Formal Aspects of Computing (1996) 8: 408–427

Formal Aspects of Computing

Real-Time Refinement in Manna and Pnueli's Temporal Logic

David Scholefield

Real-Time Research Group, Department of Computer Science, University of York, York, UK

Keywords: Refinement, Temporal logic, Real-time; Specification

Abstract. A refinement calculus for the development of real-time systems is presented. The calculus is based upon a wide-spectrum language called TAM (the Temporal Agent Model), within which both functional and timing properties can be expressed in either abstract or concrete terms. A specification oriented semantics is given for the language. Program development is considered as a refinement process i.e. the *calculation* of a structured program from an unstructured specification. An example program is developed.

1. Introduction

Traditionally, temporal logics have been used for the verification of concurrent programs [Bar85,MaP83,Gal87], and more recently real-time programs [Ost89, Hoo91]. To accommodate reasoning about real-time, the qualitative operators of temporal logic have been extended to include some notion of quantitative time. In Ostroff's verification calculus [Ost89], a local variable T is introduced which holds the current time. In Hooman's verification calculus [Hoo91], Koymans' Metric Temporal Logic [KdR85] is used, in which the model operators are extended to include real-time interval constraints. More recently, the Duration Calculus [ZHR91] provides for a formal analysis of *hybrid* real-time systems i.e. systems which are concerned with the interface between analogue control theory and the digital computer.

However, in all of these cases the verification calculus depends upon *a posteriori* verification of the program against the specification. Such a method in prone

Correspondence and offprint requests to: David Scholefield, Real-Time Research Group, Department of Computer Science, University of York, York YO1 5DD, UK. Email djsminster.york.ac.uk

to repeated reformulation of the program due to the verification process failing [Sch92]. One solution to this problem is to use a top-down *refinement calculus*, where each refinement towards a program is guaranteed correct with respect to the specification due to the soundness of the laws of the calculus.

This paper introduces a refinement calculus for real-time programs in which Manna and Pnueli's temporal logic [MaP83] is extended to a real-time logic by axiomatising the behaviour of a local variable which represents the current time. TAM, a wide-spectrum language, is defined which enables the program developer to express both program specifications and implementations. A refinement relation is defined along with a refinement calculus which is sound with respect to the refinement relation. The refinement calculus guides the step-wise refinement of a specification by the gradual replacement of parts of the specification with executable code. An example real-time program is refined. The refinement relation is proven compositional (monotonic), and the proofs of soundness of the refinement laws are given.

2. The TAM Language

2.1. TAM Syntax

We define a real-time system as a collection of concurrently executing time constrained computation agents which communicate asynchronously via time-stamped shared data areas called shunts. The time-stamps refer to the time of the last write to that shunt. Shunts may only have one writer but any number of readers. Time is global i.e. a single clock is accessible by every agent and shunt. The time domain is discrete, linear, and modelled by the positive integers. There is a unique 'first time' instant from which we assume all systems will measure their execution release times for all agents. We denote this time by 0.

The syntax of TAM is defined recursively in terms of agents $\mathscr{A}$ by:

$$\mathscr{A} \ ::= \ [w \bullet \mathscr{A}] \mid \vartheta \rightsquigarrow s \mid w : \Phi \mid \mathscr{A} | \mathscr{A} \mid [S]\mathscr{A} \mid \mathscr{A}; \mathscr{A} \mid \bigsqcup_{i \in I} g_i \Rightarrow \mathscr{A}_i \mid \mu_n \mathscr{A}$$

Where w is a set of shunt names, s is a shunt name, I is some finite indexing set, g_i are boolean expressions on shunts, ϑ is an expression on shunts, and S is a set of times.

$[w \bullet \mathscr{A}]$ defines the environment for the agent $\mathscr{A}$. The set of shunts in w are exactly those shunts which may be written to by $\mathscr{A}$ (and may not be written to by any other concurrently executing agent). The shunts in w are also associated with a type (a set of values). We use the convention of calling those shunts which may be written to by an agent the *output* shunts of that agent. We also call those shunts which may not be written to by an agent the *input* shunts of an agent.

$\vartheta \rightsquigarrow s$ performs an asynchronous output to the shunt s. The current time is written to the shunt, along with the value of the expression ϑ (which may refer to shunts). We use the notation $s.v$ to refer to the value found in the shunt s, and $s.ts$ for the time-stamp. The user does not explicitly have to specify the time-stamp to be written to the shunt, and may assume that the run-time environment will perform this task.

$w : \Phi$ in an agent which specifies required behaviour on shunts. This statement

form is discussed in detail below. The shunts listed in w are not given a type (they are already typed by the surrounding environment).

$\mathscr{A}|\mathscr{A}$ executes the two agents concurrently. The two agents are released at the same time, and the concurrent composition terminates some arbitrary time after both sub-agents terminate.

$[S]\mathscr{A}$ imposes a duration equal to one of the values in the set S on $\mathscr{A}$. This agent form is therefore capable of describing a *deadline* on agent termination.

$\mathscr{A};\mathscr{A}$ defines the sequential composition of two agents. There is no delay between the termination of the first agent and the execution of the second.

$\bigsqcup_{i\in I} g_i \Rightarrow \mathscr{A}_i$ evaluates all of the guards g_i (which are boolean expressions on shunts), and executes one of the agents corresponding to a true guard. If no guard evaluates to true, then the choice agent simply terminates. The indexing set I is always finite.

$\mu_n\mathscr{A}$ executes the agent $\mathscr{A}$ in sequence n times.

In addition we use the shorthand $\prod_{i\in I}\mathscr{A}_i$ to denote indexed concurrent composition (I is finite), and similarly $\sum_{i\in I}\mathscr{A}_i$ to denote indexed sequential composition (again, I is finite).

The syntax of the specification agent in TAM is defined as:

$$w : \Phi$$

where w is a set of shunt names which may be changed during the behaviour defined by the specification, and Φ is a timed logic formula which describes that behaviour. Thus, given a specification agent with an environment (i.e. $[w_0 \bullet w : \Phi]$), then w can be seen as a further partitioning of w_0 into a stable set $w_0 - w$ and a changeable set w.

The logic used for the specification is Manna and Pnueli's temporal logic [MaP83]. We assume a discrete linear time domain (positive integers i.e. $Time = Z^+$), and use a special local variable τ to denote the current time. We axiomatize the behaviour of τ so that the value increments with each transition to a new state. This axiomatisation is given in the next section.

In a specification $w : \Phi$, the formula Φ may also contain two unique free variables $t, t' \in Time$ which represent the time at which the behaviour described by Φ starts (t), and the time at which the behaviour terminates (t'). Other free variables in Φ represent shunts that appear in the program.

2.2. Example Agents

The 'watchdog timer' is a process which monitors a number of input channels for activity, and any channel which remains inactive for a given number of time units causes an alarm to be sounded. We model n input channels by shunts which will be labelled $d_1, d_2, ..., d_n$, and model the alarm with an output shunt called *Alarm*. There is an maximum inactivity period for each input shunt which is denoted I_i, and a maximum interval before the alarm should 'sound' (be written to) denoted T_{alarm} (the same value for all input channels). All shunts are of type boolean.

