Weakest Precondition Semantics for Time and Concurrency

D.Scholefield H.S.M. Zedan
Formal Systems Research Group
Department of Computer Science
University of York
Heslington, York YO1 5DD.
{djs, zedan} @minster.york.ac.uk

August 15, 1992

Abstract

A weakest precondition semantics for a real-time concurrent language is defined. An example in verification is presented, and the use of predicate transformers as the basis of a refinement calculus is also discussed.

Keywords: formal semantics, weakest preconditions, real-time systems, concurrency

1 Introduction

Weakest precondition predicate transformers have been used to define the semantics of transformational sequential programming languages for almost two decades. Dijkstra first introduced them [Dij76] as a semantics for his Guarded Command Language, and extensions for time have been proposed [MSB91]. The popularity of weakest precondition semantics has been partly due to their natural correspondence to the axiomatic proof methods of Floyd and Hoare [Hoa69], and partly due to their success in providing a foundation for refinement methods [Mor87] [MRG88] [Mor90] [BW89] where refinement is defined as an entailment on weakest preconditions.

As yet, the definition of a weakest precondition semantics for a timed concurrent language (which might be found in a distributed real-time environment) has not been undertaken. This paper shows how the technique could be extended to give semantics for a concurrent real-time language.

2 The Language

We assume a real-time system to be composed of a set of **agents**, each of which has local state and may communicate asynchronously through time-stamped shared variables called **shunts**. Time is modelled by the positive integers, and is available to all shunts via a global

clock. Shunts may only be written to by a single agent, but can be read by many, thus an agent can be said to *own* a given shunt. Writing is destructive but reading is not. When a shunt is written to, the current time is written as well as the value; therefore all reads result in a pair of values, the first part of which is the time-stamp corresponding to the most recent write.

All agents will terminate within finite time, have a finite data space, and have a finite number of shunts which may be written to or read from. Agents may be given a release time, a relative deadline, and intermediate deadlines on communication and computation. Abstraction is available through shunt and variable hiding.

An agent is defined by

$$
\begin{array}{lll}
\mathcal{A} ::= & x := e & \text{(assign)} \\
| & x \leftarrow s & \text{(input)} \\
| & x \rightarrow s & \text{(output)} \\
| & (x:T)\mathcal{A} & \text{(private variable)} \\
| & \mathcal{A}/s:T & \text{(private shunt)} \\
| & \mathcal{A}||\mathcal{A} & \text{(concurrent)} \\
| & [t]\mathcal{A}:\mathcal{A} & \text{(deadline sequence)} \\
| & \bigsqcup_{i \in I_t} g_i \Rightarrow \mathcal{A}_i & \text{(timed guards)} \\
| & \mu_n^t \mathcal{A} & \text{(iteration)}
\end{array}
$$

Where t is any time (positive integer), x is a variable name, s is a shunt name, T is a type (a set of values), I is some finite indexing set, and n is a positive integer which declares the number of iterations.

In $[t]\mathcal{A}:\mathcal{A}$ the second agent will start execution at the time of the termination of the first. The time parameter t is passed to the run-time execution environment and the user can expect the scheduler to respect this deadline on the termination of the first agent. The time t on the guards is a guard evaluation deadline, and in addition we use the shorthand $g_t \Rightarrow \mathcal{A}$ for a single guarded agent. The agent $\mu_n^t \mathcal{A}$ iterates $\mathcal{A}$ a total of n times with a deadline of t on each iteration.

We impose the syntactic restriction that no agent may share its local state space with a concurrently executing agent, and only one concurrently executing agent may write to any given shunt. In addition, a system is described by a single agent which is assumed, without loss of generality, to be released at time=0. For all but the most simple of systems the agent will be a concurrent composition of simpler agents.

