SOFTWARE VERIFICATION RESEARCH CENTRE

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 02-31

Linear Approximation of Execution Time Constraints

Karl Lermer, Colin Fidge and Ian Hayes

October 2002
Version 1

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533
http://svrc.it.uq.edu.au

Linear Approximation of Execution Time Constraints

Karl Lermer[†], Colin Fidge[†] and Ian Hayes[‡]

[†]Software Verification Research Centre, The University of Queensland, Queensland 4072, Australia.
[‡]School of Information Technology and Electrical Engineering, The University of Queensland, Queensland 4072, Australia.

Keywords: Real-time program analysis; Timing-path analysis; Timing prediction; Worst and best case execution times; Automatic constraint determination; Predicate transformer semantics.

Abstract. This paper defines an algorithm for predicting worst and best case execution times, and determining execution time constraints, of control-flow paths through real-time programs using their partial correctness semantics. The algorithm produces a linear approximation of path traversal conditions, worst and best case execution times and strongest postconditions for timed paths in abstract real-time programs. We further derive techniques to determine the set of control-flow paths with decidable worst and best case execution times. The approach is based on a weakest liberal precondition semantics and relies on supremum and infimum calculations similar to standard computations from Linear Programming and Presburger Arithmetic. The methodology is generic in that it is applicable to any executable language that can be supplied with a predicate transformer semantics and hence provides a verification basis for high level as well as assembler level execution time analysis techniques.

1. Introduction

Programming real-time systems is extremely challenging. The functional behaviour of a program is immediately obvious in its source code. However its timing behaviour depends on the compiler's code generation strategy and the run-time environment, and is therefore defined incompletely, if at all, by the source program. Real-time programmers are therefore forced to carefully analyse their programs, to ensure that they have the correct timing behaviour. Typically this involves identifying each possible control-flow path through the program, determining the essential timing constraints on it, and defining an enforceable bound on the worst case execution time of the object code sufficient to ensure that the timing constraints will always be satisfied. Our overall goal is to automate as much of this process as possible.

As a motivational example, consider the simple message transmitter program depicted in Figure 1 [GHF98], which displays a message of N characters, one character at a time, in a shared memory location, under the constraint that the first character must be displayed from (at least) time 5 to time 11, the

Fig. 1. The message transmitter and its four timing paths.

```
assume τ ≤ 3 ;        — τ stands for the current time
 (var n : ℕ •
   n := 0;
   do n ≤ N − 1 →
     write(msg(n), out);
     deadline 5 + 10 ∗ n ;        — hard time deadline
     delay until 11 + 10 ∗ n;
     n := n + 1
   od);
deadline 5 + 10 ∗ N
```

– Path A (loop entry)

```
assume τ ≤ 3;
(var n : ℕ •
n := 0;
coerce n ≤ N − 1;
write(msg(n), out);
deadline 5 + 10 ∗ n)
```

– Path B (single iteration)

```
(var n : ℕ •
deadline 5 + 10 ∗ n;
delay until 11 + 10 ∗ n;
n := n + 1;
coerce n ≤ N − 1;
write(msg(n), out);
deadline 5 + 10 ∗ n)
```

– Path C (loop bypass)

```
assume τ ≤ 3;
(var n : ℕ •
n := 0;
coerce n ≥ N);
deadline 5 + 10 ∗ N
```

– Path D (loop exit)

```
(var n : ℕ •
deadline 5 + 10 ∗ n;
delay until 11 + 10 ∗ n;
n := n + 1;
coerce n ≥ N);
deadline 5 + 10 ∗ N
```

second character must be displayed from time 15 to time 21, and so on. Thus a new character appears every 10 milliseconds, and remains visible for (at least) 6 milliseconds. The *delay until* statement forces the character to remain visible for the required amount of time, and the *deadline* states that the program must have produced the character by the required time. Deadline statements formally document critical timing points in a program [FHW99]. The variable τ represents the current time and the initial assumption states that this is before time 3.

A path extraction algorithm [GHF98] can identify the four timed paths of Figure 1 as primitives for timing analysis of the transmitter program. The deadline command in the loop body makes it possible to cut the loop into four primitive control-flow paths that are sufficient for timing analysis of this program fragment. There is no need to assume maximal or minimal loop iteration bounds. Each of these control-flow paths represents a possible execution of the program from an initial assumption or deadline to a following deadline. An implementation of the transmitter that fulfils the deadlines of these timed paths would guarantee the deadlines of the original program. Hayes et al. [GHF98] explain how to deduce a verifiable timing constraint from each of the four timed paths with the help of the underlying predicate transformer semantics. These constraints can be ultimately used in the timing analysis of the corresponding machine code where worst case execution times for assembler instruction sequences are checked against the timing constraints. For instance, the end-to-end timing constraint for Path A is 2. If the worst case execution time of the corresponding machine code is less than or equal to 2 milliseconds, it is certain that the high-level deadline will always be satisfied.

However, this abstract, semantics-based approach does not permit automatic computation of the timing constraints from the semantics of the timed path. To solve this problem we defined an abstract program path language [HFL01, LFH02b], gave it a partial correctness semantics and derived a theory for the computation of execution times and execution time constraints for program paths on the base of their semantics [LFH02c, LFH02a]. In this paper we present an implementation of these techniques. In particular, we state algorithms that allow one to automatically derive execution times and execution time constraints for program paths

via their semantics. Our general semantics-based approach provides a universal derivation, verification and validation technique for execution time approximation, estimation and computation methods, since low-level and high-level program analysis techniques must always rely on some kind of underlying program semantics. In particular this enables us to compute the execution times and execution time constraints for the timing paths of the transmitter example.

In Section 3.1 we define the syntax and the weakest liberal precondition semantics of timed program paths. The abstract algorithm for the computation of path traversal conditions and worst and best case execution times is then stated in a theorem in Section 3.2. In Section 3.3 we give a formal program path semantics for the transmitter example from the introduction. In addition, we introduce the notions of weakest and strongest execution time constraints for timed paths. We outline how weakest and strongest execution time constraints can be used in the timing analysis of a real-time program in our language and derive execution times and constraints for the transmitter paths. A strongest postcondition semantics for timed paths is defined in Section 3.4. The main part of this paper focuses on the automated computation of timing constraints. In Section 4.1 we show how first-order predicates can be projected onto linear predicates and how this can be used in the approximation of path traversal conditions and strongest postconditions. The automation of worst and best case execution time computations is investigated in Section 4.2. We show how the predicates that are involved in the computation of worst and best case execution times can be reduced to decidable fractions. Section 4.3 presents an algorithm for the approximation of worst and best case execution times. Finally, Section 4.4 contains an algorithm for the approximation of weakest and strongest execution time constraints. An appendix at the end of the paper provides some basic theorems and proofs.

2. Related work

There are essentially two different approaches to timing analysis of real-time programs. First there is the traditional worst case execution time (WCET) analysis of machine code programs based on 'cycle counting' of basic code blocks comprising linear instruction sequences. This is complemented by the more recent approach that partitions timing analysis of a real-time program into a high-level and a low-level phase. The high-level analysis phase derives timing constraints on program paths on the base of the high-level program semantics, and the low-level phase checks these constraints against WCETs of corresponding paths in the machine code. A variety of different, but related, techniques have been applied to the problem. Particular challenges that face all approaches are: how to syntactically extract control-flow paths from the program; how to recognise and eliminate semantically 'infeasible' paths [Alt96]; and how to put a bound on the number of iterations for loops.

Early work applied timing analysis to high-level language real-time programs [PK89]. In this approach worst case execution times are computed with the help of user provided annotations. Necessary requirements are that programs do not contain recursion in any form and for every loop either an upper iteration bound or an upper execution time bound must be known. Nassi-Schneidermann diagrams and formulas for basic program constructs are used to represent the programs and compute worst case execution time bounds.

Park's work [Par93] is one of the most cited contributions to path analysis and computation of worst case execution times. Park defines the language IDL (Information Description Language) that allows the programmer to provide user annotations along with the labelled source program. Static path analysis of the program leads to a set of possible static program paths which are restricted by IDL statements. It is thus possible to exclude some infeasible paths, but not necessarily all of them. Of course, user provided annotations can be wrong, hence should be checked thoroughly. Park addresses this verification problem in his thesis [Par92] with the help of Hoare triples and weakest precondition techniques. The computation of worst case execution times is performed with the help of so-called timing schemas [Sha89, PS90], which are essentially tables providing lower and upper bounds for each programming statement.

Our approach to timing analysis is close in spirit to Chapman's comprehensive and practical work [Cha95] which combines functional and timing analysis and advocates the use of timing annotations. The elimination of infeasible paths and the computation of accurate worst case execution times is again based on user provided annotations. The SPATS tool was designed in this methodology to allow the extraction of worst case execution times on programs in a certain safe subset of SPARK Ada [CBW94, CBW96].

Puschner and Schedl [PS97] transform the real-time program into a timed graph (T-graph), a control flow graph supplied with worst case execution times and maximal iteration times for each statement. Computing worst case execution times is transformed into the graph theoretical problem of finding the circulation

with maximal cost in the T-graph. Integer Linear Programming techniques are used to solve the resulting maximisation problem. The overall assumptions on the program are that all feasible paths are known and that for each part of the program its maximum number of repetitions is provided. It is hence up to the user to supply all information about infeasible paths and maximal iteration bounds for each part of the program. The authors prove that any program fulfilling the above mentioned assumptions can be transformed into a T-graph with a corresponding set of capacity constraints (linear inequalities having at least as many unknowns as different edges in all feasible paths). This program transformation is completed by relying on information about the program structure, infeasible/feasible paths and iteration frequencies of different program parts. Note that the search for a sufficient set of capacity constraints can be done algorithmically and standard linear programming software packages can be used to solve the resulting maximisation problem.

Li and Malik [LM95] also base their methodology for the computation of best and worst case execution times of real-time programs on linear programming methods. Their approach is closely related to Puschner and Schedl's [PS97] and similar techniques are applied. Adequate disjoint sets of program paths are identified and a linear programming problem is constructed for each path set. The worst case execution time of each path set is computed and the maximum over all path sets is taken as the worst case execution time of the entire program. Infeasible paths can be identified by means of contradicting linear equations which correspond to a certain equivalence class of paths. This means that the user has to provide all the information before the linear programming solver can work. The approach incorporates hardware features such as instruction caches and pipelining effects into the timing analysis [LMW95] and the timing analysis tool Cinderella [LMW97] includes cache memory and pipelining estimates.

Altenbernd [Alt96] provides an algorithm for the computation of the longest executable path in a sequential high-level language program. The algorithm is based on the knowledge of loop bounds (maximum loop counts), maximum function recursion depths and the execution times of basic blocks. This approach combines static path analysis with symbolic simulation of the program. The latter is applied to detect infeasible paths. The paper gives no details of how the symbolic simulation of the program is performed and does not remark on the efficiency of the proposed algorithm. Importantly the algorithm is not safe in the sense that for certain inputs it can produce overly optimistic worst case execution times.

The program timing analyser (PTA) tool [SA00] is based on Altenbernd's algorithm [Alt96] and incorporates low-level and high level aspects, such as caching, pipelining and extraction of infeasible paths. The tool determines worst case execution times for real-time programs that contain no loops, no procedure calls and only static variables. PTA is based on a simple program simulation technique where the program is symbolically executed, variable values are calculated wherever possible, 'unknown' values are used otherwise and ranges (upper and lower bounds) of variables are determined by analysing boolean guards of 'if' statements.

Ermedahl and Gustafsson [EG97] define the semantics of a (imperative) program as the environment that can be created by executing the program. Timing analysis and calculation of worst case execution times is then performed through semantic evaluation of the program. For this the program is systematically supplied with break points, and values or value ranges are then assigned to each variable (if not possible, the simulator uses 'unknown') and systematically updated at break points. Any 'if' statement leads to two different updates (so-called split variable ranges, such as $v \rightarrow 1..5$ or $3..4$) and loops are unrolled to iterated 'if' statements which either terminate or exceed their upper (timing) bounds. The analysis method does not allow the expression of conditions relating different variables. Because of this, and unknown values, only rough approximations can be expected. For instance, only some infeasible paths can be automatically detected. Closely related to this simulation approach via the program semantics is Lundquist and Stenström's path algorithm [LS98]. Their worst case execution times are computed by symbolic simulation of the (high or low-level) program. This seems less effective than Ermedahl and Gustafsson's algorithm [EG97] since no variable ranges are respected, and variables are either of a certain value or 'unknown'. Nevertheless, the algorithm allows merging of program paths during the analysis which increases its efficiency. The authors furthermore mention how caching and pipelining effects can be incorporated in their analysis method [LS98].

Liu and Gomez [LG98] show how the semantics of a functional programming language can be extended to enable automatic timing analysis via semantic evaluation. So-called time bound functions are added to the semantics of the functional programming language for the computation of worst case execution times on feasible paths. Partially known input values and values of program variables are simulated (again with 'unknown' if no specific value can be determined). Thanks to the functional approach only single values rather than value ranges have to be considered for each variable. No loop bounds or bounds on recursion depths are required in this setting.

Lim, et al. [LBJ+95] extend timing schemas [Sha89, PS90] to the timing analysis of programs running on

RISC processors. The extended timing schema is called worst case timing abstraction (WCTA). It integrates caching (instruction and data) and pipelining effects into the analysis method with timing schemas. The user has to provide loop bounds and iteration bounds for the program under analysis. The analysis of loops is reduced to a graph theoretical problem and both, exact as well as approximate, solutions are given for computing the WCTA of a loop. The approximate solution is based on the maximum cycle mean (the maximum over all mean weights of cycles in the graph) of a weighted graph and importantly, is independent of the loop bound. Worst case timing analysis for optimised assembler code is also investigated [LKM98]. The analysis method is based on Park's extended timing schemas (ETS) and tackles the problem with the help of user provided annotations. Importantly, infeasible paths cannot be detected with this method.

Gunter and Peled [GP99] present a path exploration tool (PET) that allows the user to select a path from a linear or parallel program and compute the corresponding 'path traversal condition'. The programming language is a C hybrid consisting of assignments, test statements (guards), wait statements (semaphores for concurrent programs) and case statements. All variables are assumed to be integer valued and expressions are built from integer arithmetic. The computation of the path traversal condition is performed from the end to the beginning of the path by iterated substitution and conjunction of predicates. The tool has an interface to the HOL theorem prover for expression simplification and predicate computation. For instance, satisfiability and validity of predicates in Presburger Arithmetic can be checked with the help of the HOL prover.

Lo Ko, et al. [LAYH+99] describe an environment that permits the user to specify best and worst case execution constraints in a C program and to check whether the corresponding machine code meets these time bounds. Upper and lower time bounds can be specified for functions, loops and the branches of "if" clauses. The user is then permitted to select a program path for analysis. The program analyser then computes worst and best case specified execution times for the high-level path. If possible the tool automatically derives all basic blocks (with possibly several machine instructions) for the corresponding path in the machine code and computes the predicted best and worst case execution times. The tool finally compares the specified execution times with the predicted ones and determines whether upper and lower time bounds are met. Note that it may not always be possible to find the low-level program path that corresponds to the high-level path due to compiler optimisations. In such a case it is the responsibility of the user to select the right low-level code by hand. The low-level timing prediction tool may then compare the predicted and specified execution times. Note that basic blocks are atomic entities for the timing analyser and if code optimisations move instructions across basic blocks the comparison of high and low-level paths can be impossible.

Based on this previous work our goal is to devise a general semantics-oriented algorithm for identifying timing constraints and predicting execution times. We begin with a simple, generic language for modelling control-flow paths in real-time programs and then define timing analysis algorithms for this path language. This provides a basis for timing analysis of any language which can be defined in terms of the path language primitives.

3. Execution times and timing constraints in abstract real-time programs

This section presents a simplified version of a theory for program paths and execution time constraints in abstract real-time programs [LFH02b, LFH02c]. The approach is based on a partial correctness semantics that permits the computation of path traversal conditions, worst and best case execution times and execution time constraints. The framework of this section builds the base for the implementation that we outline in the following section. As an example we perform the timing analysis of the transmitter program of the introduction.

3.1. Weakest liberal precondition semantics for timed paths

Table 1 states the syntax and weakest liberal precondition semantics of a timed path. For a path S it also defines the identifiers of a path $Idf(S)$ and the syntactic substitution of a variable x by a variable z, $S[z/x]$. This path language conforms with one presented earlier [LFH02c] but does not provide for nondeterministic choice. We omit explicit nondeterministic choice since worst (best) case execution times for nondeterministic constructs can be handled by maximisation (minimisation) over primitive paths. Let x be a list of program variables; T be a type; S be a statement in our path language; I be a predicate on the program state; and

Table 1. Weakest liberal precondition (wlp) semantics for timed paths.

Command S	Weakest liberal precondition $wlp(S, R)$	Identifier $Idf(S)$
$(x\colon [Q])^I$	$\forall x', \tau' ((I \wedge I[x', \tau'/x, \tau] \wedge Q \wedge \tau \leq \tau') \Rightarrow R[x', \tau'/x, \tau])$	all free variables in Q, I and x, x', τ, τ'
$(\mathbf{var}\, v : T \bullet S_1)^I$	$\forall w\, (wlp(S_1[w/v]^{I \wedge w \in T}, R))$ with $w \notin Idf(S) \cup Fv(R)$	$Idf(S_1^{I \wedge v \in T}) \cup \{v, v'\}$
$(S_1^{I_1})^{I_2}$	$wlp(S_1^{I_1 \wedge I_2}, R)$	$Idf(S_1^{I_1 \wedge I_2})$
$(S_1 \,;\, S_2)^I$	$wlp(S_1^I, wlp(S_2^I, R))$	$Idf(S_1^I) \cup Idf(S_2^I)$

Command S	Syntactic substitution $S[z/x]$ with $z \notin Idf(S)$
$(y, x\colon [Q])^I$	$(y, z\colon [Q[z, z'/x, x']])^{I[z/x]}$
$(\mathbf{var}\, x : T \bullet S_1)^I$	$(\mathbf{var}\, z : T \bullet S_1[z/x])^{I[z/x]}$
$(S_1^{I_1})^{I_2}$	$S_1^{I_1 \wedge I_2}[z/x]$
$(S_1 \,;\, S_2)^I$	$S_1^I[z/x] \,;\, S_2^I[z/x]$

Q be a predicate on undecorated (prestate) variables x, and primed (post state) variables x'. The basic path constructors are then the timed specification statement, $x\colon [Q]$, [HU97, HU01], the variable declaration, $(\mathbf{var}\, x : T \bullet S_1)$ and the sequential composition of program paths, $(S_1 \,;\, S_2)$. Any path S may be seen in a certain program context predicate I, indicated by S^I. Although trivial, this language is sufficient to model linear control-flow paths through imperative program code. The specification statement models (possibly nondeterministic) updates to the program state, the variable declaration statement allows the state space to be changed, sequential composition is the basic constructor of paths from subpaths, and context annotations allow paths to inherit information from the surrounding code. Commands in an actual real-time language, such as assignment, delays and deadlines may be defined in terms of an equivalent specification statement (see Table 3 in Section 3.3).

