



The United Nations  
University

**UNU-IIST**

International Institute for  
Software Technology

---

# Reasoning about QoS Contracts in the Probabilistic Duration Calculus

---

Dimitar P. Guelev and Dang Van Hung

September 2007

## UNU-IIST and UNU-IIST Reports

UNU-IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macao, and was founded in 1991. It started operations in July 1992. UNU-IIST is jointly funded by the government of Macao and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endowment Fund. As well as providing two-thirds of the endowment fund, the Macao authorities also supply UNU-IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU-IIST is to assist developing countries in the application and development of software technology.

UNU-IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU-IIST, and
7. Dissemination, in which UNU-IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU-IIST is on formal methods for software development. UNU-IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU-IIST produces a report series. Reports are either Research R, Technical T, Compendia C or Administrative A. They are records of UNU-IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU-IIST at P.O. Box 3058, Macao or visit UNU-IIST's home page: <http://www.iist.unu.edu>, if you would like to know more about UNU-IIST and its report series.

G. M. Reed, Director



The United Nations  
University

**UNU-IIST**

International Institute for  
Software Technology

P.O. Box 3058  
Macao

---

# Reasoning about QoS Contracts in the Probabilistic Duration Calculus

---

Dimitar P. Guelev and Dang Van Hung

## Abstract

The notion of *contract* was introduced to component-based software development in order to facilitate the semantically correct composition of components. We extend the form of this notion which is based on *designs* to capture probabilistic requirements on execution time. We show how reasoning about such requirements can be done in an infinite-interval-based system of probabilistic duration calculus.

Dimitar P. Guelev is a research fellow at the Section of Mathematical Logic of the Institute of Mathematics and Informatics of the Bulgarian Academy of Sciences. The research presented in this report was carried out during D. Guelev's research visit to UNU/IIST in August-September 2007. E-mail: [gelevdp@math.bas.bg](mailto:gelevdp@math.bas.bg)

Dang Van Hung is from the Institute of Information Technology of National Center for Natural Science and Technology of Vietnam, where he is a researcher. He was a fellow of UNU/IIST from April 1994 until July 1995. He has been a research fellow of UNU/IIST since October 1995. His research interests include formal techniques of programming and real-time embedded systems. E-mail: [dvh@iist.unu.edu](mailto:dvh@iist.unu.edu)

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Preliminaries</b>	<b>2</b>
1.1 <i>ITL</i> with infinite intervals . . . . .	2
1.2 <i>DC</i> with infinite intervals . . . . .	3
1.3 The least-fixed-point operator and higher-order quantifiers . . . . .	4
1.4 Probabilistic <i>ITL</i> and <i>DC</i> with infinite intervals . . . . .	4
1.4.1 Models and satisfaction . . . . .	4
1.4.2 Proof system for <i>PITL</i> . . . . .	6
1.4.3 Global probability in <i>PITL</i> . . . . .	6
1.4.4 Probabilistic <i>DC</i> . . . . .	7
<b>2 A toy concurrent programming language and its semantics in <i>DC</i> with infinite intervals</b>	<b>7</b>
2.1 Syntax . . . . .	7
2.2 Semantics . . . . .	8
<b>3 Reasoning about timed programs in probabilistic <i>DC</i></b>	<b>10</b>
<b>4 Probabilistic timed designs</b>	<b>15</b>
4.1 Definition . . . . .	15
4.2 Describing designs in <i>PDC</i> . . . . .	15
4.3 Refinement of probabilistic timed designs . . . . .	16
<b>5 Probabilistic timed contracts</b>	<b>16</b>
5.1 Definition . . . . .	16
5.2 Composing probabilistic timed contracts . . . . .	17
<b>Concluding remarks</b>	<b>18</b>
<b>References</b>	<b>18</b>
<b>A Proof systems</b>	<b>20</b>
A.1 Proof system for <i>ITL</i> with infinite intervals . . . . .	20
A.2 Axioms and rules for <i>DC</i> with infinite intervals . . . . .	21
A.3 Proof system for <i>PITL</i> . . . . .	22
A.4 Useful theorems and derived rules for <i>PITL</i> . . . . .	22
A.5 The rule <i>Seq</i> . . . . .	25



## Introduction

Building software by combining off-the-shelf and dedicated components has become an established approach to achieving reuse, modularity, productivity and reliability in the software development process. *Contracts* have been proposed as an instrument to facilitate the correct use of components. A contract is a collection of conditions written in terms of the interface of the component. It specifies requirements which any implementation of the component is supposed to satisfy, provided that the other components on which it depends for its implementation similarly satisfy corresponding requirements, which are included in the contract too. Ideally, it can be determined whether a given combination of components would work correctly by checking whether the contracts of the individual components agree on the relevant requirements. Four levels of contracts have been identified in [BJPW99]. These are the *syntactical* level, which is essentially about interface compatibility as known in programming languages, the *behavioural* level, the *synchronization* level and the *quality of service* level. Quality of Service (QoS) is a collective term for non-functional requirements such as worst-case and average execution time, and the consumption of resources such as memory, power, bandwidth, etc.

Component models are built around appropriate formalisations of the notions of *interface*, *contract*, *component composability*, *composition*, etc. A contract theory for components based on the notion of *design* from [HH98] has been proposed in [HLL06, HXZ06] and has become known as the rCOS model. Since designs capture input-output relations, this model is mostly about the *functional* requirements on components, that is, about contracts which do not capture the last level from [BJPW99]. QoS has not been given special attention in this model. In our previous work, we have extended the rCOS component model to enable the specification and reasoning on requirements on timing and resources [Dan05, HD07]. We have considered *hard* requirements, where, e.g., not meeting a deadline is regarded as fatal. We used the Duration Calculus (*DC*) as our notation. QoS is mainly concerned with *soft* requirements, where, e.g., not meeting a deadline can be tolerated as long as its probability is low and the delay is reasonably small. That is why the formal reasoning about requirements on the QoS of components involves reasoning about probability. This was confirmed by our experience with the benchmark CoCoME example [CHD<sup>+</sup>07].

In this paper we develop a technique to reason about QoS of real-time embedded systems using a probabilistic extension of *DC*. We extend designs to capture probabilistic requirements on execution time. We show how reasoning about such requirements can be done in an infinite-interval-based system of probabilistic *DC* (*PDC*). A probability operator was first introduced to *DC* in [LRSZ93] for the case of discrete time. A real-time system of *PDC* was first proposed in [DZ99]. The system of *PDC* we are using in this paper was proposed in [Gue07] as an extension of a corresponding system of Probabilistic Interval Temporal Logic with infinite intervals (*PITL*). *PDC* with infinite intervals subsumes the systems of *PDC* from [LRSZ93, DZ99, Gue00b] and has a relatively complete proof system to support formal reasoning. The fitness of (non-probabilistic) *DC* for reasoning about real-time systems has been asserted by numerous case studies [ZZ94, DW96, SX98, Dan98, LH99]. Since *DC* is interval-based, reasoning about the behaviour of whole method executions, including their execution time, is relatively straightforward in *DC*. By using a probabilistic extension of *DC* we are able to enjoy this advantage when reasoning about QoS requirements too.

The paper is organised as follows. We first give preliminaries on *PITL* and *PDC* and its underlying system of *ITL* with infinite intervals. The proof system for *PITL* from [Gue07] is minimal and barely complete. To facilitate deduction, we extend it by a number of useful theorems and derived rules, which we give in the Appendix. Next we propose a toy real-time concurrent programming language in order to illustrate our approach. We give a formal semantics of this programming language in *DC* with infinite intervals. Then we give some examples of reasoning about the probabilistically distributed execution time of programs written in this language using this formal semantics. In the examples we demonstrate

the use the derived rules and theorems from our extended proof system for *PITL* for the calculation of the probability of execution time. Having thus provided some motivation for our setting, we propose a way to extend designs as known from [HH98] to *probabilistic timed designs* and introduce a corresponding notion of *probabilistic timed contracts*. We define composition of probabilistic timed contracts and show how the execution time distributions appearing in the composition can be calculated using those given in the contracts being composed. We conclude the paper by some comments on our approach.

## 1 Preliminaries

Here follows a brief formal introduction to the systems of *PITL* and *PDC* from [Gue07]. We consider only the extended set of the real numbers  $\bar{\mathbf{R}} = \mathbf{R} \cup \{\infty\}$  as the flow of time. We first describe the underlying non-probabilistic logic *ITL* with infinite intervals [ZDL95, PWX98, SX98, WX04]. In order to facilitate the description of repetitive behaviour, we include a least-fixed-point operator for non-probabilistic formulas, which was introduced in [Pan95] and studied in [Gue00a].

### 1.1 *ITL* with infinite intervals

*ITL* extends the syntax of predicate logic by a binary modality  $(.;.)$ , known as *chop*.<sup>1</sup> Non-logical symbols are divided into *rigid* and *flexible* depending on whether their meaning is required to be the same at all reference intervals or not. Individual variables are rigid.

**Models and satisfaction** We use the sets of intervals

$$\mathbf{I}^{fin} = \{[\tau_1, \tau_2] : \tau_1, \tau_2 \in \mathbf{R}, \tau_1 \leq \tau_2\}, \mathbf{I}^{inf} = \{[\tau, \infty] : \tau \in \mathbf{R}\} \text{ and } \tilde{\mathbf{I}} = \mathbf{I}^{fin} \cup \mathbf{I}^{inf}.$$

Given  $\sigma_1 \in \mathbf{I}^{fin}$  and  $\sigma_2 \in \tilde{\mathbf{I}}(T)$  such that  $\max \sigma_1 = \min \sigma_2$ ,  $\sigma_1; \sigma_2$  stands for  $\sigma_1 \cup \sigma_2$ .

An *interpretation of a vocabulary*  $\mathbf{L}$  is a function  $I$  on  $\mathbf{L}$  which maps the symbols from  $\mathbf{L}$  to members of  $\bar{\mathbf{R}}$ , functions and predicates on  $\bar{\mathbf{R}}$ , according to the type and arity of symbols.  $I(s)$  takes an interval from  $\tilde{\mathbf{I}}$  as an additional argument in case  $s$  is flexible. The interpretations of  $0$ ,  $\infty$ ,  $+$  and  $=$ , which are mandatory in *ITL* vocabularies, are always the standard ones. Given an interpretation  $I$ , the values  $I_\sigma(t)$  of terms  $t$  at intervals  $\sigma \in \tilde{\mathbf{I}}(T)$  are defined in the usual way, with the reference interval being the additional argument for flexible symbols. There is a mandatory flexible constant  $\ell$ , which always evaluates to the length of the reference interval. The satisfaction relation  $\models$  is defined by the clauses:

$$\begin{array}{ll} I, \sigma \not\models \perp & \\ I, \sigma \models R(t_1, \dots, t_{\#R}) & \text{iff } I(R)(I_\sigma(t_1), \dots, I_\sigma(t_{\#R})) = 1 \text{ for rigid } R \\ I, \sigma \models R(t_1, \dots, t_{\#R}) & \text{iff } I(R)(\sigma, I_\sigma(t_1), \dots, I_\sigma(t_{\#R})) = 1 \text{ for flexible } R \\ I, \sigma \models (\varphi \Rightarrow \psi) & \text{iff either } I, \sigma \not\models \varphi \text{ or } I, \sigma \models \psi \\ I, \sigma \models (\varphi; \psi) & \text{iff } I, \sigma_1 \models \varphi \text{ and } I, \sigma_2 \models \psi \\ & \text{for some } \sigma_1 \in \mathbf{I}^{fin} \text{ and } \sigma_2 \in \tilde{\mathbf{I}} \text{ such that } \sigma_1; \sigma_2 = \sigma \\ I, \sigma \models \exists x \varphi & \text{iff } I_x^d, \sigma \models \varphi \text{ for some } d \in \bar{\mathbf{R}} \end{array}$$

<sup>1</sup>Many authors write chop as  $\varphi \frown \psi$  instead of  $(\varphi; \psi)$ .

Here  $\#R$  stands for the arity of  $R$ , and  $I_x^d$  is defined by the equalities  $I_x^d(x) = d$  and  $I_x^d(s) = I(s)$  for  $s \neq x$ .

**Abbreviations and precedence of operators** *Infix* notation for arithmetics and equality and  $\top$ ,  $\wedge$ ,  $\Rightarrow$ ,  $\Leftrightarrow$  and  $\forall$  are used in the usual way. The *universal closure*  $\forall x_1 \dots \forall x_n \varphi$  of a formula  $\varphi$  where  $\{x_1, \dots, x_n\}$  is the set  $FV(\varphi)$  of the free variables of  $\varphi$  is written as  $\forall \varphi$ .

Since  $(.;.)$  is associative, we omit the nested parentheses in formulas with consecutive occurrences of  $(.;.)$ . Here follow the infinite-interval versions of some *ITL* abbreviations:

$$\diamond \varphi \Leftrightarrow (\top; \varphi; \top) \vee (\top; \varphi) , \quad \square \varphi \Leftrightarrow \neg \diamond \neg \varphi .$$

Note that the meanings of  $\square$  and  $\diamond$  in the original discrete-time system of *ITL* of Moszkowski [CMZ, Mos86] are different. Our usage originates from the literature on *DC*. The disjunctive member  $(\top; \varphi)$  in the definition of  $\diamond$  is relevant only at infinite intervals. The formula  $(\top; \varphi; \top)$  without it restricts the subinterval which satisfies  $\varphi$  to be finite.

