A CSP Approach To Action Systems

Michael J. Butler

Wolfson College

Thesis submitted for the degree of Doctor of Philosophy
at the University of Oxford

Michaelmas Term, 1992

A CSP Approach To Action Systems

Michael J. Butler, Wolfson College

Thesis submitted for the degree of Doctor of Philosophy
at the University of Oxford, Michaelmas Term, 1992

Abstract

The *communicating sequential processes* (CSP) formalism, introduced by Hoare [Hoa85], is an event-based approach to distributed computing. The action-system formalism, introduced by Back & Kurki-Suonio [BKS83], is a state-based approach to distributed computing. Using weakest-precondition formulae, Morgan [Mor90a] has defined a correspondence between action systems and the failures-divergences model for CSP. *Simulation* is a proof technique for showing refinement of action systems. Using the correspondence of [Mor90a], Woodcock & Morgan [WM90] have shown that simulation is sound and complete in the CSP failures-divergences model.

In this thesis, Morgan's correspondence is extended to the CSP infinite-traces model [Ros88] in order to deal more properly with unbounded nondeterminism. It is shown that simulation is sound in the infinite-traces model, though completeness is lost in certain cases.

The new correspondence is then extended to include a notion of internal action. This allows the definition of a hiding operator for action systems that is shown to correspond to the CSP hiding operator. Rules for simulation steps involving internal actions are developed.

A parallel operator for action systems is defined, in which interaction is based on synchronisation over shared actions. This operator is shown to correspond to the CSP parallel operator.

The correspondence between action systems and CSP is extended again so that actions may have input and output parameters. This allows parallel action-systems to pass values on synchronisation.

The original motivation for the work described in this thesis was the use of the action system formalism in the development of telecommunications systems, where interaction is often based on synchronised value-passing. The techniques developed here are applied to a series of case studies involving telecommunications-type systems. The techniques are used to refine and decompose abstract specifications of these systems into parallel sub-systems that interact via synchronised value-passing.

Acknowledgements

I am indebted to my supervisor, Carroll Morgan. Apart from providing some of the foundations, he was a constant source of encouragement, and often forced me to simplify various problems so that I reached better solutions more easily. I was also influenced, to my advantage, by the ethos of the Programming Research Group under Tony Hoare's leadership.

The work would not have been possible without the financial support of Broadcom Ltd, Dublin. Thanks especially to Fiona Williams and David Kennedy of Broadcom for their faith, and to colleagues in the RACE ARISE project for their encouragement. Thanks to Wolfson College for the use of a study carrel during my final year.

Thanks to various friends in Oxford and to my parents and brothers for their support. Finally, thanks to Susan for love and companionship.

Contents

Chapter 1

Introduction

The telecommunications systems that originally motivated the work described in this thesis are forms of *reactive* systems. Reactive systems maintain an on-going interaction with their environment and may consist of parallel subsystems [Pnu86]. In this chapter, we look at some existing formalisms for modelling and reasoning about reactive systems. We shall be interested in how these formalisms deal with specification, refinement, and composition of reactive systems.

The formalisms may be classified as *state-based* or *event-based* approaches. In state-based approaches, the behaviour of a system is described in terms of observations about its state. In event-based approaches, the state is not observed only the events that cause the state changes.

Section 1.1 looks at communicating processes, which are an event-based approach to reactive systems. Section 1.2 looks at a state-based approach to sequential programs. Section 1.3 looks at some state-based approaches to reactive systems. Section 1.4 looks at ways in which the state-based and event-based approaches may be linked. Finally, Section 1.5 gives an overview of the remaining chapters.

1.1 Communicating Processes

Well-known event-based approaches to concurrency include Hoare's *communicating sequential processes* (CSP) [Hoa85], Milner's *calculus for communicating systems* (CCS) [Mil89], and Bergstra & Klop's *algebra for communicating processes* (ACP) [BK85]. In each of these approaches, a process communicates with its environment by engaging in atomic events, and the behaviour of a process is defined in terms of the temporal ordering of events. The use of algebraic expressions and laws to specify and reason about process behaviour is common to the three approaches, though there are some differences between them. Here we present an introduction to CSP, then mention some differences between it and CCS and ACP.

1.1.1 CSP Algebra

The set of events in which a process $\mathcal{P}$ can engage is called its *alphabet*, written $\alpha\mathcal{P}$. The behaviour of $\mathcal{P}$ may be specified by writing $\mathcal{P} \mathrel{\widehat=} E$, where E is an algebraic expression constructed from elements of $\alpha\mathcal{P}$, *basic* processes, and CSP *operators*. An example of a basic process is the process $STOP$ that refuses to engage in any event.

Sequencing of events is described by the prefix operator ($\rightarrow$): the expression $a \rightarrow \mathcal{P}$ describes the process that engages in the event a and then behaves as process $\mathcal{P}$. External choice of behaviour is described by the choice operator ($[]$): $\mathcal{P} [] \mathcal{Q}$ represents the process that offers the choice to the environment between behaving as $\mathcal{P}$ or as $\mathcal{Q}$. CSP also has a separate nondeterministic choice operator ($\sqcap$): $\mathcal{P} \sqcap \mathcal{Q}$ represents the process that internally chooses between behaving as $\mathcal{P}$ or as $\mathcal{Q}$, leaving the environment with no control over that choice.

The parallel composition of two processes $\mathcal{P}$ and $\mathcal{Q}$ is expressed $\mathcal{P} \parallel \mathcal{Q}$. Here $\mathcal{P}$ and $\mathcal{Q}$ interact by synchronising over common events in $\alpha\mathcal{P} \cap \alpha\mathcal{Q}$, while events not in $\alpha\mathcal{P} \cap \alpha\mathcal{Q}$ can occur independently. An event common to both $\mathcal{P}$ and $\mathcal{Q}$ becomes a single event in $\mathcal{P} \parallel \mathcal{Q}$ and can be offered by $\mathcal{P} \parallel \mathcal{Q}$ only when both $\mathcal{P}$ and $\mathcal{Q}$ are prepared to offer it. As an example of the algebra, such synchronisation is represented in the following law:

$$(a \rightarrow \mathcal{P}) \parallel (a \rightarrow \mathcal{Q}) \quad = \quad a \rightarrow (\mathcal{P} \parallel \mathcal{Q}).$$

Independent occurrence of events is represented in the following algebraic law (assume $b \notin \alpha\mathcal{P}$, $a \notin \alpha\mathcal{Q}$):

$$\begin{aligned} (a \rightarrow \mathcal{P}) \parallel (b \rightarrow \mathcal{Q}) \quad = \quad & a \rightarrow (\mathcal{P} \parallel (b \rightarrow \mathcal{Q})) \\ & [] \;\; b \rightarrow ((a \rightarrow \mathcal{P}) \parallel \mathcal{Q}). \end{aligned}$$

Infinitary behaviour can be described by a recursively defined expression, written $(\mu\, X \bullet E(X))$, where $E(X)$ is some expression containing X. For example, the expression $(\mu\, X \bullet a \rightarrow X)$ describes the process that continually engages in the a-event. We sometimes write $\mathcal{P} \mathrel{\widehat=} E(\mathcal{P})$ for $\mathcal{P} \mathrel{\widehat=} (\mu\, X \bullet E(X))$.

When composing processes using the parallel operator it is convenient to be able to hide the interaction between them from the environment. This is achieved using the CSP hiding operator ($\setminus$): if $C \subseteq \alpha\mathcal{P}$, then $\mathcal{P} \setminus C$ describes the process that behaves as $\mathcal{P}$ but with each event in C hidden. The effect of hiding is represented in the following algebraic laws:

$$\begin{aligned}(a \to \mathcal{P}) \setminus C &= a \to (\mathcal{P} \setminus C) && \text{if } a \notin C \\ (c \to \mathcal{P}) \setminus C &= \mathcal{P} \setminus C && \text{if } c \in C.\end{aligned}$$

Hiding possibly infinite behaviour causes a process to *diverge*, i.e. behave chaotically. The process that diverges immediately is called *CHAOS*, and we have, for example, that

$$(\mu X \bullet a \to X) \setminus \{a\} \quad = \quad \mathit{CHAOS}.$$

1.1.2 CSP Semantics

The simplest semantic model for CSP is the *traces* model in which process behaviour is modelled by a non-empty, prefix-closed set of event-traces. For example, the process, with alphabet $\{a, b\}$, that performs an a-event, then a b-event, and then stops is modelled by the set

$$\{\ \langle\rangle,\ \langle a\rangle,\ \langle a, b\rangle\ \}.$$

However, the traces model does not distinguish internal and external choice, nor does it model divergence. In the *failures-divergences* model, a process with alphabet A is modelled by a set of failures F and a set of divergences D. A failure is a pair of the form (s, X), where $s \in A^*$ (the set of finite sequences of elements of A) is an event-trace and $X \subseteq A$ is a *refusal* set. If (s, X) is in F, then after engaging in the sequence of events s, a process may refuse all events X. For example, the process, with alphabet $\{a, b\}$, that nondeterministically offers an a-event or a b-event then stops is modelled by the failure set

$$\begin{aligned}&\{\ (\langle\rangle, \{\}),\ (\langle\rangle, \{a\}),\ (\langle\rangle, \{b\})\ \}\ \cup \\ &\{\ \langle a\rangle, X \mid X \subseteq \{a, b\}\ \}\ \cup\ \{\ \langle b\rangle, X \mid X \subseteq \{a, b\}\ \}.\end{aligned}$$

A divergence is simply a finite event-trace and $s \in D$ means that, after engaging in s, a process may diverge.

A CSP process model is then a tuple of the form (A, F, D). To be a proper model of a CSP process, F and D must satisfy seven well-formedness conditions **C0–C6** listed in [Hoa85, p.103]. For example, **C1** says that the traces should be prefix-closed: for $s, t \in A^*$, $X \subseteq A$,

$$(st, X) \in F \quad \Rightarrow \quad (s, \{\}) \in F.$$

Process models satisfying **C0–C6** constitute the (failures-divergences) semantic domain for CSP.

The semantics of an expression $\mathcal{P}$ is written $(\alpha\mathcal{P}, \mathcal{F}[\![\mathcal{P}]\!], \mathcal{D}[\![\mathcal{P}]\!])$, or $[\![\mathcal{P}]\!]$ for short. Corresponding to the algebraic operators there are functions on the semantic domain, and $[\![\mathcal{P}]\!]$ is defined by structural induction using these. For example, assuming '$\backslash$' is the semantic function for hiding,

$$[\![\mathcal{P} \setminus C]\!] \;\;\widehat{=}\;\; [\![\mathcal{P}]\!] \setminus C.$$

The semantic function is used to justify the algebraic laws: for expressions $\mathcal{P}$, $\mathcal{Q}$,

$$\mathcal{P} = \mathcal{Q} \quad \text{iff} \quad [\![\mathcal{P}]\!] = [\![\mathcal{Q}]\!].$$

1.1.3 CSP Refinement

The CSP algebraic laws can be used to show that a process $\mathcal{P}'$ is a valid implementation of a process $\mathcal{P}$. For example, suppose

$$\mathcal{P} \;\;\widehat{=}\;\; (a \to b \to c \to \mathcal{P}) \,[\!]\, (b \to a \to c \to \mathcal{P})$$

$$\mathcal{Q} \;\;\widehat{=}\;\; (a \to c \to \mathcal{Q}) \qquad\qquad \mathcal{Q}' \;\;\widehat{=}\;\; (b \to c \to \mathcal{Q}').$$

Using the algebraic laws of CSP it can be shown that

$$\mathcal{P} \;\;=\;\; \mathcal{Q} \parallel \mathcal{Q}',$$

i.e. $\mathcal{Q} \parallel \mathcal{Q}'$ is a valid implementation of $\mathcal{P}$.

More general than equality, however, is the CSP notion of *refinement*: a process model $\Theta = (A, F, D)$ is refined by $\Theta' = (A, F', D')$, written $\Theta \sqsubseteq \Theta'$, if

$$F \supseteq F' \quad \text{and} \quad D \supseteq D'.$$

Note that the alphabet of Θ and Θ' must be the same. For processes $\mathcal{P}$ and $\mathcal{P}'$, we write $\mathcal{P} \sqsubseteq \mathcal{P}'$ if $[\![\mathcal{P}]\!] \sqsubseteq [\![\mathcal{P}']\!]$. This means that any behaviour exhibited by $\mathcal{P}'$ may be exhibited by $\mathcal{P}$, and so $\mathcal{P}'$ is a valid implementation of $\mathcal{P}$. Clearly, the CSP refinement ordering is transitive.

As well as justifying the algebraic laws, the semantic model may be used to write specifications for CSP processes. A trace-specification is a condition on a variable tr, representing the traces of a process. For example, consider the specification

$$S(tr) \;\;\widehat{=}\;\; tr \leq \langle a, b, a, b, \ldots \rangle.$$

This is satisfied by any process that continually performs an a-event then a b-event in sequence. Similarly a trace-refusal specification $S(tr, X)$ can be used to describe process behaviour in terms of traces and refusals. For specification $S(tr, X)$ on alphabet A, let

$$\Theta \;\widehat{=}\; (A,\ \{\ tr, X \mid S(tr, X)\ \},\ \{\}).$$

Then $\mathcal{P}$ satisfies the specification $S(tr, X)$ if $\Theta \sqsubseteq [\![\mathcal{P}]\!]$. Hoare [Hoa85] presents a set of proof rules that may be used to show when a process satisfies its specification. These rules are defined on the algebraic structure of $\mathcal{P}$ and describe the specifications satisfied by $\mathcal{P}$ in terms of those satisfied by its components.

1.1.4 CCS and ACP

The algebraic notations of CCS [Mil89] and ACP [BK85] are similar in principle to that of CSP. One significant difference, however, is their treatment of hiding. Both CCS and ACP have a special event, denoted τ, representing the unobservable communication. If an event is to be hidden it is simply renamed to τ. *Bisimulation* ($\approx$) is then used to compare processes: $\mathcal{P} \approx \mathcal{Q}$ if any observable behaviour exhibited by $\mathcal{P}$ may be exhibited by $\mathcal{Q}$, and vice versa. For example, CCS[1] has the following law:

$$a.\tau.\mathcal{P} \;\approx\; a.\mathcal{P}\ .$$

Unlike in CSP, hiding of infinite behaviour in CCS and ACP does not cause a process to behave chaotically. For example, while in CSP we have

$$(\mu\, X \bullet c \rightarrow X) \setminus \{c\} \;=\; \mathit{CHAOS},$$

in CCS we have

$$(\mu\, X \bullet \tau.X) \;\approx\; \mathit{NIL}\ .$$

ACP treats hiding in a manner similar to CCS.

Another difference is that neither CCS nor ACP have a separate operator for internal choice. Instead, the τ-event may be used to describe internal choice. For example, the internal choice between $\mathcal{P}$ and $\mathcal{Q}$ may be expressed in CCS as

$$\tau.\mathcal{P} + \tau.\mathcal{Q}\ .$$

Here, the environment has no control over which τ-event will occur. Bisimulation distinguishes internal and external choice since

$$\tau.\mathcal{P} + \tau.\mathcal{Q} \;\not\approx\; \mathcal{P} + \mathcal{Q}\ .$$

[1] CCS uses '.' for CSP's '$\rightarrow$', '+' for '$[\,]$', and 'NIL' for 'STOP'.

The semantics of an expression in CCS is defined by a set of *transition* rules. A process $\mathcal{P}$ makes a transition to a process $\mathcal{Q}$ by engaging in some event. The transition rules are defined on the algebraic structure of process expressions and describe all possible transitions of a process. Bisimulation between processes is defined by comparing their observable transition sequences.

The semantics of an expression in ACP is defined by mapping it to a *process graph*. A process graph is directed outwards from a root node and each edge is labelled by an event. Corresponding to each algebraic operator, there is a function on process graphs. Equivalence of process graphs is based on bisimulation.

1.2 Sequential Programs

We briefly consider Dijkstra's guarded command-language and its associated weakest-precondition calculus [Dij76]. Programs in the guarded command-language act on variables, and are sequential and intended to be terminating. The language includes assignment to variables, sequential composition of statements (programs), IF-statements, and DO-loops. The behaviour of a program is specified in terms of a condition on the initial values of the program variables (precondition) and a condition on the final values of the program variables (postcondition).

Programs may be modelled as relations between initial states and final states. However, dealing with non-termination in this model requires the introduction of a bottom state $\perp$ which complicates the model. The weakest precondition semantics introduced by Dijkstra deals with termination in a simpler fashion. For statement S and postcondition $post$, $wp(S, post)$ (weakest precondition of S w.r.t. $post$) represents all those initial states from which S is guaranteed to terminate in a state satisfying $post$. The weakest-precondition semantics of assignment is defined in terms of variable substitution, while the weakest-precondition semantics of compound statements is defined in terms of its components.

S satisfies a specification $(pre, post)$ if

$$pre \quad \Rightarrow \quad wp(S, post).$$

Dijkstra [Dij76] presents a calculus for verifying that programs satisfy their specifications.

Various authors have extended Dijkstra's calculus to develop an approach referred to as the *refinement calculus* [Bac80, Bac90a, Mor90b, MRG88, Mrr87]. In the refinement calculus, specifications are given a weakest-precondition semantics and are themselves regarded as statements. A statement S' is said to refine S if S' satisfies any specification satisfied by S. Program development then involves stepwise refinement of an abstract program into an executable program by application of a series of correctness preserving transformations.

Data refinement involves replacing abstract variables with concrete variables that are more easily implemented [Bac90a, GM88, HHS86, Mrr87]. A representation relation *Rep* relating the abstract variables a and concrete variables c is used to replace abstract statements with concrete statements. Let $(\textbf{var } x \bullet S)$ be the statement S with the variable x localised. Then we say that "S is data-refined by S' under *Rep*" if *Rep* can be used to show that $(\textbf{var } a \bullet S)$ is refined by $(\textbf{var } c \bullet S')$.

The VDM [Jon86] and Z [Spi89] methods are closely related to the refinement-calculus approach. Both provide rich specification notations, based on predicate calculus and set theory, that allow abstract programs to be specified in a clear, structured manner. They also provide refinement rules for program development.

1.3 State-Based Reactive Systems

Typically, a state-based reactive system is modelled by a *transition system* [Lam89, Pnu86]. In this section, we look at transition systems, and also look at the related *action systems*.

1.3.1 Transition Systems

A transition system T consists of a set of states S, an initialisation $I \subseteq S$, and a set of transition relations R, where each $r \in R$ is a relation on S. T starts in some initial state $s_0 \in I$. Then some transition $r \in R$ such that $s_0 \in dom(r)$, is selected and T enters some state s_1, where $(s_0, s_1) \in r$. This process of selection and execution continues until a terminating state is reached, i.e. a state not in the domain of any transition relation.

A *state-trace* u of a transition system $T = (S, I, R)$ is a finite or infinite sequence of states from S,

$$u \quad = \quad s_0 \to s_1 \to \cdots$$

where $s_0 \in I$, and for each $i \geq 0$ there is some transition $r_i \in R$ such that $(s_i, s_{i+1}) \in r_i$. If u is finite then the last element of u is a terminating state. A state-trace u represents a possible behaviour of T, either terminating (u is finite) of non-terminating (u is infinite). Alternatively, a terminating behaviour may be represented by an infinite sequence with constant tail. We denote the set of state-traces of a transition system T by $tr(T)$.

The behaviour of a transition system may be nondeterministic in two ways: the selection of a transition may be nondeterministic, and the effect of executing a transition may be nondeterministic. However, the state-trace model does not distinguish these forms of nondeterminism: consider the transition systems *T1* and *T2* as follows:

$$\begin{array}{lcl} T1 & = & (\ \{0,1\},\ \{0\},\ \{r_0 = \{0 \mapsto 0\}, r_1 = \{0 \mapsto 1\}\}\ ) \\ T2 & = & (\ \{0,1\},\ \{0\},\ \{r_2 = \{0 \mapsto 0, 0 \mapsto 1\}\}\ ). \end{array}$$

T1 has two deterministic transitions (r_0 and r_1), while *T2* has a single nondeterministic transition (r_2), yet we can show that $tr(T1) = tr(T2)$.

A transition system may be specified using *properties* as shown by Lamport [Lam89] and Pnueli [Pnu86]. A property is a set of state-traces, and a transition system T is said to satisfy a property P if $tr(T) \subseteq P$. Usually a property is specified as the intersection $M \cap L$ of a *safety* property M and a *liveness* property L. A safety property asserts that "something bad never happens", or more precisely: if a partial state-trace violates a safety property, then there is no extension of that trace satisfying the property. A liveness property asserts that "something good eventually happens", or more precisely: for any partial state-trace, there is always some extension of that trace satisfying the liveness property.

Rather than specifying properties as sets of state-traces, they can be specified using *temporal logic* formulae [Lam83, Pnu86]. For example, if p is a temporal logic formula, then $\Box p$ (invariant p) specifies the safety property

$$\{\ u \in S^\omega \mid (\forall\, i \geq 0 \bullet u[i..]\ \mathit{satisfies}\ p)\ \},$$

while $\Diamond p$ (eventually p) specifies the liveness property

$$\{\ u \in S^\omega \mid (\exists\, i \geq 0 \bullet u[i..]\ \mathit{satisfies}\ p)\ \}.$$

Here S^ω is the set of infinite traces of elements of S, and $u[i..]$ is the trace starting at the ith position of u (note $u = u[0..]$).

Fairness requirements can be used to constrain the selection of transitions [Fra86, Pnu86]. These determine the frequency with which enabled transitions should be executed. To model fairness requirements, the transitions are included in state-traces as follows:

$$u \quad = \quad s_0 \xrightarrow{r_0} s_1 \xrightarrow{r_1} \cdots \ .$$

An example of a fairness requirement is *weak* fairness. For $R' \subseteq R$, R' is said to be enabled if some transition from R' is enabled. Then a trace is said to be *weakly* fair with respect to R' if whenever R' is continuously enabled from some position i on the trace, then some transition from R' is executed at some position after i. In contrast, a trace is said to be *strongly* fair with respect to R' if whenever R' is infinitely often enabled, then infinitely often some transition from R' is executed. Fairness is a form of liveness property since any partial trace can always be extended in such a way that it satisfies a fairness requirement.

1.3.2 Action Systems

We shall use the term *action system* to refer to a transition system in which the state space is represented by one or more program variables, and the initialisation and transitions are represented by statements in Dijkstra's guarded command-language. Typical examples include Back's action system formalism [BKS83], and Chandy & Misra's UNITY [CM88].

In Back's action systems, an action (transition) is a statement of the form

$$g \to com,$$

where g is a guard (condition on the state variables) and com is a command (program statement). An action is enabled in any state satisfying g. An action system proceeds by firstly executing the initialisation. Then, repeatedly, an enabled action is selected and executed. An action system terminates if it reaches a state in which no action is enabled. In addition, an action system aborts if either its initialisation or one of its actions aborts.

If it is intended that an action system, with initialisation I and action set $A = \{A_0, A_1, \ldots\}$ should terminate, then it can be modelled simply as the statement

$$I \; ; \; \mathbf{do} \; A_0 \; [\!] \; A_1 \; [\!] \cdots \; \mathbf{od}.$$

If an action system is intended to be reactive, then it is modelled by a set of state-traces as described in the previous subsection. Aborting behaviour is modelled by a trace of the form

$$u \quad = \quad s_0 \to s_1 \to \cdots \to s_n \to \perp \quad .$$

Usually only weak fairness with respect to the full set of actions is assumed (i.e. an action system terminates only if no action is enabled), though stronger fairness requirements can be specified [Bac90b, BKS90].

In Back's formalism, an action system is specified explicitly with an initialisation command and a set of guarded commands, rather than being specified by a set of properties. Examples specified using the formalism include a mutual-exclusion algorithm [Bac92b], a Gaussian-elimination algorithm [BS89], and a telephone exchange [KSK88].

A UNITY program [CM88] is an action system in which each action is deterministic, non-aborting, and always enabled[2]. In addition, there is a fairness requirement on a UNITY program which says that each action must be executed infinitely often.