There are two requirements on the watchdog timer behaviour: the first is a liveness requirement which asserts when the alarm should be sounded:

$$\begin{aligned} R.1 =_{def} \forall m \in [t, t'] \\ (\forall i \in [1, n]((m + I_i + T_{alarm} \leq t' \wedge stable(d_i, m, m + I_i)) \Rightarrow \\ \Diamond(m + I_i \leq \tau \leq m + I_i + T_{alarm} \\ \wedge Alarm = (\tau, true)))) \end{aligned}$$

(Note that the predicate *stable* asserts that a shunt does not change value for a given interval – it is formally defined in the next section.) The second requirement is a safety requirement that only allows an alarm to be sounded in those cases when there has been an inactive input shunt:

$$\begin{aligned} R.2 =_{def} \forall m \in [t, t'] \\ (\Diamond(\tau = m) \wedge Alarm.ts = m \Rightarrow \\ \exists i \in [1, n](m - I_i \geq t \wedge \\ \exists l, g \in [t, m](l - I_i = g \wedge l \geq m = T_{alarm} \\ \wedge stable(d_i, g, l)))) \end{aligned}$$

The watchdog timer behaviour can now be specified by the agent:

$$[\{Alarm : bool\} \bullet \{Alarm\} : R.1 \wedge R.2]$$

Note that the environment and the specification frame have the same single shunt *Alarm*. In most instances, the initial specification of a system will have the same shunts in both the environment and the frame; there seems little justification for having shunts in the initial environment if they can never change value.

Consider a system which is required to search an input list of a given fixed size for a given element within d time units. One way in which this might be achieved is by dividing the list into two halves and searching each half concurrently (thus potentially reducing the execution time given appropriate hardware support). We use this example to illustrate how specification statements can be mixed with executable code.

We assume an input shunt S_l which contains the list to search, and an input shunt S_e which contains the element to be searched for (assuming types *Element* and *List*(*Element*)). In addition we assume two intermediate boolean shunts F_1 and F_2 which will indicate whether or not one of the searching agents has found a match, and a final boolean output shunt F which will provide the result of the overall search. We also assume two functions *FindInFirst*() and *FindInSecond*() which map an element and an element list to a boolean, and which searches the first and second halves respectively.

There are two concurrent searching agents:

$$Search1 =_{def} FindInFirst(S_l.v, S_e.v) \rightsquigarrow F_1$$

$$Search2 =_{def} FindInSecond(S_l.v, S_e.v) \rightsquigarrow F_2$$

There is also an agent which collates the results from F_1 and F_2 and outputs the result into F:

$$\begin{aligned} Collate =_{def} \{F\} : \Diamond(\tau = t \wedge (F_1.v \vee F_2.v \Rightarrow \Diamond(\exists l \in [t, t'](stable(F, t, l - 1) \wedge \\ \Diamond(\tau = m \wedge F = (l, true)))) \wedge stable(F, l, t')) \end{aligned}$$

The three agents may now be composed into a single system along with the appropriate deadline:

$$[\{F_1, F_2, F : bool\} \bullet [d]((Search1 | Search2); Collate)]$$

Note that the collating agent is quite complex, and if we had written it as an executable agent, we could have expressed it as:

$$Collate =_{def} F_1.v \Rightarrow (true \rightsquigarrow F) \sqcup F_2.v \Rightarrow (true \rightsquigarrow F)$$

or even:

$$Collate =_{def} F_1.v \vee F_2.v \rightsquigarrow F$$

But in both of these cases we have restricted our system in terms of eventual implementation; the purpose of a specification is to provide an abstract description which is free from implementation bias. Mixed programs are usually the result of stepwise refinement within a wide-spectrum language rather than an initial system description.

2.3. Derived Agent Forms

There are a number of useful agent forms which are commonly found in real-time system design. These are defined below.

Deadline: We define a deadline as a duration set with values which range from 0 to a given maximum. We overload the duration operator syntax:

$$[n]\mathscr{A} =_{def} [\{0..n\}]\mathscr{A}$$

Skip: We define an agent which may have a duration of zero, and which changes no shunts:

$$skip =_{def} \emptyset : true$$

Delay: We define an agent which delays for a minimum of n time unts, and changes no shunts:

$$\delta n =_{def} \emptyset : t' \geq t + n$$

2.4. Semantics

The semantics of TAM agents are given by formulae in the temporal logic. We also make extensions to axiomatise the behaviour of the real-time variable τ.

PAx. (Progress axiom) $\quad \forall n \Box(\tau = n \Rightarrow \circ\tau = n + 1)$

UAx (Uniqueness axiom) $\quad \bigwedge_{i \in I} \Diamond(\tau = n_i \wedge \Phi_i)$

$$\Leftrightarrow \Diamond(\tau = n_j \wedge \Phi_j \wedge \bigwedge_{i \in I - j} \Diamond(\tau = n_i \wedge \Phi_i))$$

The progress axiom asserts that in each state the value of τ is exactly one greater than the value in the previous state, and the uniqueness axiom asserts that any formula which is temporally anchored to the earliest time (of that set of times) may precede all other formulae.

We define some useful predicates on shunts in order to simplify the semantic definitions. The predicate '*stable*' asserts that the shunt s will not be changed during the given interval:

Definition (Stable)

$$stable(x, n, m) =_{def} \exists v(\bigwedge_{\sigma \in [n,m]} \diamond(\tau = \sigma \wedge x = v))$$

In addition, the definition for *stable* is extended to sets of shunts.

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:

Definition (Write)

$$write(s, \vartheta, n, m) =_{def}$$

$$\exists l \in [n, m](stable(s, n, l-1) \wedge \diamond(\tau = l \wedge s = (l, \vartheta)) \wedge stable(s, l, m))$$

We also define a chop operator:

Definition (Chop). Given two timed logic formulae $\mathscr{A}$ and $\mathscr{B}$, then,

$$\mathscr{A}^\frown\mathscr{B} =_{def} \exists m \in [t, t'](\mathscr{A}[m/t'] \wedge \mathscr{B}[m/t])$$

The semantics of an agent are now given by a timed logic formula. The specification statement is defined in this manner also, giving a natural interpretation for the refinement relation. Note that the semantics always assume an environment (i.e. no program can be given a semantics without the explicit definition of an environment).

Definition (Semantics)

$$[\![[w \bullet w_0 : \Phi]]\!] =_{def} \diamond(\tau = t \wedge stable(w - w_0, t, t') \wedge \Phi)$$

$$[\![[w \bullet \vartheta \rightsquigarrow s]]\!] =_{def} [\![[w \bullet \{s\} : write(s, \vartheta, t, t')]]\!]$$

$$[\![[w \bullet A; B]]\!] =_{def} [\![[w \bullet A]]\!]^\frown[\![[w \bullet B]]\!]$$

$$[\![[w \bullet A|B]]\!] =_{def} ([\![[w_A \bullet A]]\!]^\frown stable(w_A, t, t')) \wedge ([\![[w_B \bullet B]]\!]^\frown stable(w_B, t, t'))$$

$$[\![[w \bullet \bigsqcup_{i \in I} g_i \Rightarrow A_i]]\!] =_{def}$$

$$\diamond(\tau = t \wedge ((\bigwedge_{i \in I} \neg g_i \wedge stable(w, t, t')) \vee (\bigvee_{i \in I} (g_i \wedge [\![[w \bullet A_i]]\!]))))$$

$$[\![[w \bullet [S]A]]\!] =_{def} [\![[w \bullet A]]\!] \wedge t' - t \in S$$

$$[\![[w \bullet \mu_n + 1A]]\!] =_{def} [\![[w \bullet A]]\!]^\frown [\![[\mu_n A]]\!]$$

$$[\![[w \bullet \mu_0 A]]\!] =_{def} stable(w, t, t')$$

In the semantic definition of the concurrent operator, the partitioning of the environment w to w_A and w_B has to be complete (i.e. $w_A \cup w_B = w$) and disjoint (i.e. $w_A \cap w_B = \emptyset$), and all variables written to by A must be in w_A, and similarly for B. Variables and shunts which cannot be written by either agent may appear in either environment.