The semantics of a timed path is based on a traditional predicate transformer semantics with predicates ranging over a first-order language that allows for arithmetic on the real numbers $\mathbb{R}$. We assume that predicates have free variables in the abstract variable space *Var*. The first-order predicates *Pred* are interpreted as boolean valued functions on the set of bindings *Bnd*, i.e., the set of all functions from *Var* with values in an abstract universe *Val*. Predicates may have unprimed and primed free variables representing the state before and after execution of a command, respectively. The free variables of a predicate P are denoted by $Fv(P)$. Substitution of a variable x by a term t in a predicate P is denoted by $P[t/x]$. For two predicates P_1 and P_2 with free variables x we write $P_1 \Rrightarrow P_2$ iff $\forall x(P_1 \Rightarrow P_2)$, and we write $P_1 \equiv P_2$ iff $P_1 \Rrightarrow P_2$ and $P_2 \Rrightarrow P_1$. A predicate transformer is defined as a function from *Pred* to *Pred* that is monotonic with respect to the ordering $\Rrightarrow$ [Dij76, DS90, BvW98]. Table 1 defines the predicate transformer $wlp(S, R)$ for a timed path S in our language. We apply the following rules and conventions.

The expressions Q, R and I in Table 1 denote first-order predicates, where R and I do not have free occurrences of any primed variables and Q may have free primed variables within the identifiers in the list x', τ'. A primed copy of a variable denotes the state of the variable after execution of the statement. The list x is named the frame of the timed specification statement $x\colon [Q]$ and denotes the variables that may be modified by the statement. The identifier τ is a reserved variable which represents the current time, ranging over the domain $\mathbb{R}$. It is implicitly in the frame of a specification statement. That time does not go backward is hardcoded into the semantics of the timed specification statement with the predicate $\tau \leq \tau'$. The timed specification statement underlies a partial correctness semantics based on predicate transformers: given a command S and a predicate R, the weakest liberal precondition $wlp(S, R)$ characterises those initial states from which S achieves R or fails to terminate [Dij76, DS90]. The context I usually determines the types of program variables and values of symbolic constants.

Any statement S^I constructed from commands in Table 1 with a context predicate I shall be henceforth denoted a *timed path* provided $I \equiv I \wedge \tau \in \mathbb{R}$. The context of a timed command thus always ensures that the time variable τ ranges over $\mathbb{R}$. For any two timed paths S_1 and S_2 in Table 1, $S_1 \sqsubseteq_{wlp} S_2$ denotes the situation when $wlp(S_1, R) \Rrightarrow wlp(S_2, R)$ holds for all predicates R without free primed variables. In this case we say that S_1 *liberally refines* to S_2, or S_2 is a *liberal refinement* of S_1. Refinement in this partial correctness semantics may reduce the set of states where the program terminates, but it maintains program behaviour on terminating 'states' by possibly reducing nondeterminism. Although this semantics is insensitive

to nontermination, it is a sufficient basis for timing analysis because execution times can only be computed for terminating program paths.

3.2. An abstract method for the computation of execution times

To capture the duration of a timed path we enhance the timed specification statement with an auxiliary variable δ that ranges over $\mathbb{R}$. Elsewhere we showed that this permits the formal expression of the duration of any sequence of commands from the weakest liberal precondition semantics and computation of best and worst case execution times of timed paths [LFH02c]. In the following we give a simplified version of this theory based on first-order predicates.

Definition 3.1 (Duration variable). For the timed specification statement $x{:}[Q]^I$ and the fresh variable δ, $\delta \notin \mathit{Idf}(x{:}[Q]^I)$ we define the δ-specification statement by

$$d(x{:}[Q]^I) \stackrel{\text{def}}{=} x, \delta{:}[Q \wedge \delta' = \delta - (\tau' - \tau)]^{I \wedge \delta \in \mathbb{R}} .$$

The δ-transformation of a timed path is defined by transforming all timed specification statements into δ-specification statements.

Thus the auxiliary variable δ records the elapsed time required to traverse the path. The following theorem provides an abstract algorithm for the computation of the predicates $\mathit{wlp}(S, \delta \geq 0)$ and $\mathit{wlp}(S, \delta \leq 0)$ for any path which is constructed from δ-specification statements. These expressions alow us then to calculate lower and upper execution time bounds for S.

We first define the notions of infimum and supremum in the logic space that underlies the predicate transformer semantics. Remember that Var denotes the universal set of variable identifiers. Let $\mathcal{V} \subseteq \mathit{Var}$, $B \in \mathit{Pred}$ and θ a function on Bnd with values in $\mathbb{R} \cup \{\infty, -\infty\}$. For $\sigma \in \mathit{Bnd}$, $\sup\{\mathcal{V}|B \bullet \theta\}(\sigma)$ shall denote the supremum of the set

$$\{\theta(b) \ : \ b \in \mathit{Bnd} \wedge B(b) \wedge b(v) = \sigma(v) \text{ for all } v \in \mathit{Var} \setminus \mathcal{V}\}$$

in $\mathbb{R} \cup \{\infty, -\infty\}$. For predicates B_j and functions θ_j, where $j \in \mathcal{J}$, we denote by $\sup_{\mathcal{J}}\{B_j \bullet \theta_j\}(\sigma)$ the supremum of the set

$$\cup_{j \in \mathcal{J}}\{\theta_j(\sigma) \ : \ B_j(\sigma)\} .$$

The case $\sup_{\mathcal{J}}\{\mathit{true} \bullet \theta_j\}$ will be abbreviated by $\sup_{\mathcal{J}} \theta_j$. In other words, $\sup\{\mathcal{V}|B \bullet \theta\}(\sigma)$ denotes the least upper bound of function θ over all bindings, where predicate B holds, that may differ from σ only in the variables $\mathcal{V}$. The expression $\sup_{\mathcal{J}}\{B_j \bullet \theta_j\}(\sigma)$ defines the pointwise supremum over the functions $\theta_j(\sigma)$ where $B_j(\sigma) = \mathit{true}$ holds. The infimum is defined equivalently. Note that the supremum and infimum as defined above are always total functions (not having undefined values) with range $\mathbb{R} \cup \{\infty, -\infty\}$.

Theorem 3.2 (Execution time bounds). Let S be a timed path and let θ, $\psi : \mathit{Bnd} \to \mathbb{R} \cup \{-\infty, \infty\}$ be functions that do not depend on δ, and let A be a predicate without free occurrences of δ and primed variables. Then there is a predicate $\overline{A}$ without free occurrences of δ and functions $\overline{\theta}$, $\overline{\psi} : \mathit{Bnd} \to \mathbb{R} \cup \{-\infty, \infty\}$ not depending on δ such that the following equivalences hold.

$$\mathit{wlp}(d(S), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \geq \theta) \quad \equiv \quad (\delta \in \mathbb{R} \wedge \overline{A}) \Rightarrow \delta \geq \overline{\theta}$$
$$\mathit{wlp}(d(S), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \leq \psi) \quad \equiv \quad (\delta \in \mathbb{R} \wedge \overline{A}) \Rightarrow \delta \leq \overline{\psi}$$

Proof. The proof is by structural induction on the path S. Note that $\sup \emptyset = -\infty$ by definition. The following holds for a specification statement $S \stackrel{\text{def}}{=} x{:}[Q]$,

$$\begin{aligned}
& \mathit{wlp}(d(S^I), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \geq \theta) \\
\equiv \ & \forall \tau', \delta', x' \, ((I \wedge \tau \leq \tau' \wedge \delta' = \delta - (\tau' - \tau) \wedge \delta, \delta' \in \mathbb{R} \wedge Q \wedge \\
& \qquad (I \wedge A)[\tau', x'/\tau, x]) \Rightarrow \delta' \geq \theta[\tau', x'/\tau, x]) \\
\equiv \ & \forall \tau', x' \, ((I \wedge \tau \leq \tau' \wedge Q \wedge \delta \in \mathbb{R} \wedge (I \wedge A)[\tau', x'/\tau, x]) \Rightarrow \delta \geq \theta[\tau', x'/\tau, x] + \tau' - \tau) \\
\equiv \ & \forall x', \tau' \, ((A_1 \wedge \delta \in \mathbb{R}) \Rightarrow \delta \geq \sup\{\{x', \tau'\}|A_1 \bullet \theta[\tau', x'/\tau, x] + \tau' - \tau\})
\end{aligned}$$

Table 2. The computation of $\Theta(S, A)$ and $\Upsilon(S, A, \theta)$.

Command S	Path traversal condition $\Theta(S, A)$	Worst case execution time $\Upsilon(S, A, \theta)$
$(x{:}\,[Q])^I$	$\exists \tau', x' \,(I \wedge (I \wedge A)[x', \tau'/x, \tau] \wedge$ $Q \wedge \tau \leq \tau')$	$\sup\{\{\tau', x'\} \mid I \wedge (I \wedge A)[x', \tau'/x, \tau] \wedge Q \wedge \tau \leq \tau' \bullet$ $\theta[x', \tau'/x, \tau] + \tau' - \tau\}$
$(\mathbf{var}\, x : T \bullet S_1)^I$	$\exists w \,(\Theta(S_1[w/v]^{I \wedge w \in T}, A))$ with $w \notin \mathit{Idf}(S) \cup \mathit{Fv}(A)$	$\sup\{\{w\} \mid \Theta(S_1[w/v]^{I \wedge w \in T}, A) \bullet \Upsilon(S_1[w/v]^{I \wedge w \in T}, A, \theta)\}$ with $w \notin \mathit{Idf}(S) \cup \mathit{Fv}(A)$
$(S_1^{I_1})^{I_2}$	$\Theta(S_1^{I_1 \wedge I_2}, A)$	$\Upsilon(S_1^{I_1 \wedge I_2}, A, \theta)$
$(S_1 \,;\, S_2)^I$	$\Theta(S_1^I, \Theta(S_2^I, A))$	$\Upsilon(S_1^I, \Theta(S_2^I, A), \Upsilon(S_2^I, A, \theta))$

$$\equiv \quad (\delta \in \mathbb{R} \wedge \exists \tau', x' A_1) \Rightarrow \delta \geq \overline{\theta}$$
$$\equiv \quad (\delta \in \mathbb{R} \wedge \overline{A}) \Rightarrow \delta \geq \overline{\theta}$$

where

$$A_1 \stackrel{\text{def}}{=} I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A)[\tau', x'/\tau, x]$$
$$\overline{A} \stackrel{\text{def}}{=} \exists \tau', x' \, A_1$$
$$\overline{\theta} \stackrel{\text{def}}{=} \sup\{\{\tau', x'\} \mid A_1 \bullet \theta[\tau', x'/\tau, x] + \tau' - \tau\} \; .$$

The induction step for sequential composition $(S_1 \,;\, S_2)^I$ is as follows. By the induction hypothesis for S_1^I and S_2^I we can assume that there are predicates A_1, A_2 and functions θ_1, θ_2 such that the following equivalences hold.

$$\mathit{wlp}(d(S_2^I), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \geq \theta) \quad \equiv \quad (\delta \in \mathbb{R} \wedge A_2) \Rightarrow \delta \geq \theta_2$$
$$\mathit{wlp}(d(S_1^I), (\delta \in \mathbb{R} \wedge A_2) \Rightarrow \delta \geq \theta_2) \quad \equiv \quad (\delta \in \mathbb{R} \wedge A_1) \Rightarrow \delta \geq \theta_1$$

From this we can conclude the following equivalences.

$$\begin{aligned} \mathit{wlp}(d((S_1 \,;\, S_2)^I), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \geq \theta) \quad &\equiv \quad \mathit{wlp}(d(S_1^I), \mathit{wlp}(d(S_2^I), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \geq \theta)) \\ &\equiv \quad \mathit{wlp}(d(S_1^I), (\delta \in \mathbb{R} \wedge A_2) \Rightarrow \delta \geq \theta_2) \\ &\equiv \quad (\delta \in \mathbb{R} \wedge A_1) \Rightarrow \delta \geq \theta_1 \end{aligned}$$

Next, we show the induction step for variable declarations $S \stackrel{\text{def}}{=} (\mathbf{var}\, v : T \bullet S_1)^I$. Let w be a fresh variable such that $w \notin \mathit{Fv}(R) \cup \mathit{Idf}(S)$. The induction hypothesis for $S_1[w/v]^{I \wedge w \in T}$ then provides us with a predicate A_1 and a function θ_1 such that the following equivalence holds.

$$\mathit{wlp}(d(S_1[w/v]^{I \wedge w \in T}), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \geq \theta) \quad \equiv \quad (\delta \in \mathbb{R} \wedge A_1) \Rightarrow \delta \geq \theta_1 \tag{1}$$

This permits us to conclude the following chain of equivalences,

$$\begin{aligned} \mathit{wlp}(d(S), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \geq \theta) \quad &\equiv \quad \text{“By Table 1”} \\ &\quad\; \forall w \,(\mathit{wlp}(d(S_1[w/v]^{I \wedge w \in T}), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \geq \theta)) \\ &\equiv \quad \text{“By induction hypothesis (1)”} \\ &\quad\; \forall w \,((\delta \in \mathbb{R} \wedge A_1) \Rightarrow \delta \geq \theta_1) \\ &\equiv \quad \text{“By taking the supremum over all possible values of } w \text{ ”} \\ &\quad\; \forall w \,((\delta \in \mathbb{R} \wedge A_1) \Rightarrow \delta \geq \sup\{\{w\} \mid A_1 \bullet \theta_1\}) \\ &\equiv \quad \text{“Since } w \text{ is not free for the sup expression”} \\ &\equiv \quad (\delta \in \mathbb{R} \wedge \overline{A}) \Rightarrow \delta \geq \overline{\theta} \end{aligned}$$

where $\overline{A} \stackrel{\text{def}}{=} \exists w \, A_1$ and $\overline{\theta} \stackrel{\text{def}}{=} \sup\{\{w\} \mid A_1 \bullet \theta_1\}$. The second part of our assertion can be proven by reversing the inequality signs and by replacing all ‘supremum’ computations by ‘infimum’ computations in the above proofs. Note that the above defined predicates $\overline{A}$ and A_1 are identical for the supremum and infimum computations. □

Table 3. A simple real-time language

Command S	Equivalent command
x `:=` E	$x{:}\,[x' = E]$
`assume` P	$:[P \wedge \tau = \tau']$
`coerce` G	$:[G]$
`deadline` D	$:[\tau \leq D \wedge \tau = \tau']$
`delay until` D	$:[D \leq \tau' \leq \max\{\tau, D\} + 1]$
`write` (x, ω)	$\omega{:}\,[\omega' = x]$

Definition 3.3 (Execution time bounds). Let S be a timed path, let θ, $\psi : Bnd \to \mathbb{R} \cup \{-\infty, \infty\}$ and let A be a predicate without free primed variables. The unique predicate $\overline{A}$ and expressions $\overline{\theta}$, $\overline{\psi}$ that are computed in Theorem 3.2 for $wlp(d(S), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \geq \theta)$ and $wlp(d(S), (\delta \in \mathbb{R} \wedge A) \Rightarrow \delta \leq \psi)$ will be denoted by $\Theta(S, A)$, $\Upsilon(S, A, \theta)$ and $\Lambda(S, A, \psi)$. We abbreviate $\Theta(S, \mathit{true})$ by $\Theta(S)$ and call it the *path traversal condition* of S. Similarly, $\Upsilon(S, \mathit{true}, 0)$ and $\Lambda(S, \mathit{true}, 0)$ are abbreviated by $\Upsilon(S)$ and $\Lambda(S)$ and are called the *worst case execution time* and the *best case execution time* of path S, respectively. Path S is called a *dead* path if $\Theta(S) \equiv \mathit{false}$. Path S is called *unrestricted from below* (*above*) if $\Lambda(S) = 0$ ($\Upsilon(S) = \infty$).

Table 2 states the abstract algorithm for the computation of the 'path traversal' predicate $\Theta(S, A)$ and the 'worst case execution time' function $\Upsilon(S, A, \theta)$ on the base of the proof of Theorem 3.2. The algorithm for the computation of the 'best case execution time' function $\Lambda(S, A, \theta)$ can be derived from the one for $\Upsilon(S, A, \theta)$ by just replacing the supremum with the infimum. The following heuristics apply: from how predicate $\Theta(S)$ is computed in Theorem 3.2 the equivalence $\neg\Theta(S) \equiv wlp(S, \mathit{false})$ can be derived which shows that $\Theta(S)$ defines the path traversal condition for path S in the sense that it characterises initial states from which path S *will* terminate [DS90, GP99, Car89, HFL01, LFH02c]. The constraints within the path permit S to take up at most $\Upsilon(S)$ time units; hence $\Upsilon(S)$ gives an upper bound on the execution time of S. Similarly, S takes up at least $\Lambda(S)$ time units and therefore, $\Lambda(S)$ gives a lower bound on the execution time of path S. Both expressions depend on the values of the program variables before the path is entered [LFH02c].

With the help of Theorem 3.2 and Table 2 it is then straightforward to prove that the relations $\neg\Theta(S) \Rrightarrow (\Upsilon(S) = -\infty \wedge \Lambda(S) = \infty)$ and $\Lambda(S) \geq 0$ and $\Theta(S) \Rrightarrow \Lambda(S) \leq \Upsilon(S)$ hold. Furthermore, the following monotonicity properties are true.

Corollary 3.4 (Constraint monotonicity). Let S be a timed path, A_1 and A_2 be predicates without free primed variables, and θ_1 and θ_2 be functions on Bnd with values in $\mathbb{R} \cup \{-\infty, \infty\}$. If $A_1 \Rrightarrow A_2$ and $A_2 \Rrightarrow \theta_1 \leq \theta_2$ then $\Theta(S, A_1) \Rrightarrow \Theta(S, A_2)$ and $\Upsilon(S, A_1, \theta_1) \leq \Upsilon(S, A_2, \theta_2)$.

3.3. A formal semantics for the transmitter example paths

Table 3 defines the syntax and semantics of a simple language for timed paths that can be extracted from a real-time program on the base of the timed specification statement and the path semantics and conventions of Table 1. The identifier x denotes a non-empty list of local variables. The output variable ω is not allowed to occur free in the expressions P, G, D and E. This means that the 'write' statement is the only command allowed to change an output variable. In a similar fashion we can define other commands as a 'read' statement that permits one to read from input variables. We have not done this here since the intention of this language is to provide the four timed paths of the transmitter example of the introduction with a formal semantics. Nor do we allow for undefined expressions. This can be done by incorporating a domain *def*(E) for each of the expressions defining the states where E is well-defined [HFL01]. We simply assume that all expressions are well-defined in their context. The 'assume' statement, which is used for stating properties believed true at this point in the code takes no time, whereas the 'coerce' statement, which models evaluation of guards on conditional and iterative statements, is time consuming. The 'delay until' statement suspends execution until the specified time, but may finish slightly 'late' with a maximal overrun here of 1 millisecond due to the practical limitations of an actual implementation. The 'deadline' statement specifies necessary timing points in the program [FHW99].

We apply Theorem 3.2 and the algorithm of Table 2 to our path language of Table 3. The resulting rules

for the computation of the expressions $\Theta(S, A)$ and $\Upsilon(S, A, \theta)$ for each of the commands of our language are depicted in Table 4. The commands are evaluated with respect to some abstract context I.