We may derive a number of useful agents:

$$\text{skip} =_{def} false_0 \Rightarrow (x:T)x := x$$

$$delay(n) =_{def} \mu_n^1 (x:T)x := x$$

3 Semantics

In his seminal paper [Dij76], Dijkstra defines a language semantics using transformer rules of the form:

$$wp(S, \phi)$$

where S is a program statement and ϕ is a

postcondition predicate on the program state. The transformer rule defines the weakest predicate that needs to hold on the program state prior to the execution of S in order to guarantee that the postcondition ϕ holds after S terminates.

Later extensions by Marzullo, Schneider, and Budhiraja [MSB91] provide transformers:

$$wp(< S >_\delta, \phi)$$

where the parameter δ models the passage of time, and asserts that S will execute for δ seconds.

None of the extensions proposed so far have dealt with concurrency. However, in order to give a semantics to concurrent and communicating processes it is not enough to define a relation between initial and final states; the *reactivity* of the system during execution must also be considered. We therefore define predicate transformers of the form:

$$wp^u_\delta(\mathcal{A}, \phi, \psi) : w$$

where $[\delta, u]$ is the execution interval given to the agent $\mathcal{A}$, ϕ is the usual postcondition, ψ is a predicate which is used as a record of the communications undertaken by the agent (it is defined by the weakest pre-condition rules and is not initially provided by the user), and w is a *frame* i.e. a set of shunts which are owned by $\mathcal{A}$. The predicate ψ and the frame w are extensions to help calculate the preconditions of concurrent agents. The predicate transformer returns a pair of predicates ϕ', ψ' (allowing the usual nested definition for sequential composition). The predicate ψ' records the outputs of $\mathcal{A}$ during the interval, and adds them to the outputs which may already be recorded in nested sequential agents by the predicate ψ.

In order to describe the timed behaviour of variables and shunts we make conservative extensions to first-order predicate logic. We add terms of the form '$x@t$' which represent the value of the variable or shunt x at some time t. The postcondition ϕ is then a predicate in the extended logic with variables referred to at times δ and u, and shunts referred to at times in $[\delta, u]$; the precondition ϕ' is a predicate on variables at time δ only, along with shunts at times in $[\delta, u]$.

In the following transformers we use predefined predicates on shunt and variable stability.

For a shunt s,

$$stable(s, n, m) =_{def} \bigwedge_{\sigma \in [n+1, m]} s@\sigma = s@(\sigma - 1)$$

In addition, the definitions for *stable* are extended to sets of shunts:

$$all_stable(S, n, m) =_{def} \bigwedge_{s \in S} stable(S, n, m)$$

The predicate *write* asserts that a given value is written to a shunt within an interval, and that the shunt remains stable at all other times within the interval:

$$write(s, x@n, n, m) =_{def} \bigvee_{\sigma \in [n+1, m]} stable(s, n, \sigma - 1)$$

$\wedge\ s@\sigma = (\sigma, x@n)\ \wedge\ stable(s, \sigma, m)$

The tuple accessors $tuple.ts$ and $tuple.v$ will be used for the left projection (time stamp) and right projection (value). In addition we use the notation $[\![\phi]\!]^u_\delta$ to represent the predicate ϕ with all variables (not shunts) accessed at time u to be replaced by the variable accessed at time δ. For example:

$$[\![x@\delta = (x@u) + (y@u) + s@u.v]\!]^u_\delta$$

$$\equiv\ x@\delta = (x@\delta) + (y@\delta) + s@u.v$$

(where s is a shunt).

We often need to refer to both halves of the resulting pair of an application of a transformer; the accessors $wp.f$ and $wp.c$ refer to the first and second halves of such a pair. We are now in a position to define the predicate transformers.

Local Variable

$$wp^u_\delta((x:T)\mathcal{A}, \phi, \psi) : w\ =_{def}$$

$$\exists x : T(wp^u_\delta(\mathcal{A}, \phi, \psi) : w.f)\ ,$$

$$\exists x : T(wp^u_\delta(\mathcal{A}, \phi, \psi) : w.c)$$

where if x appears in ϕ, ϕ', ψ or ψ' then it is renamed in the declaration to a unique name, and all references in $\mathcal{A}$ renamed consistently.

