The Gamut of Dynamic Logics*

Jan van Eijck† and Martin Stokhof‡

15th July 2005

Abstract

Dynamic logic, broadly conceived, is the logic that analyses change by decomposing actions into their basic building blocks and by describing the results of performing actions in given states of the world. The actions studied by dynamic logic can be of various kinds: actions on the memory state of a computer, actions of a moving robot in a closed world, interactions between cognitive agents performing given communication protocols, actions that change the common ground between speaker and hearer in a conversation, actions that change the contextually available referents in a conversation, and so on.

In each of these application areas, dynamic logics can be used to model the states involved and the transitions that occur between them. Dynamic logic is a tool for both state description and action description. Formulae describe states, while actions or programs express state change. The levels of state descriptions and transition characterisations are connected by suitable operations that allow reasoning about pre- and postconditions of particular changes.

From a computer science perspective, dynamic logic is a formal tool for reasoning about programs. Dynamic logics provides the means for formalising correctness specifications, for proving that these specifications are met by a program under consideration, and for reasoning about equivalence of programs. From the perspective of the present paper, this is but one of many application areas. We will also look at dynamic logics for cognitive processing, for communication and information updating, and for various aspects of natural language understanding.

Contents

* To appear in D. Gabbay & J. Woods (eds), *The Handbook of History of Logic. Volume 6 – Logic and the Modalities in the Twentieth Century*, Elsevier, Amsterdam
† Centrum voor Wiskunde en Informatica, Amsterdam
‡ ILLC / Department of Philosophy, Universiteit van Amsterdam

1 Introduction

Notions involving change often have a dual character, an interplay between process and product. While travelling from one place to another, one can either focus on the process of 'being on the road' or on the result of this process, 'being somewhere else'. Intellectual activities also have this dual nature: *scientific discovery* denotes a process of reaching for new insights but also the resulting insights, *judgement* denotes both the process of reaching a rational decision and the decision that results from that process, *computation* involves a process of stepwise changes, and the outcome of such a process, and so on.

The logical study of the interplay between process and product is called dynamic logic. This paper gives an overview of various systems of dynamic logic, with illustrations drawn from various application areas: programming, communicative action and interaction, cognitive processing, natural language understanding. It is aimed at researchers who have an interest in the formal analysis of computational and communicative processes. A more extended textbook introduction to dynamic logics that is explicitly geared to computer science is the informative [63]. An earlier overview is [62]. Cf., also [15] for an introduction that focuses on cognitive applications.

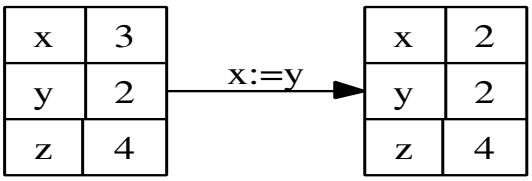
Dynamic logic can be viewed as dealing with the logic of action and the result of action, and it can be used to model various kinds of actions and their results. A rough classification might be the following. First of all there are *computations*, i.e., actions performed on computers. Examples are computing the factorial function, computing square roots, etc. Such actions typically involve changing the memory state of a machine. Another type of action is that of *communicative actions*, such as reading an English sentence and updating one's state of knowledge accordingly, engaging in a conversation, sending an email with cc's, telling one's husband a secret. These actions typically change the cognitive states of the agents involved. And then there are *actions in the world*, such as building churches, destroying bridges, spilling milk. Such actions change the state of the world. Of course there are connections between these categories and actions of a mixed nature: a communicative action will usually involve some computation involving memory, and the utterance of an imperative is a communicative action that aims at an action in the world.

For a researcher who is interested in the formal analysis of actions of various kinds dynamic logic can be viewed as a tool box: it provides concepts and methods for description of actions and means to characterise the properties of the resulting systems. Using these tools the researcher can then develop specialised, tailored systems for dealing with specific kinds of actions: logics of computation, logics of communication, logics of action. Inasmuch as they are geared toward specific applications such systems may differ quite widely, but in many cases their core can nevertheless be characterised formally in a uniform way: many of these logics can be related to one or more varieties of modal logic,

taken in a suitably broad sense, viz., as the logic of 'labelled transition systems'.

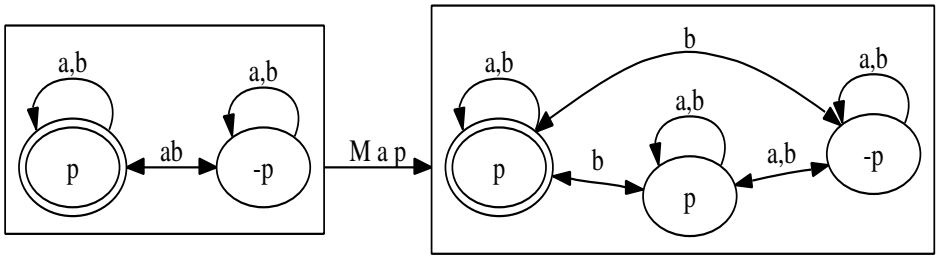
A labelled transition system (or LTS, or multi-modal Kripke model) over signature P, A, with P a set of propositions and A a set of actions, is a triple $\langle S, V, R\rangle$ where S is a set of states, $V : S \to \mathcal{P}(P)$ is a valuation function, and $R = \{\stackrel{a}{\to} \subseteq S \times S \mid a \in A\}$ is a set of labelled transitions, i.e., binary relations on S, one for each label a. Let us illustrate the idea of an LTS by a few simple examples.

If one interprets the labelled transitions as the changes in the memory state of a computer, LTSs model computations, for example the simple assignment $x := y$:



The command to put the value of register y in register x makes the contents of registers x and y equal. Pioneer papers in the logic of computation are [43, 70].

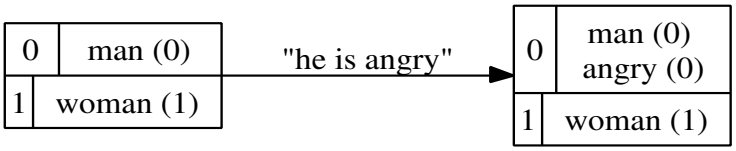
If one interprets the labelled transitions as accessibility relations on the cognitive state space of a group of agents, LTSs can be used to model the information that such agents have about the world, about each other's information about the world, each other's information about each other's information about the world, and so on. And it can be used to describe changes in such information states:



On the left is an epistemic situation where p is in fact the case (indicated by a double circle), but a and b cannot distinguish between p and $\neg p$. If in such a situation a receives the message that p is the case, while b is not informed of this, the epistemic situation changes to what is pictured on the right. In the new situation, a knows that p, and a also is aware of the fact that b does not know, while b still does not not know, and b still assumes that a does not know. See [68] for one of the earliest treatments of epistemic logic along these lines. An overview of the development of epistemic logic is given in [48]. Cf., also [15].

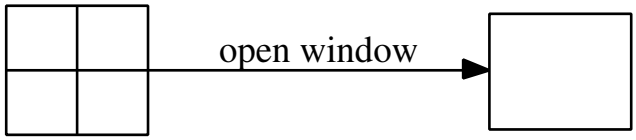
Communicative actions may provide more detailed information about the world than the information that a certain state of affairs is realised. In a discourse

(text, conversation), information is (often) conveyed piecemeal, and languages contains various means for keeping track of what has been said about what. Anaphoric pronouns are a case in point. Their role can be modelled by interpreting states as consisting of discourse items to which information is added in an incremental fashion. The following illustrates the action on such a state that is triggered by the use of an anaphoric pronoun:



In a discourse where a man and a woman have been mentioned recently, an utterance of 'He is angry' receives a natural interpretation by linking the pronoun to the most salient appropriate discourse item, viz., the man that was just mentioned. Early work in this area is in [65, 79, 83]. See [47] for an overview.

Yet another illustration of how LTSs can be used to model action is when one interprets labelled transitions as actions on the state of the world. In that case LTSs model changes in the world itself:



The action of window-opening changes a state in which the window is closed into one in which it is open. More complex actions call for more complex models, of course, in particular when we are interested in a more fine grained analysis of the causality involved in bringing about changes. An early overview of the logic of action is in [134]. For a more recent survey, cf., [114]. A different approach is the *stit*-logic of Belnap, cf. [12].

These examples illustrate that it is possible to approach a wide variety of kinds of actions from a unified perspective. What follows is intended to show that this is not only possible, but also fruitful. Note that the diversity of applications of dynamic logic also indicates that it is difficult to trace the various systems and application to a single historic root. In fact, some of what appears uniform now, as a matter of historical fact had quite diverse origins. For this reason we have opted for a mainly systematic treatment, with occasional historical side remarks where relevant.

*

The larger part of the survey of dynamic logic that follows is devoted to an exposition of two core systems of dynamic logic, viz., *propositional dynamic logic* and *quantificational dynamic logic*, and three illustrative areas of application,

viz., programming, communicative action and dynamic semantics of natural language.

One of the seminal papers in computer science is Hoare's [70]. where the following notation is introduced for specifying what an imperative program does:

$$\{P\} \quad C \quad \{Q\}.$$

Here C is a program from a formally defined programming language for imperative programming, and P and Q are conditions on the programming variables used in C. Statement $\{P\}\ C\ \{Q\}$ is true if whenever C is executed in a state satisfying P and if the execution of C terminates, then the state in which execution of C terminates satisfies Q. The 'Hoare-triple' $\{P\}\ C\ \{Q\}$ is called a partial correctness specification; P is called its precondition and Q its postcondition. Floyd-Hoare logic, as the logic of reasoning with such correctness specifications is called, is the precursor of all the dynamic logics known today. We will demonstrate Floyd-Hoare logic in Section 2.4, for the toy language specified in Section 2.1. The specification of a toy programming language has the additional benefit that it will allow us to demonstrate various approaches to the semantics of programming. We will present example programs, formulate questions about their behaviour, and show how some of these questions are answered with Floyd-Hoare logic. After that, we turn to dynamic logic proper as a more general means of tackling such questions.

In section 3 we present what is perhaps the most basic system of dynamic logic, propositional dynamic logic (PDL), a logic in which basic actions are primitives. This feature makes PDL applicable in a wide variety of cases. For example, if one interprets the basic actions as communicative actions that effect cognitive states of sets of interacting agents, then dynamic logic takes the shape of dynamic epistemic logic. This important area of application is treated in detail in section 4.

When one takes memory change as the basic action, one gets quantified dynamic logic (QDL), the system that is introduced and discussed in section 5. QDL has its origin in correctness reasoning based on annotating programs with pre- and postconditions. These historical connections are briefly traced. It is possible to interpret QDL programs also in a different way, viz., as changing the cognitive state of a language user. This potential relevance of QDL for an understanding of natural language was actualised in what has been called the 'dynamic turn' in natural language semantics. In section 6 we focus on dynamic predicate logic (DPL) as a subsystem of QDL. A more detailed treatment of the application of dynamic concepts in natural language semantics is given in section 7.

2 Describing Change and Reasoning about Change

Consider the following problem concerning the outcome of a pebble drawing action.

> A vase contains 35 white pebbles and 35 black pebbles. Proceed as follows to draw pebbles from the vase, as long as this is possible. Every round, draw two pebbles from the vase. If they have the same colour, then put a black pebble back into the vase, if they have different colours, then put the white pebble back. You may assume that there are enough additional black pebbles. In every round one pebble is removed from the vase, so after 69 rounds there is a single pebble left. What is the colour of this pebble?

Here is an implementation of this procedure, where the vase is represented as a list of integers, the white pebbles are the occurrences of 0, and the black pebbles the occurrences of 1. The `draw` function is coded in the programming language Haskell [78]:

```
draw :: [Integer] -> [Integer]
draw [x] = [x]
draw (0:0:xs) = draw (1:xs)
draw (1:1:xs) = draw (1:xs)
draw (0:1:xs) = draw (0:xs)
draw (1:0:xs) = draw (0:xs)
```

The question: if this function is called with a list of thirty-five 0's and thirty-five 1's, in unknown order, will the outcome of the function be [0] or [1]?

The key to the solution is finding an *invariant* of the procedure, i.e., finding a condition that does not change when a single pebble is removed from the vase. It is not hard to see that when a pebble is drawn, the number of white pebbles always remains odd. It follows that the last pebble is white. So the `draw` function will return [0] on any permutation of the list of thirty-five 0's and thirty-five 1's.

With this piece of reasoning we are in the realm of dynamic logic. Rather than encode examples in an existing programming language like *Haskell* or *Java*, it will turn out useful to introduce our own toy language for illustrations. As dynamic logic describes the interplay between actions and resulting states, the action description language is part and parcel of the dynamic logic language.

2.1 The WHILE Language

In what follows we define a simple programming language for programming over the data type of the natural numbers, i.e., the set $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$, with functions $+$ for addition, $*$ for product, and $\dot{-}$ for cut-off subtraction.

First, we distinguish between numbers and their names. Numbers are objects in the mathematical realm, names are syntactic objects. A numeral is a name

for a natural number. E.g., '5' is a name for the natural number 5. Assume N is a set of numerals. Assume V is a set of variables. The sets N and V may have further internal structure, but we will not bother to spell this out. Given sets N, V, arithmetic expressions can be defined by means of $+, *, \dot{-}$, as follows (assume n ranges over the numerals and v over the variables):

$$a \quad ::= \quad n \mid v \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 \dot{-} a_2.$$

This says that $345 * (67 + 8)$ and $(345 * 67) + 8$ are arithmetic expressions. (The brackets indicate the manner of construction).

In terms of these arithmetic expressions we will now fix a small programming language for programming with the natural numbers. We assume two further primitive relation symbols '$=$' for 'equal', and '$\leq$' for 'less than or equal'. This allows us to define Boolean expressions (named after [20]), as follows:

$$B \quad ::= \quad \top \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg B \mid B_1 \vee B_2$$

Note that instead of listing equalities $a_1 = a_2$ explicitly, we might have introduced them by way of abbreviation, as shorthand for $a_1 \leq a_2 \wedge a_2 \leq a_1$. Arithmetic expressions and Boolean expressions figure in programming commands, as follows:

$$C \quad ::= \quad \text{SKIP} \mid v := a \mid C_1 \ ; \ C_2 \mid \text{IF } B \text{ THEN } C_1 \text{ ELSE } C_2 \mid \text{WHILE } B \text{ DO } C.$$

The basic programming constructs of the WHILE language are SKIP for the program that does nothing, and $v := a$ for the program that assigns the value of a to the variable v. Programs or commands can be composed by means of sequencing, by means of conditionalisation, and by means of guarded repetition. Further programming constructs can now be defined, e.g., *REPEAT*:

$$\text{REPEAT } C \text{ UNTIL } B \quad := \quad C \ ; \ \text{WHILE } \neg B \text{ DO } C.$$

The WHILE language looks deceptively simple, but it is extremely expressive. In fact, this little language is Turing complete, i.e., one can specify the behaviour of any Turing machine in it ([124]). This means in turn that anything that can be computed on the natural numbers can (in principle) be computed by means of a WHILE program.

2.2 Semantics

To specify the semantics, we take the natural numbers $\mathbb{N}$ with the operations $+, *, \dot{-}$ and the relation $\leq$ as given. We also assume that every numeral n in N has an interpretation $I(n) \in \mathbb{N}$. Let g be a mapping from V to $\mathbb{N}$ (an assignment of natural numbers to the variables). The arithmetic expressions of the language are now interpreted relative to assignment g, as follows:

$$
\begin{array}{rcl}
[\![n]\!]_g & := & I(n) \\
[\![v]\!]_g & := & g(v) \\
[\![a_1 + a_2]\!]_g & := & [\![a_1]\!]_g + [\![a_2]\!]_g \\
[\![a_1 * a_2]\!]_g & := & [\![a_1]\!]_g * [\![a_2]\!]_g \\
[\![a_1 \dot{-} a_2]\!]_g & := & [\![a_1]\!]_g \dot{-} [\![a_2]\!]_g
\end{array}
$$

The semantics of the Boolean expressions (or 'Booleans') of the language is defined as follows:

$$
\begin{array}{rcl}
[\![\top]\!]_g & := & T \\
[\![a_1 = a_2]\!]_g & := & \begin{cases} T & \text{if } [\![a_1]\!]_g = [\![a_2]\!]_g \\ F & \text{otherwise} \end{cases} \\
[\![a_1 \leq a_2]\!]_g & := & \begin{cases} T & \text{if } [\![a_1]\!]_g \leq [\![a_2]\!]_g \\ F & \text{otherwise} \end{cases} \\
[\![\neg B]\!]_g & := & \begin{cases} T & \text{if } [\![B]\!]_g = F \\ F & \text{otherwise} \end{cases} \\
[\![B_1 \vee B_2]\!]_g & := & \begin{cases} T & \text{if } [\![B_1]\!]_g = T \text{ or } [\![B_2]\!]_g = T \\ F & \text{otherwise} \end{cases}
\end{array}
$$

2.2.1 Natural Semantics for Commands

The semantics of the commands can be given in various styles. First we give the so-called natural semantics, in the form of a specification of a transition system.

For any valuation g, any variable v and any natural number d, let $g[v \mapsto d]$ be the valuation g' that differs from g at most in the fact that $g'(v) = d$. This notion is familiar from the semantics of first order logic. Then the transition for assignment commands is given by:

$$
g \xrightarrow{v:=a} g[v \mapsto [\![a]\!]_g]
$$

The SKIP command does nothing:

$$
g \xrightarrow{\text{SKIP}} g
$$

Sequential composition combines two transition arrows:

$$
\frac{g \xrightarrow{C_1} g' \quad g' \xrightarrow{C_2} g''}{g \xrightarrow{C_1 \ ; \ C_2} g''}
$$

Conditional action makes a choice from two transition relations, depending on the evaluation of the condition.

$$\frac{g \xrightarrow{C_1} g'}{g \xrightarrow{\text{IF } B \text{ THEN } C_1 \text{ ELSE } C_2} g'} \; [\![B]\!]_g = T$$

$$\frac{g \xrightarrow{C_2} g'}{g \xrightarrow{\text{IF } B \text{ THEN } C_1 \text{ ELSE } C_2} g'} \; [\![B]\!]_g = F$$

Guarded iteration does nothing if the guard fails to hold:

$$\frac{}{g \xrightarrow{\text{WHILE } B \text{ DO } C} g} \; [\![B]\!]_g = F$$

Otherwise the guarded action is performed and the WHILE command is executed again in the result state.

$$\frac{g \xrightarrow{C} g' \quad g' \xrightarrow{\text{WHILE } B \text{ DO } C} g''}{g \xrightarrow{\text{WHILE } B \text{ DO } C} g''} \; [\![B]\!]_g = T$$

These rules define a transition relation $\xrightarrow{C}$ on the set of all valuations, for every command C. In order to derive a transition $g \xrightarrow{C} g'$, construct a finite derivation tree with $g \xrightarrow{C} g'$ at the root, with axioms at the leaves and each internal nodes licensed by a transition rule. Here is an example, for the command $z := x \;;\; x := y \;;\; y := z$, executed in the state $g = \{x \mapsto 3, y \mapsto 2, z \mapsto 5\}$. We use g_1 as shorthand for $\{x \mapsto 3, y \mapsto 2, z \mapsto 3\}$, g_2 as shorthand for $\{x \mapsto 2, y \mapsto 2, z \mapsto 3\}$, g_3 as shorthand for $\{x \mapsto 2, y \mapsto 3, z \mapsto 3\}$.

$$\frac{\dfrac{}{g \xrightarrow{z:=x} g_1} \qquad \dfrac{\dfrac{}{g_1 \xrightarrow{x:=y} g_2} \qquad \dfrac{}{g_2 \xrightarrow{y:=z} g_3}}{g_1 \xrightarrow{x:=y \;;\; y:=z} g_3}}{g \xrightarrow{z:=x \;;\; x:=y \;;\; y:=z} g_3}$$

This command computes the remainder upon division of x by y in x:

$$\text{WHILE } y \leq x \text{ DO } x := x \dot{-} y.$$

The following variant computes the result of the division of x by y in z, and the remainder in x:

$$z := 0 \;;\; \text{WHILE } y \leq x \text{ DO } (x := x \dot{-} y \;;\; z := z + 1).$$

Abbreviate $\neg a_1 = a_2$ as $a_1 \neq a_2$, $\neg a_1 \leq a_2$ as $a_1 > a_2$ and $\neg a_1 \geq a_2$ as $a_1 < a_2$. Euclid's well known Greatest Common Divisor algorithm is now readily expressed as a WHILE command. The following program computes the GCD of x and y in x (and in y).

$$\text{WHILE } x \neq y \text{ DO IF } x > y \text{ THEN } x := x \dot{-} y \text{ ELSE } y := y \dot{-} x. \qquad (1)$$

For state $g = \{x \mapsto 24, y \mapsto 9\}$, program (1) leads to the following execution:

$$\begin{array}{lll} \{x \mapsto 24, y \mapsto 9\} & x := x \dot{-} y & \{x \mapsto 15, y \mapsto 9\} \\ & x := x \dot{-} y & \{x \mapsto 6, y \mapsto 9\} \\ & y := y \dot{-} x & \{x \mapsto 6, y \mapsto 3\} \\ & x := x \dot{-} y & \{x \mapsto 3, y \mapsto 3\}. \end{array}$$

Consider the following command:

$$y := 1 \ ; \ \text{WHILE } x \neq 1 \text{ DO } (y := y * x \ ; \ x := x \dot{-} 1). \tag{2}$$

Let g be a valuation with $g(x) = 3$. Then one can use the transition rules to show:

$$g \xrightarrow{y:=1 \ ; \ \text{WHILE } x \neq 1 \text{ DO } (y:=y*x \ ; \ x:=x \dot{-} 1)} g[x \mapsto 1, y \mapsto 6].$$

When executed in a state g, command (2) computes the factorial of $g(x)$ in y.

We say that a command C *terminates* in state g if there is a state g' with $g \xrightarrow{C} g'$, and that C *loops* in state g if C does not terminate in state g. It can be shown by induction that it holds for all C that if $g \xrightarrow{C} g'$ and $g \xrightarrow{C} g''$ then $g' = g''$ (WHILE programs are *deterministic*).

In simple cases it is easy to say whether a command terminates in a given state. For example, the factorial command terminates for all states g, and the command

$$\text{WHILE } x > 0 \text{ DO } x := x + 1$$

loops for all states g with $g(x) \neq 0$. In general, however, termination of WHILE programs for infinite state sets is undecidable. As an example of a difficult decision problem about program termination, take the question whether the following program terminates for all states with positive x:

$$\text{WHILE } x \neq 1 \text{ DO } \ \text{IF } \ \text{even } (x) \text{ THEN } x := x/2 \text{ ELSE } x := (3 * x) + 1$$

Note that this example uses an operator / for integer division and a predicate for evenness, but this is not crucial, for these extensions are definable in the

WHILE language. Here is an example run of the program:

$$\begin{array}{rcl} & & x_0 = 7 \\ x := (3 * x) + 1 & \rightarrow & x_1 = 22 \\ x := x/2 & \rightarrow & x_2 = 11 \\ x := (3 * x) + 1 & \rightarrow & x_3 = 34 \\ x := x/2 & \rightarrow & x_4 = 17 \\ x := (3 * x) + 1 & \rightarrow & x_5 = 52 \\ x := x/2 & \rightarrow & x_6 = 26 \\ x := x/2 & \rightarrow & x_7 = 13 \\ x := (3 * x) + 1 & \rightarrow & x_8 = 40 \\ x := x/2 & \rightarrow & x_9 = 20 \\ x := x/2 & \rightarrow & x_{10} = 10 \\ x := x/2 & \rightarrow & x_{11} = 5 \\ x := (3 * x) + 1 & \rightarrow & x_{12} = 16 \\ x := x/2 & \rightarrow & x_{13} = 8 \\ x := x/2 & \rightarrow & x_{14} = 4 \\ x := x/2 & \rightarrow & x_{15} = 2 \\ x := x/2 & \rightarrow & x_{16} = 1 \end{array}$$

Counterexamples against termination have never been found, but a proof of termination has not been found either. This termination problem was posed by Collatz in 1937, and it is still open.

2.2.2 Structural Operational Semantics for Commands

An alternative fashion of specifying the semantics of an imperative programming language, due to Plotkin [104], specifies the transition system for a program in a slightly different way, focusing on the smallest steps that a computation can take. Here are the rules of what is called 'structural operational semantics', or 'small step semantics'. The transitions are now from pairs of a state and a command to a state (such a transition expresses that the command finishes in a single step), and from pairs of a state and a command to a new state and a new command (such a transition expresses that the first step of the command causes a shift to the new state, where the remainder of the command is left to be executed).

Assignment commands finish in one step:

$$(g, v := a) \implies g[v \mapsto [\![a]\!]_g].$$

The SKIP command also finishes in a single step, and it does not change the state.

$$(g, \text{SKIP}) \implies g.$$

If the first command of a command sequence finishes in a single step, then the second command of the sequence is all that is left:

$$\frac{(g, C_1) \implies g'}{(g, C_1 \ ; \ C_2) \implies (g', C_2)}$$

If the first command of a command sequence does not finish in a single step, we get:

$$\frac{(g, C_1) \implies (g', C_1')}{(g, C_1 \ ; \ C_2) \implies (g', C_1' \ ; \ C_2)}$$

Rules for conditional action: the action depends on the outcome of the test.

$$\frac{}{(g, \text{IF } B \text{ THEN } C_1 \text{ ELSE } C_2) \implies (g, C_1)} \ [\![B]\!]_g = T$$

$$\frac{}{(g, \text{IF } B \text{ THEN } C_1 \text{ ELSE } C_2) \implies (g, C_2)} \ [\![B]\!]_g = F$$

Finally, the guarded iteration command. If the guard is not satisfied, the command finishes in a single step, and it does not change the state:

$$\frac{}{(g, \text{WHILE } B \text{ DO } C) \implies g} \ [\![B]\!]_g = F$$

Otherwise the first step of the guarded action is performed, and in the result state the remainder of the action plus the conditional iteration command are put on the to-do list:

$$\frac{(g, C) \implies (g', C')}{(g, \ \text{WHILE } B \text{ DO } C) \implies (g', C'; \text{WHILE } B \text{ DO } C)} \ [\![B]\!]_g = T$$

To see how this works, consider the command $z := x \ ; \ x := y \ ; \ y := z$, executed in the state $g = \{x \mapsto 3, y \mapsto 2, z \mapsto 5\}$. The structural operational semantics rules yield the following:

$$\begin{aligned} & (\{x \mapsto 3, y \mapsto 2, z \mapsto 5\}, z := x \ ; \ x := y \ ; \ y := z) \\ \implies \ & (\{x \mapsto 3, y \mapsto 2, z \mapsto 3\}, x := y \ ; \ y := z) \\ \implies \ & (\{x \mapsto 2, y \mapsto 2, z \mapsto 3\}, y := z) \\ \implies \ & \{x \mapsto 2, y \mapsto 3, z \mapsto 3\} \end{aligned}$$

It can now be proved by induction that these rules define the same 'extensional' behaviour as the original rules, in the sense that $g \xrightarrow{C} g'$ iff $(g, C) \implies^* g'$.

The difference between natural semantics (large step semantics) and structural operational semantics (small step semantics) shows up as soon as we add a construct for error abortion to the language. Suppose ABORT is a program that in any state g stops execution without yielding a new output state. Then the difference between SKIP and ABORT is that we have $(g, \text{SKIP}) \implies g$ and $g \xrightarrow{\text{SKIP}} g$, while from (g, ABORT) there are no $\implies$ arrows, and there are no

states g' with $g \xrightarrow{\text{ABORT}} g'$. It turns out that in natural semantics there is no way to distinguish between abnormal termination and looping behaviour, while in structural operational semantics there is. In natural semantics, ABORT and WHILE $\top$ DO SKIP are equivalent, but in structural operational semantics they are not, for the first has no derivation sequence at all, while the second has an infinite one:

$$\begin{aligned}(g, \text{WHILE } \top \text{ DO SKIP}) &\Longrightarrow (g, \text{WHILE } \top \text{ DO SKIP}) \\ &\Longrightarrow (g, \text{WHILE } \top \text{ DO SKIP}) \\ &\Longrightarrow \ldots\end{aligned}$$

The natural semantics can be made more expressive by adding a special error state $\bullet$ different from all the regular states, and adding the transition rules $g \xrightarrow{\text{ABORT}} \bullet$, and $\bullet \xrightarrow{C} \bullet$ for all commands C. Under this modification ABORT and WHILE $\top$ DO SKIP become distinguishable again in natural semantics, for the first has a transition to $\bullet$ from anywhere, and the second has no transitions from anywhere.

2.2.3 Interpreted versus Uninterpreted Semantics

The WHILE language over $\mathbb{N}$ is an example of an interpreted language. We can also choose to interpret WHILE over different data structures. To see that this makes a difference, consider the following program:

$$\text{WHILE } x \neq 0 \text{ DO } x := p(x)$$

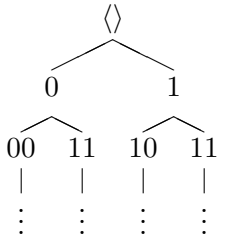
If p is interpreted as predecessor, this program will always terminate when executed on $\mathbb{N}$, but it will only terminate for states with a non-negative value for x when executed on $\mathbb{Z}$ (the domain of integers). As another example, let $\mathcal{T}$ be the infinite binary tree given by:

$$\mathcal{T} \quad ::= \quad \langle\rangle \mid \mathcal{T}0 \mid \mathcal{T}1$$

with a unary function $\uparrow :: \mathcal{T} \to \mathcal{T}$ defined by means of

$$\uparrow\langle\rangle = \langle\rangle, \uparrow\mathcal{T}0 = \uparrow\mathcal{T}1 = \mathcal{T}.$$

This specifies the following infinite binary tree:



Then the following WHILE program over $\mathcal{T}$

$$\text{WHILE } x \neq \langle\rangle \wedge y \neq \langle\rangle \text{ DO } (x := \uparrow x \ ; \ y := \uparrow y)$$

will always terminate in a state where $x = \langle\rangle$ or $y = \langle\rangle$, depending on which of x, y is closer to the root $\langle\rangle$ in the initial state.

WHILE programs can also be studied under the aspect of uninterpreted computation. Given a first order signature σ, we may be interested in equivalence of WHILE programs for arbitrary σ models. E.g., the commands

$$\text{IF } B \text{ THEN } C_1 \text{ ELSE } C_2$$

and

$$\text{IF } \neg B \text{ THEN } C_2 \text{ ELSE } C_1$$

are equivalent for any choice of B, C_1, C_2 and any model $\mathcal{M}$ for the predicate and function symbols that occur in B, C_1, C_2. Uninterpreted reasoning is the right level for comparing expressive power of programming language constructs, for on the fixed domain $\mathbb{N}$ with zero, successor, addition and multiplication all reasonable programming language have the same expressive power: they all compute exactly the partial recursive functions. At the uninterpreted level, extending the WHILE language with a construct for non-deterministic choice C_1 OR C_2 strictly increases expressive power.

2.3 Non-determinism

Non-deterministic WHILE is the extension of WHILE with a construct for choice C_1 OR C_2, with semantics given by the following transition rules:

$$\frac{g \xrightarrow{C_1} g'}{g \xrightarrow{C_1 \text{ OR } C_2} g'}$$

$$\frac{g \xrightarrow{C_2} g'}{g \xrightarrow{C_1 \text{ OR } C_2} g'}$$

What this says is that a program like $x := x + 1$ OR $x := x + 2$, when executed in a state $\{x \mapsto 3\}$ will produce two output states $\{x \mapsto 4\}$ and $\{x \mapsto 5\}$.

The structural operational semantics rules for choice are as follows:

$$\frac{}{(g, C_1 \text{ OR } C_2) \implies (g, C_1)}$$

$$\frac{}{(g, C_1 \text{ OR } C_2) \implies (g, C_2)}$$

Now consider program (3).

$$(\text{WHILE } \top \text{ DO SKIP}) \text{ OR } x := x + 2. \tag{3}$$

According to the natural semantics, for no input state g is there an output state g' with $g \xrightarrow{\text{WHILE } \top \text{ DO SKIP}} g'$. Therefore, program (3) will only get one derivation tree, namely that for:

$$g \xrightarrow{\text{(WHILE } \top \text{ DO SKIP) OR } x:=x+2} g\{x \mapsto x+2\}.$$

According to the structural operational semantics, we get two derivation sequences, one infinite

$$\begin{array}{l}(g,\ \text{(WHILE } \top \text{ DO SKIP) OR } x := x+2)\\ \Longrightarrow (g,\ \text{(WHILE } \top \text{ DO SKIP)}\\ \Longrightarrow (g,\ \text{(WHILE } \top \text{ DO SKIP)}\\ \Longrightarrow \ldots\end{array}$$

and the other finite

$$\begin{array}{l}(g,\ \text{(WHILE } \top \text{ DO SKIP) OR } x := x+2)\\ \Longrightarrow (g, x := x+2)\\ \Longrightarrow g\{x \mapsto x+2\}.\end{array}$$

This illustrates that the structural operational semantics is more 'fine-grained' than the natural semantics. It also shows that the presence of non-determinism may make looping behaviour more difficult to detect.

Programming language semantics in various styles for WHILE and its extensions are discussed in [98]. Classics on denotational semantics for programming are [119] and [112].

2.4 Floyd-Hoare Logic

One way of reasoning about WHILE commands (or about imperative programs in general) is by using first order predicate logic for making assertions about command execution. Floyd [43] and Hoare [70] proposed to use *correctness statements* of the following form:

$$\{\varphi\}\ C\ \{\psi\}$$

This expresses that command C takes us from a precondition φ, true at the state where the command gets executed (the input state), to a postcondition ψ, true immediately after execution of the command. Since we are programming over the natural numbers, we interpret the pre- and postconditions in $\mathbb{N}$. This gives the following formal interpretation of Floyd-Hoare correctness triples:

$$\begin{array}{ll}\mathbb{N} \models \{\varphi\}\ C\ \{\psi\} & \text{iff}\\ & \text{for all } g, h, \quad \text{if } \mathbb{N} \models_g \varphi \text{ and } g \xrightarrow{C} h, \text{ then } \mathbb{N} \models_h \psi.\end{array}$$

An example of a true correctness statement is the following:

$$\{x! = Z\}\ y := 1\ ;\ \text{WHILE } x \neq 1 \text{ DO } (y := y * x\ ;\ x := x \dot{-} 1)\ \{y = Z\}$$

Figure 1: Floyd-Hoare Calculus for WHILE

assignment	$\dfrac{}{\{\varphi_a^v\}\ v := a\ \{\varphi\}}$
skip	$\dfrac{}{\{\varphi\}\ \text{SKIP}\ \{\varphi\}}$
sequence	$\dfrac{\{\varphi\}\ C_1\ \{\psi\} \qquad \{\psi\}\ C_2\ \{\chi\}}{\{\varphi\}\ C_1\ ;\ C_2\ \{\chi\}}$
conditional choice	$\dfrac{\{\varphi \wedge B\}\ C_1\ \{\psi\} \qquad \{\varphi \wedge \neg B\}\ C_2\ \{\psi\}}{\{\varphi\}\ \text{ if } B \text{ then } C_1 \text{ else } C_2\ \{\psi\}}$
guarded iteration	$\dfrac{\{\varphi \wedge B\}\ C\ \{\varphi\}}{\{\varphi\}\ \text{ while } B \text{ do } C\ \{\varphi \wedge \neg B\}}$
precondition strengthening	$\dfrac{\mathbb{N} \models \varphi' \to \varphi \qquad \{\varphi\}\ C\ \{\psi\}}{\{\varphi'\}\ C\ \{\psi\}}$
postcondition weakening	$\dfrac{\{\varphi\}\ C\ \{\psi\} \qquad \mathbb{N} \models \psi \to \psi'}{\{\varphi\}\ C\ \{\psi'\}}$

In connection with Floyd-Hoare style correctness assertions, the notions of *strongest postcondition* and *weakest liberal precondition* arise in a natural way. The strongest postcondition $\mathrm{SP}(\varphi, C)$ of a predicate logical formula φ and a command C is the condition that holds in a state g if there is a state h satisfying φ that has a C transition to g. Formally:

$$\mathbb{N} \models_g \mathrm{SP}(\varphi, C) \text{ iff there is an } h \text{ with } \mathbb{N} \models_h \varphi \text{ and } h \xrightarrow{C} g.$$

The weakest liberal precondition $\mathrm{WLP}(C, \varphi)$ of a predicate logical formula φ and a command C has the following interpretation:

$$\mathbb{N} \models_g \mathrm{WLP}(C, \varphi) \text{ iff there is an } h \text{ with } \mathbb{N} \models_h \varphi \text{ and } g \xrightarrow{C} h.$$

The connection with Floyd-Hoare correctness statements is as follows:

$$\mathbb{N} \models \{\varphi\}\ C\ \{\mathrm{SP}(\varphi, C)\},$$

$$\text{if } \mathbb{N} \models \{\varphi\}\ C\ \{\psi\} \text{ then } \mathbb{N} \models \mathrm{SP}(\varphi, C) \to \psi,$$

$$\mathbb{N} \models \{\mathrm{WLP}(C, \varphi)\}\ C\ \{\varphi\},$$

$$\text{if } \mathbb{N} \models \{\varphi\}\ C\ \{\psi\} \text{ then } \mathbb{N} \models \varphi \to \mathrm{WLP}(C, \psi).$$

This illustrates the view of WHILE programs as predicate transformers, mapping weakest precondition predicates on the natural numbers into strongest postcondition predicates on the natural numbers.