A UNITY program may be specified explicitly as an initialisation and a set of actions, or alternatively, may be specified using UNITY *logic*. UNITY logic is a form of temporal logic, though, rather than characterising a set of state-traces, a formula in UNITY logic constrains the actions of a program. For example, if p and q are conditions on state variables, then the formula p *unless* q declares that each action A of a program must satisfy the following:

$$p \wedge \neg\, q \quad \Rightarrow \quad wp(A,\; p \vee q\;).$$

Thus p *unless* q is a safety constraint on a program that says p remains true at least until q becomes true. The formula p *ensures* q is used to specify liveness constraints: this declares that some action of a program must establish q from p, and because of the UNITY fairness requirement, an action that establishes q will eventually be executed.

A specification in UNITY consists of a set of logic formulae, and a program satisfies a specification if its actions satisfy each of the formulae. Examples specified in UNITY logic include the Dining-Philosophers problem [CM88], various sorting and searching algorithms [CM88], and a distributed Electronic Funds-Transfer system [Sta88].

1.3.3 Internal State

When developing systems it is convenient to be able to make part of the state unobservable. Abadi & Lamport [AL88] use the notion of external and internal state to achieve this in transition systems. Elements of the state-space consist of pairs of the form (e, h), where e is the external component and h is the internal component. A state-trace will have the form

$$(e_0, h_0) \rightarrow (e_1, h_1) \rightarrow \cdots\ .$$

[2] UNITY does allow conditional assignment, but the conditional assignment '$x := E,$ **if** c' is the same as the statement '**if** $c \rightarrow x := E \;[\!]\; \neg\, c \rightarrow$ **skip fi**' which is always enabled.

Since only the external behaviour of a transition system T is of interest, we assume that $tr_e(T)$ strips away the internal components from state-traces, so a state-trace in $tr_e(T)$ looks like

$$e_0 \to e_1 \to \cdots \ .$$

Transitions in which only the internal state is changed, e.g.

$$(e_0, h_0) \to (e_0, h_1) \to (e_0, h_2) \to \cdots \ ,$$

result in *stuttered* (repetition of) external behaviour. Finite stuttering of external state is not regarded as significant, and two external state-traces are equivalent if they differ only by finite amounts of stuttering. So we further assume that $tr_e(T)$ is the closure under stuttering of the external state-traces of T. The specification of a system is then a property P on the external state, that should be closed under stuttering, and T is said to satisfy P if $tr_e(T) \subseteq P$.

1.3.4 Refinement

A property P' is said to refine a property P if $P' \subseteq P$. If properties are written using temporal logic, then temporal proof-rules, such as those developed by Manna & Pnueli [MP84], can be used prove the correctness of such refinements. Showing that a transition system T satisfies a property P is also a form of refinement, and when properties are separated into safety and liveness, then satisfaction of a safety property can be proven using an invariance argument, while satisfaction of a liveness property can be proven using a well-foundedness argument [AS87]. Chandy & Misra [CM88] have also developed proof rules for verifying that a UNITY logic specification is refined (satisfied) by another, and that a specification is satisfied by a program.

A transition system T' is said to refine a transition system T if $tr_e(T') \subseteq tr_e(T)$. Refinement mappings, as developed by Abadi & Lamport [AL88], can be used to show refinement between transition systems. We shall now look at the refinement mappings of Abadi & Lamport, and then at Back's approach to action-system refinement.

Refinement mappings are defined for transition systems that are more general than those already presented: a transition system has components (S, I, R) as before, as well as an extra component L, which is a liveness property. The full state-traces of a transition system $T = (S, I, R, L)$ are defined as $tr(S, I, R) \cap L$. When showing that $T = (S, I, R, L)$ is refined by $T' = (S', I', R', L')$, the external state of T will be the same as that of T', though their internal states may be different. Let $(s, t) \in R$ be short for $(\exists\, r \in R \bullet (s, t) \in r)$. Assume f is a function from S' to S that preserves the externally visible component. Then f is a refinement mapping from T' to T if it satisfies:

1. $f(I') \subseteq I$ (f takes initial states to corresponding initial states)

2. $(s,t) \in R' \Rightarrow (f(s), f(t)) \in R \lor f(s) = f(t)$ (a transition in T' either corresponds to a transition in T, or causes a stutter)

3. $f(tr(T')) \subseteq L$ (the combined traces of T' correspond to some subset of L, the liveness property of T).

Conditions 1 and 2 ensure that T' satisfies the safety properties of T, and can be checked by reasoning about states and individual transitions. Condition 3 ensures that T' satisfies the liveness properties of T and is more difficult to check since it involves state-traces. It is usually checked using a well-foundedness argument. Abadi & Lamport show that if there is a refinement mapping f from T' to T, then $tr_e(T') \subseteq tr_e(T)$, i.e. the refinement-mapping technique is *sound*.

Abadi & Lamport also introduce two correctness-preserving transformations on transition systems: addition of a *history* variable, and addition of a *prophecy* variable. They show that if $tr_e(T') \subseteq tr_e(T)$, then this can always be shown using a combination of refinement mapping, history variable, and prophecy variable, i.e. the refinement-mapping technique is *complete*[3].

In Back's formalism, the technique of data refinement for sequential programs is applied to action systems to show that one action system is refined by another [Bac90b, Bac92b]. The state variables of an action system are partitioned into two sets, internal and external. To show that action system T is refined by T', the internal variables of T are regarded as abstract variables, while the internal variables of T' are regarded as concrete variables. Back's first rule for action system refinement doesn't allow for stuttering, so there must be a direct correspondence between actions in T' and in T. If Rep is a representation relation on variables of $T' = (I', A')$ and $T = (I, A)$, then T' is a refinement of T under Rep if

1. I is data refined by I' under Rep

2. A is data refined by A' under Rep

3. $Rep \land gd(A) \Rightarrow gd(A')$.

Here A is short for the nondeterministic composition of all the actions in the set A, and $gd(A)$ is the guard of A.

[3] All completeness results referred to here are valid only for systems that are boundedly nondeterministic. We shall discuss this in Section 3.6

Conditions 1 and 2 are similar to Conditions 1 and 2 of Abadi & Lamport's refinement-mapping rule (without allowing for stuttering) which ensure preservation of safety properties. Condition 3 ensures that T' can only terminate if T terminates, so that T' satisfies the liveness properties of T. One difference between Back and Lamport & Abadi, however, is that Rep is a relation between the concrete and abstract state-spaces rather than a function. This makes the use of history variables unnecessary.

Back also presents a more general refinement rule that allows for stuttering actions in the concrete action systems. As well as the main actions A', the concrete action system T' is allowed to contain a set of auxiliary actions H' that cause stuttering steps. Now, $T' = (I', A', H')$ is a refinement of $T = (I, A)$ if Conditions 1 and 2 hold as before and the following conditions also hold:

3'. $Rep \wedge gd(A) \;\Rightarrow\; gd(A') \vee gd(H')$

4 . **skip** is data refined by H' under Rep

5 . Rep implies termination of **do** H' **od**.

Condition 4 ensures that the auxiliary actions of T' cause only stuttering transitions, while Condition 5 ensures that T' introduces only finite amounts of stuttering. Back shows that action-system refinement is sound in the state-traces model [Bac90b].

1.3.5 Parallel Composition

Several authors have developed ways of composing transition systems into hierarchies of interacting systems. Component systems are intended to represent concurrently-executing systems and, typically, interaction between systems is based on shared state or variables.

In Back's formalism, two action systems are composed by simply joining together their respective action sets [Bac90b, Bac92b]: assume

$$T1 \;=\; (x := x_0, A1), \qquad\qquad T2 \;=\; (y := y_0, A2),$$

where x and y are the internal variables of $T1$ and $T2$ respectively. Then the action system $T1 \parallel T2$, representing their parallel composition, is defined as:

$$(x, y := x_0, y_0, A1 \cup A2).$$

$T1$ and $T2$ interact by sharing external variables.

A *decomposition* step involves showing that T is refined by $T1 \parallel T2$, and can be achieved by constructing $T1 \parallel T2$ and then showing that T is refined by $T1 \parallel T2$ under some Rep.

Back shows that under certain conditions parallel composition is monotonic w.r.t. data refinement, i.e. if $T1$ is refined by $T1'$ under Rep, then

$$T1 \parallel T2 \text{ is refined by } T1' \parallel T2 \text{ under } Rep.$$

Such monotonicity holds when the representation relation Rep is invariant in the action system $T2$. This condition is necessary since Rep may make assumptions about variables shared by $T1$ and $T2$, and these assumptions must be preserved by $T2$.

The *union* operator for UNITY programs [CM88] is similar to parallel composition in Back's formalism. Chandy & Misra have developed compositional proof-rules for proving decomposition steps correct. These allow one to reason about assertions on a composite system $T1 \parallel T2$ in terms of assertions on $T1$ and $T2$. For example, they have the following rule about *unless*:

$$p \textit{ unless } q \text{ in } T1 \parallel T2 \quad \text{iff} \quad p \textit{ unless } q \text{ in } T1 \;\wedge\; p \textit{ unless } q \text{ in } T2.$$

Abadi & Lamport [AL90] approach composition in terms of properties. The composition of properties $P1$ and $P2$ is defined simply as $P1 \cap P2$. A specification P has the form $E \Rightarrow M \cap L$, where E represents assumptions about the environment (e.g. the environment only reads a certain variable) and M and L are safety and liveness properties respectively. If $P1 \;\widehat{=}\; E1 \Rightarrow M1 \cap L1$ and $P2 \;\widehat{=}\; E2 \Rightarrow M2 \cap L2$ are to be composed, then $P1$ should satisfy $E2$ and $P2$ should satisfy $E1$. Abadi & Lamport also describe some compositional proof-rules for proving the correctness of decomposition steps.

Owicki & Gries [OG76] have developed proof rules for reasoning about parallel programs in a Dijkstra-like language that share variables. These rules involve reasoning about preconditions and postconditions of statements in component programs, and soundness is based on the assumption that only reads and writes to shared variables are atomic. However, the Owicki-Gries method is not truly compositional in that component programs must be constructed before the parallel proof-rule can be applied. The *rely/guarantee* method of Jones [Jon83] is a compositional extension of the Owicki-Gries method that is used to reason about programs in the style of VDM. A specification has the form (P, R, G, Q), where P is a condition describing a set of states, and R, G, and Q are conditions on state-transitions (i.e. relations between before-states and after-states). The precondition P and rely condition R constitute assumptions about the environment, and in return, an implementation must satisfy the guarantee condition G and the postcondition Q when operated in an environment fulfilling the assumptions. Stirling [Sti88] has also developed a compositional extension of the Owicki-Gries method. Again, Stirling uses rely and guarantee conditions in specifications, though these are conditions on states rather than on state-transitions. Stølen [Stø91] extends the rely/guarantee method by introducing a *wait* condition that describes when a program may be safely blocked by its environment. This allows synchronisation between programs to be dealt with. Xu & He [XH91] also extend the rely/guarantee method and introduce a *run* condition to deal with synchronisation. The environment must establish the run condition when a program is blocked.

1.4 Linking Events and State

In this section, we look at some approaches in which the event-based and state-based views of reactive systems are linked. Each of these use transition systems, but primary importance is placed on the events (transition labels) rather than the state. We look at the work of various authors on a CSP approach to transition systems and action systems, then briefly at I/O-automata.

1.4.1 Transition Systems and CSP

A *labelled* transition-system is a transition system in which each transition has an associated label. A labelled transition system may be regarded as a CSP process by treating the transition labels as events. Josephs [Jos88] and He Jifeng [He89] have given a CSP semantics to labelled transition-systems. Here we look at the work of He Jifeng — the work of Josephs is similar though it doesn't deal with divergence.

Consider a transition system $T = (A, S, I, T_A)$, where A is a set of labels, S is a set of states, $I \subseteq S$ is an initialisation, and $T_A = \{\ T_a \mid a \in A\ \}$ is a set of labelled transitions on S. For finite label trace $t = \langle a_0, \ldots, a_n \rangle$, let T_t be the relational composition of $T_{a_0}, \ldots, T_{a_n}$:

$$T_t \ \ \widehat{=}\ \ T_{a_0};\ \ldots;\ T_{a_n}.$$

Also, let $T_{\langle\rangle}$ be the identity relation on S. Now, T can engage in the sequence of transitions labelled from t if T_t relates some initial state to some other state. So $t \in A^*$ is a *trace* of T if

$$(\exists\, i \in I, s \in S \bullet i(T_t)s).$$

When T is in state s, it can refuse the transition labelled a if $s \notin dom(T_a)$. So for trace $t \in A^*$ and refusal set $X \subseteq A$, (t, X) is a *failure* of T if

$$(\exists\, i \in I, s \in S \bullet i(T_t)s \wedge (\forall\, a \in X \bullet s \notin dom(T_a))).$$

The bottom state $\perp$ is introduced to represent the divergent state, and a divergence is any trace that can reach $\perp$. Also, the definition of failures is extended to ensure that the CSP well-formedness conditions **C0**–**C6** [Hoa85, p.130] are satisfied.

In order to show that transition system T is refined by T' in the CSP sense, He Jifeng uses *simulation* relations. Assume T and T' have the same label sets though possibly different state sets, S and S'. A *downwards*[4] simulation is a relation $d \in S \leftrightarrow S'$ satisfying:

1. $I;\ d \quad \supseteq \quad I'$
2. $T_a;\ d \quad \supseteq \quad d;\ T'_a$ $\qquad$ for each $a \in A$
3. $dom(T_a);\ d \quad \subseteq \quad dom(T'_a)$ $\qquad$ for each $a \in A$.

If there is a downwards simulation between T and T', then it is shown that T is refined by T' in the CSP sense, i.e.

$$failures(T) \supseteq failures(T') \quad \text{and} \quad divergences(T) \supseteq divergences(T').$$

Upwards[5] simulation is complementary to downwards simulation and involves a the use of a relation $u \in S' \leftrightarrow S$. He Jifeng shows that downwards and upwards simulation are jointly complete. The proof of completeness involves constructing a canonical form for T, denoted $T^\#$. It is shown, using a combination of downwards and upwards simulation, that T and $T^\#$ have exactly the same failures-divergences semantics. If the failures and divergences of T include those of T', then there is always a downwards simulation from $T^\#$ to T'. So using a combination of downwards and upwards simulation, it can be shown that T is refined by T'. Downwards and upwards simulation were introduced by He, Hoare & Sanders [HHS86] as a complete method for data refinement of sequential programs in the relational framework.

[4] Also called *forwards* simulation.

[5] Also called *backwards* simulation.

Olderog & Hoare [OH86] have shown how safety and liveness are represented in the CSP semantic model. The traces of a process represent safety, since they describe those event-traces that a process may safely engage in. Refusal-sets represent liveness, since they may describe what a process must not refuse to engage in. In light of this, a correspondence between He Jifeng's simulation, the refinement mappings of Abadi & Lamport [AL88], and Back's action-system refinement [Bac90b, Bac92b] can be seen: Conditions 1 and 2 of downwards simulation are sufficient to ensure trace inclusion (safety), while Condition 3 ensures that T' may only refuse an event if T refuses it (liveness).

He Jifeng defines operators on labelled transition-systems that are shown to correspond to the CSP operators on processes. For example, the hiding operator hides a transition labelled c to form $T \setminus \{c\}$. The definition of $T \setminus \{c\}$ is somewhat complicated, and involves composing the transitive closure of T_c with each of the remaining transitions, and calculating the divergences introduced by hiding c. The simulations described above do not allow for stuttering. However, based on his hiding operator for transition systems, He Jifeng has developed simulations that allow the concrete systems to have hidden transitions [He90].

The parallel operator composes $T1$ and $T2$ to form $T1 \parallel T2$. The state-space of $T1 \parallel T2$ is the Cartesian product of the state-spaces of $T1$ and $T2$. The transitions of $T1 \parallel T2$ are formed from those of $T1$ and $T2$ in such a way that $T1 \parallel T2$ can engage in transitions whose labels are common to $T1$ and $T2$ only when both $T1$ and $T2$ can engage in them. In this way, interaction between $T1$ and $T2$ is by synchronisation over shared events as in CSP.

1.4.2 Action Systems and CSP

Morgan [Mor90a] has defined failures-divergences semantics for (labelled) action systems. However, rather than defining them in terms of relations as in [He89, Jos88], Morgan defines them in terms of weakest-precondition formulae. Using weakest preconditions means that there is no need to introduce a special divergent state and so the definitions are less complicated.

Morgan & Woodcock [WM90] have formulated upwards and downwards simulation in terms of weakest preconditions and have shown them to be sound methods for showing that one action system is refined by another in the CSP sense. They have also shown that both simulation methods are jointly complete by constructing canonical forms for action systems in the manner of [He89, Jos88]. Morgan's CSP semantics for action systems, along with simulation, will be described in Chapter 2.

1.4.3 I/O-Automata

I/O-automata, introduced by Lynch & Tuttle [LT87], are labelled transition-systems with some fairness requirements. The labels of an I/O-automaton are partitioned into input events, output events, and internal events. Fairness requirements are usually specified by *justice sets*: a justice set is a set of transition labels, and a system should be weakly fair w.r.t. each justice set. The full behaviours of an I/O-automaton are of the form:

$$u \;=\; s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \;.$$

The event-traces of an I/O-automaton are formed by stripping the states and internal event labels from all its fair behaviours. Note that, unlike CSP, sets of event-traces are not necessarily prefix-closed. An I/O-automaton may be specified by a set of finite and infinite traces of input and output events, and refinement is defined in terms of trace inclusion.

The parallel composition operator for I/O-automata is similar to the one defined by He Jifeng: commonly-labelled actions are combined so that interaction is based on shared events.

There are a number of restrictions on I/O-automata however. Firstly, input events should always be enabled and, when using the parallel operator, output events should only be composed with input events. This is because I/O-automata are intended to model asynchronous communications-networks, and so, an output event should never be blocked by the receiving end. Secondly, fairness requirements should not constrain input events. This simplifies composition of fairness requirements, since only input events can be blocked by composition.

Lynch [Lyn90] has defined a form of refinement mapping for I/O-automata called a *multivalued-possibilities mapping*. This maps elements of the concrete state-space to subsets of the abstract state-space, and so is akin to using a simulation relation. Jonsson [Jss90] takes a similar approach to Lynch & Tuttle. Instead of using multivalued-possibilities mappings to show refinement, he uses downwards- and upwards-simulation relations. Thus he provides a complete method for checking refinement between I/O-automata.

1.5 Overview

Morgan's CSP semantics of action systems [Mor90a], mentioned in the previous section, forms the basis for the work presented in this thesis. Although simulation rules for CSP refinement of action systems are presented in [WM90], no means, for example, of hiding actions or of composing action systems are presented. Some of these shortcomings are addressed here.

Many of the results presented depend on properties of weakest preconditions. Thus, in Chapter 2, such properties are summarised, along with some extensions to Dijkstra's guarded-command language. Chapter 2 also presents Morgan's failures-divergences semantics of action systems, as well as simulation for action systems.

Chapter 4 describes a way of hiding actions. However, it will be shown that unbounded nondeterminism in actions means that our definition of action system hiding does not always correspond to the failures-divergences definition of CSP hiding. It was discovered that this problem could be overcome by extending the CSP semantics of action systems to the *infinite-traces* model for CSP. The infinite-traces model was introduced by Roscoe [Ros88] specifically to deal with unbounded nondeterminism in CSP. The infinite-traces model is described in Chapter 3, and the infinite-traces semantics of action systems is defined. Also, simulation is shown to be sound in the infinite-traces model, under certain conditions.

Then, in Chapter 4, hiding of actions is defined and shown to correspond to CSP hiding in the infinite-traces model. Action hiding is defined using a program construct for nondeterministic iteration. Since this construct is given a weakest-precondition semantics, it is not necessary to deal separately with the introduction of divergence, as is required in He Jifeng's relational approach [He89]. A simulation rule that turns out to be similar to Back's stuttering-refinement rule for action systems is also developed in Chapter 4.

Chapter 5 describes a parallel operator for action systems in which interaction is based on synchronisation over common actions. This operator is defined by firstly introducing a parallel operator for pairs of actions. Again, the action system operator is shown to correspond to CSP parallel composition in the infinite-traces model.

In Chapter 6, action systems are extended to deal with communication of values. This extension is based on the CSP notion of value communication presented in [Hoa85]. Individual actions are parameterised by input and output variables, representing input and output value communications. The parallel operator is extended to allow passing of values between action systems. Hiding and simulation are also extended.

Chapter 7 presents some case studies that show how the action system techniques developed may be used in practice. These case studies involve the specification, refinement, and decomposition to parallel sub-systems of some telecommunications examples. Since the refinement orderings we use are transitive, any refinements may be carried out in stepwise fashion. An advantage of defining the

CSP semantics of an action system in terms of weakest preconditions rather than relations is that it is readily linked to the refinement calculus. Chapter 7 demonstrates how some techniques of the refinement calculus may be used to specify action systems, and to verify refinement conditions between action systems.

Chapter 2

Preliminaries

This chapter presents some mathematical background and definitions needed in later chapters. It describes predicate transformers, some extensions to the guarded-command language, and Morgan's CSP semantics for action systems.

2.1 Predicates

Let Σ be some non-empty set of states. We interpret a *predicate* ϕ to be a subset of Σ and not, for instance a formula in a fixed langauge like first-order logic [Ham78]. Our view of predicates is found, for example, in [Nel89], and avoids the necessity of higher-order logic when quantifying over predicates: with our approach, the statement "for all predicates ϕ" is interpreted as "for all subsets ϕ of Σ".

Two predicates ϕ and ψ are *equivalent*, written $\phi \equiv \psi$, if ϕ and ψ are equal as sets. Predicate ϕ *entails* ψ, written $\phi \Rrightarrow \psi$, if ϕ is a subset of ψ. Entailment ($\Rrightarrow$) is an ordering on predicates with bottom element $\{\}$, written *false*, and top element Σ, written *true*. Non-emptiness of Σ means that *false* and *true* are distinct.

Let ϕ_i represent an i-indexed set of predicates. The least upper-bound and greatest lower-bound of the set of predicates ϕ_i are written respectively as

$$(\vee\ i \bullet \phi_i) \quad \text{and} \quad (\wedge\ i \bullet \phi_i).$$

Both $(\vee\ i \bullet \phi_i)$ and $(\wedge\ i \bullet \phi_i)$ exist: $(\vee\ i \bullet \phi_i)$ is the predicate containing all those states of Σ in at least one ϕ_i, while $(\wedge\ i \bullet \phi_i)$ is the predicate containing all those states of Σ in each ϕ_i. $(\vee\ i \bullet \phi_i)$ and $(\wedge\ i \bullet \phi_i)$ are also called disjunction and conjunction respectively. Note that the range of any disjunction or conjunction used in this thesis will always be bounded by some ordinal to ensure that the disjunction or conjunction is over a set of predicates.

We write $\phi_1 \vee \phi_2$ for $(\vee\ i \in \{1, 2\} \bullet \phi_i)$ and $\phi_1 \wedge \phi_2$ for $(\wedge\ i \in \{1, 2\} \bullet \phi_i)$. The *negation* of predicate ϕ, written $\neg\ \phi$ is simply $\Sigma - \phi$. We write $\phi \Rightarrow \psi$ for

$\neg\, \phi \vee \psi$.

It is assumed that Σ is some Cartesian product-space. Variables are the coordinate functions that project the state space onto its Cartesian factors. A predicate ϕ is *independent* of a variable x if any two states that differ in the x coordinate only are either both in ϕ or both not in ϕ [Nel89]. A variable x is said to be *free* in ϕ if ϕ is not independent of x.