An important property of the semantics is that of 'temporal independence' i.e. the ability to remove any temporal anchor to the start time t. We express this property in the following theorem:

Theorem (Temporal Independence). Given any agent $\mathscr{A}$, then:

$$\diamond(\tau = t \wedge [\![A]\!]) \Leftrightarrow [\![A]\!]$$

The proof of this theorem is given in the appendix. It is used extensively in soundness proofs of the refinement calculus.

3. Refinement

3.1. A Refinement Relation

Refinement is defined as the *strengthening* of the logic formula given by the semantics of the agent. This models the intuitive definition of refinement as a lessening of non-determinism. We define a partial order on agents, denoted $\sqsubseteq$.

Definition (Refinement)

$$\mathscr{A} \sqsubseteq \mathscr{B} \text{ iff } [\![\mathscr{B}]\!] \Rightarrow [\![\mathscr{A}]\!]$$

It would be very time consuming to convert all agents to their logic formulae, and then prove that the implication holds; for any system of a reasonable size such proofs would be too complex. Instead we provide a number of refinement laws which together provide an (incomplete) (in)equational theory on agents, and which is sound with respect to the refinment relation. We list these rules below.

3.2. Refinement Laws

Each semantic definition in the previous section gives rise to a refinement law which we will label 'def...', for example:

$$[\{s : bool\} \bullet \{s\} : write(s, true, t, t')] \sqsubseteq [\{s : bool\} \bullet true \rightsquigarrow s] \quad (\text{def} \rightsquigarrow)$$

In addition we present a number of basic refinement laws below. Their soundness proofs rely on Manna and Pnueli's proof system, and theorems and inference

rules from this system are labelled Tn or Dn in the proofs in the appendix. These labels correspond to the ones given in [MaP83].

Specification

(contract frame) $w : \Phi \sqsubseteq w/s : \Phi$ (any shunt S)

(expand frame) $[w_0 \bullet w : \Phi] \sqsubseteq [w \cup \{x : T\} \bullet w \cup \{x\} : \Phi]$ (x new variable)

(strengthen) $w : \Phi \sqsubseteq w : \Psi$ (if $\Psi \Rightarrow \Phi$)

Sequence

(seq-assoc) $\mathscr{A};(\mathscr{B};\mathscr{C}) \sqsubseteq (\mathscr{A};\mathscr{B});\mathscr{C}$

Concurrent

(con-assoc) $\mathscr{A}|(\mathscr{B}|\mathscr{C}) \sqsubseteq (\mathscr{A}|\mathscr{B})|\mathscr{C}$

(split) $[w_0 \bullet \bigcup_{i\in I} w_i : \bigwedge_{i\in I} \Phi_i] \sqsubseteq [w_0 \bullet \prod_{i\in I} [w_{0i} \bullet w_i : \Phi_i]]$

(if $w_i \cap w_j = \emptyset$ for $i \neq j$, $w_{0i} \cap w_{0j} = \emptyset$ for $i \neq j$)

$\bigcup_{i\in I} w_{0i} = w_0$, $w_i \subseteq w_{0i}$ each i)

Deadline

(shorten) $[S]\mathscr{A} \sqsubseteq [S - S']\mathscr{A}$ (combine) $[S][T]\mathscr{A} \sqsubseteq [S \cap T]\mathscr{A}$

Guards

(choose) $\bigsqcup_{i\in I} g_i \Rightarrow \mathscr{A}_i \sqsubseteq \mathscr{A}_j$ (if $j \in I \wedge g_j$)

(nd) $g_1 \Rightarrow \mathscr{A} \sqcup g_2 \Rightarrow \mathscr{A} \sqsubseteq g_1 \vee g_2 \Rightarrow \mathscr{A}$

Recursion

(basis) $w : \exists \tilde{m}_i (\bigwedge_{i\in I} (\Phi_i[m_{i-1}/t][m_i/t']) \wedge stable(w_0, m_n, t')[t/m_0])$
$\sqsubseteq \mu_n w : \Phi$ (if $I = [1, n]$)

where $\exists \tilde{m}_i$ denotes the nested existential quantification of the variables m_i with some indexing set such that $i \in I$.

Composite agents allow us to describe complex behaviour by combining simpler agents. However, in system development we would not want to complicate the refinement method in proportion to the complexity of the system, instead we would wish to have simple refinement obligations which could be trivially composed in order to discharge complex refinement proof obligations.

In [Hoo91] the importance of compositionality of proof systems for concurrent real-time formalisms is discussed. We assert that the refinement calculus for TAM is compositional, i.e. systems can be sub-divided into sub-systems which may

then be refined in isolation, and recomposed to give a system which is a valid refinement of the original specification. This form of compositionality is clearly dependent upon the fact that refinement cannot break the interference constraint on concurrent systems, and similarly cannot introduce unrestricted shunts. This property of compositionality also holds for the other agent constructors (deadline, sequence, variable declaration, shunt restriction, guards and recursion), and in the refinement calculus, compositionality equates to the property of *monotonicity.*

We therefore assert the following theorem:

Theorem (Refinement Monotonicity)

If $\mathscr{A} \sqsubseteq \mathscr{B}$ then for any context $\mathscr{C}(\_)$ we have $\mathscr{C}(\mathscr{A}) \sqsubseteq \mathscr{C}(\mathscr{B})$

The proof of this theorem is given in the appendix.

4. Example

Consider a manufacturing plant in which a control system is required to count the number of items on a conveyor belt. Each item may be of one of two colours: black or red. Assuming that the items do not arrive at a faster rate than one every five time units, then the system is required to count the number of items of each colour that arrive over a given interval.

We can model this system by a single input shunt labelled '*in*', and two output shunts labelled '$count_b$' and $count_r$. Writes to the input shunt represent the arrival of items, and the numbers held in the output shunts represent the total number of items of each colour. For simplicity we will assume that all positive integer shunts initially have the value 0. We start by defining a useful predicate '*count*':

$$count(x, X) =_{def}$$

$$\exists n(n = |\{l \in [t, t' - 5] : \diamond(\tau = l \wedge in = (l, x))\}| \wedge \diamond(\tau = t' \wedge X.v = n))$$

This predicate asserts that at the end of the given interval (t') the count in shunt X is equal to the number of instances in time when a value x is written to the shunt *in*.

We may now give a complete specification:

$$Spec =_{def} [\{count_b, count_r : Z^+\}\bullet$$

$$\{count_b, count_r\} : count(black, count_b)$$

$$\wedge count(rec, count_r)]$$

In addition we assert an environment axiom which provides an upper bound on the item arrival rate, and an environment axiom which asserts that initially the shunts all have a reasonable value.