Example 3.5. The following shows the computation of the path traversal condition $\Theta(S)$ and the worst case execution time $\Upsilon(S)$ for Path A (loop entry) of the transmitter example. The computations are performed with the rules of Table 2 and Table 4. We fix an appropriate program context I.

$$I \stackrel{\text{def}}{=} \tau \in \mathbb{R} \land out \in \texttt{Char} \land N \in \mathbb{N} \land msg \in seq\ (\texttt{Char})$$

The expression $seq\ (\texttt{Char})$ shall denote the set of all finite sequences with values in the set of characters `Char`. Furthermore, we define some convenient abbreviations for a few subpaths of Path A.

$$\begin{aligned}
S_1 &\stackrel{\text{def}}{=} \texttt{deadline}\, 5 + 10 * n \\
S_2 &\stackrel{\text{def}}{=} \texttt{write}(msg(n), out)\ ;\ S_1 \\
S_3 &\stackrel{\text{def}}{=} \texttt{coerce}\, n \leq N - 1\ ;\ S_2 \\
S_4 &\stackrel{\text{def}}{=} n := 0\ ;\ S_3 \\
S_5 &\stackrel{\text{def}}{=} (\texttt{var}\ n \in \mathbb{N} \bullet S_4)
\end{aligned}$$

The path traversal condition for Path A is then computed as follows.

$$\begin{aligned}
\Theta(S_1^{I \land n \in \mathbb{N}}) &\equiv I \land n \in \mathbb{N} \land \tau \leq 5 + 10 * n \\
\Theta(S_2^{I \land n \in \mathbb{N}}) &\equiv I \land n \in \mathbb{N} \land \exists \tau', out (\tau \leq \tau' \land I[\tau'/\tau] \land out = msg(n) \land \tau' \leq 5 + 10 * n) \\
&\equiv I \land n \in \mathbb{N} \land \tau \leq 5 + 10 * n \\
\Theta(S_3^{I \land n \in \mathbb{N}}) &\equiv I \land n \in \mathbb{N} \land \exists \tau' (\tau \leq \tau' \land n \leq N - 1 \land I[\tau'/\tau] \land \tau' \leq 5 + 10 * n) \\
&\equiv I \land n \in \mathbb{N} \land n \leq N - 1 \land \tau \leq 5 + 10 * n \\
\Theta(S_4^{I \land n \in \mathbb{N}}) &\equiv I \land n \in \mathbb{N} \land \exists \tau' (\tau \leq \tau' \land I[\tau', 0/\tau, n] \land \tau' \leq 5) \\
&\equiv I \land n \in \mathbb{N} \land \tau \leq 5 \\
\Theta(S_5^I) &\equiv I \land \exists n (n \in \mathbb{N} \land I \land \tau \leq 5) \\
&\equiv I \land \tau \leq 5 \\
\Theta((\text{Path A})^I) &\equiv \tau \leq 3 \land I \land \tau \leq 5 \\
&\equiv \tau \leq 3 \land I
\end{aligned}$$

Thus, the path is feasible provided the initial assumption and context requirements hold. Next we compute the worst case execution time of Path A with the help of Table 2 and Table 4. We hereby assume that the computation is performed on a binding $\beta \in Bnd$ that makes $\Theta(\text{Path A})$ 'true', i.e., $\Theta(\text{Path A})(\beta) = true$, in order to apply Theorem A.2 of Appendix A.

$$\begin{aligned}
\Upsilon(S_1^{I \land n \in \mathbb{N}}) &= 0 \\
\Upsilon(S_2^{I \land n \in \mathbb{N}}) &= \sup\{\{\tau', out\} \mid I \land \tau \leq \tau' \land out = msg(n) \land \\
&\qquad I[\tau'/\tau] \land n \in \mathbb{N} \land \tau' \leq 5 + 10 * n \bullet \tau'\} - \tau \\
&= \text{“By Theorem A.2 and Theorem A.1”} \\
&\quad \sup\{\{\tau'\} \mid \tau \leq \tau' \ \land \tau' \in \mathbb{R} \land \tau' \leq 5 + 10 * n \bullet \tau'\} - \tau \\
&= 5 + 10 * n - \tau \\
\Upsilon(S_3^{I \land n \in \mathbb{N}}) &= \sup\{\{\tau'\} \mid I \land n \in \mathbb{N} \land \tau \leq \tau' \land n \leq N - 1 \land \\
&\qquad I[\tau'/\tau] \land \tau' \leq 5 + 10 * n \bullet 5 + 10 * n\} - \tau \\
&= \text{“By Theorem A.2”} \\
&\quad \sup\{\{\tau'\} \mid \ true \ \bullet 5 + 10 * n\} - \tau \\
&= \text{“By Theorem A.1”} \\
&\quad 5 + 10 * n - \tau \\
\Upsilon(S_4^{I \land n \in \mathbb{N}}) &= \sup\{\{\tau'\} \mid I \land \tau \leq \tau' \land I[\tau', 0/\tau, n] \land \tau' \leq 5 \bullet 5\} - \tau
\end{aligned}$$

Table 4. Path traversal condition and worst case execution time computation in the example language.

Command S	Path traversal condition $\Theta(S^I, A)$ for some context I
x := E	$I \wedge \exists \tau'(\tau \leq \tau' \wedge (I \wedge A)[\tau', E/\tau, x])$
`assume` P	$P \wedge A \wedge I$
`coerce` G	$I \wedge \exists \tau'(\tau \leq \tau' \wedge G \wedge (I \wedge A)[\tau'/\tau])$
`deadline` D	$I \wedge \tau \leq D \wedge A$
`delay until` D	$I \wedge \exists \tau'(\tau \leq \tau' \wedge D \leq \tau' \leq \max\{\tau, D\} + 1 \wedge (I \wedge A)[\tau'/\tau])$
`write` (x, ω)	$I \wedge \exists \tau', \omega(\tau \leq \tau' \wedge \omega = x \wedge (I \wedge A)[\tau'/\tau])$
Command S	Worst case execution time $\Upsilon(S^I, A, \theta)$ for some context I
x := E	$\sup\{\{\tau'\} \mid I \wedge \tau \leq \tau' \wedge (I \wedge A)[\tau', E/\tau, x]) \bullet \theta[\tau', E/\tau, x] + \tau' - \tau\}$
`assume` P	$\sup\{\emptyset \mid I \wedge P \bullet \theta\}$
`coerce` G	$\sup\{\{\tau'\} \mid I \wedge \tau \leq \tau' \wedge G \wedge (I \wedge A)[\tau'/\tau]) \bullet \theta[\tau'/\tau] + \tau' - \tau\}$
`deadline` D	$\sup\{\emptyset \mid I \wedge \tau \leq D \bullet \theta\}$
`delay until` D	$\sup\{\{\tau'\} \mid I \wedge \tau \leq \tau' \wedge D \leq \tau' \leq max\{\tau, D\} + 1 \wedge (I \wedge A)[\tau'/\tau] \bullet \theta[\tau'/\tau] + \tau' - \tau\}$
`write` (x, ω)	$\sup\{\{\tau', \omega\} \mid I \wedge \tau \leq \tau' \wedge \omega = x \wedge (I \wedge A)[\tau'/\tau] \bullet \theta[\tau'/\tau] + \tau' - \tau\}$

$$
\begin{aligned}
&= \text{“By Theorem A.2”} \\
&\quad \sup\{\{\tau'\} \mid \mathit{true} \bullet 5\} - \tau \\
&= 5 - \tau \\
\Upsilon(S_5^I) &= \sup\{\{n\} \mid \Theta(S_4^{I \wedge n \in \mathbb{N}}) \bullet 5 - \tau\} \\
&= \text{“By Theorem A.2”} \\
&\quad \sup\{\{n\} \mid \mathit{true} \bullet 5 - \tau\} \\
&= 5 - \tau \\
\Upsilon((\text{Path A})^I) &= 5 - \tau
\end{aligned}
$$

This leaves us with $\Upsilon((\text{Path A})^I) = 5 - \tau$ for the worst case execution time of Path A. The expression depends on the particular starting time τ and tells us that we have from this time until time 5 to perform the computation defined by the path. Similarly, we obtain $\Lambda((\text{Path A})^I) = 0$ for the best case execution time. We conclude that Path A is not dead and unrestricted from below. □

To simplify the timing analysis of real-time programs and their implementations we propagated the use of deadline constructs [HU97, FHW99] and variable-independent timing constraints [GHF98, LFH02a]. We showed that the extraction of a set of primitive timed paths $\mathcal{M}$ ending with deadlines, and the supremum and infimum of $\Lambda(S)$ and $\Upsilon(S)$, $S \in \mathcal{M}$, over all program variables lead to timing constraints that, if satisfiable, guarantee the correct timing behaviour of the program [LFH02a]. For the transmitter program such a set $\mathcal{M}$ of primitive paths consists of Path A, Path B, Path C and Path D.

Definition 3.6 (Execution time constraints). Let S be a timed path. The *strongest execution time constraint* and the *weakest execution time constraint* of S are defined by

$$\mathit{SETC}(S) \stackrel{\text{def}}{=} \sup\{\mathcal{V} \mid \Theta(S) \bullet \Lambda(S)\}, \; \mathit{WETC}(S) \stackrel{\text{def}}{=} \inf\{\mathcal{V} \mid \Theta(S) \bullet \Upsilon(S)\}$$

where $\mathcal{V}$ contains all free variables of $\Theta(S)$, $\Lambda(S)$ and $\Upsilon(S)$.

If S is permitted to take up to at most $\mathit{WETC}(S)$ milliseconds for its execution we know that all upper execution time bounds expressed via $\Theta(S)$ and $\Upsilon(S)$ are obeyed. Similarly for $\Lambda(S)$ and $\mathit{SETC}(S)$. Definition 3.6 is of great importance for the constraint analysis approach we propose. In the introduction we outlined how to decompose a real-time program with deadline commands into a set of primitive subpaths $\mathcal{M}$, such that the verification of the program's timing behaviour can be reduced to the verification of the timing constraints of the individual subpaths, i.e., the verification of $\mathit{SETC}(S)$ and $\mathit{WETC}(S)$ for all $S \in \mathcal{M}$.

Example 3.7. With the expressions defined in Example 3.5 we show the derivation of the strongest and weakest execution time constraints of Path A. Let $\mathcal{V}$ denote the free variables of the path traversal condition $\Theta((\text{Path A})^I)$.

$$\mathit{SETC}((\text{Path A})^I) = \sup\{\mathcal{V} \mid \Theta((\text{Path A})^I) \bullet 0\}$$

Table 5. Strongest postcondition semantics.

Command S	$sp(S, R)$
$(x\colon [Q])^I$	$(\exists x,\, \tau(I \wedge I[x', \tau'/x, \tau] \wedge Q \wedge \tau \leq \tau' \wedge R))[x, \tau/x', \tau']$
$(\mathbf{var}\ v\ :T \bullet S_1)^I$	$\exists w\, (sp(S_1[w/v]^{I \wedge w \in T}, R))$ with $w \notin Idf(S)$ and not free in R
$(S_1^{I_1})^{I_2}$	$sp(S_1^{I_1 \wedge I_2}, R)$
$(S_1\ ;\ S_2)^I$	$sp(S_2^I, sp(S_1^I, R))$

$$
\begin{aligned}
&= \text{“By Theorem A.1”} \\
&\quad \sup\{\emptyset \mid \exists \mathcal{V}\, \Theta((\text{Path A})^I) \bullet 0\} \\
&= \sup\{\{\tau\} \mid \mathit{true} \bullet 0\} \\
&= 0 \\
&< 2 \\
&= \inf\{\{\tau\} \mid \tau \leq 3 \wedge \tau \in \mathbb{R} \bullet 5 - \tau\} \\
&= \inf\{\{\tau\} \mid \exists \mathcal{V} \setminus \{\tau\}\, \Theta((\text{Path A})^I) \bullet 5 - \tau\} \\
&= \text{“By Theorem A.1”} \\
&\quad \inf\{\mathcal{V} \mid \Theta((\text{Path A})^I) \bullet 5 - \tau\} \\
&= \mathit{WETC}((\text{Path A})^I)
\end{aligned}
$$

This shows that the least of the worst case execution times permitted by Path A is 2 and the greatest of the best case execution times permitted by this path is 0. This tells us that if the actual execution time of the path is always between 0 and 2 milliseconds then the path's timing constraints and in particular its deadline will always be satisfied. An implementation of this path which would be obviously the code for the variable allocation, the evaluation of the boolean guard, the assignment, the write command and the variable deallocation, would fulfil the specified timing behaviour if it consumed the most 2 milliseconds for its execution. □

3.4. Strongest postcondition semantics for timed paths

For the formal analysis of complex timed paths it is often necessary to incorporate some information of a predecessor path into the analysis. Table 5 therefore defines the strongest postcondition semantics for a timed path S in our language. Given a predecessor S_1 of a path S_2, $sp(S_1, \mathit{true})$ describes the strongest predicate that holds after execution of path S_1 and which therefore can be taken as starting condition for path S_2. In other words, the path $(S_1\ ;\ S_2)$ is liberal refinement equivalent to the following path [LFH02b].

S_1 ; `assume` $sp(S_1, \mathit{true})$; S_2

More generally for a predicate R, $sp(S_1, R)$ denotes the strongest predicate that holds after execution of S_1 assuming R was 'true' before.

Definition 3.8. For a timed path S, the predicate $sp(S, \mathit{true})$ is named the *exit condition* $\Xi(S)$ of S.

Strongest postcondition semantics for programs have been extensively used in formal program analysis [Gri81, DS90, BvW93]. The equivalent expressive power of weakest liberal precondition and strongest postcondition semantics has been shown by Dijkstra and Scholten [DS90].

Example 3.9. For Path B (single iteration) of the transmitter we perform a similar computation as in Example 3.5 and 3.7, but partition the path first into the subpath containing the deadline command and the succeeding delay command, and the subpath that follows afterwards, starting with the assignment command. We compute the exit condition of the preceding path with the help of Table 5. For this we will again introduce abbreviations for some subpaths of Path B.

$$S_1 \stackrel{\text{def}}{=} \texttt{deadline}\ 5 + 10 * n$$

Fig. 2. Path B', liberal refinement equivalent to Path B.

$(\texttt{var}\, n : \mathbb{N} \bullet$
$\texttt{deadline}\ 5 + 10 * n\ ;$
$\texttt{delay until}\ 11 + 10 * n\ ;$
$\texttt{assume}\ \Xi(S_2^{I \wedge n \in \mathbb{N}})\ ;$
$n := n + 1\ ;$
$\texttt{coerce}\ n \leq N - 1\ ;$
$\texttt{write}(msg(n), out)\ ;$
$\texttt{deadline}\ 5 + 10 * n)$

$$
\begin{array}{lcl}
S_2 & \stackrel{\text{def}}{=} & S_1\ ;\ \texttt{delay until}\ 11 + 10 * n \\
S_3 & \stackrel{\text{def}}{=} & \texttt{write}(msg(n), out)\ ;\ S_1 \\
S_4 & \stackrel{\text{def}}{=} & \texttt{coerce}\ n \leq N - 1\ ;\ S_3 \\
S_5 & \stackrel{\text{def}}{=} & n := n + 1\ ;\ S_4 \\
S_6 & \stackrel{\text{def}}{=} & \texttt{assume}\, \Xi(S_2^{I \wedge n \in \mathbb{N}})\ ;\ S_5
\end{array}
$$

The following shows the computation of the exit condition of path S_2 under the context $I \wedge n \in \mathbb{N}$.

$$
\begin{array}{lcl}
\Xi(S_1^{I \wedge n \in \mathbb{N}}) & \equiv & \exists \tau (I \wedge n \in \mathbb{N} \wedge I[\tau'/\tau] \wedge \tau' = \tau \wedge \tau' \leq 5 + 10 * n)[\tau/\tau'] \\
& \equiv & I \wedge n \in \mathbb{N} \wedge \tau \leq 5 + 10 * n \\
\Xi(S_2^{I \wedge n \in \mathbb{N}}) & \equiv & I \wedge n \in \mathbb{N} \wedge \exists \tau (I \wedge n \in \mathbb{N} \wedge \tau \leq \tau' \wedge 11 + 10 * n \leq \tau' \leq max\{11 + 10 * n, \tau\} + 1 \wedge \\
& & \qquad \tau \leq 5 + 10 * n)[\tau/\tau'] \\
& \equiv & I \wedge n \in \mathbb{N} \wedge 11 + 10 * n \leq \tau \leq 12 + 10 * n
\end{array}
$$

The range of values for the finishing time τ is caused by the nondeterminism in the 'overrun' or 'lateness' of the delay statement. Then path S_2, with context $I \wedge n \in \mathbb{N}$, is liberal refinement equivalent to the following path.

$S_2^{I \wedge n \in \mathbb{N}}\ ;\ \texttt{assume}\, \Xi(S_2^{I \wedge n \in \mathbb{N}})$

Therefore we can deduce that Path B, under the context I, is liberal refinement equivalent to Path B' in Figure 2. We proceed by computing the path traversal condition for path S_6 under context $I \wedge n \in \mathbb{N}$.

$$
\begin{array}{lcl}
\Theta(S_1^{I \wedge n \in \mathbb{N}}) & \equiv & I \wedge n \in \mathbb{N} \wedge \tau \leq 5 + 10 * n \\
\Theta(S_3^{I \wedge n \in \mathbb{N}}) & \equiv & I \wedge n \in \mathbb{N} \wedge \tau \leq 5 + 10 * n \\
\Theta(S_4^{I \wedge n \in \mathbb{N}}) & \equiv & I \wedge n \in \mathbb{N} \wedge n \leq N - 1 \wedge \tau \leq 5 + 10 * n \\
\Theta(S_5^{I \wedge n \in \mathbb{N}}) & \equiv & I \wedge n \in \mathbb{N} \wedge n + 1 \leq N - 1 \wedge \exists \tau' (\tau \leq \tau' \wedge I[\tau', n + 1/\tau, n] \wedge \tau' \leq 5 + 10 * (n + 1)) \\
& \equiv & I \wedge n \in \mathbb{N} \wedge n \leq N - 2 \wedge \tau \leq 5 + 10 * (n + 1) \\
\Theta(S_6^{I \wedge n \in \mathbb{N}}) & \equiv & I \wedge n \in \mathbb{N} \wedge n \leq N - 2 \wedge 11 + 10 * n \leq \tau \leq 12 + 10 * n
\end{array}
$$

This provides the path traversal condition for path S_6. Thus the context requirements must hold, counter n must not exceed $N-2$ (otherwise this would make the coercion false), and the subpath's starting time τ must be between the earliest and latest times at which the 'delay until' statement can finish. We next derive the worst case execution time of this path under context $I \wedge n \in \mathbb{N}$. We assume again that the computations are done for a binding $\beta \in Bnd$ such that $\Theta(S_6^{I \wedge n \in \mathbb{N}})(\beta) = true$ which makes it possible to apply Theorem A.2.

$$
\begin{array}{lcl}
\Upsilon(S_1^{I \wedge n \in \mathbb{N}}) & = & 0 \\
\Upsilon(S_3^{I \wedge n \in \mathbb{N}}) & = & \sup\{\{\tau', out\} \mid I \wedge \tau \leq \tau' \wedge out = msg(n) \wedge (I \wedge \Theta(S_1^{I \wedge n \in \mathbb{N}}))[\tau'/\tau] \bullet \tau'\} - \tau \\
& = & \text{“By Theorem A.1”}
\end{array}
$$

$$
\begin{array}{rcl}
 & & \sup\{\{\tau'\} \mid \exists out (I \wedge \tau \leq \tau' \wedge out = msg(n) \wedge (I \wedge \Theta(S_1^{I \wedge n \in \mathbb{N}}))[\tau'/\tau]) \bullet \tau'\} - \tau \\
 & = & \text{“By Theorem A.2”} \\
 & & \sup\{\{\tau'\} \mid \tau' \in \mathbb{R} \wedge \tau \leq \tau' \leq 5 + 10 * n \bullet \tau'\} - \tau \\
 & = & 5 + 10 * n - \tau \\
\Upsilon(S_4^{I \wedge n \in \mathbb{N}}) & = & \sup\{\{\tau'\} \mid I \wedge \tau \leq \tau' \wedge n \leq N - 1 \wedge (I \wedge \Theta(S_3^{I \wedge n \in \mathbb{N}}))[\tau'/\tau] \bullet 5 + 10 * n\} - \tau \\
 & = & \text{“By Theorem A.1”} \\
 & & 5 + 10 * n - \tau \\
\Upsilon(S_5^{I \wedge n \in \mathbb{N}}) & = & \sup\{\{\tau'\} \mid I \wedge \tau \leq \tau' \wedge (I \wedge \Theta(S_4^{I \wedge n \in \mathbb{N}}))[\tau', n + 1/\tau, n] \bullet 5 + 10 * (n + 1)\} - \tau \\
 & = & \text{“By Theorem A.1”} \\
 & & \sup\{\emptyset \mid \exists \tau' (I \wedge \tau \leq \tau' \wedge (I \wedge \Theta(S_4^{I \wedge n \in \mathbb{N}}))[\tau', n + 1/\tau, n]) \bullet 5 + 10 * (n + 1)\} - \tau \\
 & = & 5 + 10 * (n + 1) - \tau \\
\Upsilon(S_6^{I \wedge n \in \mathbb{N}}) & = & 5 + 10 * (n + 1) - \tau
\end{array}
$$