A Floyd-Hoare calculus for WHILE programs is given in Figure 2.4. In the rule for assignment, φ_a^v denotes the result of substitution of a for v in φ. At first sight, one might think that the assignment axiom should run $\{\varphi\}\ v := a\ \{\varphi_a^v\}$ instead of $\{\varphi_a^v\}\ v := a\ \{\varphi\}$. This would be a mistake, for consider the example where φ equals the statement $v = 0$, and a equals $v + 1$. Then the rule $\{\varphi\}\ v := a\ \{\varphi_a^v\}$ yields the *incorrect* statement $\{v = 0\}\ v := v + 1\ \{v + 1 = 0\}$, while the correct rule $\{\varphi_a^v\}\ v := a\ \{\varphi\}$ yields the correct statement $\{v+1 = 0\}\ v := v+1\ \{v = 0\}$.

Note that the rules of precondition strengthening and postcondition weakening in $\mathbb{N}$ are a kind of oracle rules, for implications $\psi \to \psi'$ on the natural numbers may be undecidable.

Illustration To illustrate the use of the calculus, consider the factorial program (2) again. Here are the correctness statements that prove the fact that this program actually computes the factorial function:

1. $\{x! = Z\}\ y := 1\ \{y * x! = Z\}$
2. $\{y * x! = Z \wedge x \neq 0\}\ y := y * x\ \{y * x! = Z * x\}$
3. $\{y * x! = Z * x \wedge x \neq 0\}\ x := x \dot{-} 1\ \{y * x! = Z\}$
4. $\{y * x! = Z \wedge x \neq 0\}\ y := y * x\ ;\ x := x \dot{-} 1\ \{y * x! = Z\}$
5. $\{y{*}x! = Z\}$ WHILE $x \neq 0$ DO $(y := y{*}x\ ;\ x := x \dot{-} 1)\ \{y{*}x! = Z \wedge x = 0\}$.

6. $\{x! = Z\}$
   $y := 1 \ ; \ \text{WHILE } x \neq 0 \text{ DO } (y := y * x \ ; \ x := x \dot{-} 1)$
   $\{y * x! = Z \wedge x = 0\}$.
7. $\{x! = Z\}$
   $y := 1 \ ; \ \text{WHILE } x \neq 0 \text{ DO } (y := y * x \ ; \ x := x \dot{-} 1)$
   $\{y = Z\}$.

2.4.1 Properties

The Floyd-Hoare calculus for WHILE programs is sound, in the following sense: if $\{\varphi\}\ C\ \{\psi\}$ is derivable, using the rules for precondition strengthening and postcondition weakening in $\mathbb{N}$, then $\mathbb{N} \models \{\varphi\}\ C\ \{\psi\}$. Soundness is easily shown by induction on the length of Floyd-Hoare derivations.

The presence of the precondition strengthening and postcondition weakening introduce an element of *model checking* into the Floyd-Hoare calculus, making it into a hybrid tool for deduction and evaluation in $\mathbb{N}$.

Since arithmetical truth is not effectively axiomatisable, the true correctness statements for WHILE programs over $\mathbb{N}$ are not effectively axiomatisable either. Indeed, we have, for every arithmetical formula φ:

$$\mathbb{N} \models \varphi \text{ iff } \mathbb{N} \models \{\top\} \text{ SKIP } \{\varphi\}.$$

However, because strongest postconditions can be expressed in the language of $\mathbb{N}$ by means of encoding, we can get around this by allowing members of $\mathrm{Th}(\mathbb{N})$ (the set of all predicate logical statements that are true on the natural numbers) in correctness proofs [28]:

Theorem 1 (Cook, Relative Completeness) $\mathbb{N} \models \{\varphi\}\ C\ \{\psi\}$ *implies that* $\{\varphi\}\ C\ \{\psi\}$ *is derivable using Floyd-Hoare rules together with* $Th(\mathbb{N})$.

Proof. An induction on the structure of programs works. We just give the case of guarded iterations. Let $\mathbb{N} \models \{\varphi\}$ WHILE B DO C $\{\psi\}$. Now use the fact that strongest postconditions are encodable in $\mathbb{N}$ to define

$$\chi = \exists y_1 \cdots y_n(\mathrm{SP}(\varphi,\ \text{WHILE } B \wedge (x_1 \neq y_1 \vee \cdots \vee x_n \neq y_n) \text{ DO } C))$$

where $x_1, \ldots, x_n$ are all the variables occurring in C, and $y_1, \ldots, y_n$ are new. Then χ defines the states that can be reached from a φ state by means of a finite number of C transitions through B states. Thus, $\mathbb{N} \models \{\chi \wedge B\}\ C\ \{\chi\}$. This formula is derivable by the induction hypothesis. By the Floyd-Hoare rule for guarded iteration, it follows from this that

$$\{\chi\} \text{ WHILE } B \text{ DO } C\ \{\chi \wedge \neg B\}$$

is derivable too. Since $\varphi \to \chi$ and $\chi \wedge \neg B \to \psi$ are both true in $\mathbb{N}$ (the latter because $\chi \wedge \neg B$ is equivalent to SP(φ, WHILE B DO C)), by the rules for precondition strengthening and postcondition weakening we get that

$$\{\varphi\} \text{ WHILE } B \text{ DO } C \{\psi\}$$

must be derivable too. □

It is important to note that Floyd-Hoare correctness statements if this simple form are not expressive enough to reason about termination. The following correctness statement is true:

$\{x \geq 1\}$
WHILE $x \neq 1$ DO IF even (x) THEN $x := x/2$ ELSE $x := (3 * x) + 1$
$\{x = 1\}$

This expresses that *if* the command is executed in a state where x has a positive value, after termination x will have value 1. It does *not* express that the command will terminate for all states with x positive. This is the reason that Floyd-Hoare correctness statements are sometimes called partial correctness statements.

To remedy this, calculi have been proposed with a stronger interpretation, for reasoning about Floyd-Hoare triples expressing total correctness:

$$\{\varphi\}\ C\ \{\Downarrow \psi\}$$

Such a total correctness statement expresses that if precondition φ is fulfilled then C is guaranteed to terminate in a state satisfying ψ. To make this work, the rule for guarded iteration has to be reformulated in terms of a decreasing measure function M on the natural numbers, as follows (it is assumed that $\mathbb{N} \models (\varphi \wedge M = i + 1) \to B$ and $\mathbb{N} \models (\varphi \wedge M = 0) \to \neg B$):

$$\frac{\{\varphi \wedge M = i + 1\}\ C\ \{\Downarrow \varphi \wedge M = i\}}{\{\exists i(\varphi \wedge M = i)\} \text{ WHILE } B \text{ DO } C\ \{\Downarrow \varphi \wedge M = 0\}}$$

An overview of the development of Floyd-Hoare reasoning can be found in [2]. Floyd-Hoare reasoning is still a dominant tradition in program verification; pre- and postcondition annotations can be used as formal specifications with respect to which a program can be verified, where the verification process can be partially automated [51, 75].

Floyd-Hoare reasoning, the original flavour of dynamic logic for the analysis of programming, is applicable to sequential transformational programs. Sequential programs run on a single processor without involving concurrency. Transformational programs are programs that are expected to terminate with an output after a finite number of steps. Sequential transformational programs are in the realm of dynamic logic in the sense of the present paper.

Reactive systems are systems that are expected to 'run forever'; examples are text editors, operating systems. Concurrent reactive systems also involve interaction between processes; examples can be found in hardware systems, and embedded systems like the software that controls ignition and fuel injection of cars. The analysis and verification of (concurrent) reactive systems calls for model checking methods using temporal computation tree logics such as CTL, LTL and CTL* [26, 27, 105], and is outside the scope of our survey (but see Section 3.6.8 below).

3 Propositional Dynamic Logic

The language of propositional dynamic logic was defined by Pratt in [106,108] as a generic language for reasoning about computation. Axiomatisations were given independently by Segerberg [113], Fisher/Ladner [42], and Parikh [99]. These axiomatisations make the connection between propositional dynamic logic and modal logic very clear.

3.1 Language

Propositional dynamic logic can be viewed as a basic logic of change. Propositional dynamic logic abstracts over the set of basic actions, in the sense that basic actions are atoms. This means that its range of applicability is vast. In the WHILE language, the basic actions are definite assignments $v := a$ and the trivial action SKIP. Now the basic actions can be anything. The only thing that matters about a basic action a is that it is interpreted by some binary relation on a state set.

Dynamic logics have two basic syntactic categories: formulae and programs. Formulae are used for talking about states, programs for classifying transitions between states.

The same distinction can be found in all imperative programming languages, by the way. Imperative programming languages have programs (often called 'statements') versus formulae (often called 'Boolean expressions'). In the case of the WHILE language, the booleans appeared as conditions in conditional statements and as guards in guarded iterations.

Propositional dynamic logic is an extension of propositional logic with programs, just like basic modal logic is an extension of propositional logic with modalities. Let a set of basic propositions P be given. Appropriate states will contain valuations for these propositions. Assume a set of basic actions A. Every basic action corresponds to a binary relation on the state set.

Let p range over the set of basic propositions P, and let a range over a set of basic actions A. Then the formulae φ and programs α of propositional dynamic logic are given by:

$$\begin{array}{lcl} \varphi & ::= & \top \mid p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\alpha\rangle\varphi \\ \alpha & ::= & a \mid ?\varphi \mid \alpha_1 \ ; \ \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha^* \end{array}$$

We employ the usual abbreviations: $\bot$ is shorthand for $\neg\top$, $\varphi_1 \wedge \varphi_2$ is shorthand for $\neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2$ is shorthand for $\neg\varphi_1 \vee \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$ is shorthand for $(\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$, and $[\alpha]\varphi$ is shorthand for $\neg\langle\alpha\rangle\neg\varphi$. Also, we will use α^n for the program consisting of a sequence of n copies of α, i.e., we define α^n by means of $\alpha^0 :=?\top, \alpha^{n+1} := \alpha \ ; \ \alpha^n$.

Taking the basic actions to be computations, we can use PDL to talk about programming: for any program α, $\langle\alpha\rangle\top$ expresses that the program has at least one successful computation, and $[\alpha]\bot$ expresses that the program fails (does not produce any output). If the basic actions are communicative actions, e.g., public announcements, then $\langle\alpha\rangle\varphi$ expresses that a public announcement of α may have the effect that φ holds. If the basic actions are changes in the world, such as spilling milk S or cleaning C, then $[C\ ;\ S]d$ expresses that cleaning up followed by spilling milk always results in a dirty state, while $[S\ ;\ C]\neg d$ expresses that the occurrence of these events in the reverse order always results in a clean state.

Nor does this exhaust the application areas of PDL. In [19] and [87], variants of PDL are used for defining a variety of structural relations in syntax trees for natural language, and in [90] PDL is used to analyse *XPath*, a node addressing language of XML documents.

3.2 Semantics

If R_1, R_2 are binary relations on a state set S, then the relational composition $R_1 \circ R_2$ of R_1 and R_2 is given by:

$$R_1 \circ R_2 = \{(t_1, t_2) \in S \times S \mid \exists t_3 \in S\ ((t_1, t_3) \in R_1 \wedge (t_3, t_2) \in R_2)\}.$$

Let I be the identity relation on S. Then the n-fold composition of a binary relation R on S with itself is defined by recursion, as follows:

$$\begin{aligned} R^0 &= I \\ R^n &= R \circ R^{n-1} \end{aligned}$$

The reflexive transitive closure of R is given by:

$$R^* = \bigcup_{n \in \mathbb{N}} R^n.$$

The semantics of PDL over P, A is given relative to a labelled transition system $\mathbf{M} = \langle S, V, R\rangle$ for signature P, A. The formulae of PDL are interpreted as subsets of $S_{\mathbf{M}}$, the actions a of PDL as binary relations on $S_{\mathbf{M}}$ (with the

interpretation of basic actions a given as $\xrightarrow{a}$), as follows:

$$\begin{aligned}
[\![\top]\!]^{\mathbf{M}} &= S_{\mathbf{M}} \\
[\![p]\!]^{\mathbf{M}} &= \{s \in S_{\mathbf{M}} \mid p \in V_{\mathbf{M}}(s)\} \\
[\![\neg\varphi]\!]^{\mathbf{M}} &= S_{\mathbf{M}} - [\![\varphi]\!]^{\mathbf{M}} \\
[\![\varphi_1 \vee \varphi_2]\!]^{\mathbf{M}} &= [\![\varphi_1]\!]^{\mathbf{M}} \cup [\![\varphi_2]\!]^{\mathbf{M}} \\
[\![\langle\alpha\rangle\varphi]\!]^{\mathbf{M}} &= \{s \in S_{\mathbf{M}} \mid \exists t\ (s,t) \in [\![\alpha]\!]^{\mathbf{M}} \text{ and } t \in [\![\varphi]\!]^{\mathbf{M}}\}
\end{aligned}$$

$$\begin{aligned}
[\![a]\!]^{\mathbf{M}} &= \xrightarrow{a}_{\mathbf{M}} \\
[\![?\varphi]\!]^{\mathbf{M}} &= \{(s,s) \in S_{\mathbf{M}} \times S_{\mathbf{M}} \mid s \in [\![\varphi]\!]^{\mathbf{M}}\} \\
[\![\alpha_1\ ;\ \alpha_2]\!]^{\mathbf{M}} &= [\![\alpha_1]\!]^{\mathbf{M}} \circ [\![\alpha_2]\!]^{\mathbf{M}} \\
[\![\alpha_1 \cup \alpha_2]\!]^{\mathbf{M}} &= [\![\alpha_1]\!]^{\mathbf{M}} \cup [\![\alpha_2]\!]^{\mathbf{M}} \\
[\![\alpha^*]\!]^{\mathbf{M}} &= ([\![\alpha]\!]^{\mathbf{M}})^*
\end{aligned}$$

If $s \in S_{\mathbf{M}}$ then we use $\mathbf{M} \models_s \varphi$ for $s \in [\![\varphi]\!]^{\mathbf{M}}$.

These definitions specify how formulae of PDL can be used to make assertions about PDL models. The formula $\langle a\rangle\top$, when interpreted at some state in a PDL model, expresses that that state has a successor in the $\xrightarrow{a}$ relation in that model.

A PDL formula φ is *true* in a model if it holds at every state in that model, i.e., if $[\![\varphi]\!]^{\mathbf{M}} = S_{\mathbf{M}}$. Truth of the formula $\langle a\rangle\top$ in a model expresses that $\xrightarrow{a}$ is serial in that model.

A PDL formula φ is *valid* if it holds for all PDL models $\mathbf{M}$ that φ is true in that model, i.e., that $[\![\varphi]\!]^{\mathbf{M}} = S_{\mathbf{M}}$. An example of a valid formula is $\langle a\ ;\ b\rangle\top \leftrightarrow \langle a\rangle\langle b\rangle\top$.

Note that ? is an operation for mapping formulae to programs. Programs of the form $?\varphi$ are called *tests*; they are interpreted as the identity relation, restricted to the states satisfying the formula.

Programming Constructs The following abbreviations illustrate how PDL expresses the key constructs of imperative programming:

$$\begin{aligned}
\text{SKIP} &:= ?\top \\
\text{ABORT} &:= ?\bot \\
\text{IF } \varphi \text{ THEN } \alpha_1 \text{ ELSE } \alpha_2 &:= (?\varphi\ ;\ \alpha_1) \cup (?\neg\varphi\ ;\ \alpha_2) \\
\text{WHILE } \varphi \text{ DO } \alpha &:= (?\varphi\ ;\ \alpha)^*\ ;\ ?\neg\varphi \\
\text{REPEAT } \alpha \text{ UNTIL } \varphi &:= \alpha\ ;\ (?\neg\varphi\ ;\ \alpha)^*\ ;\ ?\varphi.
\end{aligned}$$

3.3 PDL Equivalences

The two PDL programs β ; WHILE φ DO β and REPEAT β UNTIL $\neg\varphi$ are equivalent, in the sense that they will receive the same interpretations in all PDL models, for any choice of PDL formula φ and PDL program β. What this means is that for any formula ψ, the formula

$$\langle \beta \ ; \ \text{WHILE } \varphi \text{ DO } \beta \rangle \psi \ \leftrightarrow \ \langle \text{REPEAT } \beta \text{ UNTIL } \neg\varphi \rangle \psi$$

will be true in all PDL models.

Similarly, the formula

$$\langle \text{ IF } \varphi \text{ THEN } \beta \text{ ELSE } \gamma \rangle \psi \ \leftrightarrow \ \langle \text{ IF } \neg\varphi \text{ THEN } \gamma \text{ ELSE } \beta \rangle \psi$$

will be true in all PDL models, for all choices of $\beta, \gamma, \varphi, \psi$.

The regular expressions over a finite alphabet Σ are given by (σ ranges over Σ):

$$E \ ::= \ \epsilon \mid \sigma \mid E_1 \ ; \ E_2 \mid E_1 \cup E_2 \mid E^*$$

The denotations of regular expressions over Σ are precisely the regular languages over Σ. Two regular expressions are equivalent if they denote the same language. It is clear that if the basic actions are taken as the alphabet Σ, regular expressions correspond to PDL programs (take $?\top$ for the empty string ϵ).

Regular expression equivalence can be expressed in PDL, as follows. The regular expressions $(A \cup B)^*$ and $(A^* \ ; \ B^*)^*$ are equivalent. This law translates into PDL as the equivalence of the programs $(\alpha \cup \beta)^*$ and $(\alpha^* \ ; \ \beta^*)^*$ (or the equivalence of the formulae $\langle (\alpha \cup \beta)^* \rangle \varphi$ and $\langle (\alpha^* \ ; \ \beta^*)^* \rangle \varphi$). And so on.

3.4 Axiomatisation

The logic of PDL is axiomatised as follows. Axioms are all propositional tautologies, plus the following axioms (we give box ($[\alpha]$)versions here, but every axiom has an equivalent diamond ($\langle\alpha\rangle$) version):

$$\begin{aligned}
\text{(K)} \ &\vdash \quad [\alpha](\varphi \to \psi) \to ([\alpha]\varphi \to [\alpha]\psi) \\
\text{(test)} \ &\vdash \quad [?\varphi_1]\varphi_2 \leftrightarrow (\varphi_1 \to \varphi_2) \\
\text{(sequence)} \ &\vdash \quad [\alpha_1 \ ; \ \alpha_2]\varphi \leftrightarrow [\alpha_1][\alpha_2]\varphi \\
\text{(choice)} \ &\vdash \quad [\alpha_1 \cup \alpha_2]\varphi \leftrightarrow [\alpha_1]\varphi \wedge [\alpha_2]\varphi \\
\text{(mix)} \ &\vdash \quad [\alpha^*]\varphi \leftrightarrow \varphi \wedge [\alpha][\alpha^*]\varphi \\
\text{(induction)} \ &\vdash \quad (\varphi \wedge [\alpha^*](\varphi \to [\alpha]\varphi)) \to [\alpha^*]\varphi
\end{aligned}$$

and the following rules of inference:

(modus ponens) From $\vdash \varphi_1$ and $\vdash \varphi_1 \to \varphi_2$, infer $\vdash \varphi_2$.

(modal generalisation) From $\vdash \varphi$, infer $\vdash [\alpha]\varphi$.

The first axiom is the familiar K axiom from modal logic. The second captures the effect of testing, the third captures concatenation, the fourth choice. These axioms together reduce PDL formulae without $*$ to formulae of multi-modal logic. The fifth axiom, the so-called mix axiom, expresses the fact that α^* is a reflexive and transitive relation containing α, and the sixth axiom, the axiom of induction, captures the fact that α^* is the *least* reflexive and transitive relation containing α.

All axioms have dual forms in terms of $\langle\alpha\rangle$, derivable by propositional reasoning. For example, the dual form of the test axiom reads

$$\vdash \langle ?\varphi_1\rangle\varphi_2 \leftrightarrow (\varphi_1 \wedge \varphi_2).$$

The dual form of the induction axiom reads

$$\vdash \langle\alpha^*\rangle\varphi \rightarrow \varphi \vee \langle\alpha^*\rangle(\neg\varphi \vee \langle\alpha\rangle\varphi).$$

Use $\Gamma \vdash \varphi$ to express that φ is derivable using hypotheses from Γ by means of the axioms and inference rules of PDL. By induction on the length of proofs it can be shown that PDL satisfies the deduction theorem:

$$\Gamma \cup \{\varphi\} \vdash \psi \text{ iff } \Gamma \vdash \varphi \rightarrow \psi.$$

The deduction theorem will be used to facilitate PDL reasoning in what follows.

The following theorem shows that in the presence of the other axioms, the induction axiom is equivalent to the so-called loop invariance rule:

$$\frac{\varphi \rightarrow [\alpha]\varphi}{\varphi \rightarrow [\alpha^*]\varphi}$$

Theorem 2 *In PDL without the induction axiom, the induction axiom and the loop invariance rule are interderivable.*

Proof. For deriving the loop invariance rule from the induction axiom, assume the induction axiom. Suppose

$$\vdash \varphi \rightarrow [\alpha]\varphi.$$

Then by modal generalisation:

$$\vdash [\alpha^*](\varphi \rightarrow [\alpha]\varphi).$$

Now assume φ. Then:

$$\varphi \vdash \varphi \wedge [\alpha^*](\varphi \rightarrow [\alpha]\varphi).$$

From this by the induction axiom and propositional reasoning:

$$\varphi \vdash [\alpha^*]\varphi.$$

From this by conditionalisation (the left-to-right direction of the deduction theorem):

$$\vdash \varphi \to [\alpha^*]\varphi.$$

Now assume the loop invariance rule. We have to establish the induction axiom. Assume φ and $[\alpha^*](\varphi \to [\alpha]\varphi)$. Then by the mix axiom:

$$\varphi, [\alpha^*](\varphi \to [\alpha]\varphi) \vdash \varphi \to [\alpha]\varphi.$$

From this, by propositional reasoning:

$$\varphi, [\alpha^*](\varphi \to [\alpha]\varphi) \vdash [\alpha]\varphi.$$

Conditionalisation:

$$\vdash (\varphi \wedge [\alpha^*](\varphi \to [\alpha]\varphi)) \to [\alpha]\varphi.$$

Applying the loop invariance rule to this yields the induction axiom:

$$\vdash (\varphi \wedge [\alpha^*](\varphi \to [\alpha]\varphi)) \to [\alpha^*]\varphi.$$

□

3.5 PDL and Floyd-Hoare Reasoning

Floyd-Hoare correctness assertions are expressible in PDL, as follows. If φ, ψ are PDL formulae and α is a PDL program, then

$$\{\varphi\}\ \alpha\ \{\psi\}$$

translates into

$$\varphi \to [\alpha]\psi.$$

Clearly, $\{\varphi\}\ \alpha\ \{\psi\}$ holds in a state in a model iff $\varphi \to [\alpha]\psi$ is true in that state in that model.

The Floyd-Hoare inference rules can now be derived in PDL. As an example we derive the rule for guarded iteration:

$$\frac{\{\varphi \wedge \psi\}\ \alpha\ \{\psi\}}{\{\psi\}\ \text{WHILE}\ \varphi\ \text{DO}\ \alpha\ \{\neg\varphi \wedge \psi\}}$$

Let the premise $\{\varphi \wedge \psi\}\ \alpha\ \{\psi\}$ be given, i.e., assume (4).

$$\vdash (\varphi \wedge \psi) \to [\alpha]\psi. \tag{4}$$

We wish to derive the conclusion

$$\vdash \{\psi\}\ \text{WHILE}\ \varphi\ \text{DO}\ \alpha\ \{\neg\varphi \wedge \psi\},$$

i.e., we wish to derive (5).

$$\vdash \psi \rightarrow [(?\varphi; \alpha)^* \ ; \ ?\neg\varphi](\neg\varphi \wedge \psi). \tag{5}$$

From (4) by means of propositional reasoning:

$$\vdash \psi \rightarrow (\varphi \rightarrow [\alpha]\psi).$$

From this, by means of the test and sequence axioms:

$$\vdash \psi \rightarrow [\varphi \ ; \ \alpha]\psi.$$

Applying the loop invariance rule gives:

$$\vdash \psi \rightarrow [(\varphi \ ; \ \alpha)^*]\psi.$$

Since ψ is propositionally equivalent with $\neg\varphi \rightarrow (\neg\varphi \wedge \psi)$, we get from this by propositional reasoning:

$$\vdash \psi \rightarrow [(\varphi \ ; \ \alpha)^*](\neg\varphi \rightarrow (\neg\varphi \wedge \psi)).$$

The test axiom and the sequencing axiom yield the desired result (5).

3.6 Properties

3.6.1 Failure of Compactness

The presence of the * (Kleene star) operator causes true infinitary behaviour. In particular, the compactness theorem, which says that finite satisfiability of an infinite set of formulae Γ implies satisfiability of Γ, fails for PDL. Here is an example of a set of PDL formulae that is finitely satisfiable but not satisfiable:

$$\{\langle a^* \rangle p\rangle\} \cup \{\neg p, \neg\langle a\rangle p, \neg\langle a^2\rangle p, \ldots\}.$$

3.6.2 Finite Model Property

A logic has the finite model property (fmp) if every non-theorem of the logic has a finite counterexample. Having the fmp implies decidability, but not conversely (there are decidable logics without the fmp). We will now show that PDL has the fmp.

For normal modal logic, the fmp can be shown by means of the so-called filtration method [18, Ch 2], using subformula closed sets of formulae. Because of the presence of the star operator, in the case of PDL closure under subformulae is not enough. We also need to make sure that program modalities are decomposed in an appropriate way. For this, we use so-called Fisher-Ladner closures [42].

Define FL(φ), the Fisher-Ladner closure of a PDL formula φ, as follows. FL(φ) is the smallest set of formulae X containing φ that is closed under the following operations (the definition assumes diamond modalities here; an equivalent formulation in terms of box modalities is also possible):

- if $\neg\psi \in X$ then $\psi \in X$,
- if $(\psi_1 \vee \psi_2) \in X$ then $\psi_1 \in X, \psi_2 \in X$,
- if $\langle\alpha\rangle\psi \in X$ then $\psi \in X$,
- if $\langle\alpha_1 \ ; \ \alpha_2\rangle\psi \in X$ then $\langle\alpha_1\rangle\langle\alpha_2\rangle\psi \in X$,
- if $\langle\alpha_1 \cup \alpha_2\rangle\psi \in X$ then $\langle\alpha_1\rangle\psi \vee \langle\alpha_2\rangle\psi \in X$,
- if $\langle ?\psi_1\rangle\psi_2 \in X$ then $\psi_1 \in X, \psi_2 \in X$,
- if $\langle\alpha^*\rangle\psi \in X$ then $\langle\alpha\rangle\langle\alpha^*\rangle\psi \in X$.

Note that FL(φ) is always finite. E.g., FL($\langle(a \ ; \ b)^*\rangle(p \vee q)$) equals

$$\{\langle(a \ ; \ b)^*\rangle(p \vee q), p \vee q, p, q,$$
$$\langle(a \ ; \ b)\rangle\langle(a \ ; \ b)^*\rangle(p \vee q), \langle a\rangle\langle b\rangle\langle(a \ ; \ b)^*\rangle(p \vee q), \langle b\rangle\langle(a \ ; \ b)^*\rangle(p \vee q)\}.$$

Using FL(φ), define filtrations of LTSs, as follows. Let $\mathbf{M} = (S, V, R)$ be an LTS. For every s, let $\bar{s} = \{\psi \in \text{FL}(\varphi) \mid \mathbf{M} \models_s \psi\}$.

Set $\bar{s}\bar{R}_a\bar{t}$ if $\exists u, v \in S$ such that uR_av and $\bar{u} = \bar{s}$ and $\bar{v} = \bar{t}$. Finally, put $\bar{V}(\bar{s}) = \{p \in P \mid p \in \bar{s}\}$. Let $\overline{\mathbf{M}} = (\bar{S}, \bar{V}, \bar{R})$. Then one can prove:

Lemma 3 (Filtration Lemma) *For all $\psi \in FL(\varphi)$, all $s \in S$:*

$$\mathbf{M} \models_s \psi \text{ iff } \overline{\mathbf{M}} \models_{\bar{s}} \psi.$$

Proof. One shows with induction on the complexity of formulae and programs occurring in FL(φ) that:

- $\mathbf{M} \models_s \psi$ iff $\overline{\mathbf{M}} \models_{\bar{s}} \psi$.
- if $sR_\alpha t$ then $\bar{s}\bar{R}_\alpha\bar{t}$.

The crucial step is the following. Suppose that $\langle\alpha\rangle\psi$ is true in $\overline{\mathbf{M}}$ on $\bar{s}$. Then there exists a computation path for α consisting of a finite sequence of atomic transitions

$$\bar{s} \to \bar{s}_1 \to \cdots \to \bar{s}_n = \bar{t},$$

with appropriate atomic $\bar{R}_a$ links between $\bar{s}_i$ and $\bar{s}_{i+1}$, and possible appropriate tests $?\chi_i$ at $\bar{s}_i$, and with ψ true at $\bar{t}$.

By the definition of $\bar{R}_a$, there has to be a corresponding 'pseudo computation path'

$$s \sim u \to s_1 \sim u_1 \to \cdots \to u_n \sim t,$$

where $x \sim y$ expresses that $\bar{x} = \bar{y}$. Moreover, we have by the induction hypothesis that the same test conditions $?\chi_i$ hold at s_i and u_i, and that ψ holds at u_n and t.

Next, prove by induction on α:

> If $\langle\alpha\rangle\psi \in \text{FL}(\varphi)$ and there is pseudo computation path for α from s to t with $\mathbf{M} \models_t \psi$ then $\mathbf{M} \models_s \langle\alpha\rangle\psi$.

This clinches the argument. □

3.6.3 Decidability

Decidability follows from the filtration lemma:

Theorem 4 *Universal validity for PDL is decidable*

Proof. By the filtration lemma, counterexamples for a formula φ must already show up in models with at most $2^{|\mathrm{FL}(\varphi)|}$ states. It is possible, in principle, to inspect all of these. □

It follows immediately that satisfiability for PDL is decidable too: to check that φ is satisfiable, just find a satisfying model with at most $2^{|\mathrm{FL}(\varphi)|}$ states.

3.6.4 Converse

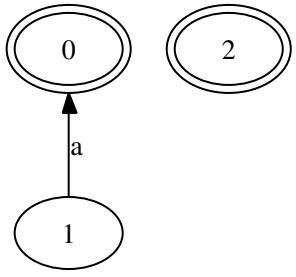
Let $^\smile$ (converse) be an operator on PDL programs with the following interpretation:

$$[\![\alpha^\smile]\!]^{\mathbf{M}} \;=\; \{(s,t) \mid (t,s) \in [\![\alpha]\!]^{\mathbf{M}}\}.$$

It is easy to see that the following equations hold:

$$\begin{aligned}(\alpha \;;\; \beta)^\smile &= \beta^\smile \;;\; \alpha^\smile \\ (\alpha \cup \beta)^\smile &= \alpha^\smile \cup \beta^\smile \\ (\alpha^*)^\smile &= (\alpha^\smile)^*\end{aligned}$$

This means that it is enough to add converse to the PDL language for atomic programs only. To see that adding converse in this way increases expressive power, observe that in state 0 in the following picture $\langle a^\smile\rangle\top$ is true, while in state 2 in the picture $\langle a^\smile\rangle\top$ is false. On the assumption that 0 and 2 have the same valuation, no PDL formula without converse can distinguish the two states.



Suitable axioms to enforce that $a^\smile$ behaves as the converse of a are well known from temporal logic (read $\langle a\rangle$ as F 'once in the future', $[a]$ as G 'always in the future', $\langle a^\smile\rangle$ as P 'once in the past', $[a^\smile]$ as H 'always in the past', [109, 110]):

$$\begin{aligned}\varphi &\rightarrow [a]\langle a^\smile\rangle\varphi \\ \varphi &\rightarrow [a^\smile]\langle a\rangle\varphi\end{aligned}$$

3.6.5 Wellfoundedness, Halting

For deterministic programs α, formula $\langle\alpha\rangle\top$ expresses that α does not loop. For non-deterministic programs α, however, there turns out to be no PDL way to express non-looping behaviour. If α is non-deterministic, $\langle\alpha\rangle\top$ merely says that in the current state there exists a terminating run for α, it does not preclude the existence of diverging runs. For example, formula $\langle(?\top)^*\rangle\top$ will be true at any state, while $(?\top)^*$ has diverging runs from every state.

One way to deal with this situation is to add a predicate to PDL to express wellfoundedness. A relation R is wellfounded in s_0 if there does *not* exist an infinite sequence $s_0, s_1, \ldots$ with

$$s_0 R s_1, s_1 R s_2, \ldots$$

Let **wellfounded** be a predicate for this. Then its interpretation is:

$$[\![\mathbf{wellfounded}(\alpha)]\!]^{\mathcal{M}} \;=\; \{s_0 \in S_{\mathbf{M}} \mid \neg\exists s_1, s_2, \ldots \forall i \geq 0 (s_i, s_{i+1}) \in [\![\alpha]\!]^{\mathbf{M}}\}.$$

In terms of **wellfounded**, a predicate **halt** for program termination can be defined as follows:

$$\begin{aligned}
\mathbf{halt}(a) &:\equiv \top \\
\mathbf{halt}(?\varphi) &:\equiv \top \\
\mathbf{halt}(\alpha\ ;\ \beta) &:\equiv \mathbf{halt}(\alpha) \wedge [\alpha]\mathbf{halt}(\beta) \\
\mathbf{halt}(\alpha \cup \beta) &:\equiv \mathbf{halt}(\alpha) \wedge \mathbf{halt}(\beta) \\
\mathbf{halt}(\alpha^*) &:\equiv \mathbf{wellfounded}(\alpha) \wedge [\alpha^*]\mathbf{halt}(\alpha)
\end{aligned}$$

What the definition of **halt** for programs of the form α^* says is that for α^* to halt it has to be the case that α is wellfounded at the present state (so that its execution can not be repeated without end), and also α has to halt at all states that can be reached in a finite number of α steps from the present state. This expresses that α^* can loop for two reasons: (i) because α can be repeated without end, or (ii) because after repeated execution of α there is a state where α itself does not terminate.

Applying this to the example program $(?\top)^*$, we get:

$$\begin{aligned}
\mathbf{halt}((?\top)^*) &\equiv \mathbf{wellfounded}(?\top) \wedge [(?\top)^*]\mathbf{halt}(?\top) \\
&\equiv \mathbf{wellfounded}(?\top) \wedge [(?\top)^*]\top \\
&\equiv \mathbf{wellfounded}(?\top) \wedge \top \\
&\equiv \bot
\end{aligned}$$

What this says is that $(?\top)^*$ does not halt because the test $?\top$ is not wellfounded (for $?\top$ can be repeated an arbitrary number of times).

