Available online at [www.sciencedirect.com](http://www.sciencedirect.com/)

[Electronic Notes in Theoretical Computer Science 305 (2014) 5–18](http://dx.doi.org/10.1016/j.entcs.2014.06.002)

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Bisimilar and Logically Equivalent Programs in PDL

Mario R. F. Benevides[1](#_bookmark0)*,*[2](#_bookmark0)

*Computer Science Department and Systems and Computer Engineering Program Federal University of Rio de Janeiro*

*Brazil*

**Abstract**

In standard Propositional Dynamic Logic (PDL) literature [[5](#_bookmark21),[16](#_bookmark24),[4](#_bookmark20)] the semantics is given by Labeled Tran- sition Systems, where for each program π we a associate a binary relation R*π* . Process Algebras [[1](#_bookmark17),[8](#_bookmark22),[10](#_bookmark23),[2](#_bookmark18)] also give semantics to process (terms) by means of Labeled Transition Systems. In both formalisms, PDL and Process Algebra, the key notion to compare processes is bisimulation. In PDL, we also have the no- tion of logic equivalence, that can be used to prove that two programs π1 and π2 are logically equivalent π1 ϕ π2 ϕ. Unfortunately, logic equivalence and bisimulation do not match in PDL. Bisimilar

*▶ ⟨ ⟩ ↔ ⟨ ⟩*

programs are logic equivalent but the converse does not hold.

This paper proposes a semantics and an axiomatization for PDL that makes logically equivalent programs also bisimilar. We prove soundness, completeness and the finite model property.

*Keywords:* Bisimulation, Propositional Dynamic Logic, Modal Logic

# Motivation

In standard PDL literature [[5](#_bookmark21),[16](#_bookmark24),[4](#_bookmark20)] the semantics is given by Labeled Transition Systems, where for each program *π* we associate a binary relation *R*π. The sequential composition and non-deterministic choice operators are defined as the composition and union of relations respectively.

*R*π1;π2 = *R*π1 *◦ R*π2 *R*π1*∪*π2 = *R*π1 *∪ R*π2

Process Algebras [[1](#_bookmark17),[8](#_bookmark22),[10](#_bookmark23),[2](#_bookmark18)] also give semantics to process (terms) by means of Labeled Transition Systems. In both formalisms, PDL and Process Algebra, the key notion to compare processes is bisimulation. In PDL, we also have the notion of logic equivalence, that can be used to prove that two programs *π*1 and *π*2 are logically equivalent *▶ ⟨π*1*⟩ϕ ↔ ⟨π*2*⟩ϕ* (where *⟨π*i*⟩ϕ* means that after the execution

1 This work was supported by the Brazilian research agencies CNPq and CAPES.

2 Email: [mario@cos.ufrj.br](mailto:mario@cos.ufrj.br)

<http://dx.doi.org/10.1016/j.entcs.2014.06.002>

1571-0661/© 2014 Elsevier B.V. All rights reserved.

of program *π*i formula *ϕ* holds). Unfortunately, logic equivalence and bisimulation do not match in PDL. Bisimilar programs are logic equivalent but the converse does not hold. For instance, take programs *π*1 = *a*; (*π*3 *∪ π*4) and *π*2 = *a*; *π*3 *∪ a*; *π*4

*⟨a*; (*π*3 *∪ π*4)*⟩ϕ ↔ ⟨a⟩⟨*(*π*3 *∪ π*4)*⟩ϕ*

*↔ ⟨a⟩*(*⟨π*3*⟩ϕ ∨ ⟨π*4*⟩ϕ*)

*↔ ⟨a⟩⟨π*3*⟩ϕ ∨ ⟨a⟩⟨π*4*⟩ϕ*

*↔ ⟨a*; *π*3*⟩ϕ ∨ ⟨a*; *π*4*⟩ϕ*

*↔ ⟨a*; *π*3 *∪ a*; *π*4*⟩ϕ*

But it is not difficult to see that *π*1 and *π*2 are non-bisimilar programs, for after first *a* step on *π*1 it arrives at *π*3 *∪ π*4, and this is matched by neither of the two possibilities on *π*2: *π*3 or *π*4.

*◦*

π3

J

*π*2

z*z*

a

~*s*

a

*◦*

π4

J

*π*1

a

J

*◦*

π3

~*s*

π4

z*z*

*◦ ◦ ◦ ◦*

Fig. 1. Non-Bisimilar Programs

One interesting discussion on trace semantics appears on [[2](#_bookmark18)]. They define and compare various semantics for concrete sequential processes and provide algebraic axiomatization and semantical modal characterization (no modal axiomatization) for them.

The main motivation of this work is to propose a new semantics for PDL, based on traces with context, which matches the notion of logic equivalence and with bisimulation, i. e., two programs *π*1 and *π*2 are logically equivalent (*▶ ⟨π*1*⟩ϕ ↔*

*⟨π*2*⟩ϕ*) if and only if they are bisimilar. We provide an axiomatization and prove completeness w.r.t this new semantics. The proof completeness yields finite model property and decidability.

It is important to notice that our contribution is on PDL and not on process theory.

# Propositional Dynamic Logic

In this section, we present the syntax and semantics of PDL.

**Definition 2.1** The PDL language consists of a set Φ of countably many proposi- tion symbols, a set Π of countably many basic programs, the boolean connectives

*¬* and *∧*, the program constructors ;, *∪* and ٨ and a modality *⟨π⟩* for every program

*π*. The formulas are defined as follows:

*ϕ* ::= *p |Т| чϕ | ϕ*1 *Λ ϕ*2 *| ⟨π⟩ϕ,* with *π* ::= *a | π*1; *π*2 *| π*1 *∪ π*2 *| π*٨*,*

where *p ∈* Φ and *a ∈* Π.

In all the logics that appear in this paper, we use the standard abbreviations

*⊥≡ чТ*, *ϕ V φ ≡ ч*(*чϕ Λ чφ*), *ϕ → φ ≡ ч*(*ϕ Λ чφ*) and [*π*]*ϕ ≡ ч⟨π⟩чϕ*.

**Definition 2.2** A *frame* for PDL is a tuple *7* = (*W,* **R**) where

* *W* is a non-empty set of states;
* **R** = *{R*a *| a ∈* Π*}*, *R*a are binary relations over *W* , for each basic program *a ∈* Π;
* We can inductively define a binary relation *R*π, for each non-basic program *π*, as follows
  + *R*π1;π2 = *R*π1 *◦ R*π2 ,
  + *R*π1*∪*π2 = *R*π1 *∪ R*π2 ,
  + *R*π*s* = *R*٨ , where *R*٨ is the reflexive transitive closure of *R*π.

