Reasoning about QoS Contracts in the Probabilistic Duration Calculus

Dimitar P. Guelev* and Dang Van Hung†

February 24, 2008

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.

Introduction

Combining off-the-shelf and dedicated components has become an established approach to achieving reuse, modularity, productivity and reliability. Contracts facilitate the correct use of components. A contract is a collection of requirements which are written in terms of the component interface. Contract requirements should be satisfied by implementations of the component, provided that the items imported from other components also satisfy the requirements appearing in the contract for them. Four levels of contracts have been identified in [BJPW99]. These are the syntactical level, 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 and leaves out the QoS level from [BJPW99]. In our previous work we extended the rCOS model to capture requirements on timing and resources [Dan05, HD07]. We have considered hard requirements, where, e.g., missing 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., missing a deadline by little and not too often is tolerable. Handling requirements on the QoS involves reasoning about probability.

In this paper we extend designs to capture probabilistic requirements on execution time and develop a technique to reason about QoS of real-time embedded systems using an infinite-interval-based system of probabilistic DC (PDC) which 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.
1 Preliminaries

We consider only the extended set of the real numbers \( \mathbb{R} = \mathbb{R} \cup \{\infty\} \) as the flow of time in PITL and PDC. 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]. ITL with infinite intervals [ZDL95, PWX98, SX98, WX04] is the underlying non-probabilistic logic of PITL and PDC. It extends the syntax of predicate logic by a binary modality (\( \cdot \)), known as \textit{chop}.\(^1\) Non-logical symbols are divided into \textit{rigid} and \textit{flexible} depending on whether their meaning is required to be the same at all reference intervals or not. Individual variables are rigid.

An \textit{interpretation} of a \textit{vocabulary} \( L \) is a function \( I \) on \( L \) which maps the symbols from \( L \) to members of \( \mathbb{R} \), functions and predicates on \( \mathbb{R} \), according to the type and arity of symbols. \( I(s) \) takes an interval from \( \mathcal{I} \) as an additional argument in case \( s \) is flexible. We use the sets of intervals

\[
I_{\text{fin}} = \{(\tau_1, \tau_2] : \tau_1, \tau_2 \in \mathbb{R}, \tau_1 \leq \tau_2\}, I_{\text{inf}} = \{[\tau, \infty) : \tau \in \mathbb{R}\} \text{ and } \mathcal{I} = I_{\text{fin}} \cup I_{\text{inf}}.
\]

Given \( \sigma_1 \in I_{\text{fin}} \) and \( \sigma_2 \in \mathcal{I} \) such that \( \max \sigma_1 = \min \sigma_2, \sigma_1; \sigma_2 \) stands for \( \sigma_1 \cup \sigma_2 \). Given an interpretation \( I \), the values \( I_t(t) \) of terms \( t \) at intervals \( \sigma \in \mathcal{I} \) are defined in the usual way, with the reference interval being an additional argument for flexible symbols. Satisfaction \( \models \) is defined with respect to an interpretation and a reference interval. Flexible relation symbols are interpreted predicates which take the reference interval as an argument too. The clauses for \( \bot \), \( \Rightarrow \) and \( \exists \) are the usual ones. The clause for \( (\cdot \cdot \cdot) \) is

\[
I, \sigma \models (\varphi; \psi) \text{ iff } I, \sigma_1 \models \varphi \text{ and } I, \sigma_2 \models \psi \text{ for some } \sigma_1 \in I_{\text{fin}} \text{ and } \sigma_2 \in \mathcal{I} \text{ such that } \sigma_1; \sigma_2 = \sigma.
\]

0, \( \infty \), \( + \) and = are mandatory in ITL vocabularies and always have the usual interpretation. A mandatory flexible constant \( \ell \) always evaluates to the length of the reference interval. \textit{Infix} notation for + = and = and \( \land \), \( \lor \), \( \Rightarrow \) and \( \forall \) are used in the usual way. ITL-specific abbreviations include

\[
(\varphi_1; \ldots; \varphi_n) \equiv (\varphi_1; (\varphi_2; \ldots; \varphi_n)), \varphi \equiv (\top; \varphi; \top) \lor (\top; \varphi), \square \varphi \equiv \neg \diamond \neg \varphi.
\]

\( \diamond \) and \( \Box \) bind more tightly and \( (\cdot \cdot \cdot) \) binds less tightly than the boolean connectives.

A complete proof system for ITL with infinite intervals with respect to an appropriate abstract domain of durations was presented in [WX04].

\textit{Vocabularies} for DC with infinite intervals additionally include \textit{state variables} \( P, Q, \ldots \); \textit{state expressions} \( S \) are boolean combinations of state variables with the logical constants written as \( 0 \) and \( 1 \) and in turn appear as the argument of \textit{duration terms} \( \int S \), which are the DC-specific construct in the syntax of DC terms. Formulas in DC are as in ITL. State variables evaluate to piece-wise constant functions of type \( \mathbb{R} \rightarrow \{0, 1\} \). The value \( I_t(S) \) of state expression \( S \) at time \( t \) is defined using \( I_t(P)(\tau) \) for the involved state variables \( P \) in the usual way. Values of duration terms are defined by the equality

\[
I_t(\int S) = \max_{\sigma \in I_{\text{inf}}} I_t(S|\sigma) d\tau
\]

\( I_t(\int S) \) can be \( \infty \) for \( \sigma \in I_{\text{inf}} \). The expression \( [S] \) abbreviates \( \ell \neq 0 \land \int \neg S = 0 \) and \( \ell \) can be viewed as an abbreviation for \( \int 1 \) in DC.

Axioms and rules for DC (with infinite intervals) which are complete relative to validity in real-time ITL (with infinite intervals), have been presented in [H9Z2] (resp. [Gue07].)

The \textit{least-fixed-point operator} If \( \varphi_1, \ldots, \varphi_n \) have no negative occurrences of the propositional variables \( X_1, \ldots, X_n \) and \( i \in \{1, \ldots, n\} \), then \( \mu_i X_1 \ldots X_n \varphi_1, \ldots, \varphi_n \) is well-formed and \( I, \sigma \models \mu_i X_1 \ldots X_n \varphi_1, \ldots, \varphi_n \) iff \( \sigma \in A_i \), where \( A_1, \ldots, A_n \) are the smallest subsets of \( \mathcal{I} \) which satisfy the equalities

\[
A_i = \{\sigma \in \mathcal{I} : I_{X_1}^{\lambda_1 \sigma \in A_i, \ldots, X_n}^{\lambda_n \sigma \in A_n} \models \varphi_i\}, i = 1, \ldots, n.
\]

\textit{Iteration}, also known as Kleene star, can be defined using \( \mu \) by the equivalence \( \varphi^* \equiv \mu \varphi X. \ell = 0 \lor (\varphi; X) \).

\( I, \sigma \models \varphi^* \) can be defined independently by the condition:

\[
\min \sigma = \max \sigma \text{ or } \sigma = \sigma_1; \ldots; \sigma_n \text{ and } I, \sigma_i \models \varphi_i, i = 1, \ldots, n, \text{ for some } n < \omega, \sigma_1, \ldots, \sigma_n \in \mathcal{I}.
\]

\(^1\)Many authors write chop as \( \varphi \neg \psi \) instead of \( (\varphi; \psi) \).
Axioms and rules for μ and * in DC were proposed in [Pan95, Gue00a, GD05].

Higher-order quantifiers We use Π on flexible constants and state variables with the usual meaning, in order to describe the semantics of local variables. The deductive power of some axioms and rules for this usage has been studied in [ZG00, Gue00a, GD05].