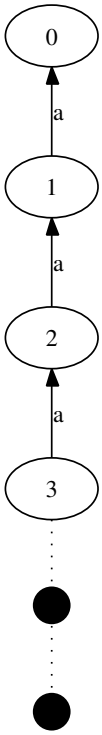
Floyd-Hoare total correctness statements for PDL programs α,

$$\{\varphi\}\ \alpha\ \{\Downarrow \psi\}$$

can now be expressed as:

$$\varphi \rightarrow [\alpha]\psi \wedge \varphi \rightarrow \mathbf{halt}(\alpha).$$

Every state in the infinite model of the following picture satisfies $\mathbf{halt}(a)$, but clearly, any filtration of this model must collapse some of the states, and in these collapsed states $\mathbf{halt}(a)$ will fail. This shows that extending PDL with a **halt** predicate (and, a fortiori, extending PDL with a **wellfounded** predicate) increases expressive power.



3.6.6 Further Extensions and Variations

Other possible extensions of PDL are with intersection and nominals [101]. The extension with nominals turns PDL into a kind of hybrid logic [4]. Replacing the regular programs of PDL by finite automata yields a formalism with the same expressive power but allowing more succinct descriptions: see [63]. Replacing the regular programs of PDL with another data structure such as pushdown automata or context free grammars or flowcharts yields more expressive (but also more complex) formalisms.

3.6.7 Complexity

Although satisfiability checking in individual LTSs can be done quite efficiently (i.e., in polynomial time), the above algorithm for checking satisfiability is highly inefficient, because the size of the models to check is exponential in the size of the formula, and the number of these models is doubly exponential in the size of the formula. So the naive satisfiability checking algorithm is doubly exponential in the size of the formula.

Time complexity of the satisfiability problem for PDL is singly exponential: an exponential algorithm is given in [107]. One cannot do better than this: [42] establishes an exponential-time lower bound for PDL satisfiability, by showing how PDL formulae can encode computations of linear-space-bounded alternating Turing machines. An exponential time satisfiability algorithm for PDL with converse is given in [120]. Intuitively, adding converse does not increase complexity, for converses of atomic programs a can be taken as atoms, and the definition of converse for complex programs is linear in the size of the programs.

3.6.8 Modal μ calculus

For a proper perspective on PDL, it is useful to contrast it with a much more expressive dynamic logic, the modal μ calculus.

Let a set of proposition letters $P = \{p_0, p_1, \ldots\}$, a set of actions $A = \{a_0, a_1, \ldots\}$, and a set of variables $V = \{X_0, X_1, \ldots\}$ be given. Assume p ranges over P, a ranges over A, and X ranges over V. Then the set of μ formulae is given by the following definition:

$$\varphi \quad ::= \quad \top \mid p \mid X \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle a \rangle \varphi \mid \mu X.\varphi,$$

with the syntactic restriction on $\mu X.\varphi$ that occurrences of X in φ are positive. An occurrence of X in a formula φ is positive if the occurrence is in the scope of an even number of negation signs.

Interpretation is in LTSs $\mathbf{M}$, relative to an assignment $g : V \to \mathcal{P}(S_{\mathbf{M}})$. If T is a subset of $S_{\mathbf{M}}$, $g[X \mapsto T]$ is the assignment that is like g except for the fact that it maps X to T.

$$\begin{aligned}
[\![\top]\!]^{\mathbf{M}}_g &= S_{\mathbf{M}} \\
[\![p]\!]^{\mathbf{M}}_g &:= \{s \in S_{\mathbf{M}} \mid p \in V_{\mathbf{M}}(s)\} \\
[\![X]\!]^{\mathbf{M}}_g &:= g(X) \\
[\![\neg\varphi]\!]^{\mathbf{M}}_g &= S_{\mathbf{M}} - [\![\varphi]\!]^{\mathbf{M}}_g \\
[\![\varphi_1 \vee \varphi_2]\!]^{\mathbf{M}}_g &= [\![\varphi_1]\!]^{\mathbf{M}}_g \cup [\![\varphi_2]\!]^{\mathbf{M}}_g \\
[\![\langle a \rangle \varphi]\!]^{\mathbf{M}}_g &= \{s \in S_{\mathbf{M}} \mid \exists t \; s \xrightarrow{a} t \text{ and } t \in [\![\varphi]\!]^{\mathbf{M}}_g\} \\
[\![\mu X.\varphi]\!]_g &= \bigcap \{T \subseteq S_{\mathbf{M}} \mid [\![\varphi]\!]^{\mathbf{M}}_{g[X \mapsto T]} \subseteq T\}
\end{aligned}$$

The clause for $\mu X.\varphi$ expresses that the interpretation of this formula is the least fixed point of the operation $T \mapsto [\![\varphi]\!]^{\mathbf{M}}_{g[T \mapsto S]}$. Thanks to the fact that X only occurs positively in φ, this operation is monotone:

$$\text{if } T \subseteq S \text{ then } [\![\varphi]\!]^{\mathbf{M}}_{g[X \mapsto T]} \subseteq [\![\varphi]\!]^{\mathbf{M}}_{g[X \mapsto S]}.$$

It follows, by a theorem of Knaster and Tarski (see, e.g., [29]), that the operation has a least fixed point, and that this least fixed point is given by the semantic clause for $\mu X.\varphi$. The proof of this fact is instructive.

For simplicity we use $[\varphi]T$ for $[\![\varphi]\!]^{\mathbf{M}}_{g[X \mapsto T]}$, and $[\varphi]$ for $T \mapsto [\varphi]T$. Let

$$\begin{array}{rcl} W & := & \bigcap \{T \subseteq S_{\mathbf{M}} \mid [\varphi]T \subseteq T\} \\ \mathcal{F} & := & \{T \subseteq S_{\mathbf{M}} \mid [\varphi]T \subseteq T\}. \end{array}$$

We have to show that W is the least fixed point of $[\varphi]$.

First we show $[\varphi]W \subseteq W$. Observe that for all $U \in \mathcal{F}$ we have $W \subseteq U$ and $[\varphi]U \subseteq U$. By monotonicity of $[\varphi]$, $[\varphi]W \subseteq [\varphi]U$, and therefore, by $[\varphi]U \subseteq U$, $[\varphi]W \subseteq U$. From the fact that for all $U \in \mathcal{F}$ it holds that $[\varphi]W \subseteq U$ we get the desired result $[\varphi]W \subseteq W$.

Next we show $W \subseteq [\varphi]W$. We start out from the previous result $[\varphi]W \subseteq W$. By monotonicity of $[\varphi]$ we get from this that $[\varphi][\varphi]W \subseteq [\varphi]W$. This shows that $[\varphi]W \in \mathcal{F}$, whence $W \subseteq [\varphi]W$.

Finally, to show that W is the least fixpoint, observe that any fixpoint U of $[\varphi]$ is in $\mathcal{F}$, so that $W \subseteq U$.

The modal μ calculus translates into second order predicate logic as follows:

$$\begin{array}{rcl} X^\circ & := & X(x) \\ (\mu X.\varphi)^\circ & := & \forall X(\forall x(\varphi^\circ \to X(x)) \to X(x)). \end{array}$$

This translation is called the *standard translation* into monadic second order logic, *monadic* because the predicate variables X quantified over in the translation are unary.

The μ calculus can be presented in PDL format by distinguishing between formulae and programs, as follows:

$$\begin{array}{rcl} \varphi & ::= & \top \mid p \mid X \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\alpha\rangle\varphi \mid \mu X.\varphi \\ \alpha & ::= & a \mid ?\varphi \mid \alpha_1 \cup \alpha_2 \mid \alpha_1; \alpha_2 \mid \alpha^* \end{array}$$

again with the syntactic restriction on $\mu X.\varphi$ formulae that X occurs only positively in φ.

This PDL version of the μ calculus does not have greater expressive power than the original, for we have the following equivalences:

$$\begin{aligned} \langle ?\varphi_1\rangle\varphi_2 &\equiv \varphi_1 \wedge \varphi_2 \\ \langle \alpha_1 \cup \alpha_2\rangle\varphi &\equiv \langle\alpha_1\rangle\varphi \vee \langle\alpha_2\rangle\varphi \\ \langle \alpha_1; \alpha_2\rangle\varphi &\equiv \langle\alpha_1\rangle\langle\alpha_2\rangle\varphi \\ \langle \alpha^*\rangle\varphi &\equiv \mu X.(\varphi \vee \langle\alpha\rangle X). \end{aligned}$$

To see that $\langle\alpha^*\rangle\varphi$ and $\mu X.(\varphi \vee \langle\alpha\rangle X)$ are equivalent, observe that the least fixpoint of the operation

$$T \mapsto [\![\varphi]\!]^{\mathbf{M}} \cup \{s \in S_{\mathbf{M}} \mid \exists t \in T. s \xrightarrow{\alpha} t\}$$

is equal to the set

$$\{s \in S_{\mathbf{M}} \mid \exists t \in [\![\varphi]\!]^{\mathbf{M}}. s \xrightarrow{\alpha^*} t\}.$$

We will now show that the μ calculus has greater expressive power than PDL. In PDL, there is no way to express that a program is wellfounded. The following formula expresses wellfoundedness of α in the μ calculus:

$$\mu X.[\alpha]X.$$

The meaning of this may not be immediately obvious, so let us analyse this a bit further. Let

$$W := \{s \in S_{\mathbf{M}} \mid \text{ there is no infinite } \alpha \text{ path from } s\}.$$

Then clearly, $\{s \in S_{\mathbf{M}} \mid \text{ if } s \xrightarrow{\alpha} t \text{ then } t \in W\} = W$. If there is no infinite α path starting form s, then there is no infinite α path from any α successor of α, and if at no α successor of s an infinite α path starts, then no infinite α path starts from s. In other words, W is a fixpoint of the operation

$$T \mapsto \{s \in S_{\mathbf{M}} \mid \text{ if } s \xrightarrow{\alpha} t \text{ then } t \in T\}.$$

We still have to show that W is also the least fixpoint of the operation. So suppose U is another solution:

$$\{s \in S_{\mathbf{M}} \mid \text{ if } s \xrightarrow{\alpha} t \text{ then } t \in U\} = U. \qquad (*)$$

We have to show that $W \subseteq U$. Assume, for a contradiction, that there is some $s \in W$ with $s \notin U$. From (*),

$$s \notin \{s \in S_{\mathbf{M}} \mid \text{ if } s \xrightarrow{\alpha} t \text{ then } t \in U\}.$$

It follows that for some $t \in S_{\mathbf{M}}$ we have $s \xrightarrow{\alpha} t$ and $t \notin U$. Continuing like this, we find $t \xrightarrow{\alpha} t'$ with $t' \notin U$, $t' \xrightarrow{\alpha} t''$ with $t'' \notin U$, and so on, an infinite α path starting from s, which contradicts the assumption that $s \in W$.

To define a greatest fixpoint operator dual to μ, use

$$\nu X.\varphi := \neg\mu X.(\varphi[X \mapsto \neg X]),$$

where $\varphi[X \mapsto \neg X]$ denotes the result of replacing every occurrence of X in φ by $\neg X$.

The μ calculus originates in [86]. It has great expressive power (it subsumes PDL, CTL, LTL and CTL*), it is decidable and has the finite model property [121], but it has greater complexity than PDL: known decision procedures use doubly exponential time.

Kozen [86] proposed an elegant proof system: the axioms and rules of multi-modal logic together with the axiom

$$\mu X.\varphi \leftrightarrow \varphi[X \mapsto \mu X.\varphi]$$

and the following rule of inference:

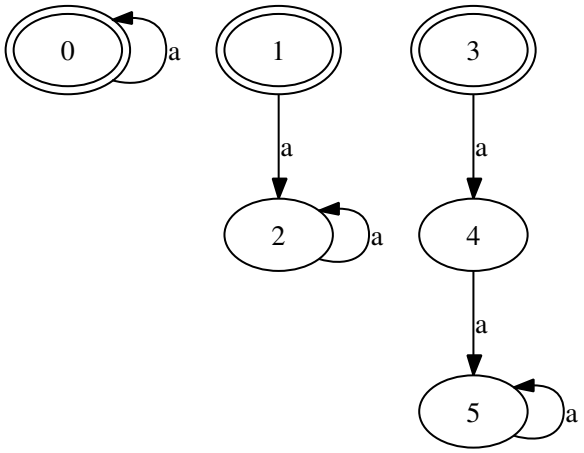
$$\frac{\varphi[X \mapsto \psi] \rightarrow \psi}{\mu X.\varphi \rightarrow \psi}$$

This axiomatisation is sound and complete.

Alternatively, PDL style μ calculus is axiomatised by the axioms and rules of PDL plus the μ axiom and the μ rule of inference.

3.6.9 Bisimulation

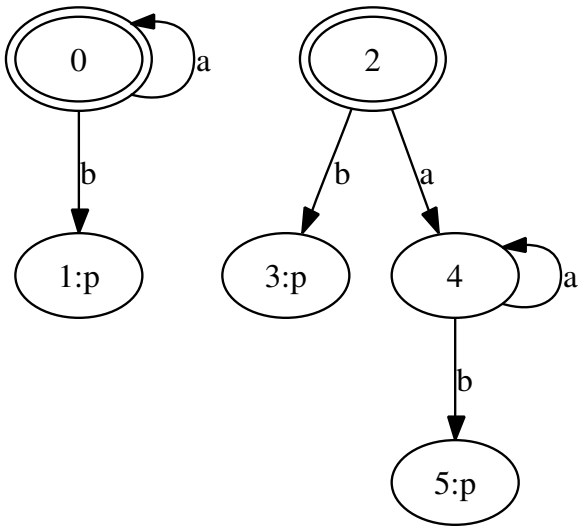
PDL and modal μ calculus are both interpreted in LTSs. But the correspondence between LTSs and processes is not one-to-one. The process that produces an infinite number of a transitions and nothing else can be represented as an LTS in lots of different ways. The following representations are all equivalent:



The notion of bisimulation is intended to capture such process equivalences. A bisimulation C between LTSs $\mathbf{M}$ and $\mathbf{N}$ is a relation on $S_{\mathbf{M}} \times S_{\mathbf{N}}$ such that if sCt then the following hold:

Invariance $V_{\mathbf{M}}(s) = V_{\mathbf{N}}(t)$ (the two states have the same valuation),
Zig if for some $a \in S_1$ $s \xrightarrow{a} s' \in R_{\mathbf{M}}$ then there is a $t' \in S_2$ with $t \xrightarrow{a} t' \in R_{\mathbf{N}}$ and $s'Ct'$.
Zag same requirement in the other direction.

One uses $\mathbf{M}, s \leftrightarrows \mathbf{N}, t$ to indicate that there is a bisimulation that connects s and t. In such a case one says that s and t are bisimilar.



In the LTSs of the picture, $0 \leftrightarrows 2 \leftrightarrows 4$ and $1 \leftrightarrows 3 \leftrightarrows 5$.

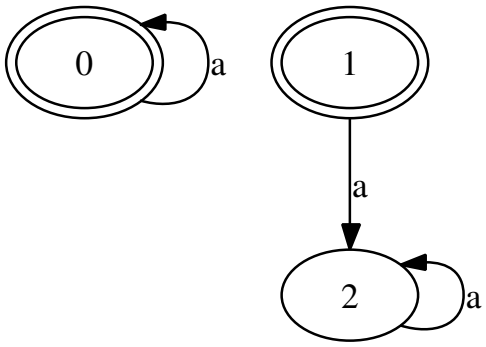
Bisimulation is intimately connected to modal logic, as follows. Modal logic is a sublogic of PDL. It is given by restricting the set of programs to atomic programs. Usually, one writes $\Diamond_a$ for $\langle a \rangle$:

$$\varphi \quad ::= \quad \top \mid p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle a \rangle \varphi$$

Bisimulations can be viewed as a motivation for modal logic. A global property of LTSs is a function $\mathbf{P}$ that assigns to any LTS $\mathbf{M}$ over a given signature a property $\mathbf{P}_{\mathbf{M}} \subseteq S_{\mathbf{M}}$. A global property $\mathbf{P}$ is *invariant for bisimulation* if whenever C is a bisimulation between $\mathbf{M}$ and $\mathbf{N}$ with sCt, then $s \in \mathbf{P}_{\mathbf{M}}$ iff $t \in \mathbf{P}_{\mathbf{N}}$.

Modal formulae may be viewed as global properties, for if φ is a modal formula, then $\lambda\mathbf{M}.[\![\varphi]\!]^{\mathbf{M}}$ is a global property. Similarly for formulae of first order logic.

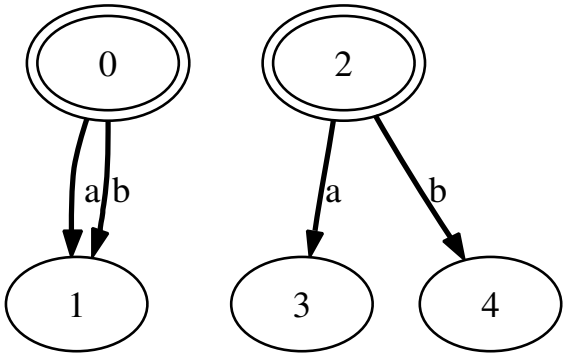
An example of a first order logic formula that is *not* invariant for bisimulation is the formula $R_a(x, x)$. This formula is true in state 0, but false in bisimilar state 1 in the following picture:



Another example of a first order logic formula that is *not* invariant for bisimulation:

$$\varphi(x) = \exists y(R_a(x,y) \wedge R_b(x,y)).$$

The picture below indicates that $\varphi(x)$ is not invariant for the example bisimulation that links 0 to 2 and 1 to 3 and 4. The state 0 satisfies $\varphi(x)$ while 2 does not, and the two states are bisimilar.



Clearly, all modal formulae are invariant for bisimulation: If φ is a modal formula that is true of a state s, and s is bisimilar to t, then an easy induction on the structure of φ establishes that φ is true of t as well.

More surprisingly, it turns out that all first order formulae that are invariant for bisimulation are translations of modal formulae. If first order logic is given and bisimulation is given, modal logic results from the following theorem:

Theorem 5 (Van Benthem, [13]) *A first order formula $\varphi(x)$ is* invariant *for bisimulation iff $\varphi(x)$ is equivalent to a modal formula.*

One direction of this can easily be verified by the reader: if φ is a modal formula, it can be proved by induction on formula structure that φ cannot distinguish between bisimilar points.

The argument for the other direction is more involved. We give a sketch of the proof. Define Ψ as the set of modal formulae that are implied by $\varphi(x)$, as follows:

$$\Psi := \{\psi \mid \psi \text{ is a modal formula and } \varphi(x) \models \psi\}.$$

Next, if we can prove that $\Psi \models \varphi(x)$, then the compactness theorem for FOL gives us $\{\psi_1, \ldots, \psi_n\} \subseteq \Psi$ with $\psi_1, \ldots, \psi_n \models \varphi(x)$, and we see that $\varphi(x)$ is equivalent to the modal formula $\psi_1 \wedge \cdots \wedge \psi_n$.

So suppose $\mathbf{M} \models_s \Psi$. We are done if we can show that $\mathbf{M} \models_s \varphi(x)$. For this, consider the modal theory of s, i.e., the set of modal formulae true at s:

$$\Phi := \{\varphi \mid \varphi \text{ is a modal formula and } \mathbf{M} \models_s \varphi\}.$$

Now $\Phi \cup \{\varphi(x)\}$ must be finitely satisfiable (i.e., any finite subset must be satisfiable), for if not then there are $\varphi_1, \ldots, \varphi_n \in \Phi$ with $\varphi(x) \models \neg\varphi_1 \vee \cdots \vee \neg\varphi_n$, which contradicts the fact that $\neg\varphi_1 \vee \cdots \vee \neg\varphi_n$ is false at s. Using the compactness theorem for FOL again, we see that there must be some node t in an LTS $\mathbf{N}$ with $\mathbf{N} \models_t \Phi \cup \{\varphi(x)\}$.

There is one given that we haven't used yet: $\varphi(x)$ is invariant for bisimulation. To use that given, we replace $\mathbf{M}$ and $\mathbf{N}$ by so-called ω saturated elementary extensions $\mathbf{M}^\bullet$ and $\mathbf{N}^\bullet$.

A FOL model $\mathbf{M}$ is ω saturated if whenever $\Phi(x, y_1, \ldots, y_n)$ is a set of first order formulae, and $d_1, \ldots, d_n$ are elements of the domain of $\mathbf{M}$, then $\Phi[x, d_1, \ldots, d_n]$ is finitely satisfiable, i.e., for every finite subset Φ_0 of Φ we can find a d in the domain of $\mathbf{M}$ with $\mathbf{M} \models \Phi[d, d_1, \ldots, d_n]$.

Every FO model has a an ω saturated elementary extension (see Chang and Keisler [23, Ch 6] for a proof), so the replacement of $\mathbf{M}, \mathbf{N}$ by $\mathbf{M}^\bullet, \mathbf{N}^\bullet$ is warranted. Moreover, $\mathbf{N}^\bullet \models \varphi(x)$, for truth of $\varphi(x)$ is preserved under the extension.

Lemma: If $\mathbf{M}, \mathbf{N}$ are ω saturated, then the relation of modal equivalence is a bisimulation between them.

Proof of the lemma: Let $\mathbf{M}, \mathbf{N}$ be ω saturated. Let $\equiv$ be the relation of being modally equivalent. Let $\mathbf{M}, s \equiv \mathbf{N}, t$. We show that $s \leftrightarrows t$, by checking the clauses for bisimulation:

Invariance Clearly, s and t have the same valuation.

Zig Suppose $s \xrightarrow{a} s'$. Let Φ be the set of modal formulae that are true at s'. Then for every finite subset Φ_0 of Φ, $\mathbf{M} \models_s \langle a \rangle \bigwedge \Phi_0$. Since $s \equiv t$, $\mathbf{M} \models_t \langle a \rangle \bigwedge \Phi_0$, so there is a t' with $t \xrightarrow{a} t'$ and $\mathbf{M} \models_{t'} \Phi_0$. Thus, Φ is finitely satisfiable in a successors of t. By the fact that $\mathbf{N}$ is ω saturated, it follows that there is a t' with $t \xrightarrow{a} t'$ and $\mathbf{N} \models_{t'} \Phi$.

Zag Same argument in the other direction.

Back to the main proof. $\mathbf{N}^\bullet \models_t \Phi \wedge \varphi(x)$ and $\mathbf{M}^\bullet \models_s \Phi$, where Φ is the modal theory of s. Thus, s, t have the same modal theory, and invoking the lemma

we see that $s \leftrightarrows t$. Since $\varphi(x)$ is invariant for bisimulation, $\mathbf{M}^\bullet \models_s \varphi(x)$, hence $\mathbf{M} \models_s \varphi(x)$. ∎

Bisimulations are also intimately connected to PDL, as follows.

A global relation is a function $\mathbf{R}$ that assigns to any LTS $\mathbf{M}$ over a given signature a relation $\mathbf{R_M} \subseteq S_\mathbf{M} \times S_\mathbf{M}$. A global relation $\mathbf{R}$ is *safe for bisimulation* if whenever C is a bisimulation between $\mathbf{M}$ and $\mathbf{N}$ with sCt, then:

Zig: if $s\mathbf{R_M}s'$ then there is a t' with $t\mathbf{R_N}t'$ and $s'Ct'$,
Zag: vice versa: if $t\mathbf{R_N}t'$ then there is an s' with $s\mathbf{R_M}s'$ and $s'Ct'$.

An example of a relation that is not safe for bisimulation is the relation given by the following first order formula:

$$\varphi(x, y) = R_a(x, y) \wedge x = y.$$

Look at the counterexample picture for invariance of $R_a(x, x)$ again. Formula $\varphi(x, y)$ is true of state pair $(0, 0)$ and false of the state pair $(1, 2)$ in that picture, but 0 and 1 are bisimilar, and $(0, 0)$ satisfies the zig, and $(1, 2)$ the zag condition for bisimulation.

Another counterexample for safety for bisimulation is provided by the following formula:

$$\psi(x, y) = R_a(x, y) \wedge R_b(x, y).$$

Look at the counterexample picture for invariance of $\exists y(R_a(x, y) \wedge R_b(x, y))$ again. Formula $\psi(x, y)$ is true of state pair $(0, 1)$ and false of state pairs $(2, 3)$ and $(2, 4)$, while 0 and 2 are bisimilar, $(0, 1)$ satisfies the zig condition, and both $(2, 3)$ and $(2, 4)$ satisfy the zag condition for bisimulation.

In fact, invariance for bisimulation and safety for bisimulation are closely connected. If $\varphi(x)$ is invariant for bisimulation then $\varphi(x) \wedge x = y$ is safe for bisimulation. Conversely, if $\varphi(x, y)$ is safe for bisimulation, and P is some unary predicate that does not occur in φ then $\exists y(\varphi(x, y) \wedge P(y))$ is invariant for bisimulation.

Note that the notion of safety for bisimulation generalises the zig and zag conditions of bisimulations, while invariance for bisimulation generalises the invariance condition of bisimulations.

A modal program is a PDL program that does not contain *. Modal programs can be viewed as global relations, for if α is a modal program, then $\lambda\mathbf{M}.[\![\alpha]\!]^\mathbf{M}$ is a global relation.

It is not difficult to see that all modal programs are safe for bisimulation. The surprising thing is the converse: all first order relations that are safe for bisimulation turn out to be translations of modal programs.

Theorem 6 (Van Benthem [14]) *A first order formula $\varphi(x, y)$ is* safe *for bisimulation iff $\varphi(x, y)$ is equivalent to a modal program.*

Proofs of this can be found in [14, 73]. The perspective on Van Benthem's characterisations of modal logic and PDL is from [73]. In fact, Van Benthem gives a slightly different characterisation. He proves that any bisimulation safe first order formula can be generated from atomic tests $?p$, atomic actions a, sequential composition $;$, choice $\cup$ and *dynamic negation* $\sim$, where $\sim\alpha$ is interpreted by:

$$[\![\sim\alpha]\!]^{\mathbf{M}} \quad = \quad \{(s,s) \in S_{\mathbf{M}} \times S_{\mathbf{M}} \mid \neg\exists t (s,t) \in [\![\alpha]\!]^{\mathbf{M}}\}$$

The two characterisations are equivalent, for $\sim\alpha$ is definable as the PDL program $?([\alpha]\bot)$, while any modal PDL test $?\varphi$ can be expressed in terms of dynamic negation using the following translation:

$$\begin{array}{rcl} (?\top)^\circ & = & \sim\bot \\ (?(\varphi_1 \vee \varphi_2))^\circ & = & (?\varphi_1)^\circ \vee (?\varphi_2)^\circ \\ (?\neg\varphi)^\circ & = & \sim(?\varphi)^\circ \\ (?\langle\alpha\rangle\varphi)^\circ & = & \sim\sim(\alpha \ ; \ (?\varphi)^\circ) \end{array}$$

Looking at PDL programs from an algebraic perspective, the obvious notion to be axiomatised is that of PDL program equivalence. A calculus that produces precisely the equations of the form $\alpha_1 = \alpha_2$ for those α_1, α_2 that have the same interpretation in any PDL model is given in [71] (see also [72], where equivalence of modal PDL programs is axiomatised). The axiomatisation has the following quasi-equations between programs:

associativity of $;$	$\alpha \ ; \ (\beta \ ; \ \gamma) = (\alpha \ ; \ \beta) \ ; \ \gamma$
associativity for $\cup$	$\alpha \cup (\beta \cup \gamma) = (\alpha \cup \beta) \cup \gamma$
commutativity of $\cup$	$\alpha \cup \beta = \beta \cup \alpha$
idempotency of $\cup$	$\alpha \cup \alpha = \alpha$
left distributivity	$(\alpha \cup \beta) \ ; \ \gamma = ((\alpha \ ; \ \gamma) \cup (\beta \ ; \ \gamma))$
right distributivity	$\alpha \ ; \ (\beta \cup \gamma) = ((\alpha \ ; \ \beta) \cup (\alpha \ ; \ \gamma))$
left identity	$?\top \ ; \ \alpha = \alpha$
right identity	$\alpha \ ; ?\top = \alpha$
left zero	$?\bot \ ; \ \alpha = ?\bot$
right zero	$\alpha \ ; ?\bot = ?\bot$
zero sum	$\alpha \cup ?\bot = \alpha$
$*$ expansion	$\alpha^* = ?\top \cup (\alpha \ ; \ \alpha^*)$
left induction	$\alpha \ ; \ \beta \leq \beta \Rightarrow \alpha^* \ ; \ \beta \leq \beta$
right induction	$\beta \ ; \ \alpha \leq \beta \Rightarrow \beta \ ; \ \alpha^* \leq \beta$
test choice	$?(\varphi \vee \psi) = ?\varphi \cup ?\psi$
test sequence	$?(\varphi \wedge \psi) = ?\varphi \ ; \ ?\psi$
domain test	$?\langle\alpha\rangle\top \ ; \ \alpha = \alpha$

where $\alpha \leq \beta$ is defined as $\alpha \cup \beta = \beta$, and the following equations between booleans hold:

equations of boolean algebra	
choice	$\langle \alpha \cup \beta \rangle \varphi = \langle \alpha \rangle \varphi \vee \langle \beta \rangle \varphi$
sequence	$\langle \alpha \ ; \ \beta \rangle \varphi = \langle \alpha \rangle \langle \beta \rangle \varphi$
iteration	$\langle \alpha^* \rangle \varphi = \varphi \vee \langle \alpha \rangle \langle \alpha^* \rangle \varphi$
induction	$\langle \alpha^* \rangle \varphi = \varphi \vee \langle \alpha^* \rangle (\neg \varphi \wedge \langle \alpha \rangle \varphi)$
test diamond	$\langle ? \varphi \rangle \psi = \varphi \wedge \psi$

If one restricts attention to the modal part of PDL (PDL without *, for this is equivalent to multi-modal logic), the quasi-equations for * drop out, and an equational axiomatisation of modal PDL results.

We end with mentioning an intimate connection between modal μ calculus and bisimulation:

Theorem 7 (Janin and Walukiewicz [76]) *A monadic second order formula $\varphi(x)$ is invariant for bisimulation iff it is equivalent to the standard translation in monadic second order logic of a μ sentence.*

4 Analysing the Dynamics of Communication

Dynamic logic is the logic of action and the results of action, but it is also a branch of modal logic, and it enjoys the same breadth of applications as modal logic. What happens if we reinterpret the atomic action modalities as something else? In epistemic logic, atomic accessibilities denote epistemic similarity relations of agents in a multi-agent epistemic setting. Epistemic PDL is the result of reinterpreting the basic action modalities as epistemic relations. Now $[a;b]\varphi$ means that agent a knows that agent b knows that φ. This is more expressive than multi-agent epistemic logic. E.g., $[(a \cup b)^*]\varphi$ expresses that φ is common knowledge among a and b, and it is well known that common knowledge for a, b cannot be expressed in terms of basic modalities $[a], [b]$ alone.

As an aside, expressing *implicit knowledge* would require extending epistemic PDL with an intersection operation. Implicit knowledge among a, b that φ can be expressed in this extended language as $[a \cap b]\varphi$. This extension results in a logic that is still decidable, but the invariance for bisimulation gets lost. Implicit knowledge will not concern us in what follows.

Interestingly, the shift of application from computation to epistemics turns PDL into a description tool for static situations, for under this interpretation LTSs denote multi-agent epistemic situations instead of sets of computations within a set of states. Still, at a higher level, there is again a dynamic turn. We can study how multi-agent epistemic situations evolve as a result of communicative actions. An important example of such actions is public announcement. What happens to the knowledge of a set of participating agents if it is suddenly announced to all that φ is the case? On the assumption that none of the agents takes φ to be impossible, this should result in a new epistemic state of affairs where it is common knowledge among the agents that φ. In this section we will see that epistemic PDL (PDL, with the basic modalities interpreted as epistemic relations) is eminently suited for the analysis of the dynamics of communication.

Dynamic epistemic logic (cf., e.g., [5–8]) analyses the changes in epistemic information among sets of agents that result from various communicative actions, such as public announcements, group messages and individual messages. The logics studied in [8] add information update operations to epistemic description languages with a common knowledge operator, in such a way that the addition increases expressive power. This makes axiomatisations complicated and completeness proofs hard. In [85] it is demonstrated how update axioms can be made susceptible to reduction axioms, by the simple means of switching to more expressive epistemic description languages. In particular, it is shown in [85] how generic updates with epistemic actions can be axiomatised in automata PDL [63, Chapter 10.3].

We will follow [37] in giving a direct reduction of the logic of generic updates with epistemic actions in the style of [7, 8] to PDL.

4.1 System

Let $\mathcal{L}$ be a language that can be interpreted in labelled transition systems. Then action models for $\mathcal{L}$ look like this:

Definition 8 (Action models for $\mathcal{L}$, Ag) *Let a set of agents* Ag *and an LTS language $\mathcal{L}$ with label set* Ag *be given. An action model for $\mathcal{L}$, Ag is a triple*

$$A = ([s_0, \ldots, s_{n-1}], pre, T)$$

where $[s_0, \ldots, s_{n-1}]$ is a finite list of action states, $pre : \{s_0, \ldots, s_{n-1}\} \to \mathcal{L}$ assigns a precondition to each action state, and $T : \mathrm{Ag} \to \mathcal{P}(\{s_0, \ldots, s_{n-1}\}^2)$ assigns an accessibility relation $\xrightarrow{a}$ to each agent $a \in \mathrm{Ag}$.

$\mathcal{L}$ actions can be executed in labelled transition systems for $\mathcal{L}$, by means of the following product construction:

Definition 9 (Action Update) *Let an LTS $\mathbf{M} = (W, V, R)$, a world $w \in W$, and a pointed action model (A, s), with $A = ([s_0, \ldots, s_{n-1}], pre, T)$, be given. Then the result of executing (A, s) in $(\mathbf{M}, w)$ is the model $(\mathbf{M} \otimes A, (w, s))$, with $\mathbf{M} \otimes A = (W', V', R')$, where*

$$\begin{aligned} W' &= \{(w, s) \mid s \in \{s_0, \ldots, s_{n-1}\}, w \in [\![pre(s)]\!]^{\mathbf{M}}\} \\ V'(w, s) &= V(w) \\ R'(a) &= \{((w, s), (w', s')) \mid (w, w') \in R(a), (s, s') \in T(a)\}. \end{aligned}$$

For the set of basic propositions P and the set of agents Ag, the language of $\mathrm{PDL}^{\mathrm{DEL}}$ (which we will call 'update PDL') over P, Ag is like that for standard PDL over P, Ag, but with a construct for action update added: if φ is an update PDL formula, and $[A, s]$ is a single pointed action model, then $[A, s]\varphi$ is an update PDL formula. If B is a set of agents $\{b_1, \ldots, b_n\}$, then we abbreviate $b_1 \cup \cdots \cup b_n$ as B. Now $[B]\varphi$ expresses that φ is *general* knowledge among B (they all know φ, but they need not know that the others know φ) and $[B^*]\varphi$ expresses that φ is *common* knowledge among B (they all know φ and they all know that the others know φ).

The semantics of $\mathrm{PDL}^{\mathrm{DEL}}$ is given by the standard PDL clauses, with the following clause for update added:

$$[\![[A, s]\varphi]\!]^{\mathbf{M}} = \{w \in W_{\mathbf{M}} \mid \text{if } \mathbf{M} \models_w \mathrm{pre}(s) \text{ then } (w, s) \in [\![\varphi]\!]^{\mathbf{M} \otimes A}\}.$$

Using $\langle A, s\rangle\varphi$ as shorthand for $\neg[A, s]\neg\varphi$, we see that the interpretation for $\langle A, s\rangle\varphi$ turns out as:

$$[\![\langle A, s\rangle\varphi]\!]^{\mathbf{M}} = \{w \in W_{\mathbf{M}} \mid \mathbf{M} \models_w \mathrm{pre}(s) \text{ and } (w, s) \in [\![\varphi]\!]^{\mathbf{M} \otimes A}\}.$$

Updating with multiple pointed update actions is also possible. A multiple pointed action is a pair (A, S), with A an action model, and S a subset of the state set of A. Extend the language with updates $[A, S]\varphi$, and interpret this as follows:

$$[\![[A, S]\varphi]\!]^{\mathbf{M}} \quad = \quad \{w \in W_{\mathbf{M}} \mid \forall s \in S(\text{ if } \mathbf{M} \models_w \text{pre}(s) \text{ then } \mathbf{M} \otimes A \models_{(w,s)} \varphi)\}.$$

The reason to employ multiple pointed models for updating is that it allows us to handle choice. Suppose we want to model the action of testing whether φ followed by a public announcement of the result. More precisely:

> A test is performed to check whether φ holds in the actual world. If the outcome of the test is affirmative, then φ gets announced. If the test reveals that φ does not hold, then $\neg\varphi$ gets announced.