π

π

**Definition 2.3** A *model* for PDL is a pair *M* = (*7,* **V**), where *7* is a PDL frame and **V** is a valuation function **V** :Φ *→* 2W .

**Definition 2.4** Let *M* = (*7,* **V**) be a model. The notion of *satisfaction* of a formula *ϕ* in a model *M* at a state *w*, notation *M,w* H *ϕ*, can be inductively defined as follows:

* *M,w* H *p* iff *w ∈* **V**(*p*);
* *M,w* H *Т* always;
* *M,w* H *чϕ* iff *M,w /*H *ϕ*;
* *M,w* H *ϕ*1 *Λ ϕ*2 iff *M,w* H *ϕ*1 and *M,w* H *ϕ*2;
* *M,w* H *⟨π⟩ϕ* iff there is *wj ∈ W* s.t. *wR*π*wj* and *M, wj* H *ϕ*.

**Example 2.5** *M,w* H *⟨*(*π*1; *π*2)*⟩p* iff there is *wj ∈ W* s.t. *wR*π1;π2 *wj* and *M, wj* H *p*, iff there is *wj ∈ W* s.t. *wR*π1 *◦ R*π2 *wj* and *M, wj* H *p*

π1;π2

*w*

π1

J

*◦*

π2

J*z*

*p wj*

# Process Calculus

In this section, we propose a very small process (program) calculus for the PDL programs presented in the previous section [2](#_bookmark2). We prove that two processes are bisimilar if and only if they have the same set of finite possible runs with context. It is inspired on [[2](#_bookmark18)].

|  |  |  |
| --- | --- | --- |
| *α →*α *√* | *α.π →*α *π* | π*→α* π*′* π;τ *→α* π*′*;τ |
| π *α* π*′*  *→*  π+τ *α* π*′*  *→* | *β*  τ *→*τ *′*  π+τ *β* τ *′*  *→* | π *α* π*′*  *→*  π*∗ α* π*′*;π*∗*  *→* |

Table 1 Transition Relation

Let *N* = *{a, b, c, . . .}* be a set of names or actions, denoted by *α, β,* The

language can be defined as follows.

*π* ::= *α | π*1; *π*2 *| π*1 + *π*2 *| π*٨*, where α ∈N*

We use *π* and *τ* to denote processes (programs) and *α*, *β* and *γ* to denote actions.

We write *π →*α *πj* to express that the process *π* can perform the action *α* and

after that behave as *πj*. We write *π →*α *√* to express that the process *π* successfully

finishes after performing the action *α*. A process finishes when there is no possible

*√*

β

action left for it to perform. For example, *β →* .

The semantics of our process calculus can be given by the transition rules pre- sented in the table below.

The concept of bisimulation is a key notion in any process algebra. It is an equivalence relation between processes which have mutaly similar behavior. The intuition is that two bisimilar processes cannot be distinguished by an external observer.

**Definition 3.1** Let *P* be the set of all processes. A set *Z ⊆ P × P* is a *strong bisimulation* if (*π, τ* ) *∈ Z* implies the following:

* If *π →*

α

α

* If *τ →*

*πj*, then there is *τj ∈P* such that *τ →*α *τj*, then there is *πj ∈P* such that *π →*α

*τj* and (*πj,τj*) *∈ Z*; *πj* and (*πj,τj*) *∈ Z*;

* *π →*

α

*√* if and only if *τ →*α *√*.

**Definition 3.2** Two process *π* and *τ* are *strongly bisimilar* (or simply *bisimilar* ), denoted by *π ∼ τ* , if there is a strong bisimulation *Z* such that (*π, τ* ) *∈ Z*.

**Proposition 3.3** *π*1; (*π*2 + *π*3) */∼ π*1; *π*2 + *π*1; *π*3

See figue [1](#_bookmark1).

*3.1 Runs with Context*

In this section, we introduce the key concept of *ﬁnite possible runs with context* of a process. This concept plays a central role in the semantics of our logics.

**Definition 3.4** A sequence of action with context, denoted by *−→α* c, is a sequence of actions and finite sets of actions of the form

*α*1*{β*1 *··· β*1 *}.α*2*{β*2 *··· β*2 *}, ··· .α*n*{β*n *··· β*n *}. ··· ,*

1

k1

1

k2

1

k*n*

where *α*i */∈ {β*i *··· β*i *}*, for 1 *≤ i ≤ n*.

1

k*i*

If *−→α* c = *α*1*C*1*.α*2*C*2 *.α*n*C*n is a finite sequence of action with context, we say that the length of *−→α* c is *n*.

*···*

**Definition 3.5** Let *−→α* c = *α*1*C*1*.α*2*C*2 *··· .α*n*C*n and

*−→*c

*β* = *β*1*D*1*.β*2*D*2 *··· .β*n*D*n sequences of action with context of length *n*. We define a strict partial order over sequences of actions with context as follows

*−→α* c *≺ −→β* c iff for all *i*, 1 *≤ i ≤ n*, *α*i = *β*i,

*C*i *⊆ D*i,

for at least one *i*, *C*i *⊂ D*i.

**Definition 3.6** Let *−→α* c = *α*1*{β*1*...β*1 *}α*2*{β*2*...β*2 *} ...α*n*{β*n *··· β*n *}* be a sequence

1

k1

1

k2

1

k*n*

of action with context. We say that *−→α* c *matches a process π*0 if *π*0 α1

*→*

*→*

*π*1 α2

*π*2 *··· π*n*−*1 α*→n π*n and for all *i*, 0 *≤ i < n*, *{α*i*, β*i *··· β*i *}* are all the actions that *π*i

can perform.

1 k*i*

We write *π −→*α *c πj* to express that *−→α* c matches *π* and the process *π* may perform

*⇒ −→*c

*j −→*α *c √*

the sequence of actions *α* and after that behave as *π* . We write *π ⇒* to express

that *−→α* c matches *π* and the process *π* may successfully finish after performing the sequence of actions *−→α* c (this, in particular, implies that *−→α* c is finite).

**Definition 3.7** We define the set of finite possible runs with context of a process

*π*, denoted by *−R→*c (*π*), as *−R→*c (*π*)= *{−→α* c : *π −→*α *c √* .

f

f

*⇒*

*}*

In order to obtain the desired relation between bisimulation and logic equiva- lence, we introduce the concept of *ﬁnite* possible runs with context of processes, i.e., situations in which the processes successfully finish. Thus, we present some useful results about finite possible runs with context. It is important to notice that in our process calculus all processes, at any state in their execution, can only perform a finite set of actions, i.e., they are *image ﬁnite*.