$(\exists\, x \bullet \phi)$ is defined as [Nel89] the least predicate ψ satisfying

$$\psi \text{ is independent of } x \text{ and } \phi \Rrightarrow \psi.$$

$(\forall\, x \bullet \phi)$ is defined as [Nel89] the greatest predicate ψ satisfying

$$\psi \text{ is independent of } x \text{ and } \phi \Lleftarrow \psi.$$

Two variables have the same *type* if their corresponding Cartesian factors are the same set. For variables x, y of the same type, and predicate ϕ, $\phi[x\backslash y]$ represents ϕ with all free occurrences of x replaced by y. A state s satisfies $\phi[x\backslash y]$ if s' satisfies ϕ, where s' is a state that coincides with s in all coordinates except x, where it has the value $y(s)$ [Nel89]. For expression E, $\phi[x\backslash E]$ is short for $(\exists\, y \bullet\ y = E \wedge \phi[x\backslash y])$. Substitutions may be composed and distributed in the usual manner.

Standard laws of the predicate calculus [Ham78] may be used to that $\phi \equiv \psi$, or $\phi \Rrightarrow \psi$.

2.2 Predicate Transformers

A predicate transformer is a function from predicates to predicates. Variable substitution is an example of a predicate transformer: consider the function *sub*, where for any predicate ϕ,

$$sub(\phi) \quad \widehat{=} \quad \phi[x\backslash E].$$

Predicate transformer f is *strict* if $f(\mathit{false}) \equiv \mathit{false}$ and is *monotonic* if

$$\phi \Rrightarrow \psi \quad \text{implies} \quad f(\phi) \Rrightarrow f(\psi).$$

f is *conjunctive* if it distributes over any conjunction of predicates in its domain:

$$f(\wedge\, i \bullet \phi_i) \quad \equiv \quad (\wedge\, i \bullet f(\phi_i)).$$

f is *positively*-conjunctive if it distributes over any non-empty conjunction. A set of predicates ϕ_i is a *chain* if for each i, j

$$\phi_i \Rrightarrow \phi_j \quad \text{or} \quad \phi_j \Rrightarrow \phi_i.$$

A predicate transformer is $\wedge$-*continuous* if it is conjunctive over a chain.

Disjunctivity is the dual of conjunctivity:

$$f(\vee\ i \bullet \phi_i) \quad \equiv \quad (\vee\ i \bullet f(\phi_i)).$$

We also have *positive-disjunctivity* and $\vee$-*continuity*.

The *conjugate* of a predicate transformer f is written $\overline{f}$ and defined as:

$$\overline{f}(\phi) \quad \widehat{=} \quad \neg\, f(\neg\, \phi).$$

Note that the conjugate of $\overline{f}$ is f. It can easily be shown that if f is monotonic, then so is $\overline{f}$, and that $\overline{f}$ enjoys the dual of any junctivity properties enjoyed by f, e.g. if f is $\vee$-continuous then $\overline{f}$ is $\wedge$-continuous.

2.3 Specification Language

The specification language used in this thesis is an extension of Dijkstra's guarded-command language [Dij76]. The meaning of a statement S in this language is given by the *weakest-precondition* predicate-transformer: S transforms postcondition ϕ into the precondition $wp(S, \phi)$. We write the predicate transformer explicitly as $wp(S, \_)$. For statements S and T

$$S = T \quad \text{iff} \quad wp(S, \phi) \equiv wp(T, \phi) \;\; \text{for each predicate } \phi.$$

For any statement S in Dijkstra's original language definition, $wp(S, \_)$ is strict, monotonic, positively conjunctive, and $\vee$-continuous. In our extended language, $wp(S, \_)$ will be monotonic and positively conjunctive, but we lose strictness (also called Law of Excluded Miracle) and $\vee$-continuity (also called Bounded Nondeterminism). Statement S is *miraculous* if $wp(S, \_)$ is not strict.

The language extensions are defined in terms of weakest preconditions. The first is the *"naked" guarded-command* $(G \rightarrow S)$:

Definition 2.1 *For predicate G, statement S, postcondition ϕ,*

$$wp(G \rightarrow S, \phi) \quad \widehat{=} \quad G \Rightarrow wp(S, \phi).$$

□

$G \rightarrow S$ is a miracle provided $G \not\equiv \mathit{true}$. The extreme case is the statement **miracle** defined as $\mathit{false} \rightarrow \mathbf{skip}$.

The *choice* of a collection of statements S_i is written $([]\ i \bullet S_i)$ and defined as:

Definition 2.2 *For postcondition* ϕ*,*

$$wp(\ ([\!]\ i \bullet S_i),\ \phi) \ \ \widehat{=}\ \ (\wedge\ i \bullet wp(S_i, \phi)).$$

□

We will sometimes use the simpler case of $(S_1 [\!] S_2)$. $([\!]\ i \bullet S_i)$ is a miracle if the range of i is empty and need not be $\vee$-continuous (consider the unboundedly-nondeterministic statement $([\!]\ i \in \mathbb{N} \bullet x := i)$ which does not distribute over the chain-disjunction $(\vee\ j \in \mathbb{N} \bullet j > x)$).

The *specification statement* is written $v : [pre, post]$ and specifies that variable v may change to a state satisfying predicate *post* assuming predicate *pre* is satisfied initially. Both *pre* and *post* may refer to state variables, besides v, which remain unchanged, and *post* may refer to the initial value of v as v_0. The specification statement is defined by:

Definition 2.3 *For postcondition* ϕ*,*

$$wp(v : [pre, post],\ \phi) \ \ \widehat{=}\ \ pre \wedge (\forall\ v \bullet post \Rightarrow \phi)[v_0 \backslash v].$$

□

For example, the statement $x : [x \geq y, x = x_0 - y]$ decrements x by y provided $x \geq y$, otherwise it aborts. We adopt the convention that $v : [post]$ is short for $v : [true, post]$[1]. A specification statement may be miraculous (e.g. $v : [false]$) and need not be $\vee$-continuous (consider the unboundedly-nondeterministic statement $x : [x \in \mathbb{N}]$).

We sometimes use two special cases of the specification statement. The first is *generalised assignment*, written $v :\odot\ e$, defined as:

Definition 2.4 $v :\odot\ e \ \ \widehat{=}\ \ v : [\ v \odot e[v \backslash v_0]\ ].$

Here e is some expression and $\odot$ is a binary relation such as $=$, $<$, $\in$. The second case is the *coercion*, written $[post]$, which is a specification statement in which no state variables are allowed to change value. We usually use it sequentially composed with some other statement and write $S[post]$ for $S\ ;\ [post]$. The effect is to reduce the nondeterminism of S by forcing it to take a path that satisfies *post*.

[1] This is different to [Mor88b] where it is short for $v : [\ (\exists\ v \bullet post)[v_0 \backslash v],\ post\ ]$.

For example, $(x :\in \mathbb{N})[x < 10]$ assigns some natural number less than 10 to x:

$$\begin{array}{lll} & wp(\ (x :\in \mathbb{N})[x < 10], \phi) & \\ \equiv & wp(x :\in \mathbb{N}, x < 10 \Rightarrow \phi) & \textit{Definition 2.3} \\ \equiv & (\forall\, x \bullet x{\in}\mathbb{N} \Rightarrow (x < 10 \Rightarrow \phi)) & \textit{Definitions 2.4,2.3} \\ \equiv & (\forall\, x \bullet (x{\in}\mathbb{N} \wedge x < 10) \Rightarrow \phi) & \textit{predicate calculus} \\ \equiv & wp(x : [x \in \mathbb{N} \wedge x < 10], \phi) & \textit{Definition 2.3.} \end{array}$$

If S cannot satisfy *post* then it becomes miraculous, e.g.

$$(x := 10)[x \neq 10] \quad = \quad \textbf{miracle}.$$

The ability to coerce the outcome of the preceding statement is shared by all miracles.

Other extensions are local variables and substitution-by-value:

Definition 2.5 *For postcondition ϕ independent of x,*

$$wp(\ (\textbf{var}\ x \bullet S),\ \phi) \quad \widehat{=} \quad (\forall\, x \bullet wp(S, \phi)).$$

□

Definition 2.6 *For expression E, postcondition ϕ independent of x,*

$$wp(S[\textbf{value}\ x\backslash E],\ \phi) \quad \widehat{=} \quad wp(S, \phi)[x\backslash E].$$

□

For example, we have

$$\begin{array}{rcl} (\textbf{var}\ x \bullet x := m\ ;\ m := n\ ;\ n := x) & = & m, n := n, m \\ (m := x)[\textbf{value}\ x\backslash 5] & = & m := 5. \end{array}$$

We sometimes use $halt(S)$ to represent those initial states from which statement S is guaranteed to terminate:

Definition 2.7 *For statement S,* $\qquad halt(S) \ \widehat{=}\ wp(S, true).$

Sources: Miracles are introduced by Nelson [Nel89], Morris [Mrr87], and Morgan [Mor88b], and "naked" guarded-commands and choice are defined by each author. Back [Bac80] introduced the weakest-precondition semantics of specification statements, though his specification statements were restricted to be non-miraculous. Morris [Mrr87] and Morgan [Mor88b] also define specification statements, and our definition of specification statements (including generalised assignment and coercion) is from [Mor88b]. Local variables are defined in various places including [Nel89] and [Mor90b], and substitution-by-value is defined in [Mor88a]. Originally, bounded nondeterminism was required in the guarded-command language in order to define the weakest-precondition semantics of iteration by natural induction [Dij76]. However, Boom [Boo82] showed how the restriction to bounded nondeterminism could be removed by using ordinal induction instead. Back [Bac81] used an infinitary logic to remove the restriction to bounded nondeterminism in the guarded-command language.

2.4 Conjugate Weakest Precondition

The conjugate of predicate transformer $wp(S, \_)$ is written $\overline{wp}(S, \_)$. Since $wp(S, \_)$ is monotonic and positively conjunctive in our language, we have that $\overline{wp}(S, \_)$ is monotonic and positively disjunctive. Because

$$\overline{wp}(S, \phi) \quad \equiv \quad \neg\, wp(S, \neg\, \phi),$$

$\overline{wp}(S, \phi)$ is understood to represent those initial states from which S could *possibly* establish ϕ [Mor90a]. This also means that a non-terminating statement could possibly establish anything:

$$\begin{array}{llll} \neg\, halt(S) & \equiv & \overline{wp}(S, false) & \qquad \textit{Definition 2.7} \\ & \Rrightarrow & \overline{wp}(S, \phi) & \qquad \textit{monotonicity.} \end{array}$$

wp and $\overline{wp}$ are related by the following theorem which follows from monotonicity and positive conjunctivity of $wp(S, \_)$:

Theorem 2.8 *For any statement S, predicates ϕ and ψ,*

$$wp(S, \phi) \wedge \overline{wp}(S, \psi) \quad \Rrightarrow \quad \overline{wp}(S, \phi \wedge \psi).$$

□

2.5 Statement & Variable Independence

A statement is independent of a variable x if it is neither affected by nor affects x. We say that statement S is an x-statement if it is independent of all variables except x. Nelson [Nel89] has defined variable independence formally, and his definition is similar[2] to the following:

Definition 2.9 *A statement S is independent of variable x if*

1. *if predicate ϕ is independent of x, then so is $wp(S, \phi)$,*
2. *for x-predicate ψ, any ϕ,* $\quad wp(S, \phi \vee \psi) \;\equiv\; wp(S, \phi) \vee (\psi \wedge halt(S))$.

□

For example, the statement $y := f(y, z)$ is independent of x.

2.6 Refinement of Statements

Several authors have investigated refinement of statements in terms of weakest preconditions, [Bac90a, Nel89, Mrr87, GM88]. We use the definitions of Gardiner & Morgan [GM88] and write $S \preceq T$ for statement S is (*algorithmically*) refined by statement T:

Definition 2.10 $S \preceq T$ *if for each predicate ϕ,*

$$wp(S, \phi) \;\Rrightarrow\; wp(T, \phi).$$

□

Clearly, $\preceq$ is a transitive ordering.

Data refinement is used when S has abstract variable a and T has concrete variable c (and both may have variable z in common). If S is an (a, z)-statement, T is a (c, z)-statement, *rep* is a predicate transformer that maps a-dependent predicates to c-dependent predicates, then we write $S \preceq_{rep} T$ for "S is data-refined by T under representation function *rep*":

Definition 2.11 $S \preceq_{rep} T$ *if for each predicate ϕ independent of concrete variable c,*

$$rep(wp(S, \phi)) \;\Rrightarrow\; wp(T, rep(\phi)).$$

□

[GM88] shows that if *rep* enjoys certain properties and $S \preceq_{rep} T$, then

$$(\mathbf{var}\ a \bullet S) \;\preceq\; (\mathbf{var}\ c \bullet T).$$

[2] Nelson's definition is in terms of weakest *liberal* precondition (wlp) formulae [Dij76] rather than wp.

2.7 Action Systems and CSP

This section presents the correspondence between action systems and CSP introduced by Morgan [Mor90a]. An action system P has the form

$$P \;=\; (A,\; v,\; P_\imath,\; P_A)$$

where A is an alphabet of labels (possibly infinite), v is the state-variable[3], $P_\imath$ is an *initialisation*, and $P_A = \{\ P_a \mid a \in A\ \}$ is a set of labelled *actions*. $P_\imath$ and each P_a are statements in the extended Dijkstra language. This is similar to Back's formalism, except that we are also interested in action labels.

For a labelled action P_a, we understand the predicate $\neg\, wp(P_a, \mathit{false})$, the initial condition under which P_a is non-miraculous, to represent its *enabling* condition or *guard*. This is written $gd(P_a)$ where

Definition 2.12 *For statement S,* $\quad gd(S) \;\hat{=}\; \overline{wp}(S, \mathit{true}).$

For example, if S is non-miraculous, then $gd(G \rightarrow S) \equiv G$.

The correspondence between action systems and CSP is defined by giving a failures-divergences semantics to action systems. Firstly we have some definitions and conditions. For trace $s \in A^*$, we write P_s for the sequential composition of P actions drawn from s:

Definition 2.13 *For $a \in A,\;\; s, t \in A^*$,*

$$\begin{aligned} P_{\langle\rangle} \;&\hat{=}\; \mathbf{skip} \\ P_{\langle a\rangle} \;&\hat{=}\; P_a \\ P_{st} \;&\hat{=}\; P_s \,;\, P_t. \end{aligned}$$

□

For set $X \subseteq A$, we write $gd(P_X)$ for the disjunction of the guards of actions drawn from X:

Definition 2.14 *For $X \subseteq A$,* $\quad gd(P_X) \;\hat{=}\; (\vee\ a \in X \bullet gd(P_a)).$

We require that the initialisation $P_\imath$ is non-miraculous ($gd(P_\imath) \equiv \mathit{true}$), and that the outcome of $P_\imath$ is independent of the initial value of the state variable, that is, for any v-predicate ϕ,

$$wp(P_\imath, \phi) \equiv \mathit{true} \quad \text{or} \quad wp(P_\imath, \phi) \equiv \mathit{false}.$$

The failures-divergences semantics of action system P, denoted $\{\!\![P]\!\!\}$, are given by the following definition [Mor90a]:

[3] It is not necessary to mention v, since it can be inferred from $P_\imath$ and P_A. However, we shall find it useful later on to have it explicitly mentioned.

Definition 2.15 *For action system $P = (A,\ v,\ P_\imath,\ P_A)$,*

$$\{\!|P|\!\} \ \widehat{=}\ (A, \mathcal{F}\{\!|P|\!\}, \mathcal{D}\{\!|P|\!\})$$

where $\mathcal{F}\{\!|P|\!\}$ are those $s \in A^*$, $X \subseteq A$ satisfying

$$\overline{wp}(P_{\langle\imath\rangle s}, \neg\, gd(P_X))$$

$\mathcal{D}\{\!|P|\!\}$ are those $s \in A^*$ satisfying

$$\neg\, halt(P_{\langle\imath\rangle s}).$$

□

(s, X) is a failure if, after initialisation, P could possibly engage in the action trace s and then refuse all actions X. Trace s is a divergence if $P_{\langle\imath\rangle s}$ aborts. Morgan [Mor90a] claims that using properties of $\overline{wp}$, it can be shown that $\{\!|P|\!\}$ satisfies well-formedness conditions **C0–C6** of [Hoa85, p.130][4]. For example, **C1** says that the traces of P must be prefix-closed (let $\mathcal{T}\{\!|P|\!\} \mathrel{\widehat{=}} \{t \mid (t, \{\}) \in \mathcal{F}\{\!|P|\!\}\}$):

$$\begin{aligned}
st \in \mathcal{T}\{\!|P|\!\} &\equiv \overline{wp}(P_{\langle\imath\rangle st}, true) && \textit{Definition 2.15}\\
&\equiv \overline{wp}(P_{\langle\imath\rangle s}, \overline{wp}(P_t, true)) && \textit{Definition 2.13}\\
&\Rrightarrow \overline{wp}(P_{\langle\imath\rangle s}, true) && \textit{monotonicity}\\
&\equiv s \in \mathcal{T}\{\!|P|\!\} && \textit{Definition 2.15.}
\end{aligned}$$

Note that the requirement that the initialisation of an action system be non-miraculous is necessary in order to ensure that well-formedness condition **C0** is satisfied [Mor90a]. **C0** says that the traces of a process must contain the empty trace at least:

$$\begin{aligned}
\langle\rangle \in \mathcal{T}\{\!|P|\!\} &\equiv true\\
\text{iff}\quad \overline{wp}(P_\imath, true) &\equiv true && \textit{Definition 2.15}\\
\text{iff}\quad wp(P_\imath, false) &\equiv false.
\end{aligned}$$

For action systems P, Q, we write $P = Q$ for $\{\!|P|\!\} = \{\!|Q|\!\}$.

Figure 2.A shows the notation used to specify simple action systems. *K1* is an action system with alphabet $\{a, b\}$ and boolean variable g. Using Definition 2.15 it can be shown that *K1* has the same failures-divergences as the CSP process $\mathcal{K}1$ defined as:

$$\mathcal{K}1 \ \widehat{=}\ (\mu X \bullet (a \to X) \sqcap (b \to X)).$$

[4] Roscoe [Ros88] uses a stronger version of **C3** to deal with infinite alphabets. It can easily be shown that $\{\!|P|\!\}$ satisfies this version.

$$K1 \;\widehat{=}\; \left(\begin{array}{lll} \textbf{var } g : \textbf{bool} & & \\ \textbf{initially} & g :\in \{true, false\} & \\ a :- & g \rightarrow g :\in \{true, false\} & \\ b :- & \neg\, g \rightarrow g :\in \{true, false\} & \end{array} \right)$$

Figure 2.A: Example action system.

2.8 Refinement of Action Systems

Recall from Chapter 1 that the CSP refinement ordering is defined in terms of failures-divergences inclusion:

$$(A, F, D) \sqsubseteq (A, F', D') \quad \text{iff} \quad F \supseteq F',\; D \supseteq D'.$$

For actions systems P, Q, we write $P \sqsubseteq Q$ for $\{\![P]\!\} \sqsubseteq \{\![Q]\!\}$. Woodcock & Morgan [WM90] have defined downwards simulation and upwards simulation for action systems, and show that if there is a simulation of either form between P and Q, then $P \sqsubseteq Q$ (i.e. simulation is sound).

A simulation is a relation between action systems with the same alphabet but possibly different state spaces:

$$P = (A,\; v,\; P_\imath,\; P_A) \quad \text{and} \quad Q = (A,\; w,\; Q_\imath,\; Q_A).$$

Assume that $v = (v', z)$ and $w = (w', z)$, so that v' is the abstract variable, w' is the concrete variable and z is common to both P and Q. Simulation relies on data refinement between corresponding actions of P and Q. To deal with initialisation statements we introduce a special case of data refinement ($\preceq'_{rep}$):

Definition 2.16 *$S \preceq'_{rep} T$ if for each predicate ϕ independent of concrete variable c,*

$$wp(S, \phi) \;\Rrightarrow\; wp(T, rep(\phi)).$$

□

Unlike [WM90], we make no distinction between upwards and downwards simulation, but simply define a *simulation* as follows:

Definition 2.17 *A simulation, with representation function rep, is a relation on action systems, denoted* $\sqsubseteq_{rep}$*, such that for action systems*

$$P = (A,\ v,\ P_\imath,\ P_A) \quad and \quad Q = (A,\ w,\ Q_\imath,\ Q_A)$$

$P \sqsubseteq_{rep} Q$ *if each of the following conditions hold:*

1. $P_\imath \preceq'_{rep} Q_\imath$
2. $P_a \preceq_{rep} Q_a, \qquad each\ a \in A$
3. $rep(gd(P_X)) \Rrightarrow gd(Q_X), \qquad each\ X \subseteq A.$

□

Note: Condition 3 implies that *rep* is strict (take $X = \{\}$). We shall refer to 1 and 2 as data-refinement conditions, and 3 as a progress condition.

Soundness of simulation is given by the following theorem:

Theorem 2.18 *If* $P \sqsubseteq_{rep} Q$ *then* $P \sqsubseteq Q$.

Proof: from Conditions 1 and 2 it can be shown by induction over $s \in A^*$ that

$$wp(P_{\langle\imath\rangle s}, \phi) \Rrightarrow wp(Q_{\langle\imath\rangle s}, rep(\phi)). \tag{i}$$

For failures inclusion:

$$\begin{aligned}
(s, X) \in \mathcal{F}[\![Q]\!] &\equiv \overline{wp}(Q_{\langle\imath\rangle s}, \neg\, gd(Q_X)) && \textit{Definition 2.15} \\
&\Rrightarrow \overline{wp}(Q_{\langle\imath\rangle s}, \overline{rep}(\neg\, gd(P_X))) && \textit{Condition 3, monotonicity} \\
&\Rrightarrow \overline{wp}(P_{\langle\imath\rangle s}, \neg\, gd(P_X)) && \textit{by}\ \text{(i)} \\
&\equiv (s, X) \in \mathcal{F}[\![P]\!] && \textit{Definition 2.15.}
\end{aligned}$$

For divergences inclusion:

$$\begin{aligned}
s \in \mathcal{D}[\![Q]\!] &\equiv \overline{wp}(Q_{\langle\imath\rangle s}, \mathit{false}) && \textit{Definition 2.15} \\
&\Rrightarrow \overline{wp}(Q_{\langle\imath\rangle s}, \overline{rep}(\mathit{false})) && \textit{monotonicity} \\
&\Rrightarrow \overline{wp}(P_{\langle\imath\rangle s}, \mathit{false}) && \textit{by}\ \text{(i)} \\
&\equiv s \in \mathcal{D}[\![P]\!] && \textit{Definition 2.15.}
\end{aligned}$$

□

If *rep* is disjunctive then, rather than ensuring progress over subsets of A (Condition 3, Definition 2.17), we only need to ensure it for individual actions:

Lemma 2.19 *If rep is disjunctive and*

$$rep(gd(P_a)) \;\Rrightarrow\; gd(Q_a), \qquad \text{each } a \in A$$

then
$$rep(gd(P_X)) \;\Rrightarrow\; gd(Q_X), \qquad \text{each } X \subseteq A.$$

□

The proof of Theorem 2.18 is similar to the proofs of soundness of both upwards and downwards simulation presented by Woodcock & Morgan [WM90]. Their downwards simulation is the same as our simulation with the following representation function, (where a represents the abstract variable):

$$rep_d(\phi) \;\;\widehat{=}\;\; (\exists\, a \bullet I \wedge \phi).$$

Here the predicate I describes a correspondence between the abstract variable and the concrete variable. In downwards simulation, it is only necessary to ensure progress over individual actions, since rep_d is disjunctive (cf. Lemma 2.19). Their upwards simulation is the same as our simulation with the following (non-disjunctive) representation function:

$$rep_u(\phi) \;\;\widehat{=}\;\; (\forall\, a \bullet I \Rightarrow \phi).$$

Woodcock & Morgan have shown that upwards and downwards simulation are jointly complete. Since downwards and upwards simulation are just special cases of Definition 2.17, their completeness result means that Definition 2.17 is complete. Combining downwards and upwards simulation into a single definition follows the work of Gardiner & Morgan [GM89] which shows that Definition 2.11 is complete for data refinement of programs.