Assignment

$$wp^u_\delta(x := e, \phi, \psi) : w\ =_{def}$$

$$[\![\phi[e@\delta/x@u]]\!]^u_\delta\ \wedge\ \delta < u\ ,$$

$$[\![\psi[e@\delta/x@u]]\!]^u_\delta\ \wedge$$

$$all_stable(w, \delta, u)\ \wedge\ \delta < u$$

Input

$$wp^u_\delta(x \leftarrow s, \phi, \psi) : w\ =_{def}$$

$$[\![\textstyle\bigwedge_{\sigma\in[\delta+1,u]} \phi[s@\sigma/x@u]]\!]^u_\delta\ \wedge\ \delta < u\ ,$$

$$[\![\textstyle\bigvee_{\sigma\in[\delta+1,u]} \psi[s@\sigma/x@u]]\!]^u_\delta\ \wedge\ \delta < u\ \wedge$$

$$all_stable(w, \delta, u)$$

Output

$$wp^u_\delta(x \rightarrow s, \phi, \psi) : w\ =_{def}$$

$$[\![\phi]\!]^u_\delta\ \wedge\ \delta < u\ ,$$

$$write(s, x@\delta, \delta, u)\ \wedge$$

$$[\![\psi]\!]^u_\delta\ \wedge\ all_stable(w - s, \delta, u)$$

Deadline Sequence

$$wp^u_\delta([t]\mathcal{A} : \mathcal{B}, \phi, \psi) : w\ =_{def}$$

$$(\textstyle\bigvee_{\sigma\in[\delta,n]} wp^\sigma_\delta(\mathcal{A}, true, true) : w.f)\ \wedge$$

$$\textstyle\bigwedge_{\sigma\in[\delta,n]} (wp^\sigma_\delta(\mathcal{A}, true, true) : w.f\ \Rightarrow$$

$$wp^\sigma_\delta(\mathcal{A}, wp^u_\sigma(\mathcal{B}, \phi, \psi) : w) : w.f)\ ,$$

$$\textstyle\bigvee_{\sigma\in[\delta,n]} (wp^\sigma_\delta(\mathcal{A}, true, true) : w.f\ \wedge$$

$$wp^\sigma_\delta(\mathcal{A}, wp^u_\sigma(\mathcal{B}, \phi, \psi) : w) : w.c)$$

where $n = minimum(\delta + t, u)$

Note that the initial conjunct of the functional transformation requires that there is an interval within $[\delta, u]$ which is long enough for the agent $\mathcal{A}$ to execute. The constraint $wp^m_n(\mathcal{A}, true, true) : w.f$ is true for exactly those intervals $[n, m]$ of sufficient length for the time required for inputs, outputs, assignments, timeouts, and guards, in $\mathcal{A}$. The dead-

line sequence then requires that for each interval greater or equal to this minimum interval, if the first agent executes over this interval and the second over the remaining time, then the postcondition will be met.

Concurrent

$$wp^u_\delta(\mathcal{A}||\mathcal{B}, \phi, \psi) : w \quad =_{def}$$

$$\exists X^*(wp^u_\delta(\mathcal{A}, \phi_\mathcal{A}, \psi_\mathcal{A}) : w_\mathcal{A}.f \wedge$$

$$wp^u_\delta(\mathcal{B}, \phi_\mathcal{B}, \psi_\mathcal{B}) : w_\mathcal{B}.f) \ ,$$

$$\exists X^*(wp^u_\delta(\mathcal{A}, \phi_\mathcal{A}, \psi_\mathcal{A}) : w_\mathcal{A}.c \wedge$$

$$wp^u_\delta(\mathcal{B}, \phi_\mathcal{B}, \psi_\mathcal{B}) : w_\mathcal{B}.c)$$