**Definition 3.8** Let *R* and *S* be sets of finite sequences of actions with context. We can define the following operations on these sets:

1. *R ◦ S* = *{−→α* c*.−→β* c : *−→α* c *∈ R* and *−→β* c *∈ S}*;
2. *R ∪ S* = *{−→α* c : *−→α* c *∈ R* or *−→α* c *∈ S}*;
3. *R*0 = *{−→ε }*, *R*n = *R ◦ R*n*−*1(*n ≥* 1);
4. *R∗* = S

n*∈*N

*R*n.

**Lemma 3.9** *If π ∼ τ, then, for every −→α* c*, π −→*α *c √ iff τ −→*α *c √.*

*⇒ ⇒*

A proof of this lemma can be found in appendix [A](#_bookmark25). This proof is based in similar one presented in [[3](#_bookmark19)].

**Lemma 3.10** *If for every −→α* c*, π −→*α *c √ if and only if τ −→*α *c √, then π ∼ τ*

*⇒*

*⇒*

**Proof.** Suppose *π −→*α *c √* if and only if *τ −→*α *c √* and *π /∼ τ* . Then there exists *α* such

that *π →*α1 *π*

*/→*

*⇒*

and for all *τ*

1

α1

either *τ τ*

1

*⇒*

(1) or *π*

1

1

*/∼ τ*

1

(2). But (1) cannot be true,

1

because it contradicts the hypothesis that *π* and *τ* are able to perform the same set of actions, because if *α*1 is the first action in some sequence of action *−→α* c, then its context contains all the actions that *π* and *τ* can perform. The only remaining possibility is (2), *π*1 */∼ τ*1. If we apply the same reasoning for *π*2 */∼ τ*2 and so on

for *π*i */∼ τ*i and *π*i and *τ*i must be able to perform the same set of actions. As all processes eventually terminate, we must eventually reach a pair *π*n and *τ*n such that

*π*n */∼ τ*n and *π*n and *τ*n must be able to perform the same set of actions *γ*1*, ··· γ*k*n*

*√ √ √* n n

and either *π*n = or *τ*n = or both *π*n = *τ*n = . The first two cases are not

possible because *π*n and *τ*n must be able to perform the same set of action and *√* does not perform any action and any process different of *√* must be able to perform at least one action. Thus, the only possibility is *π*n = *τ*n = *√*, which yields that

*π*n *∼ τ*n, which is a contradiction. Therefore, *π ∼ τ* .

*2*

**Theorem 3.11** *π ∼ τ if and only if −R→*c (*π*)= *−R→*c (*τ* )*.*

f

f

**Proof.** (*⇒*) Suppose that *−→α* c *∈ −R→*c (*π*). Then, *π —→*α *c √*. As *π ∼ τ* , this implies, by

f

*—→c √*

*⇒ −→ −→*

lemma [3.9](#_bookmark7), that *τ ⇒* , which means that *−→α* c *∈ −R→*c (*τ* ). Thus, *R*c (*π*) *⊆ R*c (*τ* ).

α

f f f

The proof that *−R→*c (*τ* ) *⊆ −R→*c (*π*) is entirely analogous.

f

f

(*⇐*) Suppose that *−R→*c (*π*) = *−R→*c (*τ* ). By the definition of *−R→*c (*π*) and *−R→*c (*τ* ),

*π —→*α *c √*

f f f f

*—→*α *c √*

*⇒* if and only if *τ ⇒*

. And by lemma [3.10](#_bookmark8), *π ∼ τ* .

*2*

Next, we present some equalities between sets of finite possible runs that are useful to the soundness proof of our axiomatization.

**Definition 3.12** Let *−→α* c = *α*1*C*1*.α*2*C*2 *··· .α*n*C*n *∈ −R→*c (*π*1),

f

*{γ*1*, ··· , γ*m*}* be the set of all actions that *π*2 can perform and *C*1*j* = *C*1*∪{γ*1*, ··· , γ*m*}*

and *C*1*jj* = *C*1 *\ {γ*1*, ··· , γ*m*}*. We define

* *−→α* c *|*+π2 = *α*1*C*1*j .α*2*C*2 *··· .α*n*C*n
* *−→α* c *|—*π2 = *α*1*C*1*jj.α*2*C*2 *··· .α*n*C*n
* *−R→*c (*π*1) *|*+π2 = *{−→α* c *|*+π2 *| −→α* c *∈ −R→*c (*π*1)*}*

f

f

* *−R→*c (*π*1) *|—*π2 = *{−→α* c *|—*π2 *| −→α* c *∈ −R→*c (*π*1)*}*

f

f

**Theorem 3.13** *The following set equalities are true:*

* 1. *−R→*c (*α*)= *{α};*

f

* 1. *−R→*c (*π*1; *π*2)= *−R→*c (*π*1) *◦ −R→*c (*π*2)*;*

f

f

f

* 1. *−R→*c (*π*1 + *π*2)= *−R→*c (*π*1) *|*+π2 *∪−R→*c (*π*2) *|*+π1 *;*

f

f

f

* 1. *−R→*c (*π∗*)= (*−R→*c (*π*))*∗.*

f

f

**Proof.** The proof is straightforward from table [1](#_bookmark6). *2*

# PDL+

In this section we present the language, semantics and an axiomatic system of our Propositional Dynamic Logic with a non-deterministic choice operator.

* 1. *Language and Semantics*

The language is similar to the one presented in definition [2.1](#_bookmark3), where we replace + for *∪*.

*ϕ* ::= *p |Т| чϕ | ϕ*1 *Λ ϕ*2 *| ⟨π⟩ϕ,* with *π* ::= *a | π*1; *π*2 *| π*1 + *π*2 *| π*٨*,*

where *p ∈* Φ and *a ∈ N* .

**Definition 4.1** A *frame* for PDL+ is a tuple *7* = (*W, R*a) where

* *W* is a non-empty set of states;
* *R*a is a binary relation over *W* , for each basic program *a ∈* Π;
* We can inductively define a binary relation *R*π, for each non-basic program *π*, as follows
  + *R*π1;π2 = *R*π1 *◦ R*π2 ,
  + *R*π1+π2 = *{*(*s, t*) *|* [(*s, t*) *∈ R*π1 *and ∃r*(*s, r*) *∈ R*π2 ] *or*

[(*s, t*) *∈ R*π2 *and ∃r* (*s, r*) *∈ R*π1 ]*}*

* + *R*π*s* = *R*٨ , where *R*٨ is the reflexive transitive closure of *R*π.