Probabilistic ITL and DC with infinite intervals (PITL) extends the syntax of ITL terms by probability terms of the form p(φ) where φ is a formula. Formula syntax is as in ITL, with μ and higher-order quantifiers included. A PITL model is based on a collection of interpretations of a given vocabulary L. Each interpretation is meant to describe a possible behaviour of the modelled system. Consider a non-empty set W, a function I on W into the set of the ITL interpretations of L and a function P of type W × R × 2^W → [0, 1]. Let I^w and P^w abbreviate I(w) and λτ,X.P(w,τ,X), respectively, for all w ∈ W. I^w and P^w, w ∈ W, are intended to represent the probability branches of that model starting at w. The probability functions λX.P^w(τ,X) is a probability measure on the boolean algebra on other symbols. representing the probabilistic branching of w from time τ onwards.

Definition 1 Let τ ∈ R. Then w ≡τ v iff
I^w(s) = I^v(s) for all rigid symbols s ∈ L, except possibly the individual variables;
I^w(s)(d_1, ..., d_#s) = I^v(s)(d_1, ..., d_#s) for all flexible symbols s ∈ L, and d_1, ..., d_#s ∈ R and all σ ∈ I such that max σ ≤ τ;
P^w(τ', X) = P^v(τ', X) for all X ⊆ W and all τ' ≤ τ.

Clearly ≡τ is an equivalence relation on W for all τ ∈ R. Members of W which are τ-equivalent model the same behaviour up to time τ. If τ_1 > τ_2, then ≡τ_1 ≈ ≡τ_2 and w ≡∞ v holds if P^w = P^v and I^w and I^v agree on all symbols, except possibly some individual variables. [w]_≡ is the set of those v ∈ W which represent the probabilistic branching of w from time τ onwards.

Definition 2 A general PITL model for L is a tuple of the form ⟨W, I, P⟩ where F, W, I, P are as above and satisfy the following requirements for every w ∈ W:

• W is closed under variants of interpretations. If w ∈ W, x is an individual variable from L and a ∈ R, then there is a v ∈ W such that P^w = P^v and I^v = (I^w)^a_x, where (I^w)^a_x maps x to a and is the same as I^w on other symbols.

• The functions P^w are probability measures. For every w ∈ W and τ ∈ R the function λX.P^w(τ,X) is a probability measure on the boolean algebra (2^W, ∩, ∪, 0, W). Furthermore λX.P^w(τ,X) is required to be concentrated on [w]_≡: P^w(τ,X) = P^w(τ,X ∩ [w]_≡) for all X ⊆ W.

Informally, the probability for a behaviour in X ⊆ [w]_≡ to be chosen is P^w(τ,X). Satisfaction |= is defined in PITL with respect to a model M = ⟨W, I, P⟩, a w ∈ W, and a σ ∈ I. If ψ is a sentence, then

\[ [ψ]_{M,w,σ} = \{ v ∈ [w]_≡ | \text{Max } σ, ∞ | |= ψ \}. \]

This means that [ψ]_{M,w,σ} consists of the interpretations v which are max σ-equivalent to w and satisfy ψ at the infinite interval starting at min σ. Since any with free variables x_1, ..., x_n, M, v, [min σ, ∞ | |= ψ should be evaluated with I^w(x_1), ..., I^w(x_n) as the values of x_1, ..., x_n, in order to preserve the intended meaning:

\[ [ψ]_{M,w,σ} = \{ v ∈ [w]_≡ | (ψ' ∈ W)(P^w' = P^w \land I^v' = (I^w)^{x_1}_{x_1} \cap \cdots \cap I^w(x_n) ⇒ M, v', [min σ, ∞ | |= ψ) \}. \]

Using this notation, term values w_σ(t) of probability terms t can be defined by putting

\[ w_σ(p(ψ)) = P^w(\text{Max } σ, [ψ]_{M,w,σ}). \]

Values of terms of other forms are defined as in the (non-probabilistic) ITL.

The probability functions λX.P^w(τ,X) for w ∈ W and τ ∈ T in general PITL models M = ⟨W, I, P⟩ are needed just as much as they provide values for probability terms. That is why we accept structures of the form ⟨W, P, I⟩ with their probability functions λX.P^w(τ,X) be defined just on the (generally smaller) algebras ⟨{[ψ]}_{M,w,σ} : ψ ∈ L, σ ∈ I, max σ = τ⟩, ∩, ∪, , [w]_≡, as general PITL models too.

PITL is a conservative extension of ITL. Axioms and a proof rule which extend the proof system for ITL with infinite intervals to a system for PITL were shown in [Gue07] to be complete with respect to a generalisation of the R-based semantics, where R is replaced by an abstract domain and the probability measures are required to be only finitely additive.
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 with an origin of time $T_0 = \min T$ and a distinguished $w_0 \in W$ such that $[w_0]_{T_0} = W$ and $\lambda X. P^{w_0}(\tau_0, X)$ can be regarded as the global probability function. Then, given an arbitrary $w \in W$ and $\tau \in \mathbb{R}$, the probability function $\lambda X. P^w(\tau, X)$ should represent conditional probability, the condition being $\tau$. Hence we should have

$$P^{w_0}(\tau, A) = \int_{w \in [w_0]_{\tau}} P^w(\tau', A) d(\lambda X. P^{w_0}(\tau, X)). \quad (1)$$

The following rules enable approximating (1) with arbitrary precision in PITL proofs:

\[
\begin{align*}
(P) \quad & \exists \ell : \neg (\varphi; \ell \neq 0) \\
& \ell = 0 \land p(\varphi \land p(\psi) < x; \top) = 0 \Rightarrow p((\varphi; \top) \land \psi) \geq x.p(\varphi; \top) \\
(P) \quad & \exists \ell : \neg (\varphi; \ell \neq 0) \\
& \ell = 0 \land p(\varphi \land p(\psi) > x; \top) = 0 \Rightarrow p((\varphi; \top) \land \psi) \leq x.p(\varphi; \top)
\end{align*}
\]

The proof system for PITL from [Gue07] is minimal. Using the abbreviations

\[
\begin{align*}
\varphi^h \equiv \varphi \land \ell \geq l \land \ell \leq h \quad \text{and} \quad [ET_\varphi \in [l, h]]_x \equiv \ell = 0 \land p(\varphi; \top) = 1 \Rightarrow p(\varphi^h; \top) = x,
\end{align*}
\]

we can write the derived rule

\[
\text{(Seg)} \quad \alpha \Rightarrow \neg (\alpha; \ell \neq 0), \beta \Rightarrow \neg (\beta; \ell \neq 0), [ET_\alpha \in [l_1, h_1]]_{x_1}, [ET_\beta \in [l_2, h_2]]_{x_2}, \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha^{h_1}; \beta^{h_2}; \top) = x_1 x_2,
\]

which is particularly important to our examples.

The system of probabilistic DC (PDC) with infinite intervals which we use in this paper is obtained by adding state variables and duration terms to PITL in the way used 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 $\mathbb{R}$.

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

We propose a toy language to illustrate our approach. It is shaped after that from [GD02] and has restricted form of method call, in order to set the stage for the use of components and contracts.