$\diamond$  and  $\square$  bind more tightly and  $(.;.)$  binds less tightly than the boolean connectives.

**Proof system** Axioms and rules which have been shown to form a complete proof system for *ITL* with infinite intervals when added to a Hilbert-style proof system for classical first-order predicate logic with respect to an appropriate abstract domain of durations in [WX04] are given in Section A.1.

## 1.2 *DC* with infinite intervals

*DC vocabularies* are *ITL* vocabularies extended by *state variables*  $P, Q, \dots$ , which are used to build *state expressions*  $S$  using boolean connectives. Logical constants are written  $\mathbf{0}$  and  $\mathbf{1}$  in *DC* state expressions. State expressions in turn appear as the argument of *duration terms*  $\int S$ , which are the *DC*-specific construct in the syntax of *DC* terms. Formulas are as in *ITL*. In *DC interpretations* state variables evaluate to piece-wise constant functions of type  $\overline{\mathbf{R}} \rightarrow \{0, 1\}$ . The value  $I_\tau(S)$  of state expression  $S$  at time  $\tau$  is defined using  $I(P)(\tau)$  for the involved state variables  $P$  in the usual way. The value  $I_\sigma(\int S)$  of duration term  $\int S$  at interval  $\sigma \in \tilde{\mathbf{I}}(\overline{\mathbf{R}})$  is

$$I_\sigma(\int S) = \int_{\min \sigma}^{\max \sigma} I_\tau(S) d\tau$$

$I_\sigma(\int S)$  can be  $\infty$  for  $\sigma \in \mathbf{I}^{inf}(\overline{\mathbf{R}})$ . The values of other kinds of terms and  $\models$  are defined as in *ITL*. Flexible relation symbols of arity 0 are called *propositional temporal letters* in *DC*.

The expression  $[S]$  abbreviates  $\ell \neq 0 \wedge \int \neg S = 0$  and  $\ell$  can be viewed as an abbreviation of  $\int \mathbf{1}$  in *DC*.

**Proof system** Axioms and rules for both *DC* which have been shown to be complete relative to validity in real-time *ITL* in [HZ92], are given in Section A.2.

### 1.3 The least-fixed-point operator and higher-order quantifiers

If  $\varphi_1, \dots, \varphi_n$  are formulas with no negative occurrences of the propositional temporal letters  $X_1, \dots, X_n$  and  $i \in \{1, \dots, n\}$ , then  $\mu_i X_1 \dots X_n. \varphi_1, \dots, \varphi_n$  is a formula and  $I, \sigma \models \mu_i X_1 \dots X_n. \varphi_1, \dots, \varphi_n$  iff  $\sigma \in A_i$ , where  $A_1, \dots, A_n$  are the smallest subsets of  $\tilde{\mathbf{I}}$  which satisfy the equalities

$$A_i = \{\sigma \in \tilde{\mathbf{I}} : I_{X_1, \dots, X_n}^{\lambda\sigma.\sigma \in A_1, \dots, \lambda\sigma.\sigma \in A_n} \sigma \models \varphi_i\}, \quad i = 1, \dots, n.$$

*Iteration*, also known as Kleene star, can be defined using  $\mu$  as follows

$$\varphi^* \Leftrightarrow \mu_1 X. \ell = 0 \vee (\varphi; X).$$

The satisfaction of  $\varphi^*$  can be defined by the clause:

$I, \sigma \models \varphi^*$  if either  $\min \sigma = \max \sigma$  or  $\sigma = \sigma_1; \dots; \sigma_n$  and  $I, \sigma_i \models \varphi$ ,  $i = 1, \dots, n$ , for some  $n < \omega$ ,  $\sigma_1, \dots, \sigma_n \in \tilde{\mathbf{I}}$ .

Axioms and rules for  $\mu$  in *DC* were proposed in [Pan95, Gue00a].

We use  $\exists$  on flexible constants and state variables with the usual meaning, in order to describe the semantics of local variables. Some axioms and rules about  $\exists$  on flexible constants and state variables and their deductive power have been studied in [ZGZ00, Gue00a].

### 1.4 Probabilistic *ITL* and *DC* with infinite intervals

*PITL* extends the syntax of *ITL* terms by *probability terms* of the form  $p(\varphi)$  where  $\varphi$  is a formula. Formula syntax is as in *ITL*.

#### 1.4.1 Models and satisfaction

A *PITL* model is based on a collection of interpretations of a given vocabulary. These interpretations are meant to describe the possible behaviours of the modelled system.

Consider a non-empty set  $\mathbf{W}$ , a function  $I$  on  $\mathbf{W}$  into the set of the *PITL* interpretations of some vocabulary  $\mathbf{L}$  and a function  $P$  of type  $\mathbf{W} \times \overline{\mathbf{R}} \times 2^{\mathbf{W}} \rightarrow [0, 1]$ . Let  $I^w$  and  $P^w$  abbreviate  $I(w)$  and  $\lambda\tau, X. P(w, \tau, X)$ , respectively, for all  $w \in \mathbf{W}$ .  $I^w$  and  $P^w$ ,  $w \in \mathbf{W}$ , are intended to represent the set of behaviours and the associated probability distributions for every  $\tau \in \overline{\mathbf{R}}$  in the *PITL* models for  $\mathbf{L}$ .

**Definition 1** Let  $\tau \in \overline{\mathbf{R}}$ . We define the equivalence relation  $\equiv_\tau$  on  $\mathbf{W}$  by putting  $w \equiv_\tau v$  iff

$I^w(s) = I^v(s)$  for all rigid symbols  $s \in \mathbf{L}$ , except possibly the individual variables;

$I^w(s)(\sigma, d_1, \dots, d_{\#s}) = I^v(s)(\sigma, d_1, \dots, d_{\#s})$  for all flexible  $s \in \mathbf{L}$ , all  $d_1, \dots, d_{\#s} \in \overline{\mathbf{R}}$  and all  $\sigma \in \tilde{\mathbf{I}}$  such that  $\max \sigma \leq \tau$ ;

$P^w(\tau', X) = P^v(\tau', X)$  for all  $X \subseteq \mathbf{W}$  and all  $\tau' \leq \tau$ .

Given  $w \in \mathbf{W}$  and  $\tau \in \overline{\mathbf{R}}$ , we denote the set

$$\{v \in \mathbf{W} : v \equiv_{\tau} w\}$$

by  $\mathbf{W}_{w,\tau}$ .

Members of  $\mathbf{W}$  which are  $\tau$ -equivalent model the same behaviour up to time  $\tau$ . If  $\tau_1 > \tau_2$ , then  $\equiv_{\tau_1} \subset \equiv_{\tau_2}$  and  $w \equiv_{\infty} v$  holds iff  $P^w = P^v$  and  $I^w$  and  $I^v$  agree on all symbols, except possibly some individual variables.  $\mathbf{W}_{w,\tau}$  is the set of those  $v \in \mathbf{W}$  which represent the probabilistic branching of  $w$  from time  $\tau$  onwards.

**Definition 2** A *general PITL model* for  $\mathbf{L}$  is a tuple of the form  $\langle \mathbf{W}, I, P \rangle$  where  $\mathbf{W}$ ,  $I$  and  $P$  are as above and satisfy the following requirements for every  $w \in \mathbf{W}$ :

$\mathbf{W}$  is closed under variants of interpretations. If  $w \in \mathbf{W}$ ,  $x$  is an individual variable from  $\mathbf{L}$  and  $a \in \overline{\mathbf{R}}$ , then there is a  $v \in \mathbf{W}$  such that  $P^v = P^w$  and  $I^v = (I^w)_x^a$ .

$P^w$  represent probability measures. For every  $w \in \mathbf{W}$  and  $\tau \in \overline{\mathbf{R}}$  the function  $\lambda X.P^w(\tau, X)$  is a probability measure on the boolean algebra

$$\langle 2^{\mathbf{W}}, \cap, \cup, \emptyset, \mathbf{W} \rangle.$$

$\lambda X.P^w(\tau, X)$  is required to be concentrated on the set  $\mathbf{W}_{w,\tau}$ .

$$P^w(\tau, X) = P^w(\tau, X \cap \mathbf{W}_{w,\tau}) \text{ for all } X \subseteq \mathbf{W}.$$

Informally, the probability for the system to choose a behaviour in  $X \subseteq \mathbf{W}_{w,\tau}$  is  $P^w(\tau, X)$ . Term values  $w_{\sigma}(t)$ , the satisfaction relation  $\models$  and its associated notation  $\llbracket \cdot \rrbracket$  in *PITL* are given by the following clauses, where the components of the model  $M$  are named as above:

$$\begin{array}{lll} w_{\sigma}(x) & = & I^w(x) & \text{for variables } x \\ w_{\sigma}(c) & = & I^w(c) & \text{for rigid } c \\ w_{\sigma}(f(t_1, \dots, t_{\#f})) & = & I^w(f)(w_{\sigma}(t_1), \dots, w_{\sigma}(t_{\#f})) & \text{for rigid } f \\ w_{\sigma}(c) & = & I^w(c)(\sigma) & \text{for flexible } c \\ w_{\sigma}(f(t_1, \dots, t_{\#f})) & = & I^w(f)(\sigma, w_{\sigma}(t_1), \dots, w_{\sigma}(t_{\#f})) & \text{for flexible } f \\ w_{\sigma}(p(\psi)) & = & P^w(\max \sigma, \llbracket \psi \rrbracket_{M,w,\sigma}) \end{array}$$

If  $\psi$  is a sentence, then

$$\llbracket \psi \rrbracket_{M,w,\sigma} = \{v \in \mathbf{W}_{w,\max \sigma} : M, v, [\min \sigma, \infty] \models \psi\}.$$

This means that  $\llbracket \psi \rrbracket_{M,w,\sigma}$  consists of the interpretations  $v$  which are  $\max \sigma$ -equivalent to  $w$  and satisfy  $\psi$  at the infinite interval starting at  $\min \sigma$ . In case  $\psi$  has free variables  $x_1, \dots, x_n$ ,  $M, v, [\min \sigma, \infty] \models \psi$  should be evaluated with  $I^w(x_1), \dots, I^w(x_n)$  as the values of  $x_1, \dots, x_n$ , in order to preserve the intended meaning. This leads to the following definition:

$$\llbracket \psi \rrbracket_{M,w,\sigma} = \{v \in \mathbf{W}_{w,\max \sigma} : (\forall v' \in \mathbf{W})(P^{v'} = P^v \wedge I^{v'} = (I^v)_{x_1, \dots, x_n}^{I^w(x_1), \dots, I^w(x_n)} \Rightarrow M, v', [\min \sigma, \infty] \models \psi)\}.$$

The clauses for  $\models$  are

$M, w, \sigma \not\models \perp$	
$M, w, \sigma \models R(t_1, \dots, t_{\#R})$	iff $I^w(R)(w_\sigma(t_1), \dots, w_\sigma(t_{\#R})) = 1$ for rigid $R$
$M, w, \sigma \models R(t_1, \dots, t_{\#R})$	iff $I^w(R)(\sigma, w_\sigma(t_1), \dots, w_\sigma(t_{\#R})) = 1$ for flexible $R$
$M, w, \sigma \models (\varphi \Rightarrow \psi)$	iff either $M, w, \sigma \not\models \varphi$ or $M, w, \sigma \models \psi$
$M, w, \sigma \models (\varphi; \psi)$	iff $M, w, \sigma_1 \models \varphi$ and $M, w, \sigma_2 \models \psi$ for some $\sigma_1 \in \mathbf{I}^{fin}(T_F)$ and $\sigma_2 \in \tilde{\mathbf{I}}(T_F)$ such that $\sigma_1; \sigma_2 = \sigma$
$M, w, \sigma \models \mu_i X_1 \dots X_n. \varphi_1, \dots, \varphi_n$	iff $I^w, \sigma \models \mu_i X_1 \dots X_n. \varphi_1, \dots, \varphi_n$ in (non-probabilistic) <i>ITL</i>
$M, w, \sigma \models \exists x \varphi$	iff $M, v, \sigma \models \varphi$ for some $v \in \mathbf{W}$ and some $a$ from the domain of the sort of $x$ such that $P^v = P^w$ and $I^v = (I^w)_x^a$

The probability functions  $\lambda X. P^w(\tau, X)$  for  $w \in \mathbf{W}$  and  $\tau \in T$  in general *PITL* models  $M = \langle \mathbf{W}, I, P \rangle$  are needed just as much as they provide values for probability terms. That is why we accept structures of the form  $\langle \mathbf{W}, P, I \rangle$  with their probability functions  $\lambda X. P^w(\tau, X)$  be defined just on the (generally smaller) algebras  $\langle \{[\psi]_{M,w,\sigma} : \psi \in \mathbf{L}, \sigma \in \tilde{\mathbf{I}}(T), \max \sigma = \tau\}, \cap, \cup, \emptyset, \mathbf{W}_{w,\tau} \rangle$  as general *PITL* models too.