π

π

The semantical notion of PDL+ model and satisfaction for PDL+ is as defined for PDL in definitions [2.3](#_bookmark4) and [2.4](#_bookmark5)

If *M,w* H *ϕ* for every state *w*, we say that *ϕ* is *globally satisﬁed* in the model *M*, notation *M* H *ϕ*. If *ϕ* is globally satisfied in all models *M* of a frame *7* , we say that *ϕ* is *valid* in *7* , notation *7* H *ϕ*. Finally, if *ϕ* is valid in all frames, we say that *ϕ* is valid, notation H *ϕ*. Two formulas *ϕ* and *ψ* are *semantically equivalent* if H *ϕ ↔ ψ*.

**Proposition 4.2** */*H *⟨π*1; (*π*2 + *π*3)*⟩p ↔ ⟨π*1; *π*2 + *π*1; *π*3*⟩p*

**Proof.** Let *M* be a model based on the frame bellow with *V* (*p*)= *{v}*. It is easy to verify that

*M,w /*H *⟨π*1; (*π*2 + *π*3)*⟩p* and

*M,w* H *⟨π*1; *π*2 + *π*1; *π*3*⟩p*

*w*

π1

~*s*

π2

z*z*

*◦ ◦*

π3 π3

JJ

*v ◦* *2*

Next definition and lemma relate our semantics with possible runs with context.

**Definition 4.3** Let *7* = (*W, R*α) be a frame, (*v*0*, v*1*,..., v*n),

*≥ 7* c *∈ −R→*c

*n* 1, be a finite path in and *−→α* (*π*), of length *n*, be a sequence of action with context of a process *π*. We say that *−→α* c *matches* path (*v*0*, v*1*,..., v*n) for

f

process *π* iff for all *i*, 1 *≤ i ≤ n*, (*−→α* c)i = *α*i*{β*i *...β*i *}* and (*v*i*—*1*, v*i) *∈ R*α and for

1

k*i*

*i*

all *β*j, 1 *≤ j ≤ k*i, there exists a w such that (*v*i*—*1*, w*) *∈ R*β . We say that *−→α* c

*j*

matches exactly path

(*v*0*, v*1*,..., v*n), if there exists a unique *w* such that (*v*i*—*1*, w*) *∈ R*γ, for all *γ ∈*

*{β*i *··· β*i *}* and 1 *≤ i ≤ n*.

1 k*i*

A frame *7 matches a process π at state w* iff for all *−→α* c *∈ −R→*c (*π*), there exists a

f

path *ρ*, in *7* , such that *−→α* c matches *ρ*.

**Lemma 4.4** *M,w* H *⟨π⟩ϕ iff 7 matches π at w, there is a ﬁnite path* (*v*0*, v*1*,...,* *v*n)*, n ≥* 1*, such that v*0 = *w, M, v*n H *ϕ and there is −→α ∈ R* (*π*) *of length n such*

f

c *−→*c

*that −→α* c *matches the path* (*v*0*,..., v*n)*.*

A proof of this lemma can be found in appendix [B](#_bookmark26).

Bellow, we present the main theorem of this section, it establishes the equivalence between bisimilar and logically equivalent programs.

**Theorem 4.5** *−R→*c (*π*)= *−R→*c (*τ* ) *if and only if* H *⟨π⟩p ↔ ⟨τ⟩p.*

f

f

**Proof.** (*⇒*) Suppose that *−R→*c (*π*) = *−R→*c (*τ* ), but */*H *⟨π⟩p ↔ ⟨τ⟩p*. Then, we may

f

f

assume, without loss of generality, that there is a model *M* and a state *v*0 in this model such that *M, v*0 H *⟨π⟩p* (1), but *M, v*0 */*H *⟨τ⟩p* (2). By lemma [4.4](#_bookmark13), (1) implies that there is a path (*v , v ,..., v* ), *n* 1, in such that *, v* H *p* (3) and there

0 1 n *≥ M M* n

is *−→α* c *∈ −R→*c (*π*) that matches this path. But as *−R→*c (*π*)= *−R→*c (*τ* ), then *−→α* c *∈ −R→*c (*τ* ).

f

f

f

f

This and (3) imply, by definition [4.4](#_bookmark13), that *, v*0 H *τ p*, which contradicts (2). (*⇐*) Suppose that H *⟨π⟩p ↔ ⟨τ⟩p* (1), but *−R→*c (*π*) */*= *−R→*c (*τ* ). Then, we may

*M ⟨ ⟩*

f

f

assume, without loss of generality, that there is *−→α* c such that *−→α* c *∈ −R→*c (*π*), but

f

*−→α* c */∈ −R→*c (*τ* ) and there is no *−→β* c *∈ −R→*c (*τ* ) such that *−→β* c *≺ −→α* c.

f

f

Let us build a frame , that matches *π*, which consists of a finite tree and has a path (*v*0*,..., v*n), *n ≥* 1, such that *−→α* c matches exactly the path (*v*0*,..., v*n) for process *π*.

*7*

Let *M* = (*7,* **V**) be a model, such that **V**(*p*) is a singleton which the only element is *v*n, *v*n *∈* **V**(*p*). Then, we have a path (*v*0*,..., v*n) such that *M, v*n H *p* and *−→α ∈ R* (*π*) matches this path. By lemma [4.4](#_bookmark13), *M, v*0 H *⟨π⟩p*.

f

c *−→*c

As *−→α* c */∈ −R→*c (*τ* ) and there is no *−→β* c *∈ −R→*c (*τ* ) such that

f

f

*−→β* c *≺ −→α* c, so as *−→α* c matches exactly the path (*v*0*,..., v*n), then (*v*0*,..., v*n) is not matched by any other sequence for process *τ* . Besides that, there is no other path (*v*0*,..., v*m), *m ≥* 1, in *M* such that *M, v*m H *p*, because *7* is a tree. Thus, by lemma [4.4](#_bookmark13), *M, v*0 */*H *⟨τ⟩p*, which contradicts (1). *2*

**Corollary 4.6** *π ∼ τ if and only if* H *⟨π⟩p ↔ ⟨τ⟩p.*

**Proof.** It follows directly from theorems [3.11](#_bookmark9) and [4.5](#_bookmark14).

*2*

* 1. *Axiomatization*

We use the standard boolean abbreviations *⊥*, *V*, *→* and *↔* and the following abbreviations for the duals: [*π*]*ϕ* := *ч⟨π⟩чϕ*.

The axiomatization presented below is the standard PDL proof theory extended with a new axiom for non-deterministic choice.