Then we obtain for the weakest execution time constraint of path $S_6^{I \wedge n \in \mathbb{N}}$ the following value. Let $\mathcal{V}$ denote the free variables of predicate $\Theta(S_6^{I \wedge n \in \mathbb{N}})$ and expression $\Upsilon(S_6^{I \wedge n \in \mathbb{N}})$.

$$
\begin{array}{rcl}
WETC(S_6^{I \wedge n \in \mathbb{N}}) & = & \inf\{\mathcal{V} \mid \Theta(S_6^{I \wedge n \in \mathbb{N}}) \bullet \Upsilon(S_6^{I \wedge n \in \mathbb{N}})\} \\
 & = & \text{“By Theorem A.1”} \\
 & = & \inf\{\{\tau, n\} \mid \exists \mathcal{V} \setminus \{\tau, n\}\, \Theta(S_6^{I \wedge n \in \mathbb{N}}) \bullet \Upsilon(S_6^{I \wedge n \in \mathbb{N}})\} \\
 & = & \inf\{\{\tau, n\} \mid n \in \mathbb{N} \wedge \tau \in \mathbb{R} \wedge 11 + 10 * n \leq \tau \leq 12 + 10 * n \bullet 5 + 10 * (n + 1) - \tau\} \\
 & = & \text{“By Theorem A.4”} \\
 & & \inf\{\{n\} \mid n \in \mathbb{N} \bullet \inf\{\{\tau\} \mid \tau \in \mathbb{R} \wedge 11 + 10 * n \leq \tau \leq 12 + 10 * n \bullet \\
 & & \qquad 5 + 10 * (n + 1) - \tau\}\} \\
 & = & \inf\{\{n\} \mid n \in \mathbb{N} \bullet 3\} \\
 & = & \text{“By Theorem A.1”} \\
 & & \inf\{\emptyset \mid \exists n (\, n \in \mathbb{N}) \bullet 3\} \\
 & = & 3
\end{array}
$$

To understand this, consider that in the worst case there is only 4 milliseconds between the time at which the n^{th} character may be overwritten and the $(n+1)^{th}$ character must appear. However, this time is reduced by the possible lateness of the ‘delay until’ statement. Similarly, we obtain the following for the best case execution time and strongest execution time constraint of path $S_6^{I \wedge n \in \mathbb{N}}$.

$$
\Lambda(S_6^{I \wedge n \in \mathbb{N}}) \;=\; SETC(S_6^{I \wedge n \in \mathbb{N}}) \;=\; 0
$$

□

Finally, we note that the computation of the worst case and best case execution time, and the weakest and strongest execution time constraints for all the paths of the transmitter can be performed by supremum and infimum computations over predicates with linear inequalities. The worst and best case execution times and weakest and strongest execution time constraints for all timed paths of the transmitter example are stated in Table 6. We omit the lengthy and repetitive computations for the remaining two paths. As explained above, the constraint for Path B was derived by partitioning this path and by investigating the suffix S_6. We did not discuss the prefix path which consists of a ‘deadline’ and a ‘delay’ statement. It would be typically implemented by a delay routine which guarantees to have maximal overrun 1. A similar computation can be performed for Path D by defining a suffix S_6'. In the next section we show how execution time and constraint computations of this form can be implemented with techniques from Linear Programming and Presburger Arithmetic.

Table 6. Execution times and execution time constraints for the transmitter paths.

Path S	Worst case execution time $\Upsilon(S)$	Best case execution time $\Lambda(S)$ and Strongest execution time constraint $SETC(S)$	Weakest execution time constraint $WETC(S)$
$(\text{Path A})^I$	$5-\tau$	0	2
$S_6^{I\wedge n\in\mathbb{N}}$ – $(\text{Path B})^I$	$5+10*(n+1)-\tau$	0	3
$(\text{Path C})^I$	$5-\tau$	0	2
$S_6'^{I\wedge n\in\mathbb{N}}$ – $(\text{Path D})^I$	$5+10*(n+1)-\tau$	0	3

4. Implementation

The supremum and infimum computations for the derivation of the worst and best case execution times, $\Upsilon(S)$ and $\Lambda(S)$, of a timed path S shown in Table 2 are non-discrete methods and thus may not lead to computable algorithms. In this section we show how techniques from Linear Programming and Presburger Arithmetic can be used to compute these expressions. Shostak and Bledsoe's famous algorithm [Sho77] translates the discrete problem of computing formulas in Presburger arithmetic into a non-discrete one where all computations are performed on linear inequalities in the domain of the real numbers. We describe an approach that uses these non-discrete techniques to implement the algorithm of Table 2 for the computation of worst and best case execution times of timed paths.

4.1. Linear approximation of timed paths

This section describes the linear approximation of path traversal conditions and strongest postconditions of timed paths. We define a linear projection that maps first-order predicates onto linear predicates.

We first fix some notational conventions. A *linear form* denotes an expression $\alpha_0+\sum_{i=1}^{n}\alpha_i x_i$ where α_i denotes a constant symbol and x_i denotes a variable. A *linear inequality* is defined as an expression $L_1\leq L_2$ or its negation $\neg(L_1\leq L_2)$, where L_1, L_2 are linear forms. Note that $\neg(L_1\leq L_2)$ will be abbreviated by a strict *linear inequality* $L_2<L_1$. A *conjunctive linear predicate* denotes the conjunction of a finite number of linear inequalities. A *linear predicate* denotes the disjunction of conjunctive linear predicates. A *quantified linear predicate* denotes a predicate that consists of universally and existentially quantified linear predicates.

According to standard prenex normal form for first-order predicates and disjunctive normal form transformations every first-order predicate can be transformed into an equivalent prenex normal form

$$Q_1\,x_1\,...Q_n\,x_n\,(A_1\vee\,...\,\vee\,A_m)$$

where Q_i are quantifiers, x_i are variables and A_i are conjunctions of atoms of the form $a_1^i\wedge\,...\,\wedge\,a_{k_i}^i$, where a_j^i denotes an atom or the negation of an atom. For instance, in our setting linear inequalities are atoms. There is a natural projection *Prn* defined on this normal form of the first-order predicate which removes all non-linear atoms and leaves a quantified linear predicate in disjunctive normal form. This syntactic transformation of first-order formulas does not necessarily transform logically equivalent formulas into logically equivalent ones.

Example 4.1. The projection *Prn* would transform the predicate

$$\exists a,c\,((0\leq a\wedge 4a\leq c\wedge c\leq 5)\vee(a\in\mathbb{N}\wedge a\leq c+x^2\wedge 7\leq c+x\wedge c\leq 10)) \tag{2}$$

into the following existentially quantified linear predicate.

$$\exists a,c\,((0\leq a\wedge 4a\leq c\wedge c\leq 5)\vee(7\leq x+c\wedge c\leq 10))$$

Under the theory of real number arithmetic the predicate $x^2=1$ is logically equivalent to $1\leq x\leq 1\vee -1\leq x\leq -1$ but *Prn* does not project these predicates to logically equivalent ones.

$$Prn(x^2=1)\quad\equiv\quad true\quad\not\equiv\quad 1\leq x\leq 1\vee -1\leq x\leq -1\equiv Prn(1\leq x\leq 1\vee -1\leq x\leq -1)\qquad\square$$

There are many ways of projecting predicates onto linear predicates according to the grade of approximation that has to be achieved. We do not intend to go further into different approximation techniques. For instance, Example 4.1 states two different linear approximations for the quadratic equation $x^2=1$. Instead we state some properties that we expect from a projection.

Definition 4.2 (Linear projection). A transformation Pr of first-order predicates is a (linear) projection if for all predicates A, B and variables x, $Pr(A)$ is a quantified linear predicate, $Pr(Pr(A)) \equiv Pr(A)$, $A \Rrightarrow Pr(A)$, $Pr(A \wedge B) \equiv Pr(A) \wedge Pr(B)$, $Pr(\neg A) \equiv \neg Pr(A)$ and $Pr(\exists x\, A) \equiv \exists x\, Pr(A)$ holds.

In the following we will assume that a particular linear projection Pr has been chosen. Unless a more sophisticated approximation method is determined this could be the above described canonical projection Prn.

Under the theory of real number arithmetic it is furthermore possible to transform quantified linear predicates into logically equivalent linear predicates by a standard quantifier elimination procedure. This is outlined below.

We first define the function Tr that transforms an existentially quantified conjunctive linear predicate into a logically equivalent conjunctive linear predicate. For a set of variables V and a set of linear inequalities E, $Tr(E, V) \stackrel{\text{def}}{=} E$ iff $V = \emptyset$. Otherwise, if $x \in V$ then

$$Tr(E, V) \stackrel{\text{def}}{=} Tr(E', V \setminus \{x\})$$

where E' consists of all linear inequalities of the form (i) or (ii) below.

$i)$ $A \leq B$ such that x is not free in A and B, and either (a) or (b) below hold.

$a)$ $A \leq B$ is in E;

$b)$ $A \leq x$ and $x \leq B$ are linear inequalities that result from inequalities in E by linear transformations.

$ii)$ $A < B$ such that x is not free in A and B, and either (c) or (d) below hold.

$c)$ $A < B$ is in E;

$d)$ $A < x$ and $x \leq B$; or $A < x$ and $x < B$; or $A \leq x$ and $x < B$ are linear inequalities that result from inequalities in E by linear transformations.

Example 4.3. Tr transforms the existentially quantified linear predicate

$$\exists a, c\, (0 \leq a \wedge 4a \leq c \wedge c \leq 5)$$

into an equivalent linear form as follows.

$$\begin{aligned} Tr(\{0 \leq a, 4a \leq c, c \leq 5\}, \{a, c\}) &= Tr(\,Tr(\{0 \leq a, 4a \leq c, c \leq 5\}, \{a\}), \{c\}) \\ &= Tr(\{0 \leq c/4, c \leq 5\}, \{c\}) \\ &= \{0 \leq 5\} \qquad \square \end{aligned}$$

With the following function Tr' it is then possible to transform existentially quantified linear predicates into linear predicates: for a set of variables V and n sets of linear inequalities E_i, $1 \leq i \leq n$,

$$Tr'((E_1, ..., E_n), V) \stackrel{\text{def}}{=} (\,Tr(E_1, V), ..., Tr(E_n, V))\ .$$

Example 4.4. Applied to Example 4.1, Tr' would be evaluated as follows,

$$Tr'((0 \leq a, 4a \leq c, c \leq 5\}, \{7 \leq x + c, c \leq 10\}), \{a, c\}) = (\{0 \leq 5\}, \{7 - x \leq 10\}) \qquad \square$$

The functions Tr and Tr' can be extended to linear predicates that are universally quantified with the help of a standard transformation that transforms the predicate into the negation of an existentially quantified linear predicate and then applies Tr or Tr' to the existentially quantified linear predicate and negates the resulting linear inequalities.

Last but not least, with these transformations it is possible to define a function L on the first-order predicates that returns a linear predicate by applying Pr and Tr'. The original first-order predicate logically implies the resulting linear predicate, i.e., the following property holds for all first-order predicates P.

$$P \quad \Rrightarrow \quad L(P) \tag{3}$$

This mapping can be interpreted as a *linear approximation* of first-order predicates. Furthermore, for quantified linear predicates P the following equivalence holds.

$$P \quad \equiv \quad L(P) \tag{4}$$

The transformation L can be seen as a (not necessarily equivalence preserving) projection onto the space of the linear predicates.

Table 7. A linear approximation for path traversal conditions and strongest postconditions.

Command S	Approximate path traversal condition $\Theta^a(S,R)$	Approximate strongest postcondition $sp^a(S,R)$
$(x\colon [Q])^I$	$L(\exists x', \tau'(I \wedge (I \wedge R)[x', \tau'/x, \tau] \wedge Q \wedge \tau \leq \tau'))$	$L(\exists x,\ \tau(I \wedge I[x', \tau'/x, \tau] \wedge Q \wedge \tau \leq \tau' \wedge R))[x, \tau/x', \tau']$
$(\mathbf{var}\ x : T \bullet S_1)^I$	$L(\exists w\ (\Theta^a(S_1[w/x]^{(I \wedge w \in T)}, R)))$ with $w \notin Idf(S) \cup Fv(R)$	$L(\exists w\ (sp^a((S_1[w/v])^{I \wedge w \in T}, R)))$ with $w \notin Idf(S) \cup Fv(R)$
$(S_1^{I_1})^{I_2}$	$\Theta^a(S_1^{I_1 \wedge I_2}, R)$	$sp^a(S_1^{I_1 \wedge I_2}, R)$
$(S_1\ ;\ S_2)^I$	$\Theta^a(S_1^I, \Theta^a(S_2^I, R))$	$sp^a(S_2^I, sp^a(S_1^I, R))$

For later use we define another function L', similar to L, which simply returns a set of conjunctive linear predicates such that the following relation holds for every first-order predicate P.

$$L(P) \quad \equiv \quad \vee L'(P) \tag{5}$$

A *mini-linear form* A [Sho77] is defined as either $-\infty$, or ∞, or as an expression $Min(A_1, ..., A_n)$ with linear forms A_i or $A_i \in \{-\infty, \infty\}$, $1 \leq i \leq n$. *Maxi-linear forms* $Max(A_1, ..., A_n)$ are defined similarly. An expression constructed from maxi and mini-linear forms is called a *quasi-linear form.*

The function Θ^a of Table 7 determines a linear approximation for path traversal conditions of timed paths. The function Θ^a takes as input a timed path S and a first-order predicate R and returns a linear predicate. We can therefore say that function Θ^a defines a linear approximation of timed paths. In terms of liberal refinement, this means that the transformed path is more abstract than the original path. The following theorem is a consequence of Definition 4.2 and the properties (3) and (4) of projection L.

Theorem 4.5 (Path approximation). For a timed path S and a predicate R without free primed variables the following properties hold.

i) $L(\Theta(S,R)) \equiv \Theta^a(S,R)$ and $\Theta(S,R)$ logically implies $\Theta^a(S,R)$.
ii) If S and R are constructed from quantified linear predicates then $\Theta(S,R)$ is equivalent to $\Theta^a(S,R)$.

A similar theorem holds for the approximation of strongest postconditions as defined in Table 7.

Theorem 4.6 (Postcondition approximation). For a timed path S and a predicate R without free primed variables the following properties hold.

i) $L(sp(S,R)) \equiv sp^a(S,R)$, in particular, $sp(S,R)$ logically implies $sp^a(S,R)$.
ii) If S and R are constructed from quantified linear predicates then $sp(S,R)$ is equivalent to $sp^a(S,R)$.

4.2. Linear reductions and linear execution time approximation

The computation of worst case and best case execution times for a timed path S rarely depends on the knowledge of the entire path traversal condition $\Theta(S)$ as indicated in the examples of the previous sections and in Theorem A.2 of Appendix A. For 'supremum' computations over predicates, as for the computation of the worst case execution time of a path S, Theorem A.2 provides a method of reducing the predicates by removing unnecessary conjuncts. In this section we formulate an algorithm that approximates execution times and checks whether the computation of the path's execution times can be performed by reduction to linear predicates. This method will be used in the next section to identify whether a path allows for automated execution time computation.

Definition 4.7 (Linear reduction). For predicates C, A and B and sets of variables V and W we say that (A,B) is a *linear reduction* of (C,V,W) if the following conditions hold.

i) A is a linear predicate.
ii) $C \equiv A \wedge B$
iii) $Fv(B) \cap V \cap W = \emptyset$
iv) $Fv(A) \cap Fv(B) \cap V = \emptyset$

We proceed by listing a few properties of linear reductions. For every predicate C and sets of variables V and W with $V \cap W = \emptyset$, it follows that $(true, C)$ is a linear reduction of (C,V,W). A consequence from Definition 4.7 is the following proposition.

Table 8. Linear execution time splits and execution time approximation.

Command S	All unbound variables that occur in the linear approximation $Lv(S, A, W)$
$(x\colon [Q])^I$	$(Fv(L(I \wedge (I \wedge A)[x', \tau'/x, \tau] \wedge Q \wedge \tau \leq \tau')) \cup W[x', \tau'/x, \tau] \cup \{\tau\}) \setminus \{\tau', x'\}$
$(\mathbf{var}\ x : T \bullet S_1)^I$	$(Fv(L(\Theta(S_1[w/v]^{I \wedge w \in T}, A))) \cup Lv(S_1[w/v]^{I \wedge w \in T}, A, W)) \setminus \{w\}$ with $w \notin Idf(S)$ and not free in predicate A
$(S_1^{I_1})^{I_2}$	$Lv(S_1^{I_1 \wedge I_2}, A, W)$
$(S_1\ ;\ S_2)^I$	$Lv(S_1^I, L(\Theta(S_2^I, A)), Lv(S_2^I, A, W))$
Command S	Linear execution time reduction $Ler(S, A, W)$
$(x\colon [Q])^I$	$Lr(I \wedge (I \wedge A)[x', \tau'/x, \tau] \wedge Q \wedge \tau \leq \tau', \{\tau', x'\}, W[x', \tau'/x, \tau] \cup \{\tau, \tau'\})$
$(\mathbf{var}\ x : T \bullet S_1)^I$	$Ler(S_1[w/v]^{I \wedge w \in T}, A, W) \wedge Lr(\Theta(S_1[w/v]^{I \wedge w \in T}, A), \{w\}, Lv(S_1[w/v]^{I \wedge w \in T}, A, W))$ with $w \notin Idf(S)$ and not free in predicate A
$(S_1^{I_1})^{I_2}$	$Ler(S_1^{I_1 \wedge I_2}, A, W)$
$(S_1\ ;\ S_2)^I$	$Ler(S_2^I, A, W) \wedge Ler(S_1^I, \Theta(S_2^I, A), Lv(S_2^I, A, W))$
Command S	Linear execution time approximation $\upsilon(S, A, \theta)$
$(x\colon [Q])^I$	$\sup\{\{\tau', x'\} \mid L(I \wedge (I \wedge A)[x', \tau'/x, \tau] \wedge Q \wedge \tau \leq \tau') \bullet \theta[x', \tau'/x, \tau] + \tau' - \tau\}$
$(\mathbf{var}\ x : T \bullet S_1)^I$	$\sup\{\{w\} \mid L(\Theta(S_1[w/v]^{I \wedge w \in T}, A)) \bullet \upsilon(S_1[w/v]^{I \wedge w \in T}, A, \theta)\}$ with $w \notin Idf(S)$ and not free in predicate A
$(S_1^{I_1})^{I_2}$	$\upsilon(S_1^{I_1 \wedge I_2}, A, \theta)$
$(S_1\ ;\ S_2)^I$	$\upsilon(S_1^I, L(\Theta(S_2^I, A)), \upsilon(S_2^I, A, \theta))$

Proposition 4.8. Let A, B, C be predicates and let V and W be sets of variables. If $Fv(C) \cap (V \setminus W) = \emptyset$, then the conditions (i), (ii) and (iii) of Definition 4.7 are sufficient to verify that (A, B) is a linear reduction of (C, V, W).