#### 1.4.2 Proof system for *PITL*

*PITL* is a conservative extension of *ITL*. Axioms and a proof rule which extend the proof system for *ITL* with intervals to a system for *PITL* which were shown in [Gue07] to be complete with respect to a generalisation of the  $\mathbf{R}$ -based semantics, where  $\bar{\mathbf{R}}$  is replaced by an abstract domain and the probability measures are required to be only finitely additive, are given in Section A.3.

#### 1.4.3 Global probability in *PITL*

The probability functions  $\lambda X. P^w(\tau, X)$  need not be related to each other in general models for *PITL*, whereas applications typically lead to models in which all the probability functions originate from a *global* probability function on the entire  $\mathbf{W}$ . Assume that we have an origin of time  $\tau_0 = \min T$  and a distinguished  $w_0 \in \mathbf{W}$  such that  $\mathbf{W}_{w_0, \tau_0} = \mathbf{W}$ . Then  $\lambda X. P^{w_0}(\tau_0, X)$  can be regarded as the global probability function and, given an arbitrary  $w \in \mathbf{W}$  and  $\tau \in \mathbf{R}$ , the probability function  $\lambda X. P^w(\tau, X)$  should represent *conditional* probability on sets of interpretations, the condition being  $\tau$ -equivalence with  $w$ . Hence we should have

$$(1) \quad P^{w_0}(\tau, A) = \lim_{A_0, \dots, A_n \text{ is a partition of } \mathbf{W}_{w_0, \tau}} \sum_{i=0}^n \sup_{w \in A_i} P^w(\tau, A) P^{w_0}(\tau, A_i).$$

The following rules enable the derivation of approximations of (1) of arbitrary precision in the proof system of *PITL* for  $A \subseteq \mathbf{W}$  and appropriate directed systems of partitions  $A_0, \dots, A_n$  of  $\mathbf{W}_{w_0, \tau}$  which can be defined by *ITL* formulas:

$$(P) \quad \frac{\varphi \Rightarrow \neg(\varphi; \ell \neq 0)}{\ell = 0 \wedge p(\varphi \wedge p(\psi) < x; \top) = 0 \Rightarrow p((\varphi; \top) \wedge \psi) \geq x.p(\varphi; \top)}$$

$$(\bar{P}) \quad \frac{\varphi \Rightarrow \neg(\varphi; \ell \neq 0)}{\ell = 0 \wedge p(\varphi \wedge p(\psi) > x; \top) = 0 \Rightarrow p((\varphi; \top) \wedge \psi) \leq x.p(\varphi; \top)}$$

The proof system for *PITL* from [Gue07] is minimal. Some useful *PITL* theorems and derived rules, including ones about global probability are given in Section A.4. Here follows a derived rule which is particularly important to our examples. Let

$$[ET_\varphi \in [l, h]]_x \Leftrightarrow \ell = 0 \wedge p(\varphi; \top) = 1 \Rightarrow p(\varphi \wedge \ell \geq l \wedge \ell \leq h; \top) = x$$

and

$$\varphi_l^h \Leftrightarrow \varphi \wedge \ell \geq l \wedge \ell \leq h.$$

Then

$$(Seq) \quad \frac{\alpha \Rightarrow \neg(\alpha; \ell \neq 0), \beta \Rightarrow \neg(\beta; \ell \neq 0), [ET_\alpha \in [l_1, h_1]]_{x_1}, [ET_\beta \in [l_2, h_2]]_{x_2}}{\ell = 0 \wedge p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha \wedge \ell \geq l_1 \wedge \ell \leq h_1; \beta \wedge \ell \geq l_2 \wedge \ell \leq h_2; \top) = x_1 x_2}$$

A derivation of *Seq* is given in Section A.5.

#### 1.4.4 Probabilistic *DC*

The system of probabilistic *DC* (*PDC*) we use in this paper is obtained by adding state variables and duration terms to *PITL* in the way this is done in order to obtain (non-probabilistic) *DC* from *ITL*. The axioms and rules for *DC* with infinite intervals are complete for *PDC* relative to validity in *PITL* models based on  $\overline{\mathbf{R}}$ .

## 2 A toy concurrent programming language and its semantics in *DC* with infinite intervals

In this section we describe a toy language with a minimal set of control structures to illustrate our approach. The language of the one used in [GD02]. We allow a restricted form of method call, which we need in order to set the stage for the use of components and contracts.

### 2.1 Syntax

*Components* import and/or export methods and have the form:

```

component name
  import method
  ...
  import method
  export method
  ...
  export method
end name

```

Methods have the form

$$name(parameter\ list)[code];$$

The part *code* is required only for exported methods. It is defined by the BNF

<i>code</i> ::=	<b>stop</b>	empty statement
	<b>return</b> [ <i>e</i> ]	return control and possibly a value
	<i>X</i>	continuation
	( <i>x</i> := <i>e</i> ; <i>code</i> )	assignment
	( <b>delay</b> <i>r</i> ; <i>code</i> )	delay by the specified amount of time
	( <b>call</b> [ <i>x</i> :=] <i>name</i> ( <i>parameter list</i> ); <i>code</i> )	call method and possibly obtain a value
	<b>if</b> <i>b</i> <b>then</b> <i>code</i> <b>else</b> <i>code</i>	conditional statement
	<b>letrec</b> <i>code</i> <b>where</b> <i>X</i> : <i>code</i> ; ... ; <i>X</i> : <i>code</i>	mutual recursion statement
	<b>var</b> <i>x</i> ; <i>code</i>	local variable declaration
	<i>code</i>    <i>code</i>	parallel composition

Here *b*, and *r* stand for *boolean*- and a *real*-valued expressions, respectively, and *e* stands for an arbitrary appropriately typed expression. We do not allow **var** to occur in the scope of other control statements. Assignments are assumed to be atomic. Parameters are passed by value. A mutual recursion statement can trigger an infinite computation. Tail-recursion is the only kind of repetitive behaviour that is expressible in the syntax of our language. This is achieved by explicitly requiring atomic statements other than **stop**, **return** and labels within **letrec** to be sequentially composed with some subsequent code, and allowing no subsequent code after recursive references to labels. We give no details on the type system and tacitly assume an appropriately many-sorted system of *DC*. Components are passive. The *active* part of a program is just a piece of code, typically a collection of concurrently running interleaved threads.

## 2.2 Semantics

The execution of code produces a behaviour which can be described in terms of the values of its signals, variables and parameters as functions of time. The semantic function  $\llbracket \cdot \rrbracket$  defined below maps every piece of code to a *DC* formula which describes its corresponding set of behaviours. We model each program variable *v* by a corresponding pair of flexible constants *v* and *v'* to express its behaviour. We require these constants to satisfy the axiom  $\forall x \neg(v' = x; v \neq x)$  where *x* is a rigid individual variable. This means that for all intervals  $\sigma$  the value of *v* at  $\sigma$  is the same as the value of *v'* at any  $\sigma'$  such that  $\max \sigma' = \min \sigma$ . This entails that both *v* and *v'* satisfy the *locality principle*, that is, they depend on a single time point for their values. This time point is the beginning of the reference interval for *v* and the end of the reference interval for *v'*. We model every method *m* by a corresponding flexible function symbol or predicate symbol *m*. A function symbol is used if *m* returns a value. The intended meaning of an atomic formula of the form  $v' = m(e_1, \dots, e_n)$  is that the reference interval describes a complete invocation of *m* with  $e_1, \dots, e_n$  as the input parameters and *v'* as the value.

Along with the flexible symbols for program variables and methods, we use dedicated state variables *R* and *W* to indicate that the thread is

currently using the processor, or has terminated, respectively. We also use a state variable *N* to mark computation time, which, unlike the time consumed by the execution of *delay* statements, waiting for the

reaction of the environment, etc., is regarded as Negligible, in order to simplify calculations. The idea to simplify calculations by ignoring computation time was introduced to  $DC$  in [PD98], where time was divided into *micro*- and *macro*-time. We require  $R$ ,  $W$  and  $N$  to satisfy the axioms

$$[R \Rightarrow N], [R \Rightarrow \neg W] \text{ and } \Box \neg([W]; [\neg W]),$$

which express that computation time is negligible, a process can never be both running and terminated, and, once terminated, is never re-activated. We abbreviate the conjunction of these axioms by  $\mathsf{T}(R, W)$ . We use a dedicated pair of state variables  $R$  and  $W$  to describe the status of each thread, whereas  $N$  marks negligible time for all threads. We also use the abbreviations

$$\mathsf{K}(V) \Leftrightarrow \bigwedge_{x \in V} x' = x \text{ and } \mathsf{K}^R(V) \Leftrightarrow \mathsf{K}(V) \wedge [R].$$

$\mathsf{K}(V)$  means that the variables from  $V$  preserve their values.  $\mathsf{K}^R(V)$  means that, in addition, the thread is active throughout the reference interval. The clauses below define  $\llbracket \cdot \rrbracket_V$ , where  $V$  is the set of program variables which are in the scope in the given code.

$$\begin{array}{ll}
\llbracket \mathbf{stop} \rrbracket_V & \Leftrightarrow [W] \\
\llbracket \mathbf{return } e \rrbracket_V & \Leftrightarrow ([\neg R]; \mathsf{K}^R(V) \wedge \mathbf{r}' = e) \\
\llbracket \mathbf{return} \rrbracket_V & \Leftrightarrow [\neg R] \\
\llbracket X \rrbracket_V & \Leftrightarrow X \\
\llbracket (C_1; C_2) \rrbracket_V & \Leftrightarrow (\llbracket C_1 \rrbracket_V; \llbracket C_2 \rrbracket_V) \\
\llbracket x := e \rrbracket_V & \Leftrightarrow ([\neg R]; \mathsf{K}^R(V \setminus \{x\}) \wedge x' = e) \\
\llbracket \mathbf{delay } r \rrbracket_V & \Leftrightarrow [\neg R] \wedge r = \int \neg N \\
\llbracket \mathbf{if } b \mathbf{ then } C_1 \mathbf{ else } C_2 \rrbracket_V & \Leftrightarrow ([\neg R]; (b \wedge \mathsf{K}^R(V); \llbracket C_1 \rrbracket_V) \vee (\neg b \wedge \mathsf{K}^R(V); \llbracket C_2 \rrbracket_V)) \\
\llbracket \mathbf{call } v := m(e_1, \dots, e_n) \rrbracket_V & \Leftrightarrow ([\neg R]; \mathsf{K}(V \setminus \{v\}) \wedge v' = m(e_1, \dots, e_n)) \\
\llbracket \mathbf{call } m(e_1, \dots, e_n) \rrbracket_V & \Leftrightarrow ([\neg R]; \mathsf{K}(V) \wedge m(e_1, \dots, e_n)) \\
\\
\left[ \begin{array}{l} \mathbf{letrec } C \mathbf{ where} \\ X_1 : C_1; \\ \dots \\ X_n : C_n \end{array} \right]_V & \Leftrightarrow \mu_{n+1} X_1 \dots X_n Y. \llbracket C_1 \rrbracket_V, \dots, \llbracket C_n \rrbracket_V, \llbracket C \rrbracket_V \\
\\
\llbracket \mathbf{var } v; C \rrbracket_V & \Leftrightarrow \exists v \exists v' (\Box (([\neg R] \Rightarrow v' = v) \wedge \forall x \neg (v' = x; v \neq x)) \wedge \wedge [C]_{V \cup \{v\}}) \\
\\
\llbracket (C_1 \parallel C_2)_V \rrbracket & \Leftrightarrow \exists R_1 \exists R_2 \exists W_1 \exists W_2 \left( \begin{array}{l} [W \Leftrightarrow W_1 \wedge W_2] \wedge \\ [R \Leftrightarrow R_1 \vee R_2] \wedge [\neg R_1 \wedge \neg R_2] \\ \mathsf{T}(R_1, W_1) \wedge [R_1/R, W_1/W] \llbracket C_1 \rrbracket_V \wedge \\ \mathsf{T}(R_2, W_2) \wedge [R_2/R, W_2/W] \llbracket C_2 \rrbracket_V \end{array} \right) \\
\\
\llbracket \mathbf{export } m(p_1, \dots, p_n) \mathbf{ code} \rrbracket & \Leftrightarrow \Box \forall p_1 \dots \forall p_n \forall \mathbf{r}' (\mathbf{r}' = m(p_1, \dots, p_n) \Leftrightarrow \llbracket \mathbf{code} \rrbracket_\emptyset), \text{ for } m \text{ which return a value} \\
\llbracket \mathbf{export } m(p_1, \dots, p_n) \mathbf{ code} \rrbracket & \Leftrightarrow \Box \forall p_1 \dots \forall p_n (m(p_1, \dots, p_n) \Leftrightarrow \llbracket \mathbf{code} \rrbracket_\emptyset), \text{ for } m \text{ which return no value}
\end{array}$$

Components are not supposed to have static variables and therefore no variable set subscript appears in  $\llbracket \mathbf{export } m(p_1, \dots, p_n) \mathbf{ code} \rrbracket$ . The semantics of a component is the conjunction of the formulas  $\llbracket \mathbf{export } m(p_1, \dots, p_n) \mathbf{ code} \rrbracket$  for its exported methods. Declarations of imported methods carry only typing information.