## Axioms

* + 1. *All tautologies*,
    2. [*π*](*ϕ → ψ*) *→* ([*π*]*ϕ →* [*π*]*ψ*),
    3. [*π*1; *π*2]*ϕ ↔* [*π*1][*π*2]*ϕ*,
    4. *⟨π*1 + *π*2*⟩p ↔* (*⟨π*1*⟩p V ⟨π*2*⟩p*) *Λ* (*⟨π*1*⟩Т Λ ⟨π*2*⟩Т*) ,
    5. [*π∗*]*ϕ ↔ ϕ Λ* [*π*][*π∗*]*ϕ*,
    6. [*π∗*](*ϕ →* [*π*]*ϕ*) *→* ([*π*]*ϕ →* [*π∗*]*ϕ*),

## Inference Rules

M.P. *ϕ, ϕ → ψ/ψ* U.G. *ϕ/*[*π*]*ϕ* SUB. *ϕ/σϕ*

where *σ* is a map uniformly substituting formulas for propositional variables.

Axioms 1, 2, 3, 5 and 6 and the inference rules are standard in PDL for regular programs [[5](#_bookmark21),[16](#_bookmark24),[4](#_bookmark20)]. Axiom 4 deserves some explanation. It can be re-written as

*⟨π*1 + *π*2*⟩p ↔* (*⟨π*1*⟩p Λ ⟨π*2*⟩Т*) *V* (*⟨π*1*⟩Т Λ ⟨π*2*⟩p*)

The intuitive meaning is ”whenever we perform a non-deterministic choice *π*1 + *π*2, we must be able to perform either *π*1 or *π*2, but both must be available for execution. This is what *⟨π*i*⟩Т* (*i* = 1*,* 2) assures, i.e., it is possible to perform *π*i.

**Example 4.7** *M,w* H *⟨*(*π*1 + *π*2)*⟩p* and *M,v /*H *⟨*(*π*1 + *π*2)*⟩p*

*w* *v*

π1

,*7*

π2

π1

z*z* J

*p ◦ ◦ p ◦*

* 1. *Soundness and Completeness*

In order to prove soundness it is necessary to show both that every axiom is valid in this class of frames and the inference rules also preserve the validity. The validity of axioms 1, 2, 3, 5 and 6 and the inference rules are well-known from the PDL literature [[5](#_bookmark21),[16](#_bookmark24),[4](#_bookmark20)]. Below, we present the proof for axioms 4.

**Lemma 4.8** *The following formula is valid:*

H *⟨π*1 + *π*2*⟩p ↔* (*⟨π*1*⟩p V ⟨π*2*⟩p*) *Λ* (*⟨π*1*⟩Т Λ ⟨π*2*⟩Т*) A proof of this lemma can be found in appendix [C](#_bookmark27).

**Theorem 4.9** *(***Soundness** *): PDL+ is sound.*

**Theorem 4.10** *(***Completeness for Finite PDL+ Models***): Propositional Dy- namic Logic PDL+ is complete with respect to the class of ﬁnite PDL+ models.*

A proof of this theorem can be found in appendix [D](#_bookmark28).

* 1. *Decidability and Complexity*

Section [4.3](#_bookmark15) proves that PDL+ is complete with respect to the classes of finite PDL+ models. Hence, it has have the finite model property, and moreover, that every consistent formula *ψ* can be satisfied ata state ofa model with at most 2*|*ψ*|*, where

*|ψ|* is the number of symbols of *ψ*. A naive decision procedure for the satisfiability problem of our logic could be: given a formula *ψ*, construct all Kripke models with at most 2*|*ψ*|* states, verify if they belong to the appropriate class, and test if *ψ* is satisfied at some state of them. There are approximately 22*|ψ|* such models. Therefore, this algorithm establishes a double exponential time upper bound for the satisfiability problem of our logic.

The satisfiability problem for PDL is EXPTIME-complete [[5](#_bookmark21)]. This yields an exponential time lower bound for the satisfiability problem of our logic.

# Conclusion

This paper presents a new semantics to PDL based on finite runs with context, as far as we know this is a new semantics and opens up new possibilities not only for PDL but for other modal logics as well. We propose an axiomatization and prove its soundness, completeness and finite model property. The main result is equivalence between bisimilar programs and logically equivalent programs.

We proved completeness with respect to the class of finite PDL+ and the com- plexity should be the same as for PDL.

PDL+ opens up possibilities to investigate new variants of PDL where programs are process terms from some process algebra. In [[3](#_bookmark19)], a Dynamic Logic for CCS programs was presented, the main criticism of this logic was the lack of equivalence between bisimilar processes and logically equivalent programs. This problem is completely solved with our new semantics. In [[3](#_bookmark19)], we also present a logic that uses recursion in the place of iteration. But, in order to keep decidability, we had to restrict the use of recursive equations. In the present work, we use iteration and finite runs, dealing only with terminating programs. We would like to extend this work with recursion and investigate more expressive semantics.

Another possibility for future work would be to establish the precise complexity of the satisfability problem for PDL+. We already have the EXPTIME-hardness due to PDL. We suspect it is EXPTIME-complete, as PDL, but we would like to provide an EXPTIME algorithm for the satisfability problem.

# References

1. J.A. Bergstra, A. Ponse and S.A. Smolka (editors), Handbook of Process Algebra, Elsevier, 2001.
2. R. J. van Glabbleek, *The Linear Time - Branching Time Spectrum I: The Semantics of Concrete, Sequential Processes*. In Handbook of Process Algebra (J.A. Bergstra, A. Ponse and S.A. Smolka, eds.), Chapter 1, pp. 3-99, Elsevier, 2001.
3. M. R. F. Benevides and L. M. Schechter. A propositional dynamic logic for CCS programs. In *Proceedings of the XV Workshop on Logic, Language, Information and Computation*, volume 5110 of *LNAI*, pages 83–97. Springer, 2008.
4. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Theoretical Tracts in Computer Science. Cambridge University Press, 2001.
5. D. Harel and D. Kozen and J. Tiuryn. *Dynamic Logics*. MIT Press, 2000.
6. M. Dam. On the decidability of process equivalences for the pi-calculus. *Theoretical Computer Science*, 183(2):215–228, 1997.
7. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
8. W. J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science. Springer, 2000.
9. D. Harel and D. Raz. Deciding properties of nonregular programs. *SIAM Journal on Computing*, 22(4):857–874, 1993.
10. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
11. R. Milner. *Communicating and Mobile Systems: the* π*-Calculus*. Cambridge University Press, 1999.
12. R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114(1):149–171, 1993.
13. D. Peleg. Communication in concurrent dynamic logic. *Journal of Computer and System Sciences*, 35(1):23–58, 1987.
14. D. Peleg. Concurrent dynamic logic. *Journal of the Association for Computing Machinery*, 34(2):450– 479, 1987.
15. C. Stirling. *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer, 2001.
16. R. Goldblatt. *Logics of Time and Computation*. CSLI Lecture Notes 7. Stanford, 1992.

