1 PROPOSITIONAL DYNAMIC LOGIC

1.1 The syntax and semantics of PDL

1.1.1. Transition systems

A *transition system* consists of a set S and a collection of binary relations over S. We refer to elements $s \in S$ as *states*, and to each of the system's binary relations as an *action*. Here are some examples that illustrate the underlying motivation.

EXAMPLES **1.1.1** Consider a supermarket. Let the "states" be the store's possible inventories, and the "actions" be events that change the inventory, such as deliveries, purchases, spoilage, and shop-lifting. Notice that certain actions are not allowable in all states: if the store is out of milk, then the purchase of a gallon of milk is not an allowable action. Also, the impact of an action may be ambiguous (non-deterministic). For example, suppose the store management defines "shoplifting" as any theft of up to five small items; starting with a given inventory, the "shoplifting" action has many possible "states" (inventories) as outcome. Also note that actions are discrete events, with a clear yes-or-no impact on states: we do not deal with continuous change over time.

1.1.2 Fix a Turing machine M. Let the states be the configurations of the machine. Let the actions be the machine's instructions. The operational semantics of Turing machines tells us how each "action" maps a given "state" to a new one. Note that each "action" can impact only those "states", i.e. configurations, whose internal-state corresponds to the one referred to in the "action". Also note that the resulting state is uniquely determined (even if M is non-deterministic!).

1.1.3 Consider all Turing machines, operating on a single tape (infinite to one side), over a fixed alphabet Σ. Let the "states" be the possible contents of the tape, i.e. the strings in Σ^*. Think of each machine M as an "action". An "action" M applied to a "state" $w \in \Sigma^*$ leads to state $w' \in \Sigma^*$ if the machine M, started on input w (in its initial internal-state, and scanning the first tape-cell), has a terminating computation that leaves w' on the tape. Note that if M has no termination computation on input w, then no "state" is obtained as an outcome. On the other hand, if M is a non-deterministic machine, then the "action" M might lead from "state" w to many outcome "states".

1.1.4 Let $V = (\mathbf{0}; \mathbf{s}, +, \times; <)$ be the vocabulary of arithmetic. Let A be the collection of "assignment statements", i.e. of formal expressions of the form $\mathbf{x} := \mathbf{t}$, where $\mathbf{x}$ is a variable and $\mathbf{t}$ a V-expression. Let P consist of the atomic V-formulas.

Let the "states" be the V-environments over $\mathbb{N}$, and the "actions" be the assignments. A state η is mapped by an action $x := \mathbf{t}$ to the state $[\eta(\mathbf{x})/\mathbf{x}]\,\eta$, that is the environment η' that results from η by updating the value of x to the value of $\mathbf{t}$ in η:

$$\eta'(z) =_{\mathrm{df}} \begin{cases} [\![\mathbf{t}]\!]\eta & \text{if } z \text{ is (literally) } x \\ \eta(z) & \text{otherwise} \end{cases}$$

Note that here each action can impact any state, and the outcome of each action is unique.

1.1.2. Defining compound actions

Let us define more precisely the notion of a transition system. Posit a countable (finite or infinite) set A of identifiers, the *atomic-action identifiers*. A *transition system (over A)* is a structure $(S, [\![\cdot]\!]_A)$, consisting of a set S and an assignment $[\![\cdot]\!]_A$ that maps each identifier $\mathbf{a} \in A$ a binary relation over S, that is $[\![\mathbf{a}]\!]_A \subseteq S^2$.

Simple methods for generating binary relations from given ones include composition, union, and reflexive-transitive closure. We use the semi-colon symbol (;) to denote composition, and the customary union symbol $\cup$ for union, and star for reflexive-transitive closure. Starting with atomic-action identifiers as *atomic programs*, we thus obtain names for a broad family of binary relations, which we dub *regular programs*. That is, the regular programs are generated inductively from the atomic programs by the following BNF definition, where we use $\mathbf{a}$ as syntactic parameter for atomic programs, and α and β as syntactic parameters for programs.

$$\begin{array}{rcl} A & \ni & \mathbf{a} \\ RProg(A) & \ni & \boldsymbol{\alpha}, \boldsymbol{\beta} \quad ::= \quad \mathbf{a} \mid \boldsymbol{\alpha};\boldsymbol{\beta} \mid \boldsymbol{\alpha} \cup \boldsymbol{\beta} \mid \boldsymbol{\alpha}^* \end{array}$$

Note that regular programs are simply regular expressions over the alphabet consisting of the atomic programs. (One insignificant variation here is the use of semi-colon, as in $\alpha;\beta$, for composition. Thus the phrase "regular programs" is an appropriate choice.

Given a transition system $(S, [\![\cdot]\!]_A)$, each regular program is interpreted semantically the way it is intended. That is, we extend the scope of the semantic interpretation function $[\![\cdot]\!]_A$, from to all regular programs, via the following recursive definition. For readability, we omit the subscript A,

$$\begin{array}{rcll} [\![\alpha;\beta]\!] & = & [\![\alpha]\!] \circ [\![\beta]\!] & \text{(composition of relations)} \\ [\![\alpha \cup \beta]\!] & = & [\![\alpha]\!] \cup [\![\beta]\!] & \text{(union of relations)} \\ [\![\alpha^*]\!] & = & [\![\alpha]\!]^* & \text{(reflexive-transitive closure)} \end{array}$$

Names for binary relations are usually used in infix notation rather than standard set-theory notation. For example, we write $x < y$ for numeric inequality rather than the more cumbersome $(x, y) \in <$. We similarly adopt an infix notation for actions, writing $s[\![\alpha]\!]_A t$ rather than $(s, t) \in [\![\alpha]\!]_A$. Better yet, we write $s \xrightarrow{\alpha} t$. If explicit reference to the transition system $\mathcal{S}$ under discussion is appropriate, we write more precisely $s \xrightarrow{\alpha} \mathcal{S} t$.

Using this notation, we can rephrase the definition of $[\![\cdot]\!]$ as follows:

$$\begin{array}{llllll}
s \xrightarrow{\alpha;\beta} t & \text{IFF} & s \xrightarrow{\alpha} t' & \text{and} & u \xrightarrow{\beta} t & \text{for some} \quad t' \\
s \xrightarrow{\alpha\cup\beta} t & \text{IFF} & s \xrightarrow{\alpha} t & \text{or} & s \xrightarrow{\beta} t & \\
s \xrightarrow{\alpha^*} t & \text{IFF} & \multicolumn{4}{l}{s \xrightarrow{\alpha} s_1 \xrightarrow{\alpha} s_2 \xrightarrow{\alpha} \cdots \xrightarrow{\alpha} s_k \xrightarrow{\alpha} t} \\
 & & \multicolumn{4}{l}{\qquad\qquad \text{some } k \geq 0 \text{ and } s_1 \ldots s_k \in S}
\end{array}$$

Note. One common method of generating binary relations was omitted in our discussion: the inverse operation. This can be added, but when we think of relations as actions, the inverse operation is not very compelling.

1.1.3. Propositional dynamic structures

Propositional logic focuses on the core logical structure of statements, abstracting away from the actual meaning of the basic statements. A *propositional structure* is therefore an assignment of a truth value to each one of the propositional identifiers, nothing more. Compound formulas are then built out of the propositional identifiers, and their truth within a given propositional structure is calculated by recurrence.

We take the same approach when reasoning about transition systems: we posit identifiers for basic assertions, and build compound assertions out of those. Just as for propositional logic, we posit a countable (finite or infinite) set P of *propositional identifiers*. However, here we think of each such identifier not as an assertion which is true or false in a fixed domain of discourse, but rather as being true or false depending on which *state* of the system is considered. This leads us to define a notion that combines transition systems with propositional structures.

DEFINITION 1 *A* propositional dynamic (PD) structure *(for the vocabulary P, A) is a structure $(S, [\![\cdot]\!]_A, [\![\cdot]\!]_P)$, where $(S, [\![\cdot]\!]_A)$ is a transition system for the action vocabulary A, and $[\![\cdot]\!]_P$ is a propositional structure for the propositional vocabulary P.*[1]

Since P and A are assumed disjoint, we will write $[\![\cdot]\!]$ for both $[\![\cdot]\!]_A$ and $[\![\cdot]\!]_P$, without danger of confusion.

It is often convenient to represent finite (and not too large...) PD structures as labeled directed multi-graphs. The nodes are labeled by those propositional identifiers that are true there, and an arc from node s to node t (labeled by all atomic-actions that lead from s to t) if such actions exist. For example

[INSERTED PS FILES]

1.1.4. PDL formulas

We now proceed to define inductively compound formulas, starting with the propositional identifiers. The intent is that each such formula is true or false depending

[1] that is a function from the set P of propositional-identifiers to the set $\{0, 1\}$ of truth-values.

on the state considered. For example, the formula $\varphi \wedge \neg\psi$ will be true in a state s exactly when φ is true in s and ψ is false in s. However, propositional formulas do not suffice to express properties of a state s that refer to other states, such as "φ is true in every state reached (from s) by atomic-program **a**." To render such properties, we introduce operators that generate a compound formula from a given program and a given formula. In particular, if α is a program, and φ is a formula, then we have a formula $[\boldsymbol{\alpha}]\boldsymbol{\varphi}$ (pronounced "box alpha phi"), intended to stand for "φ is true in every state reachable by program α."

We now define formally the *propositional dynamic formulas* as the expressions generated inductively by the following BNF definition (which we will slightly extend momentarily).

$$\begin{array}{rcll} P & \ni & \mathbf{p} & \\ \mathit{RPrg}(A) & \ni & \boldsymbol{\alpha} & \\ \mathit{PDFml}(P,A) & \ni & \boldsymbol{\varphi},\boldsymbol{\psi} & ::= \ \mathbf{p} \mid \neg\boldsymbol{\varphi} \mid \boldsymbol{\varphi}\wedge\boldsymbol{\psi} \mid \boldsymbol{\varphi}\vee\boldsymbol{\psi} \mid \boldsymbol{\varphi}\rightarrow\boldsymbol{\psi} \mid [\boldsymbol{\alpha}]\boldsymbol{\varphi} \mid \langle\boldsymbol{\alpha}\rangle\boldsymbol{\varphi} \end{array}$$

Given a PD structure $\mathcal{K}$ and a state s therein, the truth of a formula $\boldsymbol{\varphi}$ at s is defined by recurrence on $\boldsymbol{\varphi}$, as follows. We write $\mathcal{K}, s \models \boldsymbol{\varphi}$ for "$\boldsymbol{\varphi}$ is true in $\mathcal{K}$ at s." We abbreviate $\mathcal{K}, s \models \boldsymbol{\varphi}$ to $s \models \boldsymbol{\varphi}$, when in no danger of confusion.

$$\begin{array}{rcl} s \models \mathbf{p} & \text{iff} & [\![\mathbf{p}]\!](s) = 1 \\ s \models \neg\boldsymbol{\varphi} & \text{iff} & s \not\models \boldsymbol{\varphi} \\ s \models \boldsymbol{\varphi}\wedge\boldsymbol{\psi} & \text{iff} & s \models \boldsymbol{\varphi} \ \text{ and } \ s \models \boldsymbol{\psi} \\ s \models \boldsymbol{\varphi}\vee\boldsymbol{\psi} & \text{iff} & s \models \boldsymbol{\varphi} \ \text{ or } \ s \models \boldsymbol{\psi} \\ s \models \boldsymbol{\varphi}\rightarrow\boldsymbol{\psi} & \text{iff} & s \models \boldsymbol{\varphi} \ \text{ implies } \ s \models \boldsymbol{\psi} \\ s \models [\boldsymbol{\alpha}]\boldsymbol{\varphi} & \text{iff} & t \models \boldsymbol{\varphi} \ \text{ for all } \ t \ \text{ such that } \ s \xrightarrow{\alpha} t \\ s \models \langle\boldsymbol{\alpha}\rangle\boldsymbol{\varphi} & \text{iff} & t \models \boldsymbol{\varphi} \ \text{ for some } \ t \ \text{ such that } \ s \xrightarrow{\alpha} t \end{array}$$

Clearly, $s \models \neg\langle\boldsymbol{\alpha}\rangle\boldsymbol{\varphi}$ iff $s \models [\boldsymbol{\alpha}]\neg\boldsymbol{\varphi}$, so $\langle\boldsymbol{\alpha}\rangle$ can be viewed as an abbreviation for $\neg[\boldsymbol{\alpha}]\neg$, much as $\exists$ can be as $\neg\forall\neg$.

Recall the truth notions for first order formulas: $\mathcal{S},\eta \models \boldsymbol{\varphi}$, $\mathcal{S} \models \boldsymbol{\varphi}$, and $\models \boldsymbol{\varphi}$. Similarly, we write $\mathcal{K} \models \varphi$ if $\mathcal{K}, s \models \boldsymbol{\varphi}$ for all state s in $\mathcal{K}$, and we write $\models \boldsymbol{\varphi}$ (and say that $\boldsymbol{\varphi}$ is *valid*) if $\mathcal{K} \models \varphi$ for all transition systems $\mathcal{K}$.

1.1.5. Programs with tests

The box and diamond operators inject programs into the syntax of formulas, allowing the formal discourse (formulas) to refer to programs. Without that ability, we would not be generalizing propositional logic one bit. However, it is also natural to inject formulas into programs, and permit programs to query about the truth of formulas in a state, or at least of propositional identifiers. Indeed, such ability is at the core of even the simplest imperative programming languages (including micro-code), in the guise of branching commands. We thus extend the definition of programs, and say that $?\varphi$ is a program whenever $\varphi$is a formula. That is, we revise the BNF definitions of programs and formulas:

$$\begin{array}{rcll} A & \ni & \mathbf{a} & \\ RProg(A) & \ni & \boldsymbol{\alpha},\boldsymbol{\beta} & ::= \ \mathbf{a} \mid \boldsymbol{\alpha};\boldsymbol{\beta} \mid \boldsymbol{\alpha}\cup\boldsymbol{\beta} \mid \boldsymbol{\alpha}^* \mid ?\varphi \\ P & \ni & \mathbf{p} & \\ PDFml & \ni & \boldsymbol{\varphi},\boldsymbol{\psi} & ::= \ \mathbf{p} \mid \neg\boldsymbol{\varphi} \mid \boldsymbol{\varphi}\wedge\boldsymbol{\psi} \mid \boldsymbol{\varphi}\vee\boldsymbol{\psi} \mid \boldsymbol{\varphi}\rightarrow\boldsymbol{\psi} \mid [\boldsymbol{\alpha}]\boldsymbol{\varphi} \mid \langle\boldsymbol{\alpha}\rangle\boldsymbol{\varphi} \end{array}$$

Note that we now generate programs and formulas in tandem: in a formula $[\alpha]\varphi$ the program α may refer to a formula, and in a program $?\varphi$ the formula φ may refer to programs.