Programs consist of components, which import and/or export methods. Their syntax is:

\[
\text{component} ::= \text{component name} \quad \quad \text{method} ::= \text{name}(\text{parameter list})[\text{code}];
\]

\[
\quad \quad \{\text{import method}\}^* \quad \quad \{\text{export method}\}^*
\]

end name

The part code is required only for exported methods. It has the syntax

\[
\text{code} ::= \text{stop} | \quad \quad \text{(thread) termination statement}
\]

\[
\text{return} [e] | \quad \quad \text{return control and possibly a value}
\]

\[
X \quad \quad \text{continuation}
\]

\[
(x := e; \text{code}) | \quad \quad \text{assignment}
\]

\[
\text{(delay } r; \text{ code}) | \quad \quad \text{delay by the specified amount of time}
\]

\[
\text{(call } [x := \text{name}(\text{parameter list}); \text{code}) | \quad \quad \text{call method and possibly obtain a value}
\]

\[
\text{if } b \text{ then code else code} | \quad \quad \text{conditional statement}
\]

\[
\text{letrec code where } X : \text{code} ; \ldots ; X : \text{code} | \quad \quad \text{mutual recursion statement}
\]

\[
\text{var } x \text{ code}
\]

\[
\text{code} \parallel \text{code} \quad \quad \text{parallel composition}
\]

\[
4
\]
We do not allow \texttt{var} to occur in the scope of other control statements. Assignments are atomic. Parameters are passed by value. A mutual recursion statement can trigger an infinite computation. Components are passive. The \textit{active} part of a program is just a piece of code, typically a collection of concurrently running interleaved threads. The syntax of control statements deliberately makes tail-recursion the only expressible form of repetitive behaviour. We give no details on the type system and tacitly assume an appropriately many-sorted system of DC.

The execution of code 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 defines the set of its observable behaviours. We model each program variable $v$ by a corresponding pair of flexible constants $v$ and $v'$, which denote the value of $v$ at the beginning and at the end of the reference interval and therefore satisfy the axiom $\forall x (v' = v ; v \neq x)$ where $x$ is a rigid individual variable. We model methods $m$ which return a value by a corresponding flexible function symbol. A formula of the form $v' = m(e_1, \ldots, e_n)$ means that the reference interval describes a complete invocation of $m$ with $e_1, \ldots, e_n$ as the input parameters and $v'$ as the value. We use a flexible predicate symbol for methods which return no value. We use dedicated state variables $R$ and $W$ to indicate that the thread is currently running, or has terminated, respectively. Building on the work from [PD98, GD02], we 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. $R$, $W$ and $N$ satisfy the axioms

$$T(R, W) \equiv [R \Rightarrow N] \land [R \Rightarrow \neg W] \land \square \neg([W] ; [\neg W]),$$

which express that computation time is negligible, a process can never be both running and terminated, and, once terminated, is never re-activated. A dedicated pair of state variables $R$ and $W$ describes the status of each thread. $N$ marks negligible time for all threads. The formulas

$$K(V) \equiv \bigwedge_{x \in V} x' = x \quad \text{and} \quad K^R(V) \equiv K(V) \land [R].$$

mean that the variables from $V$ preserve their values. $K^R(V)$ additionally means that 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{align*}
\text{[stop]}_V & \equiv [W] \\
\text{[return } e\text{]}_V & \equiv ([\neg R] ; K^R(V) \land r' = e) \\
\text{[return]}_V & \equiv [\neg R] \\
\text{[X]}_V & \equiv X \\
\text{[[C}_1; C_2\text{]]}_V & \equiv ([C_1]_V ; [C_2]_V) \\
\text{[x := e]}_V & \equiv ([\neg R] ; K^R(V \setminus \{x\}) \land x' = e) \\
\text{[delay r]}_V & \equiv [\neg R] \land r = \int \neg N \\
\text{[if } b \text{ then } C_1 \text{ else } C_2\text{]}_V & \equiv ([\neg R] ; (b \land K^R(V) ; [C_1]_V) \lor (\neg b \land K^R(V) ; [C_2]_V)) \\
\text{[call } v := m(e_1, \ldots, e_n)\text{]}_V & \equiv ([\neg R] ; K(V \setminus \{v\}) \land v' = m(e_1, \ldots, e_n) ) \\
\text{[call } m(e_1, \ldots, e_n)\text{]}_V & \equiv ([\neg R] ; K(V) \land m(e_1, \ldots, e_n)) \\
\text{[letrec } C \text{ where } X_1 : C_1; \ldots; X_n : C_n\text{]}_V & \equiv \mu_{n+1} X_1 \ldots X_n Y.[C_1]_V, \ldots, [C_n]_V, [C]_V \\
\text{[var } v; C\text{]}_V & \equiv \exists v \exists v' \Box (([\neg R] \Rightarrow v' = v) \land \forall x (v' = x ; v \neq x)) \land \llbracket C\rrbracket_{V \cup \{v\}} \\
\text{[[C}_1; C_2\text{]}} & \equiv \exists R_1 \exists R_2 \exists W_1 \exists W_2 \left( ([W] \Leftrightarrow W_1 \land W_2] \land [R \Leftrightarrow R_1 \lor R_2] \land [\neg R_1 \land R_2] \land \\
& \quad T(R_1, W_1) \land [R_1/R, W_1/W][C_1]_V \land \\
& \quad T(R_2, W_2) \land [R_2/R, W_2/W][C_2]_V \right) \\
\text{[export } m(p_1, \ldots, p_n)\text{ code]\]} & \equiv \Box \forall p_1 \ldots \forall p_n \forall r' (r' = m(p_1, \ldots, p_n) \Rightarrow \llbracket \text{code}\rrbracket_0), \text{ if } m \text{ returns a value} \\
\text{[export } m(p_1, \ldots, p_n)\text{ code]\]} & \equiv \Box \forall p_1 \ldots \forall p_n (m(p_1, \ldots, p_n) \Rightarrow \llbracket \text{code}\rrbracket_0), \text{ if } m \text{ returns no value}
\end{align*}

The semantics of a component is the conjunction of the formulas $\llbracket \text{export } m(p_1, \ldots, p_n)\text{ code}\rrbracket$ for its exported methods. Declarations of imported methods carry only typing information.
3 Reasoning about timed programs in PDC: pattern and examples

Let $C$ be a piece of code. Then the formula $[C]_{V}$ contains the flexible function and relation symbols for the methods with calls in $C$. Let $m$ be such a method; let $m$ return no value for the sake of simplicity. Let $B$ be the body of $m$. By replacement of equivalents we can derive

$$[\text{export } m(p_{1}, \ldots, p_{n}) \ B] \wedge [C]_{V} \Rightarrow [[B]/m][C]_{V},$$

where the substitution $[[B]/m]$ distributes over the boolean connectives, chop and quantifiers and $[B]/m(e_{1}, \ldots, e_{n})$ is defined as $[e_{1}/p_{1}, \ldots, e_{n}/p_{n}][B]_{\emptyset}$. Assume that the satisfaction of a requirement $Req_{C}$, written in $DC$ by $C$ is expressed as the equivalence

$$[C]_{V} \Rightarrow Req_{C}$$

and, according to a contract, $m$ is supposed to satisfy a requirement $Req_{m}$, that is,

$$[B]_{\emptyset} \Rightarrow Req_{m}$$

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

$$[Req_{m}/m][C]_{V} \Rightarrow Req_{C}$$

states that $C$ would satisfy $Req_{C}$, provided that the imported implementation of $m$ satisfies $Req_{M}$.