Single pointed update models do not allow us to model this.

Theorem 10 (Preservation of bisimulation; Baltag, Moss, Solecki) *The action update operation $\otimes$ preserves bisimulation on epistemic models:*

$$\textit{if } \mathbf{M} \underline{\leftrightarrow} \mathbf{N} \textit{ then } \mathbf{M} \otimes \mathbf{A} \underline{\leftrightarrow} \mathbf{N} \otimes \mathbf{A}.$$

We can also look at the update models modulo action bisimulation. An action bisimulation is like an ordinary bisimulation, with the clause for 'same valuations' replaced by a clause for 'equivalent preconditions'.

Theorem 11 (Preservation of action bisimulation) *The action update operation preserves action bisimulation:*

$$\textit{if } \mathbf{A} \underline{\leftrightarrow} \mathbf{B} \textit{ then } \mathbf{M} \otimes \mathbf{A} \underline{\leftrightarrow} \mathbf{M} \otimes \mathbf{B}.$$

Proof. Let Z be a bisimulation between $\mathbf{A}$ and $\mathbf{B}$. Define a relation relation on $\mathbf{M} \otimes \mathbf{A} \times \mathbf{M} \otimes \mathbf{B}$ by means of

$$(u, s)C(v, t) \text{ iff } u = v \text{ and } sZt.$$

It is easily shown that this is a bisimulation. □

4.2 Logics of Communication

In terms of the system just defined a variety of types of communicative actions can be described. The two most important ones are public announcements and group announcements.

4.2.1 Public Announcements

The language of **public announcements** is the language that one gets if one allows action models for public announcement. The action model for public announcement that φ consists of a single state s_0 with precondition φ and epistemic relation $\{s_0 \stackrel{a}{\rightarrow} s_0 \mid a \in Ag\}$. Call this model P_φ.

The following equivalence shows how public announcement relates to common knowledge among set of agents B:

$$[P_\varphi, s_0][B^*]\psi \leftrightarrow [(?\varphi; B)^*][P_\varphi, s_0]\psi. \tag{6}$$

What this says is that after public announcement with φ it is common knowledge among B that ψ if and only if before the update it holds at the end of every $(?\varphi \ ; \ B)^*$ path through the model that a public update with φ will result in ψ. Axiomatisations of public announcement logic are given in [103] and [45,46], for a language that cannot express common knowledge. An axiomatisation for a language with a common knowledge operator is given in [85]. Below we will show how this equivalence emerges in the axiomatisation of $\mathrm{PDL}^{\mathrm{DEL}}$ from [37].

4.2.2 Group Announcements

The language of **group announcements** is the result of allowing action models for group messages. These will be defined below. Similarly, we can define the languages of **secret group communications**, of **individual messages**, of **tests**, of **lies**, and so on [5]. All these languages are comprised in the language of $\mathrm{PDL}^{\mathrm{DEL}}$, because all these communicative actions can be characterised by appropriate action models.

4.3 Program Transformation

We will now show how $\mathrm{PDL}^{\mathrm{DEL}}$ formulae can be reduced to PDL formulae. For every action model A with states $s_0, \ldots, s_{n-1}$ we define a set of n^2 program transformers $T^A_{i,j}$ $(0 \leq i < n, 0 \leq j < n)$, as follows:

$$\begin{aligned}
T^A_{ij}(a) &= \begin{cases} ?\mathrm{pre}(s_i) \ ; \ a & \text{if } s_i \stackrel{a}{\rightarrow} s_j, \\ ?\bot & \text{otherwise} \end{cases} \\
T^A_{ij}(?\varphi) &= \begin{cases} ?\varphi & \text{if } i = j, \\ ?\bot & \text{otherwise} \end{cases} \\
T^A_{ij}(\pi_1; \pi_2) &= \bigcup_{k=0}^{n-1} (T^A_{ik}(\pi_1) \ ; \ T^A_{kj}(\pi_2)) \\
T^A_{ij}(\pi_1 \cup \pi_2) &= T^A_{ij}(\pi_1) \cup T^A_{ij}(\pi_2) \\
T^A_{ij}(\pi^*) &= K^A_{ijn}(\pi)
\end{aligned}$$

where $K^A_{ijk}(\pi)$ is a (transformed) program for all the π^* paths from s_i to s_j that can be traced through A while avoiding a pass through intermediate states s_k and higher. Thus, $K^A_{ijn}(\pi)$ is a program for all the π^* paths from s_i to s_j that can be traced through A, period.

$K^A_{ijk}(\pi)$ is defined by recursion on k, as follows:

$$K^A_{ij0}(\pi) \;=\; \begin{cases} ?\top \cup T^A_{ij}(\pi) & \text{if } i = j, \\ T^A_{ij}(\pi) & \text{otherwise} \end{cases}$$

$$K^A_{ij(k+1)}(\pi) \;=\; \begin{cases} (K^A_{kkk}(\pi))^* & \text{if } i = k = j, \\ (K^A_{kkk}(\pi))^*; K^A_{kjk}(\pi) & \text{if } i = k \neq j, \\ K^A_{ikk}(\pi) \;;\; (K^A_{kkk}(\pi))^* & \text{if } i \neq k = j, \\ K^A_{ijk}(\pi) \cup (K^A_{ikk}(\pi) \;;\; (K^A_{kkk}(\pi))^* \;;\; K^A_{kjk}(\pi)) & \text{otherwise } (i \neq k \neq j). \end{cases}$$

For some runs through example applications of these definitions, see section 4.5 below.

Lemma 12 (Kleene Path) *Suppose $(w, w') \in [\![T^A_{ij}(\pi)]\!]^{\mathbf{M}}$ iff there is a π path from (w, s_i) to (w', s_j) in $\mathbf{M} \otimes A$. Then $(w, w') \in [\![K^A_{ijn}(\pi)]\!]^{\mathbf{M}}$ iff there is a π^* path from (w, s_i) to (w', s_j) in $\mathbf{M} \otimes A$.*

Proof. Use the definition of K^A_{ijk} to prove by induction on k that $(w, w') \in [\![K^A_{ijk}(\pi)]\!]^{\mathbf{M}}$ iff there is a π^* path from (w, s_i) to (w', s_j) in $\mathbf{M} \otimes A$ that does not pass through any pairs (v, s) with $s \in \{s_k, \ldots, s_{n-1}\}$.

Base case, $i = j$: A π^* path from (w, s_i) to (w', s_j) that does not visit any intermediate states is either the empty path or a single π step from (w, s_i) to (w', s_j). Such a path exists iff $(w, w') \in [\![?\top \cup T^A_{ij}]\!]^{\mathbf{M}}$ iff $(w, w') \in [\![K^A_{ij0}(\pi)]\!]^{\mathbf{M}}$.

Base case, $i \neq j$: A π^* path from (w, s_i) to (w', s_j) that does not visit any intermediate states is a single π step from (w, s_i) to (w', s_j). Such a path exists iff $(w, w') \in [\![T^A_{ij}]\!]^{\mathbf{M}}$ iff $(w, w') \in [\![K^A_{ij0}(\pi)]\!]^{\mathbf{M}}$.

Induction step. Assume that $(w, w') \in [\![K^A_{ijk}(\pi)]\!]^{\mathbf{M}}$ iff there is a π^* path from (w, s_i) to (w', s_j) in $\mathbf{M} \otimes A$ that does not pass through any pairs (v, s) with $s \in \{s_k, \ldots, s_{n-1}\}$.

Case $i = k = j$. A path from (w, s_i) to (w', s_j) in $\mathbf{M} \otimes A$ that does not pass through any pairs (v, s) with $s \in \{s_{k+1}, \ldots, s_{n-1}\}$ now consists of an arbitrary number of π^* paths from s_k to s_k that do not visit any intermediate states with action component s_k or higher. By the induction hypothesis, such a path exists iff $(w, w') \in [\![(K^A_{kkk}(\pi))^*]\!]^{\mathbf{M}}$ iff $(w, w') \in [\![K^A_{ij(k+1)}(\pi)]\!]^{\mathbf{M}}$.

Case $i = k \neq j$. A path from (w, s_i) to (w', s_j) in $\mathbf{M} \otimes A$ that does not pass through any pairs (v, s) with $s \in \{s_{k+1}, \ldots, s_{n-1}\}$ now consists of a π^* path starting in (w, s_k) visiting states of the form (u, s_k) an arbitrary number of times, but never touching on states with action component s_k or higher in between, and ending in (v, s_k), followed by a π^* path from (v, s_k) to (w', s_j) that does not pass through any pairs (v, s) with $s \in \{s_k, \ldots, s_{n-1}\}$. By the induction hypothesis, a path from (w, s_k) to (v, s_k) of the first kind exists iff $(w, v) \in [\![(K^A_{kkk}(\pi))^*]\!]^{\mathbf{M}}$. Again by the induction hypothesis, a path from (v, s_k) to (w', s_j) of the second kind exists iff $(v, w') \in [\![K^A_{kjk}]\!]^{\mathbf{M}}$. Thus, the required path from (w, s_i) to (w', s_j) in $\mathbf{M} \otimes A$ exists iff $(w, w') \in [\![(K^A_{kkk}(\pi))^*; K^A_{kjk}(\pi)]\!]^{\mathbf{M}}$ iff $(w, w') \in [\![K^A_{ij(k+1)}(\pi)]\!]^{\mathbf{M}}$.

The other two cases are similar. □

The Kleene path lemma is the key ingredient in the following program transformation lemma.

Lemma 13 (Program Transformation) *Assume A has n states $s_0, \ldots, s_{n-1}$. Then:*

$$\mathbf{M} \models_w [A, s_i][\pi]\varphi \text{ iff } \mathbf{M} \models_w \bigwedge_{j=0}^{n-1} [T^A_{ij}(\pi)][A, s_j]\varphi.$$

Proof. Induction on the complexity of π.

Basis, epistemic link case:

$$\begin{aligned}
& \mathbf{M} \models_w [A, s_i][a]\varphi \\
\text{iff} \quad & \mathbf{M} \models_w \text{pre}(s_i) \text{ implies } \mathbf{M} \otimes A \models_{(w,s_i)} [a]\varphi \\
\text{iff} \quad & \mathbf{M} \models_w \text{pre}(s_i) \text{ implies for all } s_j \in A, \text{ all } w' \in \mathbf{M}: \\
& \text{if } s_i \xrightarrow{a} s_j, w \xrightarrow{a} w', \text{ then } \mathbf{M} \models_{w'} [A, s_j]\varphi \\
\text{iff} \quad & \text{for all } s_j \in A: \text{ if } s_i \xrightarrow{a} s_j \text{ then } \mathbf{M} \models_w [\text{pre}(s_i)\ ;\ a][A, s_j]\varphi \\
\text{iff} \quad & \mathbf{M} \models_w \bigwedge_{j=0}^{n-1} [T^A_{ij}(a)][A, s_j]\varphi.
\end{aligned}$$

Basis, test case:

$$\begin{aligned}
& \mathbf{M} \models_w [A, s_i][?\psi]\varphi \\
\text{iff} \quad & \mathbf{M} \models_w \text{pre}(s_i) \text{ implies } \mathbf{M} \otimes A \models_{(w,s_i)} [?\psi]\varphi \\
\text{iff} \quad & \mathbf{M} \models_w \text{pre}(s_i) \text{ implies } \mathbf{M} \models_w [?\psi][A, s_i]\varphi \\
\text{iff} \quad & \mathbf{M} \models_w \bigwedge_{j=0}^{n-1} [T^A_{ij}(?\psi)][A, s_j]\varphi.
\end{aligned}$$

Induction step, cases $\pi_1\ ;\ \pi_2$ and $\pi_1 \cup \pi_2$ are straightforward. The case of π^* is settled with the help of the Kleene path lemma. □

4.4 Reduction Axioms for Update PDL

The program transformations can be used to translate $\mathrm{PDL}^{\mathrm{DEL}}$ to PDL by means of the following mutually recursive definitions of translations t for formulae and r for programs:

$$\begin{array}{rcl}
t(\top) & = & \top \\
t(p) & = & p \\
t(\neg\varphi) & = & \neg t(\varphi) \\
t(\varphi_1 \wedge \varphi_2) & = & t(\varphi_1) \wedge t(\varphi_2) \\
t([\pi]\varphi) & = & [r(\pi)]t(\varphi) \\
t([A,s]\top) & = & \top \\
t([A,s]p) & = & t(\mathrm{pre}(s)) \rightarrow p \\
t([A,s]\neg\varphi) & = & t(\mathrm{pre}(s)) \rightarrow \neg t([A,s]\varphi) \\
t([A,s](\varphi_1 \wedge \varphi_2)) & = & t([A,s]\varphi_1) \wedge t([A,s]\varphi_2) \\
t([A,s_i][\pi]\varphi & = & \bigwedge_{j=0}^{n-1} [T^A_{ij}(r(\pi))]t([A,s_j]\varphi) \\
t([A,s][A',s']\varphi & = & t([A,s]t([A',s']\varphi)) \\
 & & \\
r(a) & = & a \\
r(?\varphi) & = & ?t(\varphi) \\
r(\pi_1;\pi_2) & = & r(\pi_1);r(\pi_2) \\
r(\pi_1 \cup \pi_2) & = & r(\pi_1) \cup r(\pi_2) \\
r(\pi^*) & = & (r(\pi))^*.
\end{array}$$

The correctness of this translation follows from direct semantic inspection, using the program transformation lemma for the translation of $[A,s_i][\pi]\varphi$ formulae. The translation points the way to appropriate reduction axioms, as follows.

Take all axioms and rules of PDL [42, 99, 113], plus the following reduction axioms:

$$\begin{array}{rcl}
[A,s]p & \leftrightarrow & (\mathrm{pre}(s) \Rightarrow p) \\
[A,s]\neg\varphi & \leftrightarrow & (\mathrm{pre}(s) \Rightarrow \neg[A,s]\varphi) \\
[A,s](\varphi_1 \wedge \varphi_2) & \leftrightarrow & ([A,s]\varphi_1 \wedge [A,s]\varphi_2) \\
[A,s_i][\pi]\varphi & \leftrightarrow & \bigwedge_{j=0}^{n-1} [T^A_{ij}(\pi)][A,s_j]\varphi
\end{array}$$

and necessitation for action model modalities. The reduction axioms for $[A,s]p$, $[A,s]\neg\varphi$ and $[A,s](\varphi_1 \wedge \varphi_2)$ are as in [85]. The final reduction axiom is based on program transformation and is new. Note that if we allow multiple action

models, we need the following reduction axiom for those:

$$[A, S]\varphi \quad \leftrightarrow \quad \bigwedge_{s \in S} [A, s]\varphi$$

If updates with multiple pointed action models are also in the language, we need the following additional reduction axiom:

$$[A, S]\varphi \quad \leftrightarrow \quad \bigwedge_{s \in S} [A, s]\varphi$$

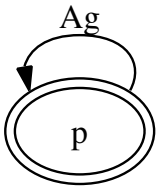
Theorem 14 (Completeness) *If* $\models \varphi$ *then* $\vdash \varphi$.

Proof. The proof system for PDL is complete, and every formula in the language of $\mathrm{PDL}^{\mathrm{DEL}}$ is provably equivalent to a PDL formula. □

4.5 Special Cases

4.5.1 Public Announcement and Common Knowledge

As introduced above, in section 4.2.1, the action model for public announcement that φ consists of a single state s_0 with precondition φ and epistemic relation $\{s_0 \xrightarrow{a} s_0 \mid a \in Ag\}$. We call this model P_φ.



We are interested in how public announcement that φ brings about common knowledge of ψ among group of agents B, i.e., we want to compute $[P_\varphi, s_0][B^*]\psi$. For this, we need $T_{00}^{P_\varphi}(B^*)$, which is defined as $K_{001}^{P_\varphi}(B)$.

To work out $K_{001}^{P_\varphi}(B)$, we need $K_{000}^{P_\varphi}(B)$, and for $K_{000}^{P_\varphi}(B)$, we need $T_{00}^{P_\varphi}(B)$, which turns out to be $\bigcup_{b \in B}(?\varphi \ ; \ b)$, or equivalently, $?\varphi; B$. Working upward from this, we get:

$$K_{000}^{P_\varphi}(B) = ?\top \cup T_{00}^{P_\varphi}(B) = ?\top \cup (?\varphi; B),$$

and therefore:

$$\begin{aligned} K_{001}^{P_\varphi}(B) &= (K_{000}^{P_\varphi}(B))^* \\ &= (?\top \cup (?\varphi; B))^* \\ &= (?\varphi; B)^*. \end{aligned}$$

Thus, the reduction axiom for the public announcement action P_φ with respect to the program for common knowledge among agents B, works out as follows:

$$\begin{array}{rcl}
[P_\varphi, s_0][B^*]\psi & \leftrightarrow & [P_\varphi, s_0][B^*]\psi \\
& \leftrightarrow & [T_{00}^{P_\varphi}(B^*)][P_\varphi, s_0]\psi \\
& \leftrightarrow & [K_{001}^{P_\varphi}(B)][P_\varphi, s_0]\psi \\
& \leftrightarrow & [(?\varphi; B)^*][P_\varphi, s_0]\psi.
\end{array}$$

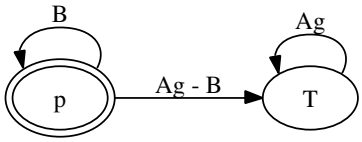
This expresses that every B path consisting of φ worlds ends in a $[P_\varphi, s_0]\psi$ world, i.e., it expresses exactly what is captured by the special purpose operator $C_B(\varphi, \psi)$ introduced in [85]. Indeed, the authors remark in a footnote that their proof system for $C_B(\varphi, \psi)$ essentially follows the usual PDL treatment for the PDL transcription of this formula.

4.5.2 Secret Group Communication and Common Belief

The logic of secret group communication is the logic of email CCs. The action model for a secret group message to B that φ consists of two states s_0, s_1, where s_0 has precondition φ and s_1 has precondition $\top$, and where the accessibilities T are given by:

$$\begin{array}{rcl}
T & = & \{s_0 \stackrel{b}{\rightarrow} s_0 \mid b \in B\} \\
& & \cup\{s_0 \stackrel{a}{\rightarrow} s_1 \mid a \in Ag - B\} \\
& & \cup\{s_1 \stackrel{a}{\rightarrow} s_1 \mid a \in Ag\}.
\end{array}$$

The actual world is s_0. The members of B are aware that action φ takes place; the others think that nothing happens. In this they are mistaken, which is why CC updates generate KD45 models: i.e., CC updates make knowledge degenerate into belief.



We work out the program transformations that this update engenders for common knowledge among some group of agents D. Call the action model CC_φ^B.

We will have to work out $K_{002}^{\mathrm{CC}_\varphi^B} D$, $K_{012}^{\mathrm{CC}_\varphi^B} D$, $K_{112}^{\mathrm{CC}_\varphi^B} D$, $K_{102}^{\mathrm{CC}_\varphi^B} D$.

For these, we need $K_{001}^{\mathrm{CC}_\varphi^B} D$, $K_{011}^{\mathrm{CC}_\varphi^B} D$, $K_{111}^{\mathrm{CC}_\varphi^B} D$, $K_{101}^{\mathrm{CC}_\varphi^B} D$.

For these in turn, we need $K_{000}^{\mathrm{CC}_\varphi^B} D$, $K_{010}^{\mathrm{CC}_\varphi^B} D$, $K_{110}^{\mathrm{CC}_\varphi^B} D$, $K_{100}^{\mathrm{CC}_\varphi^B} D$.

For these, we need:

$$\begin{aligned}
T_{00}^{\mathrm{CC}_\varphi^B} D &= \bigcup_{d \in B \cap D} (?\varphi \ ; \ d) \ = ?\varphi \ ; \ (B \cap D) \\
T_{01}^{\mathrm{CC}_\varphi^B} D &= \bigcup_{d \in D - B} (?\varphi \ ; \ d) \ = ?\varphi \ ; \ (D - B) \\
T_{11}^{\mathrm{CC}_\varphi^B} D &= D \\
T_{10}^{\mathrm{CC}_\varphi^B} D &= ?\bot
\end{aligned}$$

It follows that:

$$\begin{aligned}
K_{000}^{\mathrm{CC}_\varphi^B} D &= ?\top \cup (?\varphi \ ; \ (B \cap D)) \\
K_{010}^{\mathrm{CC}_\varphi^B} D &= ?\varphi \ ; \ (D - B) \\
K_{110}^{\mathrm{CC}_\varphi^B} D &= ?\top \cup D, \\
K_{100}^{\mathrm{CC}_\varphi^B} D &= ?\bot
\end{aligned}$$

From this we can work out the K_{ij1}, as follows:

$$\begin{aligned}
K_{001}^{\mathrm{CC}_\varphi^B} D &= (?\varphi \ ; \ (B \cap D))^* \\
K_{011}^{\mathrm{CC}_\varphi^B} D &= (?\varphi \ ; \ (B \cap D))^* \ ; \ (D - B) \\
K_{111}^{\mathrm{CC}_\varphi^B} D &= ?\top \cup D \\
K_{101}^{\mathrm{CC}_\varphi^B} D &= ?\bot.
\end{aligned}$$

Finally, we get K_{002} and K_{012} from this:

$$\begin{aligned}
K_{002}^{\mathrm{CC}_\varphi^B} D &= K_{001}^{\mathrm{CC}_\varphi^B} D \cup K_{011}^{\mathrm{CC}_\varphi^B} D \ ; \ (K_{111}^{\mathrm{CC}_\varphi^B} D)^* \ ; \ K_{101}^{\mathrm{CC}_\varphi^B} D \\
&= K_{001}^{\mathrm{CC}_\varphi^B} D \ \text{ (since the right-hand expression evaluates to } ?\bot) \\
&= (?\varphi \ ; \ (B \cap D))^* \\
K_{012}^{\mathrm{CC}_\varphi^B} D &= K_{011}^{\mathrm{CC}_\varphi^B} D \cup K_{011}^{\mathrm{CC}_\varphi^B} D \ ; \ (K_{111}^{\mathrm{CC}_\varphi^B} D)^* \\
&= K_{011}^{\mathrm{CC}_\varphi^B} D \ ; \ (K_{111}^{\mathrm{CC}_\varphi^B} D)^* \\
&= (?\varphi \ ; \ (B \cap D))^* \ ; \ (D - B) \ ; \ D^*.
\end{aligned}$$

Thus, the program transformation for common belief among D works out as follows:

$$\begin{aligned}
& [\mathrm{CC}_\varphi^B, s_0][D^*]\psi \\
\leftrightarrow \ & [(?\varphi \ ; \ (B \cap D))^*][\mathrm{CC}_\varphi^B, s_0]\psi \wedge [(?\varphi \ ; \ (B \cap D))^* \ ; \ (D - B) \ ; \ D^*][\mathrm{CC}_\varphi^B, s_1]\psi.
\end{aligned}$$

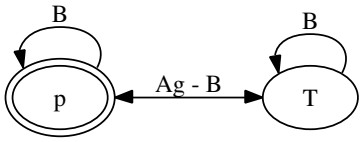
Compare [111] for a direct axiomatisation of the logic of CCs.

4.5.3 Group Messages and Common Knowledge

The action model for a group message to B that φ consists of two states s_0, s_1, where s_0 has precondition φ and s_1 has precondition $\top$, and where the accessibilities T are given by:

$$\begin{aligned} T &= \{s_0 \stackrel{b}{\to} s_0 \mid b \in B\} \\ &\quad \cup \{s_1 \stackrel{b}{\to} s_1 \mid b \in B\} \\ &\quad \cup \{s_0 \stackrel{a}{\to} s_1 \mid a \in Ag - B\} \\ &\quad \cup \{s_1 \stackrel{a}{\to} s_0 \mid a \in Ag - B\}. \end{aligned}$$

This captures the fact that the members of B can distinguish the φ update from the $\top$ update, while the other agents (the members of $Ag - B$) cannot. The actual action is s_0. Call this model G^B_φ.



A difference with the CC case is that group messages are S5 models. Since updates of S5 models with S5 models are S5, group messages engender common knowledge (as opposed to mere common belief). Let us work out the program transformation that this update engenders for common knowledge among some group of agents D.

We will have to work out $K^{G^B_\varphi}_{002} D$, $K^{G^B_\varphi}_{012} D$, $K^{G^B_\varphi}_{112} D$, $K^{G^B_\varphi}_{102} D$.

For these, we need $K^{G^B_\varphi}_{001} D$, $K^{G^B_\varphi}_{011} D$, $K^{G^B_\varphi}_{111} D$, $K^{G^B_\varphi}_{101} D$.

For these in turn, we need $K^{G^B_\varphi}_{000} D$, $K^{G^B_\varphi}_{010} D$, $K^{G^B_\varphi}_{110} D$, $K^{G^B_\varphi}_{100} D$.

For these, we need:

$$\begin{aligned} T^{G^B_\varphi}_{00} D &= \bigcup_{d \in D} (?\varphi \ ; \ d) = ?\varphi \ ; \ D, \\ T^{G^B_\varphi}_{01} D &= \bigcup_{d \in D - B} (?\varphi \ ; \ d) = ?\varphi \ ; \ (D - B), \\ T^{G^B_\varphi}_{11} D &= D, \\ T^{G^B_\varphi}_{10} D &= D - B. \end{aligned}$$

It follows that:

$$\begin{aligned}
K_{000}^{G_\varphi^B} D &= ?\top \cup (?\varphi \ ; \ D), \\
K_{010}^{G_\varphi^B} D &= ?\varphi \ ; \ (D - B), \\
K_{110}^{G_\varphi^B} D &= ?\top \cup D, \\
K_{100}^{G_\varphi^B} D &= D - B.
\end{aligned}$$

From this we can work out the K_{ij1}, as follows:

$$\begin{aligned}
K_{001}^{G_\varphi^B} D &= (?\varphi \ ; \ D)^*, \\
K_{011}^{G_\varphi^B} D &= (?\varphi \ ; \ D)^* \ ; ?\varphi \ ; \ D - B, \\
K_{111}^{G_\varphi^B} D &= ?\top \cup D \cup (D - B; (?\varphi \ ; \ D)^*; ?\varphi \ ; \ D - B), \\
K_{101}^{G_\varphi^B} D &= D - B \ ; \ (?\varphi; D)^*.
\end{aligned}$$

Finally, we get K_{002} and K_{012} from this:

$$\begin{aligned}
K_{002}^{G_\varphi^B} D &= K_{001}^{G_\varphi^B} D \cup K_{011}^{G_\varphi^B} D \ ; \ (K_{111}^{G_\varphi^B} D)^* \ ; \ K_{101}^{G_\varphi^B} D \\
&= (?\varphi \ ; \ D)^* \cup \\
&\quad (?\varphi \ ; \ D)^* \ ; ?\varphi \ ; \ D - B; \\
&\qquad (D \cup (D - B; (?\varphi \ ; \ D)^* \ ; ?\varphi \ ; \ D - B))^* \ ; \ D - B \ ; \ (?\varphi \ ; \ D)^*, \\
K_{012}^{G_\varphi^B} D &= K_{011}^{G_\varphi^B} D \ ; \ (K_{111}^{G_\varphi^B} D)^* \\
&= (?\varphi \ ; \ D)^* \ ; ?\varphi \ ; \ D - B \ ; \ (D \cup (D - B \ ; \ (?\varphi; D)^* \ ; ?\varphi \ ; \ D - B))^*.
\end{aligned}$$

Abbreviating $D \cup (D - B \ ; \ (?\varphi \ ; \ D)^* \ ; ?\varphi \ ; \ D - B)$ as π, we get the following transformation for common knowledge among D after a group message to B that φ:

$$\begin{aligned}
&[G_\varphi^B, s_0][D^*]\psi \\
&\leftrightarrow \\
&[(?\varphi \ ; \ D)^* \cup ((?\varphi \ ; \ D)^* \ ; ?\varphi \ ; \ D - B \ ; \ \pi^* \ ; \ D - B \ ; \ (?\varphi \ ; \ D)^*)][G_\varphi^B, s_0]\psi \\
&\wedge \\
&[(?\varphi \ ; \ D)^* \ ; ?\varphi \ ; \ D - B \ ; \ \pi^*][G_\varphi^B, s_1]\psi.
\end{aligned}$$

This equivalence gives a precise characterisation of two path requirements that have to hold in the original model in order for common knowledge among D to result from the group message to B. The formula may look complicated, but mechanical verification of the requirement is quite easy.

5 Quantified Dynamic Logic

The second core system of dynamic logic that will be discussed in detail is that of *quantified dynamic logic* (QDL). QDL was developed by Harel [61] and Goldblatt [49]. Both monographs were inspired by Pratt [106]. Further information about the development of QDL can be found in [50, 62, 63].

Quantified dynamic logic can be viewed as the first order version of propositional dynamic logic. It is less abstract than PDL, for program atoms now get further analysed as assignments of values to program variables or as relational tests, and states take the concrete shape of mappings from program variables to appropriate values. At the background is a first order structure **M** consisting of a domain plus interpretations of relation and function symbols.

Recall that the assignment programs of WHILE looked like $v := t$, with v a program variable and t a term of the WHILE language. In QDL, the basic actions are:

- assigning a random value to a variable:

$$v :=?,$$

- assigning a definite value to a variable:

$$v := t,$$

- and testing for the truth of a formula:

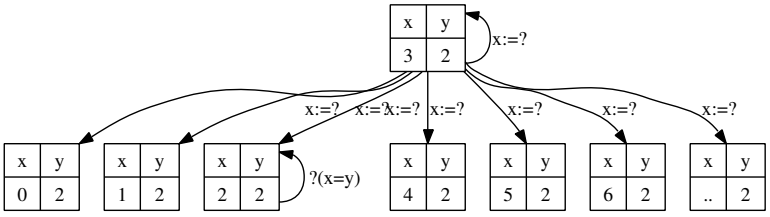
$$?\varphi.$$

Various versions of QDL result from imposing further restrictions on testing, e.g., by only allowing tests on boolean combinations of relational and equational atoms.

Consider a state where x has value 3 and y value 2. Assuming we are computing on the natural numbers, random assignment of a new value to x causes infinite branching to the states with

$$\{x \mapsto 0, y \mapsto 2\}, \{x \mapsto 1, y \mapsto 2\}, \{x \mapsto 2, y \mapsto 2\}, \{x \mapsto 3, y \mapsto 2\},$$

and so on. The subsequent test $x = y$ only succeeds for the state with $\{x \mapsto 2, y \mapsto 2\}$. The nett effect of $x :=? \ ; \ ?(x = y)$ is a transition from $\{x \mapsto 3, y \mapsto 2\}$ to $\{x \mapsto 2, y \mapsto 2\}$.



5.1 Language

Take a signature for first order logic. Define terms, formulae and programs, as follows:

$$\begin{array}{rcl} t & ::= & v \mid ft_1 \cdots t_n \\ \varphi & ::= & \top \mid Rt_1 \cdots t_n \mid t_1 = t_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists v\varphi \mid \langle\pi\rangle\varphi \\ \pi & ::= & v :=? \mid v := t \mid ?\varphi \mid \pi_1 \ ; \ \pi_2 \mid \pi_1 \cup \pi_2 \mid \pi^* \end{array}$$

Abbreviations are as in the case of PDL. In particular, the *SKIP*, *ABORT*, *WHILE*, *REPEAT*, *IF-THEN-ELSE* constructs are also defined as in the case of PDL. What Quantified Dynamic Logic gives us is a fleshed out version of PDL, with assignments (random and definite) and tests as basic actions. The assignments change relational structures, and therefore the appropriate assertion language is built from first order predicate logic rather than propositional logic, as in PDL.

Floyd-Hoare correctness statements for WHILE programs can be expressed directly in QDL. Recall the example of the correctness statement for the factorial program from section 2.4:

$$x! = Z \rightarrow [y := 1 \ ; \ \text{WHILE } x \neq 1 \text{ DO } (y := y * x \ ; \ x := x\dot{-}1)]y = Z.$$

This expresses partial correctness of the factorial program. Total correctness of the factorial program can be expressed in QDL as the conjunction of the above with the following:

$$\langle y := 1 \ ; \ \text{WHILE } x \neq 1 \text{ DO } (y := y * x \ ; \ x := x\dot{-}1)\rangle\top.$$

5.2 Semantics

A first order signature is a pair $(\mathbf{f}, \mathbf{R})$ where $\mathbf{f}$ is a list of function symbols with their arities and $\mathbf{R}$ is a list of relation symbols with their arities. Nullary function symbols are individual constants, nullary relation symbols are propositional constants, unary relation symbols are predicates.

A model for a signature $(\mathbf{f}, \mathbf{R})$ is a structure of the form

$$\mathbf{M} = (E^{\mathbf{M}}, f^{\mathbf{M}}, \ldots, R^{\mathbf{M}}, \ldots),$$

where E is a non-empty set, the $f^{\mathbf{M}}$ are interpretations in E for the members of $\mathbf{f}$ (i.e., if f is an n-ary function symbol, then $f^{\mathbf{M}} : E^n \rightarrow E$), and the $R^{\mathbf{M}}$ are interpretations in E for the members of $\mathbf{R}$ (i.e., if R is an n-ary relation symbol, then $R^{\mathbf{M}} \subseteq E^n$).

Let V be the set of variables of the language. As usual $g \sim_v h$ expresses that state h differs at most from state g on v. Interpretation of terms in $\mathbf{M}$ is defined

relative to a variable assignment $g : V \to E^{\mathbf{M}}$, as follows:

$$\begin{aligned} [\![v]\!]^{\mathbf{M}}_g &= g(v) \\ [\![ft_1 \cdots t_n]\!]^{\mathbf{M}}_g &= f^{\mathbf{M}}([\![t_1]\!]^{\mathbf{M}}_g, \ldots, [\![t_n]\!]^{\mathbf{M}}_g) \end{aligned}$$

Truth in $\mathbf{M}$ for formulae and relational meaning in $\mathbf{M}$ for programs are defined by simultaneous recursion:

$$\begin{aligned} \mathbf{M} &\models_g \top && \text{always} \\ \mathbf{M} &\models_g Rt_1 \cdots t_n && \text{iff } ([\![t_1]\!]^{\mathbf{M}}_g, \ldots, [\![t_n]\!]^{\mathbf{M}}_g) \in R^{\mathbf{M}} \\ \mathbf{M} &\models t_1 = t_2 && \text{iff } [\![t_1]\!]^{\mathbf{M}}_g \text{ is the same as } [\![t_2]\!]^{\mathbf{M}}_g \\ \mathbf{M} &\models_g \neg\varphi && \text{iff not } \mathbf{M} \models_g \varphi \\ \mathbf{M} &\models_g \varphi_1 \vee \varphi_2 && \text{iff } \mathbf{M} \models_g \varphi_1 \text{ or } \mathbf{M} \models_g \varphi_2 \\ \mathbf{M} &\models_g \exists v\varphi && \text{iff for some } h \text{ with } g \sim_v h, \mathbf{M} \models_h \varphi \\ \mathbf{M} &\models_g \langle\pi\rangle\varphi && \text{iff for some } h \text{ with } {}_g[\![\pi]\!]^{\mathbf{M}}_h, \mathbf{M} \models_h \varphi \end{aligned}$$

$$\begin{aligned} {}_g[\![v :=?]\!]^{\mathbf{M}}_h &&& \text{iff} && g \sim_v h \\ {}_g[\![v := t]\!]^{\mathbf{M}}_h &&& \text{iff} && h \text{ equals } g[v \mapsto [\![t]\!]^{\mathbf{M}}_g] \\ {}_g[\![?\varphi]\!]^{\mathbf{M}}_h &&& \text{iff} && g = h \text{ and } \mathbf{M} \models_g \varphi \\ {}_g[\![\pi_1 \ ; \ \pi_2]\!]^{\mathbf{M}}_h &&& \text{iff} && \text{there is an assignment } f \text{ with} \\ &&& && {}_g[\![\pi_1]\!]^{\mathbf{M}}_f \text{ and } {}_f[\![\pi_2]\!]^{\mathbf{M}}_h \\ {}_g[\![\pi_1 \cup \pi_2]\!]^{\mathbf{M}}_h &&& \text{iff} && {}_g[\![\pi_1]\!]^{\mathbf{M}}_h \text{ or } {}_g[\![\pi_2]\!]^{\mathbf{M}}_h \\ {}_g[\![\pi^*]\!]^{\mathbf{M}}_h &&& \text{iff} && (g, h) \in ([\![\pi]\!]^{\mathbf{M}})^* \end{aligned}$$

Validity of QDL formulae over a given signature is defined in terms of truth in all models for the signature. A QDL formula φ over a given signature is satisfiable if there is model $\mathbf{M}$ for that signature together with a variable assignment g in the domain of that model, such that $\mathbf{M} \models_g \varphi$.