Where:

$$\exists X^*(\phi_\mathcal{A} \ \wedge \ \phi_\mathcal{B}) \ \equiv \ \phi$$

$$\exists X^*(\psi_\mathcal{A} \ \wedge \ \psi_\mathcal{B}) \ \equiv \ \psi$$

$$w_\mathcal{A} \cap w_\mathcal{B} \ = \ \emptyset \quad \text{and} \quad w_\mathcal{A} \cup w_\mathcal{B} \ = \ w$$

and where no variable in the scope of $\mathcal{A}$ appears in $\phi_\mathcal{B}$, no variable in the scope of $\mathcal{B}$ appears in $\phi_\mathcal{A}$, and no shunt written to by $\mathcal{A}$ is in $w_\mathcal{B}$ and no shunt written to by $\mathcal{B}$ is in $w_\mathcal{A}$.

The set of quantification variables X^* is introduced by the user to enable the replacement of conflicting timed variables in the partitions. For example, given the agent $x := x + 1 \ || \ y := y + 1$ and the postcondition $y@u > x@u$, the predicate $\exists z((z > x@u) \wedge (z = y@u))$ provides a suitable partition.

Any partition of a postcondition which meets the requirements given above will give rise to the same precondition, and therefore any such partition will define the unique semantics of a concurrent agent. We can justify this assertion by the following reasoning.

We first divide the postcondition ϕ into any indexed conjunction $\exists X^*(\bigwedge_{i \in I} \phi_i)$ where for each formula ϕ_i, timed variables are drawn from $\mathcal{A}$ or $\mathcal{B}$ (but never both) along with variables from X^*. For those conjuncts in which only variables from X^* appear, the formula may be placed in $\phi_\mathcal{A}$ or $\phi_\mathcal{B}$ and will remain untransformed (and therefore *which* partition this conjunction appears in does not affect the final precondition). Those formula ϕ_i for which variables from $\mathcal{A}$ are present must go into $\phi_\mathcal{A}$, and similarly for $\phi_\mathcal{B}$ (i.e. there is no choice as to which formulae ϕ_i appears in). The same reasoning applies to the partitioning of the communication formula.

Guards

$$wp^u_\delta(\bigsqcup_{i \in I} {}_t g_i \rightarrow \mathcal{A}_i, \phi : \psi) : w \ =_{def}$$

$$t + \delta \leq u \ \wedge \ ((\bigwedge_{i \in I} \neg g_i@\delta \ \wedge \ [\![\phi]\!]^u_\delta) \vee$$

$$(\bigvee_{i \in I} g_i@\delta \ \wedge \ \bigwedge_{i \in I} (g_i@\delta \Rightarrow$$

$$[\![wp^u_{t+\delta}(\mathcal{A}_i, \phi, \psi) : w.f]\!]^{\delta+t}_\delta))) \ ,$$

$$t + \delta \leq u \ \wedge \ ((\bigwedge_{i \in I} \neg g_i@\delta \ \wedge$$

$$[\![\psi]\!]^u_\delta \ \wedge \ all_stable(w, \delta, u) \ ) \vee$$

$$(\bigvee_{i \in I} g_i@\delta \ \wedge \ all_stable(w, \delta, \delta + t) \wedge$$

$$(\bigvee_{i \in I} g_i@\delta \ \wedge \ [\![wp^u_{\delta+t}(\mathcal{A}_i, \phi, \psi) : w.c]\!]^{\delta+t}_\delta)))$$

The guards are all evaluated with the values of the variables at time δ. Multiple true guards results in nondeterminism. If there are no true

guards then the agent will delay for the period of guard evaluation and then terminate correctly.