### 3 Reasoning about timed programs in probabilistic *DC*

Let  $C$  be a piece of code. Then  $\llbracket C \rrbracket_V$  is a *DC* formula with occurrences of the flexible function and relation symbols which correspond to the methods which have invocations in  $C$ . Let there be calls to method  $m$  in  $C$ . Let  $m$  return no value for the sake of simplicity. Let  $B$  be the body of  $m$ . Using just replacement of equivalents in *DC*, we can prove that

$$\llbracket \mathbf{export} \ m(p_1, \dots, p_n) \ B \rrbracket \wedge \llbracket C \rrbracket_V \Rightarrow \llbracket [B]_{\emptyset}/m \rrbracket \llbracket C \rrbracket_V,$$

where

$$\begin{aligned} \llbracket [B]_{\emptyset}/m \rrbracket m(e_1, \dots, e_n) &\Leftrightarrow [e_1/p_1, \dots, e_n/p_n] \llbracket B \rrbracket_{\emptyset}, \\ \llbracket [B]_{\emptyset}/m \rrbracket v' = m(e_1, \dots, e_n) &\Leftrightarrow [v'/\mathbf{r}', e_1/p_1, \dots, e_n/p_n] \llbracket B \rrbracket_{\emptyset}, \end{aligned}$$

and  $\llbracket [B]_{\emptyset}/m \rrbracket$  distributes over the boolean connectives, chop and quantifiers. Reasoning about requirements such as what can be part of a contract can be done as follows. Assume that the satisfaction of a requirement  $Req_C$  by  $C$  is expressed as the validity of

$$\llbracket C \rrbracket_V \Rightarrow Req_C$$

and, according to a contract, method  $m$  is supposed to satisfy a certain requirement  $Req_m$ , that is, the formula

$$\llbracket B \rrbracket_{\emptyset} \Rightarrow Req_m$$

is valid for every acceptable  $B$ . Then the formula

$$\llbracket [Req_m/m] \rrbracket \llbracket C \rrbracket_V \Rightarrow Req_C$$

states that  $C$  would produce a behaviour satisfying  $Req_C$  for every implementation of  $m$  which satisfies  $Req_M$ .

Reasoning about the probability for code that makes calls to imported methods to terminate within a certain deadline can be done in this setting too. Let  $C$  and  $m$  be as above. Then the probability for  $C$  to terminate within  $d$  time units can be expressed as the *PDC* term

$$p(\llbracket C \rrbracket_V \wedge \int \neg N \leq d; \top).$$

Here, according to our assumption that computation time is negligible, we use  $\int \neg N$  and not  $\ell$  to measure execution time. This means that only time spent on the execution of **delay**, and time consumed by other processes is taken in account.

Now let  $F_m$  be a rigid function symbol which denotes a lower bound for the probability for  $m$  to terminate within some given time. Let  $P_m$  be the precondition for the successful execution of  $m$ . Then

$$\forall x(\ell = 0 \Rightarrow p(P_m(p_1, \dots, p_n) \wedge m(p_1, \dots, p_n) \wedge \int \neg N > x; \top) < 1 - F_m(x)) \vdash_{PITL} \\ p(\llbracket C \rrbracket_V \wedge \int \neg N \leq d; \top) \geq c$$

means that the probability for  $C$  to terminate within  $d$  time units is at least  $c$ , provided that  $m$  is always run with its precondition satisfied and the probability for  $m$  to terminate within  $x$  time units is bounded from below by  $F_m(x)$  for all  $x$ . The correspondence between the assumption on the execution time of  $m$  and the execution time of  $C$  can be expressed even more accurately, if we use a variable  $y$  instead of  $d$  and an expression  $F_C$  in terms of  $y$  and  $F_m$  instead of  $c$ :

$$\forall x(\ell = 0 \Rightarrow p(P_m(p_1, \dots, p_n) \wedge m(p_1, \dots, p_n) \wedge \int \neg N > x; \top) < 1 - F_m(x)) \vdash_{PITL} \\ p(\llbracket C \rrbracket_V \wedge \int \neg N > y; \top) < 1 - F_C(y, F_m).$$

Deriving  $F_C$ , which takes  $F_m$  as a parameter and is therefore a mapping from distributions to distributions, can be very difficult, but can turn feasible in some practically relevant cases. If the general form of  $F_m$  is known up to certain numerical parameters such as its *mean* and *variance*, and the pattern of calls to  $m$  in  $C$  is simple enough (e.g., a fixed number of successive calls), then  $F_C$  can be defined as a mapping from the domains of these numerical parameters instead of a space of distributions. The particular techniques for calculating  $F_C$  are a topic of classical probability theory.

**Example 1** Consider downloading e-mail, which consists of establishing a connection with the provider's server and then doing the actual download. Let the code  $C$  for downloading e-mail contain calls to two imported methods, *connect()* and *getMail()*:

```
var ok; call ok := connect(); if ok then (call getMail()); stop else stop
```

Let  $F_{connect}(t)$  be the probability for connecting within time  $t$ . Let the amount of e-mail to be downloaded be probabilistically distributed too; let the probability for the e-mail to be downloaded in time  $t$  be  $F_{getMail}(t)$ . For instance, if a dial-up connection is used, then it may take several attempts to dial the server. If the probability to receive a busy signal is  $q$ , the delay between successive attempts is  $D$ , and at most  $n$  attempts are made, then  $F_{connect}(kD + t) = 1 - q^{k+1}$ , where  $t$  is some amount of time needed to complete a successful attempt, for  $k = 1, \dots, n$ . A broadband connection would lead to a different  $F_{connect}$ .

Then lower bounds  $F_C$  for the distribution of the execution time of  $C$  satisfy the formula:

$$\ell = 0 \Rightarrow p(\llbracket \mathbf{call} \text{ ok} := \text{connect}() \rrbracket \wedge \int \neg N > t; \top) < 1 - F_{connect}(t), \\ \ell = 0 \Rightarrow p(\text{ok} \wedge \llbracket \mathbf{call} \text{ getMail}() \rrbracket \wedge \int \neg N > t; \top) < 1 - F_{getMail}(t) \vdash_{PITL} \\ p(\llbracket C \rrbracket \wedge \int \neg N > t; \top) < 1 - F_C(t)$$

Since the time for connecting and the quantity of e-mail to download can be assumed independent,  $F_C(t)$  can be expressed by the convolution:

$$(2) \quad \int_{t=0}^y F_{connect}(y-t) dF_{getMail}(t).$$

The lower bound  $F_C$  for the distribution of the execution time of  $C$  can be derived in *PITL* only approximately, because *PITL* does not capture taking limits such as those involved in the definition of the integral occurring in (2). This corresponds to the established practice to use numerical approximations for distributions. Except for a some of distributions that are well known from classical probability theory, it is seldom possible to obtain an explicit form for cumulative probability functions that arise from real-life cases. Using contracts makes it natural to work with lower bounds and not exact probabilities. The latter may as well not exist. That is why we satisfied with able to obtain approximations of arbitrary precision. Let us show such approximations can be derived for (2) using the proof system of *PITL*. We want to find a sequence  $A_k$ ,  $k = 0, 1, \dots$ , of terms written using  $F_{connect}$ ,  $F_{getMail}$  and  $t$  such that

$$p(\llbracket C \rrbracket \wedge \int \neg N > t; \top) < 1 - A_k$$

for all  $k$  can be *derived in PITL* and,  $\lim_k A_k = F_C(t)$  can be immediately established using the definition of  $\int$ . However simple, the use of the definition of  $\int$  take us *outside PITL*. As the reader might expect, the part of the derivation which is carried out within *PITL* is a formalisation of a standard probability theory argument too. Let  $\varphi_{t_1}^{t_2}$  be an abbreviation for

$$(3) \quad \varphi \wedge \int \neg N > t_1 \wedge \int \neg N \leq t_2$$

The formulas

$$connect \Rightarrow \neg(connect; \ell \neq 0) \text{ and } getMail \Rightarrow \neg(getMail; \ell \neq 0)$$

express the natural assumption that every call to *connect* (*getMail*) can terminate at most once. Using these formulas as assumptions enables an application of the rule *Seq* to derive

$$\begin{aligned} \ell = 0 \wedge p(connect; getMail; \top) = 1 \Rightarrow \\ p(connect_{\frac{lt}{k}}^{(\frac{l+1}{k}t)}; getMail_{\frac{mt}{k}}^{(\frac{m+1}{k}t)}; \top) = (F_{connect}(\frac{(l+1)t}{k}) - F_{connect}(\frac{lt}{k}))(F_{getMail}(\frac{(m+1)t}{k}) - F_{getMail}(\frac{mt}{k})) \end{aligned}$$

for all  $l, m \in \{0, \dots, k-1\}$ . Now by a repeated use of  $P_+$ , and using that  $F_{connect}(0) = 0$ , we obtain

$$\begin{aligned} p((connect; getMail) \wedge \int \neg N \leq t; \top) &\leq \sum_{l+m \leq k-1} p(connect_{\frac{lt}{k}}^{(\frac{l+1}{k}t)}; getMail_{\frac{mt}{k}}^{(\frac{m+1}{k}t)}; \top) + \\ &\sum_{l+m=k} p(connect_{\frac{lt}{k}}^{(\frac{l+1}{k}t)}; getMail_{\frac{mt}{k}}^{(\frac{m+1}{k}t)}; \top) \\ &= \underbrace{\sum_{m \leq k-1} F_{connect}(\frac{(k-m+1)t}{k})(F_{getMail}(\frac{(m+1)t}{k}) - F_{getMail}(\frac{mt}{k})) + B_k}_{S_k} \end{aligned}$$

where  $B_k \leq \max_{l \leq k-1} F_{connect}(\frac{(l+1)t}{k}) - F_{connect}(\frac{lt}{k})$ , and therefore  $\lim_k B_k = 0$ . By the definition of Stieltjes integral, we have  $\lim_k S_k = F_C(t)$ . Hence we can take  $A_k$  to be the expression on the right of  $\leq$  above.

Note that the derivation for *Seq* given in Section A.5 is with  $\varphi_{t_1}^{t_2}$  standing for  $\varphi \wedge l \geq t_1 \wedge l \leq t_2$  and not (3). However, that derivation applies to (3) as the meaning of  $\varphi_{t_1}^{t_2}$  as well.

**Example 2** Consider attempting to download 5 files in quick succession from the web. A server would typically allow at most 4 files to be downloading simultaneously. If this takes long, the 5th request can

be rejected by the browser due to a timeout. We are interested in the probability for having to re-launch the 5th download in case of a timeout. Here follows an extremely simplified variant of the relevant code:

	<b>letrec</b> $X$ <b>where</b>	1
	$X$ : <b>if</b> $userRequest$ <b>then</b>	2
	$(userRequest := \mathbf{false};$	3
(4)	$X \parallel \left( \begin{array}{l} \mathbf{call} \text{ handle} := requestDownload(url, timeout); \\ \mathbf{if} \text{ handle}! = \mathbf{null} \\ \quad \mathbf{then} (\mathbf{call} \text{ download}(handle); \mathbf{stop}) \\ \quad \mathbf{else} (\mathbf{call} \text{ signalTimeout}(url); \mathbf{stop}) \end{array} \right)$	4 5 6 7
	$)$	8
	<b>else</b> $X$	9

Aa separate process is assumed to provide that the arrival of a new download request is indicated by the shared variable  $userRequest$  becoming true and that the URL is then placed in the shared variable  $url$ . Let

$$\alpha(R, T) \Leftrightarrow \left( \begin{array}{l} (\lceil \neg R \rceil; K^R(V) \wedge \neg userRequest)^*; \\ \lceil \neg R \rceil; K^R(V) \wedge userRequest; \\ \lceil \neg R \rceil; K^R(V \setminus \{userRequest\}) \wedge userRequest' = \mathbf{false} \end{array} \right) \wedge \int \neg N = T$$

and

$$\beta(R, W, T) \Leftrightarrow \left( \begin{array}{l} \lceil \neg R \rceil; K^R(V \setminus \{handle\}) \wedge handle' = requestDownload(url, timeout); \\ \lceil \neg R \rceil; K^R(V) \wedge handle! = \mathbf{null}; \\ \lceil \neg R \rceil; K^R(V) \wedge download(handle) \wedge \int \neg N = T; \\ \lceil W \rceil \end{array} \right)$$

According to the semantics of (4),  $\alpha(R, T)$  describes the behaviour which corresponds to the repeated execution of lines 2-3 and 9 until  $userRequest$  becomes true with  $T$  denoting the overall execution time. The formula  $\beta(R, W, T)$  corresponds to the execution of lines 4-6, with  $R$  and  $W$  describing the status of the thread which becomes created in order to complete the requested download, and  $T$  denoting the time taken to complete the download. The scenario of launching the five downloads involves six threads. Five are created to handle the five downloads, and the sixth keeps the system ready to accept further requests. Let  $R_1, \dots, R_6, W_1, \dots, W_6$  describe the status of the six threads. Then the scenario can be described by the formula

$$(5) \exists R_1 \dots \exists R_6 \exists W_1 \dots \exists W_6 \left( \lceil W \rceil \Leftrightarrow \bigwedge_{i=1}^6 W_i \wedge \lceil R \rceil \Leftrightarrow \bigvee_{i=1}^6 R_i \wedge \lceil \bigwedge_{1 \leq i < j \leq 6} \neg (R_i \wedge R_j) \rceil \wedge \bigwedge_{i=1}^6 T(R_i, W_i) \wedge \gamma \right)$$

where  $\gamma$  describes the concurrent execution of the six threads. To be able to fit  $\gamma$  on the page, we use further abbreviations. Let

$$\alpha_i(T) \Leftrightarrow \alpha(R_{i+1} \vee \dots \vee R_6, W_{i+1} \wedge \dots \wedge W_6, T) \text{ and } \beta_i(T) \Leftrightarrow \beta(R_i, W_i, T).$$