This setting enables reasoning about the probability distribution of the execution time of code that calls imported methods 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

$$p([[C]_{V} \wedge \int \neg N \leq d; \top]),$$

where we use $\int \neg N$ to measure only non-negligible execution time spent on the execution of delay or by other processes. Now let $F_{m}$ be a rigid function symbol such that $F_{m}(x)$ denotes a lower bound for the probability for $m$ to terminate within time $x$. Let $P_{m}$ be the precondition for the successful execution of $m$. Let $\overline{p}$ abbreviate $p_{1}, \ldots, p_{n}$. Then

$$\forall x(t = 0 \Rightarrow p(P_{m}(\overline{p}) \wedge m(\overline{p}) \wedge \int \neg N > x; \top) < 1 - F_{m}(x)) \vdash_{PITL} p([[C]_{V} \wedge \int \neg N \leq d; \top) \geq c$$

means that the probability for $C$ to terminate within $d$ time units is at least $c$. The correspondence between the assumption on the execution time of $m$ and the derived estimate of the execution time of $C$ can be expressed even more accurately, if we make $d$ and $F_{m}$ parameters in an appropriate expression $F_{C}$ in place of $c$:

$$\forall x(t = 0 \Rightarrow p(P_{m}(\overline{p}) \wedge m(\overline{p}) \wedge \int \neg N > x; \top) < 1 - F_{m}(x)) \vdash_{PITL} p([[C]_{V} \wedge \int \neg N > d; \top) < 1 - F_{C}(d, F_{m}).$$

In general $F_{C}$ represents a mapping from distributions to distributions, but if the form of $F_{m}$ is known up to numerical parameters such as mean and variance, then $F_{C}$ can be defined as a mapping from their numerical domains instead of the space of distributions.

Example 1 Consider downloading e-mail, which consists of establishing a connection with a server, followed by the actual download. Let the code $C$ for this call two imported methods, connect() and getMail():

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

Let $F_{\text{connect}}(t)$ be the probability for connecting within time $t$. Let the amount of the e-mail be probabilistically distributed too and the probability for downloading it in time $t$ be $F_{\text{getMail}}(t)$. Then lower bounds $F_{C}$ for the distribution of the execution time of $C$ satisfy the formula:

$$t = 0 \Rightarrow p([[\text{call ok := connect()}]) \wedge \int \neg N > t; \top) < 1 - F_{\text{connect}}(t),$$

$$t = 0 \Rightarrow p(ok \wedge [[\text{call getMail()}]) \wedge \int \neg N > t; \top) < 1 - F_{\text{getMail}}(t) \vdash_{PITL} p([[C] \wedge \int \neg N > t; \top) < 1 - F_{C}(t)$$

Since the time for connecting and the quantity of e-mail to download can be assumed independent,

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

(2)

6
$F_C$ can be derived in PITL only approximately, because PITL does not capture taking the limits involved in the definition of the integral in (2). This corresponds to the established use of numerical approximations for distributions. Except for some thoroughly studied distributions, cumulative probability functions seldom have a closed form. Using contracts makes it natural to work with lower bounds and not exact probabilities. The latter may as well not exist. This makes approximations satisfactory. To derive such approximations for (2) in PITL, we find a sequence $A_k$, $k = 0, 1, \ldots$, of terms involving $F_{\text{connect}}$, $F_{\text{getMail}}$ and $t$ such that

$$p([C] \land \exists N > t; \top) < 1 - A_k$$

for all $k$ can be derived in PITL and, by the definition of $f$, $\lim_k A_k = F_C(t)$. Taking this limit briefly takes us outside PITL. The part of the derivation within PITL is a formalisation of a standard calculation. Let

$$\varphi_{l_1}^{l_2} \vDash \varphi \land \exists N > t_1 \land \exists N \leq t_2$$

(3)

Every method call can terminate at most once. This implies the validity of the formulas $\text{connect} \Rightarrow \neg(\text{connect}; \ell \neq 0)$ and $\text{getMail} \Rightarrow \neg(\text{getMail}; \ell \neq 0)$ and enables an application of Seq to derive

$$\ell = 0 \land p(\text{connect}; \text{getMail}; \top) = 1 \Rightarrow p(\text{connect}; \text{getMail}) \land \exists N \leq t; \top \leq \sum_{l + m \leq k - 1} p(\text{connect}; \text{getMail}) + \sum_{l + m = k} p(\text{connect}; \text{getMail}) + F_{\text{connect}}(\frac{l + m + 1}{k} - F_{\text{getMail}})$$

for all $t, m \in \{0, \ldots, k - 1\}$. Now by a repeated application of the PITL axiom $P_\ell$, and using that $F_{\text{connect}}(0) = 0$, we obtain

$$p(\text{connect}; \text{getMail}) \land \exists N \leq t; \top \leq \sum_{l + m \leq k - 1} p(\text{connect}; \text{getMail}) + \sum_{l + m = k} p(\text{connect}; \text{getMail}) + F_{\text{connect}}(\frac{k - m + 1}{k} - F_{\text{getMail}}) + B_k$$

where $B_k \leq \max_{l \leq k - 1} F_{\text{connect}}(\frac{l + 1}{k} - F_{\text{getMail}})$, 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 Seq was formulated with $\varphi_{l_1}^{l_2}$ standing for $\varphi \land \ell \geq t_1 \land \ell \leq t_2$, but it applies to (3) as well.

Example 2 Consider attempting to download 5 files in quick succession. With a server which allows at most 4 files to be downloading simultaneously, the 5th request can be cancelled by the browser due to a timeout. We are interested in the probability of cancellation. Here follows an extremely simplified variant of the relevant browser code:

```plaintext
letrec X where
    X : if userRequest then
        (userRequest := false;
        call handle := requestDownload(url, timeout);
        if handle! = null
            then (call download(handle); stop)
            else (call signalTimeout(url); stop)
        )
    else X
```

A separate process is assumed to indicate the arrival of a new download request by setting the shared variable userRequest and placing the URL in the shared variable url. Let

$$\alpha(R, T) \vDash \begin{cases} [\neg R]; K^R(V) \land \neg userRequest' ; [\neg R]; K^R(V) \land userRequest; \\
                          [\neg R]; K^R(V \setminus \{userRequest\}) \land userRequest' = false \end{cases} \land \exists N = T$$

and

$$\beta(R, W, T) \vDash \begin{cases} [\neg R]; K^R(V \setminus \{handle\}) \land handle' = requestDownload(url, timeout); \\
                          [\neg R]; K^R(V) \land handle! = null; [\neg R]; K^R(V) \land download(handle) \land \exists N = T; [W] \end{cases}$$
According to the semantics of (4), \( \alpha(R,T) \) describes the repeated execution of lines 2-3 and 9 until \textit{userRequest} becomes true with \( T \) denoting the overall execution time, and \( \beta(R,W,T) \) corresponds to the execution of lines 4-6, with \( R \) and \( W \) describing the status of the thread created in order to complete the requested download, and \( T \) denoting the download time. The scenario of launching the five downloads involves six threads: one for each download and one to keep the system ready for further requests. Let \( R_1, \ldots, R_6, W_1, \ldots, W_6 \) describe the status of the six threads. Then the scenario can be described by the formula

\[
\exists R_1 \ldots \exists R_6 \exists W_1 \ldots \exists W_6 \left[ [W \leftrightarrow \bigwedge_{i=1}^{6} R_i] \land [R \leftrightarrow \bigvee_{i=1}^{6} R_i] \land \bigwedge_{1 \leq i < j \leq 6} \neg(R_i \land R_j) \land \bigwedge_{i=1}^{6} T(R_i, W_i) \land \gamma \right] (5)
\]

where \( \gamma \) describes the concurrent execution of the six threads and is written using the additional abbreviations:

\[
\alpha_4(T) \equiv \alpha(R_{i+1} \lor \ldots \lor R_6, W_{i+1} \land \ldots \land W_6, T) \text{ and } \beta_i(T) \equiv \beta(R_i, W_i, T).
\]

With these abbreviations \( \gamma \) can be written as

\[
(\alpha(R, W, T_3); \left( \begin{array}{c}
\beta_1(D_1) \land \alpha_1(T_1); \\
\beta_2(D_2) \land \alpha_2(T_2); \\
\beta_3(D_3) \land \alpha_3(T_3); \\
\beta_4(D_4) \land \alpha_4(T_4; \xi \land \eta)
\end{array} \right))
\]