2.9 Internal and External Choice

The definition of simulation used by us is similar to Back's rule for data refinement of action systems (without stuttering) as described on Page 12 of this thesis. One difference, however, is that, in our case, the second condition of Definition 2.17 compares individual actions as follows:

$$P_a \;\preceq_{rep}\; Q_a, \qquad \text{each } a \in A,$$

whereas in Back's rule, actions of the respective systems are bunched together as follows:

$$([\!]\, a \in A \bullet P_a) \;\;\preceq_{rep}\;\; ([\!]\, a \in A \bullet Q_a).$$

$$K2 \mathrel{\widehat{=}} \left(\begin{array}{ll} \textbf{initially} & \textbf{skip} \\ \mathit{tea} :- & \textbf{skip} \\ \mathit{coffee} :- & \quad \textbf{skip} \end{array}\right)$$

$$K3 \mathrel{\widehat{=}} \left(\begin{array}{ll} \textbf{var } g : \textbf{bool} & \\ \textbf{initially} & g :\in \{true, false\} \\ \mathit{tea} :- & g \rightarrow g :\in \{true, false\} \\ \mathit{coffee} :- & \quad \neg\, g \rightarrow g :\in \{true, false\} \end{array}\right)$$

Figure 2.B: External and internal choice.

Similarly, our third condition is over all subsets of A (or individual actions if *rep* is disjunctive), whereas Back's third condition is simply on all actions of A together.

Back's rule allows more flexibility in refinement but it does not distinguish internal and external choice. Consider the action systems $K2$ and $K3$ of Figure 2.B. $K2$ offers the environment the choice between a *tea*-event or a *coffee*-event, whereas $K3$ chooses internally whether to offer a *tea*-event or a *coffee*-event. So, it is not the case that $K2 \sqsubseteq K3$, and Definition 2.17 fails (in particular, the progress condition fails). However, using Back's rule it can be shown that $K2$ is refined by $K3$ (take $Rep \equiv g \in \{true, false\}$). This is because Back's rule ensures state-trace refinement and, as mentioned in Section 1.3, the state-traces model of transition systems does not distinguish choice of transition (external choice) from nondeterministic effect of a transition (internal choice).

Chapter 3

Infinite-Traces Model

In this chapter, the correspondence between action systems and CSP is extended to the *infinite-traces* model for CSP. This is achieved by developing a theory of infinite sequential-composition for statements.

3.1 Introduction

It is well-known that the failures-divergences model for CSP does not deal adequately with unbounded nondeterminism [Ros88]. Consider the CSP processes $\mathcal{L}1$ and $\mathcal{L}2$ shown in Figure 3.A. $\mathcal{L}1$ may continue to perform a-events or may nondeterministically deadlock at any time. $\mathcal{L}2$ performs an unbounded number of a-events and then stops. The essential difference between these processes is that $\mathcal{L}1$ might perform a-events continually, while $\mathcal{L}2$ is guaranteed to stop eventually.

But the failures-divergences model fails to distinguish $\mathcal{L}1$ and $\mathcal{L}2$, since

$$[\![\mathcal{L}1]\!] \;=\; [\![\mathcal{L}2]\!] \;=\; (\{a\},\; \{a\}^* \times \mathbb{P}\{a\},\; \{\}).$$

The consequences of this are noticeable if the hiding operator is applied to $\mathcal{L}1$ and $\mathcal{L}2$. Divergence is introduced by the hiding operator when possible infinite behaviour is hidden and, in the failures-divergences model, infinite behaviour is interpolated from finite behaviour rather than being modelled explicitly. By the definition of the CSP hiding operator [Hoa85, p.128,131] we get

$$\mathcal{L}1 \setminus \{a\} \;=\; \mathcal{L}2 \setminus \{a\} \;=\; \mathit{CHAOS},$$

the process that diverges immediately. But intuitively we expect that

$$\mathcal{L}2 \setminus \{a\} \;=\; \mathit{STOP}.$$

The infinite-traces model for CSP was introduced by Roscoe [Ros88] in order to deal with the shortcoming. It extends the failures-divergences model by including

$$
\begin{aligned}
\mathcal{L}1 &\ \widehat{=}\ (\mu X \bullet STOP \sqcap a \rightarrow X) \\
\mathcal{L}2 &\ \widehat{=}\ (\sqcap i \in \mathbb{N} \bullet \mathcal{L}2_i) \\
& \qquad where\ \ \mathcal{L}2_0 \ \widehat{=}\ STOP, \qquad \mathcal{L}2_{i+1} \ \widehat{=}\ a \rightarrow \mathcal{L}2_i
\end{aligned}
$$

Figure 3.A: Bounded and unbounded nondeterminism in CSP processes.

all possible infinite behaviours of a process. A process model now has components (A, F, D, I). A, F and D are as in the failures-divergences model, and I is some subset of A^ω, the set of infinite sequences of elements of A. In the extended model $\mathcal{L}1$ and $\mathcal{L}2$ are distinguished:

$$
\begin{aligned}
[\![\mathcal{L}1]\!] &= (\{a\},\ \{a\}^* \times \mathbb{P}\{a\},\ \{\},\ \{a\}^\omega) \\
[\![\mathcal{L}2]\!] &= (\{a\},\ \{a\}^* \times \mathbb{P}\{a\},\ \{\},\ \{\}).
\end{aligned}
$$

The hiding operator in the infinite-traces model uses the I component to determine when divergence may be introduced [Ros88]. So now we get $\mathcal{L}1 \setminus \{a\} = CHAOS$ and $\mathcal{L}2 \setminus \{a\} = STOP$ as expected.

It is necessary to extend the correspondence between action systems and CSP to the infinite-traces model because we allow statements to be unboundedly nondeterministic. Although the correspondence with the failures-divergences model (Definition 2.15) does not depend on nondeterminism being bounded, we will show in Chapter 4 that the hiding operator we introduce for (unboundedly-nondeterministic) action systems only corresponds to CSP hiding in the infinite traces model, and not in the failures-divergences model. To define the infinite traces of an action system we introduce the predicate $\overline{inf}(\Sigma)$, for an infinite sequential-composition of statements Σ. We also extend simulation so that it is sound in the infinite-traces model, though, we show that completeness is lost in certain cases. Firstly, we look at some fixed-point theory.

3.2 Fixed-Point Theory

A set $\mathcal{A}$ with partial order $\Rrightarrow$ is a *complete lattice* if every subset $\mathcal{B}$ of $\mathcal{A}$ has a least upper bound and a greatest lower bound [Tar55]. We have that the set of all predicates, with the entailment ordering $\Rrightarrow$, forms a complete lattice.

A fixed-point of a predicate transformer f is any predicate X satisfying

$$f(X) \;\equiv\; X.$$

We write the *least* and *greatest* fixed-points of f respectively as $(\mu\, X \bullet f(X))$ and $(\overline{\mu}\; X \bullet f(X))$. The Knaster-Tarski Theorem [Tar55] proves the existence of $(\mu\, X \bullet f(X))$ and $(\overline{\mu}\; X \bullet f(X))$ for monotonic f:

Knaster-Tarski Theorem *For complete lattice $(\mathcal{A}, \Rrightarrow)$, and monotonic f on $\mathcal{A}$ to $\mathcal{A}$, $(\mu\, X \bullet f(X))$ and $(\overline{\mu}\; X \bullet f(X))$ exist. Furthermore $(\mu\, X \bullet f(X))$ is equivalent to the* least *solution of*

$$f(X) \;\Rrightarrow\; X \tag{i}$$

and $(\overline{\mu}\; X \bullet f(X))$ is equivalent to the greatest *solution of*

$$f(X) \;\Lleftarrow\; X. \tag{ii}$$

□

As well as proving the existence of $(\mu\, X \bullet f(X))$ and $(\overline{\mu}\; X \bullet f(X))$, the theorem is useful in other ways. For example, if we wish to show that for some predicate ϕ

$$\phi \;\Rrightarrow\; (\overline{\mu}\; X \bullet f(X))$$

then we only need to show that ϕ is a solution of (ii).

If f is $\vee$-continuous then it can be shown that

$$(\mu\, X \bullet f(X)) \;\equiv\; (\vee\; i \in \mathbb{N} \bullet f^{i}(\mathit{false}))$$

where f^i is application of f i times. Hitchcock & Park [HP72] have improved on this in their *Generalised Limit-Theorem*, by extending the range of i to the ordinals so that $\vee$-continuity is no longer required. We use the version presented by Nelson [Nel89]:

Theorem 3.1 (Generalised Limit-Theorem) *Let f be a monotonic function, and for ordinal α, let f^α be defined inductively by*

$$f^\alpha \;\widehat{=}\; (\vee\; \beta \mid \beta < \alpha \bullet f(f^\beta)).$$

Then f has a least *fixed-point given by f^α, for some ordinal α.*
□

Note that the range of the disjunction defining f^α is bounded by ordinal α, thus ensuring that the disjunction is over a set of predicates.

The theorem is proven by showing firstly that we can always choose an α whose cardinality exceeds the cardinality of the domain of f, in which case $f^\alpha \equiv f^\beta$ for some $\beta < \alpha$, and secondly that in this case f^α is the *least* fixed-point of f. The Generalised Limit-Theorem allows us to reason about $(\mu X \bullet f(X))$ using ordinal induction. The theorem was used by Nelson [Nel89] to introduce general recursion to the guarded-command language. The theorem has a dual as follows:

Theorem 3.2 *Let $\overline{f}$ be a monotonic function, and for ordinal α, let $\overline{f}^\alpha$ be defined inductively by*

$$\overline{f}^\alpha \;\; \widehat{=} \;\; (\wedge\, \beta \mid \beta < \alpha \bullet \overline{f}(\overline{f}^\beta)).$$

Then $\overline{f}$ has a greatest *fixed-point given by $\overline{f}^\alpha$, for some ordinal α.*
□

3.3 Infinite Sequential-Composition

To formalise infinite sequential-composition we employ infinite sequences of predicates of the form $\Phi = \langle\, \Phi_i \mid i \in \mathbb{N}\, \rangle$. A lifted ordering on such sequences is defined pointwise:

Definition 3.3 $\Phi \Rrightarrow \Psi$ *iff* $\Phi_i \Rrightarrow \Psi_i$ *for each* $i \in \mathbb{N}$.

Infinite sequences of predicate transformers are of the form $F = \langle\, F_i \mid i \in \mathbb{N}\, \rangle$. Application of F to Φ is defined by shifting Φ one place and applying F pointwise:

Definition 3.4 $F(\Phi) \;\; \widehat{=} \;\; \langle\, F_i(\Phi_{i+1}) \mid i \in \mathbb{N}\, \rangle.$

Such application is monotonic provided each individual F_i is monotonic.

Given an infinite sequence of statements $\langle\, S_i \mid i \in \mathbb{N}\, \rangle$ we write their infinite sequential-composition as

$$(\,;\, i \mid i \in \mathbb{N} \bullet S_i).$$

It is intended that $(\,;\, i \mid i \in \mathbb{N} \bullet S_i) = (S_0 \,;\, S_1 \,;\, S_2 \,;\, \ldots\,)$. Given an infinitary statement $\Sigma = (\,;\, i \mid i \in \mathbb{N} \bullet S_i)$, an infinite sequence of predicate transformers, written $\overline{wp}_\infty(\Sigma, \_)$, may be constructed as follows:

Definition 3.5 $\qquad \overline{wp}_\infty(\Sigma, \_) \;\; \widehat{=} \;\; \langle\, \overline{wp}(S_i, \_) \mid i \in \mathbb{N}\, \rangle.$

For an infinitary statement Σ, we require the predicate $\overline{\mathit{inf}}(\Sigma)$ to characterise those initial states in which execution of all of Σ in sequence is *possible*. Operationally, for this to be possible in some state it must be the case that S_0 is enabled and that execution of S_0 could result in a state in which S_1 is enabled and so on for each S_i. So for infinite predicate Φ satisfying

$$\Phi_i \quad \Rrightarrow \quad \overline{\mathit{wp}}(S_i, \Phi_{i+1}), \qquad \text{each } i \in \mathbb{N}, \tag{iii}$$

we have that Φ_0 represents a set of initial states in which execution of all of Σ is possible. (Note that $\overline{\mathit{wp}}(S_i, \Phi_{i+1}) \Rrightarrow \mathit{gd}(S_i)$.) This includes the possibility that one of the individual S_i's will abort. Now (iii) can be rewritten as

$$\Phi \Rrightarrow \overline{\mathit{wp}}_\infty(\Sigma, \Phi). \tag{iv}$$

Since we are looking for the weakest condition under which execution of all Σ is possible we define $\overline{\mathit{inf}}(\Sigma)$ to be the first element of the infinite sequence Ψ, where Ψ is the weakest (i.e. greatest) solution of (iv). By the Knaster-Tarski Theorem, the greatest solution of the equation $X \Rrightarrow f(X)$, for monotonic f, is also the greatest fixed-point of f. So $\overline{\mathit{inf}}(\Sigma)$ is defined as follows:

Definition 3.6 *For* $\Sigma = (\ ;\ i \mid i \in \mathbb{N} \bullet S_i)$ *let* $\Psi \ \widehat{=}\ (\overline{\mu}\ X \bullet \overline{\mathit{wp}}_\infty(\Sigma, X))$*. Then*

$$\overline{\mathit{inf}}(\Sigma) \quad \widehat{=} \quad \Psi_0.$$

□

Least upper-bound and greatest lower-bound of sets of infinite predicates may be calculated pointwise:

Theorem 3.7 *For a collection of infinite predicates* Φ^j*,*

$$\begin{aligned} (\vee\ j \bullet \Phi^j) &\equiv \langle\ (\vee\ j \bullet \Phi^j_i) \mid i \in \mathbb{N}\ \rangle \\ (\wedge\ j \bullet \Phi^j) &\equiv \langle\ (\wedge\ j \bullet \Phi^j_i) \mid i \in \mathbb{N}\ \rangle. \end{aligned}$$

□

The set of infinite predicates, with the lifted ordering, forms a complete lattice, so $(\overline{\mu}\ X \bullet \overline{\mathit{wp}}_\infty(\Sigma, X))$ exists by the Knaster-Tarski Theorem.

The following theorem, describing a relationship between $\overline{\mathit{inf}}$ and $\overline{\mathit{wp}}$, is proven using the Knaster-Tarski Theorem (see Appendix A):

Theorem 3.8 $\overline{\mathit{inf}}(S\ ;\ \Sigma) \ \equiv\ \overline{\mathit{wp}}(S, \overline{\mathit{inf}}(\Sigma))$.

A special case of infinite sequential-composition is the homogeneous sequence S^∞, defined as

Definition 3.9 *For statement S,* $\qquad S^\infty \;\widehat{=}\; (\;;\, i \mid i \in \mathbb{N} \bullet S).$

From the Knaster-Tarski Theorem and Theorem 3.8, we can prove the following (see Appendix A):

Theorem 3.10 $\overline{inf}(S^\infty) \;\equiv\; (\overline{\mu}\; X \bullet \overline{wp}(S, X)).$

3.4 Extended CSP Correspondence

For action system $P \;=\; (A,\; v,\; P_\imath,\; P_A)$ and infinite trace $u \in A^\omega$, let

$$P_u \;\widehat{=}\; (\;;\, i \mid i \in \mathbb{N} \bullet P_{u_i}).$$

$\{\!\![P]\!\!\}$ now has an extra component: the failures and divergences of P are as given in Definition 2.15, and the infinites of P are defined by

Definition 3.11 $\mathcal{I}\{\!\![P]\!\!\}$ *are those* $u \in A^\omega$ *satisfying*

$$\overline{inf}(P_{\langle \imath \rangle u}).$$

□

Theorem 3.12 *In the infinite-traces model,* $\{\!\![P]\!\!\}$ *is well-formed.*

Proof: The components of $\{\!\![P]\!\!\}$ must satisfy the well-foundedness conditions of the failures-divergences model (**C0**–**C6**), as well as three extra conditions (**C7**–**C9**) given in [Ros88][1]. **C7** and **C8** are straightforward: for $t \in A^*$, $u \in A^\omega$,

C7 $tu \in I \Rightarrow (t, \{\}) \in F$
C8 $t \in D \Rightarrow tu \in I$.

Satisfaction of **C7** and **C8** follows easily from Theorem 3.8.

C9 is more involved. It requires that any process is equal to the nondeterministic choice of all its *pre-deterministic* implementations. A process is pre-deterministic if it is deterministic until it diverges. Showing that $\{\!\![P]\!\!\}$ satisfies **C9** involves constructing pre-deterministic implementations of P. Details are given in Appendix A.
□

[1] **C7**–**C9** are numbered **C6**–**C8** in [Ros88].

3.5 Refinement

In the infinite-traces model, the refinement ordering is defined in terms of failures-divergences-infinites inclusion [Ros88]:

$$(A, F, D, I) \sqsubseteq (A, F', D', I') \quad \text{iff} \quad F \supseteq F',\ D \supseteq D',\ I \supseteq I'.$$

The simulation relation between action systems (Definition 2.17) may be extended to the infinite-traces model, provided the representation function *rep* is $\vee$-continuous:

Theorem 3.13 *For $\vee$-continuous rep, if $P \sqsubseteq_{rep} Q$, then $P \sqsubseteq Q$ in the infinite-traces model.*

Proof: Failures-divergences inclusion is as for Theorem 2.18. From Condition 2 of simulation (Definition 2.17) and $\vee$-continuity of *rep* the following can be proven:

Lemma 3.14 *For $u \in A^\omega$,* $\qquad \overline{inf}(Q_u) \Rrightarrow \overline{rep}(\overline{inf}(P_u)).$

Proof of this lemma by the Generalised Limit-Theorem is given in Appendix A. Now, to show infinites inclusion we have, for $u \in A^\omega$,

$$
\begin{array}{lll}
 & u \in \mathcal{I}\{\!\!\{Q\}\!\!\} & \\
\equiv & \overline{wp}(Q_\iota, \overline{inf}(Q_u)) & \textit{Definition 3.11, Theorem 3.8} \\
\Rrightarrow & \overline{wp}(Q_\iota, \overline{rep}(\overline{inf}(P_u))) & \textit{monotonicity, Lemma 3.14} \\
\Rrightarrow & \overline{wp}(P_\iota, \overline{inf}(P_u)) & \textit{since } P_\iota \preceq'_{rep} Q_\iota \\
\equiv & u \in \mathcal{I}\{\!\!\{P\}\!\!\} & \textit{Theorem 3.8, Definition 3.11.}
\end{array}
$$

$\square$

We say that an action system is boundedly nondeterministic if its initialisation and each of its actions are boundedly nondeterministic. Simulation is sound for boundedly-nondeterministic action-systems even if *rep* is not $\vee$-continuous, as the following theorem shows:

Theorem 3.15 *For any rep, and boundedly-nondeterministic P, if $P \sqsubseteq_{rep} Q$, then $P \sqsubseteq Q$ in the infinite-traces model.*

Proof: Failures-divergences inclusion is as for Theorem 2.18. Since P is boundedly nondeterministic, we have the following lemma:

Lemma 3.16 $(\wedge\ s < u \bullet \overline{wp}(P_{\langle\iota\rangle s}, true)) \Rrightarrow \overline{inf}(P_{\langle\iota\rangle u}).$

This means that u is an infinite trace of P if each finite subsequence of u is a trace of P. The lemma is proven in Appendix A. Now, to show infinites inclusion we have, for $u \in A^\omega$,

$$\overline{inf}(Q_{\langle\imath\rangle u})$$

$$\Rrightarrow \quad (\wedge\, s < u \bullet \overline{wp}(Q_{\langle\imath\rangle s}, true)) \qquad \textit{Theorem 3.8, monotonicity}$$

$$\Rrightarrow \quad (\wedge\, s < u \bullet \overline{wp}(P_{\langle\imath\rangle s}, true)) \qquad \textit{since } P \sqsubseteq_{rep} Q$$

$$\Rrightarrow \quad \overline{inf}(P_{\langle\imath\rangle u}) \qquad \textit{Lemma 3.16.}$$

□

Thus, simulation is sound in the infinite-traces model if *rep* is $\vee$-continuous, or if P is boundedly nondeterministic.

3.6 Completeness

Recall the discussion of downwards and upwards simulation for action systems in Section 2.8. For downward simulation the disjunctive (and hence $\vee$-continuous) representation function rep_d is used, so that downward simulation is always sound in the infinite-traces model. However, the representation function rep_u is not in general $\vee$-continuous, so that upwards simulation is not sound in the infinite-traces model for unboundedly-nondeterministic action-systems. Therefore, the proof that downwards and upwards simulation are jointly complete, presented in [WM90], is no longer valid for the infinite-traces model. In particular, the rep_u used to show that an action system P is refined by its canonical form $P^\#$ is not $\vee$-continuous, so that for unboundedly nondeterministic P, it is not possible to show that $P = P^\#$. Simulation is complete in the infinite-traces model only for boundedly-nondeterministic action-systems, since in that case it doesn't matter that rep_u is non-continuous.

This limitation on completeness arises elsewhere. The upwards simulation of He, Hoare & Sanders [HHS86] for data refinement of sequential programs is sound only for *finitary* representation-relations (i.e. each concrete state corresponds to only finitely many abstract states). The construction of a canonical form, and hence completeness, is then only valid for boundedly-nondeterministic programs. Gardiner & Morgan's [GM89] completeness proof for data refinement of predicate transformers (Definition 2.11) is also valid only for boundedly-nondeterministic programs.

Likewise, there are limitations on the completeness of Abadi & Lamport's [AL88] refinement mappings. Refinement mappings are only complete for tran-

$$L1 \;\widehat{=}\; \left(\begin{array}{ll} \mathbf{var}\ g : \mathbf{bool} & \\ \mathbf{initially} & g :\in \{true, false\} \\ a :- & g \rightarrow g :\in \{true, false\} \end{array}\right)$$

$$L2 \;\widehat{=}\; \left(\begin{array}{ll} \mathbf{var}\ n : \mathbb{N} & \\ \mathbf{initially} & n :\in \mathbb{N} \\ a :- & n > 0 \rightarrow n := n - 1 \end{array}\right)$$

Figure 3.B: Bounded and unbounded nondeterminism in action systems.

sition systems that are *finitely invisibly nondeterministic*[2]. A transition system T is finitely invisibly nondeterministic if corresponding to any externally-visible transition $e_0 \rightarrow e_1$ in T, there are only finitely many full transitions $(e_0, h_0) \rightarrow (e_1, h_1)$. Jonsson [Jss90] places similar restrictions on his completeness result for I/O-automata.

The completeness result of He Jifeng [He89] and Josephs [Jos88] for CSP refinement of transition systems is not restricted to bounded nondeterminism. Neither is the completeness result of Woodcock & Morgan [WM90] for CSP refinement of action systems. The reason is that they only work with the failures-divergences model for CSP, rather than the infinite-traces model. Consider the action systems $L1$ and $L2$ of Figure 3.B. It is easy to see that they correspond respectively to the CSP processes $\mathcal{L}1$ and $\mathcal{L}2$ of Figure 3.A. Note that the initialisation of $L2$ is unboundedly nondeterministic. Using Woodcock & Morgan's simulation rules, it can be shown that $L2 \sqsubseteq L1$. This is not surprising, since they both have the same failures-divergences semantics. Yet $L1$ may engage in a-events continually, while $L2$ must terminate eventually. In the infinite-traces model, it is not true that $L2 \sqsubseteq L1$, so there can be no $\vee$-continuous representation function rep such that $L2 \sqsubseteq_{rep} L1$.

Despite losing completeness, we shall continue to assume that our action systems are unboundedly nondeterministic, and so representation functions shall be required to be $\vee$-continuous.

[2]Completeness in Lamport & Abadi also depends on *internal continuity* (cf. [AL88]).

Chapter 4

Internal Actions

In this chapter, action systems are extended to include internal actions. We introduce the program construct **it** S **ti**, representing nondeterministic iteration over S, to give a CSP meaning to such action systems. We show that internalisation of actions corresponds to CSP hiding, and show how refinement steps involving the introduction of internal actions may be achieved.