As a result of this extension, we need one additional clause in the definition of the semantics of programs and formulas:

$$[\![?\varphi]\!] = \{(s,s) \mid s \models \varphi\}$$

Put differently, $s\xrightarrow{?\varphi}s'$ iff $s \models \varphi$ and $s' = s$.

Allowing tests for arbitrary formulas is not admitted into mainstream programming languages, for the simple reasons that it cannot be implemented (as we shall see directly). However, it turns out that in the context of propositional dynamic logic our results and proof go through virtually unchanged if we admit tests over arbitrary formulas, as opposed to restricted tests, say over propositional identifiers only.

1.1.6. The expressive power of formulas

The formulas of PDL can express many interesting properties of programs. For example, $[\boldsymbol{\alpha}]\bot$ holds in a state s iff every execution of $\boldsymbol{\alpha}$ leads from s to a state where $\bot$ is true, i.e. exactly if there is no (terminating) execution of $\boldsymbol{\alpha}$ starting in state s.

EXERCISES **1.1.5** For each one of the properties below write a formula that expresses it, i.e. such that the formula is true at a state s iff the given statement is true at that state.

1. "There is a (terminating) execution of $\boldsymbol{\alpha}$ starting at the current state."
2. " Deterministic program $\boldsymbol{\alpha}$ can be iterated any number of times."
3. There is some $n \geq 0$ such that iterating deterministic program $\boldsymbol{\alpha}$ n times leads to a state where $\boldsymbol{\beta}$ has no terminating execution.
4. By iterating execution of α we can reach a state where $\mathbf{p}$ is true.
5. $\mathbf{q}$ is true in every state that is reached from a state where $\mathbf{p}$ is true by executing α.
6. $\mathbf{p}$ is true after iterated execution of α an even number of times.

7. $\mathbf{p}$ is true after iterated execution of α an odd number of times.

Note that PDL formulas do not express all possible properties of PD structures. For example, consider the properties

1. $\mathbf{p}$ is true after every iterated execution of $\mathbf{a}$ a certain number of times, followed by an iteration of $\mathbf{b}$ the same number of times.

2. $\mathbf{p}$ is true after iterated execution of α a prime number of times.

The reader familiar with formla language theory will recognize here standard examples of languages $L \subseteq \{\mathbf{a}, \mathbf{b}\}^*$ that are not regular. Theorem **??** below shows that the property above cannot be expressed by any PD formula.

1.1.7. Understanding formulas

We have just considered how properties of PD structures can be expressed by PDL formulas. It is also of interest to start with a formula φ, and ask whether it expresses a non-trivial property, that is whether φ is sometimes true and sometimes not.

Consider, for example, the formula $[\mathbf{a}]\mathbf{p} \rightarrow \langle\mathbf{a}\rangle\mathbf{p}$. This formula is false in the structure

[INSERTED PS FILES]

and true in

[INSERTED PS FILES]

EXERCISES **1.1.6** Show that the following PDL formulas are non-trivial. Try to give examples that are as simple as possible.

1. $\langle\mathbf{a}\rangle\mathbf{p} \rightarrow [\mathbf{a}]\mathbf{p}$

2. $[\mathbf{a}]\mathbf{p} \rightarrow [\mathbf{a}^*]\mathbf{p}$

3. $[\mathbf{a}](\mathbf{p} \vee \mathbf{q}) \rightarrow ([\mathbf{a}]\mathbf{p} \vee [\mathbf{a}]\mathbf{q})$

4. $\langle\mathbf{a}\rangle\mathbf{p} \wedge \langle\mathbf{a}\rangle\neg\mathbf{p}$

1.1.8. Guarded iterative programs

The nondeterministic branching and iteration constructs, $\cup$ and $*$, are alien to deterministic imperative programming languages, such as BASIC, FORTRAN or PASCAL. Such languages use deterministic variants of the branching and iteration commands, following the following paradigms:

- *Guarded branching:* `if` $\boldsymbol{\chi}$ `then` $\boldsymbol{\alpha}$ `else` $\boldsymbol{\beta}$.

- *Guarded iteration:* `while` $\boldsymbol{\chi}$ `do` $\boldsymbol{\alpha}$.

If A is a set of action identifiers, and P a set of propositional identifiers, then the set $GIP(A,P)$ of guarded iterative programs over A and P is generated by:

$$\begin{array}{rcll} A & \ni & \mathbf{a} & \\ P & \ni & \mathbf{p} & \\ GIP(A,P) & \ni & \boldsymbol{\alpha},\boldsymbol{\beta} & ::= \ \mathbf{a} \mid \mathtt{skip} \mid \mathtt{abort} \mid \boldsymbol{\alpha};\boldsymbol{\beta} \mid \ \mathtt{if}\ \mathbf{p}\ \mathtt{then}\ \boldsymbol{\alpha}\ \mathtt{else}\ \boldsymbol{\beta} \mid \ \mathtt{while}\ \mathbf{p}\ \mathtt{do}\ \boldsymbol{\alpha} \end{array}$$

The argument $\mathbf{p}$ in $\mathtt{if}\ \mathbf{p}\ \mathtt{then}\ \cdots$ and in $\mathtt{while}\ \mathbf{p}\ \mathtt{do}\ \cdots$ is called the *guard* of the command. We use the expression $\mathtt{if}\ \mathbf{p}\ \mathtt{then}\ \boldsymbol{\alpha}$ as an abbreviation for $\mathtt{if}\ \mathbf{p}\ \mathtt{then}\ \boldsymbol{\alpha}\ \mathtt{else}\ \mathtt{skip}$.

The semantics of guarded iterative programs over a transition system $\mathcal{K} = (S, [\![\cdot]\!])$ is defined by recurrence over programs:

- For $\mathbf{a} \in A$ the relation $\xrightarrow{\mathbf{a}}$ is given by $\mathcal{K}$;
- $\xrightarrow{\textbf{skip}} =_S$;
- $\xrightarrow{\textbf{abort}} = \emptyset$;
- $s \xrightarrow{\alpha;\beta} t =_{\mathrm{df}} s \xrightarrow{\alpha} u$ and $u \xrightarrow{\beta} t$, for some state u;
- for a branching program $\boldsymbol{\gamma}$ of the form $\mathtt{if}\ \boldsymbol{\chi}\ \mathtt{then}\ \boldsymbol{\alpha}\ \mathtt{else}\ \boldsymbol{\beta}$, $s \xrightarrow{\gamma} t$ iff $s \models \boldsymbol{\chi}$ and $s \xrightarrow{\alpha} t$, or $s \not\models \boldsymbol{\chi}$ and $s \xrightarrow{\beta} t$;
- for a looping program $\boldsymbol{\gamma}$ of the form $\mathtt{while}\ \boldsymbol{\chi}\ \mathtt{do}\ \boldsymbol{\alpha}$,
  $s \xrightarrow{\mathit{while}} t$ iff there are some $u_1 \ldots u_k \in S$ $(k \geq 1)$, such that
  $s = u_1$, $u_i \xrightarrow{\alpha} u_{i+1}$ and $u_i \models \boldsymbol{\chi}$ for $i < k$, and $u_k = t$.

EXERCISES **1.1.7** Show that we may define $\mathtt{skip}$ by $?\top$. That is, for any transition system $\mathcal{K}$, the relation defined by these two programs in $\mathcal{K}$ are identical.

1.1.8 Give a definition of $\mathtt{abort}$ by a regular program.

1.1.9 Show that we may define $\mathtt{if}\ \boldsymbol{\chi}\ \mathtt{then}\ \boldsymbol{\alpha}\ \mathtt{else}\ \boldsymbol{\beta}$ as an abbreviation for the regular program $(?\boldsymbol{\chi};\boldsymbol{\alpha}) \cup (?\neg\boldsymbol{\chi};\boldsymbol{\beta})$.

1.1.10 Give a definition for $\mathtt{while}\ \mathbf{p}\ \mathtt{do}\ \boldsymbol{\alpha}$ by a regular program.

1.1.11 Can the program $\mathbf{a}^*$ be defined by a guarded iterative program? Explain.

1.2 A deductive calculus for PDL

1.2.1. An axiomatization of PDL for regular programs

Fix a formalization of quantifier-free logic, say the ND rules for the propositional connectives, in the proof-line style. We augment these rules with the following axiom-templates and inference rules. Each axiom-template is a mold, that becomes an actual axiom for any choice of formulas for the letters φ and ψ used to state the template.

I Axiom-templates and rules for programs in general:

Box Distribution: $\Rightarrow\ [\alpha](\varphi\to\psi) \to ([\alpha]\varphi\to[\alpha]\psi)$.

Generalization: from $\Rightarrow\ \varphi$ (with no assumption in the proof line) derive $\Rightarrow\ [\alpha]\varphi$.

II Axiom-templates and rules for program constructs:

Composition: $\Rightarrow\ [\alpha;\beta]\varphi \leftrightarrow [\alpha][\beta]\varphi$

Branching: $\Rightarrow\ [\alpha\cup\beta]\varphi \leftrightarrow [\alpha]\varphi \wedge [\beta]\varphi$

Schema: $\Rightarrow\ [\alpha^*]\varphi \leftrightarrow \varphi \wedge [\alpha][\alpha^*]\varphi$

Induction Rule: from $\Rightarrow\ \varphi\to[\alpha]\varphi$ derive $\Rightarrow\ \varphi\to[\alpha^*]\varphi$

Test: $\Rightarrow\ [?\chi]\varphi \leftrightarrow (\chi\to\varphi)$

1.2.2. PDL in natural deduction style

1.2.3. The Soundness Theorem

THEOREM **1.2.1** [Soundness of PDL] *Every provable formula provable in PDL is valid: if* $\vdash_{PDL}\varphi$ *then* $\models\varphi$.

1.2.4. Examples of proofs and derived rules

We describe the construction of formal proofs for PDL formulas and for derived rules.

1. *Box Distribution Rule:* From $\Rightarrow\ \varphi\to\psi$ derive $\Rightarrow\ [\alpha]\varphi\to[\alpha]\psi$:

   From $\Rightarrow\ \varphi\to\psi$ we get $\Rightarrow\ [\alpha](\varphi\to\psi)$, by the Generalization Rule. But $\Rightarrow\ [\alpha](\varphi\to\psi) \to ([\alpha]\varphi\to[\alpha]\psi)$ is an instance of the Box Distribution Axiom, so $\Rightarrow\ [\alpha]\varphi\to[\alpha]\psi$ follows by $\to$E.

2. *Conjunction Distribution Template:* $[\boldsymbol{\alpha}](\boldsymbol{\varphi} \wedge \boldsymbol{\psi}) \leftrightarrow ([\boldsymbol{\alpha}]\boldsymbol{\varphi} \wedge [\boldsymbol{\alpha}]\boldsymbol{\psi})$.

   FORWARD IMPLICATION: $\vdash \boldsymbol{\varphi} \wedge \boldsymbol{\psi} \rightarrow \boldsymbol{\varphi}$, and so $\vdash [\boldsymbol{\alpha}](\boldsymbol{\varphi} \wedge \boldsymbol{\psi}) \rightarrow [\boldsymbol{\alpha}]\boldsymbol{\varphi}$, by the Box Distribution Rule. Similarly, $\vdash [\boldsymbol{\alpha}](\boldsymbol{\varphi} \wedge \boldsymbol{\psi}) \rightarrow [\boldsymbol{\alpha}]\boldsymbol{\psi}$, so $\vdash [\boldsymbol{\alpha}](\boldsymbol{\varphi} \wedge \boldsymbol{\psi}) \rightarrow ([\boldsymbol{\alpha}]\boldsymbol{\varphi} \wedge [\boldsymbol{\alpha}]\boldsymbol{\psi})$.

   BACKWARD IMPLICATION: We have $\vdash \boldsymbol{\varphi} \rightarrow (\boldsymbol{\psi} \rightarrow \boldsymbol{\varphi} \wedge \boldsymbol{\psi})$, and so $\vdash [\boldsymbol{\alpha}]\boldsymbol{\varphi} \rightarrow [\boldsymbol{\alpha}](\boldsymbol{\psi} \rightarrow \boldsymbol{\varphi} \wedge \boldsymbol{\psi})$, by the Box Distribution Rule. But $\vdash [\boldsymbol{\alpha}](\boldsymbol{\psi} \rightarrow \boldsymbol{\varphi} \wedge \boldsymbol{\psi}) \rightarrow ([\boldsymbol{\alpha}]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}](\boldsymbol{\varphi} \wedge \boldsymbol{\psi}))$, by the Box Distribution Template, and so $\vdash [\boldsymbol{\alpha}]\boldsymbol{\varphi} \rightarrow ([\boldsymbol{\alpha}]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}](\boldsymbol{\varphi} \wedge \boldsymbol{\psi}))$, from which $\vdash ([\boldsymbol{\alpha}]\boldsymbol{\varphi} \wedge [\boldsymbol{\alpha}]\boldsymbol{\psi}) \rightarrow [\boldsymbol{\alpha}](\boldsymbol{\varphi} \wedge \boldsymbol{\psi})$ follows.

3. $[\boldsymbol{\alpha}^*]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}]\boldsymbol{\psi}$:

   $[\boldsymbol{\alpha}^*]\boldsymbol{\psi} \rightarrow \boldsymbol{\psi}$ by the Iteration Schema, and so $[\boldsymbol{\alpha}][\boldsymbol{\alpha}^*]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}]\boldsymbol{\psi}$ by Box Distribution. Also, $[\boldsymbol{\alpha}^*]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}][\boldsymbol{\alpha}^*]\boldsymbol{\psi}$ by the Iteration Schema, which together with the previous formula yields $[\boldsymbol{\alpha}^*]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}]\boldsymbol{\psi}$.

4. $[\boldsymbol{\alpha}^*]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}^*][\boldsymbol{\alpha}^*]\boldsymbol{\psi}$:

   Let $\boldsymbol{\varphi}$ be $[\boldsymbol{\alpha}^*]\boldsymbol{\psi}$. Then $\boldsymbol{\varphi} \rightarrow [\boldsymbol{\alpha}]\boldsymbol{\varphi}$, by the Iteration Schema, and so $\boldsymbol{\varphi} \rightarrow [\boldsymbol{\alpha}^*]\boldsymbol{\varphi}$, by the Induction Rule.

5. $[\boldsymbol{\alpha}^*]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}^*][\boldsymbol{\alpha}]\boldsymbol{\psi}$:

   We have $[\boldsymbol{\alpha}^*]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}]\boldsymbol{\psi}$ by example (3), and so $[\boldsymbol{\alpha}^*][\boldsymbol{\alpha}^*]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}^*][\boldsymbol{\alpha}]\boldsymbol{\psi}$ by Box Distribution. Combining this with example (4) we obtain $[\boldsymbol{\alpha}^*]\boldsymbol{\psi} \rightarrow [\boldsymbol{\alpha}^*][\boldsymbol{\alpha}]\boldsymbol{\psi}$.

