains no infinite } p\textit{-branch}$$

Proof. Consider the following two models for universally conjunctive PDL^{pt}_{ω}, where all the links are p-links and all the proposition letters are false everywhere in both models.

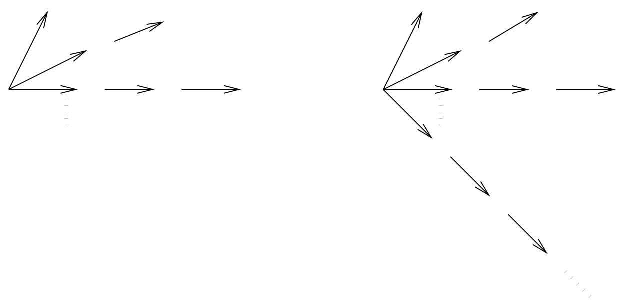

Figure 1: The models $\mathcal{I}_1$ and $\mathcal{I}_2$

Both of these models have infinitely many branches, but only $\mathcal{I}_2$ has an infinite p-branch. It can be shown by induction that in PDL^{pt}_{ω}, the same formulas hold at the roots of these models. Hence, there can be no formula defining the class of models with no infinite p-branches. □

The above proof demonstrates that PDL^{pt}_{ω} can indeed be considered the finitary restriction of PDL^{pt}. Like in most modal logics, the modal operators only allow for looking finitely far ahead; in contrast, by checking for infinite chains, PDL^{pt} introduced operators which have an essentially infinite character.

Lastly, if we restrict ourselves to or-continuous basic programs, we can re-establish expressiveness for PDL^{pt}_{ω}:

Theorem 47 *In universally conjunctive or-continuous PDL^{pt}_{ω}, we have for any atomic program p*

$$\mathcal{I} \models [p^*]^+ true \quad \textit{iff} \quad \mathcal{I} \textit{ contains no infinite } p\textit{-branch}$$

Proof. From theorem 10, we know that there is a state transformer $f_p = R^{\mathcal{I}}_p$ corresponding to $F^{\mathcal{I}}_p$ such that f_p is bounded nondeterministic. Then using corollary 40 and theorem 38, we have

$$\begin{aligned}
[[p^*]^+ true^{\mathcal{I}} &= \textstyle\bigcup_{0 \le n < \omega} (\rho^{\mathcal{I}}_{p,true})^n(\emptyset) \\
&\quad \text{where } \rho^{\mathcal{I}}_{p,true}(X) = X \cup \{s \in S^{\mathcal{I}} | s \in F^{\mathcal{I}}_p(X)\} \\
&= \mu X.\ \rho^{\mathcal{I}}_{p,true}(X) \\
&\quad \text{where } \rho^{\mathcal{I}}_{p,true}(X) = X \cup \{s \in S^{\mathcal{I}} | f_p(s) \subseteq X\} \\
&= \{s_0 \in S^{\mathcal{I}} | \forall t \in S^{\mathcal{I}} : ((s_0, t) \in \textstyle\bigcup_{0 \le n < \omega} (R^{\mathcal{I}}_p)^n \rightarrow t \ne \perp) \\
&\quad \wedge \neg \exists s_1 \exists s_2 \cdots \forall i \ge 0 (s_i, s_{i+1}) \in R^{\mathcal{I}}_p\} \\
&= \{s_0 \in S^{\mathcal{I}} | \neg \exists s_1 \exists s_2 \cdots \forall i \ge 0 (s_i, s_{i+1}) \in R^{\mathcal{I}}_p\}
\end{aligned}$$

□

7.3 A plea for PDL^{pt}

The previous results are just beginning to investigate the predicate transformer logics just proposed. However, they also suggest the naturalness of finitary PDL^{pt} and universally conjunctive PDL^{pt}. Hence, the following four logics and the relations between them seem to us worth investigating.

$$\begin{tikzcd}
 & PDL^{pt} \arrow[dl] \arrow[dr] & \\
\begin{array}{c}\text{universally conjunctive } PDL^{pt} \\ (\equiv PDL^+ \equiv LPDL)\end{array} \arrow[dr] & & PDL^{pt}_{\omega} \arrow[dl] \\
 & \text{universally conjunctive } PDL^{pt}_{\omega} &
\end{tikzcd}$$

Figure 2: Four natural predicate transformer logics

There are a number of reasons why we think that PDL^{pt} provides a useful framework for total correctness reasoning. First, PDL^{pt} is general enough to allow for non-conjunctive predicate transformers. On the one hand, this means that PDL^{pt} allows for basic programs which are not universally conjunctive. We have argued in section 3.5 that if we want to model actions of goal-sensitive agents, such programs/actions are not at all unnatural. On the other hand, PDL^{pt} would allow adding e.g. angelic choice as a new operator to the language, a language extension impossible in state transformer logics. Note also that in general PDL^{pt}, models cannot be visualized anymore by the standard diagrams used in Dynamic Logic, since these diagrams presuppose that the predicate transformers have corresponding state transformers.

Second, while being more general than PDL plus *loop* ($LPDL$), we can easily restrict PDL^{pt} to $LPDL$. So just like in modal logic where we obtain different modal logics by restricting the class of models, we obtain $LPDL$ from PDL^{pt} by restricting the class of models to universally conjunctive ones.

Third, PDL^{pt} can be easily restricted to a finitary logic PDL^{pt}_{ω}. Such a finitary logic is still expressive enough for certain program classes and may turn out to be less complex (e.g. in terms of its axiomatization) than PDL^{pt}.