Here \( T_i \) denotes the time between launching the \( i \)th thread and the \( i+1 \)st download and \( D_i \) denotes the duration of the \( i \)th download, \( i = \{1, \ldots, 5\} \). The formulas \( \xi \) and \( \eta \) denote

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

and

\[
([\neg R_6]; K^{R_6}(V) \land \neg \text{\textit{userRequest}}*; \top),
\]

and correspond to the thread for the 5th download and the thread 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 semantics of \textit{letrec} implies (5); this can be established using the validity of \( \mu X. \varphi \equiv [\mu X. \varphi/X] \varphi \). Assuming that the rate of downloading is the limiting factor for the working of the entire system, which allows us to ignore time taken for dialog, computation and by \textit{requestDownload} for the first four downloads, the 5th download becomes cancelled in case \( x \) exceeds \( \text{timeout} \), which is equivalent to

\[
\text{timeout} + \frac{1}{4} \sum_{j=1}^{4} T_i < \frac{1}{4} \sum_{i=1}^{4} \left( D_i + \sum_{j=1}^{i-1} T_j \right).
\]

Let \( F(l,t) \) be a lower bound for the probability for \textit{download} do complete a download of length \( l \) within time \( t \). It can be assumed that \( F(a,at) = F(l,t) \) for all \( a > 0 \) and that \( F(l,t) = 0 \) in case \( \frac{t}{v} \) exceeds the top transmission rate \( v \). Let \( l_i \) be the length of the \( i \)th download, \( i = \{1, \ldots, 5\} \). Let \( l_i > v(T_1 + T_2 + T_3 + T_4) \) for \( i = 1, \ldots, 4 \), that is, none of the downloads can be over before all of them have been launched, for the sake of simplicity. Then the probability \( P_i \) for \( i = \{1, \ldots, 4\} \) to be the first download to complete, and to complete before the timeout for the pending 5th download is at least

\[
\int_{\{(q_1, \ldots, q_4); l_1 < q_1 \land \ldots \land l_4 < q_4\}} \frac{1}{2^4} F(q_1,t)(l_1 - q_1, \text{timeout}) \prod_{k=1}^{4} F'(q_k, \sum_{s=k}^{4} T_s) dq_1 \ldots dq_4.
\]

The probability for the 5th download not to be cancelled is \( P_1 + \ldots + P_4 \). Approximations of the above integral can be derived in PTL using \( \text{Seq} \) much like in Example 1.

Using a contract in which the execution time of \textit{download} is approximated by a distribution depending just the amount of data to transmit is too crude. A more accurate calculation is possible by taking the amount of competing traffic in account, but the form of contract that we propose does not enable it.
4 Probabilistic timed designs

A design \( \langle P, R \rangle \), usually written as \( P \vdash R \), describes a computation by a precondition \( P \), an input-output relation \( R \). \( P \) constrains the initial values \( v \) of the variables, and \( R \) is a relation between \( v \) and the final values \( v' \) of the variables, which holds if \( v \) initially satisfy \( P \). A probabilistic timed design \( \langle P, R, F \rangle \) additionally includes an execution time distribution \( F \). \( F(v,t) \) is a lower bound for the probability for the computation to terminate within time \( t \), provided that \( P(v) \) holds. A hard bound \( d \) on execution time can be expressed by a \( F \) satisfying \( F(d') = 1 \) for all \( d' \geq d \).

4.1 Describing designs in PDC

The property of method \( m \) encoded by \( \langle P, R, F \rangle \) can be written as the PITL formulas

\[
m \Rightarrow (P(v) \Rightarrow R(v, v')) \quad \text{and} \quad \ell = 0 \Rightarrow p(P(v) \land m \land \ell > t; \top) < 1 - F(v, t).
\]

The first one is for the functional behaviour of \( m \). The second one states that if \( P(v) \) holds, then \( m \) to takes more than \( t \) time units with probability less than \( 1 - F(v, t) \). \( F \) is just a lower bound, because an exact probability need not exist.

4.2 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

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

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 \( D_2 \). Obviously if \( D_1 \sqsubseteq D_2 \), then

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

entail

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

5 Probabilistic timed contracts

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 \( V_m \times \mathbb{R}_+ \rightarrow [0, 1] \) 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 \( \{ n \in M_i : F_n \text{ occurs in } F_m \} \) by \( C_{i,n} \). Semantically, if \( m \in M_e \), then the type of \( F_m \) is

\[
\prod_{n \in C_{i,n}} V_n \times \mathbb{R}_+ \rightarrow [0, 1] \rightarrow (V_m \times \mathbb{R}_+ \rightarrow [0, 1]).
\]

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

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

1. $M'_i \subseteq M_i, M'_e \supseteq M_e$;
2. $(P_m, R_m) \subseteq (P'_m, R'_m)$ for $m \in M_i'$, $(P'_m, R'_m) \subseteq (P_m, R_m)$ for $m \in M_e$;
3. $F_m(v, t) \leq F'_m(v, t)$ for $m \in M_e, v \in V_m, t \in R_m$ regardless of the values of $F_n, n \in M_i$.

5.1 Composing probabilistic timed contracts

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

1. $P_m(v) \equiv P_m^1(v) \land P_m^2(v), R_m(v, v') \equiv R_m^1(v, v') \land R_m^2(v, v')$ and $F_m = F_m^1 = F_m^2$ for $m \in M_i^1 \cap M_i^2$;
2. $P_m = P_m^1$ and $R_m = R_m^1$ for $m \in M_i^k \cup (M_i^k \setminus M_i^{k-2})$, $k = 1, 2$;
3. $F_m = F_m^k$ for $m \in M_i^k \setminus M_i^{k-2}, k = 1, 2$.

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

$F_m = F_m^0$ for $m \in M_i^k$ of dependency depth $0$;
$F_m = \{F_m/F_m^k : n \in C_{i,m} \cap F_m^k \}$ for $m \in M_i^k \cup M_e^k$ of nonzero dependency depth.

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

If there are circular dependencies between $C^1$ and $C^2$, then the $F_m$ for the exported methods in $C^1 \cup C^2$ should be a solution of the system of equations

$$X_m = [X_n/F_m^k : n \in C_{i,m}]F_m^k.$$ Solving it without restrictions on $F_m^k$ can be hard2, but if $F_m^k$ and monotonic, then $X_m$ can be obtained as the limits of the sequences $X_m^j, j < \omega$, where $X_m^0 \equiv 0$ and

$$X_m^{j+1} = [X_n/F_m^k : n \in C_{i,m} \cup F_m^k]F_m^k \text{ for } m \in M_i^k.$$ Observe that $X_m^0 \equiv 0$ implies that $X_m^1$ would give non-zero termination probability only to runs of $m$ with no calls to other imported methods; $X_m^2$ would give non-zero probability for runs with calls to imported methods which themselves lead to no further calls, etc. Since $F_m^k$ are meant to be under-approximations, and the monotonicity of $F_m^k$ entails $X_m^s \leq X_m^{s+1} \leq \lim X_m^j$ for all $s < \omega, X_s$ can be used as $F_m$ instead of $\lim X_m^j$ for sufficiently large $s$, to achieve a crude, but less expensive approximation.

**Concluding remarks**

Here we focused just on soft requirements on execution time, but we believe that the approach can be used to capture other QoS requirements involving probability as well. The notion of QoS originated from telecommunications. Our examples come from everyday use of the Internet and need no expertise to understand. However, we believe that our technique would work just as well in other areas such as embedded systems.