With these abbreviations  $\gamma$  can be written as

$$\left( \alpha(R, W, T_0); \left( \beta_1(D_1) \wedge \left( \alpha_1(T_1); \left( \beta_2(D_2) \wedge \left( \alpha_2(T_2); \left( \beta_3(D_3) \wedge \left( \alpha_3(T_3); \left( \beta_4(D_4) \wedge \left( \alpha_4(T_4); \xi \wedge \eta \right) \right) \right) \right) \right) \right) \right) \right) \right) \right)$$

Here  $T_i$  denotes the time between launching the  $i$ th and the  $i+1$ st download and  $D_i$  denotes the duration of the  $i$ th download,  $i = 1, \dots, 5$ . The formulas  $\xi$  and  $\eta$  denote

$$([\neg R_5]; \mathbb{K}^{R_5}(V \setminus \{handle\}) \wedge handle' = requestDownload(url, timeout) \wedge \int \neg N = x; \top)$$

and

$$([\neg R_6]; \mathbb{K}^{R_6}(V) \wedge \neg userRequest)^*; \top),$$

and correspond to the behaviour of the thread which is created for the 5th download the thread which keeps sampling for subsequent user requests after the 5th download request. The occurrences of  $\top$  in them mark future behaviour which is not specified in our scenario. The connection between (5) and the semantics of **letrec**, which involves the least-fixed-point operator  $\mu$ , can be established using the validity of the equivalence

$$\mu X.\varphi \Leftrightarrow [\mu X.\varphi/X]\varphi.$$

The 5th download will be aborted in case  $x$  exceeds *timeout*, which is equivalent to

$$timeout + \sum_{j=1}^4 T_j < \min_{i=1}^4 \left( D_i + \sum_{j=1}^{i-1} T_j \right).$$

This is valid under the assumption that the rate of downloading is the limiting factor for the working of the entire system, which entails that we can ignore the small amounts of time taken for dialog, computation and the execution time of *requestDownload* for the first four downloads. Let  $F(l, t)$  be a lower bound for the probability for *download* do complete a download of length  $l$  within time  $t$ . It can be assumed that  $F(al, at) = F(l, t)$  for all  $a > 0$  and that  $F(l, t) = 0$  in case  $\frac{l}{t}$  exceeds the top rate of transmission  $v$ . Let  $l_i$  be the length of the  $i$ th download,  $i = 1, \dots, 5$ . Let  $l_i > v(T_1 + T_2 + T_3 + T_4)$  for  $i = 1, \dots, 4$ , that is, none of the downloads can be completed before all of them have been launched, for the sake of simplicity. Then the probability  $P_i$  for  $i \in \{1, \dots, 4\}$  to be the first download to complete, and to complete before the timeout for the pending 5th download is at least

$$\int_{A_i} F'(l_i - q_i, timeout) \cdot \prod_{k=1}^4 F'(q_k, \sum_{s=k}^4 T_s) dq_1 \dots dq_4,$$

where  $F'(q, t) = \frac{\partial}{\partial q} F(q, t)$  and  $A_i = \{\langle q_1, \dots, q_4 \rangle : l_i - q_i \leq l_j - q_j, j = 1, \dots, 4\}$ . The probability for the 5th download not to be cancelled is  $P_1 + \dots + P_4$ .

Note that using a contract in which the execution time of *download* is approximated by a distribution which depends just the amount of data to transmit is too crude. A more accurate calculation would be possible only if the amount of competing traffic throughout the download is taken in account. The form of probabilistic timed contract that we propose in this paper does not allow this. We give the calculations which correspond to this more accurate assumption for the sake of completeness. The main difference is that, instead of assuming that the progress of each download is described independently by  $F(q, t)$ ,  $F(q, t)$  is a lower bound for the probability to that at  $q$  bytes are received within time  $t$  for all the ongoing downloads together. The calculations below apply if the incoming data is distributed evenly among all downloads. Then the probability for an individual download to receive  $r$  bytes within a period of length  $t$  while competing with  $k - 1$  more downloads is  $F(kr, t)$ , and

$$P_i = \int_{B_i} F' \left( 4 \left( l_i - \sum_{k=i}^4 r_k \right), \text{timeout} \right) \cdot \prod_{k=1}^4 F'(kr_k, T_k) dq_1 \dots dq_4,$$

where

$$B_i = \left\{ \langle r_1, \dots, r_4 \rangle : l_i - l_j \leq \sum_{k=i}^{j-1} r_k, j = i + 1, \dots, 4; l_j - l_i \geq \sum_{k=j}^{i-1} r_k, j = 1, \dots, i - 1 \right\}.$$

Approximations of the above integrals can be derived in *PITL* using *Seq* much like in Example 1.

## 4 Probabilistic timed designs

### 4.1 Definition

A design  $\langle P, R \rangle$ , usually written as  $P \vdash R$ , describes the working of a method in terms of a *precondition*  $P$ , an *input-output relation*  $R$ .  $P$  is a predicate on the initial values  $v$  of the variables of the method, and  $R$  is a relation between the initial values  $v$  and the final values  $v'$  of the variables, which is guaranteed to hold in case the method is run with  $v$  initially satisfying  $P$ . A *probabilistic timed design*  $\langle P, R, F \rangle$  augments this description with an *execution time distribution*  $F$ .  $F$  is a function which takes the initial values  $v$  of the variables of the method and one more parameter  $t$ , which denotes execution time.  $F(v, t)$  is a lower bound for the probability for the method to terminate within time  $t$ , provided that  $P(v)$  holds. A *hard* bound  $d$  on execution time can be described using  $F$  which satisfy  $F(d') = 1$  for all  $d' \geq d$ .

### 4.2 Describing designs in *PDC*

Using our convention on variables, the property of method  $m$  encoded by  $\langle P, R, F \rangle$  can be written as the *PITL* formulas

$$m \Rightarrow (P(v) \Rightarrow R(v, v')) \text{ and } \ell = 0 \Rightarrow p(P(v) \wedge m \wedge \ell > t; \top) < 1 - F(v, t)$$

The first formula describes the *functional* behaviour of  $m$ . The second formula states that if  $P(v)$  holds, then the probability for a run of  $m$  to take more than  $t$  time units is smaller than  $1 - F(v, t)$ . Note that  $F$  is a lower bound and not an *exact* probability, which, in general, need not exist.

### 4.3 Refinement of probabilistic timed designs

Design  $D_1 = \langle P_1, R_1, F_1 \rangle$  *refines* design  $D_2 = \langle P_2, R_2, F_2 \rangle$ , written  $D_1 \sqsubseteq D_2$ , if

$$\forall x(P_2(x) \Rightarrow P_1(x)), \forall x \forall x'(R_1(x, x') \Rightarrow R_2(x, x')), \text{ and } \forall x \forall t(F_2(x, t) \leq F_1(x, t)).$$

This means that  $D_1$  has a *weaker or equivalent* precondition and a *stronger or equivalent* input-output relation, and on average terminates *at least as fast* as  $D_2$ . Clearly, if  $D_1 \sqsubseteq D_2$ , and  $D_1$  and  $D_2$  are both designs for the same method  $m$ , then the validity of

$$P_1(v) \wedge m \Rightarrow R_1(v, v') \text{ and } \forall x(\ell = 0 \Rightarrow p(P_1(v) \wedge m \wedge \ell > x; \top) < 1 - F_1(v, x))$$

entails the validity of

$$P_2(v) \wedge m \Rightarrow R_2(v, v') \text{ and } \forall x(\ell = 0 \Rightarrow p(P_2(v) \wedge m \wedge \ell > x; \top) < 1 - F_2(v, x)).$$

## 5 Probabilistic timed contracts

### 5.1 Definition

The execution time of a method depends on the execution times of the methods which have calls in its body.

**Definition 3 (component declaration)** A *component declaration* is a pair  $\langle M_i, M_e \rangle$  where  $M_i$  and  $M_e$  are disjoint sets of declarations for *imported* and *exported* methods, respectively.

**Definition 4 (probabilistic timed contract)** Let  $\langle M_i, M_e \rangle$  be a component declaration and  $V_m$  be the set of the valuations for the variables of declaration  $m$ ,  $m \in M_i \cup M_e$ . The tuple  $C = \langle D_m : m \in M_i \cup M_e \rangle$  is a *contract* for  $\langle M_i, M_e \rangle$ , if  $D_m$  are of the form  $\langle P_m, R_m, F_m \rangle$  where

- (i)  $\langle P_m, R_m \rangle$  is a (non-probabilistic) design for  $m$ ,  $m \in M_i \cup M_e$ .
- (ii)  $F_m$  is a variable of type  $V_m \times \mathbf{R}_+ \rightarrow [0, 1]$  for method declarations  $m \in M_i$  and is meant to denote a distribution of the execution time of implementations of  $m$ .
- (iii) For declarations  $m \in M_e$ ,  $F_m$  an expression for the distribution of the execution time of an implementation of declaration  $m$  as in probabilistic designs in terms of  $F_n$ ,  $n \in M_i$ .

We denote the set  $\{n \in M_i : F_n \text{ occurs in } F_m\}$  by  $C_{i,m}$ . Semantically, if  $m \in M_e$ , then  $F_m$  denotes a mapping of type

$$\left( \prod_{n \in C_{i,m}} V_n \times \mathbf{R}_+ \rightarrow [0, 1] \right) \rightarrow (V_m \times \mathbf{R}_+ \rightarrow [0, 1]).$$

Syntactically we assume that  $F_m$  is an expression such as, e.g., (2) for  $m$ . A contract  $C$  is meant to express that if the methods declared by  $m \in M_i$  satisfy their corresponding designs  $D_m$  and the distribution variables  $F_m$  are assigned lower bounds for the distributions of their execution times, then the methods declared in  $M_e$  satisfy their corresponding designs and the expressions  $F_m$  evaluate to lower bounds for the distributions of their execution times. If  $C_{i,m} = 0$  then,  $F_m$  simplifies to a distribution for execution time and  $\langle P_m, R_m, F_m \rangle$  is essentially a probabilistic timed design.

**Definition 5 (refinement of probabilistic timed contracts)** Let  $C$  and  $C'$  be probabilistic timed contracts for  $\langle M_i, M_e \rangle$  and  $\langle M'_i, M'_e \rangle$ , respectively. Let  $C = \langle \langle P_m, R_m, F_m \rangle : m \in M_i \cup M_e \rangle$  and  $C' = \langle \langle P'_m, R'_m, F'_m \rangle : m \in M'_i \cup M'_e \rangle$ . Then,  $C'$  refines  $C$ , written  $C' \sqsubseteq C$ , if

- (i)  $M'_i \subseteq M_i$ ,  $M'_e \supseteq M_e$ ;
- (ii)  $\langle P_m, R_m \rangle \sqsubseteq \langle P'_m, R'_m \rangle$  for  $m \in M'_i$ ,  $\langle P'_m, R'_m \rangle \sqsubseteq \langle P_m, R_m \rangle$  for  $m \in M_e$ ;
- (iii)  $F_m(v, t) \leq F'_m(v, t)$  for  $m \in M_e$ ,  $v \in V_m$ ,  $t \in \mathbf{R}_+$  regardless of the values of  $F_n$ ,  $n \in M_i$ .

## 5.2 Composing probabilistic timed contracts

Let  $A^k = \langle M_i^k, M_e^k \rangle$  and  $C^k = \langle \langle P_m^k, R_m^k, F_m^k \rangle : m \in M_i^k \cup M_e^k \rangle$ ,  $k = 1, 2$ , be two component declarations and probabilistic timed contracts for them, respectively.  $A^1$  and  $A^2$  are composable, if  $M_e^1 \cap M_e^2 = \emptyset$ .  $C^1$  and  $C^2$  are composable, if  $A^1$  and  $A^2$  are composable, and  $D_m^k \sqsubseteq D_m^{2-k}$  for  $m \in M_e^k \cap M_i^{2-k}$ ,  $k = 1, 2$ . The composition of  $C^1$  and  $C^2$ , written  $C^1 \cup C^2$ , is  $\langle \langle P_m, R_m, F_m \rangle : m \in M_i^1 \cup M_e^1 \cup M_i^2 \cup M_e^2 \rangle$  where:

- (i)  $P_m(v) \Leftrightarrow P_m^1(v) \wedge P_m^2(v)$ ,  $R_m(v, v') \Leftrightarrow R_m^1(v, v') \wedge R_m^2(v, v')$  and  $F_m = F_m^1 = F_m^2$  for  $m \in M_i^1 \cap M_i^2$ ;
- (ii)  $P_m = P_m^k$  and  $R_m = R_m^k$  for  $m \in M_e^k \cup (M_i^k \setminus M_i^{2-k})$ ,  $k = 1, 2$ ;
- (iii)  $F_m = F_m^k$  for  $m \in M_i^k \setminus M_i^{2-k}$ ,  $k = 1, 2$ .