Note that the presence of $v :=?$ does not increase the expressive power of the language. Indeed, we have the following validities:

$$\begin{aligned} \exists v\varphi &\leftrightarrow \langle v :=?\rangle\varphi \\ \forall v\varphi &\leftrightarrow [v :=?]\varphi \end{aligned}$$

Next, if v does not occur in t, definite assignment of t to v is equivalent to random assignment to v followed by a test of the equality $v = t$. In other words, if v does not occur in t we have the following validities:

$$\begin{aligned} \langle v := t\rangle\varphi &\leftrightarrow \langle v :=? \ ; \ ?v = t\rangle\varphi \\ [v := t]\varphi &\leftrightarrow [v :=? \ ; \ ?v = t]\varphi. \end{aligned}$$

5.2.1 Substitution and Assignment

The computational process of assigning a value to a variable is intimately linked to the syntactic process of making a substitution of a term for a variable.

Recall the situation in first order logic. There, the basic truth definition is phrased in terms of a first order model $\mathbf{M}$, a variable assignment g, and a formula φ: $\mathbf{M} \models_g \varphi$ means that variable assignment g makes φ true in $\mathbf{M}$. Let t_s^v be the result of replacing variable v everywhere in term t by term s. Then the following term substitution lemma holds for FOL and for QDL:

Lemma 15 (Term substitution) $[\![t_s^v]\!]_g^{\mathbf{M}} = [\![t]\!]^{\mathbf{M}}_{g[v \mapsto [\![s]\!]_g^{\mathbf{M}}]}$.

This is easily proved with induction on the structure of t.

Using this, one can prove the substitution lemma for FOL. Recall that a term t is substitutable for v in φ (or: free for v in φ) if the substitution process does not cause accidental capture of variables in t. Use φ_t^v for the result of substituting t for all free occurrences of v in φ. The following holds for FOL:

Lemma 16 (Substitution) *If t is free for v in φ then*

$$\mathbf{M} \models_g \varphi_t^v \text{ iff } \mathbf{M} \models_{g[v \mapsto [\![t]\!]_g^{\mathbf{M}}]} \varphi.$$

The proof uses induction on the structure of φ, using the term substitution lemma for the atomic case. In the case of QDL, we can rephrase this as follows:

Lemma 17 (Assignment)

$$\mathbf{M} \models_g [v := t]\varphi \text{ iff } \mathbf{M} \models_{g[v \mapsto [\![t]\!]_g^{\mathbf{M}}]} \varphi.$$

What this means is that in QDL we can replace syntactic substitutions φ_t^v by $[v := t]\varphi$.

Below we will be interested in the subsystem of QDL defined by

$$\pi \quad ::= \quad ?Rt_1 \cdots t_2 \,|?t_1 = t_2 \mid v :=? \mid \sim\pi \mid \pi_1 \;;\; \pi_2.$$

where $\sim\pi$ is an abbreviation of $?[\pi]\bot$.

It turns out that this subsystem, baptised DPL in [55], has the same expressive power as first order logic, but its quantifier $v :=?$ has different binding behaviour from the quantifiers of first order logic. [55] proposes to employ the dynamic binding behaviour of the DPL quantifiers for analysing anaphoric linking (establishing the links between pronouns and their antecedents) in natural language.

5.2.2 Expressiveness

We can immediately see that the expressive power of QDL is greater than that of FOL. The following formula in the language of natural number arithmetic expresses induction on the natural numbers:

$$\forall y \langle x := 0 \ ; \ \text{WHILE } x \neq y \text{ DO } x := x + 1 \rangle \top. \tag{7}$$

This asserts that for all y the program $x := 0 \ ; \ \text{WHILE } x \neq y \text{ DO } x := x + 1$ has a terminating execution. That is, every y can be reached by starting from 0 and repeatedly applying the successor function. This defines the natural numbers up to isomorphism, and no first order formula can do that. Let $\varphi_{\mathbb{N}}$ be the conjunction of formula (7) with the Peano axioms for arithmetic except the induction axiom. Then the valid QDL sentences of the form $\varphi_{\mathbb{N}} \to \psi$, with ψ a first order sentence, specify the first order sentences ψ that are true on $\mathbb{N}$. But we know from Gödel's incompleteness theorem and Church's Thesis that this set of sentences cannot be effectively enumerated.

5.3 Interpreted versus Uninterpreted Reasoning

As was the case with the WHILE-language and other systems, we are often interested in computation with respect to some standard structure, such as the natural numbers. In such cases, we evaluate QDL formulae and programs in this structure, and talk, e.g., about $\mathbb{N}$-validity: truth for all variable assignments in $\mathbb{N}$, and so on.

Note that all WHILE programs over a given signature are QDL programs over that same signature. Thus, we can use QDL for making assertions about the behaviour of WHILE programs. When interpreting with respect to $\mathbb{N}$, we can specify Euclid's GCD algorithm as the following QDL program:

$$\pi = \ \text{WHILE } x \neq y \text{ DO IF } x > y \text{ THEN } x := x - y \text{ ELSE } y := y - x.$$

Clearly, Floyd-Hoare correctness statements about WHILE programs can be expressed in QDL. E.g., the following QDL statements about the GCD program, expressing the total correctness of the program, are valid in $\mathbb{N}$:

$$(x = x' \wedge y = y' \wedge x \times y > 0) \to [\pi] \ x = \gcd(x', y').$$

$$x \times y > 0 \to \langle \pi \rangle \top$$

The first of these says that if program π over $\mathbb{N}$ terminates then in the output state x holds the value of the GCD of x' and y'. The second of these expresses that the program does indeed terminate for all states with $x \times y > 0$, for $\langle \pi \rangle \top$ expresses termination for all deterministic programs.

In the case of uninterpreted reasoning we are interested in truth in all structures. The following is valid in all models:

$$(x = x' \wedge y = y') \to [z := x \ ; \ x := y \ ; \ y := z](x = y' \wedge y = x').$$

5.4 Undecidability and Completeness

QDL is a proper extension of classical FOL, and, as we have seen, its validity problem is not effectively enumerable. This means that there can be no proof theory for QDL based on an enumerable set of axioms and an enumerable set of decidable inference rules. A proof theory will have to be based on infinitary (hence undecidable) inference rules.

The following axioms relate random assignment to quantification and definite assignment to substitution:

$$
\begin{array}{ll}
\forall v\varphi \leftrightarrow [v :=?]\varphi & \\
\forall v\varphi \rightarrow [v := t]\varphi & \\
\forall w[v := w]\varphi \rightarrow \forall v\varphi & w \notin \{v\} \cup var(\varphi) \\
\forall v\varphi \rightarrow [v := t]\forall v\varphi & \\
\forall w[v := t]\varphi \rightarrow [v := t]\forall w\varphi & w \notin \{v\} \cup var(t) \\
\langle v := t\rangle\varphi \leftrightarrow [v := t]\varphi & \\
[v := t]Rt_1 \cdots t_n \leftrightarrow Rt_1{}^v_t \cdots t_n{}^v_t & \\
[v := t]t_1 = t_2 \leftrightarrow t_1{}^v_t = t_2{}^v_t & \\
[v := t][v := s]\varphi \rightarrow [v := s^v_t]\varphi & \\
[v := t][w := s]\varphi \rightarrow [w := s^v_t][v := t]\varphi & w \notin \{v\} \cup var(t) \\
s = t \rightarrow ([v := t]\varphi \leftrightarrow [v := s]\varphi) &
\end{array}
$$

Now take as axiom schemes the following:

- All instances of valid FOL formulae,
- all instances of valid PDL formulae,
- the assignment axiom schemes above,

and as rules of inference:

- modus ponens
- quantifier generalisation

$$\frac{\varphi}{\forall v\varphi}$$

- program generalisation

$$\frac{\varphi}{[\pi]\varphi}$$

- and infinitary convergence:

$$\frac{\varphi \rightarrow [\pi^n]\psi, \quad n \in \mathbb{N}}{\varphi \rightarrow [\pi^*]\psi}$$

where π^n is given by $\pi^0 =?\top$, $\pi^{n+1} = \pi \ ; \ \pi^n$

A proof in this calculus may have infinitely many premises. This infinitary proof system is sound and complete (Harel [62] or Goldblatt [50]):

Theorem 18 *For any QDL formula φ,*

$$\models \varphi \textit{ iff } \vdash \varphi.$$

6 DPL as a fragment of QDL

In the introduction we mentioned that dynamic logic is also used in linguistics, in particular in the analysis of various phenomena involving information flow in discourse (text, conversation). In this section we turn to the study of a particular formalism, that of Dynamic Predicate Logic (DPL), that has played a prominent role in the development of dynamic semantic theories for natural language.

The DPL system is a representative instance of a whole variety of systems that have been developed in formal semantics of natural language to deal with dynamic aspects of meaning and information flow: the contribution of declaratives to the 'common ground', presuppositional phenomena, anaphoric links across sentence boundaries, the temporal structure of discourse, the semantic effects of imperatives, and so on. DPL is an illustrative example in the present context because of its obvious affinities with systems developed in other areas, in particular with PDL and QDL. The formal properties of the DPL system have been studied quite extensively (cf., e.g., [15] and the references below). Also, DPL provides a nice illustration of some of the central concepts of QDL. A more eleborate discussion of the specific linguistic issues involved can be found in section 7.

6.1 System

DPL is the subsystem of QDL that is given by the following syntax:

Definition 19 (DPL syntax)

$$\begin{array}{rcl} t & ::= & v \mid c \mid f t_1 \cdots t_n \\ \pi & ::= & ?Rt_1 \cdots t_2 \mid ?t_1 = t_2 \mid v :=? \mid \sim\pi \mid \pi_1 \ ; \ \pi_2. \end{array}$$

Semantics: as in the definition of QDL. The meaning of $\sim\pi$ is given by:

$${}_g[\![\sim\pi]\!]_h \quad \text{iff} \quad g \text{ equals } h \text{ and for no } g' \text{ it holds that } {}_g[\![\pi]\!]^{\mathbf{M}}_{g'}.$$

As was noted earlier, $\sim\pi$ can be taken as an abbreviation of $?[\pi]\bot$.

FOL can be interpreted in DPL, as follows:

$$\begin{array}{rcl} (Rt_1 \cdots t_n)^\bullet & = & ?Rt_1 \cdots t_n \\ (t_1 = t_2)^\bullet & = & ?t_1 = t_2 \\ (\neg\varphi)^\bullet & = & \sim\varphi^\bullet \\ (\varphi_1 \vee \varphi_2)^\bullet & = & \sim(\sim\varphi_1^\bullet \ ; \ \sim\varphi_2^\bullet) \\ (\exists v\varphi)^\bullet & = & \sim\sim(v :=? \ ; \ \varphi^\bullet) \end{array}$$

6.1.1 DPL and FOL

An inspection of the DPL semantics yields:

Lemma 20 (Embedding) *For all FOL formulae φ, all models $\mathbf{M}$ for the signature of φ, all assignments g, h in $\mathbf{M}$:*

$$\mathbf{M} \models_g \varphi \text{ and } g = h \quad \text{iff} \quad {}_g[\![\varphi^\bullet]\!]^{\mathbf{M}}_h.$$

DPL programs can be reversed, as follows:

$$\begin{array}{rcl} (?Rt_1 \cdots t_n)^\smile & = & ?Rt_1 \cdots t_n \\ (?t_1 = t_2)^\smile & = & ?t_1 = t_2 \\ (v :=?)^\smile & = & v :=? \\ (\sim\pi)^\smile & = & \sim\pi \\ (\pi_1 \ ; \ \pi_2)^\smile & = & {\pi_2}^\smile \ ; \ {\pi_1}^\smile \end{array}$$

This definition shows that $^\smile$ is definable in DPL, because $?Rt_1 \cdots t_n$, $?t_1 = t_2$, $\sim\pi$, $v :=?$ and $\sim\pi$ are all symmetric and hence self-converse. What this means is that adding a converse operator to DPL does not increase expressive power. The following reversal lemma is proved by induction on DPL program structure:

Lemma 21 (Reversal) *For all DPL programs π, all models $\mathbf{M}$, all assignments g, h in $\mathbf{M}$:*

$${}_g[\![\pi]\!]^{\mathbf{M}}_h \text{ iff } {}_h[\![\pi^\smile]\!]^{\mathbf{M}}_g.$$

6.1.2 DPL and DPL′

One of the features of DPL is that it does not have the distinction between programs (interpreted as binary relations on a set of appropriate valuations) and formulae (interpreted as predicates on a set of appropriate valuations). Still, it is sometimes useful to be able to make statements about DPL programs. For this, we define DPL′ formulae as follows (π ranges over DPL programs):

$$\varphi \quad ::= \quad \top \mid Rt_1 \cdots t_n \mid t_1 = t_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists v\varphi \mid \langle\pi\rangle\varphi.$$

Statements about DPL programs can now be made in DPL′. The formula $\langle\pi\rangle\top$ characterises the assignments where π succeeds. In [55] this is called the satisfaction set of π. The set of possible output assignments for π (the production set of π) is characterised by $\langle\pi^\smile\rangle\top$. The following formula expresses that π_1 and π_2 have the same satisfaction and production sets:

$$(\langle\pi_1\rangle\top \leftrightarrow \langle\pi_2\rangle\top) \quad \wedge \quad (\langle{\pi_1}^\smile\rangle\top \leftrightarrow \langle{\pi_2}^\smile\rangle\top). \tag{8}$$

Note that it does not follow from (8) that π_1 and π_2 are equivalent. Let π_1 be $?x = x$ and let π_2 be $x :=?$. Then $\langle \pi_1 \rangle \top \leftrightarrow \top \leftrightarrow \langle \pi_1 \rangle \top$ and $\langle \pi_1{}^\smile \rangle \top \leftrightarrow \top \leftrightarrow \langle \pi_1{}^\smile \rangle \top$, but the two programs are not equivalent. The interpretation of π_1 is the identity relation on the set of assignments, that of π_2 is the set of all pairs g, h such that $g \sim_x h$.

6.2 Proof theory

6.2.1 Reduction to FOL

There are various proof systems for DPL or closely related logics. An early example is the Floyd-Hoare-type system of Van Eijck and De Vries [39]. Basically, this calculus uses Floyd-Hoare rules to reduce DPL to FOL. We can also use QDL to reduce DPL to FOL. Here is a translation function from DPL′ to FOL:

$$\begin{aligned}
(\top)^\circ &= \top \\
(Rt_1 \cdots t_n)^\circ &= Rt_1 \cdots t_n \\
(t_1 = t_2)^\circ &= t_1 = t_2 \\
(\neg\varphi)^\circ &= \neg\varphi^\circ \\
(\varphi_1 \vee \varphi_2)^\circ &= \varphi_1^\circ \vee \varphi_2^\circ \\
(\exists v\varphi)^\circ &= \exists v\varphi^\circ \\
(\langle ?Rt_1 \cdots t_n \rangle\varphi)^\circ &= Rt_1 \cdots t_n \wedge \varphi^\circ \\
(\langle ?t_1 = t_2 \rangle\varphi)^\circ &= t_1 = t_2 \wedge \varphi^\circ \\
(\langle v :=? \rangle\varphi)^\circ &= \exists v\varphi^\circ \\
(\langle \sim\pi \rangle\varphi)^\circ &= \neg(\langle \pi \rangle \top)^\circ \wedge \varphi^\circ \\
(\langle \pi_1 \ ; \ \pi_2 \rangle\varphi)^\circ &= (\langle \pi_1 \rangle\langle \pi_2 \rangle\varphi)^\circ.
\end{aligned}$$

Direct inspection of the semantics reveals that this translation is correct, in the following sense:

Lemma 22 (Translation Correctness) *For all DPL′ formulae φ, all FO models* **M** *for the signature of φ, all variable assignments g in* **M***:*

$$\mathbf{M} \models_g \varphi \textit{ iff } \mathbf{M} \models_g (\varphi)^\circ.$$

It follows from this that the following reduction axioms for DPL are sound:

test relation	$\langle ?Rt_1 \cdots t_n \rangle\varphi \leftrightarrow Rt_1 \cdots t_n \wedge \varphi$
test equality	$\langle ?t_1 = t_2 \rangle\varphi \leftrightarrow t_1 = t_2 \wedge \varphi$
random assignment	$\langle v :=? \rangle\varphi \leftrightarrow \exists v\varphi$
dynamic negation	$\langle \sim\pi \rangle\varphi \leftrightarrow \neg\langle \pi \rangle \top \wedge \varphi$
sequence	$\langle \pi_1 \ ; \ \pi_2 \rangle\varphi \leftrightarrow \langle \pi_1 \rangle\langle \pi_2 \rangle\varphi.$

The boxed counterparts of these axioms can be derived by propositional reasoning:

test relation	$[?Rt_1 \cdots t_n]\varphi \leftrightarrow (Rt_1 \cdots t_n \rightarrow \varphi)$
test equality	$[?t_1 = t_2]\varphi \leftrightarrow (t_1 = t_2 \rightarrow \varphi)$
random assignment	$[v :=?]\varphi \leftrightarrow \forall v\varphi$
dynamic negation	$[\sim\pi]\varphi \leftrightarrow ([\pi]\bot \rightarrow \varphi)$
sequence	$[\pi_1 \ ; \ \pi_2]\varphi \leftrightarrow [\pi_1][\pi_2]\varphi.$

The calculus for DPL′ can now consist of the axioms for FOL, the axioms for test relation, test equality, dynamic negation and sequence (either in their box or in their diamond versions), and the inference rules of FOL: modus ponens and generalisation. It follows from the translation lemma that this axiomatisation is sound. The axiomatisation is also complete.

Theorem 23 (DPL′ completeness)
For all DPL′ formulae φ: if $\models \varphi$ then $\vdash \varphi$.

Proof. The proof system for FOL is complete, and every DPL′ formula φ is provably equivalent to some FOL formula. □

By way of example of the application of the calculus we give the derivation of the FOL counterpart to the DPL rendering of so-called 'donkey sentences' (cf., section 7.2.1 below for more extensive discussion of this type of phenomenon):

1 *If a farmer owns a donkey then he beats it.*

DPL translates this using a defined operator for dynamic implication, given by:

$$\varphi \Rightarrow \psi \quad :\equiv \quad \sim(\varphi \ ; \ \sim\psi).$$

The DPL rendering of (1) looks like this:

$$(x :=? \ ; \ ?Fx \ ; \ y :=? \ ; ?Dy \ ; \ ?Oxy) \Rightarrow ?Bxy.$$

Here is the reduction to FOL using the reduction axioms:

$$\begin{aligned}
& \langle\sim(x :=? \ ; \ ?Fx \ ; \ y :=? \ ; \ ?Dy \ ; \ ?Oxy \ ; \ \sim?Bxy\rangle\top \\
\leftrightarrow \ & \langle\sim(x :=? \ ; \ ?Fx \ ; \ y :=? \ ; \ ?Dy \ ; \ ?Oxy\rangle\langle\sim?Bxy\rangle\top \\
\leftrightarrow \ & [x :=? \ ; \ ?Fx \ ; \ y :=? \ ; \ ?Dy \ ; \ ?Oxy \ ; \ \sim?Bxy]\bot \\
\leftrightarrow \ & \forall x[?Fx \ ; \ y :=? \ ; \ ?Dy \ ; \ ?Oxy \ ; \ \sim?Bxy]\bot \\
\leftrightarrow \ & \forall x(Fx \rightarrow [y :=? \ ; \ ?Dy \ ; \ ?Oxy \ ; \ \sim?Bxy]\bot) \\
\leftrightarrow \ & \forall x(Fx \rightarrow \forall y[?Dy \ ; ?Oxy \ ; \ \sim?Bxy]\bot) \\
\leftrightarrow \ & \forall x(Fx \rightarrow \forall y([?Dy \ ; \ ?Oxy \ ; \ \sim?Bxy]\bot)) \\
\leftrightarrow \ & \forall x(Fx \rightarrow \forall y(Dy \rightarrow [?Oxy \ ; \ \sim?Bxy]\bot)) \\
\leftrightarrow \ & \forall x(Fx \rightarrow \forall y(Dy \rightarrow (Oxy \rightarrow [\sim?Bxy]\bot))) \\
\leftrightarrow \ & \forall x(Fx \rightarrow \forall y(Dy \rightarrow (Oxy \rightarrow ([?Bxy]\bot \rightarrow \bot)))) \\
\leftrightarrow \ & \forall x(Fx \rightarrow \forall y(Dy \rightarrow (Oxy \rightarrow Bxy))).
\end{aligned}$$

Clearly, this is the desired universal reading of the example.

6.2.2 Axiomatisation

Axiomatising DPL becomes more of a challenge if one is after an axiomatisation at the level of programs, without recourse to a static assertion language like FOL. Such a direct axiomatisation is provided in Van Eijck [34]. Key element of the calculus is an appropriate treatment of substitution in DPL.

For readability, it is useful to slightly rephrase the DPL language, by leaving out the spurious test operators and by using quantifier notation for random assignment:

Definition 24 (DPL syntax again)

$$\pi \quad ::= \quad \top \mid Rt_1 \cdots t_2 \mid t_1 = t_2 \mid \exists v \mid \sim\pi \mid \pi_1 \ ; \ \pi_2.$$

Types of Variable Occurrences Let V be the variables of the DPL language. The set of variables which have a fixed occurrence in a DPL program π is given by a function *free* : DPL $\to \mathcal{P}V$, the set of variables which are introduced in a formula is given by a function *intro* : DPL $\to \mathcal{P}V$, and the set of variables which have a classically bound occurrence in a formula is given by a function *cbnd* : DPL $\to \mathcal{P}V$.

The introduced variables of π (called 'blocked' variables in [133]) are the variables y such that π contains an $\exists y$ not in the scope of a negation. The free variables of π are the variables on which input valuations have to agree on output valuations. The classically bound variables of π are the variables that behave like the bound variables of FOL. Let $var(Pt_1 \cdots t_n)$ be the set of all variables among $t_1 \cdots t_n$.

Definition 25 (free, intro, cbnd)

- $free(\top) := \emptyset, intro(\top) := \emptyset, cbnd(\top) := \emptyset.$
- $free(\exists v \ ; \ \pi) := free(\pi) - \{v\},$
  $intro(\exists v \ ; \ \pi) := \{v\} \cup intro(\pi),$
  $cbnd(\exists v \ ; \ \pi) := cbnd(\pi).$
- $free(Pt_1 \cdots t_n \ ; \ \pi) := var(Pt_1 \cdots t_n) \cup free(\pi),$
  $intro(Pt_1 \cdots t_n \ ; \ \pi) := intro(\pi),$
  $cbnd(Pt_1 \cdots t_n \ ; \ \pi) := cbnd(\pi).$
- $free(\sim(\pi_1) \ ; \ \pi_2) := free(\pi_1) \cup free(\pi_2),$
  $intro(\sim(\pi_1) \ ; \ \pi_2) := intro(\pi_2),$
  $cbnd(\sim(\pi_1) \ ; \ \pi_2) := intro(\pi_1) \cup cbnd(\pi_1) \cup cbnd(\pi_2).$
- $free((\pi_1\pi_2) \ ; \ \pi_3) := free(\pi_1 \ ; \ (\pi_2 \ ; \ \pi_3)),$
  $intro((\pi_1\pi_2) \ ; \ \pi_3) := intro(\pi_1 \ ; \ (\pi_2 \ ; \ \pi_3)),$
  $cbnd((\pi_1\pi_2) \ ; \ \pi_3) := cbnd(\pi_1 \ ; \ (\pi_2 \ ; \ \pi_3)).$

Some examples may clarify this definition. Let

$$\pi := \exists v \ ; \ \exists w \ ; \ \ Ruvw.$$

Then $\mathit{intro}(\pi) = \{v, w\}$, $\mathit{free}(\pi) = \{u\}$, $\mathit{cbnd}(\pi) = \emptyset$. The occurrence of u in $Ruvw$ is free.

Variables introduced within the scope of negation become classically bound. Let

$$\pi := \ \sim(\exists v \ ; \ \exists w \ ; \ \ Ruvw).$$

Then $\mathit{intro}(\pi) = \emptyset$, $\mathit{free}(\pi) = \{u\}$, $\mathit{cbnd}(\pi) = \{v, w\}$. The occurrence of u in $Ruvw$ is still free.

A variable can have fixed, bound and introduced occurrences in an expression. Let

$$\pi := Px \ ; \ \ \exists x \ ; \ \ \sim Px \ ; \ \ \sim(\exists x \ ; \ Qx).$$

Then $\mathit{intro}(\pi) = \{x\}$, $\mathit{free}(\pi) = \{x\}$, $\mathit{cbnd}(\pi) = \{x\}$. The leftmost occurrence of x is free, the other occurrences are not.

Binding Note that for all DPL programs π, $\mathit{intro}(\pi) \cap \mathit{free}(\pi) = \emptyset$. Let $g \sim_X h$ if variable assignments g and h differ at most in the values of variables among X. Let $g[X]h$ if $g \sim_{V-X} h$, where V is the set of all variables. Thus, $g[X]h$ expresses that g and h agree on the values of variables in X.

Lemma 26 (DPL binding) *If* ${}_g[\![\varphi]\!]^{\mathbf{M}}_h$ *then* $g \sim_{\mathit{intro}(\varphi)} h$ *and* $g[\mathit{free}(\varphi)]h$.

Thus, the leftmost occurrence of x in $Px \ ; \ \exists x \ ; \ \sim Px \ ; \ \sim(\exists x \ ; \ Qx)$ is free, the other occurrences are not. Use π^v_t for the result of substituting t for all free occurrences of v in π:

Definition 27 (π^v_t)

$$\begin{array}{lcl}
\top^v_t & := & \top \\
(Rt_1 \cdots t_n \ ; \ \pi)^v_t & := & Rt_1{}^v_t \cdots t_n{}^v_t \ ; \ \pi^v_t \\
(t_1 = t_2 \ ; \ \pi)^v_t & := & t_1{}^v_t = t_n{}^v_t \ ; \ \pi^v_t \\
(\exists v \ ; \ \pi)^v_t & := & \exists v \ ; \ \pi \\
(\exists w \ ; \ \pi)^v_t & := & \exists w \ ; \ \pi^v_t \\
(\sim(\pi_1) \ ; \ \pi_2)^v_t & := & \sim(\pi_1{}^v_t) \ ; \ \pi_2{}^v_t \\
((\pi_1 \ ; \ \pi_2) \ ; \ \pi_3)^v_t & := & (\pi_1 \ ; \ (\pi_2 \ ; \ \pi_3))^v_t
\end{array}$$

Note that this definition of substitution takes the dynamic binding force of $\exists v$ over the text that follows into account (cf. the clause for $(\exists v \ ; \ \pi)^v_t$, where the occurrence of $\exists v$ blocks off the π that follows). Visser [133] calls this substitution 'left' substitution.

Figure 2: The Calculus for DPL

test axiom	$\dfrac{}{T \Longrightarrow T}$
transitivity	$\dfrac{\varphi \Longrightarrow \psi \qquad \psi \Longrightarrow \chi}{\varphi \Longrightarrow \chi}\ \mathit{intro}(\psi) \cap \mathit{free}(\chi) = \emptyset$
test swap	$\dfrac{C_1 T_1 \ ; \ T_2 C_2 \Longrightarrow \varphi}{C_1 T_2 \ ; \ T_1 C_2 \Longrightarrow \varphi}$
quantifier move	$\dfrac{C_1 T \ ; \ \exists v\ C_2 \Longrightarrow \varphi}{C_1 \exists v \ ; \ T C_2 \Longrightarrow \varphi}\ v \notin \mathit{free}(T) \qquad \dfrac{C_1 \exists v \ ; \ T C_2 \Longrightarrow \varphi}{C_1 T \ ; \ \exists v\ C_2 \Longrightarrow \varphi}\ v \notin \mathit{free}(T)$
quantifier intro	$\dfrac{\varphi \Longrightarrow \psi^v_t}{\varphi \Longrightarrow \exists v \ ; \ \psi}\ t$ free for v in ψ
var refreshment	$\dfrac{C_1 \exists v\ C_2 \Longrightarrow \varphi}{C_1 \exists w (C_2 \Longrightarrow \varphi)^v_w}\ w \notin \mathit{intro}(C_1) \cup \mathit{free}(C_1)$
sequencing	$\dfrac{\psi \Longrightarrow \chi}{\varphi \ ; \ \psi \Longrightarrow \chi} \qquad \dfrac{\varphi \Longrightarrow \psi \qquad \varphi \Longrightarrow \chi}{\varphi \Longrightarrow \psi \ ; \ \chi}\ \mathit{intro}(\psi) \cap \mathit{free}(\chi) = \emptyset$
negation	$\dfrac{\varphi \Longrightarrow \psi}{\varphi \ ; \ \sim\psi \Longrightarrow \bot} \qquad \dfrac{\varphi \ ; \ \psi \Longrightarrow \bot}{\varphi \Longrightarrow \sim\psi}$
double negation	$\dfrac{\varphi \Longrightarrow \sim\sim\psi}{\varphi \Longrightarrow \psi} \qquad \dfrac{\varphi \ ; \ \sim\sim\psi \Longrightarrow \bot}{\varphi \ ; \ \psi \Longrightarrow \bot}$

Sequent Deduction Rules Figure 6.2.2 gives a set of sequent deduction rules for DPL, using the format $\varphi \Longrightarrow \psi$, where $\Longrightarrow$ is the sequent separator. Note that $\varphi \Longrightarrow \bot$ expresses that φ is inconsistent. The calculus defines a relation $\Longrightarrow \subseteq \mathrm{DPL}^2$ by means of: $\varphi \Longrightarrow \psi$ iff $\varphi \Longrightarrow \psi$ is at the root of a finite tree with sequents at its nodes, such that the sequents at a leaf node are axioms of the calculus, and the sequents at the internal nodes follow by means of a rule of the calculus from the sequent(s) at the daughter node(s) of that internal node.

In the calculus, C, with and without subscripts, is used as a variable over contexts, where a context is a formula or the empty list ϵ. Substitution and evaluation are extended to contexts in the obvious way. If C is a context and φ a formula, then we use $C\varphi$ for the formula given by: $C\varphi := \varphi$ if $C = \epsilon$, $C\varphi := \psi; \varphi$ if $C = \psi$. Similarly for φC, and for $C_1 \varphi C_2$.

It is convenient to extend the definition of substitution to sequents.

Definition 28 ($(C \Longrightarrow \varphi)^v_t$) *Induction on the structure of* C

$$(\epsilon \Longrightarrow \varphi)^v_t \quad := \quad \epsilon \Longrightarrow \varphi^v_t$$

$$(\psi \Longrightarrow \varphi)^v_t \quad := \quad \begin{cases} \psi^v_t \Longrightarrow \varphi & \text{if } v \in \mathit{intro}(\psi) \\ \psi^v_t \Longrightarrow \varphi^v_t & \text{otherwise.} \end{cases}$$

Substitution for sequents carries in its wake a notion of being free for a variable in a sequent:

Definition 29 (t **is free for** v **in** $C \Longrightarrow \psi$)

1. *t is free for v in $\epsilon \Longrightarrow \psi$ if t is free for v in ψ.*
2. *t is free for v in $\varphi \Longrightarrow \psi$ if t is free for v in φ, and either $v \in \mathit{intro}(\varphi)$ or t is free for v in ψ.*

When a rule mentions a substitution φ^v_t in the consequent of a sequent then the standard assumption is made hat t is free for v in φ. When a rule mentions a substitution $C_1(C_2 \Longrightarrow \varphi)^v_t$ then it is assumed that t is free for v in $C_2 \Longrightarrow \varphi$.

In the rules of Figure 6.2.2 T is used as an abbreviation of formulae φ with $\mathit{intro}(\varphi) = \emptyset$ (T for *Test* formula).

Here is an example application of the quantifier intro rule.

$$\frac{Rxx \Longrightarrow Rxx}{Rxx \Longrightarrow \exists y \ ; \ Rxy}$$

Rxx equals $(Rxy)^y_x$, so this is indeed a correct application of the rule.

Variable refreshment allows the liberation of a captured variable, e.g., of the first two occurrences of x in $\exists x \ ; \ Px \ ; \ \exists x \ ; \ Qx$, by means of replacement by a variable that does not occur as an introduced or free variable in the left context in the given sequent:

$$\frac{\exists x \ ; \ Px \ ; \ \exists x \ ; \ Qx \Longrightarrow Qx}{\exists y \ ; \ Py \ ; \ \exists x \ ; \ Qx \Longrightarrow Qx}$$

It is also possible to change the other occurrences of x in the same example. The following is also a correct application of the rule:

$$\frac{\exists x \ ; \ Px \ ; \ \exists x \ ; \ Qx \Longrightarrow Qx}{\exists x \ ; \ Px \ ; \ \exists y \ ; \ Qy \Longrightarrow Qy}$$

Note that the rule can also be used to recycle a variable:

$$\frac{\exists y \ ; \ Py \ ; \ \exists x \ ; \ Qx \Longrightarrow Qx}{\exists x \ ; \ Px \ ; \ \exists x \ ; \ Qx \Longrightarrow Qx}$$

This application is also correct, for

$$(\exists x \ ; \ Px \ ; \ \exists x \ ; \ Qx \Longrightarrow Qx) = (\exists x \ ; \ (Py \ ; \ \exists x \ ; \ Qx \Longrightarrow Qx)^y_x).$$

An example application of the rule for ; right is:

$$\frac{Rxx \Longrightarrow \exists y \ ; \ Ryx \qquad\qquad Rxx \Longrightarrow \exists z \ ; \ Rxz}{Rxx \Longrightarrow \exists y \ ; \ Ryx \ ; \ \exists z \ ; \ Rxz} \ ; \ \text{right}$$

In case the condition on the rule for ; right is not satisfied, e.g. for the two sequents $\sim Px \ ; \ \exists x \ ; \ Px \Longrightarrow \exists x \ ; \ \sim Px$ and $\sim Px \ ; \ \exists x \ ; \ Px \Longrightarrow Px$, this can always be remedied by one or more applications of $\exists$ Right to the second premise.

It is not hard to see that the rules of the calculus are sound. The calculus is also complete. For the proof — a modification of the standard Henkin style completeness proof for classical first order logic — we refer to [34].

6.3 Computational DPL

In [3] a computational interpretation of standard first order logic is proposed, with as key ingredient a new interpretation of identity statements (in suitable contexts) as assignment actions. Computation states are partial maps of variables to values. The gist of the proposal is this: in a state α that is defined for a term t but undefined for a variable v, an identity statement $v = t$ or $t = v$ is interpreted as an instruction to assign the value t^α to the variable v.

Let $\mathcal{M} = (M, I)$ be a FO model, and let V be a set of variables. Let $\mathbf{A} := \{\alpha \in M^X \mid X \subseteq V\}$. If $\alpha \in M^X$, then call X the domain of α; a term t is α-closed if all variables in t are in X, an atom $Pt_1 \cdots t_n$ is α-closed if all t_i are α-closed, and an identity $t_1 = t_2$ is α-closed if both of t_1, t_2 are. Use $\uparrow$ for 'undefined' and $\downarrow$ for 'defined'. Term interpretation in model $\mathbf{M} = (M, I)$ with respect to valuation α has now to take the possibility into account that the value of the

term under α is undefined.

$$
\begin{array}{lcl}
v^\alpha & := & \begin{cases} \alpha(v) & \text{if } v \text{ is } \alpha\text{-closed} \\ \uparrow & \text{otherwise} \end{cases} \\
(ft_1 \cdots t_n)^\alpha & := & \begin{cases} I(f)t_1^\alpha \cdots t_n^\alpha & \text{if } t_1, \ldots, t_n \ \alpha\text{-closed} \\ \uparrow & \text{otherwise} \end{cases}
\end{array}
$$

An identity $t_1 = t_2$ is an α-assignment if either $t_1 \equiv v$, $t_1^\alpha = \uparrow$, $t_2^\alpha = \downarrow$, or $t_2 \equiv v$, $t_1^\alpha = \downarrow$, $t_2^\alpha = \uparrow$. An α-assignment can be used as a statement that extends a valuation α with a new value.