EXERCISE **1.2.1** Show that the following formulas are provable in PDL.

1. $[?[\boldsymbol{\alpha}]\bot; \boldsymbol{\alpha}]\boldsymbol{\varphi}$
2. $[\boldsymbol{\alpha}]\mathbf{p} \wedge \langle\boldsymbol{\alpha}\rangle\mathbf{q} \rightarrow \langle\boldsymbol{\alpha}\rangle\mathbf{p}$
3. $[\boldsymbol{\alpha}]\mathbf{p} \wedge \langle\boldsymbol{\alpha} \cup \boldsymbol{\beta}\rangle\neg\mathbf{p} \rightarrow \langle\boldsymbol{\beta}\rangle\top$
4. $[(\boldsymbol{\alpha} \cup \boldsymbol{\beta})^*]\varphi \rightarrow [\boldsymbol{\alpha}^*][\boldsymbol{\beta}^*]\varphi$
5. $[(\boldsymbol{\alpha} \cup \boldsymbol{\beta})^*]\boldsymbol{\varphi} \rightarrow [\boldsymbol{\alpha}][\boldsymbol{\beta}^*]\boldsymbol{\varphi}$

1.2.5. An axiomatization of PDL for guarded iterative programs

The axiomatization of PDL for guarded iterative programs is the same as for PDL for regular programs, except that the axiom-templates and rules for $\cup$ and * are replaced by rules for guarded branching and guarded iteration, and rules are added for the `skip` and `abort` commands.

Skip:	$\Rightarrow\ [\mathtt{skip}]\boldsymbol{\varphi} \leftrightarrow \boldsymbol{\varphi}$
Abort	$\Rightarrow\ [\mathtt{abort}]\boldsymbol{\varphi}$
Guarded Branching:	$\Rightarrow\ [\mathtt{if}\ \boldsymbol{\chi}\ \mathtt{then}\ \boldsymbol{\alpha}\ \mathtt{else}\ \boldsymbol{\beta}]\boldsymbol{\varphi}\ \leftrightarrow\ (\boldsymbol{\chi} \wedge [\boldsymbol{\alpha}]\boldsymbol{\varphi}) \vee (\neg\boldsymbol{\chi} \wedge [\boldsymbol{\beta}]\boldsymbol{\varphi})$
Guarded Iteration:	$\Rightarrow\ [\mathit{while}]\boldsymbol{\varphi}\ \leftrightarrow\ (\neg\boldsymbol{\chi} \wedge \boldsymbol{\varphi}) \vee (\boldsymbol{\chi} \wedge [\boldsymbol{\alpha}][\mathit{while}]\,\boldsymbol{\varphi})$, where *while* abbrreviates $\mathtt{while}\ \boldsymbol{\chi}\ \mathtt{do}\ \boldsymbol{\alpha}$;
Guarded Induction Rule:	from $\Rightarrow\ (\boldsymbol{\chi} \wedge \boldsymbol{\varphi}) \to [\boldsymbol{\alpha}]\boldsymbol{\varphi}$ derive $\Rightarrow\ \boldsymbol{\varphi} \to [\mathit{while}](\boldsymbol{\varphi} \wedge \neg\boldsymbol{\chi})$.

1.3 The Completeness Theorem for PDL

1.3.1. Completeness and the Finite Model Property

THEOREM **1.3.1** [Completeness] *If $\boldsymbol{\varphi}$ is a PDL formula, and $\not\vdash_{PDL} \boldsymbol{\varphi}$, then there is a PD structure $\mathcal{K} = (S, [\![\cdot]\!])$ and some $s_0 \in S$ such that $\mathcal{K}, s_0 \not\models \boldsymbol{\varphi}$.*

Moreover, S is finite, with $\leq 2^{2^n}$ states, where n is the size of $\boldsymbol{\varphi}$.

We shall prove the Completeness Theorem for PDL momentarily. (Knowledge of the proof is not essential for the sequel.) An equivalent formulation of the Completeness Theorem is this:

COROLLARY **1.3.2** *If a PDL formula $\boldsymbol{\varphi}$ fails in some PD structure then it fails in some finite PD structure.*

Proof. If $\boldsymbol{\varphi}$ fails in some PD structure, then, by the Soundness Theorem, it is not provable. Therefore, by the Completeness Theorem, it fails in some finite PD structure. $\dashv$

The statement of the Corollary is sometimes refered to as the *Finite Model property*, i.e. the Corollary states that PDL has the Finite Model property.

An important cosequence of the Finite Model property is the decidability of PDL:

THEOREM **1.3.3** *There is an effective procedure which decides, given a PDL formula $\boldsymbol{\varphi}$, whether it is valid or not.*

Proof. Given a PDL formula $\boldsymbol{\varphi}$, the procedue considers some canonical enumeration $\pi_1\pi_2\ldots$ of all PDL proofs, and a canonical enumeration $\mathcal{K}_1\mathcal{K}_2\ldots$ of all finite PD structures. The procedure repeats for $i = 1, 2, \ldots$ the following: Check if π_i is a proof of $\boldsymbol{\varphi}$. If it is, declare $\boldsymbol{\varphi}$ valid and terminate. Else, check if $\boldsymbol{\varphi}$ fails at some state in $\mathcal{K}_i$; if it does, declare $\boldsymbol{\varphi}$ non-valid. Else proceed to $i + 1$.

If φ is valid, then it is provable, by the Completeness Theorem, and therefore it is eventually declared valid by the procedure above (since every proof is examined).

If φ is non-valid, then it is unprovable, by the Soundness Theorem. Thus, by the Completeness Theorem, φ is false at some state in some *finite* transition system. Thus, φ is eventually declared non-valid by the procedure, since every finite transition system is examined. The procedure cannot not declared φ valid at an earlier stage, by the Soundness Theorem. $\dashv$

1.3.2. Motivating example: a proof of the completeness of propositional logic

To motivate our proof method for Theorem **??**, we give a proof, using the same underlying idea, for the Completness Theorem for propositional logic.

Given a list $\vec{\chi} = \chi_1 \ldots \chi_k$ of formulas, a *boolean combination* over $\vec{\chi}$ is a list $\xi_1 \ldots \xi_k$, where each ξ_i is either χ_i or $\neg\chi_i$. We sometimes refer to a boolean combination as a single formula, namely the conjunction of all k formulas therein.

LEMMA **2** *Let S be the set consisting of all booelan combinations over $\vec{\chi}$. Then $\vdash \bigvee_{s \in S} s$.*

The proof is by induction on k.

THEOREM **3** *The ND system for Propositional Logic is complete: for each propositional formula φ, if $\models \psi$ then $\vdash \psi$.*

Proof. First, observe that if φ is any formula with all propopsitional identifiers among $\mathbf{p}_1 \ldots \mathbf{p}_k$, and s is a boolean combination of $\mathbf{p}_1 \ldots \mathbf{p}_k$, then

$$\begin{aligned} s \models \varphi \quad & \text{IMPLIES} s \vdash \varphi \\ s \not\models \varphi \quad & \text{IMPLIES} s, \varphi \vdash \bot \end{aligned}$$

Here we write s also for the propositional structure (i.e. truth-value assignemnt) which assigns to $\mathbf{p}_i$ the value 1 iff $\mathbf{p}_i$ appears non-negated in C. This claim is proved by induction on φ.

Now let φ be a propositional formula, over propositional identifiers $\mathbf{p}_1 \ldots \mathbf{p}_k$. Suppose that $\not\vdash \varphi$. If S is the set of boolean combinations over $\mathbf{p}_1 \ldots \mathbf{p}_k$, then $\vdash \bigvee_{s \in S} s$, as observed above. So $\bigvee_{s \in S} \not\vdash \varphi$. But then, for some $s \in S$, $s \not\vdash \varphi$. But then $s \not\models \varphi$. $\dashv$

EXERCISES **1.3.1** Give a proof of Lemma 2.

1.3.2 Give a proof of the claim in the proof above of Theorem 3.

1.3.3. The Fisher-Ladner closure

To prove Theorem 1.3.1, we assume $\not\vdash \varphi$, and construct a PD structure $\mathcal{K}$, such that $\mathcal{K}, s_0 \not\models \varphi$ for some state s_0 of $\mathcal{K}$. The idea of the proof is that the pertinent information about the truth of φ in various states of a structure is encapsulated in a finite set of formulas, just as the truth of a propositional formula in a propositional structure is encapsulated in the truth value assigned to the propositional identifiers. Here, however, the description of a model of φ cannot make do with the propositional identifiers, because PDL formulas are generated not only from propositional identifiers but also from programs.

We start by generating from φ a finite set of formulas F_φ, which intuitively includes all the information about the states needed to determine whether φ φ is true or not for any state in a structure. It may surprise at first that a finite set suffices, since the meaning of a formula $[\alpha^*]\psi$ is described in terms of α^i for all i. However, the truth of $[\alpha^*]\psi$ at a state s depends only on the truth of ψ at s and the truth of $[\alpha^*]\psi$ in α-neighbors of s, i.e. the truth of $[\alpha][\alpha^*]\psi$ at s. That is, we can describe the truth of $[\alpha^*]\psi$ at s in terms of the truth of the three formulas ψ, $[\alpha^*]\psi$ and $[\alpha^*]\psi$ in *all* states.

The *Fishher-Ladner productions* are the following rules for generating PD formulas from given ones.

- From $(\neg\psi)$ obtain ψ.
- From $(\psi \wedge \chi)$ obtain ψ, χ. Similarly for other binary propositional conenctives, if used.
- From $[\alpha]\psi$ obtain ψ.
- From $[\alpha;\beta]\psi$ obtain $[\alpha][\beta]\varphi$.
- From $[\alpha \cup \beta]\psi$ obtain $[\alpha]\varphi$ and $[\beta]\varphi$.
- From $[\alpha^*]\psi$ obtain $[\alpha][\alpha^*]\psi$.
- From $[?\chi]\psi$ obtain χ.

We write $\varphi \Longrightarrow_{FL} \psi$ if ψ is obtained from φ by one ofhte rules above. The set $C(\varphi)$, called the *Fisher-Ladner closure of* φ, is the relextive and transitive closure of $\{\varphi\}$ under $\Longrightarrow_{FL}$, that is,

$$C(\varphi) =_{\text{df}} \{\psi \mid \varphi \Longrightarrow^*_{FL} \psi\}$$

EXAMPLE **1.3.3** If

$$\varphi \quad \equiv \quad [(a \cup b)^*]((\neg p) \wedge [a;b][?r]q)$$

then

$$\begin{array}{rcll} C(\varphi) & = & \{ & \varphi,\ \ [a \cup b]\varphi,\ \ [a]\varphi,\ \ [b]\varphi, \\ & & & \neg p \wedge [a;b][?[a]r]q,\ \ \neg p,\ \ [a;b][?[a]r]q, \\ & & & p,\ \ [a][b][?[a]r]q,\ \ [b][?[a]r]q,\ \ [?[a]r]q, \\ & & & [a]r,\ \ r,\ \ q \quad \} \end{array}$$

EXERCISES

1.3.4 Calculate C_φ for the following formulas φ.

1. $(p \wedge \neg q) \rightarrow [a;b](\neg p \wedge q)$
2. $\langle (a;b)^* \rangle p$ (first convert the box to a diamond).
3. $[(?p;a)^*(?\neg p)]q$

1.3.4. The size of the closure

It is not obvious that $C(\varphi)$ must be finite, because the closure conditions require that for a formula $[\alpha^*]\psi$ in C_φ we place in $C(\varphi)$ a larger formula. Even if we believe that $C(\varphi)$ must be finite, we might expect an exponential number of formulas, because some of the closure conditions place two formulas in $C(\varphi)$.

However, it turns out that $C(\varphi)$ has at most as many formulas as the length of φ. To see this we shall first show that the closure of a formula $[\alpha]\varphi$ can be factored into productions that deal with α alone and ones that deal with φ alone. To capture this intent formally, we shall refer to a propositional identifier $\bullet$, which we use as a "placeholder" for the scope φ in $[\alpha]\varphi$. If ψ and χ are PDL formulas, we write $\psi \triangleleft \chi$ for the result of substituting χ simultaneously for all occurrence of $\bullet$ in ψ. If Ψ is a set of PDL formulas, then $\Psi \triangleleft \chi =_{\rm df} \{\psi \triangleleft \chi \mid \psi \in \Psi\}$.

LEMMA **1.3.4** *If* $[\alpha^*]\varphi \Longrightarrow^n_{FL} \psi$, *then one of the following holds.*

1. ψ *is* $[\alpha^*]\varphi$
2. ψ *is of the form* $B[\alpha^*]\varphi$ *(B a block of boxes), and* $[\alpha]\bullet \Longrightarrow^*_{FL} B\bullet$
3. $\varphi \Longrightarrow^*_{FL} \psi$.

Proof. The proof is by induction on n, and is left as an exercise. ⊣

COROLLARY **1.3.5** $C([\alpha^*]\varphi) = \{[\alpha^*]\varphi\} \cup C([\alpha]\bullet) \triangleleft ([\alpha^*]\varphi) \cup C(\varphi)$

LEMMA **1.3.6** $C([\alpha]\varphi) = C([\alpha]\bullet) \triangleleft \varphi \cup C(\varphi)$

Proof. By induction on α.