---

2In practice $F_m^k$ can be non-monotonic: increasing the execution time of an imported method may indeed shorten the execution time of code which would abort if an imported method misses a deadline.
References


relative to validity in real-time

The axioms and rules below were proposed for

**A.2 Axioms and rules for**

not occur in

[72x246]relative to validity in real-time

The axioms and rules below were proposed for

A.2 Axioms and rules for

is correct only if no variable in

axioms about an abstract domain of durations in [WX04]:

The following axioms and rules have been shown to form a complete proof system for

ITL

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]:

(A1) \((\varphi; \psi) \land \neg(\chi; \psi) \Rightarrow (\varphi \land \neg \chi; \psi), (\varphi; \psi) \land \neg(\varphi; \chi) \Rightarrow (\varphi; \psi \land \neg \chi)\)

(A2) \(((\varphi; \psi); \chi) \Rightarrow (\varphi; (\psi; \chi))\)

(R) \((\varphi; \psi) \Rightarrow \varphi, (\psi; \varphi) \Rightarrow \varphi\) if \(\varphi\) is rigid

(B) \((\exists x \varphi; \psi) \Rightarrow \exists x(\varphi; \psi), (\psi; \exists x \varphi) \Rightarrow \exists x(\varphi; \psi)\) if \(x\) has no free occurrences in \(\psi\)

(L1) \((\ell = x; \varphi) \Rightarrow \neg(\ell = x; \neg \varphi), (\varphi; \ell = x \land x \neq \infty) \Rightarrow \neg(\neg \varphi; \ell = x)\)

(L2) \(\ell = x + y \land x \neq \infty \Rightarrow (\ell = x; \ell = y)\)

(L3) \(\varphi \Rightarrow (\ell = 0; \varphi), \varphi \land \ell \neq \infty \Rightarrow (\varphi; \ell = 0)\)

(S1) \((\ell = x \land \varphi; \psi) \Rightarrow \neg(\ell = x \land \neg \varphi; \chi)\)

(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) \(\frac{\varphi \Rightarrow \psi}{(\varphi; \chi) \Rightarrow (\psi; \chi)}, \frac{\varphi \Rightarrow \psi}{(\chi; \varphi) \Rightarrow (\chi; \psi)}\)

Using the first order logic axiom

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

is correct only if no variable in \(t\) becomes bound due to the substitution, and either \(t\) is rigid or \((..)\) 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].

- (DC1) \(\int 0 = 0\)
- (DC2) \(\int 1 = \ell\)
- (DC3) \(\int S \geq 0\)
- (DC4) \(\int S_1 + \int S_2 = \int(S_1 \lor S_2) + \int(S_1 \land S_2)\)
- (DC5) \((\int S = x; \int S = y) \Rightarrow \int S = x + y\)
- (DC6) \(\int S_1 = \int S_2\) if \(S_1\) and \(S_2\) are propositionally equivalent
- (IR1) \(\frac{[\ell = 0/A] \varphi \Rightarrow [A \lor (A; [S] \lor [\neg S])] / A] \varphi}{[\ell = 0/A] \varphi\)
- (IR2) \(\frac{[\ell = 0/A] \varphi \Rightarrow [A \lor (\{S\} \lor [\neg S]; A)] / A] \varphi}{[\ell = 0/A] \varphi\)
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:

(T1) \( \ell = 0 \lor ([S]; \top) \lor ([\neg S]; \top) \)

(T2) \( \ell = 0 \lor \ell = \infty \lor (\top; [S]) \lor (\top; [\neg S]) \)

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 \(\mathbb{R}\)-based semantics, where \(\mathbb{R}\) is replaced by an abstract domain and the probability measures are required to be only finitely additive.

**Extensionality**

(P\(_\bot\)) \( p(\bot) = 0 \)

(P\(_\top\)) \( p(\top) = 1 \)

(P\(_\lor\)) \( p(\varphi) + p(\psi) = p(\varphi \lor \psi) + p(\varphi \land \psi) \)

**Arithmetics of probabilities**

(P\(_\infty\)) \( \ell = \infty \Rightarrow (\varphi \leftrightarrow p(\varphi) = 1) \)

(P\(_\leq\)) \( \vdash (\varphi; \ell = \infty) \Rightarrow (\psi \Rightarrow \chi) \)

A.4 Useful theorems and derived rules for PITL

All the theorems and rules below except \(P'_r\) are valid in general PITL models. \(P'_r\) is valid in PITL models with global probability.

(P\(_\infty\)) \( (\varphi; \ell = \infty) \lor (\varphi \land \ell = \infty) \Rightarrow (\psi \Rightarrow \chi) \)

(P\(_ITL1\)) \( \frac{p(\varphi) \leq p(\psi)}{\varphi \Rightarrow \psi} \frac{p(\varphi) = p(\psi)}{p(\varphi) \leftrightarrow \psi} \)

(P\(_ITL2\)) \( p(\varphi) + p(\neg \varphi) = 1 \)

(P\(_ITL3\)) \( p(\varphi) < p(\psi) \Rightarrow p(\psi \land \neg \varphi) \neq 0 \)

(P\(_ITL4\)) \( p(\varphi) = p(\varphi \land \ell = \infty) \)

(P\(_ITL5\)) \( p(\varphi) \leq 1 \)

(P\(_ITL6\)) \( \frac{\neg \varphi \Rightarrow p(\varphi) = 1}{\neg \varphi} \)

(P\(_ITL7\)) \( \frac{p(\varphi) = 1 \lor p(\varphi; \top) = 1}{(\varphi; \top) \Rightarrow p(\varphi; \top) = 1} \)

(P\(_ITL8\)) \( p(\varphi) = 1 \land p(\psi) = x \Rightarrow p(\varphi \land \psi) = x \)

(P\(_ITL9\)) \( p(\varphi \Rightarrow \psi) = 1 \Rightarrow (p(\varphi) = 1 \Rightarrow p(\psi) = 1) \)

(P\(_ITL10\)) \( p(\varphi) \land p(\psi) > 1 \Rightarrow p(\varphi \land \psi) > 0 \)

(P\(_P\)) \( \frac{\varphi \Rightarrow \neg (\varphi; \ell \neq 0)}{(\varphi; p(\varphi) = x) \Rightarrow p(\varphi; \psi) = x} \)

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.
$P^\infty$: 

1. $(\varphi; \ell = \infty) \Rightarrow (\psi \Rightarrow \chi)$ assumption, ITL
2. $\varphi \land \ell < \infty \Rightarrow p(\psi) \leq p(\chi)$ 1, $P_\leq$
3. $\ell = \infty \land \varphi \Rightarrow (p(\psi) = 0 \land p(\chi) = 0)$ assumption, $P_\infty$, PITL2

$\lor (p(\psi) = 0 \land p(\chi) = 1)$

$\lor (p(\psi) = 1 \land p(\chi) = 1)$

4. $\varphi \land \ell = \infty \Rightarrow p(\psi) \leq p(\chi)$ 3, ITL
5. $\ell < \infty \land \ell = \infty$ ITL
6. $\varphi \Rightarrow p(\psi) \leq p(\chi)$ 2, 4, 5

**PITL1:**

1. $\varphi \Rightarrow \psi$ assumption
2. $(\top; \ell = \infty) \lor (\top \land \ell = \infty) \Rightarrow (\varphi \Rightarrow \psi)$ 1, ITL
3. $p(\varphi) \leq p(\psi)$ 2, $P^\infty_\leq$