To facilitate the understanding, we first define  $F_m$ ,  $m \in M_e^1 \cup M_e^2$ , in case  $C^1$  and  $C^2$  allow no *circular dependency* between the methods, that is, if there is no sequence  $m_0, \dots, m_{2s-1}$  such that  $m_r \in M_e^1 \cap M_i^2$  for  $r = 1, 3, \dots, 2s-1$ ,  $m_r \in M_e^2 \cap M_i^1$  for  $r = 0, 2, \dots, 2s-2$  and  $m_r \in C_{i, m_{r+1 \bmod 2s}}^{2-r \bmod 2}$ ,  $r = 0, \dots, 2s-1$ . Given that there is no circular dependency, we can define *dependency depth* of  $m$  from  $C^1 \cup C^2$  as the length  $s$  of the longest sequence of the form  $m_1, \dots, m_s$  such that  $m_1 \in C_{i, m}$  and  $m_{r+1} \in C_{i, m_r}^k$ , where  $k$  is such that  $m_r \in M_e^k$ , for  $r = 1, \dots, s-1$ , and we can define  $F_m$  by induction on the dependency depth of  $m$  by the clauses:

$$F_m = F_m^k \text{ for } m \in M_e^k \text{ of dependency depth } 0;$$

$$F_m = [F_n / F_n^k : n \in C_{i, m}^k] F_m^k \text{ for } m \in M_e^k \text{ of nonzero dependency depth.}$$

Note that the substitution replaces  $F_n^k$  with the expression for it from  $C^{2-k}$ , in case  $n \in M_e^{2-k}$ . Otherwise  $F_n^k$  is not affected by this substitution.

Now assume that there may be circular dependencies between  $C^1$  and  $C^2$ . Then the  $F_m$ s for the exported methods in  $C^1 \cup C^2$  should be a solution of the system of equations

$$X_m = [X_n/F_n^k : n \in C_{i,m}]F_m^k.$$

It is difficult to develop a method that would produce such a solution without restrictions on  $F_m^k$ . Standard techniques allow this to be done for monotonic  $F_m^k$ . In practice  $F_m^k$  can be non-monotonic: increasing the execution time of an imported method may as well shorten the execution time of the code which calls it, as this code may be designed to abort calls to the imported method which exceed a deadline, and cancel subsequent activities, thus achieving faster though abnormal termination.

If all the considered  $F_m^k$  are monotonic, then  $X_m$  can be obtained as the limit of the sequences  $X_m^j$ ,  $j < \omega$ , where  $X_m^0$  is the constant 0 and

$$X_m^{j+1} = [X_n^j/F_n^k : n \in C_{i,m}^k]F_m^k \text{ for } m \in M_e^k.$$

Observe that the initial approximation  $X_m^0 = 0$  for  $X_m$  corresponds to the initial assumption that the probability for  $m$  to terminate within any given time is at least 0. This means that  $X_m^1$  would give non-zero probability for termination only to runs of  $m$  which do not involve calls to other imported methods;  $X_m^2$  would give non-zero probability just for runs involving calls to imported methods which themselves lead to no further calls and so on. Since  $F_m^k$  are meant to be *under*-approximations, and the monotonicity of  $F_m^k$  entails that  $X_m^s \leq X_m^{s+1} \leq \lim_j X_m^j$  for all  $s < \omega$ ,  $X_m^s$  can be used as  $F_m$  instead of  $\lim_j X_m^j$  for sufficiently large  $s$ . Such an approximation would be more crude, yet possibly computationally less expensive.

## Concluding remarks

In this paper we focus just on soft requirements on execution time for the sake of simplicity. We believe that our approach can be adapted to reasoning about other types of QoS soft requirements which can be expressed using probability. The notion of quality of service originated from telecommunications. We use examples that come from everyday experience with the Internet and require no special technical expertise to understand. However, we believe that our approach would work just as well for problems from other areas such as embedded systems.

## Acknowledgement

The authors are grateful to Zhan Naijun for spotting some mistakes in an earlier version of this report.

## References

- [BJPW99] Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, and Damien Watkins. Making Components Contract Aware. *Computer*, 32(7):38–45, 1999.
- [CHD<sup>+</sup>07] Chen Zhenbang, Abdel Hakim Hannousse, Dang Van Hung, Istvan Knoll, Li Xiaoshan, Liu Zhiming, Liu Yang, An Qu, Joseph C. Okika, Anders P. Ravn, Volker Stolz, Yang Lu, and Zhan Naijun. Modelling with Relational Calculus of Object and Component Systems - rCOS. Research Report 382, UNU/IIST, P.O.Box 3058, Macau, September 2007.
- [CMZ] Antonio Cau, Ben Moszkowski, and Hussein Zedan. Interval Temporal Logic. URL: <http://www.cms.dmu.ac.uk/~cau/itlhomepage/itlhomepage.html>.
- [Dan98] Dang Van Hung. Modelling and Verification of Biphase Mark Protocols in Duration Calculus Using PVS/DC<sup>-</sup>. In *Proceedings of the 1998 International Conference on Application of Concurrency to System Design (CSD'98)*, pages 88–98. IEEE Computer Society Press, March 1998.
- [Dan05] Dang Van Hung. Toward a formal model for component interfaces for real-time systems. In Tiziana Margaria and Mieke Massink, editors, *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 106 – 114. ACM Press, 2005.
- [DW96] Dang Van Hung and Wang Ji. On The Design of Hybrid Control Systems Using Automata Models. In *Proceedings of FST TCS 1996*, volume 1180 of *LNCS*, pages 156–167. Springer, 1996.
- [DZ99] Dang Van Hung and Zhou Chaochen. Probabilistic Duration Calculus for Continuous Time. *Formal Aspects of Computing*, 11(1):21–44, 1999.
- [GD02] Dimitar P. Guelev and Dang Van Hung. Prefix and Projection onto State in Duration Calculus. In *Proceedings of TPTS'02*, volume 65(6) of *ENTCS*. Elsevier Science, 2002.
- [Gue00a] Dimitar P. Guelev. A Complete Fragment of Higher-order Duration  $\mu$ -calculus. In *Proceedings of FST TCS 2000*, volume 1974 of *LNCS*, pages 264–276. Springer, 2000.
- [Gue00b] Dimitar P. Guelev. Probabilistic Neighbourhood Logic. In Mathai Joseph, editor, *Proceedings of FTRTFT 2000*, volume 1926 of *LNCS*, pages 264–275. Springer, 2000. A proof-complete version is available as UNU/IIST Technical Report 196 from <http://www.iist.unu.edu>.
- [Gue07] Dimitar P. Guelev. Probabilistic Interval Temporal Logic and Duration Calculus with Infinite Intervals: Complete Proof Systems. *Logical Methods in Computer Science*, 3(3), 2007. URL: <http://www.lmcs-online.org/>.
- [HD07] Hung Ledang and Dang Van Hung. Concurrency and Schedulability Analysis in Component-based Real-Time System Development. In *Proceedings of the 1st IEEE & IFIP International Symposium on Theoretical Aspects of Software Engineering*. IEEE Computer Society Press, 2007.
- [HH98] C.A.R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall, 1998.
- [HLL06] He Jifeng, Li Xiaoshan, and Liu Zhiming. A Theory of Reactive Components. In Liu Zhiming and Luis Barbosa, editors, *Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005)*, volume 160 of *ENTCS*, pages 173–195. Elsevier, 2006.
- [HXZ06] He Jifeng, Xiaoshan Li, and Zhiming Liu. A refinement calculus of object systems. *Theoretical Computer Science*, 365(1-2):109–142, 2006.

- [HZ92] Michael R. Hansen and Zhou Chaochen. Semantics and Completeness of Duration Calculus. In *Real-Time: Theory and Practice*, volume 600 of *LNCS*, pages 209–225. Springer, 1992.
- [LH99] Li Li and He Jifeng. A Denotational Semantics of Timed RSL using Duration Calculus. In *Proceedings of RTCSA'99*, pages 492–503. IEEE Computer Society Press, 1999.
- [LRSZ93] Liu Zhiming, A. P. Ravn, E. V. Sørensen, and Zhou Chaochen. A Probabilistic Duration Calculus. In H. Kopetz and Y. Kakuda, editors, *Dependable Computing and Fault-tolerant Systems Vol. 7: Responsive Computer Systems*, pages 30–52. Springer, 1993.
- [Mos86] Ben Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, 1986. URL: <http://www.cse.dmu.ac.uk/cau/papers/tempura-book.pdf>.
- [Pan95] Paritosh K. Pandya. Some extensions to Mean-Value Calculus: Expressiveness and Decidability. In *Proceedings of CSL'95*, volume 1092 of *LNCS*, pages 434–451. Springer, 1995.
- [PD98] Paritosh K. Pandya and Dang Van Hung. Duration Calculus of Weakly Monotonic Time. In *Proceedings of FTRTFT'98*, volume 1486 of *LNCS*, pages 55–64. Springer, 1998.
- [PWX98] Paritosh K. Pandya, Wang Hanping, and Xu Qiwen. Towards a Theory of Sequential Hybrid Programs. In D. Gries and W.-P. de Roever, editors, *Proceedings of IFIP Working Conference PROCOMET'98*, pages 336–384. Chapman & Hall, 1998.
- [SX98] Gerardo Schneider and Xu Qiwen. Towards a Formal Semantics of Verilog Using Duration Calculus. In Anders P. Ravn and Hans Rischel, editors, *Proceedings of FTRTFT'98*, volume 1486 of *LNCS*, pages 282–293. Springer, 1998.
- [WX04] Wang Hanpin and Xu Qiwen. Completeness of Temporal Logics over Infinite Intervals. *Discrete Applied Mathematics*, 136(1):87–103, 2004.
- [ZDL95] Zhou Chaochen, Dang Van Hung, and Li Xiaoshan. A Duration Calculus with Infinite Intervals. In Horst Reichel, editor, *Fundamentals of Computation Theory*, volume 965 of *LNCS*, pages 16–41. Springer, 1995.
- [ZGZ00] Zhou Chaochen, Dimitar P. Guelev, and Zhan Naijun. A Higher-order Duration Calculus. In *Millennial Perspectives in Computer Science*, pages 407–416. Palgrave, 2000.
- [ZZ94] Zheng Yuhua and Zhou Chaochen. A Formal Proof of a Deadline Driven Scheduler. In *Proceedings of FTRTFT'94*, volume 863 of *LNCS*, pages 756–775. Springer, 1994.

## A Proof systems

### A.1 Proof system for *ITL* with infinite intervals

The following axioms and rules have been shown to form a complete proof system for *ITL* with infinite intervals when added to a Hilbert-style proof system for classical first-order predicate logic and appropriate axioms about an abstract domain of durations in [WX04]:

$$\begin{array}{ll}
(A1) & (\varphi; \psi) \wedge \neg(\chi; \psi) \Rightarrow (\varphi \wedge \neg\chi; \psi), (\varphi; \psi) \wedge \neg(\varphi; \chi) \Rightarrow (\varphi; \psi \wedge \neg\chi) \\
(A2) & ((\varphi; \psi); \chi) \Leftrightarrow (\varphi; (\psi; \chi)) \\
(R) & (\varphi; \psi) \Rightarrow \varphi, (\psi; \varphi) \Rightarrow \varphi \text{ if } \varphi \text{ is rigid} \\
(B) & (\exists x\varphi; \psi) \Rightarrow \exists x(\varphi; \psi), (\psi; \exists x\varphi) \Rightarrow \exists x(\psi; \varphi) \text{ if } x \notin FV(\psi) \\
(L1) & (\ell = x; \varphi) \Rightarrow \neg(\ell = x; \neg\varphi), (\varphi; \ell = x \wedge x \neq \infty) \Rightarrow \neg(\neg\varphi; \ell = x) \\
(L2) & \ell = x + y \wedge x \neq \infty \Leftrightarrow (\ell = x; \ell = y) \\
(L3) & \varphi \Rightarrow (\ell = 0; \varphi), \varphi \wedge \ell \neq \infty \Rightarrow (\varphi; \ell = 0) \\
(S1) & (\ell = x \wedge \varphi; \psi) \Rightarrow \neg(\ell = x \wedge \neg\varphi; \chi) \\
(P1) & \neg(\ell = \infty; \varphi) \\
(P2) & (\varphi; \ell = \infty) \Rightarrow \ell = \infty \\
(P3) & (\varphi; \ell \neq \infty) \Rightarrow \ell \neq \infty \\
\\ 
(N) & \frac{\varphi}{\neg(\neg\varphi; \psi)}, \frac{\varphi}{\neg(\psi; \neg\varphi)} \\
\\ 
(Mono) & \frac{\varphi \Rightarrow \psi}{(\varphi; \chi) \Rightarrow (\psi; \chi)}, \frac{\varphi \Rightarrow \psi}{(\chi; \varphi) \Rightarrow (\chi; \psi)}
\end{array}$$

Using the first order logic axiom

$$(\exists_r) [t/x]\varphi \Rightarrow \exists x\varphi.$$

is correct only if no variable in  $t$  becomes bound due to the substitution, and either  $t$  is rigid or  $(.; \cdot)$  does not occur in  $\varphi$ .