Thus, we hope to have convinced the reader that a further investigation of these logics is worthwhile. Two lines for further investigation seem most important: On the one hand, as exemplified in the previous subsection, we should investigate the expressiveness of these logics, obtaining more general results than the ones given here, maybe placing these logics in an expressiveness hierarchy. On the other hand, we should investigate axiomatizations for the predicate transformer logics. It is not immediately clear whether/how the standard PDL axiom system can be augmented and proved complete for these logics. Given that the finitary logics are less expressive, we may obtain a simpler axiom system or a simpler completeness proof for these logics. The natural place to start may be working towards an axiomatization of PDL^+ which mirrors the axiomatization of PDL.

8 Conclusions

Transforming predicates or updating states? We showed that the predicate transformer approach has a number of advantages. First, it provides for a mathematically "nicer" model, by talking about mappings over $\mathcal{P}(S)$ rather than about functions from states to sets of states which may include a special error state. Second, this mathematical model provides for a more elegant fixpoint theory. Third, predicate transformers are more general in that certain non-conjunctive programming constructs such as angelic choice can only be modeled by predicate transformers. Fourth, the conceptualization of programs in terms of functions from postcondition to weakest preconditions may be quite close to the practice of the working computer scientist who wants to actually prove the total correctness of her program.

On the other hand, the predicate transformer approach also has some drawbacks. First, not all the information contained in a state transformer can be captured in the weakest precondition transformer. Depending on what our goal is, we may however be interested in this extra information, which requires us to introduce a second predicate transformer for the weakest liberal precondition. However, with two predicate transformers necessary to model one state transformer, the earlier comments regarding the mathematical simplicity of the PT-approach lose some force.

Second, if we decide that we want to distinguish between abort and while *true* do skip, we also need to modify the basic predicate transformer approach. Such a modification would also have to add some kind of failure state to the set of states, and so the resulting transformers would also lose the simplicity which made them attractive. This is probably also the reason why in most of the literature on predicate transformers, abort is either not considered to be part of the language or simply identified with while *true* do skip.

Returning to the general perspective of the introduction, the paper has demonstrated how applying different models of actions to particular logical systems shapes and transforms these systems. The study of infinitary logic uncovered a complexity boundary between bounded nondeterministic actions and unbounded nondeterministic actions, whereas the study of dynamic logic put the crucial boundary between deterministic and nondeterministic actions, although it turned out that under suitable modifications of the semantics, the complexity difference between bounded and unbounded nondeterminism could be recovered.

Beyond comparing two semantic traditions, we have also demonstrated that interesting results can be obtained from their cross-fertilization. In the case of Dynamic Logic, it has been demonstrated how one can modify a logic which uses one particular model of actions by changing that model, in our case by replacing state transformers by predicate transformers. Besides investigating conditions under which the two logics are equivalent, the new logic becomes an object of investigation in its own right. The natural next step is to carry out such an investigation in more detail.

Acknowledgments. I would like to thank Johan van Benthem, Jan van Eijck and Anne Kaldewaij for the many ideas and comments they had regarding this paper, also concerning matters of presentation. Having three advisors can sometimes create a cognitive overload in terms of the ideas one has to absorb, especially if Johan van Benthem is one of the advisors. In my case this was even more true since the perspective of logicians is often rather different from the perspective of a computer scientist. Nonetheless, I am very thankful for the rich intellectual stimulation I received from these three people.

References

[AP86] K.R. Apt and G.D. Plotkin. Countable nondeterminism and random assignment. *Journal of the ACM*, 33:724–767, 1986.

[Bac80] R.J. Back. Semantics of unbounded nondeterminism. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 51–63. Springer Verlag, 1980.

[Bac81] R.J. Back. Proving total correctness of nondeterministic programs in infinitary logic. *Acta Informatica*, 15:233–249, 1981.

[BAHP82] M. Ben-Ari, J. Halpern, and A. Pnueli. Deterministic propositional dynamic logic: Finite models, complexity and completeness. *Journal of Computer and System Sciences*, 25:402–417, 1982.

[Bar77] Jon Barwise. An introduction to first-order logic. In Jon Barwise, editor, *Handbook of Mathematical Logic*. North-Holland Publishing Company, 1977.

[BES95] Rudolf Berghammer, Birgit Elbl, and Ulf Schmerl. Formalizing dijkstra's predicate transformer wp in weak second-order logic. *Theoretical Computer Science*, 146:185–197, 1995.

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, 1976.

[Doe94] Kees Doets. *From Logic to Logic Programming*. Foundations of Computing Series. MIT Press, 1994.

[DS90] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics*. Texts and Monographs in Computer Science. Springer Verlag, 1990.

[Har79] David Harel. *First-Order Dynamic Logic*, volume 68 of *Lecture Notes in Computer Science*. Springer Verlag, 1979.

[Har84] David Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II. D. Reidel Publishing Company, 1984.

[Hes90] Wim H. Hesselink. Modalities of nondeterminacy. In W. H. J. Feijen et al., editors, *Beauty is our business: a birthday salute to Edsger W. Dijkstra*, Texts and Monographs in Computer Science. Springer Verlag, 1990.

[HKT95] David Harel, Dexter Kozen, and Jerzy Tiuryn. Dynamic logic. unpublished draft, June 1995.

[HP78] D. Harel and V. Pratt. Nondeterminism in logics of programs. In *Proceedings of the 5th annual ACM Symposium on Priinciples of Programming Languages*, pages 203–213, 1978.

[HS82] D. Harel and R. Sherman. Looping vs. repeating in dynamic logic. *Information and Control*, 55:175–192, 1982.

[Kei71] H.J. Keisler. *Model Theory for Infinitary Logic.* North-Holland Publishing Company, 1971.

[NN92] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications.* Wiley, 1992.