1. $C([a]\varphi)$

$$\begin{array}{lll} = & \{[a]\varphi\} \cup C(\varphi) & \text{by the definition of } \Longrightarrow_{FL} \\ = & \{\varphi, [a]\varphi\} \cup C(\varphi) & \text{since } \varphi \in C(\varphi) \\ = & \{\bullet, [a]\bullet\} \triangleleft \varphi \cup C(\varphi) & \\ = & C([a]\bullet) \triangleleft \varphi \cup C(\varphi) & \end{array}$$

2. We have $C([\alpha;\beta]\bullet)$

$$\begin{array}{lll} = & \{[\alpha;\beta]\bullet\} \ \cup \ C([\alpha][\beta]\bullet) \ \cup \ \{\bullet\} & \text{by the definition of} \Longrightarrow_{FL} \\ = & \{[\alpha;\beta]\bullet\} \ \cup \ C([\alpha]\bullet) \triangleleft ([\beta]\bullet) \ \cup \ C([\beta]\bullet) \ \cup \ \{\bullet\} & \text{IH for } \alpha \\ = & \{[\alpha;\beta]\bullet\} \ \cup \ C([\alpha]\bullet) \triangleleft ([\beta]\bullet) \ \cup \ C([\beta]\bullet) \ \cup \ \{\bullet\} & \text{IH for } \beta \end{array}$$

Calculating now for $[\alpha;\beta]\varphi$,
$C([\alpha;\beta]\varphi)$

$$\begin{array}{lll} = & \{[\alpha;\beta]\varphi\} \ \cup \ C([\alpha][\beta]\varphi) \ \cup \ C(\varphi) & \text{by the definition of} \Longrightarrow_{FL} \\ = & \{[\alpha;\beta]\varphi\} \ \cup \ C([\alpha]\bullet) \triangleleft ([\beta]\varphi) \ \cup \ C([\beta]\varphi) \ \cup \ C(\varphi) & \text{IH for } \alpha \\ = & \{[\alpha;\beta]\varphi\} \ \cup \ C([\alpha]\bullet) \triangleleft ([\beta]\varphi) \ \cup \ C([\beta]\bullet) \triangleleft \varphi \ \cup \ C(\varphi) & \text{IH for } \beta \\ = & (\{[\alpha;\beta]\bullet\} \ \cup \ C([\alpha]\bullet) \triangleleft ([\beta]\bullet) \ \cup \ C([\beta]\bullet)) \triangleleft \varphi & \\ & \qquad \cup \, C(\varphi) & \\ = & C([\alpha;\beta]\bullet) \triangleleft \varphi \ \cup \ C(\varphi) & \text{by our previous calculation} \end{array}$$

3. $C([\alpha \cup \beta]\varphi)$

$$\begin{array}{lll} = & \{[\alpha \cup \beta]\varphi\} \ \cup \ C([\alpha]\varphi) \ \cup \ C([\beta]\varphi) \ \cup \ C(\varphi) & \text{by the definition of} \Longrightarrow_{FL} \\ = & \{[\alpha \cup \beta]\varphi\} \ \cup \ C([\alpha]\bullet) \triangleleft \varphi \ \cup \ C([\beta]\bullet) \triangleleft \varphi & \\ & \qquad \cup \, C(\varphi) & \text{IH} \\ = & (\{[\alpha \cup \beta]\bullet\} \ \cup \ C([\alpha]\bullet) \ \cup \ C([\beta]\bullet) \triangleleft \varphi & \\ & \qquad \cup \, C(\varphi) & \text{by the definition of } \triangleleft \\ = & C([\alpha \cup \beta]\bullet) \triangleleft \varphi \ \cup \ C(\varphi) & \end{array}$$

4. We have $C([\alpha^*]\bullet) = \{[\alpha^*]\bullet\} \ \cup \ C([\alpha]\bullet) \triangleleft ([\alpha^*]\bullet \ \cup \{\bullet\}$ by Corollary 1.3.5 applied to $\bullet$. So
$C([\alpha^*]\varphi)$

$$\begin{array}{lll} = & \{[\alpha^*]\varphi\} \ \cup \ C([\alpha]\bullet) \triangleleft ([\alpha^*]\varphi) \ \cup \ C(\varphi) & \text{by Corollary 1.3.5} \\ = & \{[\alpha^*]\varphi\} \ \cup \ C([\alpha]\bullet) \triangleleft ([\alpha^*]\varphi) \ \cup \ \{\varphi\} \ \cup \ C(\varphi) & \text{since } \varphi \in C(\varphi) \\ = & C([\alpha^*]\bullet) \triangleleft \ \cup C(\varphi) & \end{array}$$

5. $C([?\chi]\varphi)$

$$\begin{array}{lll} = & \{[?\chi]\varphi\} \ \cup \ C(\chi) \ \cup \ C(\varphi) & \text{by the definition of} \Longrightarrow_{FL} \\ = & (\{[?\chi]\bullet\} \ \cup \ C(\chi)) \triangleleft \varphi \ \cup \ C(\varphi) & \text{since } \chi \text{ is } \textit{bullet}\text{-free} \\ = & (C([?\chi]\bullet) \triangleleft \varphi \ \cup \ C(\varphi) & \end{array}$$

$\dashv$

Given a set D we write $\#D$ for the number of elements in D. If σ is a PD formula or a PD program, we write $|\sigma|$ for the size of σ, that is the number of nodes in its syntax tree.

THEOREM **1.3.7** *For all PD formulas φ and PD programs α,*

1. $\#C(\varphi) \leq |\varphi|$.
2. $\#C([\alpha]\bullet) \leq |\alpha|$.

Proof. By simultaneous induction on programs and formulas.

1. $\#C(\mathbf{p}) = \#(\{\mathbf{p}\}) = 1 = |\mathbf{p}|$.
2. $\#C([\mathbf{a}]\bullet) = \#\{[\mathbf{a}]\bullet, \bullet\} = 2 = |[\mathbf{a}]\bullet|$
3. $C(\varphi \wedge \psi)$
$$\begin{array}{lll} = & \{\varphi \wedge \psi\} \cup \#(C(\varphi) \cup C(\psi)) & \\ \leq & 1 + \#C(\varphi) + \#C(\psi) & \\ \leq & 1 + |\varphi| + |\psi| & \text{IH} \\ = & |\varphi \wedge \psi| & \end{array}$$
4. $\#C(\neg\varphi)$
$$\begin{array}{lll} = & \#(\{\neg\varphi\} \cup C(\varphi) & \\ = & 1 + \#C(\varphi) & \\ \leq & 1 + |\varphi| & \text{IH} \\ = & |\neg\varphi| & \end{array}$$
5. $\#C([\alpha]\varphi)$
$$\begin{array}{lll} = & \#(C([\alpha]\bullet) \triangleleft \varphi \cup C(\varphi))\text{by Lemma 1.3.6} & \\ \leq & \#C([\alpha]\bullet) + \#C(\varphi) & \\ \leq & |\alpha| + |\varphi| & \text{IH} \\ \leq & |[\alpha]\varphi| & \end{array}$$
6. $\#C([\alpha;\beta]\bullet)$
$$\begin{array}{lll} = & \#(\{[\alpha;\beta]\bullet\} \cup C([\alpha][\beta]\bullet) \cup \{\bullet\}) & \text{by the definition of } \Longrightarrow_{FL} \\ = & \#(\{[\alpha;\beta]\bullet\} \cup C([\alpha][\beta]\bullet)) & \text{since } \bullet \in C([\alpha][\beta]\bullet) \\ \leq & 1 + \#(C([\alpha]\bullet) \triangleleft ([\beta]\bullet) \cup C([\beta]\bullet)) & \text{by Lemma 1.3.6} \\ \leq & 1 + \#C([\alpha]\bullet) \triangleleft ([\beta]\bullet) + \#C([\beta]\bullet)) & \\ = & 1 + \#C([\alpha]\bullet) + \#C([\beta]\bullet)) & \\ \leq & 1 + |\alpha| + |\beta| & \text{IH} \\ = & |\alpha;\beta| & \end{array}$$
7. $\#C([\alpha \cup \beta]\bullet)$
$$\begin{array}{lll} = & \#(\{[\alpha \cup \beta]\bullet\} \cup C([\alpha]\bullet) \cup C([\beta]\bullet) \cup \{\bullet\}) & \text{by the definition of } \Longrightarrow_{FL} \\ = & \#(\{[\alpha \cup \beta]\bullet\} \cup C([\alpha]\bullet) \cup C([\beta]\bullet)) & \\ \leq & 1 + \#C([\alpha]\bullet) + \#C([\alpha]\bullet) & \\ \leq & 1 + |\alpha| + |\beta| & \text{IH} \\ = & |\alpha \cup \beta| & \end{array}$$
8. $\#C([?\chi]\bullet)$
$$\begin{array}{lll} = & \#(C(\chi) \cup \{\bullet\})\text{by the definition of } \Longrightarrow_{FL} & \\ \leq & \#C(\chi) + 1 & \\ \leq & |\chi| + 1 & \text{IH} \\ = & |?\chi| & \end{array}$$

1.3.5. Definition of the counter-model

We now prove the Completeness Theorem **??**. Suppose given φ where $\not\vdash \varphi$. We construct a PD structure $\mathcal{K}_\varphi = (S, [\![\cdot]\!]_\mathcal{K})$ as follows.

- The states are the boolean combinations over F_φ.
- $[\![p]\!](s)$ is defined to be true iff $p \in s$.
- $s \xrightarrow{a} t$ is defined to hold iff for every $[a]\psi \in C(\varphi)$, if $t \vdash \neg\psi$ then $s \vdash \neg[a]\psi$. Intuitively: $s \xrightarrow{a} t$ holds exactly when, for every relevant assertion ψ, s rejects a promise that ψ will hold after a whenever t rejects ψ.

Note. We do not require that each state (as a set of formulas) be consistent under the inference rules of PDL.

LEMMA **1.3.8** *1. For every $G \subseteq F_\varphi$, $\vdash (\bigvee_{u\in G} u) \longleftrightarrow (\bigwedge_{v \in S-G} \neg v)$.*

2. For every $\boldsymbol{\psi} \in F_\varphi$, $\vdash \boldsymbol{\psi} \leftrightarrow \bigvee_{\boldsymbol{\psi} \in s \in S} s$.

3. For every $\boldsymbol{\psi} \in F_\varphi$, $\vdash \boldsymbol{\psi} \leftrightarrow \bigwedge_{\psi \notin s \in S} \neg s$.

Proof.

1. By the definition of the boolean combinations, $u \rightarrow \neg v$ for any distinct $u, v \in S$. So $(\bigvee_{u\in G} u) \rightarrow (\bigwedge_{v\in S-G} \neg v)$. Conversely, by Lemma 2, $\vdash (\bigvee_{u \in G} u) \vee (\bigvee_{v\in S-G} v)$, so $\vdash \neg(\bigvee_{v \in S-G} v) \rightarrow (\bigvee G)$, i.e. $\vdash (\bigwedge_{v\in S-G} \neg v) \rightarrow (\bigvee G)$.

2.

$$\begin{array}{lll} \psi & \leftrightarrow (\psi \wedge \bigvee_{s\in S} s) & \text{by (1)} \\ & \leftrightarrow \bigvee_{s\in S} (\psi \wedge s) & \\ & \leftrightarrow \bigvee_{\psi \in s} s & \text{since } \psi \text{ is absorbed in these disjuncts,} \\ & \text{and contradicts the others} & \end{array}$$

3. Set $G = \{s \in S \mid \boldsymbol{\psi} \in S\}$. Then

$$\begin{array}{lll} \boldsymbol{\psi} & \leftrightarrow \bigvee_{s \in G} s & \text{by (2)} \\ & \leftrightarrow \bigwedge_{\psi \notin s} \neg s & \text{by (1)} \end{array}$$

$\dashv$

1.3.6. Proof of the Completeness Theorem 1.3.1

PROPOSITION **1.3.9** *Referring to $\mathcal{K}_\varphi$ defined above, we have for all ψ and α that occur in $C(\varphi)$,*

1. If $s \models \psi$ then $s \vdash \psi$.

2. *If* $s \not\models \psi$ *then* $s \vdash \neg\psi$.

3. *If* $s \xrightarrow{\alpha} t$ *then for all* $[\alpha]\xi$ *in* $C(\varphi)$, *if* $t \vdash \neg\xi$ *then* $s \vdash \neg[\alpha]\xi$.

4. *If* $s \not\xrightarrow{\alpha} t$, *then* $s \vdash [\alpha]\neg t$.

From Proposition 1.3.9 we obtain the Completeness Theorem for PDL (1.3.1 as follows. Given φ, construct $\mathcal{K}_\varphi$. If $\not\vdash \varphi$, then $(\bigvee_{s\in S} s) \not\vdash \varphi$, by Lemma 2. Therefore (by propositional logic), there is some $s \in S$ for which $s \not\vdash \varphi$. By part (1) of Proposition 1.3.9, this implies that $\mathcal{K}_\varphi, s \not\models \varphi$, concluding the proof.

proof of Proposition 1.3.9. We proceed by structural induction on expressions ψ and α, simultaneously for all four parts.

1 and 2.

- $s \models \mathbf{p}$

$$\begin{array}{lll} \Longrightarrow & \mathbf{p} \in s & \text{by definition of } \mathcal{K}_\varphi \\ \Longrightarrow & s \vdash \mathbf{p} & \end{array}$$

- $s \not\models \mathbf{p}$

$$\begin{array}{lll} \Longrightarrow & \mathbf{p} \notin s & \text{by definition of } \mathcal{K}_\varphi \\ \Longrightarrow & \neg\mathbf{p} \in s & \text{since } s \text{ is a boolean combination of } C(\varphi) \\ \Longrightarrow & s \vdash \neg\mathbf{p} & \end{array}$$

- $s \models \neg\psi$

$$\begin{array}{lll} \Longrightarrow & s \not\models \psi & \\ \Longrightarrow & s \vdash \neg\psi & \text{IH} \end{array}$$

- $s \not\models \neg\psi$

$$\begin{array}{lll} \Longrightarrow & s \models \psi & \\ \Longrightarrow & s \vdash \psi & \text{IH} \\ \Longrightarrow & s \vdash \neg\neg\psi & \end{array}$$

- $s \models \psi \wedge \chi$

$$\begin{array}{lll} \Longrightarrow & s \models \psi \quad \text{and} \quad s \models \chi & \\ \Longrightarrow & s \vdash \psi \quad \text{and} \quad s \vdash \chi & \text{IH} \\ \Longrightarrow & s \vdash \psi \wedge \chi & \end{array}$$

- $s\neg \models \psi \wedge \chi$

$$\begin{array}{lll} \Longrightarrow & s\neg \models \psi \quad \text{or} \quad s\neg \models \chi & \\ \Longrightarrow & s \vdash \neg\psi \quad \text{or} \quad s \vdash \neg\chi & \text{IH} \\ \Longrightarrow & s \vdash \neg(\psi \wedge \chi) & \end{array}$$

- $s \models [\alpha]\psi$

$$\begin{array}{lll}
\Longrightarrow & \text{for all states } t, s \stackrel{\alpha}{\not\longrightarrow} t \quad \text{or} \quad t \models \psi & \\
\Longrightarrow & \text{for all states } t, s \vdash [\alpha]\neg t \quad \text{or} \quad t \vdash \psi & \text{IH} \\
\Longrightarrow & \text{for all states } t, s \vdash [\alpha](t \rightarrow \psi) & \text{by PDL rules} \\
\Longrightarrow & s \vdash \bigwedge_{t \in S} [\alpha](t \rightarrow \psi) & \text{by prop logic} \\
\Longrightarrow & s \vdash [\alpha] \bigwedge_{t \in S} (t \rightarrow \psi) & \text{by PDL} \\
\Longrightarrow & s \vdash [\alpha]((\bigvee_{t \in S} t) \rightarrow \psi) & \text{by prop logic and box distribution} \\
\Longrightarrow & s \vdash [\alpha](\top \rightarrow \psi) & \text{by Lemma } \textbf{??} \\
\Longrightarrow & s \vdash [\alpha]\psi &
\end{array}$$

- $s \not\models [\alpha]\psi$

$$\begin{array}{lll}
\Longrightarrow & \text{for some state } t, s \xrightarrow{\alpha} t \quad \text{and} \quad t \not\models \psi & \\
\Longrightarrow & \text{for some state } t, s \xrightarrow{\alpha} t \quad \text{and} \quad t \vdash \neg\psi & \text{IH(2)} \\
\Longrightarrow & \text{for some state } t, s \vdash \neg[\alpha]\psi & \text{IH(3)}
\end{array}$$

3 and 4. • For α an atomic program a, (3) holds by the definition of $\mathcal{K}_\varphi$.