A first order predicate with its arguments $Pt_1 \cdots t_n$ is interpreted as a test that can either fail or succeed, provided that all of the t_i are defined for the input state α; otherwise an error is generated. The empty conjunction is interpreted as the instruction to succeed in any state α, with output α.

This is then extended to finite conjunctions of implications, negations, disjunctions and existential quantifications, according to the following rule set: $[\![\varphi]\!]_\alpha$ denotes the computation tree for φ on input α. A tree is successful if it contains at least one leaf consisting of just a variable map, failed if all its leafs equal *fail*.

$$
\begin{array}{c}
\exists v\varphi \wedge \psi, \alpha \\
| \\
[\![\varphi \wedge \psi]\!]_\alpha
\end{array}
\quad \text{if } v \notin \mathrm{dom}\,(a),\ v \text{ not free in } \psi.
$$

$$
\begin{array}{c}
\neg\varphi \wedge \psi, \alpha \\
| \\
[\![\psi]\!]_\alpha
\end{array}
\quad \text{if } \varphi \ \alpha\text{-closed}, [\![\varphi]\!]_\alpha \text{ failed.}
$$

$$
\begin{array}{c}
\neg\varphi \wedge \psi, \alpha \\
| \\
\mathit{fail}
\end{array}
\quad \text{if } \varphi \ \alpha\text{-closed}, [\![\varphi]\!]_\alpha \text{ successful.}
$$

$$
\begin{array}{c}
(\varphi_1 \to \varphi_2) \wedge \psi, \alpha \\
| \\
[\![\psi]\!]_\alpha
\end{array}
\quad \text{if } \varphi_1 \ \alpha\text{-closed}, [\![\varphi_1]\!]_\alpha \text{ failed.}
$$

$$
\begin{array}{c}
(\varphi_1 \to \varphi_2) \wedge \psi, \alpha \\
| \\
[\![\varphi_2 \wedge \psi]\!]_\alpha
\end{array}
\quad \text{if } \varphi_1 \ \alpha\text{-closed}, [\![\varphi_1]\!]_\alpha \text{ successful.}
$$

$$
\begin{array}{ccc}
 & (\varphi_1 \vee \varphi_2) \wedge \psi, \alpha & \\
 \swarrow & & \searrow \\
[\![\varphi_1 \wedge \psi]\!]_\alpha & & [\![\varphi_2 \wedge \psi]\!]_\alpha
\end{array}
$$

All cases not listed generate an error.

This computation procedure has the property that for any φ and any input valuation α, the valuations at success nodes in $[\![\varphi]\!]_\alpha$ are extensions of α. Computations never *change* the input valuations. In particular, $\exists x\varphi \wedge \psi$ is treated as

equivalent with $\varphi \wedge \psi$ provided the variable conditions hold. Thus, the quantifier has no computational effect, but acts as a prohibition sign: its only function is to rule out occurrences of x in the outside context of $\exists x\varphi$.

The computational engine can be adapted to a setting where quantifiers are read dynamically, by giving assignments $v :=?$ an appropriate computational meaning. The relational interpretation of $v :=?$ is computationally infeasible, for the instruction to replace the value of register v by an arbitrary new value is awkward if one is computing over an infinite domain, say the domain of natural numbers. As a statement on $\mathbb{N}$, $v :=?$ is an instruction to pick an arbitrary natural number and assign it to v. Since this can be done in an infinity of ways, this does not represent any finite computational procedure.

In the computational interpretation of DPL one therefore changes the quantifier action as follows. Instead of letting the quantifier action $v :=?$ perform its full duty, the action $v :=?$ is split into two tasks:

1. throwing away the old value of v, and
2. identifying appropriate new values for v.

On infinite domains any attempt to perform task (2) immediately will cause an infinite branching transition, and therefore this task is *postponed*. The duty of finding an appropriate new value for v is relegated to an appropriate *identifying statement* for v further on. This move is inspired by the computational interpretation of identity statements from [3]. See [40] and [64] for more information on computing with DPL.

6.4 Extensions of DPL

DPL can be viewed as the most basic of a hierarchy of formulae-as-programs languages. We will now look at extensions of DPL with the six operations $\cup$, $\breve{}$, σ, $\breve{\sigma}$, $\cap$, $\exists\!\!\!-$. Extensions of DPL with $\cap$ (relation intersection) and $\exists\!\!\!-$ (local variable declaration) are studied in [133], while in [40], an extension of DPL with $\cup$ (relation union) and σ (simultaneous substitution) is axiomatised, and ω-completeness is proved for the extension of DPL with $\cup$, σ and Kleene star.

6.4.1 Extended Semantics

A substitution is a finite set of bindings $x \mapsto t$, with the usual conditions that no binding is trivial (of the form $x \mapsto x$) and that every x in the set has at most one binding (substitutions are functional). Examples of substitutions are $\{x \mapsto f(x)\}$ ("set new x equal to f-value of old x"), $\{x \mapsto y, y \mapsto x\}$ ("swap values of x and y"). If a substitution contains just a single binding we omit the curly brackets and write just the *assignment statement* $x := t$. Note that if x occurs in t, the assignment $x := t$ is not expressible in DPL. Similarly , there is no DPL program that is equivalent to $\{x \mapsto y, y \mapsto x\}$.

Left-to-right substitutions σ have right-to-left counterparts $\breve{\sigma}$ (converse substitutions). For pre- and postcondition reasoning with extensions of DPL, converse substitution and relation converse $\breve{}$ are attractive.

A converse substitution is a finite set of converse bindings $(x \mapsto t)\breve{}$, with the same conditions as those for substitutions. An example is $(x \mapsto f(x))\breve{}$ ("set old x equal to f-value of new x", or "look at all inputs g that differ from the output h only in x, and that satisfy $f(g(x)) = h(x)$").

The semantics definition for the new operators runs:

$$\begin{aligned}
[\![\sigma]\!]^{\mathbf{M}} &= \{(g, g^{x_1 \cdots x_n}_{d_1 \cdots d_n}) \mid \{x_1, \ldots, x_n\} = \mathrm{dom}(\sigma) \text{ and } d_i = \sigma(x_i)^{\mathbf{M},g} \\
[\![\breve{\sigma}]\!]^{\mathbf{M}} &= \{(g^{x_1 \cdots x_n}_{d_1 \cdots d_n}, g) \mid \{x_1, \ldots, x_n\} = \mathrm{dom}(\sigma) \text{ and } d_i = \sigma(x_i)^{\mathbf{M},g} \\
[\![\exists x(\pi)]\!]^{\mathbf{M}} &= \{(g, k^x_{g(x)}) \mid \text{for some } d : (g^x_d, k) \in [\![\pi]\!]^{\mathbf{M}}\} \\
[\![\pi_1 \cap \pi_2]\!]^{\mathbf{M}} &= [\![\pi_1]\!]^{\mathbf{M}} \cap [\![\pi_2]\!]^{\mathbf{M}} \\
[\![\pi\breve{}]\!]^{\mathbf{M}} &= \{(g, h) \mid (h, g) \in [\![\pi]\!]^{\mathbf{M}}\}
\end{aligned}$$

The $\exists$ operator allows for the declaration of local variables. Simultaneous substitution permits performing certain computations without the use of auxiliary variables. Converse and converse simultaneous substitution are useful for pre- and postcondition reasoning, as they allow us to define the inverses of programs under certain conditions [52, Chapter 21].

6.4.2 Left-to-Right and Right-to-Left Substitution

Because the semantics of DPL programs is completely symmetric, performing a substitution in a DPL program can be done in two directions: left-to-right and right-to-left [133] (see also [131], where substitutions for DPL with a stack semantics are studied). Left-to-right substitutions affect the left-free variable occurrences, right-to-left substitutions the right-free (or 'actively bound') variable occurrences.

DPL has two directional analogues to the substitution lemma from FOL: one for left-to-right substitution and one for right-to-left substitution. For left-to-right substitution we get that $g[\![\sigma(\pi)]\!]^{\mathbf{M}}h$ iff $g_\sigma[\![\pi]\!]^{\mathbf{M}}h$. Viewing the substitution itself as a state change, we can decompose this into $g[\![\sigma]\!]^{\mathbf{M}}g'[\![\pi]\!]^{\mathbf{M}}h$. This uses $g[\![\sigma]\!]^{\mathbf{M}}k$ iff $k = g_\sigma$.

The right-to-left substitution lemma for DPL says that $g[\![\breve{\sigma}(\pi)]\!]^{\mathbf{M}}h$ iff $g[\![\pi]\!]^{\mathbf{M}}h_\sigma$. Viewing the right-to-left substitution itself as a state change, we can decompose this into $g[\![\pi]\!]^{\mathbf{M}}h'[\![\breve{\sigma}]\!]^{\mathbf{M}}h$. This uses $k[\![\breve{\sigma}]\!]^{\mathbf{M}}h$ iff $k = h_\sigma$. Again, since in general $\breve{\sigma}$ is not expressible in DPL, we have a motivation to extend the language with converse substitutions.

Use $\circ$ for relational composition of substitution expressions, defined as follows:

Definition 30 (Composition of substitutions) *Let*

$$\sigma = \{v_1 \mapsto t_1, \ldots, v_n \mapsto t_n\} \text{ and } \rho = \{w_1 \mapsto r_1, \ldots, w_m \mapsto r_m\}$$

be substitutions. Then $\sigma \circ \rho$ is the result of removing from the set

$$\{w_1 \mapsto \sigma(r_1), \ldots, w_m \mapsto \sigma(r_m), v_1 \mapsto t_1, \ldots, v_n \mapsto t_n\}$$

the bindings $w_1 \mapsto \sigma(r_i)$ for which $\sigma(r_i) = w_i$, and the bindings $v_j \mapsto t_j$ for which $v_j \in \{w_1, \ldots, w_m\}$.

It is easily proved now that $\sigma \ ; \ \rho$ is equivalent to $\sigma \circ \rho$. E.g., $x := x+1 \ ; \ y := x$ is equivalent to $\{x \mapsto x+1, y \mapsto x+1\}$, and $x := y \ ; \ x := x+1$ is equivalent to $x := y + 1$.

Every DPL$(\cup, \sigma)$ formula can be written with $;$ associating to the right, as a list of predicates, quantifications, negations, choices and substitutions, with a substitution ρ at the end (possibly the empty substitution). Left-to-right substitution in DPL$(\cup, \sigma)$ is defined by:

$$\begin{array}{rcl}
\sigma(\rho) & := & \sigma \circ \rho \\
\sigma(\rho \ ; \ \pi) & := & \sigma \circ \rho \ ; \ \pi \\
\sigma(\exists v \ ; \ \pi) & := & \exists v \ ; \ \sigma'\pi \text{ where } \sigma' = \sigma \backslash \{v \mapsto t \mid t \in T\} \\
\sigma(P\bar{t} \ ; \ \pi) & := & P\sigma\bar{t}; \sigma\pi \\
\sigma(t_1 = t_2 \ ; \ \pi) & := & \sigma t_1 = \sigma t_2 \ ; \ \sigma\pi \\
\sigma(\sim(\pi_1) \ ; \ \pi_2) & := & \neg(\sigma\pi_1) \ ; \ \sigma\pi_2 \\
\sigma((\pi_1 \cup \pi_2); \pi_3) & := & \sigma(\pi_1; \pi_3) \cup \sigma(\pi_2; \pi_3)
\end{array}$$

A term t is left-to-right free for v in π if all variables in t are input-constrained in all positions of the left-free occurrences of v in π. A substitution σ is safe for π if all bindings $v \mapsto t$ of σ are such that t is left-to-right free for v in π. This allows us to prove:

Lemma 31 (Left-to-Right Substitution) *If σ is safe for π then $g[\![\sigma(\pi)]\!]h$ iff $g_\sigma[\![\pi]\!]h$.*

Right-to-left substitution is defined in a symmetric fashion, now reading the formulae in a left-associative manner, with a converse substitution at the front, and overloading the notation by also using $\circ$ for the relational composition of

converse substitutions (defined as one would expect, to get $\breve{\sigma} \circ \breve{\rho} = (\rho \circ \sigma)^{\smile}$):

$$\begin{array}{rcl}
\breve{\sigma}(\breve{\rho}) & := & \breve{\sigma} \circ \breve{\rho} \\
\breve{\sigma}(\pi \ ; \ \breve{\rho}) & := & \pi \ ; \ \breve{\sigma} \circ \breve{\rho} \\
\breve{\sigma}(\pi; \exists v) & := & \breve{\sigma}'\pi \ ; \ \exists v \text{ where } \breve{\sigma}' = \breve{\sigma} \backslash \{(v \mapsto t)^{\smile} \mid t \in T\} \\
\breve{\sigma}(\pi \ ; \ P\bar{t}) & := & \breve{\sigma}\pi \ ; \ P\sigma\bar{t} \\
\breve{\sigma}(\pi \ ; \ t_1 = t_2) & := & \breve{\sigma}\pi \ ; \ \sigma t_1 = \sigma t_2 \\
\breve{\sigma}(\pi_1 \ ; \ \sim(\pi_2)) & := & \breve{\sigma}\pi_1 \ ; \ \sim(\breve{\sigma}\pi_2) \\
\breve{\sigma}(\pi_1 \ ; \ (\pi_2 \cup \pi_3)) & := & \breve{\sigma}(\pi_1; \pi_2) \cup \breve{\sigma}(\pi_1; \pi_3)
\end{array}$$

A term t is right-to-left free for v in π if all variables in t are output-constrained in all positions of the right-free (actively bound) occurrences of v in π. A converse substitution $\breve{\sigma}$ is safe for π if all converse bindings $(v \mapsto t)^{\smile}$ of $\breve{\sigma}$ are such that t is right-to-left free for v in π. This allows us to prove:

Lemma 32 (Right-to-Left Substitution) *If $\breve{\sigma}$ is safe for π then $g[\![\breve{\sigma}(\pi)]\!]h$ iff $g[\![\pi]\!]h_\sigma$.*

6.4.3 Expressive Power

The following results are from [22]; many of the proofs are adapted from proofs given in [133].

Theorem 33 *$DPL(\exists)$ is equally expressive as $DPL(\cup, \cap, {}^{\smile}, \sigma, \breve{\sigma}, \exists)$.*

Proof. Let a formula π be given, and let V be the set of variables occurring in π. Furthermore, let V' and V'' be sets of variables, such that V, V' and V'' are mutually disjoint and of equal cardinality. Let $V = \{x_1, \ldots, x_n\}, V' = \{x'_1, \ldots, x'_n\}$, and $V'' = \{x''_1, \ldots x''_n\}$. The following function C translates a formula from $DPL(\cup, \cap, {}^{\smile}, \sigma, \breve{\sigma}, \exists)$ into a test from DPL.

$$\begin{array}{lcl}
C(\exists y) & = & \bigwedge_{x \in V \backslash \{y\}} x' = x \\
C(Rt_1 \ldots t_n) & = & \bigwedge_{x \in V} x' = x \ ; \ Rt_1 \ldots t_n \\
C(t_1 = t_2) & = & \bigwedge_{x \in V} x' = x \ ; \ t_1 = t_2 \\
C(\sim\pi) & = & \bigwedge_{x \in V} x' = x \ ; \ \sim(\exists x'_1; \ldots; \exists x'_n \ ; \ C(\pi)) \\
C(\pi_1; \pi_2) & = & \sim\sim(\exists x''_1; \ldots; \exists x''_n; C(\pi_1)^{[x'_1/x''_1, \ldots, x'_n/x''_n]} \ ; \ C(\pi_2)^{[x_1/x''_1, \ldots, x_n/x''_n]}) \\
C(\pi_1 \cap \pi_2) & = & C(\pi_1); C(\pi_2) \\
C(\pi_1 \cup \pi_2) & = & \sim(\sim C(\pi_1); \sim C(\pi_2)) \\
C(\pi^{\smile}) & = & C(\pi)^{[x_1/x'_1, \ldots, x_n/x'_n, x'_1/x_1, \ldots, x'_n/x_n]} \\
C(\sigma) & = & \bigwedge_{x \in dom(\sigma)} x' = \sigma(x); \bigwedge_{x \in V \backslash dom(\sigma)} x' = x \\
C(\breve{\sigma}) & = & \bigwedge_{x \in dom(\sigma)} x = \sigma(x)^{[x_1/x'_1, \ldots, x_n/x'_n]}; \bigwedge_{x \in V \backslash dom(\sigma)} x' = x \\
C(\exists x.\pi) & = & \sim\sim(\exists x; \exists x'; C(\pi)); x' = x
\end{array}$$

Here, $\bigwedge$ is used as a shorthand for a long composition, which is non-ambiguous because the order of the particular sentences involved doesn't matter. By induction, it can be shown that every π containing only variables in V, is equivalent to $\exists x'_1 \ldots x'_n(C(\pi); x_1 := x'_1; \ldots; x_n := x'_n)$. □

Theorem 34 *$DPL(^*,\exists)$ is equally expressive as $DPL(^*,\cup,\cap,^\smile,\sigma,\breve{\sigma},\exists)$*

Proof. As the proof of Theorem 33, now adding the following clause to the definition of C.

$$C(\pi^*) \;=\; \neg\neg(\exists x''_1 \;;\; \ldots \;;\; \exists x''_n; (C(\pi)^{[x'_i/x''_i]} \;;\; \textstyle\bigwedge_{x\in V} x := x'')^* \;;\; \bigwedge_{x\in V} x = x')$$

□

It follows immediately that every formula $\pi \in DPL(\cup,\cap,^\smile,\sigma,\breve{\sigma},\exists)$ is equivalent to a first order logic formula, in the sense that π can be executed in $\mathcal{M}$ with input assignment g iff the first order translation of π is true in $\mathcal{M}$ under g.

Theorem 35 (Visser) *$DPL(\exists)$ can be embedded into $DPL(\cap)$.*

Proof. Let π be of the form $\exists x(\psi)$, and let $\{y_1,\ldots,y_n\} = I(\pi)\backslash\{x\}$, where $I(\pi)$ are the introduced variables of π, i.e., the variables in *intro*(π), i.e., the variables y such that π contains an $\exists y$ not in the scope of a negation. Then π is equivalent to $(\exists x; \psi; \exists x) \cap (\exists y_1; \ldots; \exists y_n)$ □

In a similar way, the following can be proved:

Theorem 36 *$DPL(^*,\exists)$ can be embedded into $DPL(^*,\cap)$.*

It is also easy to show that * gets us beyond first order expressive power:

Theorem 37 *The formula*

$$\neg(\exists y \;;\; y = 0 \;;\; (\exists z \;;\; z = f(y) \;;\; \exists y \;;\; y = f(z))^* \;;\; x = y)$$

cannot be expressed in $DPL(\cup,\cap,^\smile,\sigma,\breve{\sigma},\exists)$.

Proof. On the natural numbers (interpreting f as the successor relation), this formula defines the odd numbers. Oddness on the natural numbers cannot be captured in a first order formula with only successor. □

Definition 38 *A substitution $\{x_1 \mapsto t_1,\ldots,x_n \mapsto t_n\}$ is* full *if every x_i occurs in some t_j and every t_i contains some x_j.*

Examples of full substitutions are $x := f(x)$ and $\{x \mapsto y, y \mapsto x\}$, while the substitution $x := y$ is not full. It is easy to see that full substitutions are closed under composition. Note that a substitution without function symbols is full iff

it is a renaming. Also, note that any formula of DPL(σ) or any of its extensions can be transformed into a formula in the same language containing only full substitutions, by replacing bindings of the form $x \mapsto t$, where t does not contain variables, by $\exists x \ ; \ x = t$.

Lemma 39 *Every formula $\pi \in DPL(\sigma)$ is equivalent to a formula of one of the following forms (for some ψ, x, χ, σ, where σ is full):*

1. $\neg\neg\chi; \sigma$.
2. $\psi; \exists x; \neg\neg\chi; \sigma$.

Proof. First rewrite π into a formula that contains only full substitutions. After that, the only non-trivial case in the translation instruction is the case of $\tau \ ; \ \psi$, where τ is full and ψ is of the first form, i.e., where ψ is equivalent to $\neg\neg\chi; \sigma$, for some χ, σ, with σ full. In this case, $\tau \ ; \ \psi$ is equivalent to $\neg\neg(\tau; \chi) \ ; \ \tau \circ \sigma$, where $\tau \circ \sigma$ is full because σ and τ are. □

Theorem 40 $(\exists x \cup \exists y)$ *cannot be expressed in $DPL(\sigma)$.*

Proof. Suppose $\pi \in DPL(\sigma)$ is equivalent to $(\exists x \cup \exists y)$. Take a model with as domain the natural numbers, and let R be the interpretation of π. By Lemma 39, it follows that π is equivalent to $\psi; \exists z; \neg\neg\chi; \sigma$, for some formulae ψ, χ, some variable z and some full substitution σ (otherwise, π would be deterministic). Two cases can be distinguished.

1. z does not occur in σ. Without loss of generality, assume that $z \neq x$. Take any pair of assignments g, h such that $g \neq h$ and $g \sim_x h$. Then gRh. Take any $k \neq h$ such that $k \sim_z h$. Then gRk, but g and k differ with respect to two variables (x and z), which is in contradiction with the fact that π is equivalent to $(\exists x \cup \exists y)$.
2. z occurs in σ. By the fact that there are no function symbols involved, and by the fact that σ is full, there must be exactly one binding in σ of the form $u \mapsto z$. We can apply the same argument as before, now using u instead of z, and again we arrive at a contradiction.

□

Every substitution is equivalent to a DPL formula containing only full substitutions, and since every full substitution without function symbols is a renaming, and therefore has a converse that is also a renaming, we get:

Lemma 41 *Every converse substitution containing no function symbols is equivalent to a formula in $DPL(\sigma)$.*

This immediately gives:

Theorem 42 $(\exists x \cup \exists y)$ *cannot be expressed in $DPL(\sigma, ^\smile)$.*

Lemma 43 *Every formula in $DPL(\sigma, \cup)$ is equivalent to a formula of the form $\pi_1 \cup \ldots \cup \pi_n$ ($n \geq 1$) where each $\pi_i \in DPL(\sigma)$.*

Theorem 44 *$(x \mapsto f(x))^{\smile}$ cannot be expressed in $DPL(\sigma, \cup)$.*

Proof. Suppose $\pi \in DPL(\sigma, \cup)$ is equivalent to $(x \mapsto f(x))^{\smile}$. By Lemma 43, we can assume that π is of the form $\pi_1 \cup \ldots \cup \pi_n$, where each $\pi_i \in DPL(\sigma)$. Consider the model with as domain $\{0, \ldots, n\}$, and where f is interpreted as the "successor modulo $n+1$" function.

Let us say that a relation R fixes a variable x if for $\forall gh \in cod(R)$: $g \sim_x h$ implies that $h = g$. Analysing each π_i, we can distinguish the following two cases.

- π_i is equivalent to $\neg\neg\chi; \sigma$, with σ full. Then $[\![\pi_i]\!]$ fixes x.
- π_i is equivalent to $\psi; \exists y; \neg\neg\chi; \sigma$, again with σ full. If y occurs in σ, then let z_i be the (unique) variable such that σ contains a binding of the form $z_i \mapsto f^k(y)$. If σ does not contain y then let $z_i = y$. Then it must be the case that $[\![\pi_i]\!]$ fixes z_i, for otherwise $[\![\pi_i]\!]$ is not injective.

Thus, we have that every π_i fixes some variable z_i. Let $\{z_1, \ldots, z_m\}$ be all variables that are fixed by some π_i (where $m \leq n$).

Consider all possible ways of assigning objects from the domain to the variables $z_1, \ldots z_m$ (assigning 0 to all other variables). This gives us $(n+1)^m$ assignments, each of which is in the co-domain of π. Now, of this space of assignments, each π_i can cover only a small part: at most $(n+1)^{m-1}$ (since one variable is fixed). So, together, $\pi_1, \ldots, \pi_n$ can cover at most $n*(n+1)^{m-1} = (n+1)^m - (n+1)^{m-1} < (n+1)^m$ assignments, which means that some assignments are not in the co-domain of π. This is in contradiction with the fact that π is equivalent to $(x \mapsto f(x))^{\smile}$. □

By symmetry, we get the following

Theorem 45 *$x \mapsto f(x)$ cannot be expressed in $DPL(\breve{\sigma}, \cup)$.*

Finally we have

Theorem 46 *$\exists y(y = x; \exists x \ ; \ Rxy)$ cannot be expressed in $DPL(\cup, \sigma, {}^{\smile})$.*

Proof. The same proof as for Theorem 44 can be used. Assume a signature without function symbols. Let the domain of the model be the set $\{0, \ldots, n\}$. Let R be interpreted as "successor modulo $n+1$". Then R is interpreted in the same way as f was in the proof of Theorem 44. Notice that, under this interpretation, $\exists y(y = x \ ; \ \exists x \ ; \ Rxy)$ means the same as $(x \mapsto f(x))^{\smile}$ did in the proof of Theorem 44. It follows that $\exists y(y = x; \exists x \ ; \ Rxy)$ cannot be expressed in $DPL(\cup, \sigma)$. Since the signature contains no function symbols, it follows by Lemma 41 that this formula cannot be expressed in $DPL(\cup, \sigma, {}^{\smile})$ either. □

6.4.4 DPL and Dynamic Relational Algebra

Yet another way in which the logic of DPL and sundry systems has been studied is by looking at the connection with dynamic relational algebra.

A dynamic relation algebra is an algebra for the signature $\{\bot, \sim, ;\}$, i.e., it consists of all binary relations on a set B (all members of $\mathcal{P}(B \times B)$), with $\bot$ interpreted as the empty relation , ; as relation composition, and $\sim$ as dynamic negation. A dynamic relation algebra is completely determined by its carrier set B.

Note that this is different from the usual relational algebra in the sense of [122], where the signature consists of the Boolean operations $\{-, \cap, \cup, \bot, \top\}$ and the order operations plus the identity relation $\{\circ, \breve{}, \mathrm{id}\}$. In fact, dynamic relation algebra can be viewed as a small non-Boolean fragment of relation algebra. Dynamic negation can be defined in ordinary relation algebra by means of:

$$\sim R := \mathrm{id} \cap -(R; \top)$$

Hollenberg [72] gives the following axiomatisation of dynamic relation algebra:

$$\begin{aligned}
\sim R; R &= \bot \quad \text{(falsum definition)} \\
R; \bot &= \bot \quad \text{(falsum right)} \\
\mathrm{id}; R &= R \quad \text{(identity left)} \\
R; (S; T) &= (R; S); T \quad \text{(associativity)} \\
\sim R; \sim S &= \sim S; \sim R \quad \text{(test permutation)} \\
R &= (\sim\sim R); R \quad \text{(domain test)} \\
\sim\sim(\sim R; \sim S) &= \sim R; \sim S \quad \text{(test composition)} \\
\sim(R; S); R &= (\sim(R; S)R); \sim S \quad \text{(modus ponens)} \\
\sim(R : (S \vee T)) &= \sim((R : S) \vee (R; T)) \quad \text{(distribution)},
\end{aligned}$$

where $R \vee S$ is an abbreviation of $\sim(\sim R; \sim S)$.

Note that $\sim R; R = \bot$ can be viewed as a definition of $\bot$. Order is important, for $R; \sim R$ does not always denote the empty relation.

Tests are subsets of the identity relation. $\sim R$ is always a test, and R is a test iff $\sim\sim R = R$, so $\sim\sim(\sim R; \sim S) = \sim R; \sim S$ expresses that the composition of two tests is again a test.

The fact that $\sim(R; S); R = (\sim(R; S)R); \sim S$ is called *modus ponens* is explained by defining $R \Rightarrow S$ as $\sim(R; \sim S)$ and substituting $\sim S$ for S. This gives:

$$(R \Rightarrow S); R = (R \Rightarrow S); R; \sim\sim S.$$

Hollenberg [72] has a proof that this axiomatisation is sound and complete for dynamic relation algebra. In [74] it is proved that in any model $(M, \bot, \sim, ;)$

of this axiom system, dynamic negation is fully determined by the underlying monoid $(M, ;)$.

In [16] it was shown that DPL-negation $\sim$ is the only permutation-invariant operator in dynamic relational algebra that satifies the following conditions:

$$\begin{aligned} \sim\bot &= \mathrm{id} \\ \sim(\cup_i R_i) &= \cup_i(\sim R_i) \\ \sim\sim R \cup (R;\top) &= R;\top \\ \sim R; R &= R. \end{aligned}$$

Permutation-invariant operators are operators O satisfying

$$\pi(O(R,S)) = O(\pi(R), \pi(S))$$

for every permutation π on the state set on which the relations are defined.

This result about DPL-negation led [16] to conjecture that DPL is complete for dynamic relational algebra, in the sense that counterexamples to relational identities in the vocabulary $\{\bot, \sim, ;\}$ are expressible in DPL. This conjecture was proved in [132].

Theorem 47 (Visser) *Schematic validity in DPL is complete for dynamic relational algebra.*

Proof. Suppose some relational equation E in the vocabulary $\{\bot, \sim, ;\}$ is refuted by a family of binary relations $\{R_a \mid a \in A\}$ over some carrier set B, where A is the set of atomic relation symbols occurring in the equation E.

We will consider DPL formulae over the variables x, y. Consider the space $B^{\{x,y\}}$ of all assignments in B to x and y.

DPL formulae in x, y denote relations between input and output assignments to $\{x, y\}$. For each R_a we define a new relation $\widehat{R}_a$ on $B^{\{x,y\}}$, by setting

$$\widehat{R}_a = \{(\{x \mapsto s_1, y \mapsto s_2\}, \{x \mapsto s_3, y \mapsto s_4\}) \mid R_a s_1 s_3\}.$$

The crucial insight is that the function $g \mapsto g(x)$ is a functional bisimulation (also known as: a p-morphism) from the transition system of the $\widehat{R}_a$ on $B^{\{x,y\}}$ to the transition system of the R_a on B, since $\sim$ and ; are safe for bisimulation.

Let the new relation symbol I denote identity in $(B, \{R_a \mid a \in A\})$. Then the relations $\widehat{R}_a$ can be defined in DPL by means of:

$$\exists y; R_a xy; \exists x; Ixy; \exists y.$$

If the relations at the lefthand and the righthand side of E are different, their originals under g are different too. Thus, an inequality defined in terms of $\sim$ and ; on $(B, \{R_a \mid a \in A\})$ corresponds to an inequality on

$$(B^{\{x,y\}}, \{\widehat{R}_a \mid a \in A\}).$$

This shows that the left- and righthand sides of the equation E yield a pair of non-equivalent DPL formulae. □

7 Dynamic logic and natural language semantics

7.1 Introduction

As we saw in Section 6 the difference between dynamic predicate logic (DPL) and quantified dynamic logic (QDL) is that whereas the latter makes a distinction, both in the syntax and in the semantics, between static formulae and dynamic programs, the former has basically only one kind of construct: programs. All formulae are programs, so there is no distinction either in syntactic category or in semantic type, between different kinds of linguistic constructions: all constructs are given a dynamic interpretation. The motivation for this is not a matter of expressive power, but one of 'ideology'. The difference can be characterised as follows: whereas QDL acknowledges two different notions of meaning: one descriptive and one imperative, DPL embodies a unified conception: all meanings are relations between states. By doing so, DPL instantiates a conception of meaning that has become prominent in natural language semantics from the early eighties onward and that sometimes is summarised in the slogan 'Meaning is context change potential'.

This view on meaning is often referred to as 'dynamic semantics'. Various people have contributed to it, motivated by various concerns. Broadly speaking we may discern two main trends. First of all there is work that focuses on epistemic and pragmatic issues, that arise in connection with presuppositions, the structure of information exchange, but also with conditionals and modal expressions. Very influential in this trend is the early work by Stalnaker on assertion and presuppositions [115, 116]. Other early work is that of Veltman [125]. A second influx of ideas derives from issues concerning semantics, in particular pronominal reference and quantification. This is exemplified by work of Heim [65, 66] and Kamp [79, 81]. Somewhat orthogonal to these two trends is the work on game-theoretical semantics for natural language explored by Hintikka and others [69]. Another approach that has clear affinities with a dynamic approach is that of situation semantics [10].

The variety of empirical subjects that prompted the use of dynamic concepts have resulted in an analogous variety of systems. Also, different authors entertain different views on how the use of these concepts affect the notion of meaning as it applies to natural language. Some maintain a truth conditional, propositional notion of meaning and relegate dynamics to the realm of language use, i.e., pragmatics, whereas others argue that the notion of meaning as such needs to be viewed as a dynamic concept. Yet others take a middle position and locate the dynamic aspects in the construction of representations that themselves have a static interpretation. Cf., [56, 80, 117] for discussion. In what follows we focus on those systems in which the use of dynamic concepts directly interacts with the concept of meaning that is modelled.

The general characteristic of dynamic systems is that formulae are interpreted as

entities that change the context. In natural language semantics and pragmatics, 'context' is an umbrella concept, that covers a wide variety of elements that are somehow tied to the use and the interpretation of expressions. Speaker and addressee, time and place, elements from preceding discourse, objects and properties introduced in conversation, information of speech participants about the world, themselves, each other, and so on, — all these factors may be involved in linguistic exchanges.

Within a particular system the relevant aspects of the context are represented in the system as states. Which aspects counts as relevant depends on the specific application and/or the expressive resources of the system. For example, in DPL states are simply assignments of values to variables, and this reflects that DPL is focused on those aspects of context that concern binding relationships between antecedents, i.e., quantified noun phrases and proper names, and anaphoric expressions, i.e., pronouns. When one extends or alters the scope of application, the notion of a state changes as well, resulting in a modification or extension of the original system. In this type of system states consist of objects and their properties and relationships and dynamic interpretation changes them by adding new objects, establishing new relationships, and so on.

As we noted, another important aspect of the context is the information of the speech participants. On a dynamic view the utterance of a sentence is to be regarded as an instruction to the speech participants to update their information with the content of the utterance. (Hence the name 'update semantics'.) A system modelling this will have states that represent the informational states of speech participants, e.g., as sets of propositions, sets of worlds, possibilities, or situations. Utterances then are interpreted as updates of such states. For example, a dynamic ('update') semantics for a conditional $\varphi \rightarrow \psi$ would (roughly) be defined as an operation that checks whether every update of a given set of possibilities with the antecedent satisfies satisfy the consequent.

Actually, these points of view are not incompatible. For example, we can look upon DRT- and DPL-like systems as concerned with information as well, viz., with information *about the discourse*: the entities that have been introduced, their properties and relationships, and the various possibilities that are available for anaphoric reference. Information in the update sense is then information *about the world*: information about the actual state of things as well as possibilities that are still open. As a matter of fact, combining these two perspectives is a more interesting exercise than just putting two orthogonal systems together: there are interesting interactions between the two.

In the remainder of this section we start with the use of dynamic logic in accounting for certain problems in semantics . Then we will turn to systems motivated by epistemic-pragmatics concerns. Finally, we will briefly look at combined systems.

7.2 Dynamic Semantics

7.2.1 Dynamic Phenomena

Discourse Representation Theory (DRT, [41, 79, 81]), File Change Semantics (FCS, [66]), dynamic predicate logic (DPL, [55]) are systems that originated in the late eighties, early nineties of the last century. Their initial motivation was linguistic. They grew out of attempts to deal with certain facts concerning anaphora and binding that had resisted adequate treatment in the Montague framework that dominated natural language semantics at the time. Other important areas of application are tense and aspect, presupposition, plurality. For more extensive discussion of the linguistic applications of these systems, cf., [25], [17], and the references given above. Here it suffices to give just a brief illustration of one example of the kind of phenomena these systems were intended to deal with: scope and binding. Basically, in this area there are two groups of problems: cross-sentential anaphoric relationships and so-called 'donkey'-constructions, which present a particular form of intra-sentential binding.

Cross-sentential anaphora refers to constructions such as:

A man entered the pub. He wore a black hat.