The problem of finding a linear reduction is undecidable in general. Nevertheless it is possible to provide algorithms that detect linear reductions in many cases like, for instance, simple syntactic checks based on the linear and nonlinear atoms of predicate C. For example for all the predicates that appeared in the worst case execution time computations of the transmitter paths it can be syntactically checked that they allow for linear reductions.

Example 4.9. Let predicate C and function θ be defined by

$$C \stackrel{\text{def}}{=} x^2 = 5 \wedge n \leq 6 \wedge n \leq x\ , \qquad \theta \stackrel{\text{def}}{=} x - 2 * n$$

with $L(C) \equiv n \leq 6 \wedge n \leq x$ and $Fv(\theta) = \{n, x\}$. The tuple $(L(C), x^2 = 5)$ is a linear reduction of $(C, \{n\}, Fv(\theta))$. Under the assumption that we compute the supremum expression

$$\sup\{\{n\} \mid C \bullet \theta\}$$

on a binding that makes predicate $\exists n\, C$ 'true', we can apply Theorem A.2, remove the nonlinear conjunct $x^2 = 5$ and perform the following simplification to a mini-linear form.

$$\begin{aligned} \sup\{\{n\} \mid C \bullet \theta\} &= \text{'By Theorem A.2'} \\ &\quad \sup\{\{n\} \mid L(C) \bullet \theta\} \\ &= x - 2 * Min(6, x) \quad \square \end{aligned}$$

We will henceforth assume the existence of an abstract boolean valued function $Lr(C, V, W)$ that, if it returns 'true', guarantees that $L(\exists(V \setminus W)\, C)$ is a linear reduction of $(\exists(V \setminus W)\, C, V, W)$. (For convenience, here, an expression like $\exists\{y, x\}\, C$ abbreviates the quantified predicate $\exists y, x\, C$.) Such a function can be implemented as mentioned above as a simple syntactic check of the predicate. Given such a function, Table 8 defines the function Ler that takes a path S, a predicate A and a set of variables W and returns 'true' if the path's execution time computation allows for linear reductions on the base of function Lr in each step of the algorithm of Table 2. Furthermore, Table 8 defines the set of variables $Lv(S, A, Fv(\theta))$ that are free in the linear approximation $\upsilon(S, A, \theta)$ of the expression $\Upsilon(S, A, \theta)$ where only linear predicates have been used in the computation. The following theorem states the correspondences between these functions.

Theorem 4.10 (Execution time approximation). Let S be a timed path, let A be a predicate and let θ be a real-valued function. Then with the definitions of Table 8 the following properties hold.

i) $\upsilon(S, A, \theta) = \upsilon(S, L(A), \theta)$
ii) $Fv(\upsilon(S, A, \theta)) = Lv(S, A, Fv(\theta))$
iii) $\Upsilon(S, A, \theta) \leq \upsilon(S, A, \theta)$.
iv) If $Ler(S, A, Fv(\theta)) = true$ then $\Theta(S, A) \Rrightarrow \Upsilon(S, A, \theta) = \upsilon(S, A, \theta)$.

Proof. Since it is a straightforward structural induction over the path language we omit the proof for assertions (i), (ii) and (iii).

Assertion (iv): The base case $S \stackrel{\text{def}}{=} (x{:}[Q])^I$. We use the following abbreviations.

$$\begin{aligned} V &\stackrel{\text{def}}{=} \{x', \tau'\} \\ W &\stackrel{\text{def}}{=} Fv(\theta[x', \tau'/x, \tau]) \cup \{\tau, \tau'\} \\ C &\stackrel{\text{def}}{=} I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A)[\tau', x'/\tau, x] \end{aligned}$$

Because of the assumption on path S and Table 8 we have the following correspondence.

$$\begin{aligned} true &\equiv Ler(S, A, Fv(\theta)) \\ &\equiv Lr(C, V, W) \end{aligned}$$

This means that there is a predicate B such that $(L(\exists(V \setminus W)\, C), B)$ is a linear reduction of $(\exists(V \setminus W)\, C, V, W)$. According to Table 8 then the following equality holds. For the computations we assume that $\Theta(S, A)$ holds.

$$\begin{aligned} \upsilon(S, A, \theta) &= \text{“By Table 8”} \\ &\quad \sup\{\{\tau', x'\} \mid L(C) \bullet \theta[\tau', x'/\tau, x] + \tau' - \tau\} \\ &= \text{“By Theorem A.1”} \\ &\quad \sup\{\{\tau', x'\} \mid L(\exists\,(V \setminus W)\, C) \bullet \theta[\tau', x'/\tau, x] + \tau' - \tau\} \\ &= \text{“By } Ler(S, A, Fv(\theta)) = true \text{ Definition 4.7, } \Theta(S, A) \text{ holds and Theorem A.2”} \\ &\quad \sup\{\{\tau', x'\} \mid \exists\,(V \setminus W)\, C \bullet \theta[\tau', x'/\tau, x] + \tau' - \tau\} \\ &= \text{“By Theorem A.1”} \\ &\quad \sup\{\{\tau', x'\} \mid C \bullet \theta[\tau', x'/\tau, x] + \tau' - \tau\} \\ &= \text{“By Table 2”} \\ &\quad \Upsilon(S, A, \theta) \end{aligned}$$

The step case $S \stackrel{\text{def}}{=} (\mathbf{var}\, v\ : T \bullet S_1)^I$. We use the following abbreviations.

$$\begin{aligned} V &\stackrel{\text{def}}{=} \{w\} \\ W &\stackrel{\text{def}}{=} Lv(S_1[w/x]^{I \wedge w \in T}, A, Fv(\theta)) \\ C &\stackrel{\text{def}}{=} \Theta(S_1[w/x]^{I \wedge w \in T}, A) \end{aligned}$$

Since according to the assumption on the path S a linear reduction is possible and because of Table 8 we have the following equivalences.

$$\begin{aligned} true &\equiv Ler(S, A, Fv(\theta)) \\ &\equiv Lr(C, V, W) \wedge Ler(S_1[w/x]^{I \wedge w \in T}, A, Fv(\theta)) \end{aligned}$$

This ensures the existence of a predicate B such that $(L(\exists(V \setminus W)\, C), B)$ is a linear reduction of $(\exists(V \setminus W)\, C, V, W)$. Furthermore, $Ler(S_1[w/x]^{I \wedge w \in T}, A, Fv(\theta)) = true$. The induction hypothesis can then be formulated as follows.

$$\Theta(S_1[w/x]^{I \wedge w \in T}, A) \Rrightarrow \upsilon(S_1[w/x]^{I \wedge w \in T}, A, \theta) = \Upsilon(S_1[w/x]^{I \wedge w \in T}, A, \theta) \tag{6}$$

Then the following equalities hold. We assume again that $\Theta(S, A)$ holds.

$$
\begin{array}{lcl}
\Upsilon(S, A, \theta) & = & \text{“By Table 2”} \\
 & & \sup\{\{w\} \mid C \bullet \Upsilon(S_1[w/x]^{I \wedge w \in T}, A, \theta)\} \\
 & = & \text{“By induction hypothesis (6)”} \\
 & & \sup\{\{w\} \mid C \bullet \upsilon(S_1[w/x]^{I \wedge w \in T}, A, \theta)\} \\
 & = & \text{“By Theorem A.1”} \\
 & & \sup\{\{w\} \mid \exists (V \setminus W)\, C \bullet \upsilon(S_1[w/x]^{I \wedge w \in T}, A, \theta)\} \\
 & = & \text{“By } Ler(S, A, Fv(\theta)) = true \text{ , Theorem 4.10 (ii) and Theorem A.2”} \\
 & & \sup\{\{w\} \mid L(\exists (V \setminus W) C) \bullet \upsilon(S_1[w/x]^{I \wedge w \in T}, A, \theta)\} \\
 & = & \text{“By Theorem A.1”} \\
 & & \sup\{\{w\} \mid L(C) \bullet \upsilon(S_1[w/x]^{I \wedge w \in T}, A, \theta)\} \\
 & = & \text{“By Table 8”} \\
 & & \upsilon(S, A, \theta)
\end{array}
$$

The induction step for sequential composition $S \stackrel{\text{def}}{=} (S_1 \ ; \ S_2)^I$. Because of the assumption on path S and Table 8 we have the following equivalences.

$$
\begin{array}{lcl}
true & \equiv & Ler(S, A, Fv(\theta)) \\
 & \equiv & Ler(S_2^I, A, Fv(\theta)) \wedge Ler(S_1^I, \Theta(S_2^I, A), Lv(S_2^I, A, Fv(\theta))) \\
 & \equiv & \text{“By Theorem 4.10 } (ii)\text{”} \\
 & & Ler(S_2^I, A, Fv(\theta)) \wedge Ler(S_1^I, \Theta(S_2^I, A), Fv(\upsilon(S_2^I, A, \theta)))
\end{array}
$$

Then, according to the induction hypothesis for S_1^I and S_2^I the following implications hold.

$$\Theta(S_2^I, A) \ \Rrightarrow \ \Upsilon(S_2^I, A, \theta) = \upsilon(S_2^I, A, \theta) \tag{7}$$

$$\Theta(S_1^I, \Theta(S_2^I, A)) \ \Rrightarrow \ \Upsilon(S_1^I, \Theta(S_2^I, A), \upsilon(S_2^I, A, \theta)) = \upsilon(S_1^I, \Theta(S_2^I, A), \upsilon(S_2^I, A, \theta)) \tag{8}$$

With the help of Theorem 3.2 this can be used to derive the following equalities. For this we assume again that $\Theta(S, A)$ holds.

$$
\begin{array}{lcl}
\Upsilon(S, A, \theta) & = & \text{“By Table 2”} \\
 & & \Upsilon(S_1^I, \Theta(S_2^I, A), \Upsilon(S_2^I, A, \theta)) \\
 & = & \text{“By induction hypothesis (7) and Corollary 3.4 ”} \\
 & & \Upsilon(S_1^I, \Theta(S_2^I, A), \upsilon(S_2^I, A, \theta)) \\
 & = & \text{“By induction hypothesis (8) ”} \\
 & & \upsilon(S_1^I, \Theta(S_2^I, A), \upsilon(S_2^I, A, \theta)) \\
 & = & \text{“By Theorem 4.10 } (i)\text{ ”} \\
 & & \upsilon(S_1^I, L(\Theta(S_2^I, A)), \upsilon(S_2^I, A, \theta)) \\
 & = & \text{“By Table 8 ”} \\
 & & \upsilon(S, A, \theta) \qquad \square
\end{array}
$$

With Table 8 appropriately modified to define a function $\lambda(S, A, \theta)$, i.e., by replacing the supremum with the infimum in the definition of $\upsilon(S, A, \theta)$, a similar theorem can be obtained for the best case execution time $\Lambda(S, A, \theta)$ and its approximation $\lambda(S, A, \theta)$.

4.3. An implementation of worst and best case execution time computation

In Theorem 3.2 we showed that the computation of execution times for timed paths can be performed by calculating the supremum and infimum of certain real-valued expressions. Furthermore, in the previous

sections we provided a linear approximation for timed paths, based on linear predicates. In this section we show how standard programming techniques can be used to implement execution time computation for timed paths.

For the computation of predicates in Presburger Arithmetic, Bledsoe and Shostak [Ble74, Ble75, Sho77] defined a number of functions: for instance the function $SUP_M(A, B)$ applied to a set of non-strict linear inequalities M, a mini-linear form A, and a set of variables B, returns a mini-linear form which under certain circumstances equals the supremum of A over the solutions of M dependent on the variables in B. Similarly, for a maxi-linear form A, the function $INF_M(A, B)$ returns the infimum of A over the solutions of M dependent on the variables in B.

In the following we use the three functions $SIMP(A)$, $SUP_M(A, B)$ and $INF_M(A, B)$ [Sho77] for the approximation of the expressions $\Upsilon(S)$ and $\Lambda(S)$ for timed paths S. The function $SIMP$ applied to a quasi-linear form A distributes $*$ over $+$ and pulls maxima and minima out to the front and returns an equivalent quasi-linear form. For instance,

$$SIMP(4 * (3 + Max(2, x)) \quad = \quad Max(20, 12 + 4 * x)$$

According to Shostak [Sho77, Theorem 16 and 17], $SUP_M(A, B)$ and $INF_M(A, B)$ will always return the correct value if A equals a variable. But, for an arbitrary linear form A this is not true in general. Since we intend to use Shostak's functions for worst and best case execution time computations with maxi and mini-linear forms it is necessary to provide a transformation procedure that translates any supremum expression with maxi or mini-linear forms into a supremum over a single variable. We therefore define the following functions $Sup_M(A, B)$ and $Inf_M(A, B)$ which can be later applied in the more general situation where A equals a mini (maxi)-linear form to produce the correct answer.

Let M be a set of linear inequalities, B a set of variables and A be a mini (maxi)-linear form.

$$Sup_M(A, B) \stackrel{\text{def}}{=} \begin{cases} SUP_{M'}(f, B) & \text{iff } A = Min(A_1, .., A_n) \text{ and } f \text{ fresh}, \\ & \quad M' \stackrel{\text{def}}{=} M \cup \{f \leq A_i : 1 \leq i \leq n\} \\ SUP_M(\nu, B) & \text{iff } A = \nu \end{cases}$$

$$Inf_M(A, B) \stackrel{\text{def}}{=} \begin{cases} INF_{M'}(f, B) & \text{iff } A = Max(A_1, .., A_n) \text{ and } f \text{ fresh}, \\ & \quad M' \stackrel{\text{def}}{=} M \cup \{A_i \leq f : 1 \leq i \leq n\} \\ INF_M(\nu, B) & \text{iff } A = \nu \end{cases}$$

For a timed path S, the function Υ^a of Table 9 computes a linear approximation of the path traversal condition $\Theta(S)$ and the worst case execution time $\Upsilon(S)$. The function Υ^a is defined with the help of the above mentioned functions of Shostak's algorithms [Sho77] for computations in Presburger Arithmetic and the linear approximation L defined in Section 4.1. A simple transformation of linear inequalities into non-strict ones is performed with the function $\mathcal{O}$. This function transforms any set of linear inequalities M into a set of linear inequalities without strict linear inequalities by simply replacing every strict inequality sign $<$ and $>$ by $\leq$ or $\geq$, respectively. The following implication holds trivially when interpreting a set M of linear inequalities as a linear predicate by taking the conjunction of all linear inequalities in M.

$$M \quad \Rrightarrow \quad \mathcal{O}(M) \tag{9}$$

Finally, the boolean valued function Sat is used as a satisfiability check on linear predicates. Applied to a set M of linear inequalities, $Sat(M)$ returns 'true' if and only if the set of linear inequalities M has a solution. $Sat(M)$ can be implemented as a straightforward variable elimination procedure. Such a procedure has been discussed in Section 4.1 for the implementation of the existential quantifier on linear inequalities. With these notions the following theorem holds. The proof is given in Appendix B.

Theorem 4.11 (Execution time computation). Let S be a timed path and let A'_k, B'_k be predicates and θ'_k, $k \in \mathcal{J}'$, mini-linear forms that are not dependent on τ. Let the set $\{(A'_k, B'_k, \theta'_k + \tau) : k \in \mathcal{J}'\}$ be denoted by M and let $\Upsilon^a(S, M)$ be denoted by the indexed set $\{(A_j, B_j, \theta_j) : j \in \mathcal{J}\}$. If $\Upsilon^a(S, M)$ is nonempty then the following properties hold.

i) For every $j \in \mathcal{J}$, B_j is a conjunctive linear predicate, A_j is equivalent to a conjunctive linear predicate and θ_j is a mini-linear form such that $\theta_j = \overline{\theta}_j + \tau$ where $\overline{\theta}_j$ does not depend on τ.

ii) $\Theta^a(S, \vee_{k \in \mathcal{J}'} A'_k) \equiv \vee_{j \in \mathcal{J}} L(A_j)$

iii) $\tau + \Upsilon(S, \vee_{k \in \mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}) \leq \sup_{\mathcal{J}}\{A_j \bullet \theta_j\}$.

Table 9. An abstract algorithm for worst case execution time computation based on the path semantics.

Command S	Linear execution time approximation $\Upsilon^a(S, M)$ where $M \stackrel{\text{def}}{=} \{(A_i, B_i, \theta_i) : i \in \mathcal{I}\}$
$(x{:}\,[Q])^I$	$\cup_{i\in\mathcal{I}}\mathcal{D}_i$, where $\mathcal{E}_i \stackrel{\text{def}}{=} L'(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A_i)[\tau', x'/\tau, x])\,,\ i \in \mathcal{I}$ $A(B) \stackrel{\text{def}}{=} \exists \tau', x'\ B$ $C(B,\theta) \stackrel{\text{def}}{=} SIMP(Sup_{\mathcal{O}(B)}(\theta[\tau', x'/\tau, x], (Fv(B) \cup Fv(\theta[\tau', x'/\tau, x])) \setminus \{\tau', x'\}))$ $\mathcal{D}_i \stackrel{\text{def}}{=} \{(A(B), B, C(B,\theta_i)) : B \in \mathcal{E}_i \wedge Sat(B)\}\,,\ i \in \mathcal{I}$
$(\mathbf{var}\ v : T \bullet S_1)^I$	$\cup_{j\in\mathcal{J}}\mathcal{D}_j$, where $\{(A_j, B_j, \theta_j) : j \in \mathcal{J}\} \stackrel{\text{def}}{=} \Upsilon^a(S_1[w/v]^{I\wedge w\in T}, M)$ $\mathcal{E}_j \stackrel{\text{def}}{=} L'(A_j)\,,\ j \in \mathcal{J}$ $A(B) \stackrel{\text{def}}{=} \exists w\ B$ $C(B,\theta) \stackrel{\text{def}}{=} SIMP(Sup_{\mathcal{O}(B)}(\theta, (Fv(B) \cup Fv(\theta)) \setminus \{w\}))$ $\mathcal{D}_j \stackrel{\text{def}}{=} \{(A(B), B, C(B,\theta_j)) : B \in \mathcal{E}_j \wedge Sat(B)\}\,,\ j \in \mathcal{J}$ with $w \notin Idf(S)$ and not free in expression M
$(S_1^{I_1})^{I_2}$	$\Upsilon^a(S_1^{I_1\wedge I_2}, M)$
$(S_1\ ;\ S_2)^I$	$\Upsilon^a(S_1^I, \Upsilon^a(S_2^I, M))$

iv) If all A'_k, $k \in \mathcal{J}'$ are linear predicates and $Ler(S, \vee_{k\in\mathcal{J}'} A'_k, Fv(\sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\})) = true$, in other words, if a linear execution time reduction exists, then $\Theta(S, \vee_{k\in\mathcal{J}'} A'_k) \Rrightarrow \tau + \Upsilon(S, \vee_{k\in\mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}) = \sup_{\mathcal{J}}\{A_j \bullet \theta_j\}$ holds.