Shunt Restriction

$$wp^u_\delta(\mathcal{A}/s:T,\phi,\psi):w \;=_{def}$$

$$wp^u_\delta(\mathcal{A}[s'/s]/s',\phi,\psi):w$$

$$(\text{if } s \in shunts(\phi) \text{ or } s \in shunts(\psi))$$

otherwise:

$$\forall s:T(discharge(wp^u_\delta(\mathcal{A},\phi,\psi):w\cup\{s\}))$$

$$\exists s:T(wp^u_\delta(\mathcal{A},\phi,\psi):w\cup\{s\})$$

where $discharge(\phi,\psi)$ is the least (ordered by entailment) predicate χ such that $\psi\wedge\chi \equiv\rangle \phi$.

Iteration

$$wp^u_\delta(\mu^t_n\mathcal{A},\phi,\psi):w \;=_{def}$$

$$wp^u_\delta(\mathcal{A}^n(skip,t),\phi,\psi):w$$

where:

$$\mathcal{A}^0(skip,t) \;=_{def}\; skip$$

$$\mathcal{A}^{n+1}(skip,t) \;=_{def}\; [t]\mathcal{A}:(\mathcal{A}^n(skip,t))$$

4 Derivation and Verification

In this section we show how a behavioural property $\mathcal{P}$, expressed in our extended logic, can be proven to hold of a given agent $\mathcal{A}$ over a given execution interval $[n,m]$. The property $\mathcal{P}$ describes the required behaviour as a predicate over variables at the start and end of the execution interval, and shunts at any time within the interval; $\mathcal{P}$ holds of the agent $\mathcal{A}$ if

$$wp^u_\delta(\mathcal{A},\mathcal{P},true):w.c$$

$$\equiv\rangle\;\; wp^u_\delta(\mathcal{A},\mathcal{P},true):w.f$$

that is, if the communication record of the agent ensures that the weakest precondition is met.

In addition, we may add an assumption Ass about the environment, which is a predicate over any shunts during the execution of the agent. The assumption can be used to help discharge the weakest precondition by including it in the calculation:

$$\text{Ass} \;\wedge\; wp^u_\delta(\mathcal{A},\mathcal{P},true):w.c$$

$$\equiv\rangle\;\; wp^u_\delta(\mathcal{A},\mathcal{P},true):w.f$$

Consider a simple real-time agent in which during the first time unit two variables are read from shunts s_1 and s_2. The time-stamps in the variables are compared, and the value with the freshest stamp is written to the shunt s_f within a further four time units. The agent is denoted (for any type T) by:

$$(x_1 \;:\; [Time\times T])(x_2 \;:\; [Time\times T])\;\; \mathcal{A}$$

where $\mathcal{A} \;=_{def}\; [1]\mathcal{B}:\mathcal{C}$

where $\mathcal{B} \;=_{def}\; \begin{pmatrix} x_1 \leftarrow s_1 \\ || \\ x_2 \leftarrow s_2 \end{pmatrix}$

where $\mathcal{C} \;=_{def}$

$$\begin{pmatrix} x_1.ts \geq x_2.ts \Rightarrow [2](x_1.v \rightarrow s_f) \\ \sqcup_2 \\ x_1.ts < x_2.ts \Rightarrow [2](x_2.v \rightarrow s_f) \end{pmatrix}$$

The agent is going to be executed within a time

interval of six units ($[0,5]$) and we also assume that the environment guarantees that the input shunts are stable:

$$\text{Ass} \;=_{def}\; all_stable(\{s_1, s_2\}, 0, 5)$$

We wish to verify that the most recent value is indeed written; the property which expresses this is denoted:

$$\text{Prop} \;=_{def}$$

$$s_1@0.ts \geq s_2@0.ts \;\Rightarrow\; s_f@5.v \;=\; s_1@0.v \wedge$$

$$s_1@0.ts < s_2@0.ts \;\Rightarrow\; s_f@5.v \;=\; s_2@0.v$$

The environment of the agent is simply the set of shunts which might be written to during the execution i.e. $\{s_f\}$. The verification theorem is therefore:

Theorem :

$$\text{Ass} \;\wedge\; wp_0^5(\mathcal{A}, \text{Prop}, true) : \{s_f\}.c$$

$$\equiv\rangle\; wp_0^5(\mathcal{A}, \text{Prop}, true) : \{s_f\}.f$$

Proof :

We calculate the functional precondition and the communication record separately. We start with the functional precondition. The local variable declaration gives rise to an existential quantification on the variables x_1 and x_2, and we postpone this transformation until the inner transformation has been calculated.

The outermost agent form is therefore the deadline sequence of $[1]\mathcal{B} : \mathcal{C}$, and this gives rise to the transformation:

$$wp_0^5(\mathcal{A}, Prop, true) : \{s_f\}.f \;\equiv$$

$$\bigvee_{\sigma \in [0,n]} wp_0^\sigma(\mathcal{B}, true, true) : \{s_f\}.f \quad \text{(i)}$$

$$\wedge \bigwedge_{\sigma \in [0,n]} wp_0^\sigma(\mathcal{B}, true, true) : \{s_f\}.f \quad \equiv\rangle$$

$$wp_0^\sigma(\mathcal{B}, wp_\sigma^5(\mathcal{C}, Prop, true)\{s_f\}) : \{s_f\}.f \quad \text{(ii)}$$

We need a lemma in order to simplify this transformation:

Lemma 1:

$$\sigma > 0 \;\equiv\; wp_0^\sigma(\mathcal{B}, true, true) : \{s_f\}.f \quad \square$$

Now by Lemma 1 and $n = minimum(0 + 1, 5) = 1$:

$$(i) \;\equiv\; true \text{ (by } \sigma = 1)$$

Also, in (ii) we only need to consider the case where $\sigma = 1$ (because this is the only intermediate time allowed by the deadline [1]). This reduces the indexed conjunction in the deadline sequence transformation to the single indexing set $\{1\}$, simplifying the transformation to:

$$wp_0^1(\mathcal{B}, wp_1^5(\mathcal{C}, Prop, true) : \{s_f\}) : \{s_f\}.f$$

We take the inner transformer first (i.e. the agent $\mathcal{C}$ – the guard) which transforms to:

$$2 + 1 \leq 5 \;\wedge\; ((\neg(x_1@1.ts \geq x_2@1.ts)$$

$$\wedge\; \neg(x_1@1.ts < x_2@1.ts))$$

$$\vee\; ((x_1@1.ts \geq x_2@1.ts \vee x_1@1.ts < x_2@1.ts) \wedge$$

$$x_1@1.ts \geq x_2@1.ts$$

$$\Rightarrow [\![wp_3^5(\mathcal{D}_1, Prop, true) : \{s_f\}.f]\!]_1^3$$

$$\wedge$$

$$x_1@1.ts < x_2@1.ts$$

$$\Rightarrow [\![wp_3^5(\mathcal{D}_2, Prop, true) : \{s_f\}.f]\!]_1^3))$$

(def $\bigsqcup_t$) where:

$$\mathcal{D}_1 =_{def} [2]x_1.v \rightarrow s_1 \quad \mathcal{D}_2 =_{def} [2]x_2.v \rightarrow s_2$$

but one of the two guards must be true so we have:

$$x_1@1.ts \geq x_2@1.ts$$

$$\Rightarrow [\![wp_3^5(\mathcal{D}_1, Prop, true) : \{s_f\}.f]\!]_1^3 \ \wedge$$

$$x_1@1.ts < x_2@1.ts$$

$$\Rightarrow [\![wp_3^5(\mathcal{D}_2, Prop, true) : \{s_f\}.f]\!]_1^3$$

We expand the agents $\mathcal{D}_1$ and $\mathcal{D}_2$, and both give:

$$3 < 5 \ \wedge \ [\![Prop]\!]_3^5 \ \equiv \ [\![Prop]\!]_3^5$$

When placed back into the guard semantics this gives us:

$$x_1@1.ts \geq x_2@1.ts \quad \Rightarrow [\![Prop]\!]_1^5) \ \wedge$$

$$x_1@1.ts < x_2@1.ts \quad \Rightarrow [\![Prop]\!]_1^5)$$

$$\equiv \quad [\![Prop]\!]_1^5$$

which is the transformation of the agent $\mathcal{C}$. This is placed back into the deadline sequence to give:

$$wp_0^1(\mathcal{B}, [\![Prop]\!]_1^5, \psi) : \{s_f\}.f$$

(for some ψ)

$$\equiv \quad [\![Prop]\!]_0^5 \quad \equiv \quad Prop$$

The outer existential quantifiers arising from the local variables x_1 and x_2 can be dropped because neither of these variables appear in the final precondition. It is not surprising that the functional weakest precondition has not transformed $Prop$ because it is expressed purely in terms of shunts. It is the communication predicate which is more important in this case. We calculate that next.

We start with the inner predicate transformer for the agent $\mathcal{C}$, which arises from the deadline sequence $[1]\mathcal{B} : \mathcal{C}$

$$wp_1^5(\mathcal{C}, Prop, true) : \{s_f\}.c$$

$$\equiv \quad all_stable(\{s_f\}, 1, 3) \wedge$$

$$((x_1@1.ts \geq x_2@1.ts \wedge$$

$$[\![wp_3^5(\mathcal{D}_1, Prop, true) : \{s_f\}.c]\!]_1^3)$$

$$\vee (x_1@1.ts < x_2@1.ts \wedge$$

$$[\![wp_3^5(\mathcal{D}_2, Prop, true) : \{s_f\}.c]\!]_1^3))$$

The expansion of the transformer for $\mathcal{D}_1$ gives us:

$$((s_f@4 = (4, x_1@3.v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, x_1@3.v)) \wedge$$

$$[\![true]\!]_3^5 \ \ (\text{def} \rightarrow)$$

which gives us:

$$((s_f@4 = (4, x_1@3.v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, x_1@3.v))$$

(similarly for second guard). Back into guards gives us:

$$all_stable(\{s_f\}, 1, 3) \wedge$$

$$((x_1@1.ts \geq x_2@1.ts \wedge$$

$$((s_f@4 = (4, x_1@1.v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, x_1@1.v))) \quad \vee$$

$$(x_1@1.ts < x_2@1.ts \wedge$$

$$((s_f@4 = (4, x_2@1.v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, x_2@1.v))))$$

Back in the deadline sequence we need to divide the communication predicate into ψ_1 and ψ_2, this is achieved by quantifying over the variables v and w thus:

$$\psi_1 \ =_{def} \quad all_stable(\{s_f\}, 1, 3) \wedge$$

$$x_1@1.ts \geq v.ts \wedge$$

$$((((s_f@4 = (4, x_1@1.v) \wedge s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, x_1@1.v))) \quad \vee$$

$$(x_1@1.ts < v.ts \wedge$$

$$((s_f@4 = (4, x_2@1.v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, x_2@1.v))))$$

$$\wedge \ \ w = x_1@1$$

$$\psi_2 \ =_{def} \quad all_stable(\{s_f\}, 1, 3) \wedge$$

$$((w.ts \geq x_2@1.ts \wedge$$

$$((s_f@4 = (4, x_1@1.v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, x_1@1.v))) \quad \vee$$

$$(w.ts < x_2@1.ts \wedge$$

$$((s_f@4 = (4, x_2@1.v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, x_2@1.v))))$$

$$\wedge \ \ v = x_2@1$$

we may now calculate:

$$wp_0^1(x_1 \leftarrow s_1, \phi, \psi_1) : \{s_f\}$$

$$\equiv$$

$$((s_1@1.ts \geq v \wedge$$

$$((s_f@4 = (4, s_1@1.v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, s_1@1.v))) \quad \vee$$

$$(x_1@1.ts < v.ts \wedge$$

$$((s_f@4 = (4, s_2@1.v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, s_2@1.v))))$$

$$\wedge \ \ w = s_1@1 \ \wedge \ all_stable(\{s_f\}, 0, 1)$$

and similarly for the agent $x_2 \leftarrow s_2$. Back in conjunction this gives us the communication predicate:

$$all_stable(\{s_f\}, 0, 3) \wedge$$

$$((s_1@1.ts \geq s_2@1.ts \wedge$$

$$((s_f@4 = (4, s_1@1, v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, s_1@1.v))) \quad \vee$$

$$(s_1@1.ts < s_2@1.ts \wedge$$

$$((s_f@4 = (4, s_2@1, v) \ \wedge \ s_f@5 = s_f@4) \vee$$

$$(s_f@4 = s_f@3 \ \wedge \ s_f@5 = (5, s_2@1.v))))$$

The outer existential quantification due to the local variable declarations can be dropped because there are no local variables. Now clearly the assumption, together with this predicate, entails the predicate $Prop$ $\Box$.

5 Discussion

Although we have presented a weakest precondition semantics for real-time systems in which

properties may be verified, we do not expect the transformers to be used directly in this manner; the rules are too complex for realistic sized problems. Instead we propose that the language is extended with an abstract specification statement, and the semantics used as a basis for proving the soundness of a refinement calculus. In such a calculus the user would not use the transformers directly, and could appeal to higher level laws which greatly simplify verification. A summary of such a calculus is given in [SZ92a] [SZ92b] [Sch92].

The transformers presented here exhibit a number of pleasant properties, including *monotonicity*, and the distribution of conjunction through transformers. In addition, other language extensions such as timeouts (through synchronous communication), funnels (multiple agents writing to a single shunt), and imprecise agents (taking advantage of scheduler slack time) are defined in [Sch92].

6 References

[BW89] R. J. R. Back, J Wright, **"Refinement Calculus, part I: Sequential Programs"**, REX, Workshop For Refinement Of Distributed Systems, Nijmegen. 1989.

[Dij76] E. Dijkstra, **"A Discipline Of Programming"**, Prentice-Hall. 1976.

[Hoa69] C.A.R. Hoare, **"An Axiomatic Basis For Computer Programming"**, Communications Of The ACM, 12, p576-580. 1969.

[Mor90] C. Morgan, **"Programming From Specifications"**. PHI. 1990.

[Mor87] J. M. Morris, **"A Theoretical Basis For Stepwise Refinement And The Programming Calculus"**, Science Of Computer Programming, 9, p287-306. 1987.

[MRG88] C. Morgan, K. Robinson, P. Gardiner, **"On The Refinement Calculus"**, PRG-70, Oct 1988.

[MSB91] K. Marzullo, F. B. Schneider, N. Budhiraja, **"Derivation of Sequential, Real-Time, Process-Control Programs"**, in "Foundations of Real-Time Computing", A. M. van Tilborg, G. M. Koob (eds), Kluwer Academic Publishers. 1991.

[Sch92] D. Scholefield, **"A Refinement Calculus for Real-Time Systems"**, University of York, Dept. Computer Science Technical Report. *to appear.*

[SZ92a] D. Scholefield, H.S.M. Zedan, **"TAM: A Formal Framework for the Development of Distributed Real-Time Systems"**, in **"Formal Techniques in Real-Time and Fault-Tolerant Systems"**, Nijmegen LNCS 571. 1992.

[SZ92b] D. Scholefield, H.S.M. Zedan, **"Real-Time Refinement in TAM"**, Euromicro '92 Workshop on Real-Time Systems. Athens Greece. Proceedings IEEE. 1992.