EA.1 $\forall n, m \in [t, t'](\diamond(\tau = n \wedge in.ts = n)$

$$\wedge \diamond (\tau = m \wedge in.ts = m) \Rightarrow abs(n - m) \geq 5)$$

EA.2 $\diamond(\tau = t \wedge count_b.v = 0 \wedge count_r.v = 0)$

We may now start the refinement process. Step one is to introduce a new shunt.

$$Spec \sqsubseteq [\{count_b, count_r, lastTime : Z^+\}\bullet$$

$$\{count_b, count_r mlastTime\} : count(black, count_b) \wedge count(red, count_r)]$$

(expand frame)

The new shunt will keep a record of the time of the last write to the shunt *in* so that time-stamps may be compared. From this point onwards we will drop any reference to the environment which does not change. The next refinement predicates over the new shunt and asserts that the value found in *lastTime* is always the most recent time-stamp in the shunt *in* (not including any time-stamp written in the last five time units).

$$\sqsubseteq \{count_r, count_b, lastTime\} : count(black, count_b) \wedge count(red, count_r)$$

$$\wedge \forall n \in [t, t'](\diamond(\tau = n \wedge lastTime.v = writes(t, n-5)))$$

(strengthen)

where $writes(t,n) =_{def} max\{l \in [t,n] \bullet \diamond(\tau = l \wedge in.ts = l)\}$

The next step is to partition the behaviour into sections which last for five time units (one of a number of possible design decisions).

$$\sqsubseteq \{count_b, count_r, lastTime\} :$$

$$\exists \tilde{m}_i(\bigwedge_{i\in[1,\eta]} \Phi[m_{i-1}/t][m_i/t'] \wedge stable(w, m_\eta, t'))[t/m_0]$$

(strengthen)

Where $\eta =_{def} |\{l \in (t,t'] \bullet l \ mod\ 5\}|$ (all members of $(t,t']$ which divide by 5).

Where $\Phi =_{def}$

$$\begin{aligned} t'-t = 5 \wedge \diamond(\tau = t \wedge \exists m,n,l(&count_b.v = m \wedge count_r.v = n \wedge in.ts = l \wedge \\ &((l > lastTime.v \wedge in.v = black) \Rightarrow \\ &\quad \diamond(\tau = t' \wedge count_b.v - m + 1 \\ &\quad \wedge lastTime.v = l)) \wedge \\ &((l > lastTime.v \wedge in.v = red) \Rightarrow \\ &\quad \diamond(\tau = t' \wedge count_r.v - n + 1 \\ &\quad \wedge lastTime.v = l)))) \end{aligned}$$

Note that this last refinement step is the most complex one and it is dependent upon the environment axioms $EA.1$ and $EA.2$. This partitioned form allows us to refine to an iterative agent:

$$\sqsubseteq \mu_n\{count_b, count_r, lastTime\} : \Phi$$

Now we may concentrate on the iterated specification agent. The definition of

deadline, delay, and concurrency allows us to perform the following refinement:

$$\{count_b, count_r, lastTime\} : \Phi$$

$$\sqsubseteq\ [\{5\}](\delta 5 | \{count_b, count_r, lastTime\} : \Phi')$$

where Φ' is Φ without the initial conjunct on t and t'. Now we refine the remaining specification:

$$\{count_b, count_r, lastTime\} : \Phi'$$

$$\begin{array}{l}
\sqsubseteq\ in.ts > lastTime.v \wedge in.v = black \\
\qquad \Rightarrow \{count_r, count_b, lastTime\} : write(count_b, count_b.v + 1, t, t') \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge write(lastTime, in.ts, t, t') \\
\quad \sqcup \\
\quad in.ts > lastTime.v \wedge in.v = red \\
\qquad \Rightarrow \{count_r, count_b, lastTime\} : write(count_r, count_r.v + 1, t, t') \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge write(lastTime, in.ts, t, t') \\
\quad \text{(def guards)}
\end{array}$$

Now each guarded agent may be refined to a pair of concurrent outputs:

$$\{count_b, count_r, lastTime\} : write(count_b, count_b.v + 1, t, t')$$

$$\wedge write(lastTime, in.ts, t, t')$$

$$\sqsubseteq\ count_b.v + 1 \rightsquigarrow count_b \mid in.ts \rightsquigarrow lastTime$$

(def write, concurrent)

This is the final refinement setp, and we may now compose all of the above refinements into a single program:

$$\begin{array}{l}
\sqsubseteq\ \mu_n[\{5\}](\delta 5 | in.ts > lastTime.v \wedge in.v = black \\
\qquad\qquad \Rightarrow (count_b.v + 1 \rightsquigarrow count_b \mid in.ts \rightsquigarrow lastTime) \\
\qquad \sqcup \\
\qquad\qquad in.ts > lastTime.v \wedge in.v = red \\
\qquad\qquad \Rightarrow (count_r.v + 1 \rightsquigarrow count_r \mid in.ts \rightsquigarrow lastTime))
\end{array}$$

5. Conclusions

TAM is unique in providing a wide-spectrum development language for real-time systems in which abstract specifications can be refined down to concrete executable programs. Wide-spectrum languages for non real-time systems have been studied extensively, for example in the SETL language [ShS85], and the CIP project [CIP85], wide-spectrum languages based upon predicate logic are given transformation rules which allow refinement in a manner similar to TAM.

The utility of a wide-spectrum language can be clearly seen in the refinement method used by Morgan in his calculus [Mor90,MRG88]. In this language, the

concrete syntax is provided by Dijkstra's Guarded Command Language (GCL) [Dij76]. The abstract specification syntax is provided by a statement form:

$$w : [pre, post]$$

where 'w' (called the 'frame') defines the scope of the specification, that is those state variables which may be changed by the behaviour defined by the specification, and *pre* and *post* are first-order predicate logic formulae which describe the relationship between the program state before the 'execution' of the specification statement and after the termination of the specification statement respectively. The specification statement can therefore be viewed as a description of the minimum requirements on the behaviour of any concrete statement which may replace it during refinement.

Similarly, in Back and von Wright's wide-spectrum language [BvW90], the concrete code is a version of Dijkstra's GCL and a statement called an *assert* statement is denoted $\{b\}$, where b is a formula on the local state. The assert statement will terminate correctly only if the local state satisfies the formula when 'executed', otherwise it aborts.

Original work by Back [Bac80], and later [Bac88], can be seen as the first investigation of adding specification statements to programming languages to aid in the process of verification.

However, all of these languages are transformational; they describe computations which have all input data available at the start of execution, and provide a result at their time of termination. This restriction provides the 'shape' of Morgan's specification statement – it describes a relationship on initial and final states. In real-time systems we are interested in input and output during the execution of an agent. In addition, we are interested in the time at which the output and inputs occur; our specification statement forms a timed invariant which reflects these needs.

The next step is to provide automated support for the refinement process which will enable us to tackle much more ambitious software systems. We also plan to extend TAM to assist in programming 'in the large'.

Acknowledgements

The author would like to extend his thanks to Professor Jifeng He of the Programming Research Group, Oxford University, for his advice on the specification oriented semantics.

References

[Bac80] Back, R. J. R. "Correctness Preserving Program Refinements: Proof Theory and Applications", Tract 131, Mathematisch Centrum, Amsterdam. 1980.

[Bac88] Back, R. J. R. "A Calculus of Refinements for Program Derivations", Acta-Informatica, 25, p593-624. 1988.

[Bar85] Barringer, H. "A Survey of Verification Techniques for Parallel Programs", LNCS 191, Springer-Verlag. 1985.

[BvW90] Back, R. J. R., von Wright,J. "Refinement Concepts Formalised in Higher-Order Logic", BCS Formal Aspects of Computing, Vol 2, No. 3. 1990.

[CIP85] The CIP Language Group, "The Munich Project CIP: Vol1", LNCS 183, Springer-Verlag. 1985.