## A.2 Axioms and rules for $DC$ with infinite intervals

The axioms and rules below were proposed for  $DC$  with *finite* intervals and have been shown to be complete relative to validity in real-time  $ITL$  in [HZ92].

$$\begin{array}{ll}
(DC1) & \int \mathbf{0} = 0 \\
(DC2) & \int \mathbf{1} = \ell \\
(DC3) & \int S \geq 0 \\
(DC4) & \int S_1 + \int S_2 = \int (S_1 \vee S_2) + \int (S_1 \wedge S_2) \\
(DC5) & (\int S = x; \int S = y) \Rightarrow \int S = x + y \\
(DC6) & \int S_1 = \int S_2 \text{ if } S_1 \text{ and } S_2 \text{ are propositionally equivalent} \\
(IR1) & \frac{[\ell = 0/A]\varphi \Rightarrow [A \vee (A; [\mathbb{S}] \vee [\neg\mathbb{S}])/A]\varphi}{[\top/A]\varphi} \\
(IR2) & \frac{[\ell = 0/A]\varphi \Rightarrow [A \vee ([\mathbb{S}] \vee [\neg\mathbb{S}]; A)/A]\varphi}{[\top/A]\varphi}
\end{array}$$

The completeness proof from [HZ92] involves two theorems which can be derived using the rules  $IR1$  and  $IR2$ , instead of the rules themselves. The second of these theorems does not hold for infinite intervals and therefore we modify it appropriately:

$$\begin{array}{ll}
(T1) & \ell = 0 \vee ([\mathbb{S}]; \top) \vee ([\neg\mathbb{S}]; \top) \\
(T2) & \ell = 0 \vee \ell = \infty \vee (\top; [\mathbb{S}]) \vee (\top; [\neg\mathbb{S}])
\end{array}$$

$DC1$ - $DC6$ ,  $T1$  and the infinite-interval variant of  $T2$  form a relatively complete proof system for  $DC$  with infinite intervals.

### A.3 Proof system for $PITL$

$PITL$  is a conservative extension of  $ITL$ . Adding the axioms and a proof rule below to the proof system for  $ITL$  leads to a system which is complete for  $PITL$  with respect to a generalisation of the  $\mathbf{R}$ -based semantics, where  $\mathbf{R}$  is replaced by an abstract domain and the probability measures are required to be only finitely additive.

*Extensionality*

$$\begin{aligned} (P_;) & \quad (\ell = x; p(\psi) = y) \Rightarrow p((\ell = x; \psi)) = y \\ (P_\infty) & \quad \ell = \infty \Rightarrow (\varphi \Leftrightarrow p(\varphi) = 1) \\ (P_{\leq}) & \quad \frac{\vdash (\varphi; \ell = \infty) \Rightarrow (\psi \Rightarrow \chi)}{\vdash \varphi \wedge \ell < \infty \Rightarrow p(\psi) \leq p(\chi)} \end{aligned}$$

*Aritmetics of probabilities*

$$\begin{aligned} (P_\perp) & \quad p(\perp) = 0 \\ (P_\top) & \quad p(\top) = 1 \\ (P_+) & \quad p(\varphi) + p(\psi) = p(\varphi \vee \psi) + p(\varphi \wedge \psi) \end{aligned}$$

### A.4 Useful theorems and derived rules for $PITL$

All the theorems and rules below except  $P'_;$  are valid in general  $PITL$  models.  $P'_;$  is valid in  $PITL$  models with global probability.

$$\begin{aligned} (P_{\leq}^\infty) & \quad \frac{(\varphi; \ell = \infty) \vee (\varphi \wedge \ell = \infty) \Rightarrow (\psi \Rightarrow \chi)}{\varphi \Rightarrow p(\psi) \leq p(\chi)} \\ (PITL1) & \quad \frac{\varphi \Rightarrow \psi}{p(\varphi) \leq p(\psi)}, \quad \frac{\varphi \Leftrightarrow \psi}{p(\varphi) = p(\psi)} \\ (PITL2) & \quad p(\varphi) + p(\neg\varphi) = 1 \\ (PITL3) & \quad p(\varphi) < p(\psi) \Rightarrow p(\psi \wedge \neg\varphi) \neq 0 \\ (PITL4) & \quad p(\varphi) = p(\varphi \wedge \ell = \infty) \\ (PITL5) & \quad p(\varphi) \leq 1 \\ (PITL6) & \quad \frac{\varphi}{p(\varphi) = 1}, \quad \frac{\neg\varphi}{p(\varphi) = 0} \\ (PITL7) & \quad (\varphi; \top) \Rightarrow p(\varphi; \top) = 1 \\ (PITL8) & \quad p(\varphi) = 1 \wedge p(\psi) = x \Rightarrow p(\varphi \wedge \psi) = x \\ (PITL9) & \quad p(\varphi \Rightarrow \psi) = 1 \Rightarrow (p(\varphi) = 1 \Rightarrow p(\psi) = 1) \\ & \quad p(\varphi \Rightarrow \psi) = 1 \Rightarrow (p(\psi) = 0 \Rightarrow p(\varphi) = 0) \\ (PITL10) & \quad p(\varphi) + p(\psi) > 1 \Rightarrow p(\varphi \wedge \psi) > 0 \\ (P'_;) & \quad \frac{\varphi \Rightarrow \neg(\varphi; \ell \neq 0)}{(\varphi; p(\psi) = x) \Rightarrow p(\varphi; \psi) = x} \end{aligned}$$

Here follow the proofs of the above  $PITL$  theorems and derived rules. The purely  $ITL$  parts are skipped and marked “ $ITL$ ” for the sake of brevity.

$P_{\leq}^{\infty}$ :

- |   |   |   |
|---|---|---|
| 1 | $(\varphi; \ell = \infty) \Rightarrow (\psi \Rightarrow \chi)$  | assumption, <i>ITL</i>                  |
| 2 | $\varphi \wedge \ell < \infty \Rightarrow p(\psi) \leq p(\chi)$   | 1, $P_{\leq}$                           |
| 3 | $\ell = \infty \wedge \varphi \Rightarrow (p(\psi) = 0 \wedge p(\chi) = 0)$<br>$\vee (p(\psi) = 0 \wedge p(\chi) = 1)$<br>$\vee (p(\psi) = 1 \wedge p(\chi) = 1)$ | assumption, $P_{\infty}$ , <i>PITL2</i> |
| 4 | $\varphi \wedge \ell = \infty \Rightarrow p(\psi) \leq p(\chi)$   | 3, <i>ITL</i>                           |
| 5 | $\ell < \infty \vee \ell = \infty$  | <i>ITL</i>                              |
| 6 | $\varphi \Rightarrow p(\psi) \leq p(\chi)$  | 2, 4, 5                                 |

*PITL1*:

- |   |   |                        |
|---|---|------------------------|
| 1 | $\varphi \Rightarrow \psi$  | assumption             |
| 2 | $(\top; \ell = \infty) \vee (\top \wedge \ell = \infty) \Rightarrow (\varphi \Rightarrow \psi)$ | 1, <i>ITL</i>          |
| 3 | $p(\varphi) \leq p(\psi)$   | 2, $P_{\leq}^{\infty}$ |

The second rule *PITL1* is proved by two applications the first.

*PITL2*:

- |   |   |                     |
|---|---|---------------------|
| 1 | $\varphi \wedge \neg\varphi \Leftrightarrow \perp$  | <i>ITL</i>          |
| 2 | $p(\varphi \wedge \neg\varphi) = p(\perp)$  | 1, <i>PITL1</i>     |
| 3 | $p(\varphi \wedge \neg\varphi) = 0$   | 2, $P_{\perp}$      |
| 4 | $\varphi \vee \neg\varphi \Leftrightarrow \top$   | <i>ITL</i>          |
| 5 | $p(\varphi \vee \neg\varphi) = p(\top)$   | 4, <i>PITL1</i>     |
| 6 | $p(\varphi \wedge \neg\varphi) = 1$   | 5, $P_{\top}$       |
| 7 | $p(\varphi) + p(\neg\varphi) = p(\varphi \wedge \neg\varphi) + p(\varphi \wedge \neg\varphi)$ | $P_{+}$             |
| 8 | $p(\varphi) + p(\neg\varphi) = 1$   | 2, 6, 7, <i>ITL</i> |

*PITL3*:

- |   |   |                               |
|---|---|-------------------------------|
| 1 | $p(\psi) \leq p(\varphi \vee \psi)$   | $P_{\leq}^{\infty}$           |
| 2 | $p(\varphi) + p(\psi \wedge \neg\varphi) = p(\varphi \wedge \psi \wedge \neg\varphi) + p(\varphi \vee \psi \wedge \neg\varphi)$ | $P_{+}$                       |
| 3 | $p(\varphi) + p(\psi \wedge \neg\varphi) = p(\varphi \vee \psi)$  | 2, <i>PITL1</i> , $P_{\perp}$ |
| 4 | $p(\varphi) < p(\psi) \Rightarrow p(\varphi) < p(\varphi \vee \psi)$  | 1                             |
| 5 | $p(\varphi) < p(\psi) \Rightarrow p(\psi \wedge \neg\varphi) \neq 0$  | 3, 4                          |

*PITL4* is obtained by applying  $P_{\leq}^{\infty}$  to the *ITL* theorems

$$(\top; \ell = \infty) \vee (\top \wedge \ell = \infty) \Rightarrow (\varphi \Rightarrow \varphi \wedge \ell = \infty) \text{ and } (\top; \ell = \infty) \vee (\top \wedge \ell = \infty) \Rightarrow (\ell = \infty \wedge \varphi \Rightarrow \varphi).$$

*PITL5*:

- |   |                            |                 |
|---|----------------------------|-----------------|
| 1 | $\varphi \Rightarrow \top$ | <i>ITL</i>      |
| 2 | $p(\varphi) \leq p(\top)$  | 1, <i>PITL1</i> |
| 3 | $p(\varphi) \leq 1$        | 2, $P_{\top}$   |

*PITL6:*

- 1  $\top \Rightarrow \varphi$  assumption
- 2  $p(\top) \leq p(\varphi)$  1, *PITL1*
- 3  $1 \leq p(\varphi)$  2,  $P_{\top}$
- 4  $p(\varphi) \leq 1$  *PITL5*
- 5  $p(\varphi) = 1$  3, 4

- 1  $\neg\varphi$  assumption
- 2  $p(\neg\varphi) = 1$  1, *PITL6*
- 3  $p(\varphi) = 0$  *PITL2*

*PITL7:*

- 1  $(\varphi; \top; \ell = \infty) \vee ((\varphi; \top) \wedge \ell = \infty) \Rightarrow (\top \Rightarrow (\varphi; \top))$  *ITL*
- 2  $(\varphi; \top) \Rightarrow p(\varphi; \top) = 1$   $P_{\leq}^{\infty}$

*PITL8:*

- 1  $p(\varphi) = 1 \wedge p(\psi) = x \Rightarrow p(\varphi \wedge \psi) + p(\varphi \vee \psi) = 1 + x$   $P_{+}$
- 2  $\varphi \Rightarrow (\varphi \vee \psi)$  *ITL*
- 3  $p(\varphi) \leq p(\varphi \vee \psi)$  2, *PITL1*
- 4  $p(\varphi) = 1 \Rightarrow p(\varphi \vee \psi) = 1$  3, *PITL5*
- 5  $p(\varphi) = 1 \wedge p(\psi) = x \Rightarrow p(\varphi \wedge \psi) = x$  1, 4

*PITL9:*

- 1  $p(\varphi \Rightarrow \psi) = 1 \wedge p(\varphi) = 1 \Rightarrow p((\varphi \Rightarrow \psi) \wedge \psi) = 1$  *PITL8*
- 2  $(\varphi \Rightarrow \psi) \wedge \psi \Rightarrow \psi$
- 3  $p((\varphi \Rightarrow \psi) \wedge \psi) \leq p(\psi)$  2, *PITL1*
- 4  $p(\psi) \leq 1$  *PITL5*
- 5  $p(\varphi \Rightarrow \psi) = 1 \Rightarrow (p(\varphi) = 1 \Rightarrow p(\psi) = 1)$  1 – 4

- 1  $p((\varphi \Rightarrow \psi) \Rightarrow (\neg\psi \Rightarrow \neg\varphi)) = 1$  *PITL6*
- 2  $p(\varphi \Rightarrow \psi) = 1 \Rightarrow p(\neg\psi \Rightarrow \neg\varphi) = 1$  1, *PITL9*
- 3  $p(\neg\psi \Rightarrow \neg\varphi) = 1 \Rightarrow (p(\neg\psi) = 1 \Rightarrow p(\neg\varphi) = 1)$  *PITL9*
- 4  $p(\neg\psi \Rightarrow \neg\varphi) = 1 \Rightarrow (p(\neg\psi) = 0 \Rightarrow p(\neg\varphi) = 0)$  3, *PITL2*
- 5  $p(\varphi \Rightarrow \psi) = 1 \Rightarrow (p(\neg\psi) = 0 \Rightarrow p(\neg\varphi) = 0)$  2, 4