# Proof of lemmas [3.10](#_bookmark8)

**Proof.** We prove by induction on the length n of *—→*α *c*.

* + n = 1, then *—→*α *c* = α*{*β1 *···* β1 *}*, for some action α. Then, π *α √ ⇔* π *α √*. By the hypothesis that

*→− c*

1 *k*1

*⇒ → √ √*

π *∼* τ we have that *{*α, β1 *···* β1 *}* are the only actions π and τ can perform, and π *α*

τ *α* . Finally,

τ *α √*

*→−α c √*

1 *k*1

*→ ⇔ →*

*→ ⇔* τ *⇒* . *—→*

* + I. H.: suppose that the lemma holds for all n < k. Let α *c* be a sequence of length k. Let α*{*β1 *···* β1 *}* be

1

*k*1

the first action of the sequence and let *—→*γ *c* be a sequence of length k *—* 1 such that *—→*α *c* = α*{*β1 *···* β1 *}*.*—→*γ *c*.

*→−α c √*

*′ α ′*

*′ →−γ c √*

1 *k*1

*α ′*

Then, π *⇒* if and only if there is a process π such that π *→* π and π *⇒* . But if π *→* π and π *∼* τ,

then there is a process τ *′* such that τ *α* τ *′* and π*′ ∼* τ *′*. Moreover, as π *∼* τ, then *{*α, β1 *···* β1 *}* are the

*→ —→*

only actions that π and τ can perform. Now, γ *c* is a sequence of length shorter than k, so by the induction

1 *k*1

*′ ′ ′ →−γ c √*

*′ →−γ c √*

*→−α c √*

hypothesis, as π *∼* τ and π *⇒*

, then τ *⇒*

. This means that τ *⇒* . *2*

# Proof of lemma [4.4](#_bookmark13)

**Proof.** We prove by induction on the structure of π.

The base case is for *|* π *|*= 1, for atomic program it is straightforward. Suppose it holds for *|* π *|≤* n, so we have three possibilties.

* Suppose *M*,w H *⟨*π*∗⟩*ϕ, iff iff there is v *∈* W s. t. wR*π∗* v and *M*,v H ϕ. But we know that R*π∗* = R*∗* .

Then we get we have a path wR v R ...R v. As R*∗*

*π* 1

*π*

*π*

*π*

*π*

,v H ϕ, but this is

*π*

is transitive wR v and *M*

iff ,w H π ϕ. By the induction Hypothesis there is a finite path (v0, v1,..., v*n*), n 1, such that v0 = w, v*n* = v, *M*, v*n* H ϕ and there is *—→*α *c ∈ —R→c* (π) of length n such that *—→*α *c* matches the path (v0,..., v*n*). But by theorem [3.13](#_bookmark11) we know that *—R→c* (π) *⊆* (*—R→c* (π))*∗* = *—R→c* (π*∗*)

*f*

*M ⟨ ⟩ ≥*

*f*

*f*

*f*

The cases for π = π1 + π2 and π = π1; π2 are analogous to the previous case.

*•*

*2*

# Proof of lemma [4.8](#_bookmark16)

**Proof.**

( ) Suppose that, for some model = ( , **V**) and some state w in this model, ,w H π1 + π2 p. Then, by lemma [4.4](#_bookmark13), matches π1 +π2 at w, there is a finite path (v0, v1,..., v*n*), n 1, such that v0 = w, *M*, v*n* H p and a sequence *—→*α *c ∈ —R→c* (π1 + π2) that matches this path.

*f*

*F ≥*

*⇒ M F M ⟨ ⟩*

Now, by the third equality in theorem [3.13](#_bookmark11), either *—→*α *c ∈ —R→c* (π1) *|*+*π*2 or *—→*α *c ∈ —R→c* (π2) *|*+*π*1 , It follows

*f f*

directly from definition [3.12](#_bookmark10) that *—→*α *c |−π*2 *∈ —R→c* (π1) or *—→*α *c |−π*1 *∈ —R→c* (π2).

*f*

*f*

Besides, *—→*α *c |−π*2 and *—→*α *c |−π*1 match path (v0, v1,..., v*n*)., which implies that *M*,w H *⟨*π1*⟩*p or

*M*,w H *⟨*π2*⟩*p. Thus, *M*,w H *⟨*π1*⟩*p *∨ ⟨*π2*⟩*p (1). *—→*

As *F* matches π1 + π2 at w, so *F* matches π1 and π2 at w. Then, there exist α*—→*1*c ∈ Rc* (π1) that

*f*

matches a path (w0, v1,..., w*k*) and α*—→*2*c ∈ —R→c* (π2) that matches a path (u0, v1,..., u*l*) and w = w0 = u0.

*f*

Which implies that *M*,w H *⟨*π1*⟩T* and *M*,w H *⟨*π2*⟩T*. Thus, *M*,w H *⟨*π1*⟩T ∧ ⟨*π2*⟩T* (2).

From (1) and (2), we have

*M*,w H (*⟨*π1*⟩*p *∨ ⟨*π2*⟩*p) *∧* (*⟨*π1*⟩T ∧ ⟨*π2*⟩T*)

(*⇐*) Suppose *M*,w H *⟨*π1*⟩*p *∨ ⟨*π2*⟩*p (1) and *M*,w H *⟨*π1*⟩T ∧ ⟨*π2*⟩T* (2). From (2) we have that *F* matches π1 and π2 at w.

From (1) we have that *M*,w H *⟨*π1*⟩*p (3) or *M*,w H *⟨*π2*⟩*p (4).

(3) implies that matches π1 at w and there is a finite path (v0, v1,..., v*n*), n 1, such that v0 = w,

*F ≥*

*M*, v*n* H p and a sequence α*—→*1*c ∈ —R→c* (π1) that matches this path.

*f*

As α*—→*1*c ∈ —R→c* (π1), so α*—→*1*c |*+*π*2 *∈ —R→c* (π1) *|*+*π*2 *⊆ —R→c* (π1) *|*+*π*2 *∪—R→c* (π2) *|*+*π*1 = *—R→c* (π1 + π2) (Using

*f*

*f*

*f*

*f*

*f*

theorem [3.13](#_bookmark11)(3.)). Hence, α*—→*1*c |*+*π*2 *∈ —R→c* (π1 + π2).