By running $\Upsilon^a(S, \{(true\,, true\,, \tau)\})$ we obtain either an empty set in the case the linear approximation of the path traversal condition $\Theta^a(S)$ is unsatisfiable or, we obtain a set $\{(A_j, B_j, \theta_j) : j \in \mathcal{J}\}$ with $\vee_{j\in\mathcal{J}} L(A_j) \equiv \Theta^a(S)$ and $\Upsilon(S) \leq \sup_{\mathcal{J}}\{A_j \bullet \theta_j\} - \tau$. This clearly defines a linear approximation of the path traversal condition $\Theta(S)$ and the worst case execution time $\Upsilon(S)$. In case that a linear reduction $Ler(S, true\,, \emptyset)$ is 'true' the worst case execution time approximation is exact.

The index set $\mathcal{J}$ reflects the number of linear disjuncts encountered during the computation. If a disjunctive predicate is encountered with N disjuncts the algorithm computes N different expressions. The number of computation steps is hence exponentially dependent on the number of disjuncts in the predicates of S.

For each of the transmitter paths, this algorithm returns the worst case execution time that we stated in Table 6. A linear execution time reduction is possible in every case and so the results are exact. Note that a linear execution time reduction of a timed path S is trivially possible if S is constructed from quantified linear predicates only.

A function Λ^a can be defined for the approximation of best case execution times by replacing function *Sup* in Table 9 with function *Inf*. Theorem 4.11 can then be reformulated with all inequalities reversed, and 'supremum' expressions, mini-linear forms and Υ^a replaced by 'infimum' expressions, maxi-linear forms, and Λ^a.

4.4. Approximation of weakest and strongest execution time constraints

This section discusses the linear approximation of the strongest and weakest execution time constraints $SETC(S)$ and $WETC(S)$ on the base of the algorithm in Table 9. These constraints were defined in Definition 3.6 with the help of best and worst case execution times. In the following we show how linear approximations $WETC^a(S)$ and $SETC^a(S)$ of $SETC(S)$ and $WETC(S)$ can be defined with the functions Υ^a and Λ^a of the previous section.

Let S be a timed path with nonempty set $\Upsilon^a(S, \{(true\,, true\,, \tau)\})$. Let the indexed set $\{(A_i, B_i, \theta_i) : i \in \mathcal{I}\}$ denote $\Upsilon^a(S, \{(true\,, true\,, \tau)\})$. According to Theorem 4.11 we can assume that each A_i is equivalent to a conjunctive linear predicate and each θ_i is a mini-linear form $Min(L^i_1, ..., L^i_{n_i})$, with linear forms L^i_j or $L^i_j \in \{-\infty, \infty\}$, $1 \leq j \leq n_i$, $i \in \mathcal{I}$.

Recall the functions *Inf* and $\mathcal{O}$ of Section 4.3. Then we define $WETC^a(S)$ as follows.

$$WETC^a(S) \stackrel{\text{def}}{=} \inf_{i\in\mathcal{I}} \inf_{1\leq j\leq n_i} Inf_{\mathcal{O}(L(A_i))}(L^i_j - \tau, \emptyset) \tag{10}$$

Similarly, function $SETC^a$ is defined by replacing the 'infimum' with the 'supremum', by replacing Υ^a with Λ^a and by replacing *Inf* by *Sup* in definition (10). Then, Theorem 4.11 and Theorem 4.10 imply the following relationships between $WETC^a(S)$ and $WETC(S)$.

Theorem 4.12 (Constraint approximation). Let S be a timed path with $\Upsilon^a(S, \{(true, true, \tau)\}) \neq \emptyset$. If a linear execution time reduction of S exists, i.e., if $Ler(S, true, \emptyset) = true$, then $WETC^a(S) \leq WETC(S)$ holds.

Proof. Let $\mathcal{V}$ denote the free variables of $\Theta(S)$ and $\Upsilon(S)$, and let $\{(A_i, B_i, \theta_i) : i \in \mathcal{I}\}$ denote the expression $\Upsilon^a(S, \{(true, true, \tau)\}$. We can assume that each θ_i is a mini-linear form $Min(L^i_1, ..., L^i_{n_i})$, with linear forms L^i_j or $L^i_j \in \{-\infty, \infty\}$, $1 \leq j \leq n_i$, $i \in \mathcal{I}$. Then the following correspondence holds.

$$
\begin{aligned}
WETC(S) &= \inf\{\mathcal{V} \mid \Theta(S) \bullet \Upsilon(S)\} \\
&= \text{“By Theorem 4.11 } (iv)\text{”} \\
&\quad \inf\{\mathcal{V} \mid \Theta(S) \bullet \sup_{i\in\mathcal{I}}\{A_i \bullet \theta_i\} - \tau\} \\
&\geq \text{“By Theorem 4.11 } (ii)\text{, Theorem 4.5 } (i) \text{ and monotonicity of infimum”} \\
&\quad \inf\{\mathcal{V} \mid \vee_{i\in\mathcal{I}} L(A_i) \bullet \sup_{i\in\mathcal{I}}\{A_i \bullet \theta_i\} - \tau\} \qquad (11) \\
&= \text{“By Theorem A.3 for infimum instead of supremum ”} \\
&\quad \inf_{i\in\mathcal{I}} \inf\{\mathcal{V} \mid L(A_i) \bullet \sup_{i\in\mathcal{I}}\{A_i \bullet \theta_i\} - \tau\} \\
&\geq \text{“By Theorem 4.11 } (i)\text{, } L(A_i) \equiv A_i \text{ and monotonicity of infimum”} \\
&\quad \inf_{i\in\mathcal{I}} \inf\{\mathcal{V} \mid L(A_i) \bullet \theta_i - \tau\} \\
&= \text{“By monotonicity of infimum on mini-linear forms ”} \\
&\quad \inf_{i\in\mathcal{I}} \inf\{\mathcal{V} \mid \mathcal{O}(L(A_i)) \bullet \theta_i - \tau\} \\
&= \inf_{i\in\mathcal{I}} \inf_{1\leq j\leq n_i} \inf\{\mathcal{V} \mid \mathcal{O}(L(A_i)) \bullet L^i_j - \tau\} \\
&= \text{“By [Sho77, Theorem 16]”} \\
&\quad \inf_{i\in\mathcal{I}} \inf_{1\leq j\leq n_i} Inf_{\mathcal{O}(L(A_i))}(L^i_j - \tau, \emptyset) \\
&= WETC^a(S) \qquad \square
\end{aligned}
$$

This theorem tells us that weakest execution time constraints can be approximated if linear execution time reductions exist. If no execution time reduction exists anything can happen since worst case execution time approximations can be far too pessimistic and inequality (11) may be violated. Exact approximations for execution time constraints are more difficult to achieve than for execution times. From the proof of Theorem 4.12 it is possible to derive conditions for equality between $WETC^a(S)$ and $WETC(S)$. For instance if in addition to the assumptions made in Theorem 4.12, a linear reduction of the triple $(\exists(\mathcal{V} \backslash Fv(\Upsilon(S)))\, \Theta(S), \mathcal{V}, Fv(\Upsilon(S)))$ exists, i.e., if $Lr(\Theta(S), \mathcal{V}, Fv(\Upsilon(S))) = true$ and if the predicates A_i, $i \in \mathcal{I}$ are mutually exclusive, i.e., if $A_i \wedge A_j \equiv false$ for $i \neq j$ holds, then we can conclude $WETC^a(S) = WETC(S)$.

A similar theorem holds for $SETC^a(S)$ and the strongest execution time constraint $SETC(S)$.

5. Conclusions

We outlined the implementation of a semantics based execution time and timing constraint derivation theory and its use for the timing analysis of real-time programs with deadline commands. The abstract algorithm of Table 2 for the computation of execution times is of major importance, beyond the application in high-level timing analysis, since any execution time computation or approximation methodology, no matter whether high or low-level, is based on some kind of program semantics, and thus, in principle can be modelled with the help of our abstract command language.

We did not incorporate a statement for nondeterministic choice (apart from the implicit nondeterminism that is provided by specification statements). In principle, this can be handled within our approach [LFH02b,

LFH02c] by providing an explicit statement for nondeterministic choice based on majorisation over sets of paths. This would be consistent with common approximation methods using recursion bounds for loops and subroutines. The notion of a linear projection of predicates is an abstraction of the linear approximation process of nonlinear functions. In general, the accuracy of execution time approximation depends on the quality of the projection but also on the quality of the predicates of the timed path. The first topic has not been pursued further in this paper, but we elaborated on the latter and provided criteria for the reduction of predicates to linear predicates during the execution time computation which enable exact execution time computations in many cases.

The implementation for execution time derivation and constraint computation relies on Shostak's extension of Bledsoe's supremum/infimum algorithm [Ble74, Ble75, Sho77]. This is a fast and sound, but not complete, method for the correctness proof of theorems in Presburger Arithmetic. The original problem situated in integer linear arithmetic is transformed into an optimisation problem over the real numbers. In this wider environment Bledsoe-Shostak's algorithm provides a solution for the original problem. We apply this algorithm in a completely different framework to the computation of execution times and timing constraints over the real numbers. The key for such an application is that we can formulate execution time and constraint determination for timed paths as an optimisation problem over the reals. Note that in this setting Bledsoe-Shostak's algorithm then provides a sound and complete method.

Our theoretical framework and implementation for execution time and constraint determination are based on three main problem areas: the satisfiability of constraints; the simplification of expressions; and the optimisation of functions over sets of constraints. To ensure that the supremum/infimum algorithm succeeds for a given timed path it is necessary to check whether the linear projection of the path is not dead or in other words whether the linear projection of the path traversal condition is satisfiable. To enable accurate execution time and constraint determination and check whether such is possible for programs it is important to have techniques to remove unnecessary predicates during the computation. Furthermore we have shown that the problem of determining worst case and best case execution times and timing constraints can be formulated as an abstract optimisation problem over the real numbers. These three issues are also dominating Constraint Logic Programming languages [MS98] where highly developed and explored techniques are provided for each of the problem areas. A prototype of our framework which is capable of handling examples like the simple transmitter program of this paper has been implemented in Prolog which provides limited support for Constraint Logic Programming. For the complete implementation of our approach we plan to use a Constraint Logic Programming language to fully apply predicate simplification and optimisation techniques of constraints.

Acknowledgements

We wish to thank Sibylle Peuker for correcting errors in the manuscript and John Nickerson for implementing a protoype of our approach. This research was funded by Australian Research Council Large Grant A49937045, *Effective Real-Time Program Analysis.*

A. Basic Theorems on Suprema

The following theorems were used in the proofs above.

Theorem A.1. Let $\mathcal{V} \subseteq Var$, $y \in \mathcal{V}$, B a predicate and θ a function on Bnd with values in $\mathbb{R} \cup \{\infty, -\infty\}$. If y is not a free variable of expression θ then $\sup\{\mathcal{V} | B \bullet \theta\} = \sup\{\mathcal{V} \setminus \{y\} | \exists y\, B \bullet \theta\}$.

Proof. Let $\beta \in Bnd$. For any binding $b \in Bnd$ with $B(b) = true$ and b, β being equal on the set $Var \setminus \mathcal{V}$ the following binding b_0,

$$b_0(v) \stackrel{\text{def}}{=} \begin{cases} b(v) & : \quad v \neq y \\ \beta(y) & : \quad v = y \end{cases}$$

has the properties that b_0 equals β on the variable set $(Var \setminus \mathcal{V}) \cup \{y\}$ and $(\exists y\, B)(b_0) = true$.

On the other hand, for any binding $b \in Bnd$ with $(\exists y\, B)(b) = true$ such that b equals β on the variable set $(Var \setminus \mathcal{V}) \cup \{y\}$, there is a $y_0 \in Val$ such that the following binding b_0,

$$b_0(v) \stackrel{\text{def}}{=} \begin{cases} b(v) & : \; v \neq y \\ y_0 & : \; v = y \end{cases}$$

has the property, $B(b_0) = true$ and furthermore, b_0 equals β on the set of variable $Var \setminus \mathcal{V}$. With this information the following equalities hold.

$$\begin{aligned} & \sup\{\mathcal{V} | B \bullet \theta\}(\beta) \\ = \; & \sup\{\theta(b) \,:\, b \in Bnd \wedge B(b) \wedge b, \beta \text{ are equal on } Var \setminus \mathcal{V}\} \\ = \; & \sup\{\theta(b) \,:\, b \in Bnd \wedge (\exists y\, B)(b) \wedge b, \beta \text{ are equal on } (Var \setminus \mathcal{V}) \cup \{y\}\} \\ = \; & \sup\{\mathcal{V} \setminus \{y\} | \exists y\, B \bullet \theta\}(\beta) \qquad \square \end{aligned}$$

For a set of variables $\mathcal{V} = \{v_1, ..., v_n\}$ and a predicate B we will in the following abbreviate the predicate $\exists v_1, ..., v_n\, B$ by $\exists \mathcal{V}\, B$.

Theorem A.2. Let $\mathcal{V}$ be a finite subset of Var, let A and B be predicates with no common free variables in $\mathcal{V}$, let $\beta \in Bnd$ and let θ be a function on Bnd with values in $\mathbb{R} \cup \{\infty, -\infty\}$. If B and θ have no common free variables in $\mathcal{V}$, and the equality $(\exists(\mathcal{V} \cap Fv(B))\, B)(\beta) = true$ holds, then $\sup\{\mathcal{V} | A \wedge B \bullet \theta\}(\beta) = \sup\{\mathcal{V} | A \bullet \theta\}(\beta)$.

Proof. The assertion follows by application of Theorem A.1.

$$\begin{aligned} & \sup\{\mathcal{V} | A \wedge B \bullet \theta\}(\beta) \\ = \; & \text{“Since } B, \theta \text{ have no common free variables in } \mathcal{V} \text{ and Theorem } A.1\text{”} \\ & \sup\{\mathcal{V} \setminus Fv(B) | \exists(\mathcal{V} \cap Fv(B))\, (A \wedge B) \bullet \theta\}(\beta) \\ = \; & \text{“Since } A, B \text{ have no common free variables in } \mathcal{V} \text{ ”} \\ & \sup\{\mathcal{V} \setminus Fv(B) | A \wedge \exists(\mathcal{V} \cap Fv(B))\, B \bullet \theta\}(\beta) \\ = \; & \text{“ } (\exists(\mathcal{V} \cap Fv(B))\, B)(\beta) = true\text{”} \\ & \sup\{\mathcal{V} | A \bullet \theta\}(\beta) \qquad \square \end{aligned}$$

The proofs for the following theorems for the supremum over predicate disjunctions and nested suprema with conjunctions of predicates are similar to the ones shown above.

Theorem A.3. Let $\mathcal{V}$ be a finite set of variables, A and B two predicates, $\beta \in Bnd$ and θ a function on Bnd with values in $\mathbb{R} \cup \{\infty, -\infty\}$. Then the following equality holds.

$$\sup\{\mathcal{V} | A \vee B \bullet \theta\}(\beta) \quad = \quad \sup\{\sup\{\mathcal{V} | A \bullet \theta\}(\beta), \sup\{\mathcal{V} | B \bullet \theta\}(\beta)\}$$

Theorem A.4. Let $\mathcal{V}_1$, $\mathcal{V}_2$ be finite sets of variables, let A and B be predicates with A having no free variables in $\mathcal{V}_2$ and let θ be a function on Bnd with values in $\mathbb{R} \cup \{\infty, -\infty\}$. Then the following equality holds.

$$\sup\{\mathcal{V}_1 \cup \mathcal{V}_2 | A \wedge B \bullet \theta\} \quad = \quad \sup\{\mathcal{V}_1 \,|\, A \bullet \sup\{\mathcal{V}_2 \,|\, B \bullet \theta\}\}$$

Proof. Let β be a binding in Bnd. Then, the following holds.

$$\begin{aligned} & \sup\{\mathcal{V}_1 \,|\, A \bullet \sup\{\mathcal{V}_2 \,|\, B \bullet \theta\}\}(\beta) \\ = \; & \sup\{\sup\{\theta(b') \,:\, b' \in Bnd \wedge B(b') \wedge b', b \text{ equal on } Var \setminus \mathcal{V}_2\} \,: \\ & \quad b \in Bnd \wedge A(b) \wedge \beta, b \text{ equal on } Var \setminus \mathcal{V}_1\} \\ = \; & \sup\{\theta(b') \,:\, b, b' \in Bnd \wedge B(b') \wedge b', b \text{ equal on } Var \setminus \mathcal{V}_2 \wedge A(b) \wedge \beta, b \text{ equal on } Var \setminus \mathcal{V}_1\} \\ = \; & \text{“By } A \text{ having no free variables in } \mathcal{V}_2 \text{ ”} \\ & \sup\{\theta(b') \,:\, b' \in Bnd \wedge B(b') \wedge b', b \text{ equal on } Var \setminus \mathcal{V}_2 \wedge A(b') \wedge \beta, b \text{ equal on } Var \setminus \mathcal{V}_1\} \\ = \; & \sup\{\theta(b') \,:\, b' \in Bnd \wedge B(b') \wedge b', \beta \text{ equal on } Var \setminus (\mathcal{V}_1 \cup \mathcal{V}_2) \wedge A(b')\} \\ = \; & \sup\{\mathcal{V}_1 \cup \mathcal{V}_2 | A \wedge B \bullet \theta\}(\beta) \qquad \square \end{aligned}$$

B. Proof of Theorem 4.11

Proof. The proof is by structural induction on the commands in Table 9 and with the help of Theorems 3.2 and 4.10 , and Table 2.

Assertion (i): That B_j is and that A_j is equivalent to, a conjunctive linear predicate follows from the definition of the projections L and L', and function Υ^a. According to Shostak [Sho77, Theorem 12], for the case of B being a satisfiable set of non-strict, linear inequalities, A a mini-linear form and $\mathcal{V}$ being a set of variables, $SIMP(SUP_M(A, \mathcal{V}))$ returns a mini-linear form depending on the variables in $\mathcal{V}$. Because of this and since $\mathcal{O}$ applied to a satisfiable set of linear inequalities B, returns a set of satisfiable linear inequalities, all functions θ_j that are returned by Υ^a are mini-linear forms. That θ_j is of the form $\overline{\theta}_j + \tau$ with τ not free in $\overline{\theta}_j$ follows by structural induction. We omit this proof.