The pronoun 'He' in the second sentence is most naturally taken to refer back, i.e., as an anaphoric reference to, the referent of 'a man' in the first sentence. At the time there was a preference for dealing with anaphora – antecedents relationships in terms of variable binding: the antecedent 'a man' semantically operates as a quantifier, binding the variable that corresponds to the pronoun. The problem with this type of cross-sentential antecedent – anaphora relationships is, of course, that the binding can be established only when the discourse is finished. And even then, one must take care with such antecedents as 'One man', so as not to end up with the wrong interpretation ('One man φ. He ψ' is not the same as 'One man φ and ψ')

Donkey anaphora is connected with intra-sentential binding, e.g., between antecedent and consequent in conditional constructions:

If John spots a good investment opportunity, he grasps it.

The fact to be accounted for here is the binding of the anaphoric pronoun in the consequent by the indefinite noun phrase in the antecedent in such a way that the indefinite gets 'universal' force: the sentence is most naturally taken to express that John grasps every opportunity he sees. (Not all sentences with this structure have a universal (also called 'strong') reading: 'If I have a quarter, I'll put it in the parking meter' (Pelletier & Schubert). Cf., [82] for extensive discussion of so-called 'weak' and 'strong' readings of these kinds of constructions.)

Note that in each case the problem is not finding an adequate representation of the meanings of these sentences in (first) order logic. Rather, the problem

is coming up with such a representation while using the standard meanings of the expressions involved, and deriving the representation in an 'on line', i.e., incremental fashion, without delayed interpretation or after the fact re-analysis.

7.2.2 DPL again

Although it was not the first system to be developed, we focus on DPL because it is the most 'pure' instantiation of a dynamic view on meaning. It was developed because of a certain dissatisfaction with the representational, non-compositional nature of, e.g., DRT. It intends to do away with dynamically constructed representations as part of the semantics and wants to locate the dynamics purely in the meanings themselves.

The system The standard reference is [55], earlier similar views were developed in [9] and [118]. The original DPL-system stayed as close as possible to standard first order logic FOL: it employed the same language and only changed the semantics. In section 6 the system was given in a form that stayed close to that of QDL. What follows is the original formulation, i.e., with the syntax of FOL and an adapted semantics.

$$\begin{array}{rcl} t & ::= & v \mid c \\ \varphi & ::= & Rt_1 \ldots t_n \mid t_1 = t_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists v\varphi \end{array}$$

The other connectives and the universal quantifier can be defined in the usual fashion. (But note that compared to FOL the choice of base logical constants is limited.)

The semantics uses the same ingredients as that of FOL. A model M is a pair $\langle E, F\rangle$, where E is a non-empty set and $F^{\mathbf{M}}(c) \in E$ and $F^{\mathbf{M}}(R^n) \subseteq E^n$. States $g \in S$ are assignments $V \to E$. As usual $g \sim_v h$ denotes the state h that differs from g at most on v.

Interpretation of terms is given by: $[\![t]\!]^{\mathbf{M}}_g = g(t), F^{\mathbf{M}}(t)$ for variables and constants respectively. Formulae denote subsets of $S \times S$:

$$\begin{array}{rcl} {}_g[\![Rt_1 \ldots t_n]\!]^{\mathbf{M}}_h & \text{iff} & g = h \;\&\; \langle [\![t_1]\!]^{\mathbf{M}}_g \ldots [\![t_n]\!]^{\mathbf{M}}_g\rangle \in F^{\mathbf{M}}(R) \\ {}_g[\![t_i = t_j]\!]^{\mathbf{M}}_h & \text{iff} & g = h \;\&\; [\![t_i]\!]^{\mathbf{M}}_g = [\![t_j]\!]^{\mathbf{M}}_g \\ {}_g[\![\neg\varphi]\!]^{\mathbf{M}}_h & \text{iff} & g = h \;\&\; \text{there exists no } g' \colon {}_g[\![\varphi]\!]^{\mathbf{M}}_{g'} \\ {}_g[\![\varphi_1 \wedge \varphi_2]\!]^{\mathbf{M}}_h & \text{iff} & \text{there exists a } g' \colon {}_g[\![\varphi_1]\!]^{\mathbf{M}}_{g'} \;\&\; {}_{g'}[\![\varphi_2]\!]^{\mathbf{M}}_h \\ {}_g[\![\exists v\varphi]\!]^{\mathbf{M}}_h & \text{iff} & \text{there exists a } g' \colon g \sim_v g' \;\&\; g'[\![\varphi]\!]^{\mathbf{M}}_h \end{array}$$

Note that although all formulae denotes relations between states (assignments), only conjunction and existentially quantified formulae actually change states,

the others are tests. Conjunction is effectively re-interpreted as program composition, and an existential quantified formula has the cumulative effect of resetting the state with respect to the variable and feeding the result into the formulae. It is easy to see that

$$\text{for all } \mathbf{M}, g, h\colon {}_g[\![\exists x\varphi]\!]^{\mathbf{M}}_h \text{ iff } {}_g[\![x :=?\ ;\ \varphi]\!]^{\mathbf{M}}_h$$

The definitions of truth and validity as given in section 6 carry over, as do the notions of production set and satisfaction set. Equivalence as identity of interpretation transcends identity of input (satisfaction set) and output (production set). Cf. section 6 for an example in DPL′. $\neg(Px \wedge \neg Px)$ and $\exists x\neg(Px \wedge \neg Px)$ both have S as their satisfaction set and as their production set. But their meanings are different: the identity relation on S, and the set of all pairs g, h such that $g \sim_x h$, respectively. Note the meaning of a test *can* be completely characterised in terms of its satisfaction set and its production set and that all valid tests denote the identity relation on S.

Some characteristic examples The following two examples exhibit characteristic properties of the semantics of DPL. Both concern the extended binding force of the existential quantifier.

The first one concerns the interaction of the existential quantifier and conjunction. In $\exists xPx \wedge Qx$ the existential quantifier $\exists x$ randomly assigns a value to x that is passed on to Px, and tested. If it succeeds, conjunction, which is relational composition, passes it on to Qx, to be tested again. (We leave out reference to the model M whenever this does not lead to confusion.)

$$\begin{aligned} {}_g[\![\exists xPx \wedge Qx]\!]_h \quad &\text{iff} \quad \text{there exists a } g'\colon {}_g[\![\exists xPx]\!]_{g'} \ \&\ {}_{g'}[\![Qx]\!]_h \\ &\text{iff} \quad \text{there exists a } g'\colon g \sim_x g' \ \&\ g'(x) \in F(P) \ \&\ g'(x) \in F(Q) \end{aligned}$$

This allows DPL to deal with cross-sentential anaphora of the kind: ‘A man .... He ...’

Note that extended binding can also occur across other quantifiers, as e.g., in $\exists xPx \wedge \exists yRxy$, where the occurrence of x in Rxy is bound by $\exists x$; and across negation: in $\exists xPx \wedge \neg Qx$ the x in $\neg Qx$ is also bound by $\exists x$. Note that since we do not prohibit the same quantifier to occur more than once we have to be careful which occurrence of a quantifier binds a particular variable occurrence: in $\exists xPx \wedge Qx \wedge \exists xHx$ the occurrence of x in Hx is bound by the last occurrence of $\exists x$.

The second example of extended binding concerns the behaviour of the existential quantifier in conditional constructions. Consider the formula $\exists xPx \rightarrow Qx$, which is shorthand for $\neg(\exists xPx \wedge \neg Qx)$. Here we have an existential quantifier in the antecedent of a conditional and an occurrence of x in the consequent that in FOL would be free. However, if we compute its meaning, we see that the

second occurrence is bound by the existential quantifier, and, moreover, that the latter gets universal force:

$$
\begin{array}{lll}
{}_g[\![\exists x Px \rightarrow Qx]\!]_h & \text{iff} & {}_g[\![\neg(\exists x Px \wedge \neg Qx)]\!]_h \\
& \text{iff} & \text{there exists no } g' \colon {}_g[\![\exists x Px]\!]_{g'} \,\&_{g'}\, [\![\neg Qx]\!]_h \\
& \text{iff} & \text{for all } g' \colon \text{if } {}_g[\![\exists x Px]\!]_{g'} \text{ then } {}_{g'}[\![Qx]\!]_h
\end{array}
$$

So, every way of re-setting the value of x to one that satisfies P is one that satisfies Q.

Note that the extended binding force of the existential quantifier is blocked by negation: in $\neg\exists x Px \wedge Qx$ the occurrence of x in Qx is free. This is because the negation turns $\exists x Px$ into a test: the value assigned by $\exists x$ to x remains local to Px, and is not passed on to Qx. Thus in $\exists x Px \rightarrow Qx$ the binding of the existential quantifier in the antecedent extends to the consequent, but not beyond the formula as a whole.

Thus we can distinguish between formulae that are *internally dynamic*, i.e., in which an existential quantifier binds variables outside its scope, but only in the formula itself; and those that are *externally dynamic*, in which existential quantifiers have the power to bind variables in additional formulae that are added to its right. The latter are responsible for DPL's treatment of extra-sentential, i.,e., discourse binding; the former deal with internal binding from antecedent to consequent.

Other properties Other characteristic properties of the DPL-logic follow in a straightforward manner from the semantics. Double negation fails in view of negation blocking dynamic binding; conjunction and the existential quantifier can not be defined in terms of, e.g., negation, disjunction and the universal quantifier, because of the asymmetry of the respective expressions w.r.t. binding; conjunction is not unconditionally commutative and idempotent; the existential and universal quantifiers are not fully interdefinable; and finally, we can not take alphabetic variants of existentially quantified formulae.

As for entailment, neither inclusion of truth conditions, nor meaning inclusion, provide a suitable definition. The reason is that we want existential quantifiers in the premises of an argument to be able to bind variables in the conclusion, in view of the possibilities of antecedent – anaphora links in natural language reasoning: from 'A man came in carrying a stick' we want to be able to conclude 'So, he was carrying a stick'. So ψ follows from $\varphi_1 \ldots \varphi_n$ iff in all models every interpretation of the premises (in sequential order, of course) leads to a successful interpretation of the conclusion:

$\varphi_1, \ldots, \varphi_n \models \psi$ iff
for all $\mathbf{M}, g, h$: if ${}_g[\![\varphi_1 \wedge \cdots \wedge \varphi_n]\!]^{\mathbf{M}}_h$, then there exists an $h' \colon {}_h[\![\psi]\!]^{\mathbf{M}}_{h'}$

In terms of DPL′ (see section 6):

$$\varphi_1, \ldots, \varphi_n \models \psi \quad \text{iff} \quad \text{for all } \mathbf{M}\text{: } [\![[\varphi_1 \ ; \ \cdots \ ; \ \varphi_n]\langle\psi\rangle\top]\!]^{\mathbf{M}} \text{ equals the set of all assignments.}$$

It is easily checked that, e.g., $\exists x Px \models Px$, as required. Further we have:

$$\varphi_1, \ldots, \varphi_n \models \psi \quad \text{iff} \quad \models (\varphi_1 \wedge \cdots \wedge \varphi_n) \rightarrow \psi$$

Notice that if no binding occurs from premises to conclusion, the notion of entailment defined boils down to the truth-conditional one. It is easily checked that entailment is not reflexive and also not transitive.

DPL being a first order language, it differs from FOL in its non-standard binding behaviour. As we saw in section 6, FOL can be embedded in DPL in a straightforward way. Since DPL′ can be translated into FOL (cf., section 6), the same holds for DPL.

Context As was noted above, contexts in DPL are assignments of values to variables, satisfying certain descriptive conditions. What they represent are the individuals and their properties that have been introduced in a discourse (a text, a conversation), e.g., by proper names or descriptions, or by indefinite NPs. Other expressions, such as pronouns, may draw from this pool of available referents. In DPL this is accounted for via the use of (indexed) variables. Context-change is represented through operations on assignments, as, for example, by the existential quantifier, which 'resets' the context with regard to a particular variable. (Cf., the formulation of DPL in section 6, that brings this out more explicitly, by regarding the existential quantifier as a construct of its own.)

7.2.3 Discourse Representation Theory

Now we briefly introduce a very streamlined and basic version of Discourse Representation Theory (DRT). For an extensive introduction, the reader is referred to the standard [81]. The differences between DPL and DRT are quite like those between DPL and DPL′ or DPL and QDL: whereas DPL is a 'pure' language in which no distinction is made between programs and statements, DRT, like DPL′ and QDL, does make such a distinction, between what are called 'conditions' and what are called 'discourse representation structures' (DRSs). This syntactic distinction is reflected in the semantics, and is motivated by what Kamp in his seminal paper on DRT [79] claims is essential for a proper account of natural language meaning, viz., that it 'combines a definition of truth with a systematic account of semantic representations' (*op.cit.*, p.1). Thus, the dynamics in DRT takes place in the building of semantic representations.

The system The canonical format of DRT uses so-called box-notation (see below for some examples). In order to facilitate comparison, however, we recast the syntax and semantics of DRT in a linear format. The non-logical vocabulary consists of n-place predicates, individual constants, and variables. Logical constants are negation $\neg$, implication $\rightarrow$, and identity $=$.

DRT terms are constants and variables:

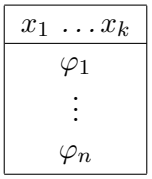
$$t \quad ::= \quad x \mid c$$

Conditions φ and DRSs Φ are defined as follows:

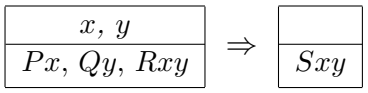
$$\begin{array}{lll} \varphi & ::= & Rt_1 \ldots t_n \mid t_1 = t_2 \mid \neg\Phi \mid \Phi_1 \Rightarrow \Phi_2 \\ \Phi & ::= & [x_1 \ldots x_k][\varphi_1 \ldots \varphi_n] \end{array}$$

Disjunction of DRSs can be defined in the usual way.

In the box notation, a DRS looks like this:



where the φ_i are conditions and the x_i introduced variables. An example of a conditional DRS built from two other DRSs in box notation looks like this:



The language of DRT resembles that of QDL and DPL′ in its 'mixed mode' nature. This carries over to the semantics.

Models for the DRS-language are the same as those for DPL, as are assignments and the interpretation of terms. Conditions are interpreted as FOL-formulae, whereas DRSs get a relational meaning. Thus, like in the case of QDL (cf., section 6)), the semantics is defined by simultaneous recursion. Note that we use total assignments instead of partial ones, as is customarily the case in DRT. For present purposes, the difference can be neglected.

$$\begin{array}{rll} \mathbf{M} \models_g Rt_1 \ldots t_n & \text{iff} & \langle [\![t_1]\!]^{\mathbf{M}}_g \ldots [\![t_n]\!]^{\mathbf{M}}_g \rangle \in F_{\mathbf{M}}(R) \\ \mathbf{M} \models_g t_1 = t_2 & \text{iff} & [\![t_1]\!]^{\mathbf{M}}_g = [\![t_2]\!]^{\mathbf{M}}_g \} \\ \mathbf{M} \models_g \neg\Phi & \text{iff} & \text{there exists no } h\colon {}_g[\![\Phi]\!]^{\mathbf{M}}_h \\ \mathbf{M} \models_g \Phi_1 \Rightarrow \Phi_2 & \text{iff} & \text{for all } h\colon \text{ if } {}_g[\![\Phi]\!]^{\mathbf{M}}_h \text{ there exists a } k\colon {}_h[\![\Phi_2]\!]^{\mathbf{M}}_k \\ {}_g[\![[x_1 \ldots x_k][\varphi_1 \ldots \varphi_n]]\!]^{\mathbf{M}}_h & \text{iff} & g \sim_{x_1 \ldots x_k} h \,\&\, \mathbf{M} \models_h \varphi_1 \ldots \mathbf{M} \models_h \varphi_n \end{array}$$

DRT and DPL The close link between DRT and DPL is illustrated by the following embedding of DRT into DPL:

$$\begin{array}{rcl}
(Rt_1 \ldots t_n)\dagger & = & Rt_1 \ldots t_n \\
(t_i = t_j)\dagger & = & t_i = t_j \\
(\neg\Psi)\dagger & = & \neg(\Psi\dagger) \\
(\Phi_1 \Rightarrow \Phi_2)\dagger & = & \Phi_1\dagger \rightarrow \Phi_2\,\dagger \\
([x_1 \ldots x_k][\varphi_1 \ldots \varphi_n])\dagger & = & \exists x_1 \ldots \exists x_n[\varphi_1\,\dagger \wedge \ldots \wedge \varphi_n\dagger]
\end{array}$$

The embedding is meaning-preserving in the following sense:

$$\begin{array}{rcl}
\mathbf{M} \models_g \varphi & \text{iff} & \text{there exists an } h\colon {}_g[\![\varphi\dagger]\!]^{\mathbf{M}}_h \\
{}_g[\![\Phi]\!]^{\mathbf{M}}_h & \text{iff} & {}_g[\![\Phi\dagger]\!]^{\mathbf{M}}_h
\end{array}$$

Context As it turns out, the notion of a context in DRT does not differ all that much from the one DPL is concerned with: both model basically the same features of a discourse context. But the two systems model context in different ways: DPL uses only assignments and operations on them, DRT uses special types of expressions in its syntax.

7.2.4 Variations and extensions

A number of variations on DRT, DPL and other systems have been proposed in the literature. Some are motivated by reasons of formal simplicity and elegance, others by conceptual and descriptive reasons. It is beyond the scope of this article to discuss them extensively; here it suffices to point to a number of issues motivating these alternatives.

Partial assignments One difference between DPL and DRT is the use that the former makes of total assignment functions, instead of the partial ones used by DRT. The choice for partial assignments, that interpret only the variables that are explicitly introduced in a discourse, is a natural one from the perspective of a procedural interpretation, which was one of the motivations of the original DRT-system (cf. above). The use of total assignments in the original DPL system was mainly motivated by a wish to stay as close as possible to the semantics of standard first order logic. Reformulating the DPL-semantics using partial assignments is an easy exercise. We simply let the interpretation be undefined in case a formula contains occurrences of variables that are not in the domain of the assignment function. The only interesting case is the existential quantifier. Here we should let the quantifier extend the domain of the assignment function, if necessary, and let it assign an arbitrary value to the new element in its domain. Cf., e.g., [129] and the system in section 7.4 below.

Fresh variables One of the advantages of using partial assignments is that it becomes more natural to constrain the use of variables in the syntax. Recall some of the more awkward logical properties of DPL, such as the failure of reflexivity of entailment:

$$Px \wedge \exists x Px \not\models Px \wedge \exists x Px$$

This essential depends on the possibility of a variable occurring in the same formula first free and then bound by an existential quantifier. One way of preventing this (and similar) issues, is to require the existential quantifier to always use a 'fresh' variable. Cf., also the discussion below, on incremental semantics.

Compositionality As the preceding discussion will have made clear, the discussion between DPL and DRT centres on compositionality. In DRT the representational level of DRSs plays an essential role, and the cognitive plausibility of the resulting system depends on their presence (cf., the discussion in [79, section 1]). Other formulations of a compositional alternative for DRT have been proposed by, among others, Zeevat [136], Muskens [97], and Van Eijck and Kamp [41]. DPL's reliance on an indexing mechanism on variables to account for anaphoric binding has been criticised since it diminishes the plausibility of the appeal to compositionality considerations, and has spurred a number of alternative approaches:, such as Dekker's 'predicate logic with anaphora' [32], [21]. Cf., also the incremental system discussed below in section 7.2.5, and the combination of update semantics and dynamic semantics in section 7.4.

Stacks and registers The use of DPL as a theory of testing an resetting registers was explored by Visser [133] and Vermeulen [129,130]. The basic idea of a stack semantics for DPL is developed in [131]. The idea is to replace the destructive assignment of ordinary DPL, which throws away old values when resetting, by a stack valued one, that allows old values to be re-used. Stack valued assignments assign to each variable a stack of values, the top of the stack being the current value. Existential quantification pushes a new value on the stack, but there is also the possibility of popping the stack, to re-use a previously assigned value. Adding explicit 'push' and 'pop'-operators to the language, has some interesting consequences. An illustrative example concerns its efficiency in expressing mixed scopes.The idea is as follows. We add $[_x$ and $_x]$ as two new programs and define their semantics as follows:

$$\begin{array}{lll} {}_g[\![\, [_x \,]\!]_h & \text{iff} & g[x\rangle h \\ {}_g[\![\, {}_x] \,]\!]_h & \text{iff} & h[x\rangle g \end{array}$$

where $g[x\rangle h$ holds by definition iff there is a d in the domain with $h(x) = d : g(x)$, (i.e., $h(x)$ equals the result of pushing d on top of the x-stack of g), and

$h(y) = g(y)$ for all y with $y \neq x$. Clearly, the programs $[_x$ and $_x]$ then function as pop and push for the x-stack.

Now consider the FOL-statement:

$$\exists x \exists y \exists z \exists u (Rxy \wedge Ryz \wedge Rzu \wedge Rux)$$

This can be expressed in DPL more succinctly as:

$$\exists x \exists y (\exists z Rxy \wedge \exists x (Ryx \wedge Rxz) \wedge Rzx)$$

But using the push and pop programs we can express the same in terms of only two variables.

$$[_x\, [_y Rxy [_x Ryx [_y Rxy_x] Ryx_y]\, _x]\, _y]$$

The variable free indexing of [36] is a special case of the Vermeulen method, where there is just a single variable. Below we take a variation on DPL with variable free indexing as point of departure for the development of a fragment of dynamic Montague grammar.

7.2.5 Incremental Semantics

Destructive assignment is the main weakness of DPL as a basis for a compositional semantics of natural language: in DPL, the semantic effect of a quantifier action $\exists x$ is such that the previous value of x gets lost. In what follows we first replace DPL by the strictly incremental system from [36]. Subsequently, we develop its type theoretic version. This will allow us to give of a fully compositional and incremental semantics that is without the destructive assignment flaw. Similar ideas were developed in [30, 31].

We start with a slight variation of the DPL language, in which $\exists$ is a separate expression and ; is used for dynamic conjunction. Assume a first order model $M = (D, F)$. We will use contexts $c \in D^*$, and replace variables by indices into contexts. The set of terms of the language is $\mathbb{N}$. We use $|c|$ for the length of context c.

Given a model $M = (D, F)$ and a context $c = c[0] \cdots c[n-1]$, where $n = |c|$ (the length of the context), we interpret terms of the language by means of $[\![i]\!]_c = c[i]$. Note that $[\![i]\!]_c$ is undefined for $i \geq |c|$; we will therefore have to make sure that indices are only evaluated in appropriate contexts. $\uparrow$ will be used for 'undefined'. This allows us to define the two relations

$$\mathbf{M} \models_c Ri_1 \cdots i_n \text{ and } \mathbf{M} =\!\!|_c Ri_1 \cdots i_n$$

by means of:

$$\mathbf{M} \models_c Ri_1 \cdots i_n \;:\Leftrightarrow\; \forall j (1 \leq j \leq n \rightarrow [\![i_j]\!]_c \neq \uparrow) \text{ and } \langle [\![i_1]\!]_c, \ldots, [\![i_n]\!]_c \rangle \in F(R),$$

$\mathbf{M} \nvDash_c Ri_1 \cdots i_n \;:\Leftrightarrow\; \forall j(1 \leq j \leq n \rightarrow [\![i_j]\!]_c \neq \uparrow)$ and $\langle [\![i_1]\!]_c, \ldots, [\![i_n]\!]_c \rangle \notin F(R)$, and similarly for the relations:

$$\mathbf{M} \models_c i_1 = i_2, \quad M \nvDash_c i_1 = i_2$$

If $c \in D^n$ and $d \in D$ we use $c\hat{}d$ for the context $c' \in D^{n+1}$ that is the result of appending d at the end of c.

The interpretation of formulae can now be given as a map in $D^* \hookrightarrow \mathcal{P}(D^*)$ (a partial function, because of the possibility of undefinedness):

$$[\![\exists]\!](c) := \{c\hat{}d \mid d \in D\}$$

$$[\![Ri_1 \cdots i_n]\!](c) := \begin{cases} \uparrow & \text{if } \exists j(1 \leq j \leq n \text{ and } [\![i_j]\!]_c = \uparrow) \\ \{c\} & \text{if } M \models_c Pi_1 \cdots i_n \\ \emptyset & \text{if } M \nvDash_c Pi_1 \cdots i_n \end{cases}$$

$$[\![i_1 = i_2]\!](c) := \begin{cases} \uparrow & \text{if } [\![i_1]\!]_c = \uparrow \text{ or } [\![i_1]\!]_c = \uparrow \\ \{c\} & \text{if } M \models_c i_1 = i_2 \\ \emptyset & \text{if } M \nvDash_c i_1 = i_2 \end{cases}$$

$$[\![\neg\varphi]\!](c) := \begin{cases} \uparrow & \text{if } [\![\varphi]\!](c) = \uparrow \\ \{c\} & \text{if } [\![\varphi]\!](c) = \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![\varphi_1 \;;\; \varphi_2]\!](c) := \begin{cases} \uparrow & \text{if } [\![\varphi_1]\!](c) = \uparrow \\ & \text{or } \exists c' \in [\![\varphi_1]\!](c) \text{ with } [\![\varphi_2]\!](c') = \uparrow \\ \bigcup\{[\![\varphi_2]\!](c') \mid c' \in [\![\varphi_1]\!](c)\} & \text{otherwise.} \end{cases}$$

The definition of $[\![\varphi_1 \;;\; \varphi_2]\!]$ employs the fact that all contexts in $[\![\varphi]\!](c)$ have the same length This property follows by an easy induction on formula structure from the definition of the relational semantics. Thus, if one element $c' \in [\![\varphi_1]\!](c)$ is such that $[\![\varphi_2]\!](c') = \uparrow$, then all $c' \in [\![\varphi_1]\!](c)$ have this property.

Dynamic implication $\varphi_1 \rightarrow \varphi_2$ is defined in terms of $\neg$ and $\;;\;$ by means of $\neg(\varphi_1 \;;\; \neg\varphi_2)$. Universal quantification $\forall\varphi$ is defined in terms of $\exists, \neg$ and $\;;\;$ as $\neg(\exists \;;\; \neg\varphi)$, or alternatively as $\exists \rightarrow \varphi$.

One advantage of the use of contexts is that indefinite NPs do not have to carry index information anymore. Thus a sentence such as 'Some man loved some woman' can be analysed as:

$$\exists \;;\; Mi \;;\; \exists \;;\; Wi+1 \;;\; Li(i+1)$$

where i denotes the length of the input context. On the empty input context, this gets interpreted as the set of all contexts $[e_0, e_1]$ that satisfy the relation

'love' in the model under consideration. The result of this is that a subsequent sentence 'He$_0$ kissed her$_1$.' can use this contextual discourse information to pick up the references. Thus we assume that pronouns carry index information. But if a procedure for reference resolution of pronouns in context is added we can do away with that assumption.

7.2.6 Extension to Type Logic

Compositionality has always been an important concern in the use of logical systems in natural language semantics. And it is through the use of higher order logics (such as type theory) that a thoroughly compositional account of, e.g., the quantificational system of natural language could be achieved. The prime example of this development is that of classical Montague Grammar [92–94]. Cf., [100] for an overview. It is only natural, therefore that the dynamic approach was extended to higher order systems.

However, the various proposals that have been made, such as [24, 33, 41, 54, 77, 84, 88, 95–97], all share a problem with the DPL-system, viz., that of making re-assignment destructive. Interestingly, DRT itself does not suffer from this problem: the discourse representation construction algorithms of [79] and [81] are stated in terms of functions with finite domains, and carefully talk about 'taking a fresh discourse referent' to extend the domain of a verifying function, for each new noun phrase to be processed.

Here we present the extension to typed logic of incremental dynamics that is based on variable free indexing and that avoids the destructive assignment problem.

We now extend incremental dynamic semantics to a higher order language. The resulting system is called Incremental Type Logic (ITL) [35]. Exploiting techniques from polymorphic type theory [67, 91] it uses type specifications of contexts that carry information about the length of the context. E.g., the type of a context is given as $[e]_i$, where i is a type variable. Here, we will cavalierly use $[e]$ for the type of any context, and ι for the type of any index, thus relying on meta-context to make clear what the current constraints on context and indexing into context are. In types such as $\iota \to [e]$, we will tacitly assume that the index fits the size of the context. Thus, $\iota \to [e]$ is really a type scheme rather than a type, although the type polymorphism remains hidden from view. Since $\iota \to [e]$ generalises over the size of the context, it is shorthand for the types $0 \to [e]_0$, $1 \to [e]_1$, $2 \to [e]_2$, and so on.

Let us illustrate this by considering how this applies to the ordinary static higher order translation of an indefinite noun phrase. In extensional Montague grammar 'a man' translates as:

$$\lambda P \exists x(\text{man } x \wedge Px).$$

In ITL this becomes:

$$\lambda P \lambda c \lambda c'.\exists x(\text{man } x \wedge P|c|(c\hat{}x)c').$$

Here P is a variable of type $\iota \rightarrow [e] \rightarrow [e] \rightarrow t$, while c, c' are variables of type $[e]$ (variables ranging over contexts). The translation as a whole has type $(\iota \rightarrow [e] \rightarrow [e]) \rightarrow [e] \rightarrow [e] \rightarrow t$. The P variable marks the slot for the VP interpretation. $|c|$ gives the length of the input context, i.e., the position of the next available slot. Note that $c\hat{}x[|c|] = x$.

Note that the translation of the indefinite NP does not introduce an anaphoric index, as would be the case for example in DMG [54]. Instead, an anaphoric index i is picked up from the input context. Also, the context is not reset but incremented: context update is not destructive, whereas it is in DPL and DMG.

In order to obtain a proper dynamic higher order system we first define the appropriate dynamic operations in typed logic. Assume φ and ψ have the type of context transitions, i.e., type $[e] \rightarrow [e] \rightarrow t$, and that c, c', c'' have type $[e]$. Note that $\hat{}$ is an operation of type $[e] \rightarrow e \rightarrow [e]$.

$$\begin{array}{rcl} \mathcal{E} & := & \lambda c c'.\exists x(c\hat{}x = c') \\ \sim\varphi & := & \lambda c c'.(c = c' \wedge \neg\exists c''\varphi c c'') \\ \varphi \ ; \ \psi & := & \lambda c c'.\exists c''(\varphi c c'' \wedge \psi c'' c') \end{array}$$

These operations encode the semantics for incremental quantification, dynamic incremental negation and dynamic incremental conjunction in typed logic. Dynamic implication, $\Rightarrow$, is defined in the usual way.

We have to assume that the lexical meanings of CNs, VPs are given as one-place predicates (type $e \rightarrow t$) and those of TVs as two place predicates (type $e \rightarrow e \rightarrow t$). We therefore define blow-up operations for lifting one-placed and two-placed predicates to the dynamic level. Let A be an expression of type $e \rightarrow t$, and B an expression of type $e \rightarrow e \rightarrow t$; we use c, c' as variables of type $[e]$, and j, j' as variables of type ι, and we employ postfix notation for the lifting operations:

$$\begin{array}{rcl} A^{\circ} & := & \lambda j c c'.(c = c' \wedge Ac[j]) \\ B^{\bullet} & := & \lambda j j' c c'.(c = c' \wedge Bc[j]c[j']) \end{array}$$

The encodings of the dynamic operations in typed logic and the blow-up operations for one- and two-placed predicates are employed in the semantic specification of the following simple fragment. The semantic specifications employ

variables P, Q of type $\iota \to [e] \to [e] \to t$, variables j, j' of type ι, and variables c, c' of type $[e]$.

We also define an operation ! : $(\iota \to [e] \to [e] \to t) \to [e] \to [e] \to t$ (from lifted one-place predicates to context transformers), to express that a lifted predicate applies to a single individual in a given context. Assuming P to be an expression of type $(\iota \to [e] \to [e] \to t)$ (a lifted predicate), and c, c' to be of type $[e]$ (contexts), we define ! as follows:

$$!P \quad := \quad \lambda cc'.\exists x \forall y(P|c|(c\hat{}y)c' \leftrightarrow x = y).$$

This expresses that P is the lift of a predicate that applies to a single individual.

As said above, we assume that pronouns are the only NPs that carry indices; pronoun reference resolution is not treated. Appropriate indices for proper names are extracted from the current context. In the rules, X refers to the semantics of the left-hand side of the syntax rule, to be defined in terms of the semantic translations of the members of the right-hand side of the syntax rule. X_i refers to the semantics of the i-th member of the right-hand side of the syntax rule.

S	::=	**NP VP**	X	::=	$(X_1 X_2)$
S	::=	*if* **S S**	X	::=	$X_2 \Rightarrow X_3$
S	::=	**S . S**	X	::=	$X_1 \,;\, X_3$
NP	::=	*Mary*	X	::=	$\lambda Pcc'.\exists j(c[j] = m \wedge Pjcc')$
NP	::=	PRO^k	X	::=	$\lambda Pcc'.(Pkcc')$
NP	::=	**DET CN**	X	::=	$(X_1 X_2)$
NP	::=	**DET RCN**	X	::=	$(X_1 X_2)$
DET	::=	*every*	X	::=	$\lambda PQc.(\sim(\mathcal{E} \,;\, P\|c\| \,;\, \sim Q\|c\|))c$
DET	::=	*some*	X	::=	$\lambda PQc.(\mathcal{E} \,;\, P\|c\| \,;\, Q\|c\|)c$
DET	::=	*no*	X	::=	$\lambda PQc.(\sim(\mathcal{E} \,;\, P\|c\| \,;\, Q\|c\|))c$
DET	::=	*the*	X	::=	$\lambda PQc.(!P \,;\, \mathcal{E} \,;\, P\|c\| \,;\, Q\|c\|)c$
CN	::=	*man*	X	::=	M°
CN	::=	*woman*	X	::=	W°
CN	::=	*boy*	X	::=	B°
RCN	::=	**CN** *that* **VP**	X	::=	$\lambda j.((X_1\ j) \,;\, (X_3\ j))$
RCN	::=	**CN** *that* **NP TV**	X	::=	$\lambda j.((X_1\ j) \,;\, (X_3(\lambda j'.((X_4\ j')j))))$
VP	::=	*laughed*	X	::=	L°
VP	::=	*smiled*	X	::=	S°
VP	::=	**TV NP**	X	::=	$\lambda j.(X_2 \,;\, \lambda j'.((X_1\ j')j))$
TV	::=	*loved*	X	::=	$L'^\bullet$
TV	::=	*respected*	X	::=	$R^\bullet$

Note that determiners do not carry indices, the appropriate index is provided by the length of the input context. It is assumed that all proper names are linked to anchored elements in context. In fact, the anchoring mechanism has been greatly improved by the switch to the incremental, non-destructive approach,

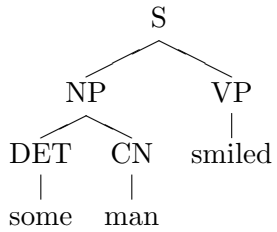
for the incremental nature of the context update mechanism ensures that no anchored elements can ever be overwritten.

The following very simple example illustrates how the system deals with cross-sentential anaphora:

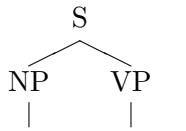
2 *Some man smiled. He laughed.*

The structures assigned to the sentences making up this sequence by the system are the following:

(2) a.

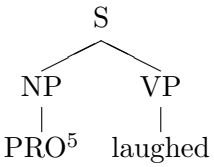


b.



Note that the tree for the second sentence in sequence 2 actually can not be produced by the rules given above: those rules assume that surface pronouns are generated as indexed abstract PRO-elements, as in:

(2) b′.



Translations of the two sentences are derived in a compositional fashion. For example, the NP 'Some man' translates as:

$$(\lambda PQc.(\mathcal{E} \mathbin{;} P|c| \mathbin{;} Q|c|)c)(M^\circ)$$

With S° as the translation of the VP 'smile', the sentence, 'Some man smiled' then receives the following translation:

$$\mathcal{E} \mathbin{;} M^\circ|c| \mathbin{;} S^\circ|c|$$

This is an expression of type $[e] \rightarrow [e] \rightarrow t$ and denotes a relation between contexts. It takes a context and extends it with an object that is both a man and that smiles, as is evident if we reduce it as follows, using the definitions of the dynamic existential quantifier, the dynamic conjunction and the lift operation.