*f*

From (2), we have that α*—→*1*c |*+*π*2 matches path (v0, v1,..., v*n*). Thus,

*M*,w H *⟨*π1 + π2*⟩*p.

Analogously, from (4) we also obtain *M*,w H *⟨*π1 + π2*⟩*p.

*2*

# Completeness Proof for PDL+

The canonical model construction is the standard one used for PDL [[5](#_bookmark21),[4](#_bookmark20),[16](#_bookmark24)].

**Definition D.1** (**Fischer and Ladner Closure**): Let Γ be a set of formulas. The **closure** of Γ, notation

C*FL*(Γ), is the smallest set of formulas satisfying the following conditions:

* 1. C*FL*(Γ) is closed under subformulas,
  2. if *⟨*π*∗⟩*ϕ *∈* C*FL*(Γ), then *⟨*π*⟩⟨*π*∗⟩*ϕ *∈* C*FL*(Γ),
  3. if *⟨*π1; π2*⟩*ϕ *∈* C*FL*(Γ), then *⟨*π1*⟩⟨*π2*⟩*ϕ *∈* C*FL*(Γ),
  4. if *⟨*π1 *∪* π2*⟩*ϕ *∈* C*FL*(Γ), then *⟨*π1*⟩*ϕ *∨ ⟨*π2*⟩*ϕ *∈* C*FL*(Γ),
  5. if *⟨*π1 *∪* π2*⟩*ϕ *∈* C*FL*(Γ), then *⟨*π1*⟩T* and *⟨*π2*⟩T ∈* C*FL*(Γ),
  6. if ϕ *∈* C*FL*(Γ) and ϕ is not of the form *¬*ψ, then *¬*ϕ *∈* C*FL*(Γ).

The proof that if Γ is a finite set of formulas, then the closure C*FL*(Γ) of Γ is also finite. We assume Γ to be finite from now on.

**Definition D.2** Let Γ be a set of formulas. A set of formulas is said to be an **atom of** Γ if it is a maximal consistent subset of C*FL*(Γ). The set of all atoms of Γ is denoted by At(Γ).

*A*

**Lemma D.3** *Let* Γ *be a set of formulas. If* ϕ *∈* C*FL*(Γ) *and* ϕ *is consistent then there exists an atom*

*A∈* At(Γ) *such that* ϕ *∈ A.*

**Proof.** We can construct the atom *A* as follows. First, we enumerate the elements oVf C*FL*(Γ)Vas φ1, *···* , φ*n*.

WVe start the construction making *A*1 = *{*ϕ*}*, then for 1 < i < n, we know that *▶ Ai ↔* ( *Ai ∧* φ*i*+1) *∨*

( *Ai ∧ ¬*φ*i*+1) is a tautology and therefore either *Ai ∧* φ*i*+1 or *Ai ∧ ¬*φ*i*+1 is consistent. We take *Ai*+1 as

the union of *Ai* with the consistent member of the previous disjunction. At the end, we make *A* = *An*.

*2*

**Definition D.4** Let Γ be a set of formulas. The **canonical relations over** Γ SΓ on At(Γ) are defined as

*π*

follows:

^ ^

*A*SΓ*B* iff *A∧ ⟨*π*⟩ B* is consistent.

*π*

**Definition D.5** Let Γ be a set of formulas. The **canonical model over** Γ is a tuple *M*Γ =< At(Γ),SΓ, **V**Γ >, where for all propositional symbols p and for all atoms *A∈* At(Γ) we have

*π*

* **V**Γ(p)= *{A ∈* At(Γ) *|* p *∈ A}* is called canonical valuation;
* SΓ and SΓ+ are the canonical relations. [3](#_bookmark33)

*π π*

**Lemma D.6** *Let A∈* At(Γ)*. Then, for all basic programs* α*,*

*⟨*α*⟩*ϕ *∈A iff there exists B ∈* At(Γ) *such that A*S*αB and* ϕ *∈ B.*

**Proof.**

V

*⇒*: Suppose *⟨*α*⟩*ϕ *∈ A*. By definition [D.2](#_bookmark29), we haVve that *A∧ ⟨*α*⟩*ϕ is consistent. VUsing the tautology

*▶* ϕ *↔* ((ϕ *∧* φ) *∨* (ϕ *∧ ¬*φ)), we have that either *A∧ ⟨*α*⟩*(ϕ *∧* φ) is consistent or *A∧ ⟨*α*⟩*(ϕ *∧ ¬*φ) is

consistent. So, bVy the appropriVate choice of φ, for all formulas φ *∈* C*FL*, we can construct an atom *B* such

that ϕ *∈B* and *A∧ ⟨*α*⟩*(ϕ *∧ B*) is consistent and by deVfinition [D](#_bookmark31)V[.4](#_bookmark31) *A*S*αB*. V

*⇐*: Suppose there is *B* such that ϕ *∈B* and *A*S*αB*. Then *A∧ ⟨*α*⟩ B* is consistent and also *A∧ ⟨*α*⟩*ϕ

is consistent. But *⟨*α*⟩*ϕ *∈* C*FL* and by maximality *⟨*α*⟩*ϕ *∈ A*.

*2*

**Lemma D.7** *Let A*, *B ∈* At(Γ)*. Then if A*S*π∗B then A*S*∗ B.*

*π*

**Proof.** Suppose *A*S*π∗B*. Let **C** = *{C ∈* At(Γ) *| A*S*π ∗C}*. We want to show that *B ∈* **C**. Let **C***∧* =

V

(V *C*1 *∨ ··· ∨ C* ).

*∨*

*n*

It is not difficult to see that **C***∧ ∧ ⟨*π*⟩¬***C***∧* is inconsistent, otherwise for some *D* not reachable from

*A*, **C***∧ ∧ ⟨*π*⟩* V *D*

*∨ ∨ C* , V *C*

*∧ ⟨*π*⟩* V *D* was also consistent, which would

*∨* would be consistent, and for some *i i* V *∧*

mean that *D ∈* **C**, whichVis not the case. From a similar reasoning we know that *A∧ ⟨*π*⟩¬***C***∨* is also

*∨*

inconsistent and hence *▶*

*A→* [π]**C***∧* is a theorem.