4.1 Introduction

An action system with internal actions has the form:

$$P = (A, v, P_\iota, P_A, P_H).$$

The intention is that any number of executions of an internal action $P_h, h \in H$, might occur in between each execution of a visible action P_a. If the action system reaches a state where internal actions can be executed infinitely then the action system diverges.

Process events in CSP are internalised by the hiding operator: hiding of a set of events $C \subseteq \alpha\mathcal{P}$ is written $\mathcal{P} \setminus C$. The semantics of process $\mathcal{P} \setminus C$ is defined in terms of the semantics of process $\mathcal{P}$:

$$[\![\mathcal{P} \setminus C]\!] \;\widehat{=}\; [\![\mathcal{P}]\!] \setminus C.$$

The definition of the semantic function '\' is given later in this chapter.

A corresponding operator for action systems is introduced in Section 4.5: hiding of the set of actions labelled C in action system P results in the action system $P \setminus C$. Hiding will be achieved simply by internalising the actions labelled C, and we will give a meaning to internal actions such that

$$\{\!|P \setminus C|\!\} = \{\!|P|\!\} \setminus C. \qquad \text{(i)}$$

The correspondence between action systems with internal actions and the CSP infinite-traces model is defined using **it** S **ti** and shown to satisfy the well-formedness conditions. We also show that equation (i) holds in the infinite-traces model and that hiding in the failures-divergences model is inappropriate for unboundedly-nondeterministic action systems.

Using a definition of action system hiding that is exactly CSP hiding gives us the benefit of those properties enjoyed by the CSP hiding operator, such as monotonicity. Also, CSP hiding is implemented by some concurrent programming-languages, e.g. occam [JG88]. One of the difficulties of using CSP hiding, however, is ensuring that divergence is not introduced. A well-foundedness theorem for the iterate construct will be given which allows a variant function to be used to prove that divergence is not introduced by a refinement step, in the same manner that loop termination is proven for sequential programs.

4.2 The Iterate Construct

The iterate construct **it** S **ti**, introduced in [Mor79], causes the statement S to be executed zero or more times and is defined by the equation

$$\mathbf{it}\ S\ \mathbf{ti} \;=\; \mathbf{skip} \,[\!]\, (S\ ;\ \mathbf{it}\ S\ \mathbf{ti}). \tag{ii}$$

That is, nondeterministically terminate, or execute S and then iterate again. If S becomes miraculous, then **it** S **ti** is forced to terminate, e.g.

$$\mathbf{it}\ \mathit{false} \rightarrow S\ \mathbf{ti} \;=\; \mathbf{skip}.$$

We require **it** S **ti** to be the *least-refined* program satisfying equation (ii), so its weakest precondition w.r.t. postcondition ϕ is the least predicate Y satisfying:

$$\begin{aligned}
Y &\equiv wp(\mathbf{it}\ S\ \mathbf{ti}, \phi) \\
&\equiv wp(\mathbf{skip} \,[\!]\, (S\ ;\ \mathbf{it}\ S\ \mathbf{ti}), \phi) && \textit{from (ii)} \\
&\equiv wp(\mathbf{skip}, \phi) \wedge wp(S, wp(\mathbf{it}\ S\ \mathbf{ti}, \phi)) && \textit{wp-calculus} \\
&\equiv \phi \wedge wp(S, Y).
\end{aligned}$$

The precise definition is

Definition 4.1 *For predicate* ϕ, $\quad wp(\mathbf{it}\ S\ \mathbf{ti}, \phi) \;\widehat{=}\; (\mu\, X \bullet \phi \wedge wp(S, X))$.

Immediately we see that **it** S **ti** is non-miraculous:

$$\begin{aligned}
wp(\mathbf{it}\ S\ \mathbf{ti}, \mathit{false}) &\equiv \mathit{false} \wedge wp(S, wp(\mathbf{it}\ S\ \mathbf{ti}, \mathit{false})) \\
&\equiv \mathit{false}.
\end{aligned}$$

As an example of the iterate construct, consider the following statement:

$$Dec \;\widehat{=}\; \textbf{it}\ n > 0 \rightarrow n := n - 1\ \textbf{ti}.$$

Assuming n is always a natural number it can be shown that $Dec = n :\leq n$, i.e. that Dec always terminates having decremented n by some nondeterministic amount. On the other hand, consider the statement

$$Flip \;\widehat{=}\; \textbf{it}\ g \rightarrow g :\in \{true, false\}\ \textbf{ti}.$$

This loop will only terminate when g becomes false, but since the loop body is not guaranteed to falsify g the loop may never terminate. It can be shown that

$$(g := true\ ;\ Flip) \;=\; \textbf{abort}.$$

The following theorem about **it** S **ti** is important and leads to other properties:

Theorem 4.2 *Let S^i be i copies of S composed sequentially ($S^0 =$ **skip**) and let S^∞ be the infinite sequential-composition of S (Definition 3.9). Then for any predicate ϕ,*

$$wp(\textbf{it}\ S\ \textbf{ti}, \phi) \;\equiv\; (\wedge\, i \in \mathbb{N} \bullet wp(S^i, \phi)) \;\wedge\; \neg\, \overline{inf}(S^\infty).$$

□

Proof of this theorem uses the Generalised Limit-Theorem (3.1) and is given in Appendix B.

It can be shown by induction over i that S^i preserves monotonicity and positive conjunctivity, hence by Theorem 4.2 the iterate construct also preserves these properties. The following lemmas, which also follow from Theorem 4.2, will be useful later:

Lemma 4.3 *If $P_B = ([\!]\, b \in B \bullet P_b)$ then*

$$wp(\textbf{it}\ P_B\ \textbf{ti}, \phi) \;\equiv\; (\wedge\, t \in B^* \bullet wp(P_t, \phi)) \;\wedge\; (\wedge\, u \in B^\omega \bullet \neg\, \overline{inf}(P_u)).$$

□

Lemma 4.4 $\quad \textbf{it}\ S\ \textbf{ti}\ ;\ \textbf{it}\ S\ \textbf{ti} \;=\; \textbf{it}\ S\ \textbf{ti}.$

Lemma 4.5 *If $P_B = ([\!]\, b \in B \bullet P_b)$ and $P_C = ([\!]\, c \in C \bullet P_c)$ then*

$$\begin{aligned} \textbf{it}\ P_B [\!]\, P_C\ \textbf{ti} &= \textbf{it}\ P_B\ \textbf{ti}\ ;\ \textbf{it}\ (P_C\ ;\ \textbf{it}\ P_B\ \textbf{ti})\ \textbf{ti} \\ &= \textbf{it}\ (\textbf{it}\ P_B\ \textbf{ti}\ ;\ P_C)\ \textbf{ti}\ ;\ \textbf{it}\ P_B\ \textbf{ti}. \end{aligned}$$

□

Each of these lemmas is proven in Appendix B.

Observation: Lemmas 4.4 and 4.5 illustrate a similarity between our specification language and regular expressions [Min67]. If " ; ", "$[]$", and "**it** S **ti**" correspond to the regular-expression operators for sequencing, choice, and iteration respectively, then these lemmas are also satisfied by regular expressions. Backhouse & van Gasteren [BvG92] refer to the regular-expression rule corresponding to Lemma 4.5 as "star decomposition". Nelson [Nel92] also uses a similar rule for his nondeterministic iterate operator on statements similar to our **it** S **ti**. However, the correspondence with regular expressions is not exact: if **skip** corresponds to the empty expression, then in regular expressions we would have **it skip ti** = **skip** whereas from Definition 4.1 we can show that **it skip ti** = **abort**. To achieve an exact correspondence with regular expressions, we would need to define an iterate construct that is the *most*-refined solution of equation (ii).

Van Gasteren has provided a proof for star decomposition in regular expressions [vG92]. However, since the correspondence with our **it** S **ti** is not exact, that proof does not apply to Lemma 4.5. In our case, Lemma 4.5 is proven using Lemma 4.3 (see Appendix B).

4.3 Correspondence with CSP

For action system $P = (A, v, P_\imath, P_A, P_H)$, let the statement IT_H be defined as follows:

$$IT_H \;\widehat{=}\; \textbf{it}\ ([]\, h \in H \bullet P_h)\ \textbf{ti}.$$

For trace t, let $P_t * H$ represent t interspersed with the statement IT_H:

Definition 4.6 *For $a \in A$, finite traces $s, t \in A^*$, infinite traces $u \in A^\omega$,*

$$\begin{array}{rcl}
P_{\langle\rangle} * H & \widehat{=} & \textbf{skip} \\
P_{\langle a\rangle} * H & \widehat{=} & P_a\ ;\ IT_H \\
P_{st} * H & \widehat{=} & (P_s * H)\ ;\ (P_t * H) \\
P_u * H & \widehat{=} & (\ ;\ i \mid i \in \mathbb{N} \bullet P_{u_i} * H).
\end{array}$$

□

Note that a trace is not preceded by IT_H. We write $P_a * H$ for $P_{\langle a\rangle} * H$. To deal with refusal sets we define $gd_H(P_X)$ as follows:

Definition 4.7 *For $X \subseteq A$,*

$$gd_H(P_X) \;\widehat{=}\; (\vee\ a \in X \bullet gd(IT_H\ ;\ P_a)).$$

□

The semantics of an action system with internal actions is given by the following definition:

Definition 4.8 *For action system $P = (A, v, P_\imath, P_A, P_H)$*

$$\{\![P]\!\} \;\widehat{=}\; (A,\ \mathcal{F}\{\![P]\!\},\ \mathcal{D}\{\![P]\!\},\ \mathcal{I}\{\![P]\!\})$$

where $\mathcal{F}\{\![P]\!\}$ are those $s \in A^*, X \subseteq A$ satisfying

$$\overline{wp}(P_{\langle \imath \rangle s} * H, \neg\, gd_H(P_X))$$

$\mathcal{D}\{\![P]\!\}$ are those $s \in A^*$ satisfying

$$\neg\, halt(P_{\langle \imath \rangle s} * H)$$

$\mathcal{I}\{\![P]\!\}$ are those $u \in A^\omega$ satisfying

$$\overline{inf}(P_{\langle \imath \rangle u} * H).$$

□

Theorem 4.9 *For an action system P with internal actions, $\{\![P]\!\}$ is well-formed in the infinite-traces model.*

Proof: Given P, construct the action system P', as follows:

$$P' \;\widehat{=}\; (A,\ v,\ (P_\imath\ ;\ IT_H),\ \{\ (IT_H\ ;\ P_a\ ;\ IT_H) \mid a \in A\ \}\ ).$$

P' is an action system with no internal actions, and so $\{\![P']\!\}$ is well-formed by Theorem 3.12. It is easy to show that P' has exactly the same failures, divergences and infinites as P, hence $\{\![P]\!\}$ is also well-formed.
□

Note: The action system P', defined in the proof of Theorem 4.9, was our original definition of the effect of internal actions on action system P. Definitions 4.6 and 4.7 were derived from this. For example, we required that $P_{\langle \imath, a\rangle} * H = P'_{\langle \imath, a\rangle}$ and we have

$$\begin{aligned} P'_{\langle \imath, a\rangle} &= (P_\imath\ ;\ IT_H)\ ;\ (IT_H\ ;\ P_a\ ;\ IT_H) \\ &= P_\imath\ ;\ IT_H\ ;\ P_a\ ;\ IT_H \qquad\qquad \textit{since } IT_H\ ;\ IT_H = IT_H. \end{aligned}$$

For refusal sets, we required that $gd_H(P_X) \equiv gd(P'_X)$, and we have

$$\begin{aligned} gd(P'_X) &\equiv (\vee\ a \in X \bullet gd(IT_H\ ;\ P_a\ ;\ IT_H)) \\ &\equiv (\vee\ a \in X \bullet gd(IT_H\ ;\ P_a)) \qquad\qquad \textit{since } gd(IT_H) \equiv \textit{true}. \end{aligned}$$

Constructing P' is difficult in practice, since IT_H can be difficult to calculate. For this reason, we keep internal actions separate and develop operators and simulation rules for action systems with internal actions.

$$
\mathit{M1} \;\widehat{=}\; \left(\begin{array}{l}
\mathbf{var}\ n : \mathbb{N} \\
\mathbf{initially}\ \ n := 10 \\
a :- \qquad n = 0 \rightarrow n := 10 \\
\mathbf{internal}\ \ b :- \qquad n > 0 \rightarrow n := n - 1
\end{array}\right)
$$

$$
\mathcal{M}\mathit{1} \;\widehat{=}\; (\mu\, X \bullet a \rightarrow X)
$$

Figure 4.A: Action system and corresponding CSP process.

4.4 Example

Figure 4.A specifies an action system with visible action a and internal action b. The internal action continually decrements n until $n = 0$, so that action a will eventually be enabled. Using Definition 4.8 it can be shown that $\mathit{M1}$ has the same semantics as CSP process $\mathcal{M}\mathit{1}$ of Figure 4.A, which always offers a-events.

One may be tempted to believe, because of the nondeterminism of the iterate construct, that the internal action needn't be executed enough times to decrease n to zero, in which case a will not be enabled and $\mathit{M1}$ will deadlock. In fact, because of the definition of $gd_H(P_X)$ and the coercion property of miracles, in this case the internal action is forced to iterate until a is enabled: consider

$$
IT_b \;;\; \mathit{M1}_a \quad = \quad \mathbf{it}\ n > 0 \rightarrow n := n - 1\ \mathbf{ti}\ \;;\; n = 0 \rightarrow n := 10.
$$

Here the guard $n = 0$ forces the preceding iterate-construct to take a path which decrements n to zero so that $\mathit{M1}$ cannot refuse to offer a. It can be shown by calculation that

$$
gd(IT_b \;;\; \mathit{M1}_a) \;\equiv\; n \geq 0.
$$

The coercion property isn't always able to force the offering of events. If the a action of $\mathit{M1}$ was instead

$$
a :- \qquad n = 1 \rightarrow n := 10,
$$

then because of the definition of the failures of an action system (Definition 4.8), where iteration of internal actions is always allowed before testing for refusal, action b may decrease n all the way to zero and cause deadlock.

$$\mathit{M2} \;\widehat{=}\; \left(\begin{array}{lll} \textbf{var}\ n : \mathbb{N} & & \\ \textbf{initially} & n := 10 & \\ a :- & n = 0 \rightarrow n := 10 & \\ b :- & n > 0 \rightarrow n := n - 1 & \end{array}\right)$$

$$\mathcal{M}\mathit{2} \;\widehat{=}\; (\mu\, X \bullet \overbrace{b \rightarrow \cdots \rightarrow b \rightarrow}^{10 \text{ times}} a \rightarrow X)$$

Figure 4.B: *M2* and corresponding CSP process.

4.5 Hiding Operator for Action Systems

The hiding operator for action systems simply internalises the actions to be hidden:

Definition 4.10 *For action system* $P = (A, v, P_\imath, P_A, P_H)$*, label set* $C \subseteq A$*,*

$$P \setminus C \;\widehat{=}\; (A - C,\ v,\ P_\imath,\ P_{A-C},\ P_{H \cup C}).$$

□

As an example of hiding, consider *M2* of Figure 4.B. *M2* is an action system with no internal actions and it can be shown that it behaves as the CSP process $\mathcal{M}\mathit{2}$ of Figure 4.B. Immediately it can be seen that action system *M1* of Figure 4.A is exactly $\mathit{M2} \setminus b$. Correspondingly, using the algebraic laws of CSP, it can be shown that the process $\mathcal{M}\mathit{2} \setminus b$ is equal to the process $\mathcal{M}\mathit{1}$ specified in Figure 4.A.

4.6 Correspondence of the Hiding Operators

The semantic function for CSP hiding is given by the following definition, which is based on that in [Ros88]:

Definition 4.11 *For process model* $\Theta = (A, F, D, I)$*,* $C \subseteq A$*, let* $B \,\widehat{=}\, (A - C)$*. Then*

$$\Theta \setminus C \;\widehat{=}\; (B,\ \Theta \setminus_{\mathcal{F}} C,\ \ \Theta \setminus_{\mathcal{D}} C,\ \ \Theta \setminus_{\mathcal{I}} C)$$

where $\Theta \setminus_{\mathcal{F}} C$ *are those* $r \in B^*$*,* $X \subseteq B$ *satisfying*

$$\begin{array}{l} (\vee\ s \in A^* \mid s \restriction B = r \bullet (s, X \cup C) \in F) \\ \vee\ (\vee\ u \in A^\omega \mid u \restriction B \leq r \bullet u \in I) \end{array}$$

$\Theta \setminus_{\mathcal{D}} C$ *are those* $r \in B^*$ *satisfying*

$$(\vee\ s \in A^* \mid s \upharpoonright B = r \bullet s \in D)$$
$$\vee\ (\vee\ u \in A^\omega \mid u \upharpoonright B \leq r \bullet u \in I)$$

$\Theta \setminus_{\mathcal{I}} C$ *are those* $v \in B^\omega$ *satisfying*

$$(\vee\ u \in A^\omega \mid u \upharpoonright B \leq v \bullet u \in I).$$

□

Thus, (r, X) is a failure of $\Theta \setminus C$ if there is some trace of A-events s such that $s \upharpoonright B = r$ and $(s, X \cup C)$ is a failure of Θ. Alternatively, (r, X) is a failure of $\Theta \setminus C$ if r is a divergence introduced by hiding C, i.e. there is some infinite trace of A-events u that, when restricted to B-events, is a finite subsequence of r. Similarly for divergences and infinites of $\Theta \setminus C$.

The main result of this section is Theorem 4.17 which shows that action-system hiding (Definition 4.10) corresponds to CSP hiding (Definition 4.11). To reach this theorem, a series of lemmas are presented which express certain constructs of action system $P \setminus C$ in terms of action system P. Each of these lemmas is proven in Appendix B. For the remainder of this section assume that $Q \mathrel{\hat=} P \setminus C$, $B \mathrel{\hat=} A - C$, and $G \mathrel{\hat=} H \cup C$. We have by Definition 4.10

$$Q \;=\; (B, v, Q_\imath, Q_B, Q_G) \;=\; (A - C, v, P_\imath, P_{A-C}, P_{H \cup C}).$$

The statement $Q_b * G$ describes the effect of executing action Q_b followed by iterating over the internal actions of Q. Using Lemma 4.5, the following can be shown:

Lemma 4.12 *For action* $b \in B \cup \{\imath\}$,

$$Q_b * G \quad = \quad P_b * H \;\; ; \;\; \textbf{it}\ ([]\ c \in C \bullet P_c * H)\ \textbf{ti}.$$

□

Combining this lemma and Lemma 4.3 we get the following conjugate weakest-precondition equivalence:

Lemma 4.13 *For action* $b \in B \cup \{\imath\}$, *predicate* ϕ,

$$\overline{wp}(Q_b * G, \phi) \;\equiv\; (\vee\ s \in C^* \bullet \overline{wp}(P_{\langle b \rangle s} * H, \phi))$$
$$\vee\ (\vee\ u \in C^\omega \bullet \overline{inf}(P_{\langle b \rangle u} * H)).$$

□

By induction over finite traces $r \in B^*$ we then get the following lemma which describes the effect of visible trace r in Q in terms of traces of P:

Lemma 4.14 *For trace $r \in B^*$, predicate ϕ,*

$$\begin{aligned}\overline{wp}(Q_{\langle\imath\rangle r} * G, \phi) \;\;\equiv\;\; & (\vee\, s \in A^* \mid s\restriction B = r \bullet \overline{wp}(P_{\langle\imath\rangle s} * H, \phi)) \\ & \vee\, (\vee\, u \in A^\omega \mid u\restriction B \leq r \bullet \overline{inf}(P_{\langle\imath\rangle u} * H)).\end{aligned}$$

□

Note: in the second disjunct of the right-hand side of this lemma, $u\restriction B \leq r$ means that u ends in an infinite sequence of C-events.

There is a similar lemma for infinite traces:

Lemma 4.15 *For trace $v \in B^\omega$,*

$$\overline{inf}(Q_{\langle\imath\rangle v} * G) \;\;\equiv\;\; (\vee\, u \in A^\omega \mid u\restriction B \leq v \bullet \overline{inf}(P_{\langle\imath\rangle u} * H)).$$

□

To deal with refusal sets we have the following lemma which relates guardedness conditions in Q to guardedness conditions in P:

Lemma 4.16 *For label set $X \subseteq B$,*

$$\overline{wp}(IT_G, \neg\, gd_G(Q_X)) \;\equiv\; \overline{wp}(IT_G, \neg\, gd_H(P_{X\cup C})).$$

□

These lemmas are now used to prove that action-system hiding corresponds to CSP hiding:

Theorem 4.17 *For action system P, $C \subseteq \alpha P$, $\quad [\![P \setminus C]\!] = [\![P]\!] \setminus C$ in the infinite-traces model.*

Proof: To show failures equality we have, for $r \in B^*$, $X \subseteq B$,

$$
\begin{array}{lll}
& (r, X) \in \mathcal{F}\{\!\!\{P \setminus C\}\!\!\} & \\
\equiv & \overline{wp}(Q_{\langle\imath\rangle r} * G, \neg\, gd_G(Q_X)) & \textit{Definition 4.8} \\
\equiv & \overline{wp}(Q_{\langle\imath\rangle r} * G, \overline{wp}(IT_G, \neg\, gd_G(Q_X)\ )) & \textit{Lemma 4.4} \\
\equiv & \overline{wp}(Q_{\langle\imath\rangle r} * G, \overline{wp}(IT_G, \neg\, gd_H(P_{X \cup C})\ )) & \textit{Lemma 4.16} \\
\equiv & \overline{wp}(Q_{\langle\imath\rangle r} * G, \neg\, gd_H(P_{X \cup C})\ ) & \textit{Lemma 4.4} \\
\equiv & (\vee\ s \in A^* \mid s{\restriction}B = r \bullet \overline{wp}(P_{\langle\imath\rangle s} * H, \neg\, gd_H(P_{X \cup C})\ )) & \\
& \vee\ (\vee\ u \in A^\omega \mid u{\restriction}B \leq r \bullet \overline{inf}(P_{\langle\imath\rangle u} * H)) & \textit{Lemma 4.14} \\
\equiv & (\vee\ s \in A^* \mid s{\restriction}B = r \bullet (s, X \cup C) \in \mathcal{F}\{\!\!\{P\}\!\!\}) & \\
& \vee\ (\vee\ u \in A^\omega \mid u{\restriction}B \leq r \bullet u \in \mathcal{I}\{\!\!\{P\}\!\!\}) & \textit{Definition 4.8} \\
\equiv & (r, X) \in \{\!\!\{P\}\!\!\} \setminus_{\mathcal{F}} C & \textit{Definition 4.11.}
\end{array}
$$

Similarly, using Lemma 4.14 with $\phi \equiv \mathit{false}$ we get, for $r \in B^*$,

$$r \in \mathcal{D}\{\!\!\{P \setminus C\}\!\!\} \ \equiv\ r \in \{\!\!\{P\}\!\!\} \setminus_{\mathcal{D}} C,$$

and using Lemma 4.15 we get, for $v \in B^\omega$,

$$v \in \mathcal{I}\{\!\!\{P \setminus C\}\!\!\} \ \equiv\ v \in \{\!\!\{P\}\!\!\} \setminus_{\mathcal{I}} C.$$

□

The proof of Theorem 4.17 may be summarised as follows: Theorem 4.2 allows us to separate the finite and infinite behaviour of **it** S **ti**; this then allows us to describe the behaviour of action system $P \setminus C$ in terms of the finite and infinite behaviour of P, in the same way that the behaviour of CSP process $\mathcal{P} \setminus C$ is defined in terms of the finite and infinite behaviour of $\mathcal{P}$.