-
- $s \stackrel{a}{\not\longrightarrow} t$

$$\begin{array}{ll}
\Longrightarrow & \text{for some } [a]\xi \text{ in } C(\varphi), \quad t \vdash \neg\xi \quad \text{and} \quad s \not\vdash \neg[a]\xi \\
\xi \vdash \neg t \quad \text{and} \quad (\neg[a]\xi) \notin s & \\
\xi \vdash \neg t \quad \text{and} \quad [a]\xi \in s & \text{since } s \text{ is a state} \\
s \vdash [a]\xi \text{and} \quad [a]\xi \vdash [a]\neg t & \text{by PDL rules} \\
s \vdash [a]\neg t &
\end{array}$$

- $s \xrightarrow{\alpha;\beta} t$

$$\begin{array}{lll}
\Longrightarrow & \text{for some state } u, \quad s \xrightarrow{\alpha} u \xrightarrow{\beta} t & \\
\Longrightarrow & \text{for all formula } [\alpha;\beta]\xi \text{ in } C(\varphi), & \\
& t \vdash \neg\xi & \\
& u \vdash \neg[\beta]\xi & \text{by IH, since } [\beta]\xi \text{ is in } C(\varphi) \\
& s \vdash \neg[\alpha][\beta]\xi & \text{by IH, since } [\alpha][\beta]\xi \text{ is in } C(\varphi) \\
& s \vdash \neg[\alpha;\beta]\xi & \text{by PDL rules}
\end{array}$$

- $s \stackrel{\alpha;\beta}{\not\longrightarrow} t$

$$\begin{array}{lll}
\Longrightarrow & \text{for all states } u, \quad s \stackrel{\alpha}{\not\longrightarrow} u \quad \text{or} \quad u \stackrel{\beta}{\not\longrightarrow} t & \\
\Longrightarrow & \text{for all states } u, \quad s \vdash [\alpha]\neg u \quad \text{or} \quad u \vdash [\beta]\neg t & \text{IH} \\
\Longrightarrow & \text{for all states } u, \quad s \vdash [\alpha](u \rightarrow [\beta]\neg t) & \text{by PDL} \\
\Longrightarrow & s \vdash [\alpha][\beta]\neg t & \text{as above} \\
\Longrightarrow & s \vdash [\alpha;\beta]\neg t &
\end{array}$$

- $s \xrightarrow{\alpha\cup\beta} t$

$$
\begin{array}{lll}
\Longrightarrow & s\xrightarrow{\alpha}t \quad \text{or} \quad s\xrightarrow{\beta}t & \\
\Longrightarrow & \text{for all formula } [\alpha]\xi \text{ in } C(\varphi), & \\
 & \qquad t \vdash \neg\xi \quad \text{implies} \quad s \vdash \neg[\alpha]\xi & \\
 & \qquad \text{or for all formula } [\beta]\xi \text{ in } C(\varphi), & \\
 & \quad t \vdash \neg\xi \quad \text{implies} \quad s \vdash \neg[\alpha]\xi & \text{IH} \\
\Longrightarrow & \text{for all formula } [\alpha\cup\beta]\xi \text{ in } C(\varphi), & \\
 & \quad t \vdash \neg\xi \quad \text{implies} \quad s \vdash \neg[\alpha]\xi \quad \text{or} \quad s \vdash \neg[\beta]\xi & \text{definition of } C(\varphi) \\
\Longrightarrow & \text{for all formula } [\alpha\cup\beta]\xi \text{ in } C(\varphi), & \\
 & \quad t \vdash \neg\xi \quad \text{implies} \quad s \vdash \neg[\alpha\cup\beta]\xi & \text{PDL rules}
\end{array}
$$

- $s \not\xrightarrow{\alpha\cup\beta} t$

$$
\begin{array}{lll}
\Longrightarrow & s \not\xrightarrow{\alpha} t \quad \text{and} \quad s \not\xrightarrow{\beta} t & \\
\Longrightarrow & s \vdash [\alpha]\neg t \quad \text{and} \quad s \vdash [\beta]\neg t & \text{IH} \\
\Longrightarrow & s \vdash [\alpha\cup\beta]\neg t & \text{by PDL rules}
\end{array}
$$

- $s\xrightarrow{\alpha^*}t$

$$
\begin{array}{lll}
\Longrightarrow & s = u_1\xrightarrow{\alpha}u_2\xrightarrow{\alpha}\cdots\xrightarrow{\alpha}u_k = t \quad \text{for seom states } u_1,\ldots,u_k & \\
\Longrightarrow & \text{for all formulas } [\alpha^*]\xi \text{ in } C(\varphi) & (\text{whence } \xi, [\alpha][\alpha^*]\alpha \text{ in } C(\\
 & \quad t \vdash \neg\xi \quad \Longrightarrow t \vdash \neg[\alpha^*]\xi & \text{by PDL rules} \\
 & \qquad\qquad \Longrightarrow u_{k-1} \vdash \neg[\alpha][\alpha^*]\xi & \text{IH} \\
 & \qquad\qquad \Longrightarrow u_{k-1} \vdash \neg[\alpha^*]\xi & \text{by PDL} \\
 & \qquad\qquad \ldots & \text{induction on } k \\
 & \qquad\qquad \Longrightarrow s \vdash \neg[\alpha^*]\xi &
\end{array}
$$

- Suppose $s \not\xrightarrow{\alpha^*} t$. Let $G = \{u \in S \mid s\xrightarrow{\alpha^*}u\}$. For $u \in G$, $v \notin G$, we have $u \not\xrightarrow{\alpha} v$, and so (by IH) $u \vdash [\alpha]\neg v$. Thus

$$
\begin{array}{lll}
\vdash \bigvee_{u\in G} u & \to \bigwedge_{v\notin G}[\alpha]\neg v & \\
 & \to [\alpha] \bigwedge_{v\notin G} \neg v & \text{by PDL} \\
 & \to [\alpha] \bigvee_{v\in G} v & \text{by prop logic}
\end{array}
$$

So by the Invariance Rule of PDL,

$$\vdash \bigvee_{u\in G} u \;\to\; [\alpha^*] \bigvee_{v\in G} v$$

Reversing the argument above, this implies that $u \vdash [\alpha^*]\neg v$ for all $u \in G$, $v \notin G$. In particular, $s \vdash [\alpha^*]\neg t$.

- $s\xrightarrow{?\chi}t$

$$
\begin{array}{lll}
\Longrightarrow & s = t \quad \text{and} \quad s \models \chi & \\
\Longrightarrow & s = t \quad \text{and} \quad s \vdash \chi & \text{IH} \\
\Longrightarrow & \text{for all formulas } [?\chi]\xi \text{ in } C(\varphi) & (\text{whence } \chi, \xi \in C(\varphi)) \\
 & \quad t \vdash \neg\xi \quad \Longrightarrow s \vdash \neg(\chi\to\xi) & \\
 & \qquad\qquad \Longrightarrow s \vdash \neg[?\chi]\xi &
\end{array}
$$

- $s \not\xrightarrow{?\chi} t$

$$\begin{array}{lll} \Longrightarrow & s \neq t \quad \text{or} \quad s \not\models \chi & \\ \Longrightarrow & s \vdash \neg t \quad \text{or} \quad s \vdash \neg\chi & \text{IH} \\ \Longrightarrow & s \vdash t \rightarrow \neg\chi & \\ \Longrightarrow & s \vdash \chi \rightarrow \neg t & \text{prop logic} \\ \Longrightarrow & s \vdash [?\chi]\neg t & \end{array}$$

$\dashv$

EXERCISES **1.3.5** Construct the structure $\mathcal{K}_\varphi$ for the following formulas φ.

1. $p \rightarrow [a]p$.
2. $p \rightarrow \langle a \rangle p$
3. $(p \rightarrow [a]q) \vee (q \rightarrow [a]p)$

1.3.7. Failure of strong completeness

The Completeness Theorem is equivalent to the statement that every consistent formula is satisfiable, that is, if φ is not provably refutable, then it is true in some structure. For propositional logic much more can be proved: every consistent *set* of formulas is satisfiable. This property is dubbed *strong completeness*, and is equivalent to the statement: for every set of formulas Γ, and formula φ, if $\Gamma \models \varphi$, then $\Gamma \vdash \varphi$. Here we write $\Gamma \models \varphi$ if φ is true in each model of Γ (i.e. each propositional structure in which all formulas in Γ are true). Also, $\Gamma \vdash \varphi$ means that there is a derivation of φ in which all open assumptions are among Γ.

A consequence of the Strong Completeness Theorem is the *Joint Satisfiability Theorem* (often called the Compactness Theorem): if every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable, then so is Γ. For if Γ is unsatisfiable, then $\Gamma \models \bot$, and so $\Gamma \vdash \bot$, by Strong Completeness. If $\mathcal{D}$ is a derivation of $\bot$ from grG, and Γ_0 is the set of open assumptions in $\mathcal{D}$, then $\Theta \vdash \bot$, and so (by the Soundness Theorem) Θ is not satisfiable.

Can we extend the Completeness Theorem for PDL to a Strong Completeness Theorem? A consequence would then be a Joint Satisfiability property for PDL. The following simple example shows that Joint Satisfiability fails, and therefore we also don't have a Strong Completeness Theorem for PDL.

PROPOSITION **1.3.10** *PDL for regular programs does not have Joint Satisfiability (aka Compactness) Property.*

Proof. Let $\Phi = \{\, \mathbf{p},\ [a]\mathbf{p},\ [a;a]\mathbf{p},\ [a;a;a]\mathbf{p}, \ldots,\ [a^i]\mathbf{p},\ \ldots\}$, and $\Gamma = \Phi \cup \{\neg[a^*]\mathbf{p}\}$. If $\Gamma_0 \subset \Gamma$ is finite, then it is satisfiable: if n is a bound on all i with $[a^i]\mathbf{p}$ in Γ_0, then the following is a model of Γ_0:

[INSERT PS]

$\dashv$

EXERCISES **1.3.6** Prove the equivalence of these two variants of completeness:

1. $\models \varphi$ implies $\vdash \varphi$.
2. Every consistent formula is satisfiable.

1.3.7 Prove the equivalence of these two variants of strong completeness:

1. $\Gamma \models \varphi$ implies $\Gamma \vdash \varphi$.
2. Every consistent set of formulas is satisfiable.

1.3.8 Prove that for finite sets Γ, the statement of strong completeness follows directly from the Completeness Theorem.

1.3.9 Show that there are Γ and φ such that $\Gamma \models \varphi$, but for no proper subset Γ' of Γ do we have $\Gamma' \models \varphi$.

1.4 Decidability of PDL

1.4.1. The virtues of consistency

We saw that the Finite Model property of PDL implies that PDL is decidable. That argument gave no clue as to the complexity of the decision, because it used no information about the *size* of the counter-model for an unprovable φ. But the proof of completeness does provide a bound on that size: for a formula φ of size n, $C(\varphi)$ has at most n elements, and so $\mathcal{K}_\varphi$ has at most 2^n states. It can be seen that, considering only the propositional and atomic-action identifiers in φ, whose number is $\leq n$, then number of models with up to 2^n states is $O(2^{2^{O(n)}})$. Consequently, the validity of φ can be decided by checking the truth of φ in all states of all $O(2^{2^{O(n)}}$ such models. This process takes $O(2^{2^n}$ computation steps (Exercise).

EXERCISE **1.4.1** Justify the two claims above: (1) that there are $O(2^{2^{O(n)}})$ structures to consider, and (2) that the outline algorithm takes time $O(2^{2^{O(n)}})$.

An algorithm that runs in double-exponential time is unfortunately of dubious significance: already 2^{2^7} is larger than the age of the universe measured in nano-seconds! Perhaps we can do better by simply constructing the particular test structure $\mathcal{K}_\varphi$, rather than considering all possible structure whose size does not exceed that of $\mathcal{K}_\varphi$. The one difficulty is that we have defined $s \xrightarrow{a} t$ in $\mathcal{K}_\varphi$ by referring to provability in PDL. But provability and validity are equivalent (as the soundness and completeness theorems show), so we fall back to what we try to determine in the first place.

Perhaps we might find an alternative counter-model, where $\xrightarrow{a}$ is easier to define. Indeed, we have a wealth of such alternative structures, provided directly by our proof of the Completeness Theorem, and the construction there of the structure $\mathcal{K}_\varphi$. We took the states of $\mathcal{K}_\varphi = (S, [\![\cdot]\!])$ to be all boolean combinations of $C(\varphi)$, but that choice was used just once: in proving $\bigvee_{t \in S} t$. Suppose we omit from S some states t that are inconsistent (under PDL), that is such that $\vdash \neg t$ in PDL. Since $\perp \vee \psi$ is equivalent to ψ, we would still have that $\vdash \bigvee_{t \in S} t$ (though *this* proof may actually rely on the axioms of PDL).

Among the possible choices for S is the minimal one, consisting just of the consistent states, nothing more. Write $\mathcal{M}_\varphi$ for this structure. In this PD structure we have some useful equivalences.

LEMMA **1.4.1** *In M_φ , for all states s, t, formula $\psi \in C(\varphi)$:*

1. *The following three statements are equivalent*
   (a) $s \models \varphi$
   (b) $s \ni \psi$
   (c) $s \vdash \psi$
2. $s \xrightarrow{\alpha} t$ *implies*

$$\forall\, [\alpha]\xi \in C(\varphi).\ [\alpha]\xi \in s \Longrightarrow \xi \in t$$

Proof.