*PITL10:*

- 1  $p(\varphi) + p(\psi) > 1 \Rightarrow p(\varphi \wedge \psi) + p(\varphi \vee \psi) > 1$   $P_{+}$
- 2  $p(\varphi \vee \psi) \leq 1$  *PITL5*
- 3  $p(\varphi) + p(\psi) > 1 \Rightarrow p(\varphi \wedge \psi) > 0$  1, 2

$P'$ :

1	$(\varphi; p(\psi) = x) \Rightarrow \exists t((\varphi \wedge \ell = t; \top) \wedge (\ell = t; p(\psi) = x))$	<i>ITL</i>
2	$(\varphi \wedge \ell = t; \top) \Rightarrow p(\varphi \wedge \ell = t; \top) = 1$	<i>PITL7</i>
3	$(\ell = t; p(\psi) = x) \Rightarrow p(\ell = t; \psi) = x$	<i>P;</i>
4	$p(\varphi \wedge \ell = t; \top) = 1 \wedge p(\ell = t; \psi) = x \Rightarrow p(\varphi \wedge \ell = t; \psi) = x$	<i>PITL8, PITL1, ITL</i>
5	$(\varphi \wedge \ell = t; \psi) \Rightarrow (\varphi; \psi)$	<i>ITL</i>
6	$p(\varphi \wedge \ell = t; \psi) = x \Rightarrow p(\varphi; \psi) \geq x$	5, <i>PITL1</i>
7	$\exists t((\varphi \wedge \ell = t; \top) \wedge (\ell = t; p(\psi) = x)) \Rightarrow \exists t(p(\varphi; \psi) \geq x)$	2 – 6, <i>ITL</i>
8	$\exists t(p(\varphi; \psi) \geq x) \Leftrightarrow p(\varphi; \psi) \geq x$	<i>ITL</i>
9	$(\varphi; p(\psi) = x) \Rightarrow p(\varphi; \psi) \geq x$	1, 7, 8
10	$(\varphi; p(\psi) = x) \Leftrightarrow (\varphi; p(\neg\psi) = 1 - x)$	<i>PITL2, ITL</i>
11	$(\varphi; p(\neg\psi) = 1 - x) \Rightarrow p(\varphi; \neg\psi) \geq 1 - x$	like 1 – 9, but with $\neg\psi$ as $\psi$
12	$(p(\varphi; \psi) > x \wedge p(\varphi; \neg\psi) \geq 1 - x) \vee (p(\varphi; \psi) \geq x \wedge p(\varphi; \neg\psi) > 1 - x) \Rightarrow p((\varphi; \psi) \wedge (\varphi; \neg\psi)) > 0$	<i>PITL10</i>
13	$(\varphi; \psi) \wedge (\varphi; \neg\psi) \wedge \neg(\varphi \wedge (\varphi; \ell \neq 0); \top) \Rightarrow \perp$	<i>ITL</i>
14	$p((\varphi; \psi) \wedge (\varphi; \neg\psi) \wedge \neg(\varphi \wedge (\varphi; \ell \neq 0); \top)) = 0$	13, <i>PITL6</i>
15	$p(\neg(\varphi \wedge (\varphi; \ell \neq 0); \top)) = 1$	assumption, <i>PITL6</i>
16	$p((\varphi; \psi) \wedge (\varphi; \neg\psi)) = 0$	14, 15, <i>PITL8</i>
17	$p(\varphi; \neg\psi) \geq 1 - x \wedge p(\varphi; \psi) \geq x \Rightarrow p(\varphi; \psi) \leq x \wedge p(\varphi; \neg\psi) \leq 1 - x$	12, 16, <i>ITL</i>
18	$(\varphi; p(\psi) = x) \Rightarrow p(\varphi; \psi) = x$	9, 11, 17, <i>ITL</i>

## A.5 The rule *Seq*

In the proof of the admissibility of *Seq* below  $\varphi \wedge \ell \geq l \wedge \ell \leq h$  is abbreviated by  $\varphi_l^h$ .

1	$(\ell = 0 \wedge p(\alpha; \beta; \top) = 1; \top) \Rightarrow p(p(\alpha; \beta; \top) = 1 \wedge \ell = 0; \top) = 1$	<i>PITL7</i>
2	$(\alpha; p(\beta; \top) = x \wedge \ell = 0) \Rightarrow p(\alpha; \beta; \top) = x$	$P'_i$ , assumptions
3	$\exists x((\alpha; \top) \Rightarrow (\alpha; p(\beta; \top) = x \wedge \ell = 0; \top))$	<i>ITL</i>
4	$p(\alpha; \beta; \top) = 1 \Rightarrow$ $\exists x((\alpha; \top) \Rightarrow p(\alpha; \beta; \top) = x \wedge (\alpha; p(\beta; \top) = x \wedge \ell = 0; \top) \wedge p(\alpha; \beta; \top) = 1)$	2, 3, <i>ITL</i>
5	$p(\alpha; \beta; \top) = 1 \Rightarrow \exists x((\alpha; \top) \Rightarrow (\alpha; p(\beta; \top) = 1 \wedge \ell = 0; \top))$	4, <i>ITL</i>
6	$p(\alpha; \beta; \top) = 1 \wedge (\alpha; \top) \Rightarrow (\alpha; p(\beta; \top) = 1 \wedge \ell = 0; \top)$	5, <i>ITL</i>
7	$p(p(\alpha; \beta; \top) = 1 \wedge (\alpha; \top) \Rightarrow (\alpha; p(\beta; \top) = 1 \wedge \ell = 0; \top)) = 1$	6, <i>PITL6</i>
8	$p(p(\alpha; \beta; \top) = 1; \top) = 1 \wedge p(\alpha; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \wedge \ell = 0; \top) = 1$	7, <i>PITL9</i>
9	$\ell = 0 \wedge p(\alpha; \beta; \top) = 1 \wedge p(\alpha; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \wedge \ell = 0; \top) = 1$	1, 8
10	$p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; \top) = 1$	<i>PITL9</i>
11	$\ell = 0 \wedge p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \wedge \ell = 0; \top) = 1$	9, 10
12	$p(\alpha; p(\beta; \top) = 1 \wedge \ell = 0; \top) = 1 \wedge p(\alpha; p(\beta; \top) \neq 1 \wedge \ell = 0; \top) = x \Rightarrow$ $p((\alpha; p(\beta; \top) = 1 \wedge \ell = 0; \top) \wedge (\alpha; p(\beta; \top) \neq 1 \wedge \ell = 0; \top)) = x$	<i>PITL8</i>
13	$p((\alpha; p(\beta; \top) = 1 \wedge \ell = 0; \top) \wedge (\alpha; p(\beta; \top) \neq 1 \wedge \ell = 0; \top)) = 0$	<i>PITL6</i>
14	$p(\alpha; p(\beta; \top) = 1 \wedge \ell = 0; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) \neq 1 \wedge \ell = 0; \top) = 0$	12, 13
15	$\ell = 0 \wedge p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) \neq 1 \wedge \ell = 0; \top) = 0$	11, 14
16	$\ell = 0 \Rightarrow (p(\beta_{l_2}^{h_2}; \top) \neq x_2 \Rightarrow p(\beta; \top) \neq 1)$	$[ET_\beta \in [l_2, h_2]]_{x_2}$
17	$\alpha_{l_1}^{h_1} \Rightarrow \alpha$	<i>ITL</i>
18	$(\alpha_{l_1}^{h_1}; p(\beta_{l_2}^{h_2}; \top) \neq x_2 \wedge \ell = 0; \top) \Rightarrow (\alpha; p(\beta; \top) \neq 1 \wedge \ell = 0; \top)$	16, 17, <i>ITL</i>
19	$\ell = 0 \wedge p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha_{l_1}^{h_1}; p(\beta_{l_2}^{h_2}; \top) \neq x_2 \wedge \ell = 0; \top) = 0$	15, 18, <i>PITL6</i> , <i>PITL9</i>
20	$\neg(\alpha \wedge (\alpha; \ell \neq 0); \top) \Rightarrow$ $((\alpha_{l_1}^{h_1}; p(\beta_{l_2}^{h_2}; \top) \neq x_2 \wedge \ell = 0; \top) \Leftrightarrow (\alpha_{l_1}^{h_1}; \top) \wedge (\alpha; p(\beta_{l_2}^{h_2}; \top) \neq x_2 \wedge \ell = 0; \top))$	<i>ITL</i>
21	$p(\neg(\alpha \wedge (\alpha; \ell \neq 0); \top)) = 1$	assumption, <i>PITL6</i>
22	$p(\alpha_{l_1}^{h_1}; p(\beta_{l_2}^{h_2}; \top) \neq x_2 \wedge \ell = 0; \top) = 0 \Leftrightarrow$ $p((\alpha_{l_1}^{h_1}; \top) \wedge (\alpha; p(\beta_{l_2}^{h_2}; \top) \neq x_2 \wedge \ell = 0; \top)) = 0$	20, 21, <i>PITL9</i>
23	$\alpha \wedge p(\alpha; \beta_{l_2}^{h_2}; \top) \neq x_2 \Rightarrow \neg(\alpha; p(\beta_{l_2}^{h_2}; \top) = x_2 \wedge \ell = 0)$	$P'_i$
24	$\neg(\alpha; p(\beta_{l_2}^{h_2}; \top) = x_2) \wedge \alpha \Rightarrow (\alpha; p(\beta_{l_2}^{h_2}; \top) \neq x_2 \wedge \ell = 0)$	<i>ITL</i>
25	$\alpha \wedge p(\alpha; \beta_{l_2}^{h_2}; \top) \neq x_2 \Rightarrow (\alpha; p(\beta_{l_2}^{h_2}; \top) \neq x_2 \wedge \ell = 0)$	23, 24
26	$p((\alpha_{l_1}^{h_1}; \top) \wedge (\alpha; p(\beta_{l_2}^{h_2}; \top) \neq x_2 \wedge \ell = 0; \top)) = 0 \Rightarrow$ $p((\alpha_{l_1}^{h_1}; \top) \wedge (\alpha \wedge p(\alpha; \beta_{l_2}^{h_2}; \top) \neq x_2; \top)) = 0$	25, <i>PITL9</i> , <i>PITL6</i>
27	$\neg(\alpha \wedge (\alpha; \ell \neq 0); \top) \Rightarrow$ $((\alpha_{l_1}^{h_1}; \top) \wedge (\alpha \wedge p(\alpha; \beta_{l_2}^{h_2}; \top) \neq x_2; \top) \Leftrightarrow (\alpha_{l_1}^{h_1} \wedge p(\alpha; \beta_{l_2}^{h_2}; \top) \neq x_2; \top))$	<i>ITL</i>
28	$p((\alpha_{l_1}^{h_1}; \top) \wedge (\alpha \wedge p(\alpha; \beta_{l_2}^{h_2}; \top) \neq x_2; \top)) = 0 \Leftrightarrow p(\alpha_{l_1}^{h_1} \wedge p(\alpha; \beta_{l_2}^{h_2}; \top) \neq x_2; \top) = 0$	21, 27, <i>PITL9</i>
29	$\ell = 0 \wedge p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha_{l_1}^{h_1} \wedge p(\alpha; \beta_{l_2}^{h_2}; \top) \neq x_2; \top) = 0$	19, 22, 26, 28
30	$\ell = 0 \wedge p(\alpha_{l_1}^{h_1} \wedge p(\alpha; \beta_{l_2}^{h_2}; \top) \neq x_2; \top) = 0 \Rightarrow$ $p((\alpha_{l_1}^{h_1}; \top) \wedge (\alpha; \beta_{l_2}^{h_2}; \top)) = x_2 \cdot p(\alpha_{l_1}^{h_1}; \top)$	$\bar{P}, \underline{P}$ , assumptions
31	$\neg(\alpha \wedge (\alpha; \ell \neq 0); \top) \Rightarrow ((\alpha_{l_1}^{h_1}; \top) \wedge (\alpha; \beta_{l_2}^{h_2}; \top) \Leftrightarrow (\alpha_{l_1}^{h_1}; \beta_{l_2}^{h_2}; \top))$	<i>ITL</i>
32	$(\alpha_{l_1}^{h_1}; \top) \wedge (\alpha; \beta_{l_2}^{h_2}; \top) \Leftrightarrow (\alpha_{l_1}^{h_1}; \beta_{l_2}^{h_2}; \top)$	assumption, 31
33	$p((\alpha_{l_1}^{h_1}; \top) \wedge (\alpha; \beta_{l_2}^{h_2}; \top)) = p(\alpha_{l_1}^{h_1}; \beta_{l_2}^{h_2}; \top)$	32, <i>PITL1</i>
34	$\ell = 0 \wedge p(\alpha; \top) = 1 \Rightarrow p(\alpha_{l_1}^{h_1}; \top) = x_1$	$[ET_\alpha \in [l_1, h_1]]_{x_1}$
35	$\ell = 0 \wedge p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha_{l_1}^{h_1}; \beta_{l_2}^{h_2}; \top) = x_2 \cdot x_1$	10, 29, 30, 33, 34