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

**PITL2:**

1. $\varphi \land \lnot \varphi \Leftrightarrow \bot$ ITL
2. $p(\varphi \land \lnot \varphi) = p(\bot)$ 1, PITL1
3. $p(\varphi \land \lnot \varphi) = 0$ 2, $P_\bot$
4. $\varphi \lor \lnot \varphi \Leftrightarrow \top$ ITL
5. $p(\varphi \lor \lnot \varphi) = p(\top)$ 4, PITL1
6. $p(\varphi \land \lnot \varphi) = 1$ 5, $P_\top$
7. $p(\varphi) + p(\lnot \varphi) = p(\varphi \land \lnot \varphi) + p(\varphi \land \lnot \varphi) + p(\varphi \land \lnot \varphi)$ $P_\top$
8. $p(\varphi) + p(\lnot \varphi) = 1$ 2, 6, 7, ITL

**PITL3:**

1. $p(\psi) \leq p(\varphi \lor \psi)$ $P^\infty_\leq$
2. $p(\varphi) + p(\psi \land \lnot \varphi) = p(\varphi \land \psi \land \lnot \varphi) + p(\varphi \lor \psi \land \lnot \varphi) + p(\varphi \land \psi \land \lnot \varphi)$ $P_\top$
3. $p(\varphi) + p(\psi \land \lnot \varphi) = p(\varphi \lor \psi)$ 2, PITL1, $P_\bot$
4. $p(\varphi) < p(\psi) \Rightarrow p(\varphi) < p(\varphi \lor \psi)$ 1
5. $p(\varphi) < p(\psi) \Rightarrow p(\psi \land \lnot \varphi) \neq 0$ 3, 4

**PITL4** is obtained by applying $P^\infty_\leq$ to the ITL theorems

$(\top; \ell = \infty) \lor (\top \land \ell = \infty) \Rightarrow (\varphi \Rightarrow \varphi \land \ell = \infty)$ and $(\top; \ell = \infty) \lor (\top \land \ell = \infty) \Rightarrow (\ell = \infty \land \varphi \Rightarrow \varphi)$.

**PITL5:**

1. $\varphi \Rightarrow \top$ ITL
2. $p(\varphi) \leq p(\top)$ 1, PITL1
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. $\lnot \varphi$ assumption
2. $p(\lnot \varphi) = 1$ 1, PITL6
3. $p(\varphi) = 0$ PITL2
PITL7:
1. \((\varphi; T; \ell = \infty) \lor ((\varphi; T) \land \ell = \infty) \Rightarrow (T \Rightarrow (\varphi; T))\)  \( \text{ITL} \)
2. \((\varphi; T) \Rightarrow p(\varphi; T) = 1\)  \( P_\leq \)

PITL8:
1. \(p(\varphi) = 1 \land p(\psi) = x \Rightarrow p(\varphi \land \psi) + p(\varphi \lor \psi) = 1 + x\)  \( P_+ \)
2. \(\varphi \Rightarrow (\varphi \lor \psi)\)  \( \text{ITL} \)
3. \(p(\varphi) \leq p(\varphi \lor \psi)\)  \( 2, \text{PITL1} \)
4. \(p(\varphi) = 1 \Rightarrow p(\varphi \lor \psi) = 1\)  \( 3, \text{PITL5} \)
5. \(p(\varphi) = 1 \land p(\psi) = x \Rightarrow p(\varphi \land \psi) = x\)  \( 1, 4 \)

PITL9:
1. \(p(\varphi \Rightarrow \psi) = 1 \land p(\varphi) = 1 \Rightarrow p((\varphi \Rightarrow \psi) \land \psi) = 1\)  \( \text{PITL8} \)
2. \(\varphi \Rightarrow \psi \land \psi \Rightarrow \psi\)  \( \text{PITL1} \)
3. \(p((\varphi \Rightarrow \psi) \land \psi) \leq p(\psi)\)  \( 2, \text{PITL1} \)
4. \(p(\psi) \leq 1\)  \( \text{PITL5} \)
5. \(p(\varphi \Rightarrow \psi) = 1 \Rightarrow (p(\varphi) = 1 \Rightarrow p(\psi) = 1)\)  \( 1 - 4 \)

PITL10:
1. \(p(\varphi) + p(\psi) > 1 \Rightarrow p(\varphi \land \psi) + p(\varphi \lor \psi) > 1\)  \( P_+ \)
2. \(p(\varphi \lor \psi) \leq 1\)  \( \text{PITL5} \)
3. \(p(\varphi) + p(\psi) > 1 \Rightarrow p(\varphi \land \psi) > 0\)  \( 1, 2 \)

\(P_+':\)
1. \((\varphi; p(\psi) = x) \Rightarrow \exists t((\varphi \land \ell = t; T) \land (\ell = t; p(\psi) = x))\)  \( \text{ITL} \)
2. \((\varphi \land \ell = t; T) \Rightarrow p(\varphi \land \ell = t; T) = 1\)  \( \text{PITL7} \)
3. \((\ell = t; p(\psi) = x) \Rightarrow p(\ell = t; \psi) = x\)  \( P_\leq \)
4. \(p(\varphi \land \ell = t; \psi) = 1 \land p(\ell = t; \psi) = x \Rightarrow p(\varphi \land \ell = t; \psi) = x\)  \( \text{PITL8, PITL1, ITL} \)
5. \((\varphi \land \ell = t; \psi) \Rightarrow (\varphi; \psi)\)  \( \text{ITL} \)
6. \(p(\varphi \land \ell = t; \psi) = x \Rightarrow p(\varphi; \psi) \geq x\)  \( 5, \text{PITL1} \)
7. \(\exists t((\varphi \land \ell = t; T) \land (\ell = t; p(\psi) = x)) \Rightarrow \exists t(p(\varphi; \psi) \geq x)\)  \( 2 - 6, \text{ITL} \)
8. \(\exists t(p(\varphi; \psi) \geq x) \Rightarrow p(\varphi; \psi) \geq x\)  \( \text{ITL} \)
9. \((\varphi; p(\psi) = x) \Rightarrow p(\varphi; \psi) \geq x\)  \( 1, 7, 8 \)
10. \((\varphi; p(\psi) = x) \Rightarrow (\varphi; p(\neg \psi) = 1 - x)\)  \( \text{PITL2, ITL} \)
11. \((\varphi; p(\neg \psi) = 1 - x) \Rightarrow p(\varphi; \neg \psi) \geq 1 - x\)  \( \text{like 1 - 9, but with } \neg \psi \text{ as } \psi \)
12. \((p(\varphi; \psi) > x \land p(\varphi; \neg \psi) \geq 1 - x) \lor (p(\varphi; \psi) \geq x \land p(\varphi; \neg \psi) > 1 - x)\)  \( \Rightarrow p((\varphi; \psi) \land (\varphi; \neg \psi)) > 0\)  \( \text{PITL10} \)
13. \((\varphi; \psi) \land (\varphi; \neg \psi) \land (\neg (\varphi; \ell = 0); T) \Rightarrow \bot\)  \( \text{ITL} \)
14. \(p((\varphi; \psi) \land (\varphi; \neg \psi) \land (\neg (\varphi; \ell = 0); T)) = 0\)  \( 13, \text{PITL6} \)
15. \(p(\neg (\varphi \land (\varphi; \ell = 0); T)) = 1\)  \( \text{assumption, PITL6} \)
16. \(p((\varphi; \psi) \land (\varphi; \neg \psi)) = 0\)  \( 14, 15, \text{PITL8} \)
17. \(p(\varphi; \neg \psi) \leq 1 - x \land p(\varphi; \psi) \geq x \Rightarrow p(\varphi; \psi) \leq x \land p(\varphi; \neg \psi) \leq 1 - x\)  \( 12, 16, \text{ITL} \)
18. \((\varphi; p(\psi) = x) \Rightarrow p(\varphi; \psi) = x\)  \( 9, 11, 17, \text{ITL} \)
A.5 The rule Seq

In the proof of the admissibility of Seq below \( \varphi \land \ell \geq l \land \ell \leq h \) is abbreviated by \( \varphi^\ell_l \).