As **C***∧ ∧ ⟨*π*⟩¬***C***∧* is inconsistent, so its negation is a theorem *▶ ¬*(**C***∧ ∧ ⟨*π*⟩¬***C***∧*) and also *▶* (**C***∧ →*

[π]**C***∧*) (1), applying generalization *▶* [π*∗*](**C***∧ →* [π]**C***∧*). Using Segerberg axiom (axiom 6), we have

*∨*

*∨*

*∨*

*∨*

*∨*

*∨*

*▶ ∧ ∗ ∧*

*∨ ∧ ∨ ∧* V *∧*

V([π]**C***∨ →* [π ]**C***∨*) and by (1) wVe obtain *▶*V(**C***∨ →* [π*∗*]**C***∨*). As *▶* V*A →* [π]**C***∨* is a theorem, then

*▶ A→* [π*∗*]**C***∧*. By supposition, *A∧ ⟨*π*∗⟩ B* is consistent and so is *B∧* **C***∧*. Therefore, for at least

*∨*

one *C ∈*

*C*

V *B∧* V *C* is consistent. By maximality, we have that *B*

*∨*

. And by the definition

**C**, we know that =

*∧*

of **C***∨*, we have *A*S*π∗B*.

*2*

**Definition D.8** Let Γ be a set of formulas. The **PDL+ model over** Γ is a tuple *M* =< At(Γ), R*π* , **V** >, where for all propositional symbols p and for all atoms *A∈* At(Γ) we have

* **V**(p)= At(Γ) p ;

*{A ∈ | ∈ A}*

* R*α* = S*α*, for all basic programs α
* R*π* is inductively defined as in definition [4.1](#_bookmark12).

**Lemma D.9** S*π ⊆* R*π.*

**Proof.** Induction on the straucture of π.

Base case is straightforward as R*α* = S*α*, for basic programs α.

Suppose it holds for programs π such that *|* π *|≤* n. We only prove the case where π = π1 + π2. The

case for π = π1; π2 and π*∗* arVe standard in PDVL literature. V V

SVuppose *A*S*π*1 +*π*2 *B*, iff *A∧ ⟨*π1 + π2*⟩ B*) is consistent. By axiom 4. *A∧* ((*⟨*π1*⟩*

*B∧ ⟨*π2*⟩T*) *∨*

(*⟨*π2*⟩*V *B∧ ⟨*π1*⟩T*V)) is consistent. Either

*A ∧*V((*⟨*π1*⟩ B∧ ⟨*π2*⟩T*) is consistent (1) or

(*⟨*π2*⟩ B* V*∧ ⟨*π1*⟩T*)) is consistent (2).

From (1) *A∧* ((*⟨*π1*⟩* V *B*) is consistent (3) and

V *A∧* (*⟨*π2*⟩T*) is consistent (4)

From (3 and (4) we get *A*S*π*1 *B* and there exists an atom *C* s.t. *A*S*π*1 *C*. By the Induction Hypothesis

*A*R*π*1 *B* and there exists *C* s.t. *A*R*π*1 *C* (5).

3 For the sake of clarity we avoid using the Γ subscripts

Analogously, from (2) we can obtain

*A*R*π*1 *B* and there exists *C* s.t. *A*R*π*1 *C*, which together with (5) allows us to conclude *A*R*π*1 +*π*2 *B*.

*2*

**Lemma D.10 Exixtence Lemma***: Let A∈* At(Γ)*. Then,*

*⟨*π*⟩*ϕ *∈A iff there exists B ∈* At(Γ) *such that A*R*π B and* ϕ *∈ B.*

**Proof.**

: This direction follows is analogous to the one presented for basic programs in lemma [D.6](#_bookmark32) and the previous lemma that states that S*π* R*π* .

*⊆*

*⇒*

: Induction on the straucture of π.

*⇐*

Base case is straightforward from lemma [D.6](#_bookmark32), for basic programs α.

Suppose it holds for programs π such that π n. We only prove the case where π = π1 + π2. The case for π = π1; π2 and π*∗* are standard in PDL literature.

*| |≤*

Suppose *A*R*π*1 +*π*2 *B* (1) and ϕ *∈B* (2). That means that either

*A*R*π*1 *B* and there exists *C* s.t. *A*R*π*2 *C* (3) or

*A*R*π*2 *B* and there exists *C* s.t. *A*R*π*1 *C* (4)

From (2), (3 and (4) and the Induction Hypothesis we have that either

*⟨*π1*⟩*ϕ *∈A* and *⟨*π2*⟩T ∈ A* (5) or *⟨*π2*⟩*ϕ *∈A* and *⟨*π1*⟩T ∈ A* (6)

V

By (5) and (6) and axiom 4., we have that *A ∧ ⟨*π1 + π2*⟩*ϕ is consistent. And by maximality

*⟨*π1 + π2*⟩*ϕ *∈A*

*2*

**Lemma D.11 Truth Lemma***: Let M* = (W, S*π* , **V**) *be a finite canonical model for* φ*. For all atoms A*

*and all* ϕ *∈* C*FL*(φ)*, M*, *A |*= ϕ *iff* ϕ *∈ A.*

**Proof.** : The proof is by induction on the construction of ϕ.

* Atomic formulas and Boolean operators: the proof is straightforward from the definition of **V**.
* Modality *⟨*x*⟩*, for x *∈ {*α, π1; π2, π1 + π2, π*∗}*.
* *⇒*: Suppose *M*, *A |*= *⟨*x*⟩*ϕ, then there exists *A′* such that *A*S*xA′* and

*M*, *A′ |*= A. By the induction hypothesis we know that ϕ *∈ A′*, and by lemma [D.10](#_bookmark34) we have *⟨*x*⟩*ϕ *∈ A*.

* *⇐*: Suppose *M*, *A |*= *⟨*x*⟩*ϕ, by the definition of satisfaction we have

*M*, *A |*= *¬⟨*x*⟩*ϕ. Then for all *A′*, *A*S*xA′* implies *M*, *A′ /|*= ϕ. By the induction hypothesis we know that

ϕ */∈ A′*, and by lemma [D.10](#_bookmark34) we have *⟨*x*⟩*ϕ */∈ A*.

*2*

**Theorem D.12** *(***Completeness for Finite PDL+ Models***): Propositional Dynamic Logic PDL+ is complete with respect to the class of finite PDL+ models.*

**Proof.** For every consistent formula ϕ we can build a canonical finite PDL+ model **M***ϕ*. By lemma [D.3](#_bookmark30), there exist an atom At(ϕ) such that ϕ , and by the truth lemma [D.11](#_bookmark35) , = ϕ. Therefore, our modal system is complete with respect to the class of finite PDL+ models.

*A ∈ ∈ A M A |*

*2*