[Dij786] Dijkstra, E. "A Discipline of Programming", Prentice-Hall. 1976.
[Hoo91] Hooman, J. "Specification and Compositional Verification of Real-Time Systems", Ph.D. Thesis, Technical University of Eindhoven. 1991.
[KdR85] Koymans, R., deRoever, W. P. "Examples of a Real-Time Temporal Logic Specification", in *The Analysis of Concurrent Systems*, LNCS 207, Springer-Verlag. 1985.
[Mor90] Morgan, C. "Programming from Specifications", Prentice-Hall International. 1990.
[MaP83] Manna, Z., Pnueli, A. "Verification of Concurrent Programs: a Temporal Proof System", Technical Report, Dept. Computer Science, Stanford University. June 1983.
[MRG88] Morgan, C., Robinson, K., Gardiner, P. "On The Refinement Calculus", Oxford University Programming Research Group, Technical Report PRG-70. October 1988.
[Sch92] Scholefield, D. "A Refinement Calculus for Real-Time Systems", Department of Computer Science D.Phil. Thesis, University of York. July 1992.
[ShS85] Schonberg, E., Shields, D. "From Prototype to Efficient Implementation: a Case Study Using SETL and C", Courant Institute of Mathematical Sciences, Dept. Computer Science, New York University. 1985.
[SZH93] Scholefield, D. J., Zedan, H. S. M., He, J. "A Specification Oriented Semantics for Real-Time Refinement", Theoretical Computer Science, Vol 131, p219-241. 1994.
[ZHR91] Chaochen, Z., Hoare, C. A. R., Ravn, A. P. "A Calculus of Durations", Information Processing Letters, Vol 40, No. 5, p269-76. 1991.

A. Proofs

A.1. Temporal Independence

Specification

Lemma. $\diamond(\Phi \wedge \diamond(\Phi \wedge \Psi)) \Leftrightarrow \diamond(\Phi \wedge \Psi)$

1.	$\diamond(\Phi \wedge \diamond(\Phi \wedge \Psi))$	(hypothesis)
2.	$\diamond(\Phi \wedge \Psi)$	1, T10,T4
3.	$\Phi \wedge \Psi$	(hypothesis)
4.	Φ	3, $\wedge$-elim
5.	$\diamond(\Phi \wedge \Psi)$	3.T1
6.	$\Phi \wedge \diamond(\Phi \wedge \Psi)$	4,5,$\wedge$-intro
7.	$\Phi \wedge \Psi \Rightarrow \Phi \wedge \diamond(\Phi \wedge \Psi)$	3,6,$\Rightarrow$-intro
8.	$\diamond(\Phi \wedge \Psi) \Rightarrow \Phi \wedge \diamond(\Phi \wedge \Psi)$	7,D5
9	$\diamond(\Phi \wedge \diamond(\Phi \wedge \Psi)) \Rightarrow \diamond(\Phi \wedge \Psi)$	1,2, $\Rightarrow$-intro
10.	$\diamond(\Phi \wedge \diamond(\Phi \wedge \Psi)) \Leftrightarrow \diamond(\Phi \wedge \Psi)$	8,9, $\Leftrightarrow$-intro

Proof. Substitute $stable(w_0 - w, t, t') \wedge \tau = t$ for Ψ in lemma. □

Deadline

Lemma. $\diamond(\tau = t \wedge \Phi) \wedge t' - t \in S \Leftrightarrow \diamond(t = \tau \wedge \diamond(\tau = t \wedge \Phi) \wedge t' - t \in S)$

1.	$\diamond(\tau = t \wedge \tau(\tau = t \wedge \Phi) \wedge t' - t \in S)$	(hypothesis)
2.	$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi)) \wedge t' - t \in S$	1,T61
3.	$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi))$	2, $\wedge$-elim
4.	$t' - t \in S$	2,$\wedge$-elim
5.	$\diamond(\tau = t) \wedge \diamond \diamond (\tau = t \wedge \Phi)$	3,T10
6.	$\diamond \diamond (\tau = t \wedge \Phi)$	5,$\wedge$-elim
7.	$\diamond(\tau = t \wedge \Phi)$	6,T4

8.	$\diamond(\tau = t \wedge \Phi) \wedge t' - t \in S$	4,7,$\wedge$-intro
9.	$\diamond(\tau = t \wedge \Phi) \wedge t' - t \in S$	(hypothesis)
10.	$\diamond(\tau = t \wedge \Phi)$	9, $\wedge$-elim
11.	$\diamond\tau = t \wedge \diamond\Phi$	10,T10
12.	$\diamond\tau = t$	11,$\wedge$-elim
13.	$\diamond\tau = t \wedge \diamond(\tau = t \wedge \Phi)$	10,12,$\wedge$-intro
14.	$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi)) \wedge t' - t \in S$	UAx,9,13,$\wedge$-elim
15.	$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge t' - t \in S)$	14,T61
16.	$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge t' - t \in S)$ $\Rightarrow \diamond(\tau = t \wedge \Phi) \wedge t' - t \in S$	1,8,$\Rightarrow$-intro
17.	$\diamond(\tau = t \wedge \Phi) \wedge t' - t \in S$ $\Rightarrow \diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge t' - t \in S)$	9,15,$\Rightarrow$-intro
17.	$\diamond(\tau = t \wedge \Phi) \wedge t' - t \in S$ $\Leftrightarrow \diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge t' - t \in S)$	9,17,$\Leftrightarrow$-intro

Proof. Substitute $[\![[w \bullet A]]\!]$ for Φ in lemma. □

Output

Proof. Defined in terms of specification statement which is temporally independent. □

Sequence

We have to show that:

$$\diamond(\tau = t \wedge \exists m \in [t, t'](([\![[w \bullet A\,]\!][m/t'] \wedge [\![[w \bullet B]]\!][m/t]))$$

$$\Leftrightarrow \exists n \in [t, t'](([\![[w \bullet A\,]\!][n/t'] \wedge [\![[w \bullet B]]\!][n/t])$$

We make the following substitutions (assuming temporal independence of agents $\mathscr{A}$ and $\mathscr{B}$):

$$[\![[w \bullet A]]\!][m/t'] = \diamond(\tau = t \wedge \Phi_m)$$

$$[\![[w \bullet A]]\!][n/t'] = \diamond(\tau = t \wedge \Phi_n)$$

$$[\![[w \bullet B]]\!][m/t] = \diamond(\tau = t \wedge \Psi_m)$$

$$[\![[w \bullet B]]\!][n/t] = \diamond(\tau = t \wedge \Psi_n)$$

We therefore have to prove the theorem:

$$\diamond(\tau = t \wedge \exists m \in [t, t'](\diamond(\tau = t \wedge \Phi_m) \wedge \diamond(\tau = t \wedge \Psi_m)))$$

$$\Leftrightarrow \exists n \in [t, t'](\diamond(\tau = t \wedge \Phi_n) \wedge \diamond(\tau = t \wedge \Psi_n))$$

1.	$\diamond(\tau = t \wedge \exists m \in [t, t'](\diamond(\tau = t \wedge \Phi_m) \wedge \diamond(\tau = t \wedge \Psi_m)))$	(hypothesis)
2.	$\diamond(\exists m \in [t, t'](\diamond(\tau = t \wedge \Phi_m) \wedge \diamond(\tau = t \wedge \Psi_m)))$	1,T10,$\wedge$-elim
3.	$\exists m \in [t, t'](\diamond(\diamond(\tau = t \wedge \Phi_m) \wedge \diamond(\tau = t \wedge \Psi_m))$	2,T48
4.	$\diamond(\diamond(\tau = t \wedge \Phi_a) \wedge \diamond(\tau = a \wedge \Psi_a))$	3,$\exists$-elim
5.	$\diamond(\diamond(\tau = t \wedge \Phi_a) \wedge \diamond\diamond(\tau = a \wedge \Psi_a))$	4,T10,