We first rewrite $\mathcal{E}$:

$$(\lambda cc'.\exists x(c\hat{}x = c') \mathbin{;} M^\circ|c| \mathbin{;} S^\circ|c|$$

and next the lifted predicates:

$$(\lambda cc'.\exists x(c\hat{}x = c') \mathbin{;} (\lambda cc'(c = c' \wedge Mc[|c|]) \mathbin{;} (\lambda cc'(c = c' \wedge Sc[|c|])$$

The indefinite determiner extends the context with a new object. The other clauses test the last element of the current context for the properties M and S, respectively.

Rewriting the dynamic conjunction shows how the element introduced by the indefinite determiner is passed on to the other clauses. The first two clauses become:

$$\lambda cc'.\exists c''((\lambda cc'.\exists x(c\hat{}x = c')cc'' \wedge (\lambda cc'(c = c' \wedge Mc[|c|])c''c')$$

which after some reduction becomes:

$$\lambda cc'.\exists x(c\hat{}x = c' \wedge Mx)$$

Rewriting the second occurrence of the dynamic conjunction gives the following reduced translation for the first sentence:

$$\lambda cc'.\exists x(c\hat{}x = c' \wedge Mx \wedge Sx)$$

For the second sentence we get:

$$\lambda cc'.(L^{\circ}5cc')$$

which reduces to

$$\lambda cc'.(c = c' \wedge Lc[5])$$

and for the sequence as a whole we get:

$$\lambda cc'.\exists x(c\hat{}x = c' \wedge Mx \wedge Sx) \ ; \ \lambda cc'.(c = c' \wedge Lc[5])$$

which reduces to:

$$\lambda cc'.\exists x(c\hat{}x = c' \wedge Mx \wedge Sx \wedge Lc[5])$$

Note that we obtain the reading in which the pronoun in the second sentence of 2 refers back to the man introduced in the first sentence only if the index of the PRO-element is suitably chosen. This means that this approach relies on a separate pronoun resolution component in the grammar.

7.3 Update Semantics

In section we illustrate the use of dynamic logic in another area of natural language semantics, one that is concerned with epistemic concerns, modal expressions and with the interaction between issues that are strictly semantic and phenomena that are of a pragmatic nature, i.e., that pertain to the use of language in information exchange.

The gist of the dynamic approach to natural language meaning is captured in the slogan 'Meaning is context-change potential'. In the case of a theory such

as DPL, the context consists of assignments of objects (individuals) satisfying certain properties to variables, In that case, context-change means change of assignments. Such changes are brought about typically by referring expressions such as proper names or temporal expressions and by quantificational expressions such as noun phrases or tense operators. All other expressions are tests. In the case of DRT a different notion of context is used, viz. that of a discourse representation that contains discourse referents satisfying certain properties that point to objects satisfying corresponding properties: here context change is change of the discourse representation. With respect to empirical coverage that does not make a difference, again it is referential and quantificational expressions that change the context, other expressions are treated as parts of conditions.

In epistemic systems, context is yet another type of object, viz., information, modelled by a set of possible worlds or possible situations or propositions. The pioneering work in this area is that of Stalnaker (cf., among others, [115, 116]). Stalnaker focused on the context as the 'common ground', i.e., the information that is available by all speech participants and that is maintained as it gets updated during a linguistic information exchange. This common ground can be characterised as a set of worlds, viz., those worlds which are compatible with the shared information, or, alternatively, as a set of propositions. A linguistic exchange then consists of utterances that shift the context, by updating the common ground, or that test whether something holds in the context. Each utterance represents a particular way of updating or testing the common ground, and this update is conceived as the meaning of the utterance in question.

Within such an approach, sentences that are tests in DPL or conditions in DRT in most cases do have an effect on the context, and thus are treated dynamically. A simple subject-predicate sentence such as 'John is at home' updates the common ground with the information that John has the property of walking, and conjunctions are ordered updates. Examples of exceptions, i.e., sentences that do not update the context but test it, are modal sentences, such as 'John might be at home', and 'John must be at home'. These do not add new information, but check whether the existing common ground satisfies a requirement: that it is possible that John is at home, and that it not possible that John is not at home, respectively.

Another type of linguistic construction that can be treated in this fashion concerns presuppositions. A sentence carrying a presupposition typically tests the common ground for the presence of the presupposed information, besides updating it with new information And yet another example is presented by conditionals: the sentence 'If John is at home, Mary is there, too' tests the context by checking whether updating with the antecedent 'John is at home' leads to a context in which 'Mary is at home' holds.

Of particular interest is what consequences obtain if a test or an update fails.

In the case of a presupposition failing because the information is not present, but is consistent with the common ground, the presupposition is often said to be 'accommodated', i.e., an implicit update takes place [11]. In other cases, e.g., the failure of a test such as 'John might be at home', or of a straightforward update such as 'John is at home', the context needs to be down-dated, i.e., revised. This is the area of belief revision [44] another aspect of the dynamics of information exchange.

7.3.1 System

Update semantics was originally devised as a way of dealing with the semantics of modal expressions such as 'might' and 'must' [125]. These expressions have a specifically epistemic meaning, which makes implicit reference to the information states of speaker and hearer. Other uses of update semantics are, among others, in accounts of conditionals [126], defaults [127], presuppositions [11], [137], and other issues involving information exchange.

Here we present a core system that forms the basis of many variations in the literature.

Let P be a set of atomic sentences. The language is that of propositional logic, with an additional operator $\Diamond$. Assume p ranges over set of basic propositions P.

$$\begin{array}{rcl} \varphi & ::= & p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \\ \varphi' & ::= & \Diamond\varphi \end{array}$$

The other connectives are defined in the usual fashion.

A model $\mathbf{M}$ consists of a set of possible worlds W and in interpretation function $V : P \to \mathcal{P}(W)$. Information states s are subsets of W, with $\emptyset$ the absurd information state, W the state of no information, and singletons $\{w_i\}$ states of maximal information.

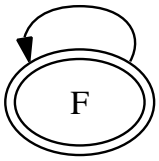
The semantics takes the form of a definition of '$s[\varphi]_{\mathbf{M}}$', i.e., the result of updating an information state s in $\mathbf{M}$ with (the information conveyed by) φ:

$$\begin{array}{rcl} s[p]_{\mathbf{M}} & = & s \cap \{s \in S \mid s \in V_{\mathbf{M}}(p)\} \\ s[\neg\varphi]_{\mathbf{M}} & = & s \setminus s[\varphi]_{\mathbf{M}} \\ s[\varphi_1 \wedge \varphi_2]_{\mathbf{M}} & = & s[\varphi_1]_{\mathbf{M}}[\varphi_2]_{\mathbf{M}} \\ s[\Diamond\varphi]_{\mathbf{M}} & = & \begin{cases} s & \text{if } s[\varphi]_{\mathbf{M}} \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \end{array}$$

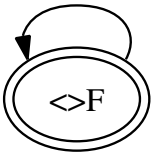
An atomic formula updates s with the information it conveys; a negation $\neg\varphi$ deletes those worlds in which the information conveyed by φ holds from s; conjunction is a sequential update with the conjuncts. The modal $\Diamond\varphi$ is a test:

it returns the original state if an update with φ is possible, the absurd state otherwise.

This system analyses a special case of public announcement logic [46, 103], where the knowledge of a single agent is modelled. The model $\mathbf{M}$ above can be viewed as an S5 model with a universal accessibility relation [38]. Updating with a propositional formula F has the effect of announcing F to the agent, i.e., updating with action model



in the sense of [6]. Updating with a modal formula $\Diamond F$ boils down to updating with the following action model:



The notion of 'acceptance in $\mathbf{M}$, s' is defined as follows:

$$s \models_{\mathbf{M}} \varphi \text{ iff } s \subseteq s[\varphi]_{\mathbf{M}}$$

Validity can be defined in a number of ways; the most common one is as follows:

$$\varphi_1 \dots \varphi_n \models \psi \text{ iff } \text{ for all } \mathbf{M}, s\colon s[\varphi_1]_{\mathbf{M}}\psi \dots [\varphi_n]_{\mathbf{M}} \models \psi$$

I.e., every state that accepts the premises, accepts the conclusion.

This system is eliminative ($s[\varphi]_{\mathbf{M}} \subseteq \varphi$); not distributive ($s \subseteq s' \not\Rightarrow s[\varphi] \subseteq s'[\varphi]$); neither right- nor left-monotone; and conjunction is not commutative. A complete sequent calculus can be found in [59, chapter 3].

7.3.2 Characteristic examples

A characteristic example, that illustrates the non-commutativity of conjunction, involves the $\Diamond$-operator. If we read it as the formal counterpart of the modal expression 'might' (in its epistemic meaning), and represent discourse sequencing as conjunction, we can explain the difference between the following two sentences:

a. Somebody is knocking at the door ... It might be John ... It is Mary
b. Somebody is knocking at the door ... It is Mary ... *It might be John

In the first sequence the second sentence 'It might be John' tests the state (that contains the information that somebody is at the door, due to the update with the first sentence) for the possibility that the person knocking is John. If that succeeds, it is only confirmed that this is a possibility. The subsequent update with the information that in fact it is Mary, is consistent with that. In the second sequence the information that it is Mary is added before the test takes place, resulting in its failure, which explains the odd status of this sequence.

The failure of right- en left-monotonicity is also due to the $\Diamond$-operator:

$$\begin{aligned} \Diamond\neg\varphi \models \Diamond\neg\varphi \quad &\text{but} \quad \Diamond\neg\varphi, \varphi \not\models \Diamond\neg\varphi \\ \models \Diamond\varphi \quad &\text{but} \quad \neg\varphi \not\models \Diamond\varphi \end{aligned}$$

Another instantiation of the ideas behind update semantics is provided by conditionals. Many aspects of conditionals in natural language can be captured in an update framework, by keeping in mind the 'modal' nature of the conditional construction:

$$s[\varphi_1 \rightarrow \varphi_2]_{\mathbf{M}} \;=\; \{i \in s \mid \text{ if } i \in s[\varphi_1]_{\mathbf{M}} \text{ then } i \in s[\varphi_1]_{\mathbf{M}}[\varphi_2]_{\mathbf{M}}\}$$

The update effect of a condition thus is to retain those possibilities in a given state s such that updating them with the antecedent allows a subsequent update with the consequent.

Applications of update semantics can be found in a variety of areas, such as deontic modality [123]; interrogatives [53]; imperatives [89,135]; counterfactuals and other irrealis-constructions [128].

7.4 Combining dynamic and update semantics

The dynamic semantics used in systems such as DPL and DRT can be combined with an update type of semantics as just defined. Various proposals exists (cf., e.g., [31, 57]). The idea is to put the semantics for quantified formulae in an update format. In [58] this is done as follows.

Existential quantifiers introduce new kind of objects, so-called 'pegs', modelled by the natural numbers. This notion was first introduced by Vermeulen, cf., [129]. A referent system r is a function from a finite set of variables to pegs. An existential quantifier $\exists x$ add its variable x introduces the next peg and associates x with that peg. So, if r is a referent system with domain v and range of pegs n, then $r[x/n]$ is the referent system r' which is like r except that its domain is $v \cup \{x\}$ its range is $N + 1$ and $r'(x) = n$. Let r and r' be two referent systems with domain v and v', and range n and n', respectively. Then we say that r' is an *extension* of r, $r \leq r'$, iff $v \subseteq v'$; $n \leq n'$; if $x \in v$ then $r(x) = r'(x)$ or $n \leq r'(x)$; if $x \notin v$ and $x \in v'$ then $n \leq r'(x)$.

States s are sets of triples i consisting of the same referent system r, an assignment g and a world w. So states contain information about both the world (via the possible world parameter) as well as the discourse (via the referent system). Growth of information is then twofold as well: via the elimination of possibilities, and via extension of the referent system. First we introduce:

$$\begin{aligned} i[x/d] &= \langle r[x/n], g[n/d], w\rangle \\ s[x/d] &= \{i[x/d] \mid i \in s\} \end{aligned}$$

and then we define these two notions of information growth as follows. Let $i, i' \in I, i = \langle r, g, w\rangle$ and $i' = \langle r', g', w'\rangle$, and $s, s' \in S$:

$$\begin{aligned} i' \leq i' \quad &\text{iff} \quad r \leq r', g \subseteq g', w = w' \\ s \leq s' \quad &\text{iff} \quad \text{for all } i' \in s' \text{: there exists an } i \in s \text{: } i \leq i' \end{aligned}$$

Finally, we define the update semantics for existentially quantified formulae $\exists x\varphi$ as follows (the other clauses are merely repetitions of the above):

$$s[\exists x\varphi]_{\mathbf{M}} = \cup_{d \in D_{\mathbf{M}}}(s[x/d][\varphi]_{\mathbf{M}})$$

This defines the update effect of $\exists x\varphi$ point-wise on the objects in the domain: the referent system of the state s is updated by adding a peg, the variable is associated with the peg, and an object d is selected and assigned to the peg; then the resulting state $s[x/d]$ is updated with φ; this procedure is repeated for every object in the domain; the results are collected and together make up the new state $s[\exists x\varphi]$.

The resulting system is capable of treating complex cases concerning the interaction of quantifiers and modalities. For example it can be used to show that whereas $\exists xPx \wedge \Diamond\forall y\neg Py$ is not consistent, $\exists xPx \wedge \forall y\Diamond\neg Py$ is: if we know that something has the property P this ipso facto rules out the possibility that no-one has that property, but it does not rule out the possibility that we are uninformed about the identity of this P. For other examples, involving also identity we refer the reader to [58] and [1].

8 Concluding remarks

The overview of dynamic logics and their applications presented in this paper has focused on a number of core systems (Floyd/Hoare logic, PDL, epistemic PDL, QDL, DPL), and a number of central applications: program analysis, tree description, analysis of communication, semantics of natural language. References to other applications were thrown in as an incentive to the reader for further exploration.

The field of dynamic logic, including its applications in various domains, is still developing. Dynamic logic started out as a way of studying various aspects of computation, mainly in traditional computational settings, with a focus on sequential transformational programs. When theoretical computer science broadened to encompass the theory of reactive systems and concurrency, dynamic logic evolved by developing systems that could handle these too (branching time logics and μ calculus). Thus, the core concepts of dynamic logic have proved to be applicable in a wide range of settings, allowing formalisation of a great diversity of concepts and phenomena.

In certain areas, such as natural language semantics, the use of dynamic concepts initially arose independently, and it was only subsequently that these notions were embedded in dynamic logic. This have given rise to interesting interactions, that are still being actively pursued.

The application to communicative action stays somewhat closer to the original motivation for the development of dynamic logic. Here the use of dynamic logic ties in with an existing tradition of using modal logic in the analysis of communication protocols [60]. Also in the analysis of various other phenomena that are concerned with interactions between individuals and with properties of the collectives (groups, societies) that they form, concepts of dynamic logic play a role, as is testified by work done on, for example, collective decisions (cf., [102] on game logic as an extension of propositional dynamic logic).

As more aspects of the ways in which human beings interact are brought into the picture, concepts like perception, causality, justification and intention appear. Here insights from the philosophy of action and from game theory must augment the tool set from dynamic logic, thus creating an exciting amalgam of logic, theoretical computer science, philosophy and game theory. Whatever the future holds in store for this area, it seems more than likely that concepts and results from dynamic logic will continue to play a major role in its development.

Acknowledgement Thanks to Albert Visser, Johan van Benthem, Balder ten Cate and Marc Pauly for useful feedback on various drafts of this paper.

References

[1] Aloni, M. Conceptual covers in dynamic semantics. In *Words, Proofs and Diagrams*, D. Barker-Plummer, D. Beaver, J. van Benthem, and P. Scotto di Luzio, Eds. CSLI, 2002.

[2] Apt, K. Ten years of Hoare's logic: A survey—part i. *ACM Transactions on Programming Languages and Systems 3*, 4 (1981), 431–483.

[3] Apt, K., and Bezem, M. Formulas as programs. In *The Logic Programming Paradigm: a 25 Years Perspective*, K. Apt, V. Marek, M. Truszczyski, and D. Warren, Eds. Springer Verlag, 1999, pp. 75–107. Paper available as `http://xxx.lanl.gov/abs/cs.LO/9811017`.

[4] Areces, C., Blackburn, P., and Marx, M. Hybrid logics: Characterization, interpolation and complexity. *Journal of Symbolic Logic* (2001).

[5] Baltag, A. A logic for suspicious players: epistemic action and belief-updates in games. *Bulletin of Economic Research 54*, 1 (2002), 1–45.

[6] Baltag, A., and Moss, L. Logics for epistemic programs. *Synthese 139*, 2 (2004), 165–224.

[7] Baltag, A., Moss, L., and Solecki, S. The logic of public announcements, common knowledge, and private suspicions. Tech. Rep. SEN-R9922, CWI, Amsterdam, 1999.

[8] Baltag, A., Moss, L., and Solecki, S. The logic of public announcements, common knowledge, and private suspicions. Tech. rep., Dept of Cognitive Science, Indiana University and Dept of Computing, Oxford University, 2003.

[9] Barwise, J. Noun phrases, generalized quantifiers and anaphora. In *Generalized Quantifiers: Linguistic and Logical Approaches*, P. Gärdenfors, Ed. Reidel, Dordrecht, 1987, pp. 1–30.

[10] Barwise, J., and Perry, J. *Situations and Attitudes*. MIT Press, Cambridge, Mass., 1983.

[11] Beaver, D. I. Presupposition. In *Handbook of Logic and Language*, J. v. Benthem and A. ter Meulen, Eds. Elsevier/MIT Press, Amsterdam/Cambridge Mass., 1997, pp. 939–1008.

[12] Belnap, N. D., Perloff, M., and Xu, M. *Facing the Future*. Oxford University Press, Oxford, 2001.

[13] Benthem, J. v. *Modal Correspondence Theory*. PhD thesis, University of Amsterdam, 1976.

[14] Benthem, J. v. Programming operations that are safe for bisimulation. *Studia Logica 60*, 2 (1994), 311–330.

[15] Benthem, J. v. *Exploring Logical Dynamics*. CSLI, Stanford, 1996.

[16] Benthem, J. v., and Cepparello, G. Tarskian variations. dynamic parameters in classical semantics. Research Report CS-R9419, Centrum voor Wiskunde en Informatica, Amsterdam, 1994.

[17] Benthem, J. v., Muskens, R., and Visser, A. Dynamics. In *Handbook of Logic and Linguistics*, J. v. Benthem and A. ter Meulen, Eds. Elsevier, 1997, pp. 587–648.

[18] Blackburn, P., de Rijke, M., and Venema, Y. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001.

[19] Blackburn, P., Gardent, C., and Meyer-Viol, W. Talking about trees. In *Proceedings of the sixth conference of the European chapter of the Association for Computational Linguistics* (1993), Association for Computational Linguistics, pp. 21–29.

[20] Boole, G. *An investigation of the laws of thought, on which are founded the mathematical theories of logic and probabilities*, dover (reprint) ed. Dover, 1854 (first edition).

[21] Butler, A., and mathieu, E. *The Syntax and Semantics of Split Constructions*. Palgrave, London, 2004.

[22] Cate, B. t., van Eijck, J., and Heguiabehere, J. Expressivity of extensions of dynamic predicate logic. In *Proceedings of the Thirteenth Amsterdam Colloquium, December 17–19, 2001* (2001), R. van Rooy and M. Stokhof, Eds., pp. 61–66.

[23] Chang, C., and Keisler, H. *Model Theory*. North-Holland, Amsterdam, 1973.

[24] Chierchia, G. Anaphora and dynamic binding. *Linguistics and Philosophy 15*, 2 (1992), 111–183.

[25] Chierchia, G. *Dynamics of Meaning*. University of Chicago Press, Chicago, 1995.

[26] Clarke, E., and Emerson, E. Synthesis of synchronisation skeletons for branching time temporal logic. In *Logic of Programs Workshop* (1981), D. Kozen, Ed., no. 131 in LNCS, Springer.

[27] Clarke, E., Grumberg, O., and Long, D. Verification tools for finite-state concurrent systems. In *A Decade of Concurrency* (1993), no. 803 in LNCS, Springer, pp. 124–175.

[28] Cook, S. Soundness and completeness of an axiom system for program verification. *SIAM Journal of Computing 7*, 1 (1978), 70–90.

[29] Davey, B., and Priestley, H. *Introduction to Lattices and Order (Second Edition)*. Cambridge University Press, Cambridge, 2002. First edition: 1990.

[30] Dekker, P. Predicate logic with anaphora (seven inch version). In *Proceedings of the Fourth SALT Conference* (Cornell University, 1988), L. Santelmann and M. Harvey, Eds., DMLL Publications.

[31] Dekker, P. The values of variables in dynamic semantics. *Linguistics and Philosophy 19* (1996), 211–57.

[32] Dekker, P. Meaning and use of indefinite expressions. *Journal of Logic, Language and Information 11* (2002), 141–94.
[33] Eijck, J. v. Typed logics with states. *Logic Journal of the IGPL 5*, 5 (1997), 623–645.
[34] Eijck, J. v. Axiomatising dynamic logics for anaphora. *Journal of Language and Computation 1* (1999), 103–126.
[35] Eijck, J. v. The proper treatment of context in NL. In *Computational Linguistics in the Netherlands 1999; Selected Papers from the Tenth CLIN Meeting* (2000), P. Monachesi, Ed., Utrecht Institute of Linguistics OTS, pp. 41–51.
[36] Eijck, J. v. Incremental dynamics. *Journal of Logic, Language and Information 10* (2001), 319–351.
[37] Eijck, J. v. Reducing dynamic epistemic logic to PDL by program transformation. CWI, Amsterdam, `www.cwi.nl:~/papers/04/delpdl/`, 2004.
[38] Eijck, J. v., and de Vries, F. Reasoning about update logic. *Journal of Philosophical Logic 24* (1995), 19–45.
[39] Eijck, J. v., and de Vries, F.-J. Dynamic interpretation and Hoare deduction. *Journal of Logic, Language and Information 1*, 1 (1992), 1–44.
[40] Eijck, J. v., Heguiabehere, J., and Nualláin, B. O. Tableau reasoning and programming with dynamic first order logic. *Logic Journal of the IGPL 9*, 3 (May 2001).
[41] Eijck, J. v., and Kamp, H. Representing discourse in context. In *Handbook of Logic and Language*, J. v. Benthem and A. ter Meulen, Eds. Elsevier, Amsterdam, 1997, pp. 179–237.
[42] Fischer, M. J., and Ladner, R. E. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences 18*, 2 (1979), 194–211.
[43] Floyd, R. Assigning meanings to programs. In *Proceedings AMS Symposium Applied Mathematics* (Providence, R.I., 1967), vol. 19, American Mathematical Society, pp. 19–31.
[44] Gärdenfors, P. The dynamics of belief as a basis for logic. *British Journal for the Philosophy of Science 35* (1984), 1–10.
[45] Gerbrandy, J. *Bisimulations on planet Kripke*. PhD thesis, ILLC, 1999.
[46] Gerbrandy, J. Dynamic epistemic logic. In *Logic, Language and Information, Vol. 2*, L. Moss et al., Eds. CSLI Publications, Stanford, 1999.
[47] Gochet, P. The dynamic turn in twentieth century logic. *Synthese 130* (2002), 175–84.
[48] Gochet, P., and Gribomont, P. Epistemic logic. In *Handbook of the History of Logic, Volume 6 – Logic and the Modalities in the Twentieth Century*, D. Gabbay and J. Woods, Eds. Elsevier, to appear.
[49] Goldblatt, R. *Axiomatising the Logic of Computer Programming*. Springer, 1982.

[50] Goldblatt, R. *Logics of Time and Computation, Second Edition, Revised and Expanded*, vol. 7 of *CSLI Lecture Notes*. CSLI, Stanford, 1992 (first edition 1987). Distributed by University of Chicago Press.

[51] Gordon, M. *Programming language theory and its implementation: applicative and imperative paradigms*. Prentice Hall, 1988.

[52] Gries, D. *The Science of Programming*. Springer, Berlin, 1981.

[53] Groenendijk, J. The logic of interrogation: Classical version. In *Proceedings of SALT IX, Santa Cruz* (1999), T. Matthews and D. Strolovitch, Eds., CLC Publications.

[54] Groenendijk, J., and Stokhof, M. Dynamic Montague grammar. In *Papers from The Second Symposium on Logic and Language*, L. Kálmán and L. Pólos, Eds. Akadémiai Kiadó, Budapest, 1990, pp. 3–48.

[55] Groenendijk, J., and Stokhof, M. Dynamic predicate logic. *Linguistics and Philosophy 14* (1991), 39–100.

[56] Groenendijk, J., and Stokhof, M. Meaning in motion. In *Reference and Anaphoric Relations*, K. von Heusinger and U. Egli, Eds. Kluwer, Dordrecht, 2000, pp. 47–76.

[57] Groenendijk, J., Stokhof, M., and Veltman, F. This might be it. In *Language, Logic and Computation: The 1994 Moraga Proceedings*, D. Westerståhl and J. Seligman, Eds. CSLI, Stanford, 1995.

[58] Groenendijk, J., Stokhof, M., and Veltman, F. Coreference and modality. In *Handbook of Contemporary Semantic Theory*, S. Lappin, Ed. Blackwell, Oxford, 1996, pp. 179–213.

[59] Groeneveld, W. *Logical Investigations into Dynamic Semantics*. PhD thesis, ILLC, Amsterdam, 1995.

[60] Halpern, J., Moses, Y., and Vardi, M. *Reasoning about Knowledge*. MIT Press, Cambridge, Mass., 1995.

[61] Harel, D. *First-Order Dynamic Logic*. No. 68 in Lecture Notes in Computer Science. Springer, 1979.

[62] Harel, D. Dynamic logic. In *Handbook of Philosophical Logic*, D. Gabbay and F. Guenthner, Eds. Reidel, Dordrecht, 1984, pp. 497–604. Volume II.

[63] Harel, D., Kozen, D., and Tiuryn, J. *Dynamic Logic. Foundations of Computing*. MIT Press, Cambridge, Massachusetts, 2000.

[64] Heguiabehere, J. Pre- and postcondition reasoning in DFOL. Manuscript, ILLC, 2001.

[65] Heim, I. *The Semantics of Definite and Indefinite Noun Phrases*. PhD thesis, University of Massachusetts, Amherst, 1982.

[66] Heim, I. File change semantics and the familiarity theory of definiteness. In *Meaning, Use, and Interpretation of Language*, R. Bäuerle, C. Schwarze, and A. von Stechow, Eds. de Gruyter, Berlin, 1983.

[67] Hindley, J. R. *Basic Simple Type Theory*. Cambridge University Press, 1997.

[68] Hintikka, J. *Knowledge and Belief: An Introduction to the Logic of the Two Notions*. Cornell University Press, Ithaca N.Y., 1962.
[69] Hintikka, J. *The Game of Language*. Reidel, Dordrecht, 1983.
[70] Hoare, C. An axiomatic basis for computer programming. *Communications of the ACM 12*, 10 (1969), 567–580, 583.
[71] Hollenberg, J. Equational axioms of test algebra. Logic Group Preprint Series 172, Department of Philosophy, Utrecht University, 1996.
[72] Hollenberg, M. An equational axiomatisation of dynamic negation and relational composition. *Journal of Logic, Language and Information 6* (1997), 381–401.
[73] Hollenberg, M. *Logic and Bisimulation*. PhD thesis, Utrecht University, 1998.
[74] Hollenberg, M., and Visser, A. Dynamic negation, the one and only. Logic Group Preprint Series 179, Department of Philosophy, Utrecht University, Heidelberglaan 8, 3584 CS Utrecht, 1997. To appear in JoLLI.
[75] Huth, M., and Ryan, M. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2000.
[76] Janin, D., and Walukiewicz, I. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *CONCUR '96: Proceedings of the 7th International Conference on Concurrency Theory* (1996), Springer-Verlag, pp. 263–277.
[77] Jansche, M. Dynamic Montague Grammar lite. Dept of Linguistics, Ohio State University, November 1998.
[78] Jones, S. P., Ed. *Haskell 98 Language and Libraries; The Revised Report*. Cambridge University Press, 2003.
[79] Kamp, H. A theory of truth and semantic representation. In *Formal Methods in the Study of Language* (1981), J. Groenendijk et al., Eds., Mathematisch Centrum, Amsterdam.
[80] Kamp, H. Comments on: J. Groenendijk & M. Stokhof, Dynamic Predicate Logic. In *Partial and Dynamic Semantics, Part I. Dyana Deliverable 2.1.A*, J. van Benthem, Ed. ILLC, Universiteit van Amsterdam, 1990, pp. 109–31.
[81] Kamp, H., and Reyle, U. *From Discourse to Logic*. Kluwer, Dordrecht, 1993.
[82] Kanazawa, M. Weak vs. strong readings of donkey sentences and monotonicity inference in a dynamic setting. *Linguistics and Philosophy 17* (1994), 109–58.
[83] Karttunen, L. Discourse referents. In *Syntax and Semantics 7*, J. McCawley, Ed. Academic Press, 1976, pp. 363–385.
[84] Kohlhase, M., Kuschert, S., and Pinkal, M. A type-theoretic semantics for λ-DRT. In *Proceedings of the Tenth Amsterdam Colloquium* (Amsterdam, 1996), P. Dekker and M. Stokhof, Eds., ILLC.
[85] Kooi, B., and van Benthem, J. Reduction axioms for epistemic actions. Manuscript, Groningen/Amsterdam, 2004.

[86] Kozen, D. Results on the propositional μ-calculus. *Theoretical Computer Science 27* (1983), 333–354.

[87] Kracht, M. Syntactic codes and grammar refinement. *Journal of Logic, Language and Information 4* (1995), 41–60.

[88] Kuschert, S. *Dynamic Meaning and Accommodation*. PhD thesis, Universität des Saarlandes, 2000. Thesis defended in 1999.

[89] Lascarides, A., and Asher, N. Imperatives in dialogue. In *Perspectives on Dialogue in the New Millenium*, K. Peter, Ed. John Benjamins, Amsterdam/Philadelphia, 2003.

[90] Marx, M. XCPath, the first order complete XPath dialect. In *Proceedings of PODS* (2004), ACM SIGMOD, pp. 13–22.

[91] Milner, R. A theory of type polymorphism in programming. *Journal of Computer and System Sciences 17* (1978).

[92] Montague, R. Universal grammar. *Theoria 36* (1970), 373–98.

[93] Montague, R. The proper treatment of quantification in ordinary English. In *Approaches to Natural Language*, J. H. e.a., Ed. Reidel, 1973, pp. 221–242.

[94] Montague, R. English as a formal language. In *Formal Philosophy; Selected Papers of Richard Montague*, R. Thomason, Ed. Yale University Press, New Haven and London, 1974, pp. 188–221.

[95] Muskens, R. A compositional discourse representation theory. In *Proceedings 9th Amsterdam Colloquium*, P. Dekker and M. Stokhof, Eds. ILLC, Amsterdam, 1994, pp. 467–486.

[96] Muskens, R. Tense and the logic of change. In *Lexical Knowledge in the Organization of Language*, U. E. et al., Ed. W. Benjamins, 1995, pp. 147–183.

[97] Muskens, R. Combining Montague Semantics and Discourse Representation. *Linguistics and Philosophy 19* (1996), 143–186.

[98] Nielson, H., and Nielson, F. *Semantics with Applications*. John Wiley and Sons, 1992.

[99] Parikh, R. The completeness of propositional dynamic logic. In *Mathematical Foundations of Computer Science 1978*. Springer, 1978, pp. 403–415.

[100] Partee, B. H. Montague grammar. In *Handbook of Logic and Linguistics*, J. van Benthem and A. ter Meulen, Eds. Elsevier, 1997, pp. 5–92.

[101] Passy, S., and Tinchev, T. An essay in combinatory dynamic logic. *Inf. Comput. 93*, 2 (1991), 263–332.

[102] Pauly, M., and Parikh, R. Game logic: An overview. *Studia Logica 75*, 2 (2003), 165–82.

[103] Plaza, J. A. Logics of public communications. In *Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems* (1989), M. L. Emrich, M. S. Pfeifer, M. Hadzikadic, and Z. W. Ras, Eds., pp. 201–216.

[104] Plotkin, G. *Structural Operational Semantics*. Aarhus University, Denmark, 1981. Lecture notes.

[105] Pnueli, A. A temporal logic of programs. *Theoretical Computer Science 13* (1981), 45–60.

[106] Pratt, V. Semantical considerations on Floyd–Hoare logic. *Proceedings 17th IEEE Symposium on Foundations of Computer Science* (1976), 109–121.

[107] Pratt, V. A practical decision method for propositional dynamic logic. In *Proceedings 10th Symposium Theory of Computation* (1978), ACM, pp. 326–337.

[108] Pratt, V. Application of modal logic to programming. *Studia Logica 39* (1980), 257–274.

[109] Prior, A. *Time and Modality*. Oxford University Press, 1957.

[110] Prior, A. *Past, Present and Future*. Clarendon Press, Oxford, 1967.

[111] Ruan, J. Exploring the update universe. Master's thesis, ILLC, Amsterdam, 2004.

[112] Schmidt, D. *Denotational Semantics: a Methodology for Language Development*. Allyn and Bacon, 1986.

[113] Segerberg, K. A completeness theorem in the modal logic of programs. In *Universal Algebra and Applications*, T. Traczyck, Ed. Polish Science Publications, 1982, pp. 36–46.

[114] Segerberg, K. Getting started: Beginnings in the logic of action. *Studia Logica 51* (1992).

[115] Stalnaker, R. Pragmatic presuppositions. In *Semantics and Philosophy*, M. Munitz and P. Unger, Eds. New York University Press, 1974, pp. 197–213.

[116] Stalnaker, R. Assertion. In *Syntax and Semantics Vol.9: Pragmatics*, P. Cole, Ed. Academic Press, New York, 1979, pp. 315–332.

[117] Stalnaker, R. On the representation of context. *Journal of Logic, Language and Information 7* (1998).

[118] Staudacher, P. Zur Semantik indefiniter Terme. In *Neue Forschungen zur Wortbildung und Historiographie der Linguistik*, B. Asbach-Schnitker and J. Roggenhofer, Eds. Gunter Narr Verlag, Tübingen, 1987, pp. 239–58.

[119] Stoy, J. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

[120] Streett, R. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control 54*, 1/2 (July/August 1982), 124–141.

[121] Streett, R., and E.A, E. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation 81*, 3 (1989), 249–264.

[122] Tarski, A. On the calculus of relations. *Journal of Symbolic Logic 6* (1941), 73–89.

[123] Torre, L. v., and Tan, Y.-H. An update semantics for deontic reasoning. In *Proceedings of DEON '98* (1998), pp. 409–26.

[124] Turing, A. On computable real numbers, with an application to the Entscheidung problems. *Proceedings of the London Mathematical Society 2*, 42 (1936), 230–265.

[125] Veltman, F. Data semantics. In *Truth, Interpretation and Information*, J. Groenendijk, T. M. Janssen, and M. Stokhof, Eds. Foris, Dordrecht, 1984, pp. 43–62.

[126] Veltman, F. Data semantics and the pragmatics of indicative conditionals. In *On Conditionals*, J. R. Elizabeth Traugott, Alice ter Meulen and C. Ferguson, Eds. Cambridge University Press, 1986.

[127] Veltman, F. Defaults in update semantics. *Journal of Philosophical Logic 25* (1996), 221–61.

[128] Veltman, F. Making counterfactual assumptions. *Journal of Semantics* (2005).

[129] Vermeulen, C. Merging without mystery. Variables in dynamic semantics. *Journal of Philosophical Logic 24* (1995), 405–50.

[130] Vermeulen, C. Variables as stacks. *Journal of Logic, Language and Information 9* (2000), 143–67.

[131] Vermeulen, C. A calculus of substitutions for DPL. *Studia Logica 68* (2001).

[132] Visser, A. Dynamic Relation Logic is the logic of DPL-relations. *Journal of Logic, Language and Information 6*, 4 (1997), 441–452.

[133] Visser, A. Contexts in dynamic predicate logic. *Journal of Logic, Language and Information 7*, 1 (1998), 21–52.

[134] Wright, G. v. *Practical Reason*. Blackwell, Oxford, 1983.

[135] Zarnic, B. Dynamic semantics, imperative logic and propositional attitudes. Preprint, Uppsala Universitet, 2002.

[136] Zeevat, H. A compositional approach to discourse representation theory. *Linguistics and Philosophy 12* (1989), 95–131.

[137] Zeevat, H. Presupposition and accomodation in update semantics. *Journal of Semantics 98* (1992), 379–412.