- $s \models \psi$

$$\begin{array}{lll} \Longrightarrow & s \vdash \psi & \text{Prop. 1.3.9} \\ \Longrightarrow & s \not\vdash \neg\psi & \text{consistency} \\ \Longrightarrow & s \not\ni \neg\psi & \\ \Longrightarrow & s \ni \psi & \text{definition of states} \\ \Longrightarrow & s \not\ni \neg\psi & \\ \Longrightarrow & s \not\models \neg\psi & \text{Prop. 1.3.9} \\ \Longrightarrow & s \models \psi & \end{array}$$

- If $s \xrightarrow{\alpha} t$ then for all $[\alpha]\xi \in C(\varphi)$

$$\begin{array}{lllllll} t \vdash \neg\xi & \Longrightarrow & s \vdash \neg[\alpha]\xi & \text{Prop } \varphi\text{main-completeness} & & & \\ \Longrightarrow & & & t \ni \neg\xi & \Longrightarrow & s \ni \neg[\alpha]\xi & \text{using (1)} \\ \Longrightarrow & & & s \ni [\alpha]\xi & \Longrightarrow & t \ni \xi & \end{array}$$

$\dashv$

By (1), we may define $s \xrightarrow{\alpha} t$ in $\mathcal{M}_\varphi$ by the condition

$$\text{for all} \quad [a]\xi \in C(\varphi) \qquad [a]\xi \in s \quad \Longrightarrow \quad \xi \in t \tag{1.4.1}$$

This is an easy condition to check, as it does not refer to provability in PDL. Perhaps we can check efficiently the validity of φ by constructing $\mathcal{M}_\varphi$, and checking whether all states therein satisfy φ? This does not at first appear to be any easier, though, than checking φ in $\mathcal{K}_\varphi$, because to construct $\mathcal{M}_\varphi$ we need to identify the consistent states, thereby referring to provability in a new way. Fortunately, it turns out that we can construct $\mathcal{M}_\varphi$ efficiently without invoking provability.

1.4.2. Compatible and witnessed structures

Consistency of a state s implies, in particular, the absence of obvious, easy-to-check, inconsistencies. Here is a list of such "sanity-checks".

$$
\begin{array}{rcl}
\psi \wedge \chi \in s & \Longleftrightarrow & \psi, \chi \in s \\
\neg\psi \in s & \Longleftrightarrow & \psi \notin s \\
[\alpha;\beta]\psi \in s & \Longleftrightarrow & [\alpha][\beta]\psi \in s \\
[\alpha \cup \beta]\psi \in s & \Longleftrightarrow & [\alpha]\psi, [\beta]\psi \in s \\
[\alpha^*]\psi \in s & \Longleftrightarrow & \psi, [\alpha][\alpha^*]\psi \in s \\
[?\chi]\psi \in s & \Longleftrightarrow & \neg\chi \in s \quad \text{or} \quad \psi \in s
\end{array}
$$

We call a state s *compatible* when it satisfies each of the conditions above, whenever the left-side formula is in $C(\varphi)$ (which implies that the corresponding right-side formula(s) are in $C(\varphi)$, by the closure conditions defining $C(\varphi)$).

Obviously, the states of $\mathcal{M}_\varphi$ are all compatible. The contents of states relates in $\mathcal{M}_\varphi$ to their semantic behavior also in ways that relate to transition. One such property is

$$\langle\alpha\rangle\xi \in s \quad \Longrightarrow \quad \text{for some} \quad t \xleftarrow{\alpha} s. \quad \xi \in t$$

If this holds we say that α *is witnessed at* s. Stated in contrapositive, using the box rather than the diamond operator, this reads:

$$(\forall t \xleftarrow{\alpha} s. \quad \xi \in t) \quad \Longrightarrow \quad [\alpha]\xi \in s \tag{1.4.2}$$

LEMMA **1.4.2** *In M_φ, all programs are witnessed at all states.*

Proof. In $\mathcal{M}_\varphi$,

$$
\begin{array}{rll}
(\forall t \xleftarrow{\alpha} s. \quad \xi \in t) & \Longrightarrow (\forall t \xleftarrow{\alpha} s. \quad t \vdash \xi) & \text{Prop 1.4.1} \\
& \Longrightarrow s \vdash [\alpha]\xi) & \text{Definition of } \models
\end{array}
$$

$\dashv$

Thus, the definition of $\mathcal{M}_\varphi$ implies two easy-to-verify "good-behavior" properties: all states are compatible, and all programs are witnessed at all states. Let us now reverse the point of view, and consider those structures (built on boolean combinations of $C(\varphi)$ as states), which satisfy these two properties.

Suppose S is a set of boolean combinations of $C(\varphi)$, that contains all the consistent combinations. We write $\mathcal{K}_S$ for the PD structure whose states are the elements of S, and where $[\![\cdot]\!]$ is defined by

- $s \vdash \mathbf{p}$ iff $\mathbf{p} \in s$
- $s \xrightarrow{a} t$ iff $\forall\, [a]\xi \in C(\varphi).\ [a]\xi \in s \Longrightarrow \xi \in t$.

(Note that the definition of $\xrightarrow{a}$ here is equivalent to the definition in $\mathcal{K}_\varphi$ if S consists exactly of the consistent states, i.e. $\mathcal{K}_S$ is $\mathcal{M}_\varphi$. But need not coincide with that definition when S inlcudes inconsistent states.)

LEMMA **1.4.3** *Let S be a set of boolean combinations of $C(\varphi)$, that contains all the consistent combinations. Suppose further that every program in φ is witnessed in S at every state. Then, in $\mathcal{K}_S$,*

1. *$s \models \psi$ iff $\psi \in s$, for all $\psi \in C(\varphi)$.*
2. *If $s \xrightarrow{\alpha} t$, then $\forall\, [\alpha]\xi \in C(\varphi).\ [\alpha]\xi \in s \Longrightarrow \xi \in t$.*
3. *If $s \xrightarrow{\alpha} t$ in $\mathcal{M}_\varphi$ then $s \xrightarrow{\alpha} t$ in $\mathcal{K}_S$.*[2]

Proof. By simultaneous structural induction on ψ and α.

- Base case for (1):
  $s \models \mathbf{p}$ iff $\mathbf{p} \in s$ by definition of $\mathcal{K}_S$.
-
$$\begin{aligned} s \models \neg\psi &\iff s \not\models \psi \\ &\iff \psi \notin s && \text{IH} \\ &\iff (\neg\psi) \in s && \text{compatibility} \end{aligned}$$
-
$$\begin{aligned} s \models \psi \wedge \chi &\iff s \models \psi \quad \text{and} \quad s \models \chi \\ &\iff \psi \in s \quad \text{and} \quad \chi \in s && \text{IH} \\ &\iff (\psi \wedge \chi) \in s && \text{compatibility} \end{aligned}$$
-
$$\begin{aligned} s \models [\alpha]\psi &\Longrightarrow \forall t \xleftarrow{\alpha} s.\ t \models \psi \\ &\Longrightarrow \forall t \xleftarrow{\alpha} s.\ \psi \in t \\ &\Longrightarrow ([\alpha]\psi) \in s && \alpha \text{ witnessed at } s \end{aligned}$$
-
$$\begin{aligned} ([\alpha]\psi) \in s &\Longrightarrow \forall t \xleftarrow{\alpha} s.\ \psi \in t && \text{IH(2) for } \alpha \\ &\Longrightarrow \forall t \xleftarrow{\alpha} s.\ t \models \psi && \text{IH(1) for } \psi \\ &\Longrightarrow s \models [\alpha]\psi \end{aligned}$$
- Base case for (2): $s \xrightarrow{a} t$ implies $\forall\, [\alpha]\xi \in C(\varphi).\ [a]\xi \in s \Longrightarrow \xi \in t$, by the definition of $\mathcal{K}_S$.

[2]This part of the Lemma is not needed to prove the first two, but will be useful momentarily.

- If $s \xrightarrow{\alpha;\beta} t$ then $s \xrightarrow{\alpha} u \xrightarrow{beta} t$ for some state u. By IH, we then have for every $[\alpha;\beta]\psi$ in $C(\varphi)$

$$\begin{array}{lll} ([\alpha;\beta]\xi) \in s & \Longrightarrow ([\alpha][\beta]\xi) \in s & \text{compatibility} \\ & \Longrightarrow ([\beta]\xi) \in u & \text{IH for } \alpha \\ & \Longrightarrow (\xi) \in t & \text{IH for } \beta \end{array}$$

- The case for $\alpha \cup \beta$ is similar, and left as an Exercise.

- If $s \xrightarrow{\alpha^*} t$ then there are $u_0 \ldots u_k$ ($k \geq 0$) such that $s = u_0 \xrightarrow{\alpha} u_1 \xrightarrow{\alpha} \cdots \xrightarrow{\alpha} u_k = t$. We then have

$$\begin{array}{lll} ([\alpha^*]\xi) \in s = u_0 & \Longrightarrow ([\alpha][\alpha^*]\xi) \in s & \text{compatibility} \\ & \Longrightarrow ([\alpha^*]\xi) \in u_1 & \text{IH for } \alpha \\ & \Longrightarrow ([\alpha][\alpha^*]\xi) \in u_1 & \text{compatibility} \\ & \Longrightarrow \cdots & \\ & ([\alpha^*]\xi) \in u_k = t & \text{induction on } k \\ & \xi \in t & \text{compatibility} \end{array}$$

- If $s \xrightarrow{?\chi} t$ then $t = s \models \chi$, which by IH implies $t \ni \chi$.

- Base case for (3): If $s \xrightarrow{\alpha} t$ in $\mathcal{M}_\varphi$ then s, t are consistent, and therefore in S. The definition of $\xrightarrow{a}$ is identical in both PD structures, so $s \xrightarrow{\alpha} t$ in $\mathcal{K}_S$.

- $s \xrightarrow{\alpha;\beta} t$ in $\mathcal{M}_\varphi$

$$\begin{array}{lll} \Longrightarrow s \xrightarrow{\alpha} u \xrightarrow{\beta} t \quad \text{in} \quad \mathcal{M}_\varphi, & \text{some consistent } u & \\ \Longrightarrow s \xrightarrow{\alpha} u \xrightarrow{\beta} t \quad \text{in} \quad \mathcal{K}_S, & \text{some } u \in S & \text{IH} \\ \Longrightarrow s \xrightarrow{\alpha;\beta} t \quad \text{in} \quad \mathcal{K}_S & & \end{array}$$

- The cases for program union and iteration are similar, and left as an Exercise.

- Finally, for a test program, we have[3] $s \xrightarrow{?\chi} t$ in $\mathcal{M}_\varphi$

$$\begin{array}{ll} \Longrightarrow t = s \models \chi \quad \text{in} \quad \mathcal{M}_\varphi & \\ \Longrightarrow t = s \ni \chi & \text{IH} \\ \Longrightarrow t = s \models \chi \quad \text{in} \quad \mathcal{K}_S & \text{IH} \\ \Longrightarrow s \xrightarrow{?\chi} t \quad \text{in} \quad \mathcal{K}_S & \end{array}$$

$\dashv$

COROLLARY

[3]This case is the reason we prove (3) in tandem with (1) here.

1.4.4 *$\mathcal{M}_\varphi$ is the unique structure $\mathcal{K}_S$ with S containing all the consistent combinations, every state compatible, and every program in φ witnessed at every state.*

Proof. By Lemma 1.4.2 the PD structure $\mathcal{M}_\varphi$ has all the properties considered. Conversely, if $\mathcal{K}_S$ has all these properties, then by Lemma 1.4.3 every state s, considered as a set of formulas, is satisfiable, namely in $\mathcal{K}_S$ at the state s. By the Soundness Theorem, it follows that every such state is consistent. ⊣

1.4.3. An exponential time decision algorithm for PDL

Note that the inductive argument proving Lemma 1.4.3 remains correct as long as the programs involved are witnessed. That is, if every program shorter than α is witnessed in all states, then $s \xrightarrow{\alpha} t$ implies

$$\forall\, [\alpha]\xi \in C(\varphi).\ [\alpha]\xi \in s \Longrightarrow \xi \in t$$

Consider a program α which fails to be everywhere witnessed: for some s and ξ we have $(\forall t \leftrightarrow \alpha s).\ \xi \in t$ but $([\alpha]\xi) \notin s$. Suppose s were consistent. Suppose that all programs shorter than α are witnessed everywhere. For every consistent t, if $t \xleftarrow{\alpha} s$ in $\mathcal{M}_\varphi$ then $t \xleftarrow{\alpha} s$ in $\mathcal{K}_S$, by Lemma 1.4.3(3), which applies since $\mathcal{K}_S$ is witnessed for all programs shorter than α. Since all consistent states are in S, it follows that $(\forall t \xleftarrow{\alpha} s).\ \xi \in t$ is true in $\mathcal{M}_\varphi$. But $\mathcal{M}_\varphi$ is witnessed, so we must have $([\alpha]\xi) \in s$, contradicting the choice of s. Thus s must be inconsistent.

This leads us to the following

Algorithm `PDL-decision`

```
S := the compatible boolean combinations of C(φ)
repeat
    Construct K_S
    for all programs α in φ,
        in non-decreasing length-order,
    repeat for s ∈ S
        if s violates witnessing for α
            then
            S := S−{s};
            exit inner loop
        end inner loop
    end outer loop
if φ ∈ s for all s ∈ S
    then output VALID
    else output NON-VALID
```

EXERCISES

1.4.2 For a PD formula φ, let $G(\varphi)$ consist of the formulas in $C(\varphi)$ that are either (1) propositional identifiers, (2) of the form $[a]\psi$, a an atomic action identifier, or (3) of the form $[\alpha^*]\psi$. Let b be a boolean combination of $G(\varphi)$. Show that there is a unique compatible extension s of b.

1.4.3 Describe an improvement of the decision procedure for PDL based on the previous exercise. Explain why the improved algorithm is still exponential time.

1.4.4 Complete the parts left as exercises in the proof of Lemma 1.4.3.

1.4.5 Explain why the algorithme `PDL-decision` runs in time exponential in the size of φ.

1.4.6 Apply the Algorithm `PDL-decision` to the following formulas.

1. $\mathbf{p} \rightarrow [a]\neg\mathbf{p}$
2. $[a^*]\mathbf{p} \rightarrow [a]\mathbf{p}$
3. $\mathbf{p} \wedge [a]\mathbf{p} \rightarrow [a^*]\mathbf{p}$

1.4.4. Summary In broad strokes, we proved the completeness of PDL and its exponential-time decidability as follows. Given a formula φ, we considered throughout PD structures whose states are boolean combinations of $C(\varphi)$. For completeness we constructed a PD structure $\mathcal{K}_\varphi$, defining $s \xrightarrow{a} t$ by

$$(\forall\, ([a]\xi) \in C(\varphi).\quad t \vdash \neg\xi \quad \Rightarrow \quad s \vdash \neg[a]\xi$$

We referred heavily to provability in PDL particularly in the proof of Proposition 1.3.9(4). For that proof to work we only need to know that all the consistent states are present (so that Lemma 1.3.8 is available); we referred to all possible states to keep matters as simple as possible.