6.	$\diamond(\diamond(\tau = t \wedge \Phi_a))$	5, $\wedge$-elim
7.	$\diamond(\tau = t \wedge \Phi_a))$	6,T4
8.	$\diamond\diamond(\tau = t \wedge \Psi_a))$	4, $\wedge$-elim
9.	$\diamond(\tau = t \wedge \Psi_a))$	8,T4
10.	$\diamond(\tau = t \wedge \Phi_a) \wedge \diamond(\tau = t \wedge \Psi_a)$	7,9,$\wedge$-intro
11.	$\exists m \in [t,t'](\diamond(\tau = t \wedge \Phi_m) \wedge \diamond(\tau = t \wedge \Psi_m))$	10,$\exists$-intro
12.	$\diamond(\tau = t \wedge \exists m \in [t,t'](\diamond(\tau = t \wedge \Phi_m) \wedge \diamond(\tau = t \wedge \Psi_m)))$ $\Rightarrow \exists n \in [t,t'](\diamond(\tau = t \wedge \Phi_n) \wedge \diamond(\tau = t \wedge \Psi_n))$	1,11,$\Rightarrow$-intro+rename
13.	$\exists n \in [t,t'](\diamond(\tau = t \wedge \Phi_n) \wedge \diamond(\tau = t \wedge \Psi_n))$	(hypothesis)
14.	$\diamond(\tau = t \wedge \Phi_a) \wedge \diamond(\tau = t \wedge \Psi_a)$	13,$\exists$-elim
15.	$\diamond(\tau = t \wedge \Phi_a)$	14, $\wedge$-elim
16.	$\diamond\tau = t \wedge \diamond\Phi_a$	15,T10
17.	$\diamond\tau = t$	16, $\wedge$-elim
18.	$\diamond\tau = t \wedge \diamond(\tau = t \wedge \Phi_a)$	15,17,$\wedge$-intro
19.	$\diamond(\tau = t \wedge \Psi_a)$	14,$\wedge$-elim
20.	$\diamond\tau = t \wedge \diamond(\tau = t \wedge \Phi_a) \wedge \diamond(\tau = t \wedge \Psi_a)$	18,19,$\wedge$-intro
21.	$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi_a) \wedge \diamond(\tau = t \wedge \Psi_a))$	20,UAx,†
22.	$\diamond(\tau = t \wedge \exists n \in [t,t'](\diamond(\tau = t \wedge \Phi_n) \wedge \diamond(\tau = t \wedge \Psi_n)))$	21,$\exists$-intro
23.	$\exists n \in [t,t'](\diamond(\tau = t \wedge \Phi_n) \wedge \diamond(\tau = t \wedge \Psi_n))$ $\Rightarrow \diamond(\tau = t \wedge \exists n \in [t,t'](\diamond(\tau = t \wedge \Phi_n) \wedge \diamond(\tau = t \wedge \Psi_n)))$	13,22,$\Rightarrow$-intro
24.	$\diamond(\tau = t \wedge \exists m \in [t,t'](\diamond(\tau = t \wedge \Phi_m) \wedge \diamond(\tau = t \wedge \Psi_m)))$ $\Rightarrow \exists n \in [t,t'](\diamond(\tau = t \wedge \Phi_n) \wedge \diamond(\tau = t \wedge \Psi_n))$	12,23,$\Leftrightarrow$-intro

† requires $t \leq a$, but $a \in [t,t']$ so trivially holds. □

Iteration

Proof. Follows directly from temporal independence of sequence. □

Concurrent

The theorem we need to prove is that:

$$\diamond(\tau = t \wedge [\![[w_A \bullet A]]\!]^\frown stable(w_A,t,t') \wedge [\![[w_B \bullet B]]\!]^\frown stable(w_B,t,t'))$$

$$\Leftrightarrow [\![[w_A \bullet A]]\!]^\frown stable(w_A,t,t') \wedge [\![[w_B \bullet B]]\!]^\frown stable(w_B,t,t')$$

We know that the $^\frown$ operator preserves temporal independence (by proof of sequence temporal independence) so we make the following correspondence:

$$[\![[w_A \bullet A]]\!]^\frown stable(w_A,t,t') = \Phi$$

$$[\![[w_B \bullet B]]\!]^\frown stable(w_B,t,t') = \Psi$$

We now have to prove the theorem:

$$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi)) \Leftrightarrow \diamond(\tau = t \wedge \Phi) \wedge (\tau = t \wedge \Psi)$$

1.	$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi))$	(hypothesis)
2.	$\diamond\tau = t \wedge \diamond\diamond(\tau = t \wedge \Phi) \wedge \diamond\diamond(\tau = t \wedge \Psi)$	1,T10
3.	$\diamond\diamond(\tau = t \wedge \Phi)$	2,$\wedge$-elim
4.	$\diamond(\tau = t \wedge \Phi)$	3,T4
5.	$\diamond\diamond(\tau = t \wedge \Psi)$	2,$\wedge$-elim

6.	$\diamond(\tau = t \wedge \Psi)$	5,T4
7.	$\diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi)$	4,5,$\wedge$-intro
8.	$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi))$ $\Rightarrow \diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi)$	1,7,$\Rightarrow$-intro
9.	$\diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi)$	(hypothesis)
10.	$\diamond(\tau = t \wedge \Phi)$	9,$\wedge$-elim
11.	$\diamond\tau = t \wedge \diamond\Phi$	10,T10
12.	$\diamond(\tau = t \wedge \Psi)$	9,$\wedge$-elim
13.	$\diamond\tau = t$	11,$\wedge$-elim
14.	$\diamond\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi)$	10,12,13,$\wedge$-intro
15.	$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi))$	14,UAx
16.	$\diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi)$ $\Rightarrow \diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi))$	9,15,$\Rightarrow$-intro
17.	$\diamond(\tau = t \wedge \diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi))$ $\Leftrightarrow \diamond(\tau = t \wedge \Phi) \wedge \diamond(\tau = t \wedge \Psi)$	8,16,$\Leftrightarrow$-intro □

Guards

We may substitute the two cases (where one of the guards is true, and none of the cases are true), for Φ in the specification lemma. □

A.2. Refinement Monotonicity

Sequence (Right)

We need to prove that if $\mathscr{A} \sqsubseteq \mathscr{B}$ then $\mathscr{C};\mathscr{A} \sqsubseteq \mathscr{C};\mathscr{B}$

we start with $[\![[w \bullet \mathscr{C};\mathscr{B}]]\!] = \exists m \in [t,t'](([\![[w \bullet \mathscr{C}]]\!][m/t'] \wedge [\![[w \bullet \mathscr{B}]]\!][m/t])$