4.7 Failures-Divergences Model

In the introduction to Chapter 3, it was claimed that the correspondence between unboundedly-nondeterministic action-systems and the CSP failures-divergences model is lost after application of the hiding operator. We can now verify this claim. Consider, again, the action system *L2* of Figure 3.B, with the unboundedly-nondeterministic initialisation:

$$L2 \;\;\widehat{=}\;\; \begin{pmatrix} \textbf{var}\ n : \mathbb{N} & \\ \textbf{initially}\ \ n :\in \mathbb{N} & \\ a :- \qquad n > 0 \rightarrow n := n - 1 & \end{pmatrix}.$$

Using Definition 4.8 it can be shown that $L2$ has the same failures-divergences semantics as the CSP process $\mathcal{L}2$ of Figure 3.A. Recall that in the failures-divergences model $\mathcal{L}2 \setminus \{a\} = CHAOS$. However, using Definition 4.8 it can be shown that action system $L2 \setminus \{a\}$ has the same failures and divergences as the process $STOP$. Thus, in the failures-divergences model, $L2 \setminus \{a\}$ and $\mathcal{L}2 \setminus \{a\}$ do not correspond, even though $L2$ and $\mathcal{L}2$ do.

In the case that statement S is boundedly nondeterministic, it can be shown that

$$wp(\textbf{it}\ S\ \textbf{ti}, \phi) \;\;\equiv\;\; (\wedge\, i \in \mathbb{N} \bullet wp(S^i, \phi)) \;\wedge\; (\vee\, i \in \mathbb{N} \bullet wp(S^i, \mathit{false})).$$

Given this equality, the hiding operators can be shown to correspond in the failures-divergences model, provided we assume that action systems are boundedly nondeterministic and contain only a finite number of internal actions. Of course Theorem 4.17 is more general, since it assumes action systems may be unboundedly nondeterministic and the number of actions internalised may be infinite. This strengthens our claim that the infinite-traces model for CSP is a more appropriate link between action systems and CSP.

Note: Back introduced a similar characterisation of **do** S **od** for boundedly-nondeterministic S in [Bac92a]:

$$wp(\textbf{do}\ S\ \textbf{od}, \phi) \;\;\equiv\;\; (\wedge\, i \in \mathbb{N} \bullet wp(S^i[\neg\, gd(S)], \phi)) \;\wedge\; (\vee\, i \in \mathbb{N} \bullet wp(S^i, \mathit{false})).$$

This may be generalised to unbounded nondeterminism using our $\overline{inf}$:

$$wp(\textbf{do}\ S\ \textbf{od}, \phi) \;\;\equiv\;\; (\wedge\, i \in \mathbb{N} \bullet wp(S^i[\neg\, gd(S)], \phi)) \;\wedge\; \neg\, \overline{inf}(S^\infty).$$

4.8 Properties of the Hiding Operator

Because of Theorem 4.17, any properties enjoyed by the CSP hiding-operator will also be enjoyed by the action-system hiding-operator. An important property of the CSP hiding-operator is its monotonicity w.r.t. the refinement ordering: from [Ros88] we have for process models Θ_1, Θ_2, event set C:

$$\Theta_1 \sqsubseteq \Theta_2 \quad \text{implies} \quad \Theta_1 \setminus C \;\sqsubseteq\; \Theta_2 \setminus C.$$

So immediately we get, for action systems P, Q:

$$P \sqsubseteq Q \quad \text{implies} \quad P \setminus C \ \sqsubseteq \ Q \setminus C.$$

Another property of CSP hiding means that the order in which events are hidden is not important. For action systems we have

$$(P \setminus C) \setminus D \ = \ (P \setminus D) \setminus C \ = \ P \setminus (C \cup D).$$

4.9 Refinement and Internal Actions

Simulation for action systems was introduced in Section 2.8. There is a similar definition for action systems with internal actions:

Definition 4.18 *A simulation, with representation function rep, is a relation on action systems, denoted $\sqsubseteq_{rep}$, such that for action systems*

$$P = (A,\ v,\ P_\imath,\ P_A,\ P_H) \quad \textit{and} \quad Q = (A,\ w,\ Q_\imath,\ Q_A,\ Q_G),$$

$P \sqsubseteq_{rep} Q$ if each of the following conditions hold:

1. $P_\imath * H \preceq'_{rep} Q_\imath * G$
2. $P_a * H \preceq_{rep} Q_a * G, \qquad \textit{each } a \in A$
3. $rep(gd_H(P_X)) \ \Rrightarrow \ gd_G(Q_X), \qquad \textit{each } X \subseteq A.$

□

Simulation is sound provided *rep* is $\vee$-continuous:

Theorem 4.19 *For $\vee$-continuous rep, if $P \sqsubseteq_{rep} Q$, then $P \sqsubseteq Q$ in the infinite-traces model.*

Proof of this theorem is exactly the same as the proof of Theorem 3.13. If *rep* is disjunctive, then the progress condition can be simplified to (cf. Lemma 2.19):

$$rep(gd_H(P_a)) \ \Rrightarrow \ gd_G(Q_a), \qquad \text{each } a \in A.$$

4.10 Simplified Simulation-Rules

We will now show that the case where the abstract action system P has no internal actions leads to a simplified set of simulation rules. A well-foundedness theorem for IT-loops is introduced to ensure non-divergence of the internal actions of the concrete action system Q which is similar to Dijkstra's well-foundedness theorem for DO-loops [DS90]. A set WF, with irreflexive partial order $<$, is *well-founded* if each non-empty subset of WF contains a minimal element under $<$. For example, the natural numbers with the usual ordering, or the cartesian product of two or more well-founded sets with lexicographic ordering, all form well-founded sets.

Theorem 4.20 *If WF is some well-founded set ordered by $<$, E is some expression in the state-variables, and for some predicate ϕ,*

$$\begin{array}{rcl} \phi & \Rrightarrow & E \in WF \\ E = e \ \wedge \ \phi & \Rrightarrow & wp(S, E < e \ \wedge \ \phi) \end{array}$$

then

$$\phi \Rrightarrow wp(\mathbf{it}\ S\ \mathbf{ti}, \phi)$$

$\square$

E is a variant function, and the antecedent requires that S is guaranteed to preserve ϕ and to decrease E. Since E is always an element of WF, and so cannot be decreased forever, S must eventually become miraculous if it is repeatedly executed, i.e. **it** S **ti** must terminate. Theorem 4.20 is proven in Appendix B.

The well-foundedness theorem is used to simplify the data-refinement conditions as follows:

Corollary 4.21 *To show that Conditions 1 and 2 of Definition 4.18 hold when $P_H = \{\}$, it is sufficient to show for some well-founded set WF ordered by $<$, and some variant E, that the following conditions hold:*

1. $P_\imath \preceq'_{rep} Q_\imath$
2. $P_a \preceq_{rep} Q_a$, *each* $a \in A$
3. $\mathbf{skip} \preceq_{rep} Q_g$, *each* $g \in G$
4. $rep(true) \ \Rrightarrow \ E \in WF$
5. $E = e \wedge rep(true) \ \Rrightarrow \ wp(Q_g, E < e)$, *each* $g \in G$.

Proof: From Condition 4 above we get for predicate ϕ

$$rep(\phi) \Rrightarrow rep(true) \Rrightarrow E \in WF.$$

From Conditions 3 and 5 above we get

$$\begin{array}{ll} & E = e \wedge rep(\phi) \\ \Rrightarrow & (\wedge\ g \in G \bullet wp(Q_g, E < e) \wedge wp(Q_g, rep(\phi))) \\ \Rrightarrow & wp(\ ([]\ g \in G \bullet Q_g)\ , E < e \wedge rep(\phi)). \end{array}$$

Hence by Theorem 4.20 $rep(\phi) \Rrightarrow wp(IT_G, rep(\phi))$, that is

$$\textbf{skip} \;\preceq_{rep}\; IT_G. \qquad \text{(iii)}$$

Now Condition 2 of Definition 4.18 holds:

$$\begin{array}{rcll} P_a * H & = & P_a & \textit{since } H = \{\} \\ & \preceq_{rep} & Q_a & \textit{since } P_a \preceq_{rep} Q_a \\ & \preceq_{rep} & Q_a \,;\, IT_G & \text{(iii) } \textit{and monotonicity of wp} \\ & = & Q_a * G. & \end{array}$$

Similarly for Condition 1 of Definition 4.18.
□

The progress condition of Definition 4.18 is

$$rep(gd_H(P_X)) \;\Rrightarrow\; gd_G(Q_X), \qquad \text{each } X \subseteq A.$$

For $X = \{\}$, this follows from strictness of *rep*. In the case where H is empty and X is non-empty, the condition can be rewritten as

$$rep(gd(P_X)) \;\Rrightarrow\; \overline{wp}(IT_G, gd(Q_X)), \qquad \text{each } X \subseteq A.$$

This means that any concrete state corresponding to $gd(P_X)$ must enable some iteration of G actions which could establish $gd(Q_X)$. To simplify the condition, we introduce the following theorem:

Theorem 4.22 *For statement S,* $wp(\textbf{it } S \textbf{ ti}, \phi \vee gd(S)) \;\Rrightarrow\; \overline{wp}(\textbf{it } S \textbf{ ti}, \phi)$.

Proof: It is easy to show that $wp(\textbf{it } S \textbf{ ti}, \phi \vee gd(S))$ is a solution of

$$X \;\Rrightarrow\; \phi \vee \overline{wp}(S, X). \qquad \text{(iv)}$$

The theorem now follows since $\overline{wp}(\textbf{it } S \textbf{ ti}, \phi)$ is the greatest solution of (iv) by the the definition of **it** S **ti** and the Knaster-Tarski Theorem.
□

Corollary 4.23 *To show Condition 3 of Definition 4.18 holds when $H = \{\}$, it is sufficient to show that*

1. $\textbf{skip} \;\preceq_{rep}\; IT_G$
2. $rep(gd(P_X)) \;\Rrightarrow\; gd(Q_X) \vee gd(Q_G), \qquad \textit{each } X \subseteq A.$

Proof: If $X = \{\}$, then Condition 3 follows by strictness of *rep*. For $X \neq \{\}$,

$$\begin{array}{lll} & rep(gd_H(P_X)) & \\ \Rrightarrow & rep(gd(P_X)) & \textit{since } H = \{\} \\ \Rrightarrow & wp(IT_G, rep(gd(P_X))) & \textit{since } \mathbf{skip} \preceq_{rep} IT_G \\ \Rrightarrow & wp(IT_G, gd(Q_X) \vee gd(Q_G)) & \textit{assumption} \\ \Rrightarrow & \overline{wp}(IT_G, gd(Q_X)) & \textit{Theorem 4.22} \\ \Rrightarrow & gd_G(Q_X) & \textit{since } X \neq \{\} \end{array}$$

□

All the derived conditions which are sufficient to show that $P \sqsubseteq_{rep} Q$ in the case that $P_H = \{\}$ are collected in Figure 4.C. We shall refer to Figure 4.C as the *hiding-refinement* rule, and refer to Conditions 1, 2, and 3 as data-refinement conditions, Conditions 4 and 5 as non-divergence conditions, and Condition 6 as a progress condition.

In certain cases, because of the monotonicity of the hiding operator, the hiding-refinement rule can be used even if the abstract action system P has a non-empty set of internal actions: assume we have two action systems

$$P = (A, v, P_\imath, P_A, P_H) \qquad Q = (A, w, Q_\imath, Q_A, Q_{H \cup H'}).$$

Notice how each internal action of P has a correspondingly labelled internal action in Q. To show $P \sqsubseteq Q$, construct action systems P' and Q' as follows:

$$P' \mathrel{\widehat{=}} (A \cup H, v, P_\imath, P_{A \cup H}, \{\}) \qquad Q' \mathrel{\widehat{=}} (A \cup H, w, Q_\imath, Q_{A \cup H}, Q_{H'}).$$

Now, using the hiding refinement rule, show that $P' \sqsubseteq_{rep} Q'$. Hence, we have that $P' \sqsubseteq Q'$, and by monotonicity

$$P' \setminus H \;\; \sqsubseteq \;\; Q' \setminus H.$$

That is $P \sqsubseteq Q$.

4.11 Example Refinement

In Section 4.4, it was claimed that *M1* of Figure 4.A is a correct implementation of process $\mathcal{M}1$ by appeal to Definition 4.8. Of course calculating the semantic model of an action system is not a practical verification-technique. Figure 4.D specifies an action system which clearly always offers an a-action. Using the hiding-refinement

For action systems

$$P = (A, v, P_\imath, P_A, \{\}) \quad and \quad Q = (A, w, Q_\imath, Q_A, Q_G)$$

$P \sqsubseteq_{rep} Q$ if the following conditions hold, for $\vee$-continuous representation-function rep, well-founded set WF with ordering $<$, and variant E,

1. $P_\imath \preceq'_{rep} Q_\imath$
2. $P_a \preceq_{rep} Q_a$, *each* $a \in A$
3. $\mathbf{skip} \preceq_{rep} Q_g$, *each* $g \in G$
4. $rep(true) \;\Rrightarrow\; E \in WF$
5. $rep(true) \wedge E = e \;\Rrightarrow\; wp(Q_g, E < e)$, *each* $g \in G$
6. $rep(gd(P_X)) \;\Rrightarrow\; gd(Q_X) \vee gd(Q_G)$, *each* $X \subseteq A$

Furthermore, if rep is disjunctive then 6 can be replaced by

6'. $rep(gd(P_a)) \;\Rrightarrow\; gd(Q_a) \vee gd(Q_G)$, *each* $a \in A$.

Figure 4.C: Hiding-refinement rule.

rule, we shall demonstrate that $M3 \sqsubseteq M1$, i.e. that $M1$ is a correct implementation of $M3$.

To show $M3 \sqsubseteq M1$, the following disjunctive representation-function is used:

$$rep(\phi) \;\widehat{=}\; 0 \leq n \leq 10.$$

Demonstrating the data-refinement conditions is straightforward. For example, Condition 3:

$$\begin{array}{lrcl}
 & \mathbf{skip} & \preceq_{rep} & M1_b \\
\text{iff} & rep(\phi) & \Rrightarrow & wp(n > 0 \rightarrow n := n-1,\ rep(\phi)\ ) \\
\text{iff} & rep(\phi) & \Rrightarrow & n > 0 \Rightarrow rep(\phi)[n \backslash n-1] \\
\text{iff} & (0 \leq n \leq 10) & \Rrightarrow & n > 0 \Rightarrow (0 \leq n-1 \leq 10) \\
\text{iff} & & true. &
\end{array}$$

As a variant function we simply use n. It is easy to see that n is always in the

$$M3 \;\widehat{=}\; \left(\begin{array}{ll} \textbf{initially} & \textbf{skip} \\ a :- & \textbf{skip} \end{array} \right)$$

Figure 4.D: Action system that always offers a-action.

well-founded set 0..10, and that the internal action of $M1$ always decreases n:

$$\begin{array}{rcl} rep(true) & \Rrightarrow & n \in 0..10 \\ rep(true) \wedge n = n_0 & \Rrightarrow & wp(M1_b, n < n_0). \end{array}$$

Thus, the non-divergence conditions are satisfied. Finally, the progress condition is satisfied:

$$\begin{array}{ll} & rep(gd(M3_a)) \\ \Rrightarrow & 0 \leq n \leq 10 \\ \Rrightarrow & n = 0 \vee n > 0 \\ \Rrightarrow & gd(M1_a) \vee gd(M1_b). \end{array}$$

In Chapter 7 we shall present more examples of action-system refinement, and provide some detail on how the representation function is chosen.

4.12 Remarks

Our hiding operator for action systems resembles He Jifeng's hiding operator for transition systems [He89]. Suppose the transition labelled c is to be hidden in T. Then, the visible transitions of $T \setminus \{c\}$ are calculated as follows:

$$(T \setminus \{c\})_a \;\widehat{=}\; (T_c)^*;\; T_a;\; (T_c)^* \;\cup\; NewDiv.$$

Here $(T_c)^*$ is the reflexive, transitive closure of T_c and $NewDiv$ is a relation that determines if T_c introduces divergence when hidden. $NewDiv$ is calculated using the following: T_c may iterate at most n times if $(T_c)^{n+1}$ is empty, and T_C does not cause divergence if there is some $n \geq 0$ such that $(T_c)^{n+1}$ is empty.

Now the construction of $T \setminus \{c\}$ resembles the construction of the action system P' described in Theorem 4.9, since $(T_c)^*$ resembles **it** T_c **ti**. The difference is that $(T_c)^*$ doesn't deal with infinite iteration, whereas **it** P_c **ti** does. Thus, He Jifeng needs to add *NewDiv* to the definition. By giving a weakest-precondition semantics to the iterate construct, we don't need to deal with the introduction of divergence separately.

Another feature of He Jifeng's hiding operator is its treatment of unbounded nondeterminism. Since infinite iteration is interpolated from finite iterations, a transition that may iterate an unbounded number of times will be interpreted as being able to iterate infinitely. For example, if *T2* is the transition system corresponding to the action system *L2* of Figure 3.B, then $T2 \setminus \{a\} = CHAOS$. Also, He Jifeng's hiding operator allows only a finite number of transitions to be hidden. In [He90], a hiding-refinement rule for transition systems, that is similar to our rule of Figure 4.C, is developed.

Our hiding-refinement rule is also similar to Back's rule for stuttering refinement in action systems (see Page 13). Condition 3 of Figure 4.C is akin to saying that the internal actions of the concrete action system should only cause stuttering steps, while the non-divergence conditions ensure that Q only causes a finite number of stuttering steps. A slight difference is that Back's non-divergence condition is written in terms of termination of **do** S **od**, whereas our conditions are used to show termination of **it** S **ti**. However, it can be shown that **do** S **od** and **it** S **ti** have exactly the same termination condition, i.e. $halt(\mathbf{do}\ S\ \mathbf{od}) \equiv halt(\mathbf{it}\ S\ \mathbf{ti})$. A more significant difference is that only our hiding-refinement rule distinguishes internal and external choice (cf. Section 2.9).

As mentioned in Chapter 1, CCS [Mil89] and ACP [BK85] differ from CSP in their treatment of hiding. Recall that, in both CCS and ACP, possible infinite occurrence of internal events does not cause a process to behave chaotically. For example, $(\mu X \bullet \tau.X) \approx NIL$ in CCS. Event hiding like that of CCS or ACP would be quite difficult to implement in action systems. It would require addition of a mechanism that prevents infinite iteration of internal actions, though allows enough iteration for progress of the visible actions to be possible. As this chapter shows, implementation of CSP hiding in action systems is more straightforward.

Chapter 5

Parallel Composition

In this chapter, parallel composition for action systems is defined and shown to correspond to CSP parallel-composition. To achieve this, we introduce a parallel operator for statements. We also demonstrate how refinement steps involving parallel decomposition can be achieved.

5.1 Introduction

Recall that the CSP process $\mathcal{P} \parallel \mathcal{Q}$ represents the parallel composition of the processes $\mathcal{P}$ and $\mathcal{Q}$. Operationally, $\mathcal{P}$ and $\mathcal{Q}$ interact by synchronising over common events in $\alpha\mathcal{P} \cap \alpha\mathcal{Q}$, while events not in $\alpha\mathcal{P} \cap \alpha\mathcal{Q}$ can occur independently. An event common to both $\mathcal{P}$ and $\mathcal{Q}$ becomes a single event in $\mathcal{P} \parallel \mathcal{Q}$, and can be offered by $\mathcal{P} \parallel \mathcal{Q}$ only when both $\mathcal{P}$ and $\mathcal{Q}$ are prepared to offer it.

As an example consider the CSP processes

$$\mathcal{N}\mathit{1} \mathrel{\widehat{=}} (\mu\, X \bullet a \to c \to X), \qquad \mathcal{N}\mathit{2} \mathrel{\widehat{=}} (\mu\, X \bullet b \to c \to X).$$

Here c is the only event common to $\mathcal{N}\mathit{1}$ and $\mathcal{N}\mathit{2}$, and using the algebraic laws of CSP it can be shown that

$$\begin{aligned} \mathcal{N}\mathit{1} \parallel \mathcal{N}\mathit{2} \;=\; (\mu\, X \bullet \quad & a \to b \to c \to X \\ \;[\!]\; & b \to a \to c \to X \;). \end{aligned}$$

That is, the a or the b events can occur in either order, then both processes must synchronise on event c.

Correspondingly, we write the parallel composition of action systems P and Q as $P \parallel Q$. The actions of $P \parallel Q$ are formed from the actions of P and Q: commonly-labelled actions are placed in parallel, while independently-labelled actions remain independent. An important assumption we make is that action systems P and Q have no common state-variables, so that interaction between P

and Q is based purely on synchronisation. Before formally defining $P \parallel Q$ we introduce an operator for combining individual actions (statements) in parallel. This operator is only valid for statements with independent state-variables. Properties of the operator will be used in Section 5.4 to prove the correspondence between action-system and CSP parallel-composition.

5.2 Parallel Composition of Statements

For v-statement S and w-statement T in our specification language, we write $S \parallel T$ for their parallel composition provided v and w are distinct. Since v and w may represent collections of variables, we say that v and w are distinct if they have no common components. Assume that v and w are distinct for the remainder of this chapter.

Rather than completely defining the parallel operator for statements we assume that it preserves monotonicity and positive conjunctivity, and satisfies the following disjunctivity property:

Property 5.1 *For v-statement S, w-statement T, v-predicate ϕ, w-predicate ψ,*

$$\begin{aligned} wp(S \parallel T, \phi \vee \psi) \quad \equiv \quad gd(S) \wedge gd(T) \Rightarrow \; & (wp(S, \phi) \wedge halt(T) \\ & \vee\, halt(S) \wedge wp(T, \psi)). \end{aligned}$$

□

This property means that $S \parallel T$ is enabled only when both S and T are enabled. As we shall see in Section 5.4, monotonicity, positive conjunctivity and Property 5.1 are sufficient to prove the correspondence between action system and CSP parallel-composition. This is because any postcondition of interest to us can be written as two separate predicates in the state variables of the respective action systems.

The following operator can be shown to satisfy Property 5.1:

Definition 5.2 *For v-statement S, w-statement T,*

$$S \otimes T \quad \widehat{=} \quad gd(S) \wedge gd(T) \rightarrow S \; ; \; T.$$

□

Note: We could have used Definition 5.2 as the definition of the parallel operator. However, this gives more than is required. For example, we can calculate $wp(S \otimes T, v = w)$. Property 5.1 highlights what is required of the operator to show the correspondence between action-system and CSP parallel-composition.

$$N1 \;\widehat{=}\; \left(\begin{array}{ll} \textbf{var } m : \mathbb{N} & \\ \textbf{initially } \; m := 0 & \\ a :- & m = 0 \rightarrow m := 1 \\ c :- & m = 1 \rightarrow m := 0 \end{array}\right) \qquad N2 \;\widehat{=}\; \left(\begin{array}{ll} \textbf{var } n : \mathbb{N} & \\ \textbf{initially } \; n := 0 & \\ b :- & n = 0 \rightarrow n := 1 \\ c :- & n = 1 \rightarrow n := 0 \end{array}\right)$$

$$N1 \parallel N2 \;=\; \left(\begin{array}{ll} \textbf{var } m, n : \mathbb{N} & \\ \textbf{initially } \; m := 0 \;;\; n := 0 & \\ a :- & m = 0 \rightarrow m := 1 \\ b :- & n = 0 \rightarrow n := 1 \\ c :- & m = 1 \wedge n = 1 \rightarrow m := 0 \;;\; n := 0 \end{array}\right)$$

Figure 5.A: Example parallel action systems.

5.3 Parallel Composition of Action Systems

Parallel composition of action systems is defined as follows:

Definition 5.3 *For action systems*

$$P = (A, v, P_\imath, P_A, P_H) \quad \textit{and} \quad Q = (B, w, Q_\imath, Q_B, Q_G)$$

$$P \parallel Q \;\widehat{=}\; (A \cup B,\;\; (v, w),\;\; P_\imath \parallel Q_\imath,\;\; par(P_A, Q_B),\;\; P_H \cup Q_G)$$

where $\quad par(P_A, Q_B) \;\widehat{=}\; P_{A-B} \;\cup\; Q_{B-A} \;\cup\; \{\; P_c \parallel Q_c \mid c \in A \cap B \;\}.$
□

The alphabet of $P \parallel Q$ is the union of A and B. The state variable is the pair (v, w). The initialisations and the commonly-labelled actions are respectively composed using the parallel operator introduced in the previous section. Independent actions remain independent. The respective internal actions are simply bunched together.