To obtain exponential-time decidability, we modified the construction, trying to replace conditions and arguments that refer to provability by easy-to-check conditions on the syntactic objects present. First, we observed that in the structure $\mathcal{M}_\varphi$ defined using only consistent states, the definition of $s \xrightarrow{a} t$ can be rephrased in structural terms:

$$(\forall\, ([a]\xi) \in C(\varphi).\quad s \ni ([a]\xi) \quad \Rightarrow \quad t \ni \xi$$

However, identifying the consistent states refers to PDL provability, so we looked for a way to narrow attention to consistent states through structural properties. We started by dismissing the incompatible states, i.e. those that are obviously inconsistent. We then pretended that we already have $\mathcal{M}_\varphi$, and argued (Proposition **??**) that $s \models \psi$ iff $s \ni \psi$, for all $\psi \in C(\varphi)$. The proof is analogous to the proof of Proposition 1.3.9, except that part (4) of 1.3.9 is replaced by the witnessability condition, which is declared by fiat.

If all compatible states happen to be consistent (which is the case for some formulas φ), then the witnessability condition holds, and the construction yields $\mathcal{M}_\varphi$. In general, we can check for the witnessabiltiy condition; the first time it fails, at some state s, we can show that s is inconsistent. So we can dismiss s and start over. This cycle can repeat itself at most as many times as there are (compatible) states, and in the end no state is left that volates the witnessing condition. By Lemma 1.4.4 we know that the PD structure obtained is precisely $\mathcal{M}_\varphi$, and the validity of φ can be tested by testing that $\varphi \in s$ for all states s.

Note that this process of constructing $\mathcal{M}_\varphi$ by successive approximations does not supplant our earlier proof of the completeness theorem 1.3.1, but builds on it: we still need to conclude that if φ is not valid in $\mathcal{M}_\varphi$ then it is not provable from the axioms of PDL.

1.5 Exponential lower bound for decidability

1.5.1. Alternating Turing machines

We wish to reduce a decision problem complete for exponential time to the validity problem for PD formulas. We choose the acceptance problems for alternating Turing machines (TMs): it is both easy to reduce to PDL validity, and closely related to the definition of exponential-time completeness, since exponential time TMs are easy to simulate by alternating polynomial time machines [**?**].

A nondeterministic TM, used as an acceptor, accepts a word w when some computation sequence with w as input accepts. Alternatively, we can say that a configuration of the machine is accepting exactly when one of its successor configuration is accepting. We can adopt a dual definition of acceptance, whereby w is accepted if *all* computation sequences with w as input terminate with an accepting state. Equivalently, we would say that a configuration is accepting if each of its successor configuration is accepting. In an alteranting TM the states are classified into accepting, rejecting, conjunctive and disjunctive. A configuration with a conjunctive state accepts when all of its successor configurations accept, and one with a disjunctive state accepts if one these eccepts.

We start with a definition of (linearly-bounded) alternating TMs, whose particular details we'll need for the reduction. We will have in mind devices operating on words. Unlike general Turing machines, ours will not claim new space in the course of computation; in practice this might be achieved by using symbols for delimiters, say $\vdash$ and $\dashv$, and refraining from moving past $\vdash$ to the left or past $\dashv$ to the right. Also, we restrict attention to machines with the following uniformity condition: when in a conjunctive or disjunctive state, the machine will have a choice of exactly two actions. This convention does not affect generality, of course: multiple actions can be acheived by using auxiliary states for intermediate bifurcation, single action is simulated by taking two identical actions (differentiated syntactically by introducing a twin for each state), and no-action is simulated by repeating the given configuration (e.g. by overwriting the scanned symbol by itself).

Let $\mathbf{L}$ and $\mathbf{R}$ be reserved symbols, intended to stand for “move left” and “move right". Fix an alphabet Σ. A (alternating linearly-bounded Turing) *machine* over Σ is a computing device obtained by specifying

- Four finite sets A, R, C, D, disjoint from Σ and from each other.[4] The elements of $Q =_{\rm df} A \cup R \cup C \cup D$ are dubbed the *states*.
- A state s_0, dubbed the *initial state*.
- Functions

$$\begin{array}{rcl} act^+, act^- : (C \cup D) \times \Sigma & \to & \Sigma^+ \\ state^+, state^- : (C \cup D) \times \Sigma & \to & Q \end{array}$$

where $\Sigma^+ =_{\rm df} \Sigma \cup \{\mathbf{L}, \mathbf{R}\}$. The functions act^- and $state^-$ describe the first ("left") possible action of the machine when in state q and scanning symbol σ: perform action $\alpha = act^-(\mathsf{q}, \sigma)$ and switch to state $state^-(\mathsf{q}, \sigma)$. The functions act^+ and $state^+$ describe the corresponding second possible action.

When $\alpha \in \Sigma$, the action α consists in overwriting the symbol by α. When α is $\mathbf{L}$ (respectively, $\mathbf{R}$), the action α is to move the cursor to the left (right). When $act^-(\mathsf{q}, \sigma) = \alpha$ and $state^-(\mathsf{q}, \sigma) = \mathsf{q}'$, we write $\mathsf{q}\sigma \Rightarrow \alpha\mathsf{q}'$. Similarly for act^+ and $state^+$.

For a machine M, a *configuration* is a string of the form $x\mathsf{q}\sigma y$, where $\mathsf{q} \in Q$, $\sigma \in \Sigma$, and $x, y \in \Sigma^*$. The relation $\Longrightarrow_M$ between configurations is generated by the following closure conditions:

- If $\mathsf{q}\sigma \Rightarrow \sigma'\mathsf{q}'$, $\sigma' \in \Sigma$, then $x\mathsf{q}\sigma y \Longrightarrow_M x\mathsf{q}'\sigma' y$.
- If $\mathsf{q}\sigma \Rightarrow \mathbf{L}\mathsf{q}'$, then $x\mathsf{q}\sigma\sigma' y \Longrightarrow_M x\sigma\mathsf{q}\sigma' y$
- If $\mathsf{q}\sigma \Rightarrow \mathbf{R}\mathsf{q}'$, then $x\sigma\mathsf{q}\sigma' y \Longrightarrow_M x\mathsf{q}'\sigma\sigma' y$

The set of *accepted configurations* is generated by the following closure conditions.

- A configuration c whose state is in A is accepted.
- If the state of c is in C, and c' is accepted for all $c' \Longrightarrow_M c$, then c is accepted.
- If the state of c is in D, and c' is accepted for some $c \Longleftarrow_M c'$, then c is accepted.

The machine M *accepts* a (non-empty) string $x \in \Sigma^*$ if $\mathsf{s}_0 x$ is an accepted configuration.[5]

[4] The letters are chosen to correspond to *accepting, rejecting, conjunctive, and disjunctive.*

[5] This definition is over-simplified for actual use; for example, the empty string is never accepted. It is thus natural to define the language accepted by M to consist of those $w \in \Sigma$ for which the extended string $\vdash w \dashv$ is accepted, where the two delimiters are not in Σ, and M is over the alphabet However, our simplified definition obviously suffices to yield an exponential-time complete problem.

LEMMA **1.5.1** *The problem of deciding, given a machine M over Σ as above, and $w \in \Sigma^*$, whether M accepts w, is complete for the class of decision problems solvable in time $2^{O(n)}$, i.e. in time $\leq 2^c n$ for some c.*

1.5.2. Coding machines by PD formulas

Given M and $w \in \Sigma^*$ as above, we construct a PD formula $\varphi \equiv \varphi_{M,w}$, so that M accepts w iff φ is valid. The formula φ will include a characterization of the computational behavior of M, with the configurations of M as states of the intended PD structure.

Let n be the length of w. For each $i = 1 \ldots n$ and $\sigma \in \Sigma \cup Q$, let $\begin{bmatrix} i \\ \sigma \end{bmatrix}$ be a propositional identifier. The intent is that $\begin{bmatrix} i \\ \sigma \end{bmatrix}$ denotes the statement "the i'th symbol of the current configuration is σ." We shall also refer to a propositional identifier `accept`. We use just one atomic-program identifier a, intended to correspond to $\Longrightarrow_M$.

A simple but essential properties of a configuration is that no two symbols occupy the same position. We intend this to hold in all states (i.e. configurations of M), which is captured by the formula

$$NoDup \equiv_{\mathrm{df}} \bigwedge_i \bigwedge_{\substack{a,b \in \Sigma^+ \\ a \neq b}} \begin{bmatrix} i \\ a \end{bmatrix} \rightarrow \neg \begin{bmatrix} i \\ b \end{bmatrix}$$

(Other important properties of configurations, such as having every position occupied by *some* symbol, and having exactly one state per configuration, will fall out from the specific description of the initial configuration.)

We use the notation $\begin{bmatrix} i_1 & i_2 \\ \sigma_1 & \sigma_2 \end{bmatrix}$ as an abbreviation for the conjunction $\begin{bmatrix} i_1 \\ \sigma_1 \end{bmatrix} \wedge \begin{bmatrix} i_2 \\ \sigma_2 \end{bmatrix}$, and similarly for longer conjunctions. In particular, if $C = a_1 \cdots a_{n+1}$ is a configuration of M, (C) will stand for $\begin{bmatrix} 1 & \cdots & n+1 \\ a_1 & \cdots & a_{n+1} \end{bmatrix}$. It will also be convenient to let $\begin{bmatrix} 0 & 1 & \cdots \\ a & b & \cdots \end{bmatrix}$ stand for $\begin{bmatrix} 1 & \cdots \\ b & \cdots \end{bmatrix}$ (as the 0-th position does not exit).

The core of the PDL simulation of the operational behavior of M can then be captured by the following abbreviations. Writing q' for $\mathit{state}^-(\mathsf{q},\sigma)$, we define

$$\begin{bmatrix} i-1 & i & i+1 \\ \tau & \mathsf{q} & \sigma \end{bmatrix}^- \equiv_{\mathrm{df}} \begin{cases} \begin{bmatrix} i-1 & i & i+1 \\ \tau & \mathsf{q}' & \sigma' \end{bmatrix} & \text{if} \quad \mathit{act}^-(\mathsf{q},\sigma) = \sigma' \\ \begin{bmatrix} i-1 & i & i+1 \\ \tau & \sigma & \mathsf{q}' \end{bmatrix} & \text{if} \quad \mathit{act}^-(\mathsf{q},\sigma) = \mathbf{R} \\ \begin{bmatrix} i-1 & i & i+1 \\ \psi' & \tau & \sigma \end{bmatrix} & \text{if} \quad \mathit{act}^-(\mathsf{q},\sigma) = \mathbf{L} \end{cases}$$

The definition of $\begin{bmatrix} i-1 & i & i+1 \\ \tau & \mathtt{q} & \sigma \end{bmatrix}^{+}$ is analogous. Using these abbreviations, we can describe succinctly the effects of transitions on configurations:

$$\begin{array}{lll} \mathit{Act}_{q,\sigma,i,\tau} & \equiv_{\mathrm{df}} & \begin{bmatrix} i-1 & i & i+1 \\ \tau & \mathtt{q} & \sigma \end{bmatrix} \\ & & \quad \rightarrow \quad \langle a \rangle \begin{bmatrix} i-1 & i & i+1 \\ \tau & \mathtt{q} & \sigma \end{bmatrix}^{-} \\ & & \qquad \wedge \quad \langle a \rangle \begin{bmatrix} i-1 & i & i+1 \\ \tau & \mathtt{q} & \sigma \end{bmatrix}^{+} \\ & & \qquad \wedge \quad [a] \left(\begin{bmatrix} i-1 & i & i+1 \\ \tau & \mathtt{q} & \sigma \end{bmatrix}^{-} \vee \begin{bmatrix} i-1 & i & i+1 \\ \tau & \mathtt{q} & \sigma \end{bmatrix}^{+} \right) \\ \mathit{Act} & \equiv_{\mathrm{df}} & \bigwedge_{q \in C \cup D} \bigwedge_{\sigma,\tau \in \Sigma} \bigwedge_{i} \mathit{Act}_{q,\sigma,i,\tau} \end{array}$$

To state that no symbol is changed if it is not a state or a neighbor of a state we define the formula

$$\mathit{Unchanged} \quad \equiv_{\mathrm{df}} \quad \bigwedge_{i} \left(\mathit{Safe}_i \rightarrow \bigwedge_{\sigma \in \Sigma} \left(\begin{bmatrix} i \\ \sigma \end{bmatrix} \rightarrow [a] \begin{bmatrix} i \\ \sigma \end{bmatrix} \right) \right)$$

where

$$\mathit{Safe}_i \ \equiv_{\mathrm{df}} \ \bigwedge_{q \in Q} \ \bigwedge_{j = i-1, i, i+1} \neg \begin{bmatrix} j \\ q \end{bmatrix}$$

We move on to describe the conditions for acceptance. There is one condition for each of the four kinds of states (accepting, rejecting, conjunctive and disjunctive):

$$\begin{array}{ll} \bigwedge_{q \in A} \bigwedge_{i} & \left(\begin{bmatrix} i \\ q \end{bmatrix} \rightarrow \mathtt{accept} \right) \\ \bigwedge_{q \in R} \bigwedge_{i} & \left(\begin{bmatrix} i \\ q \end{bmatrix} \rightarrow \neg\mathtt{accept} \right) \\ \bigwedge_{q \in C} \bigwedge_{i} & \left(\begin{bmatrix} i \\ q \end{bmatrix} \rightarrow (\mathtt{accept} \leftrightarrow [a]\, \mathtt{accept}) \right) \\ \bigwedge_{q \in D} \bigwedge_{i} & \left(\begin{bmatrix} i \\ q \end{bmatrix} \rightarrow (\mathtt{accept} \leftrightarrow \langle a \rangle\, \mathtt{accept}) \right) \end{array}$$

We write *Acceptance* for the conjunction of all formulas above.

Now let

$$\begin{array}{lll} \varphi_{M,w} & \equiv_{\mathrm{df}} & [a^*] \ (\mathit{NoDup} \wedge \mathit{Act} \wedge \mathit{Unchanged} \wedge \mathit{Acceptance}) \\ & & \quad \rightarrow \quad ((C_0) \rightarrow \mathtt{accept}) \end{array}$$

where C_0 is the initial configuration.

1.5.3. Correctness of the reduction

LEMMA **1.5.2** *M accepts $w \in \Sigma^*$ iff* $\models \varphi_{M,w}$.

Proof. From the definitions we have (Exercise) that $C \Rightarrow_M C'$ implies

$$\models \; \mathit{NoDup} \wedge \mathit{Act} \wedge \mathit{Unchanged} \quad \rightarrow \quad ([C] \rightarrow \langle a\rangle[C']) \tag{1.5.1}$$

The converse is also true: consider the PD structure $\mathcal{K}_M$ whose states are the configurations of M, with $\xrightarrow{a}$ interpreted as $\Longrightarrow_M$, and the propositional identifiers $\begin{bmatrix} i \\ \sigma \end{bmatrix}$ and `accept` given their intended meaning. In $\mathcal{K}_M$ the formulas *NoDup*, *Act* and *Unchanged* are true in all states, so $[C] \rightarrow \langle a\rangle[C']$ is true at all states, in particular at state C. But $[C]$ is true at C, and the only state where $[C']$ is true is C'. Thus $C \Longrightarrow_M C'$.