1.	$\exists m \in [t,t'](([\![[w \bullet \mathscr{C}]]\!][m/t'] \wedge [\![[w \bullet \mathscr{B}]]\!][m/t])$	(hypothesis)
2.	$[\![[w \bullet \mathscr{C}]]\!][a/t'] \wedge [\![[w \bullet \mathscr{B}]]\!][a/t])$	(1,$\exists$-elim)
3.	$[\![[w \bullet \mathscr{C}]]\!][a/t']$	2,$\wedge$-elim
4.	$[\![[w \bullet \mathscr{B}]]\!][a/t]$	2,$\wedge$-elim
5.	$[\![[w \bullet \mathscr{B}]]\!] \Rightarrow [\![[w \bullet \mathscr{A}]]\!]$	(given)
6.	$[\![[w \bullet \mathscr{B}]]\!][a/t'] \Rightarrow [\![[w \bullet \mathscr{A}]]\!][a/t]$	5,substitution
7.	$[\![[w \bullet \mathscr{A}]]\!][a/t]$	4,6,MP
8.	$[\![[w \bullet \mathscr{C}]]\!][a/t'] \wedge [\![[w \bullet \mathscr{A}]]\!][a/t]$	3,7,$\wedge$-intro
9.	$\exists m \in [t,t'](([\![[w \bullet \mathscr{C}]]\!][m/t'] \wedge [\![[w \bullet \mathscr{A}]]\!][m/t])$ $= [\![[w \bullet \mathscr{C};\mathscr{A}]]\!]$	8,$\exists$-intro (def ;) □

Sequence (Left)

Proof. As for sequence right. □

Concurrent

We need to prove that if $\mathscr{A} \sqsubseteq \mathscr{B}$ then $\mathscr{A}|\mathscr{C} \sqsubseteq \mathscr{B}|\mathscr{C}$ (symmetric subcase is proven by the concurrent commutativity of the soundness proof below). We start by

asserting that:

$$[\![[w \bullet \mathscr{B}|\mathscr{C}]]\!] \;=\; \exists m \in [t,t'](([\![[w_B \bullet \mathscr{B}]]\!][m/t'] \wedge stable(w_B, m, t')) \wedge$$

$$\exists l \in [t,t'](([\![[w_C \bullet \mathscr{C}]]\!][l/t'] \wedge stable(w_C, l, t'))$$

1.	$\exists m \in [t,t'](([\![[w_B \bullet B]]\!][m/t'] \wedge stable(w_B, m, t')) \wedge$ $\exists l \in [t,t'](([\![[w_C \bullet C]]\!][l/t'] \wedge stable(w_C, l, t')) \wedge$	(hypothesis)
2.	$\exists m \in [t,t'](([\![[w_B \bullet B]]\!][m/t'] \wedge stable(w_B, m, t'))$	1,$\wedge$-elim
3.	$[\![[w_B \bullet B]]\!][a/t'] \wedge stable(w_B, a, t'))$	2,$\exists$-elim
4.	$[\![[w_B \bullet B]]\!][a/t']$	3,$\wedge$-elim
5.	$stable(w_B, a, t'))$	3,$\wedge$-elim
6.	$[\![[w_B \bullet \mathscr{B}]]\!] \Rightarrow [\![[w_A \bullet \mathscr{A}]]\!]$	(given)
7.	$[\![[w_B \bullet \mathscr{B}]]\!][a/t'] \Rightarrow [\![[w_A \bullet \mathscr{A}]]\!][a/t]$	6, substitution
8.	$[\![[w_A \bullet \mathscr{A}]]\!][a/t']$	3,7,MP
9.	$stable(w_B, a, t') \Rightarrow stable(w_A, a, t')$	lemma (stable)
10.	$stable(w_A, a, t')$	5,9,MP
11.	$[\![[w_A \bullet \mathscr{A}]]\!][a/t'] \wedge stable(w_A, a, t')$	8,10,$\wedge$-intro
12.	$\exists m \in [t,t'](([\![[w_A \bullet \mathscr{A}]]\!][m/t'] \wedge stable(w_A, m, t'))$	11,$\exists$-intro
13.	$\exists l \in [t,t'](([\![[w_C \bullet \mathscr{C}]]\!][l/t'] \wedge stable(w_C, l, t'))$	2, $\wedge$-elim
14.	$\exists m \in [t,t'](([\![[w_A \bullet \mathscr{A}]]\!][m/t'] \wedge stable(w_A, m, t')) \wedge$ $\exists l \in [t,t'](([\![[w_C \bullet \mathscr{C}]]\!][l/t'] \wedge stable(w_C, l, t'))$	12,13,$\wedge$-intro
	$= [\![[w \bullet \mathscr{A}|\mathscr{C}]]\!]$	(def |) □

Note that we assume that the $w_A \subset w_B$ and so we may implicitly contract the environment frame w during this refinement. Alternatively we may assert that $w_A = w_B$.

Guards

We wish to prove that of $\mathscr{A}_j \sqsubseteq \mathscr{A}_j$' then $\bigsqcup_{i \in I} g_i \Rightarrow \mathscr{A}_i \sqsubseteq \bigsqcup_{i \in I} g_i \Rightarrow \mathscr{A}_i[\mathscr{A}_j'/\mathscr{A}_j]$

The proof is trivial. Consider two cases: firstly, where none of the guards are true then the LHS and RHS are identical regardless of the agents; secondly, when one of the guards evaluates to true we have two subcases:

2.1 when g_j is not true, and the refinement is unaffected,

2.2 when g_j is true, and $\mathscr{A}_j$ is chosen we have to prove:

$$g_j \wedge [\![[w \bullet \mathscr{A}_j']]\!] \Rightarrow [\![[w \bullet \mathscr{A}_j]]\!]$$

but this is clearly true as we have $[\![[w \bullet \mathscr{A}_j']]\!] \Rightarrow [\![[w \bullet \mathscr{A}]]\!]$ □

Deadline

We need to prove that if $\mathscr{A} \sqsubseteq \mathscr{B}$ then $[S]\mathscr{A} \sqsubseteq [S]\mathscr{B}$. We start by asseting that:

$$[\![[w \bullet [S]B]]\!] = [\![[w \bullet \mathscr{B}] \wedge t' - t \in S$$

1.	$[\![[w \bullet \mathscr{B}] \wedge t' - t \in S$	(hypothesis)
2.	$[\![[w \bullet B]]\!] \Rightarrow [\![[w \bullet A]]\!]$	(given)

3.	$[\![[w \bullet B]]\!]$	1,∧-elim
4.	$[\![[w \bullet A]]\!]$	2,3,MP
5.	$t' - t \in S$	1,∧-elim
6.	$[\![[w \bullet \mathscr{A}]]\!] \wedge t' - t \in S$	4,5,∧-intro
	$= [\![[w \bullet [S]\mathscr{A}]]\!]$	(def [S]) □

Recursion

We need to prove that if $\mathscr{A} \sqsubseteq \mathscr{B}$ then $\mu_n \mathscr{A} \sqsubseteq \mu_n \mathscr{B}$

Proof. Non-zero recursion reduces to sequentiality, which has been proven, and zero recursion gives us an identical formula for any agent (by definition). □

A.3. Refinement Soundness

Proof of (contract frame)