1. \( (\ell = 0 \land p(\alpha; \beta) = 1; \top) \Rightarrow q(p(\alpha; \beta) = 1 \land \ell = 0; \top) = 1 \) \( \text{PITL} \)
2. \( (\alpha; p(\beta; \top) = x \land \ell = 0) \Rightarrow p(\alpha; \beta; \top) = x \) \( P^\ell \), assumptions
3. \( \exists x((\alpha; \top) \Rightarrow (\alpha; p(\beta; \top) = x \land \ell = 0; \top)) \) \( \text{ITL} \)
4. \( p(\alpha; \beta; \top) = 1 \Rightarrow \exists x((\alpha; \top) \Rightarrow p(\alpha; \beta; \top) = x \land \ell = 0; \top) \land p(\alpha; \beta; \top) = 1) \) \( \text{2, 3, ITL} \)
5. \( p(\alpha; \beta; \top) = 1 \Rightarrow \exists x((\alpha; \top) \Rightarrow (p(\alpha; \beta; \top) = 1 \land \ell = 0; \top)) \) \( 4, \text{ITL} \)
6. \( p(\alpha; \beta; \top) = 1 \land \ell = 0; \top \Rightarrow (p(\alpha; \beta; \top) = 1 \land \ell = 0; \top) = 1 \) \( 5, \text{ITL} \)
7. \( p(\alpha; \beta; \top) = 1 \land p(\alpha; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \land \ell = 0; \top) = 1 \) \( 7, \text{PITL9} \)
8. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \land \ell = 0; \top) = 1 \) \( 1, 9 \)
9. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \land \ell = 0; \top) = 1 \) \( \text{PITL9} \)
10. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \land \ell = 0; \top) = 1 \) \( \text{9, 10} \)
11. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \land \ell = 0; \top) = 1 \) \( \text{PITL8} \)
12. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \land \ell = 0; \top) = 1 \) \( \text{16, 17, ITL} \)
13. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \land \ell = 0; \top) = 1 \) \( \text{15, 18, PITL6, PITL9} \)
14. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \land \ell = 0; \top) = 1 \) \( \text{ITL} \)
15. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha; p(\beta; \top) = 1 \land \ell = 0; \top) = 1 \) \( \text{11, 14} \)
16. \( \ell = 0 \Rightarrow (p(\beta; \top) \neq x_2) \Rightarrow (p(\beta; \top) \neq 1) \) \( \text{ET}_{\beta} \in [l_2, h_2] \)
17. \( \alpha_i^h \Rightarrow \alpha \) \( \text{ITL} \)
18. \( (\alpha_i^h; p(\beta_i; \top) \neq x_2 \land \ell = 0; \top) \Rightarrow (\alpha; p(\beta; \top) \neq 1 \land \ell = 0; \top) \) \( \text{16, 17, ITL} \)
19. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha_i^h; p(\beta_i; \top) \neq x_2 \land \ell = 0; \top) = 0 \) \( \text{15, 18, PITL6, PITL9} \)
20. \( \neg(\alpha \land (\alpha \neq 0) \land (\ell = 0); \top) \Rightarrow ((\alpha_i^h; p(\beta_i; \top) \neq x_2 \land \ell = 0; \top) \Rightarrow (\alpha_i^h; \top) \land (\alpha; p(\beta_i; \top) \neq x_2 \land \ell = 0; \top)) \) \( \text{ITL} \)
21. \( p(\neg(\alpha \land (\alpha \neq 0) \land (\ell = 0); \top)) = 1 \) \( \text{20, 21, PITL9} \)
22. \( p(\alpha_i^h; p(\beta_i; \top) \neq x_2 \land \ell = 0; \top) = 0 \Rightarrow p(\alpha_i^h; \top) \land (\alpha; p(\beta_i; \top) \neq x_2 \land \ell = 0; \top) = 0 \) \( \text{22, 21, PITL9} \)
23. \( \alpha \land p(\alpha; \beta; \top) \neq x_2 \Rightarrow \neg(\alpha; p(\beta; \top) = x_2 \land \ell = 0) \) \( \text{P}^\ell \)
24. \( \neg(\alpha; p(\beta; \top) = x_2) \land \alpha \Rightarrow (\alpha; p(\beta; \top) \neq x_2 \land \ell = 0) \) \( \text{ITL} \)
25. \( \alpha \land p(\alpha; \beta; \top) \neq x_2 \Rightarrow (\alpha; p(\beta; \top) \neq x_2 \land \ell = 0) \) \( 23, 24 \)
26. \( p(\alpha_i^h; \top) \land (\alpha; p(\beta_i; \top) \neq x_2 \land \ell = 0; \top) = 0 \Rightarrow p(\alpha_i^h; \top) \land (\alpha \land p(\alpha; \beta_i; \top) \neq x_2; \top) = 0 \) \( \text{25, PITL9, PITL6} \)
27. \( \neg(\alpha \land (\alpha \neq 0) \land (\ell = 0); \top) \Rightarrow ((\alpha_i^h; \top) \land (\alpha \land p(\alpha; \beta_i; \top) \neq x_2; \top) \Rightarrow (\alpha_i^h; \top) \land (\alpha; p(\alpha; \beta_i; \top) \neq x_2; \top))) \) \( \text{ITL} \)
28. \( p(\neg(\alpha_i^h; \top) \land (\alpha \land p(\alpha; \beta_i; \top) \neq x_2; \top) ; \top) = 0 \Rightarrow p(\alpha_i^h; \top) \land (\alpha; p(\alpha; \beta_i; \top) \neq x_2; \top) = 0 \) \( \text{21, 27, PITL9} \)
29. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha_i^h; \top) \land (\alpha; p(\alpha; \beta_i; \top) \neq x_2; \top) = 0 \) \( 19, 22, 26, 28 \)
30. \( \ell = 0 \land p(\alpha_i^h; \top) \land (\alpha; p(\beta_i; \top) \neq x_2; \top) = 0 \Rightarrow p(\alpha_i^h; \top) \land (\alpha \land p(\beta_i; \top) \neq x_2; \top) = x_2 \) \( \text{P}^\ell \), assumptions
31. \( \neg(\alpha \land (\alpha \neq 0) \land (\ell = 0); \top) \Rightarrow ((\alpha_i^h; \top) \land (\alpha \land p(\beta_i; \top) \neq x_2; \top) \Rightarrow (\alpha_i^h; \top) \land (\alpha; p(\beta_i; \top) \neq x_2; \top))) \) \( \text{ITL} \)
32. \( (\alpha_i^h; \top) \land (\alpha \land p(\beta_i; \top) \neq x_2; \top) \Rightarrow (\alpha_i^h; \top) \land (\alpha; p(\beta_i; \top) \neq x_2; \top) \) \( \text{31, PITL1} \)
33. \( p(\alpha_i^h; \top) \land (\alpha \land p(\beta_i; \top) \neq x_2; \top) = p(\alpha_i^h; \top) \land (\alpha \land p(\beta_i; \top) \neq x_2; \top) \) \( \text{32, PITL1} \)
34. \( \ell = 0 \land p(\alpha; \top) = 1 \Rightarrow p(\alpha_i^h; \top) = x_1 \) \( \text{[ET}_\alpha \in [l_1, h_1] \})
35. \( \ell = 0 \land p(\alpha; \beta; \top) = 1 \Rightarrow p(\alpha_i^h; \top) \land (\alpha \land p(\beta_i; \top) \neq x_2, x_1) \)