As an example of parallel composition, consider the action systems *N1*, *N2* and $N1 \parallel N2$ of Figure 5.A. Here, the $\otimes$-operator has been used to calculate the action system $N1 \parallel N2$. Notice the correspondence with the CSP processes of Section 5.1.

5.4 Correspondence of Parallel Composition

The semantic function '$\|$' defines CSP parallel-composition and is given by the following definition, which is based on the definition in [Ros88]:

Definition 5.4 *For process models* $\Theta_i = (A_i, F_i, D_i, I_i)$, $i = 1, 2$, *let* $C \mathrel{\hat{=}} A_1 \cup A_2$, $T_i \mathrel{\hat{=}} \{t \mid (t, \{\}) \in F_i\}$, $\mathit{Traces}_i \mathrel{\hat{=}} T_i \cup I_i$. *Then*

$$\Theta_1 \parallel \Theta_2 \quad \hat{=} \quad (C, \quad \Theta_1 \parallel_{\mathcal{F}} \Theta_2, \quad \Theta_1 \parallel_{\mathcal{D}} \Theta_2, \quad \Theta_1 \parallel_{\mathcal{I}} \Theta_2)$$

where $\Theta_1 \parallel_{\mathcal{D}} \Theta_2$ *are those* $t \in C^*$ *satisfying*

$$\begin{aligned}(\vee\, s \leq t \bullet \quad & s\restriction A_1 \in D_1 \ \wedge \ s\restriction A_2 \in T_2 \\ & \vee \ s\restriction A_1 \in T_1 \ \wedge \ s\restriction A_2 \in D_2)\end{aligned}$$

$\Theta_1 \parallel_{\mathcal{F}} \Theta_2$ *are those* $t \in C^*$, $Z \subseteq C$ *satisfying*

$$\begin{aligned}&(\vee\, X \subseteq A_1, Y \subseteq A_2 \mid X \cup Y = Z \bullet (t\restriction A_1, X) \in F_1 \ \wedge \ (t\restriction A_2, Y) \in F_2) \\ &\vee \ \ t \in \Theta_1 \parallel_{\mathcal{D}} \Theta_2\end{aligned}$$

$\Theta_1 \parallel_{\mathcal{I}} \Theta_2$ *are those* $u \in C^\omega$ *satisfying*

$$\begin{aligned}&u\restriction A_1 \in \mathit{Traces}_1 \ \wedge \ u\restriction A_2 \in \mathit{Traces}_2 \\ &\vee \ \ (\vee\, t < u \bullet t \in \Theta_1 \parallel_{\mathcal{D}} \Theta_2 \ ).\end{aligned}$$

□

Here, $t\restriction A$ denotes the trace t restricted to elements of A.

The main result of this section is Theorem 5.17 which shows that parallel composition for action systems (Definition 5.3) corresponds to CSP parallel-composition (Definition 5.4). As in Section 4.6 we present a series of lemmas leading to Theorem 5.17, each of which is proven in Appendix C. For convenience we have the following definition:

Definition 5.5 *For statements* R *and* R', $R \cong R'$ *if for each v-predicate* ϕ *and each w-predicate* ψ,

$$wp(R, \phi \vee \psi) \quad \equiv \quad wp(R', \phi \vee \psi).$$

□

Using the definition of statement-variable independence (Definition 2.9), it can be shown that **skip** acts as a unit for parallel composition of statements:

Lemma 5.6 *For v-statement S,*

$$S \parallel \mathbf{skip} \;\cong\; S.$$

□

A relationship between the iterate construct and parallel composition of statements is described by the following lemma:

Lemma 5.7 *For v-statement S, w-statement T,*

$$\mathbf{it}\ S\ \mathbf{ti} \parallel \mathbf{it}\ T\ \mathbf{ti} \;\cong\; \mathbf{it}\ S \mathbin{[\!]} T\ \mathbf{ti}.$$

□

This means that the iteration of the individual statements S and T is independent rather than being synchronised.

Now we can express certain constructs of action system $P \parallel Q$ in terms of P and Q. Let $R \mathrel{\widehat{=}} P \parallel Q$, $C \mathrel{\widehat{=}} A \cup B$ and $I \mathrel{\widehat{=}} H \cup G$, so by Definition 5.3:

$$\begin{aligned} R &= (C,\ \ (v,w),\ \ R_\imath,\ \ R_C,\ \ R_I) \\ &= (A \cup B,\ \ (v,w),\ \ P_\imath \parallel Q_\imath,\ \ par(P_A, Q_B),\ \ P_H \cup Q_G). \end{aligned}$$

Also, let $A_\imath \mathrel{\widehat{=}} A \cup \{\imath\}$. Similarly for $B_\imath$ and $C_\imath$. We have the following definition:

Definition 5.8 *For any $c \in C_\imath$*

$$\begin{aligned} \tilde{P}_c &\mathrel{\widehat{=}} P_c, && \text{if } c \in A_\imath & \qquad \tilde{Q}_c &\mathrel{\widehat{=}} Q_c, && \text{if } c \in B_\imath \\ &\mathrel{\widehat{=}} \mathbf{skip}, && \text{otherwise} & &\mathrel{\widehat{=}} \mathbf{skip}, && \text{otherwise.} \end{aligned}$$

□

From Definition 5.3 and because **skip** acts as a unit for the statement parallel-operator, we have the following lemma about actions of R:

Lemma 5.9 *For any $c \in C_\imath$,* $\qquad R_c \;\cong\; \tilde{P}_c \parallel \tilde{Q}_c.$

The statement $R_c * I$ describes the effect of executing action c in R followed by iterating over the combined internal-actions. From Lemmas 5.7 and 5.9 we have

Lemma 5.10 *For action $c \in C_\imath$,* $\qquad R_c * I \;\cong\; (\tilde{P}_c * H) \parallel (\tilde{Q}_c * G).$

Since Property 5.1 is a disjunctivity property, the conjugate of Property 5.1 is a conjunctivity property, as the following lemma shows

Lemma 5.11 *For v-statement S, w-statement T, v-predicate ϕ, w-predicate ψ,*

$$\begin{aligned}\overline{wp}(S \parallel T, \phi \wedge \psi) \;\equiv\;& \overline{wp}(S, \phi) \wedge \overline{wp}(T, \psi)\\ &\vee\; \neg\, halt(S) \wedge gd(T)\\ &\vee\; gd(S) \wedge \neg\, halt(T).\end{aligned}$$

□

By induction over finite traces we have, from Lemmas 5.10 and 5.11, the following lemma which describes possible effects of trace t in R in terms of of traces of P and Q:

Lemma 5.12 *For action trace $t \in C_\imath^*$, v-predicate ϕ, w-predicate ψ,*

$$\begin{aligned}\overline{wp}(R_t * I, \phi \wedge \psi) \;\equiv\;& \overline{wp}(\tilde{P}_t * H, \phi) \wedge \overline{wp}(\tilde{Q}_t * G, \psi)\\ &\vee\; (\vee\, s \leq t \bullet\; \neg\, halt(\tilde{P}_s * H) \wedge gd(\tilde{Q}_s * G)\\ &\qquad\qquad\quad \vee\; gd(\tilde{P}_s * H) \wedge \neg\, halt(\tilde{Q}_s * G)).\end{aligned}$$

□

There is a similar lemma for infinite traces:

Lemma 5.13 *For action trace $u \in C_\imath^\omega$,*

$$\begin{aligned}\overline{inf}(R_u * I) \;\equiv\;& \overline{inf}(\tilde{P}_u * H) \wedge \overline{inf}(\tilde{Q}_u * G)\\ &\vee\; (\vee\, s < u \bullet\; \neg\, halt(\tilde{P}_s * H) \wedge gd(\tilde{Q}_s * G)\\ &\qquad\qquad\quad \vee\; gd(\tilde{P}_s * H) \wedge \neg\, halt(\tilde{Q}_s * G)).\end{aligned}$$

□

To deal with refusal sets we have the following lemma:

Lemma 5.14 *For $Z \subseteq C$,*

$$\begin{aligned}\overline{wp}(IT_I, \neg\, gd_I(R_Z)) \;\equiv\;& (\vee\, X \subseteq A, Y \subseteq B \mid X \cup Y = Z \bullet\\ &\qquad \overline{wp}(IT_I, \neg\, gd_H(P_X) \wedge \neg\, gd_G(Q_Y))).\end{aligned}$$

□

Note that the postcondition on the right-hand side is a conjunction of a v-predicate ($\neg\, gd_H(P_X)$) and a w-predicate ($\neg\, gd_G(Q_Y)$).

The following two lemmas relate P and $\tilde{P}$ (there are similar relations between Q and $\tilde{Q}$):

Lemma 5.15 *For finite trace $t \in C^*$,* $\qquad \tilde{P}_{\langle\imath\rangle t} * H \;=\; P_{\langle\imath\rangle(t\restriction A)} * H.$

Lemma 5.16 *For infinite trace $u \in C^\omega$,*

$$\overline{inf}(\tilde{P}_{\langle \imath \rangle u} * H) \;\equiv\; \overline{inf}(P_{\langle \imath \rangle (u \restriction A)} * H) \;\vee\; gd(P_{\langle \imath \rangle (u \restriction A)} * H).$$

$\square$

Note the disjunction on the right-hand side of Lemma 5.16. This is because $u \restriction A$ could be an infinite or a finite trace.

We now state and prove Theorem 5.17:

Theorem 5.17 *For action systems P, Q, $\quad \{\!\![P \parallel Q]\!\!\} \;=\; \{\!\![P]\!\!\} \parallel \{\!\![Q]\!\!\}$ in the infinite-traces model.*

Proof: To show divergences equality we have, for $t \in C^*$,

$$
\begin{array}{lll}
& t \in \mathcal{D}\{\!\![P \parallel Q]\!\!\} & \\
\equiv & \overline{wp}(R_{\langle \imath \rangle t} * I, \mathit{false}) & \textit{Definition 4.8} \\
\equiv & (\vee\, s \leq t \bullet \neg\, halt(\tilde{P}_{\langle \imath \rangle s} * H) \wedge gd(\tilde{Q}_{\langle \imath \rangle s} * G) & \\
& \qquad \vee\;\; gd(\tilde{P}_{\langle \imath \rangle s} * H) \wedge \neg\, halt(\tilde{Q}_{\langle \imath \rangle s} * G)) & \textit{Lemma 5.12} \\
\equiv & (\vee\, s \leq t \bullet \neg\, halt(P_{\langle \imath \rangle (s \restriction A)} * H) \wedge gd(Q_{\langle \imath \rangle (s \restriction B)} * G) & \\
& \qquad \vee\;\; gd(P_{\langle \imath \rangle (s \restriction A)} * H) \wedge \neg\, halt(Q_{\langle \imath \rangle (s \restriction B)} * G)) & \textit{Lemma 5.15} \\
\equiv & (\vee\, s \leq t \bullet s \restriction A \in \mathcal{D}\{\!\![P]\!\!\} \;\wedge\; s \restriction B \in \mathcal{T}\{\!\![Q]\!\!\} & \\
& \qquad \vee\;\; s \restriction A \in \mathcal{T}\{\!\![P]\!\!\} \;\wedge\; s \restriction B \in \mathcal{D}\{\!\![Q]\!\!\}) & \textit{Definition 4.8} \\
\equiv & t \in \{\!\![P]\!\!\} \parallel_{\mathcal{D}} \{\!\![Q]\!\!\} & \textit{Definition 5.4.}
\end{array}
$$

To show failures equality we have, for $t \in C^*$, $Z \subseteq C$,

$$
\begin{array}{lll}
 & (t, Z) \in \mathcal{F}\{\!\![P \parallel Q]\!\!\} & \\
\equiv & \overline{wp}(R_{\langle \imath \rangle t} * I, \neg\, gd_I(R_Z)) & \textit{Definition 4.8} \\
\equiv & (\vee\, X \subseteq A, Y \subseteq B \mid X \cup Y = Z \bullet & \\
 & \quad \overline{wp}(R_{\langle \imath \rangle t} * I, \neg\, gd_H(P_X) \wedge \neg\, gd_G(Q_Y))) & \textit{Lemma 5.14, disjunction} \\
\equiv & (\vee\, X \subseteq A, Y \subseteq B \mid X \cup Y = Z \bullet & \\
 & \quad \overline{wp}(\tilde{P}_{\langle \imath \rangle t} * H, \neg\, gd_H(P_X)) \;\wedge\; \overline{wp}(\tilde{Q}_{\langle \imath \rangle t} * G, \neg\, gd_G(Q_Y)) & \\
 & \quad \vee\; \overline{wp}(R_{\langle \imath \rangle t} * I, \mathit{false})) & \textit{Lemma 5.12} \\
\equiv & (\vee\, X \subseteq A, Y \subseteq B \mid X \cup Y = Z \bullet & \\
 & \quad \overline{wp}(P_{\langle \imath \rangle (t \upharpoonright A)} * H, \neg\, gd_H(P_X)) \;\wedge\; \overline{wp}(Q_{\langle \imath \rangle (t \upharpoonright B)} * G, \neg\, gd_G(Q_Y))) & \\
 & \vee\; \overline{wp}(R_{\langle \imath \rangle t} * I, \mathit{false}) & \textit{Lemma 5.15} \\
\equiv & (\vee\, X \subseteq A, Y \subseteq B \mid X \cup Y = Z \bullet & \\
 & \quad (t \upharpoonright A, X) \in \mathcal{F}\{\!\![P]\!\!\} \;\wedge\; (t \upharpoonright B, Y) \in \mathcal{F}\{\!\![Q]\!\!\}) & \\
 & \vee\; t \in \{\!\![P]\!\!\} \parallel_{\mathcal{D}} \{\!\![Q]\!\!\} & \textit{Definition 4.8} \\
\equiv & (t, Z) \in \{\!\![P]\!\!\} \parallel_{\mathcal{F}} \{\!\![Q]\!\!\} & \textit{Definition 5.4.}
\end{array}
$$

To show infinites equality we have, for $u \in C^\omega$,

$$
\begin{array}{lll}
 & u \in \mathcal{I}\{\!\![P \parallel Q]\!\!\} & \\
\equiv & \overline{\mathit{inf}}(R_{\langle \imath \rangle u} * I) & \textit{Definition 4.8} \\
\equiv & \overline{\mathit{inf}}(\tilde{P}_{\langle \imath \rangle u} * H) \wedge \overline{\mathit{inf}}(\tilde{Q}_{\langle \imath \rangle u} * G) & \\
 & \vee\; (\vee\, t < u \bullet \overline{wp}(R_{\langle \imath \rangle t} * I, \mathit{false})) & \textit{Lemma 5.13} \\
\equiv & (\overline{\mathit{inf}}(P_{\langle \imath \rangle (u \upharpoonright A)} * H) \vee gd(P_{\langle \imath \rangle (u \upharpoonright A)} * H)) & \\
 & \quad \wedge\, (\overline{\mathit{inf}}(Q_{\langle \imath \rangle (u \upharpoonright B)} * G) \vee gd(Q_{\langle \imath \rangle (u \upharpoonright B)} * G)) & \\
 & \vee\; (\vee\, t < u \bullet \overline{wp}(R_{\langle \imath \rangle t} * I, \mathit{false})) & \textit{Lemma 5.16} \\
\equiv & u \upharpoonright A \in (\mathcal{I}\{\!\![P]\!\!\} \cup \mathcal{T}\{\!\![P]\!\!\}) \;\wedge\; u \upharpoonright B \in (\mathcal{I}\{\!\![Q]\!\!\} \cup \mathcal{T}\{\!\![Q]\!\!\}) & \\
 & \vee\; (\vee\, t < u \bullet t \in \{\!\![P]\!\!\} \parallel_{\mathcal{D}} \{\!\![Q]\!\!\}) & \textit{Definition 4.8} \\
\equiv & u \in \{\!\![P]\!\!\} \parallel_{\mathcal{I}} \{\!\![Q]\!\!\} & \textit{Definition 5.4.}
\end{array}
$$

□

5.5 Properties of Parallel Composition

Because of Theorem 5.17, any properties enjoyed by the CSP parallel-operator will also be enjoyed by the action-system parallel-operator (cf. Section 4.8). Monotonicity and associativity are two such important properties. Because of associativity we write $(P \parallel Q) \parallel R$ simply as

$$P \parallel Q \parallel R.$$

Also, we write the parallel composition of a finite collection of action systems P_i as $(\parallel i \bullet P_i)$, where $(\parallel i \bullet P_i)$ is calculated by successive application of the binary parallel-operator. Such calculation can deal with multi-party interaction where more than two action systems share some action label.

5.6 Parallel Composition and Refinement

Using parallel composition in conjunction with refinement is straightforward. To verify a refinement of the form

$$(\parallel i \bullet P_i) \ \sqsubseteq \ (\parallel j \bullet Q_j),$$

calculate $(\parallel i \bullet P_i)$ and $(\parallel j \bullet Q_i)$ separately, and then use simulation. The $\otimes$-operator (Definition 5.2) may be used to calculate $(\parallel i \bullet P_i)$ and $(\parallel j \bullet Q_i)$. In Chapter 7, we shall introduce a parallel operator for specification statements that allows for straightforward calculation.

In the case studies of Chapter 7, we shall make use of design steps of the form:

$$P \ \sqsubseteq \ (\parallel j \bullet Q_j).$$

This is used to decompose a single action-system into parallel action-systems. Because of the monotonicity of the parallel operator, components of $(\parallel j \bullet Q_j)$ can then be individually refined and the composition of the refined components will continue to be a refinement of P.

As an example of parallel decomposition, consider the action system specified in Figure 5.B. We can show that $N3$ is implemented by the parallel composition of $N1$ and $N2$ of Figure 5.A, i.e.

$$N3 \ \sqsubseteq \ N1 \parallel N2.$$

$N1 \parallel N2$ has already been calculated using the $\otimes$-operator. To relate the variables of $N3$ to those of $N1$ and $N2$ we use the following (disjunctive) representation function:

$$rep(\phi) \ \widehat{=} \ (0 \leq m, n \leq 1) \ \wedge \ \phi[\ s \backslash (\{\mathtt{a} \mid m = 1\} \cup \{\mathtt{b} \mid n = 1\})\ ]$$

$$N3 \;\;\widehat{=}\;\; \left(\begin{array}{ll} \textbf{var } s \subseteq \{\mathtt{a},\mathtt{b}\} & \\ \textbf{initially } \; s := \{\} & \\ a :- & \mathtt{a} \notin s \rightarrow s := s \cup \{\mathtt{a}\} \\ b :- & \mathtt{b} \notin s \rightarrow s := s \cup \{\mathtt{b}\} \\ c :- & \{\mathtt{a},\mathtt{b}\} = s \rightarrow s := \{\} \end{array}\right)$$

Figure 5.B: Example action system

where ϕ is independent of m, n. From the definition of simulation and the definition of $N1 \parallel N2$, we have proof obligations as follows:

1. $N3_\imath \;\; \preceq'_{rep} \;\; (N1_\imath \otimes N2_\imath)$
2. $N3_a \;\; \preceq_{rep} \;\; N1_a$
3. $N3_b \;\; \preceq_{rep} \;\; N2_b$
4. $N3_c \;\; \preceq_{rep} \;\; (N1_c \otimes N2_c)$
5. $rep(gd(N3_a)) \;\; \Rrightarrow \;\; gd(N1_a)$
6. $rep(gd(N3_b)) \;\; \Rrightarrow \;\; gd(N2_b)$
7. $rep(gd(N3_c)) \;\; \Rrightarrow \;\; gd(N1_c) \wedge gd(N2_c)$.

These are easily checked using predicate calculus and wp-calculus. Note that obligation 7 relies on

$$gd(S \parallel T) \;\; \equiv \;\; gd(S) \wedge gd(T),$$

which follows easily from Property 5.1.

5.7 Remarks

Since interaction in our parallel operator is based on shared actions, it is similar to the parallel operator for transition systems defined by Josephs [Jos88] and He Jifeng [He89]. They both use relational cross-product to compose commonly labelled transitions, whereas we develop a weakest precondition characterisation of parallel composition. Our characterisation was arrived at by a mixture of informal, operational consideration and experimenting with the proof of Theorem 5.17.

Based on category-theoretic considerations, Martin [Mce91] has developed a cross-product operator for predicate transformers written $S \times T$. This represents the parallel composition of statements S and T and doesn't require the variables of S and T to be distinct. It is defined as follows:

Definition 5.18 *For u-statement S, v-statement T, (u, v)-predicate π,*

$$wp(S \times T, \pi) \;\; \widehat{=} \;\; (\vee\, \phi, \psi \mid \phi \wedge \psi \Rrightarrow \pi \bullet wp(S, \phi) \wedge wp(T, \psi))$$

where ϕ ranges over all u-predicates and ψ ranges over all v-predicates.
□

This operator preserves monotonicity and positive conjunctivity [Mce91]. Unlike our parallel operator for statements, the $\times$ operator is strict w.r.t. **abort**:

$$S \times \textbf{abort} \;=\; \textbf{abort} \quad \text{while} \quad S \parallel \textbf{abort} \;\cong\; gd(S) \rightarrow \textbf{abort}.$$

However, provided u and v are distinct, it can be shown that $\times'$ satisfies Property 5.1, where

$$S \times' T \;\; \widehat{=} \;\; gd(S) \wedge gd(T) \rightarrow (S \times T).$$

Our parallel operator also resembles the parallel operator for I/O-automata developed by Lynch & Tuttle [LT87]. However, their operator is not a binary operator, rather a generalised operator over a possibly infinite set of I/O-automata. Section 5.5 discussed how a generalised parallel operator over a finite set of action systems may be defined. In order to define a parallel operator over infinite sets of action systems, we would need to generalise the CSP definition of parallel composition (Definition 5.4) and also our parallel operator for statements.

Our parallel operator differs from Back's parallel-operator [Bac92b] and the UNITY union-operator [CM88], where interaction is based on shared variables rather than shared actions. However, Back does mention an alternative form of parallel composition in [Bac92b], in which interaction is based on shared actions.

We could easily lift our restriction that parallel action-systems have distinct variables, so that interaction would be based on both shared variables and shared actions. Using the $\times$-operator on statements given above, we could even allow shared actions to share the same variables. However, we would then lose the correspondence with CSP. For example, consider the action systems P and P' as follows:

$$P \;\; \widehat{=} \;\; \begin{pmatrix} \textbf{initially} & f := true \\ a :- & f \rightarrow \textbf{skip} \end{pmatrix} \qquad P' \;\; \widehat{=} \;\; \begin{pmatrix} \textbf{initially} & g := true \\ a :- & g \rightarrow \textbf{skip} \end{pmatrix}.$$

It is easy to see that $P = P'$. Now consider the action system Q as follows:

$$Q \;\widehat{=}\; \left(\begin{array}{ll} \textbf{initially} & g := \mathit{true} \\ b :- & g \rightarrow g := \mathit{false} \end{array} \right).$$

If Q was composed with P and P' in the usual way, then it would not be the case that $Q \parallel P \;=\; Q \parallel P'$, since $Q \parallel P'$ refuses a-actions after engaging in a b-action, while $Q \parallel P$ never refuses a-actions.

In Chapter 7, a *superposition* operator for composing action systems, that allows sharing of variables, will be introduced. However, this operator will not correspond to any CSP operator. In the next chapter, we shall introduce a means for action systems to pass values amongst each other that does correspond to CSP.

Chapter 6

Value Communication

In this chapter, the CSP notion of value communication is extended to action systems. To achieve this, we introduce *value-passing* action-systems and give them a CSP semantics. We show how refinement, hiding, and parallel composition apply to these action systems.