1.	$\tau = t \wedge \mathit{stable}(w_0 - w/x, t, t') \wedge \Phi$	(hypothesis)
2.	$\mathit{stable}(w_0 - w/x, t, t')$	1,∧-elim
3.	$\tau = t \wedge \Phi$	1,∧-elim
4.	$\mathit{stable}(w_0 - w/x, t, t') \Rightarrow \mathit{stable}(w_0 - w, t, ')$	lemma (stable)
5.	$\mathit{stable}(w_0 - w, t, t')$	2,4,MP
6.	$\tau = t \wedge \mathit{stable}(w_0 - w, t, t') \wedge \Phi$	3,5,∧-intro
7.	$\tau = t \wedge \mathit{stable}(w_0 - w/x, t, t') \wedge \Phi$	
	$\Rightarrow \tau = t \wedge \mathit{stable}(w_0 - w, t, t') \wedge \Phi$	1,6,MP
8.	$\Diamond(\tau = t \wedge \mathit{stable}(w_0 - w/x, t, t') \wedge \Phi)$	
	$\Rightarrow \Diamond(\tau = t \wedge \mathit{stable}(w_0 - w, t, t') \wedge \Phi)$	7,D5 □

Proof of (expand frame)

1.	$\tau = t \wedge \mathit{stable}(w \cup \{x : T\} - w \cup \{x\}, t, t') \wedge \Phi$	(hypothesis)
2.	$\mathit{stable}(w \cup \{x : T\} - w \cup \{x\}, t, t')$	1,∧-elim
3.	$\tau = t \wedge \Phi$	1,∧-elim
4.	$\mathit{stable}(w \cup \{x : T\} - w \cup \{x\}, t, t') \Rightarrow \mathit{stable}(w_0 - w, t, t')$	lemma (stable)
5.	$\mathit{stable}(w_0 - w, t, t')$	2,4,MP
6.	$\tau = t \wedge \mathit{stable}(w_0 - w, t, t') \wedge \Phi$	3,5,∧-intro
7.	$\tau = t \wedge \mathit{stable}(w \cup \{x : T\} - w \cup \{x\}, t, t') \wedge \Phi$	
	$\Rightarrow \tau = t \wedge \mathit{stable}(w \cup \{x : T\} - w \cup \{x\}, t, t') \wedge \Phi$	1,6,MP
8.	$\Diamond(\tau = t \wedge \mathit{stable}(w \cup \{x : T\} - w \cup \{x\}, t, t') \wedge \Phi)$	
	$\Rightarrow \Diamond(\tau = t \wedge \mathit{stable}(w \cup \{x : T\} - w \cup \{x\}, t, t') \wedge \Phi)$	7,D5

Proof of (strengthen)

1.	$\Diamond(\tau = t \wedge \mathit{stable}(w_0 - w, t, t') \wedge \Phi$	(def spec)
2.	$\tau = t \wedge \mathit{stable}(w_0 - w, t, t') \wedge \Psi$	(hypothesis)
3.	$\Psi \Rightarrow \Phi$	(given)
4.	Ψ	2, ∧-elim
5.	Φ	3,4,MP
6.	$\tau = t \wedge \mathit{stable}(w_0 - w, t, t')$	2,∧-elim
7.	$\tau = t \wedge \mathit{stable}(w_0 - w, t, t') \wedge \Phi$	2,6,∧-intro

8. $\tau = t \wedge stable(w_0 - w, t, t') \wedge \Psi$
$\Rightarrow \tau = t \wedge stable(w_0 - w, t, t') \wedge \Phi$ 2,7,⇒-intro
9. $\diamond(\tau = t \wedge stable(w_0 - w, t, t') \wedge \Phi)$ 1,8,D5 □

Proof of (seq-assoc). Trivial by associativity of $\frown$ (through associativity of conjunction). □

Proof of (con-assoc). Trivial by associativity of conjunction. □

Proof of (con-com). Trivial by commutativity of conjunction. □

Proof of (split)

$[\![[w_0 \bullet \bigcup_{i\in I} w_i : \bigwedge_{i\in I} \Phi_i[\![= \bigwedge_{i\in I} ([\![[w_{0i} \bullet w_i : \Phi_i]]\!] \frown stable(w_{0i}, t, t'))$ (def |)

1. $\bigwedge_{i\in I} ([\![[w_{0i} \bullet w_i : \Phi_i]]\!] \frown stable(w_{0i}, t, t'))$ (hypothesis)
2. $\bigwedge_{i\in I} \exists \tilde{m}_i \in [t, t'](([\![[w_{0i} \bullet w_i : \Phi_i]]\!][m_i/t'] \wedge stable(w_{0i}, m_i, t')))$
1,(def $\frown$)
3. $\bigwedge_{i\in I} \exists m_i \in [t, t'](([\![[w_{0i} \bullet w_i : \Phi_i]]\!][m/t'] \wedge stable(w_0, m, t')))$
2,(specific)
4. $\exists m \in [t, t'](\bigwedge_{i\in I} [\![[w_{0i} \bullet w_i : \Phi_i]]\!][m/t'] \wedge stable(w_0, m, t)$
3, $\bigcup_{i\in I} w_{0i} = w_0$
5. $\exists m \in [t, t'](([\![[w_0 \bullet \bigcup_{i\in I} w_i : \bigwedge_{i\in I} \Phi_i]]\!][m/t'] \wedge stable(w_0, m, t'))$
4,lemma
6. $[w_0 \bullet \bigcup_{i\in I} w_i : \bigwedge_{i\in I} \Phi_i][\![; stable(w_0, t, t')$ 5,(def ;)

and we have that $[w \bullet A; stable(w, t, t')] \sqsubseteq [w \bullet A]$ □

Proof of (shorten)

1. $[\![[w \bullet A]]\!] \wedge t' - t \in S$ (hypothesis)
2. $t' - t \in S$ 1,∧-elim
3. $[\![[w \bullet A]]\!]$ 1, ∧-elim
4. $(S' \subseteq S) \Rightarrow (t' - t \in S' \Rightarrow t' - t \in S)$ lemma (sets)
5. $S' \subseteq S$ (given)
6. $t' - t \in S' \Rightarrow t' - t \in S$ 5,4,MP
7. $t' - t \in S'$ 2,6,MP
8. $[\![[w \bullet A]]\!] \wedge t' - t \in S'$ 3,7,∧-intro □

Proof of (combine). Trivial by lemma $t' - t \in S \wedge t' - t \in T \Rightarrow t' - t \in (S \cup T)$ □

Proof of (choose). Holds due to lemma $[\![[w \bullet \mathscr{A}_j]]\!] \Rightarrow \bigvee_{i\in I} (g_i \wedge [\![[w \bullet \mathscr{A}_i]]\!])$ □

Proof of (nd). Trivial by distribution of conjunction. □

Proof of (basis) is by induction.

base case:

$$[w_0 : w : \exists \tilde{m}_i(\bigwedge_{i\in I} \Phi_i[m_{i-1}/t][m_i/t']) \wedge stable(w_0, m_n, t')[t/m_0]$$
$$= [w_0 \bullet stable(w_0, t, t')$$

because $n = 0$ and so $m_n = m_0$ which is replaced by $t = \mu_0 \mathscr{A}$

(inductive step).

We assert that $(\mu_n \mathscr{A}); \mathscr{A} = \mu_{n+1} \mathscr{A}$ and therefore:

$$w : \exists m_{max(i)}(\exists \tilde{m}_i(\bigwedge_{i\in I-max(I)} \Phi_i[m_i/t'][t/m_0])$$
$$\wedge stable(w_0, m_{n'}, t'))[m/t'] \wedge \Phi_{max(I)}[m/t])$$
$$\Rightarrow \exists \tilde{m}_i(\bigwedge_{i\in I-max(i)} \Phi_i[m_i/t'][t/m_0])$$
$$\wedge\ stable(w_0, m_n, t') \text{ where } n' = max(I)$$

Received July 1993
Accepted in revised form November 1995 by T. S. E. Maibaum