Assertion (ii): The base case $S \stackrel{\text{def}}{=} (x{:}[Q])^I$. Table 9 provides the following definition.

$$\mathcal{E}_k \stackrel{\text{def}}{=} L'(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A'_k)[x', \tau'/x, \tau]) \,,\; k \in \mathcal{J}' \tag{12}$$

According to property (5) and the definition of Υ^a, the following equivalence holds.

$$\begin{array}{lll}
\vee_{j\in\mathcal{J}} L(A_j) & \equiv & \text{“By Definition 4.2 ”} \\
& & L(\vee_{j\in\mathcal{J}} A_j) \\
& \equiv & \text{“By Table 9”} \\
& & L(\vee \{A(B) : B \in \cup_{k\in\mathcal{J}'} \mathcal{E}_k \wedge Sat(B)\}) \\
& \equiv & \text{“By Table 9”} \\
& & L(\vee \{\exists \tau', x' \; B : B \in \cup_{k\in\mathcal{J}'} \mathcal{E}_k \wedge Sat(B)\}) \\
& \equiv & \text{“By moving the existential quantifier to the front”} \\
& & L(\exists \tau', x' \; (\vee\{B : B \in \cup_{k\in\mathcal{J}'} \mathcal{E}_k \wedge Sat(B)\})) \\
& \equiv & \text{“By adding unsatisfiable predicates } B \text{ and Definition 4.2 ”} \\
& & L(\exists \tau', x' \; (\vee\{B : B \in \cup_{k\in\mathcal{J}'} \mathcal{E}_k\})) \\
& \equiv & L(\exists \tau', x' \; (\vee_{k\in\mathcal{J}'} \vee \mathcal{E}_k)) \\
& \equiv & \text{“By definition (12)”} \\
& & L(\exists \tau', x' \; (\vee_{k\in\mathcal{J}'} \vee \; L'(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A'_k)[x', \tau'/x, \tau]))) \\
& \equiv & \text{“By property (5)”} \\
& & L(\exists \tau', x' \; (\vee_{k\in\mathcal{J}'} L(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A'_k)[x', \tau'/x, \tau]))) \\
& \equiv & \text{“By Definition 4.2 and distributivity”} \\
& & L(\exists \tau', x' \; (I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge (\vee_{k\in\mathcal{J}'} A'_k))[x', \tau'/x, \tau])) \\
& \equiv & \text{“By Table 7”} \\
& & \Theta^a(S, \vee_{k\in\mathcal{J}'} A'_k)
\end{array}$$

The step case $S \stackrel{\text{def}}{=} (\mathbf{var}\, v \; : \; T \bullet S_1)^I$. Let $\{(A''_j, B''_j, \theta''_j) \; : \; j \in \mathcal{J}\}$ denote $\Upsilon^a(S_1[w/v]^{I\wedge w\in T}, M)$. The induction hypothesis for this case is as follows.

$$\Theta^a(S_1[w/x]^{I\wedge w\in T}, \vee_{k\in\mathcal{J}'} A'_k) \quad \equiv \quad \vee_{j\in\mathcal{J}} L(A''_j) \tag{13}$$

Table 9 defines the sets $\mathcal{E}_j$ by

$$\mathcal{E}_j \stackrel{\text{def}}{=} L'(A''_j) \,,\; j \in \mathcal{J} \,. \tag{14}$$

The following equivalences result.

$$\begin{array}{lll}
\vee_{j\in\mathcal{J}} L(A_j) & \equiv & \text{“By Definition 4.2 ”} \\
& & L(\vee_{j\in\mathcal{J}} A_j) \\
& \equiv & \text{“By Table 9”} \\
& & L(\vee \cup_{j\in\mathcal{J}} \{A(B) : B \in \mathcal{E}_j\})
\end{array}$$

$\equiv$ "By Table 9"
$L(\vee \cup_{j\in\mathcal{J}} \{\exists w\, B : B \in \mathcal{E}_j\})$
$\equiv$ "By shifting the existential quantifier outside"
$L(\exists w\, (\vee \cup_{j\in\mathcal{J}} \{B : B \in \mathcal{E}_j\})$
$\equiv$ "By equality (14)"
$L(\exists w\, (\vee \cup_{j\in\mathcal{J}} L'(A''_j)))$
$\equiv$ "By Definition 4.2 and properties of projections L, L' and relation (5)"
$L(\exists w\, (\vee_{j\in\mathcal{J}} L(A''_j)))$
$\equiv$ "By induction hypothesis (13) and Definition 4.2"
$L(\exists w\, (\Theta^a(S_1[w/x]^{I\wedge w\in T}, \vee_{k\in\mathcal{J}'} L(A'_k))))$
$\equiv$ "By Table 7"
$\Theta^a(S, \vee_{k\in\mathcal{J}'} L(A'_k))$

The induction step for $S \stackrel{\text{def}}{=} (S_1 \,;\, S_2)^I$. Then, according to Table 9 the following equivalence holds.

$$\Upsilon^a(S, M) \;=\; \Upsilon^a(S_1^I, \Upsilon^a(S_2^I, M))$$

We abbreviate $\Upsilon^a(S_2^I, M)$ by $\{(A''_l, B''_l, \theta''_l) : l \in \mathcal{J}''\}$. According to induction hypothesis for S_1^I and S_2^I the following properties hold.

$$\vee_{l\in\mathcal{J}''} L(A''_l) \;\equiv\; \Theta^a(S_2^I, \vee_{k\in\mathcal{J}'} A'_k) \tag{15}$$
$$\vee_{j\in\mathcal{J}} L(A_j) \;\equiv\; \Theta^a(S_1^I, \vee_{l\in\mathcal{J}''} A''_l) \tag{16}$$

This can be combined to derive the following equivalence.

$\vee_{j\in\mathcal{J}} L(A_j) \equiv$ "By equivalence (16)"
$\Theta^a(S_1^I, \vee_{l\in\mathcal{J}''} A''_l)$
$\equiv$ "By Definition 4.2 and properties of projection L and function Θ^a "
$\Theta^a(S_1^I, \vee_{l\in\mathcal{J}''} L(A''_l))$
$\equiv$ "By equivalence (15)"
$\Theta^a(S_1^I, \Theta^a(S_2^I, \vee_{k\in\mathcal{J}'} A'_k))$
$\equiv$ "By Table 7"
$\Theta^a(S, \vee_{k\in\mathcal{J}'} A'_k)$

Assertion (iii): The base case $S \stackrel{\text{def}}{=} (x{:}[Q])^I$. According to Table 9 the following equality holds.

$\Upsilon(S, \vee_{k\in\mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\})$
$=$ "By Table 2"
$\sup\{\{\tau', x'\} \mid I \wedge \tau \le \tau' \wedge Q \wedge (I \wedge \vee_{k\in\mathcal{J}'} A'_k)[\tau', x'/\tau, x] \bullet \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}[\tau', x'/\tau, x] + \tau' - \tau\}$
$=$ "By distributivity and Theorem A.3"
$\sup_{k\in\mathcal{J}'} \sup\{\{\tau', x'\} \mid I \wedge \tau \le \tau' \wedge Q \wedge (I \wedge A'_k)[\tau', x'/\tau, x] \bullet \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}[\tau', x'/\tau, x] + \tau' - \tau\}$
$=$ "By Theorem A.4 "
$\sup_{k\in\mathcal{J}'} \sup_{l\in\mathcal{J}'} \sup\{\{\tau', x'\} \mid I \wedge \tau \le \tau' \wedge Q \wedge (I \wedge A'_k \wedge A'_l)[\tau', x'/\tau, x] \bullet \theta'_l[\tau', x'/\tau, x] + \tau' - \tau\}$
$=$ "By monotonicity of supremum "
$\sup_{k\in\mathcal{J}'} \sup\{\{\tau', x'\} \mid I \wedge \tau \le \tau' \wedge Q \wedge (I \wedge A'_k)[\tau', x'/\tau, x] \bullet \theta'_k[\tau', x'/\tau, x] + \tau' - \tau\}$
$\le$ "By property (3) and monotonicity of supremum"

$$
\begin{array}{ll}
 & \sup_{k \in \mathcal{J}'} \sup\{\{\tau', x'\} \mid L(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A'_k)[\tau', x'/\tau, x]) \bullet \theta'_k[\tau', x'/\tau, x] + \tau' - \tau\} \\
= & \text{“By property (5)”} \\
 & \sup_{k \in \mathcal{J}'} \sup\{\{\tau', x'\} \mid \vee L'(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A'_k)[\tau', x'/\tau, x]) \bullet \theta'_k[\tau', x'/\tau, x] + \tau' - \tau\} \\
= & \text{“By Theorem A.3 and Table 9”} \\
 & \sup_{k \in \mathcal{J}'} \sup_{B \in \mathcal{E}_k} \sup\{\{\tau', x'\} \mid B \bullet \theta'_k[\tau', x'/\tau, x] + \tau' - \tau\} \\
= & \text{“By leaving out unsatisfiable predicates } B \text{ ”} \\
 & \sup_{k \in \mathcal{J}'} \sup_{B \in \mathcal{E}_k \wedge Sat(B)} \sup\{\{\tau', x'\} \mid B \bullet \theta'_k[\tau', x'/\tau, x] + \tau' - \tau\} \\
= & \text{“By Theorem A.4”} \\
 & \sup_{k \in \mathcal{J}'} \sup_{B \in \mathcal{E}_k \wedge Sat(B)} \{\exists \tau', x' \; B \bullet \sup\{\{\tau', x'\} \mid B \bullet \theta'_k[\tau', x'/\tau, x] + \tau' - \tau\}\} \\
= & \text{“By property (9) and continuity of supremum on mini-linear forms”} \\
 & \sup_{k \in \mathcal{J}'} \sup_{B \in \mathcal{E}_k \wedge Sat(B)} \{\exists \tau', x' \; B \bullet \sup\{\{\tau', x'\} \mid \mathcal{O}(B) \bullet \theta'_k[\tau', x'/\tau, x] + \tau' - \tau\}\} \\
= & \text{“By [Sho77, Theorem 16]”} \\
 & \sup_{k \in \mathcal{J}'} \sup_{B \in \mathcal{E}_k \wedge Sat(B)} \{\exists \tau', x' \; B \bullet \\
 & \qquad SIMP(Sup_{\mathcal{O}(B)}(\theta'_k[\tau', x'/\tau, x] + \tau', (Fv(B) \cup Fv(\theta'_k[\tau', x'/\tau, x])) \setminus \{\tau', x'\})) - \tau\} \\
= & \text{“By Table 9”} \\
 & \sup_{\mathcal{J}}\{A_j \bullet \theta_j\} - \tau
\end{array}
$$

The step case $S \stackrel{\text{def}}{=} (\mathbf{var}\, v \; : \; T \bullet S_1)^I$. Let $\{(A''_j, B''_j, \theta''_j) \; : \; j \in \mathcal{J}\}$ denote $\Upsilon^a(S_1[w/v]^{I \wedge w \in T}, M)$. The induction hypothesis can be stated as follows.

$$\tau + \Upsilon(S_1[w/x]^{I \wedge w \in T}, \vee_{k \in \mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}) \quad \leq \quad \sup_{\mathcal{J}}\{A''_j \bullet \theta''_j\} \tag{17}$$

Table 9 defines the sets $\mathcal{E}_j$ by

$$\mathcal{E}_j \quad \stackrel{\text{def}}{=} \quad L'(A''_j)\,, \; j \in \mathcal{J}\;. \tag{18}$$

Then the following inequalities hold.

$$
\begin{array}{ll}
 & \Upsilon(S, \vee_{k \in \mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}) \\
= & \text{“By Table 2”} \\
 & \sup\{\{w\} \mid \Theta(S_1[w/x]^{I \wedge w \in T}, \vee_{k \in \mathcal{J}'} A'_k) \bullet \Upsilon(S_1[w/x]^{I \wedge w \in T}, \vee_{k \in \mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\})\} \\
\leq & \text{“By induction hypothesis (17) and monotonicity of supremum”} \\
 & \sup\{\{w\} \mid \Theta(S_1[w/x]^{I \wedge w \in T}, \vee_{k \in \mathcal{J}'} A'_k) \bullet \sup_{\mathcal{J}}\{A''_j \bullet \theta''_j\} - \tau\} \\
\leq & \text{“By property (3) and monotonicity of supremum”} \\
 & \sup\{\{w\} \mid L(\Theta(S_1[w/x]^{I \wedge w \in T}, \vee_{k \in \mathcal{J}'} A'_k)) \bullet \sup_{\mathcal{J}}\{A''_j \bullet \theta''_j\} - \tau\} \\
= & \text{“By Theorem 4.5 } (i) \text{ ”} \\
 & \sup\{\{w\} \mid \Theta^a(S_1[w/x]^{I \wedge w \in T}, \vee_{k \in \mathcal{J}'} A'_k) \bullet \sup_{\mathcal{J}}\{A''_j \bullet \theta''_j\} - \tau\} \\
= & \text{“By Theorem 4.11 } (ii) \text{”} \\
 & \sup\{\{w\} \mid \vee_{j \in \mathcal{J}} L(A''_j) \bullet \sup_{\mathcal{J}}\{A''_j \bullet \theta''_j\} - \tau\} \\
= & \text{“By property (5) and definition (18)”}
\end{array}
$$

$$
\begin{aligned}
& \sup\{\{w\} \mid \vee_{j\in\mathcal{J}} \vee_{B\in\mathcal{E}_j} B \bullet \sup_{\mathcal{J}}\{A''_j \bullet \theta''_j\} - \tau\} \\
= \quad & \text{“By Theorem A.3”} \\
& \sup_{j\in\mathcal{J}} \sup_{B\in\mathcal{E}_j} \sup\{\{w\} \mid B \bullet \sup_{\mathcal{J}}\{A''_j \bullet \theta''_j\} - \tau\} \\
= \quad & \text{“By Theorem A.4”} \\
& \sup_{j\in\mathcal{J}} \sup_{l\in\mathcal{J}} \sup_{B\in\mathcal{E}_j} \sup\{\{w\} \mid B \wedge A''_l \bullet \theta''_l\} - \tau \\
= \quad & \text{“By definition (18), Theorem 4.11 } (i)\text{, } A''_l \text{ quant. lin. predicates and monotonicity of sup.”} \\
& \sup_{j\in\mathcal{J}} \sup_{B\in\mathcal{E}_j} \sup\{\{w\} \mid B \bullet \theta''_j - \tau\} \\
= \quad & \text{“By leaving out unsatisfiable predicates } B \text{ ”} \\
& \sup_{j\in\mathcal{J}} \sup_{B\in\mathcal{E}_j \wedge Sat(B)} \sup\{\{w\} \mid B \bullet \theta''_j - \tau\} \\
= \quad & \text{“By Theorem A.4”} \\
& \sup_{j\in\mathcal{J}} \sup_{B\in\mathcal{E}_j \wedge Sat(B)} \{\exists w\, B \bullet \sup\{\{w\} \mid B \bullet \theta''_j - \tau\}\} \\
= \quad & \text{“By property (9) and continuity of supremum on mini-linear forms”} \\
& \sup_{j\in\mathcal{J}} \sup_{B\in\mathcal{E}_j \wedge Sat(B)} \sup\{\exists w\, B \bullet \sup\{\{w\} \mid \mathcal{O}(B) \bullet \theta''_j - \tau\}\} \\
= \quad & \text{“By Theorem 4.11 } (i) \text{ and [Sho77, Theorem 16] ”} \\
& \sup_{j\in\mathcal{J}} \sup_{B\in\mathcal{E}_j \wedge Sat(B)} \{\exists w\, B \bullet SIMP(Sup_{\mathcal{O}(B)}(\theta''_j, (Fv(B) \cup Fv(\theta''_j)) \setminus \{w\}))\} - \tau \\
= \quad & \text{“By Table 9”} \\
& \sup_{\mathcal{J}}\{A_j \bullet \theta_j\} - \tau
\end{aligned}
$$

It remains to show the induction step for sequential composition $S \stackrel{\text{def}}{=} (S_1 \,;\, S_2)^I$. According to Table 9 the following equality holds.

$$\Upsilon^a(S, M) \quad = \quad \Upsilon^a(S_1^I, \Upsilon^a(S_2^I, M))$$

We abbreviate $\Upsilon^a(S_2^I, M)$ by $\{(A''_l, B''_l, \theta''_l) \,:\, l \in \mathcal{J}''\}$. Because of Theorem 4.11 (i), it is possible to find mini-linear forms $\overline{\theta}''_l$, $l \in \mathcal{J}''$ not dependent on τ such that the following equalities hold.

$$\theta''_l \quad = \quad \overline{\theta}''_l + \tau\,,\; l \in \mathcal{J}'' \tag{19}$$

Then, according to the induction hypothesis for S_1^I and S_2^I the following inequalities hold.

$$\tau + \Upsilon(S_2^I, \vee_{k\in\mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}) \quad \leq \quad \sup_{\mathcal{J}''}\{A''_l \bullet \theta''_l\} \tag{20}$$

$$\tau + \Upsilon(S_1^I, \vee_{l\in\mathcal{J}''} A''_l, \sup_{\mathcal{J}''}\{A''_l \bullet \overline{\theta}''_l\}) \quad \leq \quad \sup_{\mathcal{J}}\{A_j \bullet \theta_j\} \tag{21}$$

Furthermore, because of Theorem 4.5 (i) and Theorem 4.11 (ii) the following predicate equivalence holds.

$$
\begin{aligned}
L(\Theta(S_2^I, \vee_{k\in\mathcal{J}'} A'_k)) \quad \equiv \quad & \text{“By Theorem 4.11 } (ii)\text{”} \\
& \Theta^a(S_2^I, \vee_{k\in\mathcal{J}'} A'_k) \\
\equiv \quad & \text{“By Theorem 4.5 } (i) \text{ and } (ii)\text{”} \\
& \vee_{l\in\mathcal{J}''} L(A''_l)
\end{aligned} \tag{22}
$$

With the help of Theorem 3.2 this can be then used to verify the following inequalities.

$$
\begin{aligned}
& \tau + \Upsilon(S, \vee_{k\in\mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}) \\
= \quad & \text{“By Table 2”}
\end{aligned}
$$

$$
\begin{aligned}
& \tau + \Upsilon(S_1^I, \Theta(S_2^I, \vee_{k\in\mathcal{J}'} A'_k), \Upsilon(S_2^I, \vee_{k\in\mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\})) \\
\leq\ & \text{“By Corollary 3.4 and implication (3)”} \\
& \tau + \Upsilon(S_1^I, L(\Theta(S_2^I, \vee_{k\in\mathcal{J}'} A'_k)), \Upsilon(S_2^I, \vee_{k\in\mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{\ A'_k \bullet \theta'_k\})) \\
=\ & \text{“By equivalence (22)”} \\
& \tau + \Upsilon(S_1^I, \vee_{l\in\mathcal{J}''} L(A''_l), \Upsilon(S_2^I, \vee_{k\in\mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\})) \\
\leq\ & \text{“By Corollary 3.4, inequality (20) and Theorem 4.5 } (i)\text{”} \\
& \tau + \Upsilon(S_1^I, \vee_{l\in\mathcal{J}''} A''_l, \sup_{\mathcal{J}''}\{A''_l \bullet \theta''_l\} - \tau) \\
=\ & \text{“By property (19)”} \\
& \tau + \Upsilon(S_1^I, \vee_{l\in\mathcal{J}''} A''_l, \sup_{\mathcal{J}''}\{A''_l \bullet \overline{\theta}''_l\}) \\
\leq\ & \text{“By induction hypothesis (21)”} \\
& \tau + \sup_{\mathcal{J}}\{A_j \bullet \theta_j\} - \tau \\
=\ & \sup_{\mathcal{J}}\{A_j \bullet \theta_j\}
\end{aligned}
$$

This concludes the proof of assertion (iii).