Using 1.5.1 we can show that a configuration C of M is accepting iff

$$\models [a^*] \; (\; \mathit{NoDup} \wedge \mathit{Act} \wedge \mathit{Unchanged} \wedge \mathit{Acceptance}) \quad \rightarrow \quad ((C) \rightarrow \mathtt{accept})$$

The forward direction is proved by induction on the height of the computation tree for M that starts with configuration C (Exercise). The backward direction invokes the PD structure $\mathcal{K}_M$ above. In particular, M accepts w, i.e. the initial configuration C_0 is accepting, iff $\models \varphi_{M,w}$. ⊣

1.5.4. Exponential time completeness of PDL

THEOREM **1.5.3** *The validity problem for PDL formulas is complete (under log-space reductions) for the class EXP-TIME, i.e. the problems decided in time* $2^{O(n^*)}$.

1.6 PDL for non-regular programs

1.6.1. Programs as sets of sequential programs

Consider a PD structure $\mathcal{K} = (S, [\![\cdot]\!])$, with set A of atomic actions. We call a program over A *sequential* if it is defined from A and `skip` using composition only. Each a program $a_1; \cdots ; a_n$ $(n \geq 0)$ can be identifier with the string $a_1 \cdots a_n \in A^*$.

If $s \xrightarrow{\alpha} t$ holds for some program α, then in fact $s \xrightarrow{w} t$ for some sequential $w \in A^*$. The program α can be viewed as a set $\mathit{Seq}(\alpha)$ of such sequential programs. For a test-free α the set $\mathit{Seq}(\alpha)$ is simply the regular subset of A^* denoted by α as a regular expression:

$$\begin{array}{rcll} Seq(\mathtt{skip}) & = & \{\epsilon\} & \\ Seq(a) & = & \{a\} & \text{for } a \text{ atomic} \\ Seq(\alpha;\beta) & = & (Seq(\alpha))(Seq(\beta)) & \text{language concatenation;} \\ Seq(\alpha\cup\beta) & = & Seq(\alpha)\cup Seq(\beta) & \\ Seq(\alpha^*) & = & Seq(\alpha)^* & \text{Kleene's * on languages} \end{array}$$

Thus, a test-free regular program α can be identified with the regular set $Seq(\alpha)$ of sequential programs.

We can also extend the definition of PD formulas to accommodate notations for other sets of sequential programs: if L is a notation for a set of programs, then, in a PD structure $\mathcal{K}$, we define $s \xrightarrow{L} t$ to hold exactly when $s \xrightarrow{\alpha} t$ for some α in (the set denoted by) L. The semantics of formulas of the form $[L]\varphi$ and $\langle L\rangle\varphi$ is then defined exactly as for $[\alpha]\varphi$ formulas $\langle\alpha\rangle\varphi$. Note that for a test-free regular program α, $s \xrightarrow{Seq(\alpha)} t$ exactly when $s \xrightarrow{\alpha} t$; therefore $[Seq(\alpha)]\varphi$ is equivalent to $[\alpha]\varphi$.

1.6.2. Sequential programs as states

In general, there is only a loose correspondence between sequential programs and pairs of states in a PD structure $\mathcal{K}$: there can be many sequential programs leading from state s_0 to state s, and $s_0 \xrightarrow{\alpha} s$ may hold for many states s (if atomic actions are non-deterministic). A useful construction eliminate this ambiguity.

Let Σ be an alphabet. The PD structure $\mathcal{K}_\Sigma = (\Sigma^*, [\![\cdot]\!])$ over actions Σ and a single propositional identifier $\mathbf{p}$, is defined by

- $u \xrightarrow{\sigma} v$ iff $u = \sigma v$ (note that the transition is from the longer to the shorter word), and
- $\mathbf{p}$ is true just at the empty word ϵ.

LEMMA **1.6.1** *Let $L \subseteq \Sigma^*$. Then $\mathcal{K}_\Sigma, w \models \langle L\rangle\mathbf{p}$ iff $w^R \in L$.*

1.6.3. New expressiveness of non-regular programs

PROPOSITION **1.6.2** *Suppose $L \subset \Sigma^*$ is a non-regular language. There is no PD formula φ equivalent to $\langle L\rangle\mathbf{p}$.*

Proof. Consider the PD structure $\mathcal{K}_\Sigma$.

Claim 1. For every PD formula ψ, the set

$$[\![\psi]\!] \quad\equiv\quad [\![\psi]\!]_{\mathcal{K}_\Sigma} \quad =_{\mathrm{df}} \quad \{w \in \Sigma^* \mid \mathcal{K}_\Sigma, w \models \psi\}$$

is regular. We prove this simultaneously with

Claim 2. For every PD program α there is a finite collection χ_i $(i \in I)$ of formulas used as tests in α, and regular sets $L_\alpha, L_{\alpha,i} \subseteq \Sigma^*$ $(i \in I)$, such that, for all formulas ξ,

$$[\![[\alpha]\xi]\!] \quad = \quad [\![\xi]\!]\, L_\alpha \quad \cup \quad \bigcup_i -[\![\chi_i]\!]\, L_{\alpha,i}$$

The two claims are proved together by structural induction on ψ and α, using the closure conditions of the class of regular sets (Exercise).

Now suppose that $\models \langle L\rangle\mathbf{p} \;\leftrightarrow\; \varphi$, for some PDL formula φ. Then in particular, $\mathcal{K}_\Sigma, w \models \langle L\rangle\mathbf{p} \;\leftrightarrow\; \varphi$ for all $w \in \Sigma^*$. But the set of w's where $\mathcal{K}_\Sigma, w \models \langle L\rangle\mathbf{p}$ is L, by Lemma 1.6.1, whereas the set of w where $\mathcal{K}_\Sigma, w \models \varphi$ is regular, by Claim 1. This contradicts our assumption about L. $\dashv$

Proposition 1.6.2 shows that adding even one non-regular program L to the syntax of PDL increases its expressiveness. More precisely, if two logics $\mathbf{L}_1, \mathbf{L}_2$ share the same notion of semantics, then $\mathbf{L}_1$ is said to be *interpretable* in $\mathbf{L}_2$ if for every formula φ of $\mathbf{L}_1$ there is a formula φ' of $\mathbf{L}_2$ whose semantics is identical to that of $\mathbf{L}_1$. For PDL formulas, semantic-identity means being true at exactly the same PD structures and states therein. If $\mathbf{L}_1$ is interpretable in $\mathbf{L}_2$, but not vice-versa, we say that $\mathbf{L}_2$ is *more expressive* than $\mathbf{L}_1$.

EXERCISES **1.6.1** Prove by induction the Claims in the proof of Proposition 1.6.2.

1.6.2 Let *union-free-PDL* be PDL with no union operation $\cup$ in programs. Show that PDL is not more expressive than PDL-no-union. Thus, adding constructs to a PD logic does not necessarily increase its expressiveness. [Hint: Start by showing that every regular program is equivalent to the union of union-free programs.]

1.6.3 Let *PDL-with-intersection* be an extension of PDL with program intersection: if α and β are programs, then so is $\alpha \cap \beta$, with the semantics $[\![\alpha \cap \beta]\!] =_{\mathrm{df}} [\![\alpha]\!] \cap [\![\beta]\!]$. Show that PDL-with-intersection is not more expressive than PDL.

1.6.4. PDL with context free programs

A natural class of potentially non-regular programs is that of context free languages: in formal language theory it is the most obvious extension of the class of regular languages, and in programming language theory it corresponds to (non-parametric) recursive procedures, a crucial high-level programming construct. Let PDL(CFG) be the extension of PDL, in which (G) is a program for every context-free grammar G.

PROPOSITION **1.6.3** *PDL(CFG) is undecidable.*

Proof. If two CF grammars G and G' are equivalent (i.e. generate the same language) then, trivially,

$$\models \langle (G) \rangle\, \mathbf{p} \leftrightarrow \langle (G') \rangle\, \mathbf{p} \tag{1.6.1}$$

Conversely, if 1.6.1, then the equivalence holds in particular in every state of $\mathcal{K}_\Sigma$. By Lemma 1.6.1 it follows that the G and G' define the same language.

Thus, the problem of equivalence for CF grammars is reduced to the validity problem for PDL(CFG). Since the former is undecidable (see e.g. [**?**]), it follows that so is the latter. $\dashv$

Note that the decision problem is, by necessity, formulated in terms of the particular syntactic representation of CF languages, here by CF grammars. Of course, the same result would hold for any common representation, such as push-down automata. However, it would not hold under certain unusual representations. For instance, if we admit a CF grammar G as a program only if G is not equivalent to any grammar that precedes it under lexicographically ordering, then each CF language is represented by only one such grammar, and grammar equivalence is trivialized to syntactic identity, an easily decidable property. (Of course, identifying the legal programs is now undecidable.)

The undecidability result of 1.6.3 leaves open two questions: how bad is the undecidability of PDL(CFG), and how complex need the grammars be to get undecidability.

Both questions are answered by the following result, which shows that augmenting PDL even by very simple CFG's yields very high level of undecidability. Let[6]

$$a_1 b_1^\Delta a_2 b_2^\Delta \cdots a_k b_k^\Delta \quad =_{\mathrm{df}} \quad \{a_1 b_1^i a_2 b_2^i \cdots a_k b_k^i \mid i \geq 0\}$$

PROPOSITION **1.6.4** *PDL + $a^\Delta b a^\Delta$ is undecidable. In fact, it remains undecidable even if supplied with an oracle definable by a first order formula (that refers to strings, natural numbers, and computable functions).*[7]

Here are some additional interesting results about the decision problem for PDL extended with CF languages and related languages.

1. The logic PDL + $a^\Delta b^\Delta$ + $b^\Delta a^\Delta$ is undecidable, just as badly as the logic PDL + $a^\Delta b a^\Delta$.
2. In contrast, the logic PDL + $a^\Delta b^\Delta$ is decidable.
3. Moreover, the logic PDL + $a^\Delta b^\Delta c^\Delta$ is decidable. Recall that $a^\Delta b^\Delta c^\Delta$ is not a context-free language!
4. The logic PDL + a^{2^*} is undecidable. (Here $a^{2^*} =_{\mathrm{df}} \{a^{2^i} \mid i \geq 0\}$.)
5. It is not known whether PDL remains decidable when extended with a^{*^2}, though some restricted versions are known to be decidable. It is also not known whether

[6] The symbol Δ is used here because insisting on the identity of two indices is what defines the diagonal of a matrix.

[7] More technically, the decision problem is complete for the class Π^1_1.

PDL + a^{*^3} is decidable; if it is, then the decision procedure would decide some open problems in number theory, such as whether every number $> 10,000$ is the sum of 5 cubes. This question is equivalent to the validity of the formula

$$[a^{*^3}][a^{*^3}][a^{*^3}][a^{*^3}][a^{*^3}]\,\mathbf{p} \quad \leftrightarrow \quad [a^{10001}][a^*]\,\mathbf{p}$$

6. Let $\mathit{Fib} =_{\rm df} \{a^k \mid k \text{ a Fibonacci number }\}$. The logic PDL + *Fib* is undecidable.

1.6.5. The finite model property

Recall that the Finite Model Property for a PD logic **L** states that every non-valid formula fails already in some finite model. A slightly stronger condition is the Computably-bounded Model Property: there is a total computable functions B, such that every non-valid formula φ fails in some model of size $\leq B(|\varphi|)$. If **L** satisfies this property, then the following algorithm decides the validity problem for **L**: given a formula φ of **L**, compute $B(|\varphi|)$, and check φ in all states of all PD structures of size $\leq B(|\varphi|)$. It follows that the extensions of PDL that are undecidable, cannot have the Computably-bounded Model Property, and are likely to also not have the Finite Model Property.

What about *decidable* extensions of PDL? In fact, even these may fail to have the finite model property.

PROPOSITION **1.6.5** *The logic PDL + $a^\Delta b^\Delta$ does not have the finite model property.*

Proof. Consider the formula

$$\varphi \quad \equiv_{\rm df} \quad \mathbf{p} \;\wedge\; [a^*]\langle a;b^*\rangle\mathbf{p} \;\wedge\; [\alpha;b;a;\alpha]\neg\mathbf{p} \;\wedge\; [a^+][a^\Delta b^\Delta]\neg\mathbf{p} \;\wedge\; [a^\Delta b^\Delta][b^+]\neg\mathbf{p}$$

where $\alpha =_{\rm df} (a\cup b)^*$, $a^+ = (a;a^*)$, $b^+ = (b;b^*)$.

It is easy to construct a PD model of φ: Define $\mathcal{K} = (\{a,b\}^*, [\![\cdot]\!])$ by

- $u \xrightarrow{a} v$ iff $v = av$
- $u \models \mathbf{p}$ iff $u \in a^\Delta b^\Delta$.

Then $\mathcal{K}, \epsilon \models \varphi$.

In fact, if $\mathcal{K}$ is any PD structure, and $\mathcal{K}, s \models \varphi$ for some state s in $\mathcal{K}$, then the set

$$P \quad =_{\rm df} \quad \{w \in \{a,b\}^* \mid \mathcal{K}, s \models \langle w\rangle\mathbf{p}\}$$

is exactly $a^\Delta b^\Delta$: the first two conjuncts of φ gurantee that P contains a word in $a^i b^*$, for each $i \geq 0$ (by induction on i), the third excludes $w \notin a^*b^*$, and the last two exclude $w \in a^*b^* - a^\Delta b^\Delta$.

But if $\mathcal{K}$ is finite, then we can consider $\mathcal{K}$ as a finite automaton over $\{a,b\}^*$, with s as initial state, and the states where $\mathbf{p}$ is true as accepting states. The set accepted by this automaton is precisely P, contradicting the fact that $P = a^{\triangle}b^{\triangle}$ is not regular. $\dashv$

Additional examples of decidable (or probably decidable) extensions of PDL which fail the finite model property can be found in [**?**, Chapter 9].

1.6.6. Additional problems

EXERCISES **1.6.4** Given a PD structure $\mathcal{K} = (S, [\![\cdot]\!])$ over actions A, a state $s \in S$, and a PDL formula φ, define

$$L(s \to \varphi) \quad =_{\mathrm{df}} \quad \{w \in A^* \mid \mathcal{K}, s \models \langle w \rangle \varphi\}$$

Show that if $\mathcal{K}$ is finite, then $L(s \to \varphi)$ is regular.