Assertion (iv): First we prove the following property by structural induction over the path language.

$$\tau + \upsilon(S, \vee_{k\in\mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}) = \sup_{\mathcal{J}}\{A_j \bullet \theta_j\} \tag{23}$$

The base case $S \stackrel{\text{def}}{=} (x{:}[Q])^I$. According to Table 8 the following equalities hold.

$$
\begin{aligned}
& \upsilon(S, \vee_{k\in\mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}) \\
=\ & \text{“By Table 8”} \\
& \sup\{\{\tau', x'\} \mid L(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge \vee_{k\in\mathcal{J}'} A'_k)[\tau', x'/\tau, x]) \bullet \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}[\tau', x'/\tau, x]' + \tau' - \tau\} \\
=\ & \text{“By distributivity, Definition 4.2 and Theorem A.3”} \\
& \sup_{k\in\mathcal{J}'} \sup\{\{\tau', x'\} \mid L(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A'_k)[\tau', x'/\tau, x]) \bullet \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}[\tau', x'/\tau, x]' + \tau' - \tau\} \\
=\ & \text{“By Theorem A.4 ”} \\
& \sup_{k\in\mathcal{J}'} \sup_{l\in\mathcal{J}'} \sup\{\{\tau', x'\} \mid L(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A'_k)[\tau', x'/\tau, x]) \wedge A'_l[\tau', x'/\tau, x] \bullet \\
& \quad \theta'_l[\tau', x'/\tau, x] + \tau' - \tau\} \\
=\ & \text{“The predicates } A'_l \text{ are quant. linear and Definition 4.2”} \\
& \sup_{k\in\mathcal{J}'} \sup_{l\in\mathcal{J}'} \sup\{\{\tau', x'\} \mid L(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A'_k \wedge A'_l)[\tau', x'/\tau, x]) \bullet \theta'_l[\tau', x'/\tau, x] + \tau' - \tau\} \\
=\ & \text{“By monotonicity of the supremum ”} \\
& \sup_{k\in\mathcal{J}'} \sup\{\{\tau', x'\} \mid L(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A'_k)[\tau', x'/\tau, x]) \bullet \theta'_k[\tau', x'/\tau, x] + \tau' - \tau\} \\
=\ & \text{“By property (5)”} \\
& \sup_{k\in\mathcal{J}'} \sup\{\{\tau', x'\} \mid \vee L'(I \wedge \tau \leq \tau' \wedge Q \wedge (I \wedge A'_k)[\tau', x'/\tau, x]) \bullet \theta'_k[\tau', x'/\tau, x] + \tau' - \tau\} \\
=\ & \text{“By Theorem A.3 and Table 9”} \\
& \sup_{k\in\mathcal{J}'} \sup_{B\in\mathcal{E}_k} \sup\{\{\tau', x'\} \mid B \bullet \theta'_k[\tau', x'/\tau, x] + \tau' - \tau\} \\
=\ & \text{“By leaving out unsatisfiable predicates } B \text{ ”}
\end{aligned}
$$

$$
\begin{array}{ll}
 & \sup\limits_{k\in\mathcal{J}'}\ \sup\limits_{B\in\mathcal{E}_k\wedge Sat(B)}\ \sup\{\{\tau',x'\}\mid B\bullet\theta'_k[\tau',x'/\tau,x]+\tau'-\tau\}\\
= & \text{“By Theorem A.4”}\\
 & \sup\limits_{k\in\mathcal{J}'}\ \sup\limits_{B\in\mathcal{E}_k\wedge Sat(B)}\ \{\exists\tau',x'\ B\bullet\sup\{\{\tau',x'\}\mid B\bullet\theta'_k[\tau',x'/\tau,x]+\tau'-\tau\}\}\\
= & \text{“By property (9) and continuity of supremum on mini-linear forms”}\\
 & \sup\limits_{k\in\mathcal{J}'}\ \sup\limits_{B\in\mathcal{E}_k\wedge Sat(B)}\ \{\exists\tau',x'\ B\bullet\sup\{\{\tau',x'\}\mid \mathcal{O}(B)\bullet\theta'_k[\tau',x'/\tau,x]+\tau'-\tau\}\}\\
= & \text{“By [Sho77, Theorem 16] ”}\\
 & \sup\limits_{k\in\mathcal{J}'}\ \sup\limits_{B\in\mathcal{E}_k\wedge Sat(B)}\ \{\exists\tau',x'\ B\bullet\\
 & \qquad SIMP(Sup_{\mathcal{O}(B)}(\theta'_k[\tau',x'/\tau,x]+\tau',(Fv(B)\cup Fv(\theta'_k[\tau',x'/\tau,x]))\setminus\{\tau',x'\}))\}-\tau\\
= & \text{“By Table 9”}\\
 & \sup\limits_{\mathcal{J}}\{A_j\bullet\theta_j\}-\tau
\end{array}
$$

This proves property (23) for the base case. For the step case let $S\stackrel{\text{def}}{=}(\mathbf{var}\,v:T\bullet S_1)^I$ and $\{(A''_j,B''_j,\theta''_j):j\in\mathcal{J}\}$ denote $\Upsilon^a(S_1[w/v]^{I\wedge w\in T},M)$. The induction hypothesis can be formulated as follows.

$$\tau+\upsilon(S_1[w/x]^{I\wedge w\in T},\vee_{k\in\mathcal{J}'}A'_k,\sup_{\mathcal{J}'}\{A'_k\bullet\theta'_k\})\quad=\quad\sup_{\mathcal{J}}\{A''_j\bullet\theta''_j\}\tag{24}$$

Table 9 defines the sets $\mathcal{E}_j$ as follows.

$$\mathcal{E}_j\quad\stackrel{\text{def}}{=}\quad L'(A''_j)\,,\ j\in\mathcal{J}\tag{25}$$

Then the following equalities hold.

$$
\begin{array}{ll}
 & \upsilon(S,\vee_{k\in\mathcal{J}'}A'_k,\sup\limits_{\mathcal{J}'}\{A'_k\bullet\theta'_k\})\\
= & \text{“By Table 8”}\\
 & \sup\{\{w\}\mid L(\Theta(S_1[w/x]^{I\wedge w\in T},\vee_{k\in\mathcal{J}'}A'_k))\bullet\upsilon(S_1[w/x]^{I\wedge w\in T},\vee_{k\in\mathcal{J}'}A'_k,\sup\limits_{\mathcal{J}'}\{A'_k\bullet\theta'_k\})\}\\
= & \text{“By induction hypothesis (24)”}\\
 & \sup\{\{w\}\mid L(\Theta(S_1[w/x]^{I\wedge w\in T},\vee_{k\in\mathcal{J}'}A'_k))\bullet\sup\limits_{\mathcal{J}}\{A''_j\bullet\theta''_j\}-\tau\}\\
= & \text{“By Theorem 4.5 }(i)\text{ and Theorem 4.11 }(ii)\text{”}\\
 & \sup\{\{w\}\mid\ \vee_{j\in\mathcal{J}}L(A''_j)\bullet\sup\limits_{\mathcal{J}}\{A''_j\bullet\theta''_j\}-\tau\}\\
= & \text{“By property (5) and definition (25)”}\\
 & \sup\{\{w\}\mid\ \vee_{j\in\mathcal{J}}\vee_{B\in\mathcal{E}_j}B\bullet\sup\limits_{\mathcal{J}}\{A''_j\bullet\theta''_j\}-\tau\}\\
= & \text{“By Theorem A.3”}\\
 & \sup\limits_{j\in\mathcal{J}}\sup\limits_{B\in\mathcal{E}_j}\sup\{\{w\}\mid B\bullet\sup\limits_{\mathcal{J}}\{A''_j\bullet\theta''_j\}-\tau\}\\
= & \text{“By Theorem A.4 ”}\\
 & \sup\limits_{j\in\mathcal{J}}\sup\limits_{l\in\mathcal{J}}\sup\limits_{B\in\mathcal{E}_j}\sup\{\{w\}\mid B\wedge A''_l\bullet\theta''_l-\tau\}\\
= & \text{“By definition (25), Theorem 4.11 }(i)\text{, }A''_l\text{ quant. lin. predicates and monotonicity of sup.”}\\
 & \sup\limits_{j\in\mathcal{J}}\sup\limits_{B\in\mathcal{E}_j}\sup\{\{w\}\mid B\bullet\theta''_j-\tau\}\\
= & \text{“By leaving out unsatisfiable predicates }B\text{ ”}\\
 & \sup\limits_{j\in\mathcal{J}}\ \sup\limits_{B\in\mathcal{E}_j\wedge Sat(B)}\ \sup\{\{w\}\mid B\bullet\theta''_j-\tau\}
\end{array}
$$

$$
\begin{array}{ll}
= & \text{“By Theorem A.4”} \\
& \sup_{j \in \mathcal{J}} \sup_{B \in \mathcal{E}_j \wedge Sat(B)} \{\exists w\, B \bullet \sup\{\{w\} \mid B \bullet \theta''_j - \tau\}\} \\
= & \text{“By property (9) and continuity of supremum on mini-linear forms”} \\
& \sup_{j \in \mathcal{J}} \sup_{B \in \mathcal{E}_j \wedge Sat(B)} \{\exists w\, B \bullet \sup\{\{w\} \mid \mathcal{O}(B) \bullet \theta''_j - \tau\}\} \\
= & \text{“By Theorem 4.11 } (i) \text{ and [Sho77, Theorem 16] ”} \\
& \sup_{j \in \mathcal{J}} \sup_{B \in \mathcal{E}_j \wedge Sat(B)} \{\exists w\, B \bullet SIMP(Sup_{\mathcal{O}(B)}(\theta''_j, (Fv(B) \cup Fv(\theta''_j)) \setminus \{w\},))\} \\
= & \text{“By Table 9”} \\
& \sup_{\mathcal{J}}\{A_j \bullet \theta_j\}
\end{array}
$$

This proves the induction step (23) for variable declarations. It remains to verify the induction step for sequential composition $S \stackrel{\text{def}}{=} (S_1 \,;\, S_2)^I$. According to Table 9 the following equivalence holds.

$$\Upsilon^a(S, M) \quad = \quad \Upsilon^a(S_1^I, \Upsilon^a(S_2^I, M))$$

We abbreviate $\Upsilon^a(S_2^I, M)$ by $\{(A''_l, B''_l, \theta''_l) \,:\, l \in \mathcal{J}''\}$. Because of Theorem 4.11 (i), it is possible to find mini-linear forms $\overline{\theta}''_l$, $l \in \mathcal{J}''$ not dependent on τ such that the following equalities hold.

$$\theta''_l \quad = \quad \overline{\theta}''_l + \tau\,,\; l \in \mathcal{J}'' \tag{26}$$

Then, according to the induction hypothesis for S_1^I and S_2^I the following inequalities hold.

$$\tau + \upsilon(S_2^I, \vee_{k \in \mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}) \quad = \quad \sup_{\mathcal{J}''}\{A''_l \bullet \theta''_l\} \tag{27}$$

$$\tau + \upsilon(S_1^I, \vee_{l \in \mathcal{J}''} A''_l, \sup_{\mathcal{J}''}\{A''_l \bullet \overline{\theta}''_l\}) \quad = \quad \sup_{\mathcal{J}}\{A_j \bullet \theta_j\} \tag{28}$$

Furthermore, because of Theorem 4.5 (i) and Theorem 4.11 (i) and (ii), the following predicate equivalence holds.

$$L(\Theta(S_2^I, \vee_{k \in \mathcal{J}'} A'_k)) \quad \equiv \quad \vee_{l \in \mathcal{J}''} L(A''_l) \tag{29}$$

With the help of Table 8 this can be combined to derive the following chain of equalities.

$$
\begin{array}{ll}
& \tau + \upsilon(S, \vee_{k \in \mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\}) \\
= & \text{“By Table 8”} \\
& \tau + \upsilon(S_1^I, L(\Theta(S_2^I, \vee_{k \in \mathcal{J}'} A'_k)), \upsilon(S_2^I, \vee_{k \in \mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\})) \\
= & \text{“By equivalence (29)”} \\
& \tau + \upsilon(S_1^I, \vee_{l \in \mathcal{J}''} A''_l, \upsilon(S_2^I, \vee_{k \in \mathcal{J}'} A'_k, \sup_{\mathcal{J}'}\{A'_k \bullet \theta'_k\})) \\
= & \text{“By induction hypothesis (27)”} \\
& \tau + \upsilon(S_1^I, \vee_{l \in \mathcal{J}''} A''_l, \sup_{\mathcal{J}''}\{A''_l \bullet \theta''_l\} - \tau) \\
= & \text{“By property (26)”} \\
& \tau + \upsilon(S_1^I, \vee_{l \in \mathcal{J}''} A''_l, \sup_{\mathcal{J}''}\{A''_l \bullet \overline{\theta}''_l\}) \\
= & \text{“By induction hypothesis (28)”} \\
& \tau + \sup\{A_j \bullet \theta_j\} - \tau \\
= & \sup\{A_j \bullet \theta_j\}
\end{array}
$$

This concludes the proof of property (23). Assertion (*iv*) follows then from equality (23) and Theorem 4.10 (*iv*). □

References

[Alt96] P. Altenbernd. On the false path problem in hard real-time programs. In *Proc. 8th Euromicro Workshop on Real-Time Systems (WRTS)*, pages 102–107, 1996.

[Ble74] W. W. Bledsoe. The sup-inf method in presburger arithmetic. *Memo ATP-18, Math. Dept., U. of Texas at Austin*, Dec 1974.

[Ble75] W. W. Bledsoe. A new method for proving certain presburger formulas. *Advance papers 4th Int. Joint Conf. on Artif. Intell., Tibilisi, Georgia, U.S.S.R.*, pages 15–21, Sept 1975.

[BvW93] R.-J. R. Back and J. von Wright. Statement inversion and strongest postcondition. *Science of Computer Programming*, 20:223–251, 1993.

[BvW98] R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction.* Springer-Verlag, 1998.

[Car89] B. Carré. Program analysis and verification. In C. T. Sennett, editor, *High-Integrity Software*, chapter 8, pages 176–197. Plenum Press, 1989.

[CBW94] R. Chapman, A. Burns, and A. J. Wellings. Integrated program proof and worst-case timing analysis of SPARK Ada. In *ACM Workshop on language, compiler and tool support for real-time systems.* ACM Press, 1994.

[CBW96] R. Chapman, A. Burns, and A. Wellings. Combining static worst-case timing analysis and program proof. *Real-Time Systems*, 11:145–171, 1996.

[Cha95] R. Chapman. *Static Timing Analysis and Program Proof.* PhD thesis, Department of Computer Science, University of York, UK, 1995.

[Dij76] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[DS90] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics.* Springer-Verlag, 1990.

[EG97] A. Ermedahl and J. Gustafsson. Deriving annotations for tight calculation of execution time. In C. Lengauer, M. Griebel, and S. Gorlatch, editors, *Euro-Par'97: Parallel Processing*, volume 1300 of *Lecture Notes in Computer Science*, pages 1298–1307. Springer-Verlag, 1997.

[FHW99] C. J. Fidge, I. J. Hayes, and G. Watson. The deadline command. *IEE Proceedings—Software*, 146(2):104–111, April 1999. Special Issue on Real-Time Systems.

[GHF98] S. Grundon, I. J. Hayes, and C. J. Fidge. Timing constraint analysis. In C. McDonald, editor, *Computer Science '98: Proc. 21st Australasian Computer Science Conference*, pages 575–586. Springer-Verlag, 1998.

[GP99] E. L. Gunter and D. Peled. Path exploration tool. In W. R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS/ETAPS'99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 405–419. Springer-Verlag, 1999.

[Gri81] D. Gries. *The Science of Programming.* Springer-Verlag, 1981.

[HFL01] I. J. Hayes, C. J. Fidge, and K. Lermer. Semantic characterisation of dead control-flow paths. *IEE Proceedings—Software*, 148(6):175–186, 2001.

[HU97] I. J. Hayes and M. Utting. Coercing real-time refinement: A transmitter. In D. J. Duke and A. S. Evans, editors, *BCS-FACS Northern Formal Methods Workshop, 1996*, Electronic Workshops in Computing. Springer-Verlag, 1997. `http://www.ewic.org.uk/ewic/`.

[HU01] I. J. Hayes and M. Utting. A sequential real-time refinement calculus. *Acta Informatica*, 37:385–448, 2001.

[LAYH+99] Lo Ko, N. Al-Yaqoubi, C. Healy, E. Ratliff, R. Arnold, D. Whalley, and M. Harmon. Timing constraint specification and analysis. *Software—Practice and Experience*, 29(1):77–98, 1999.

[LBJ+95] Sung-Soo Lim, Young Hyun Bae, Gyu Tae Jang, Byung-Do Rhee, Sang Lyul Min, Chang Yun Park, Heonshik Shin, Kunsoo Park, Soo-Mook Moon, and Chong Sang Kim. An accurate worst case timing analysis for RISC processors. *IEEE Transactions on Software Engineering*, 21(7):593–604, July 1995.

[LFH02a] K. Lermer, C. J. Fidge, and I. J. Hayes. Extracting execution time constraints from real-time programs. Technical report, Software Verification Reseach Centre, The University of Queensland, Oct 2002.

[LFH02b] K. Lermer, C. J. Fidge, and I. J. Hayes. Formal semantics for program paths. Technical Report 02-05, Software Verification Reseach Centre, The University of Queensland, February 2002.

[LFH02c] K. Lermer, C. J. Fidge, and I. J. Hayes. A theory for execution time derivation in real-time programs. Technical Report 02-13, Software Verification Reseach Centre, The University of Queensland, April 2002.

[LG98] Y.A. Liu and G. Gomez. Automatic accurate time-bound analysis for high-level languages. In F. Mueller and A. Bestavros, editors, *Languages, Compilers, and Tools for Embedded Systems (LCTES'98)*, volume 1474 of *Lecture Notes in Computer Science*, pages 31–40. Springer-Verlag, 1998.

[LKM98] S. Lim, J. Kim, and S. L. Min. A worst case timing analysis technique for optimized programs. In *IEEE Real-Time Computing Systems and Applications*, 1998.

[LM95] Y.-T. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. *ACM SIGPLAN Notices*, 30(11):88–98, November 1995.

[LMW95] Yau-Tsun Li, S. Malik, and A. Wolfe. Efficient microarchitecture modeling and path analysis for real-time software. In *Proc. 16th IEEE Real-Time Systems Symposium*, pages 298–307. IEEE Computer Society Press, December 1995.

[LMW97] Y.-T. Li, S. Malik, and A. Wolfe. Cinderella: A retargetable environment for performance analysis of real-time software. In C. Lengauer, M. Griebel, and S. Gorlatch, editors, *Euro-Par'97: Parallel Processing*, volume 1300 of *Lecture Notes in Computer Science*, pages 1308–15. Springer-Verlag, 1997.

[LS98] T. Lundqvist and P. Stenström. Integrating path and timing analysis using instruction-level simulation techniques.

In F. Mueller and A. Bestavros, editors, *Languages, Compilers, and Tools for Embedded Systems (LCTES'98)*, volume 1474 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1998.

[MS98] K. Marriott and P. J. Stuckey. *Programming with constraints: an introduction.* The MIT Press, 1998.

[Par92] C. Y. Park. *Predicting deterministic execution times of real-time programs.* PhD thesis, University of Washington, Seattle, 1992.

[Par93] C. Y. Park. Predicting program execution times by analyzing static and dynamic program paths. *Real-Time Systems*, 5:31–62, 1993.

[PK89] P. Puschner and Ch. Koza. Calculating the maximum execution time of real-time programs. *Journal of Real-Time Systems*, 1(2):159–176, September 1989.

[PS90] C. Y. Park and A. C. Shaw. Experiments with a program timing tool based on source-level timing schema. In *Proc. IEEE Real-Time Systems Symposium*, pages 72–81, Florida, December 1990.

[PS97] P. P. Puschner and A. V. Schedl. Computing maximum task execution times: A graph-based approach. *Real-Time Systems*, 13(1):67–91, July 1997.

[SA00] F. Stappert and P. Altenbernd. Complete worst-case execution time analysis of straight-line hard real-time programs. *Journal of Systems Architecture: the EUROMICRO Journal*, 46(4):339–355, Febr 2000.

[Sha89] A. C. Shaw. Reasoning about time in higher-level language software. *IEEE Transactions on Software Engineering*, 15(7):875–889, July 1989.

[Sho77] R. E. Shostak. On the SUP-INF method for proving Presburger formulas. *Journal of the ACM*, 24(4):529–543, October 1977.