6.1 Communication in CSP

Hoare [Hoa85] introduced the notion of a *channel* for use when describing CSP processes that communicate values with their environment. A channel named c is represented by a set of events of the form $c.i$. Occurrence of an event $c.i$ represents communication of the value i over the channel c. The set of all values that a process $\mathcal{P}$ can communicate over a channel c is defined as follows:

$$values_{\mathcal{P}}(c) \quad \widehat{=} \quad \{\ i \mid c.i \in \alpha\mathcal{P}\ \}.$$

Hoare distinguishes syntactically between two forms of channel communication. A process $\mathcal{P}$ is said to be prepared to accept an *input* on a channel c when the process offers the environment the choice between each event $c.i$ in $\alpha\mathcal{P}$. The notation $c?x$ is used to describe this form of communication, and is defined by

$$(c?x \to \mathcal{P}_x) \quad \widehat{=} \quad (\Box\ i \in values_{\mathcal{P}}(c) \bullet c.i \to \mathcal{P}_i).$$

So $(c?x \to \mathcal{P}_x)$ describes a process that accepts any value x on channel c, and then behaves as process $\mathcal{P}_x$. Note that x is bound by the parentheses.

A process is said to be prepared to *output* a value on a channel c when it internally chooses which particular event $c.i$ to offer the environment. The notation $c!e$ is used to describe this form of communication, and is defined by

$$(c!e \to \mathcal{P}) \quad \widehat{=} \quad (c.e \to \mathcal{P}).$$

So $(c!e \rightarrow \mathcal{P})$ describes a process that outputs the value e (where e is an expression) on channel c, and then behaves as process $\mathcal{P}$.

As an example, we have the following CSP process $\mathcal{DOUBLE}$, that accepts numbers on channel *left* and outputs their double on channel *right*:

$$\begin{array}{rcl} \alpha\mathcal{DOUBLE} & \widehat{=} & \{\ \mathit{left}.i \mid i \in \mathbb{N}\ \} \cup \{\ \mathit{right}.i \mid i \in \mathbb{N}\ \} \\ \mathcal{DOUBLE} & \widehat{=} & (\mathit{left}?x \rightarrow \mathit{right}!(2 * x) \rightarrow \mathcal{DOUBLE}). \end{array}$$

When processes with commonly-named channels are placed in parallel, then passing of values from one to the other is possible. Assume process $\mathcal{P}$ is ready for $c!e$, and process $\mathcal{Q}$ is ready for $c?x$. If $\mathcal{P}$ and $\mathcal{Q}$ are placed in parallel, then they can engage simultaneously in the event $c.e$, and in this way the value e is passed from $\mathcal{P}$ to $\mathcal{Q}$. This is represented by the following algebraic law:

$$(c!e \rightarrow \mathcal{P}) \parallel (c?x \rightarrow \mathcal{Q}_x) \quad = \quad c.e \rightarrow (\mathcal{P} \parallel \mathcal{Q}_e).$$

Note that the right-hand side of this law can itself be written $c!e \rightarrow (\mathcal{P} \parallel \mathcal{Q}_e)$.

6.2 Communication in Action Systems

We may represent communication events in action systems with actions whose labels are of the form $c.i$. As in CSP, we group such actions into distinct sets of actions representing separate channels, so that each channel c is represented by a set of actions whose labels are of the form $c.i$, and execution of the action labelled $c.i$ represents communication of value i on channel c.

Often it is the case that for a particular channel c, each action labelled $c.i$ will have similar behaviour. For example, consider the action system *Buffer*, specified in Figure 6.A. *Buffer* can always engage in each of the actions $\mathit{left}.i$, so this set of actions represents input on channel *left*. Provided the sequence t is non-empty, *Buffer* can engage in some action $\mathit{right}.i$, so the set of actions $\mathit{right}.i$ represents an output channel.

It would be more convenient to represent the set of actions labelled $\mathit{left}.i$ with a single action parameterised by variable x, such as the following:

$$\mathit{left} :- \qquad t := t\langle x\rangle.$$

Here, the intention is that the initial value of variable x represents the input value on channel *left*.

Similarly, we could represent the set of actions labelled $\mathit{right}.i$ with a single action parameterised by variable y as follows:

$$\mathit{right} :- \qquad t \neq \langle\rangle \rightarrow y := hd(t)\ ;\ t := tl(t).$$

$$
\begin{array}{lcl}
\alpha Buffer & \mathrel{\widehat{=}} & \{\ left.i \mid i \in \mathbb{N}\ \} \cup \{\ right.i \mid i \in \mathbb{N}\ \} \\[1ex]
Buffer & \mathrel{\widehat{=}} & \left(\begin{array}{ll}
\mathbf{var}\ t : \mathbf{seq}\ \mathbb{N} & \\
\mathbf{initially}\ \ t := \langle\rangle & \\
left.0 :- & t := t\langle 0\rangle \\
left.1 :- & t := t\langle 1\rangle \\
\quad\vdots & \\
right.0 :- & t \neq \langle\rangle \wedge hd(t) = 0 \rightarrow t := tl(t) \\
right.1 :- & t \neq \langle\rangle \wedge hd(t) = 1 \rightarrow t := tl(t) \\
\quad\vdots &
\end{array} \right)
\end{array}
$$

Figure 6.A: Communicating action system.

Here, the intention is that the value assigned to y by the action *right* represents the value to be output on channel *right*.

6.3 Value-Passing Action-Systems

In light of the considerations of the previous section, we introduce *value-passing* action-systems, which have the form

$$P \quad = \quad (A,\ v,\ P_\iota,\ P_A,\ P_H,\ dir).$$

Here A is a set of channel names, rather than a set of communication events. As before, v is the state variable, and P_ι and P_H are the initialisation and the internal actions. The component dir is a total function from A to the set $\{in, out\}$, and $P_A = \{P_a \mid a \in A\}$ is a set of actions labelled by channel names. If $dir(a) = in$, then P_a is a (v, x)-statement that doesn't change variable x, and we say that P_a is an input action and x is its input parameter. If $dir(a) = out$, then P_a is a (v, y)-statement that doesn't read the initial value of y, and we say that P_a is an output action and y is its output parameter. Both x and y must be distinct from the state variable v.

A value-passing action-system can accept an input on a channel $a \in A$, where $dir(a) = in$, whenever action P_a is enabled. Variable x of P_a represents the input value on channel a. A value-passing action-system is prepared to output a value

$$
Buffer \;\widehat{=}\; \left(\begin{array}{ll}
\mathbf{var}\ t : \mathbf{seq}\ \mathbb{N} & \\
\mathbf{initially}\ \ t := \langle\rangle & \\
\mathbf{chan}\ left\ \mathbf{in}\ \ :- & x \in \mathbb{N} \rightarrow t := t\langle x\rangle \\
\mathbf{chan}\ right\ \mathbf{out}\ \ :- & t \neq \langle\rangle \rightarrow y := hd(t)\ ;\ t := tl(t)
\end{array}\right)
$$

Figure 6.B: Value-passing action-system.

on a channel $a \in A$, where $dir(a) = out$, whenever action P_a is enabled. The value assigned to variable y by P_a represents the value to be output on channel a.

Assume that $\mathcal{W}$ is an infinite set containing all possible values we will ever want to communicate. We define the alphabet of the semantic model of an action system with channel set A to be the set $A_{\mathcal{W}}$, where

$$A_{\mathcal{W}} \;\widehat{=}\; \{\ a.i \mid a \in A \wedge i \in \mathcal{W}\ \}.$$

If c is an input channel, then, unlike Hoare's input communication form, we shall not require an action system to offer the environment the choice between the set of events $\{c.i \mid i \in \mathcal{W}\}$. Rather we assume that it offers the environment the choice between some subset of $\{c.i \mid i \in \mathcal{W}\}$. This point is discussed further in Section 6.10 where *type* restrictions on channels are introduced.

We can rewrite *Buffer* of Figure 6.A as a value-passing action-system, as shown in Figure 6.B. *Buffer* has a single input channel *left*, and a single output channel *right*. The keywords **in** and **out** are used to indicate the direction of each channel. The guard of the input action ($x \in \mathbb{N}$) means that *Buffer* will only accept natural numbers on the input channel *left*.

To define the CSP semantics of a value-passing action-system P, it is necessary to determine the effect on P of an event of the form $a.i \in A_{\mathcal{W}}$, and to determine when an arbitrary set of events, $X \subseteq A_{\mathcal{W}}$, may be refused by P. We look at input actions and output actions in turn.

6.4 Input Actions

Given an input action P_a parameterised by x, and a value $i \in \mathcal{W}$, the unparameterised action corresponding to the communication event $a.i$ is denoted $P_{a.i}$. The action $P_{a.i}$ is calculated from P_a and i using substitution-by-value (Definition 2.6) as follows:

Definition 6.1 *If $dir(a) = in$, then* $\quad P_{a.i} \;\widehat{=}\; P_a[\mathbf{value}\ x \backslash i]$.

For example, the action corresponding to $left.0$ in *Buffer* is:

$$\begin{aligned} Buffer_{left.0} &= (x \in \mathbb{N} \rightarrow t := t\langle x \rangle)[\mathbf{value}\ x \backslash 0] \\ &= 0 \in \mathbb{N} \rightarrow t := t\langle 0 \rangle. \end{aligned}$$

Now, consider a refusal set of the form $\{a.i \mid i \in S\}$, where $S \subseteq \mathcal{W}$. We write this as $a.S$, and for a given $S \subseteq \mathcal{W}$, the states in which $a.S$ may be refused by P will be represented by $\neg\, commgd(P_{a.S})$. For input action P_a, $commgd(P_{a.S})$ is defined as follows:

Definition 6.2 *If $dir(a) = in$, then for $S \subseteq \mathcal{W}$,*

$$commgd(P_{a.S}) \;\widehat{=}\; (\exists\, x \in S \bullet gd(P_a)),$$

where $gd(P_a)$ is determined in the usual way and may contain x.
□

For example, *Buffer* may refuse the set of communications $left.S$ if S contains no natural numbers:

$$\begin{aligned} \neg\, commgd(Buffer_{left.S}) &\equiv \neg\,(\exists\, x \in S \bullet gd(Buffer_{left})) \\ &\equiv \neg\,(\exists\, x \in S \bullet x \in \mathbb{N}). \end{aligned}$$

6.5 Output Actions

An output action P_a chooses a value to be output on channel a by assigning a value to variable y. We shall require that any output action P_a is independent of the initial value of y, i.e. for any predicate ϕ, $wp(P_a, \phi)$ is independent of y. Given an output action P_a parameterised by y, and a value $i \in \mathcal{W}$, the unparameterised action $P_{a.i}$ is determined by forcing y to take the value i as follows:

Definition 6.3 *If dir(a) = out, then* $\quad P_{a.i} \;\widehat{=}\; (\mathbf{var}\ y \bullet P_a[y = i])$.

For example, the action corresponding to $right.0$ in *Buffer* is calculated as follows:

$$\begin{aligned} & Buffer_{right.0} \\ = \;& (\mathbf{var}\ y \bullet t \neq \langle\rangle \rightarrow y := hd(t) \,;\, t := tl(t)\ [y = 0]) && \textit{Definition 6.3} \\ = \;& t \neq \langle\rangle \rightarrow (\mathbf{var}\ y \bullet y := hd(t)[y = 0]) \,;\, t := tl(t) && y \textit{ independent of } t \\ = \;& t \neq \langle\rangle \rightarrow (\mathbf{var}\ y \bullet hd(t) = 0 \rightarrow y := hd(t)) \,;\, t := tl(t) \\ & && \textit{property of coercion} \\ = \;& t \neq \langle\rangle \wedge hd(t) = 0 \rightarrow t := tl(t) && y \textit{ is local.} \end{aligned}$$

The extended Dijkstra-language allows us to write an output action that non-deterministically chooses a value to be output. For example, if variable s is a set, then we could specify an action that outputs some value from s on channel c as follows:

$$\textbf{chan } c \textbf{ out} \quad :- \qquad y :\in s.$$

This is a useful specification-technique that will be used in the case studies of Chapter 7, but it does mean that we have to be careful when testing for refusal of a set of output communications.

Consider the action system VP with a single output-channel as follows:

$$VP \quad = \quad \left(\begin{array}{l} \textbf{initially} \;\; \textbf{skip} \\ \textbf{chan } c \textbf{ out} \quad :- \qquad y :\in \{0,1\} \end{array} \right).$$

From Definition 6.3 we get

$$\begin{array}{lcl} VP_{c.0} & = & \textbf{skip} \\ VP_{c.1} & = & \textbf{skip} \\ VP_{c.i} & = & \textbf{miracle}, \qquad \textit{for } i \notin \{0,1\}. \end{array}$$

If testing for refusal was based on the individual communications $c.i$, then VP would offer the choice between $c.0$ and $c.1$ to the environment (since **skip** is always enabled). But our intention in specifying VP was that the choice between 0 and 1 should be made internally, i.e. VP should behave as the CSP process $\mathcal{VP}$ defined as follows:

$$\mathcal{VP} \quad \widehat{=} \quad (\sqcap\, i \in \{0,1\} \bullet c!i \rightarrow \mathcal{VP}).$$

We overcome this problem by testing for refusal of a set of events $a.S$, $S \subseteq \mathcal{W}$, as follows: if P_a is an output action, then we say that P may refuse $a.S$ when P_a is disabled, or, when P_a is enabled but has a possible outcome that results in $y \notin S$, i.e. P may refuse $a.S$ in any state satisfying

$$\neg\, gd(P_a) \vee \overline{wp}(P_a, y \notin S).$$

This leads us to define $commgd(P_{a.S})$ as follows:

Definition 6.4 *If $dir(a) = out$, then*

$$commgd(P_{a.S}) \quad \widehat{=} \quad gd(P_a) \wedge wp(P_a, y \in S).$$

□

So, for example, VP may refuse $c.S$ when

$$\begin{aligned} \neg\, commgd(VP_{c.S}) &\equiv \neg\,(gd(VP_c) \wedge wp(VP_c, y \in S)) \\ &\equiv \neg\,(0 \in S \wedge 1 \in S). \end{aligned}$$

In particular, VP can refuse $c.\{0\}$ and $c.\{1\}$, though not $c.\{0,1\}$.

6.6 Correspondence with CSP

Armed with the definitions of $P_{a.i}$ and $commgd(P_{a.S})$ for both input and output actions, we can now define the CSP semantics of a value-passing action-system $P = (A, v, P_i, P_A, P_H, dir)$.

For a trace of communication events $t \in (A_{\mathcal{W}}{}^* \cup A_{\mathcal{W}}{}^\omega)$, we write $P_t * H$ for the sequential composition of actions (of the form $P_{a.i}$) drawn from t, interspersed with IT_H, the statement that iterates over the internal actions (cf. Definition 4.6).

Let $chan$ be a total function from $A_{\mathcal{W}}$ to A, such that for each $a.i \in A_{\mathcal{W}}$,

$$chan(a.i) \;\widehat{=}\; a.$$

For set $X \subseteq A_{\mathcal{W}}$, let $X \restriction a$ be defined as

$$X \restriction a \;\widehat{=}\; \{x \in X \mid chan(x) = a\}.$$

This means that for any $X \subseteq A_{\mathcal{W}}$, $X \restriction a$ will be of the form $a.S$ for some $S \subseteq \mathcal{W}$. For any $X \subseteq A_{\mathcal{W}}$, $commgd(P_X)$ is then defined as follows:

Definition 6.5 *For $X \subseteq A_{\mathcal{W}}$,*

$$commgd(P_X) \;\widehat{=}\; (\vee\, a \in chan(\!| X |\!) \bullet commgd(P_{X \restriction a})),$$

where $commgd(P_{X \restriction a})$ is given by Definitions 6.2 and 6.4.
□

So, for example, if $X = \{\mathit{left}.0, \mathit{right}.3, \mathit{left}.4, \mathit{right}.7\}$, then

$$commgd(P_X) \;\equiv\; commgd(P_{\mathit{left}.\{0,4\}}) \vee commgd(P_{\mathit{right}.\{3,7\}}).$$

To deal with the internal actions of P we define $commgd_H(P_X)$ as follows:

Definition 6.6 *For $X \subseteq A_{\mathcal{W}}$,*

$$commgd_H(P_X) \;\widehat{=}\; (\vee\, a \in chan(\!| X |\!) \bullet \overline{wp}(IT_H, commgd(P_{X \restriction a}))).$$

□

The semantics of a value-passing action-system P, denoted $\{\![P]\!\}$, is given by the following definition:

Definition 6.7 *For* $P = (A,\ v,\ P_\imath,\ P_A,\ P_H,\ dir)$,

$$\{\![P]\!\} \ \ \widehat{=} \ \ (A_{\mathcal{W}}, \mathcal{F}\{\![P]\!\}, \mathcal{D}\{\![P]\!\}, \mathcal{I}\{\![P]\!\})$$

where $\mathcal{F}\{\![P]\!\}$ *are those* $s \in A_{\mathcal{W}}{}^*, X \subseteq A_{\mathcal{W}}$ *satisfying*

$$\overline{wp}(P_{\langle\imath\rangle s} * H, \neg\ commgd_H(P_X))$$

$\mathcal{D}\{\![P]\!\}$ *are those* $s \in A_{\mathcal{W}}{}^*$ *satisfying*

$$\neg\ halt(P_{\langle\imath\rangle s} * H)$$

$\mathcal{I}\{\![P]\!\}$ *are those* $u \in A_{\mathcal{W}}{}^\omega$ *satisfying*

$$\overline{inf}(P_{\langle\imath\rangle u} * H).$$

□

The following theorem tells us that $\{\![P]\!\}$ satisfies the well-formedness conditions of the infinite-traces model, given certain restrictions:

Theorem 6.8 *For a value-passing action-system* P, $\{\![P]\!\}$ *is well-formed in the infinite-traces model, provided* $halt(P_a) \equiv true$, *for each* $a \in A$, *where* $dir(a) = out$.

Proof: Let OUT be the set of output channels $\{a \in A \mid dir(a) = out\}$. To prove that $\{\![P]\!\}$ is well-formed, we construct an action system P^+ which has an alphabet $A_{\mathcal{W}}$, and a set of labelled actions $\{P^+_{a.i} \mid i \in \mathcal{W}\}$ corresponding to each P_a. To ensure that the choice of value on an output channel is nondeterministic, we introduce an extra state variable y_a, for each $a \in OUT$. We construct a statement *CHS* which mimics the effect of choosing an output value for each output channel $a \in OUT$, by assigning a value to each y_a. The construction of *CHS* is the cause of the restriction on P. *CHS* is always executed before offering any visible actions, and any output action $P^+_{a.i}$ is guarded by "$y_a = i$". Now since P^+ is a normal action system, with simple non-parameterised actions, $\{\![P^+]\!\}$ is given by Definition 4.8 and $\{\![P^+]\!\}$ is well-formed by Theorem 4.9. It can be shown that $\{\![P]\!\} = \{\![P^+]\!\}$, and therefore $\{\![P]\!\}$ is also well-formed. Details are given in Appendix D.
□

We shall require that any value-passing action-system satisfies the restriction imposed on output actions by Theorem 6.8. Requiring that output actions should always terminate does not prevent us from specifying an action system that outputs a value and then diverges, as an output action can be made to enable an infinite sequence of internal actions.

It can be shown that the semantic model of a value-passing action-system P satisfies the following condition: for $t \in A_{\mathcal{W}}{}^*$, $a.i \in A_{\mathcal{W}}$, where $dir(a) = out$,

$$t\langle a.i\rangle \in \mathcal{T}[\![P]\!] \quad \Rightarrow \quad (t, \{a.j \mid j \neq i\}) \in \mathcal{F}[\![P]\!].$$

This says that if P can output a value i on channel a, then it can refuse to output all other values on channel a, i.e. P never offers a deterministic choice of values on an output channel. This is a consequence of the definition of *commgd*.

6.7 Refinement

In this section, the simulation conditions of Definition 4.18 are modified to deal with value-passing action-systems. Simulation will now be used to relate action systems of the form:

$$P = (A, v, P_\imath, P_A, P_H, dir) \quad \text{and} \quad Q = (A, w, Q_\imath, Q_A, Q_G, dir),$$

where P is the abstract action-system and Q is the concrete action-system. Note that P and Q have the same set of channels A, and that the channel directions are the same in both. Each of the input actions must be parameterised by variable x, and each of the output actions must be parameterised by variable y.

It is easy to show that $P \sqsubseteq Q$ if the following conditions hold for some representation function *rep*:

1. $P_\imath * H \ \preceq'_{rep} \ Q_\imath * G$
2. $P_{a.i} * H \ \preceq_{rep} \ Q_{a.i} * G$, each $a.i \in A_{\mathcal{W}}$
3. $rep(commgd_H(P_X)) \ \Rrightarrow \ commgd_G(Q_X)$, each $X \subseteq A_{\mathcal{W}}$.

However, these conditions would be difficult to check in practice, since they involve comparison of individual communications $P_{a.i}$ and $Q_{a.i}$ rather than simply parameterised actions. If *rep* is disjunctive, then only parameterised actions need be compared, as we now show.

Definition 6.9 *A simulation, with representation function rep, is a relation on value-passing action-systems, denoted* $\sqsubseteq_{rep}$*, such that for*

$$P = (A, v, P_\imath, P_A, P_H, dir) \quad \textit{and} \quad Q = (A, w, Q_\imath, Q_A, Q_G, dir),$$

$P \sqsubseteq_{rep} Q$ *if each of the following conditions hold:*

1. $P_i \preceq'_{rep} Q_i$

2. $P_a \preceq_{rep} Q_a$,   *each* $a \in A$

3. $IT_H \preceq_{rep} IT_G$

4. *for each* $a \in \{\ a \in A \mid dir(a) = in\ \}$,
$$rep(\overline{wp}(IT_H, gd(P_a))) \ \Rrightarrow\ \overline{wp}(IT_G, gd(Q_a))$$

5. *for each* $a \in \{\ a \in A \mid dir(a) = out\ \}$, *any y-predicate* ψ,
$$rep(\overline{wp}(IT_H, gd(P_a) \wedge wp(P_a, \psi))) \ \Rrightarrow\ \overline{wp}(IT_G, gd(Q_a) \wedge wp(Q_a, \psi)).$$

□

If the representation function *rep* satisfies certain conditions, then simulation for value-passing action-systems is sound:

Theorem 6.10 *For value-passing action-systems P and Q, and representation function rep, $P \sqsubseteq_{rep} Q$ implies $P \sqsubseteq Q$ in the infinite-traces model, provided rep satisfies the following conditions:*

(a). *rep is disjunctive*

(b). *for $i \in \mathcal{W}$, predicate ϕ,* $rep(\phi[x \backslash i]) \Rrightarrow rep(\phi)[x \backslash i]$

(c). *for y-predicate ψ,* $rep(\psi) \Rrightarrow \psi$

where x is the input parameter used by input actions, and y is the output parameter used by output actions.

Proof: From (a), (b), (c) and $P_a \preceq_{rep} Q_a$, we can prove the following lemma (see Appendix D):

Lemma 6.11 *For $a.i \in A_{\mathcal{W}}$,* $P_{a.i} \ \preceq_{rep}\ Q_{a.i}$.

Then, since $IT_H \preceq_{rep} IT_G$, we get

$$(P_{a.i} * H) \ \preceq_{rep}\ (Q_{a.i} * G), \qquad \textit{each } a.i \in A_{\mathcal{W}}. \tag{i}$$

From (a) and Conditions 4 and 5 of Definition 6.9, we can prove the following lemma (see Appendix D):

Lemma 6.12 *For $X \subseteq A_{\mathcal{W}}$,* $rep(commgd_H(P_X)) \ \Rrightarrow\ commgd_G(Q_X)$.

Finally by (i), Lemma 6.12, and $P_i \preceq'_{rep} Q_i$, we can show that $P \sqsubseteq Q$ in the manner of Theorem 3.13.
□

In the case of output actions, Condition 2 of Definition 6.9 allows Q_a to be more deterministic in its choice of output value than P_a. For example, consider the action systems VP and VP' as follows:

$$VP = \left(\begin{array}{l} \textbf{initially} \;\; \textbf{skip} \\ \textbf{chan} \; c \; \textbf{out} \;\; :- \;\; y :\in \{0,1\} \end{array} \right) \qquad VP' = \left(\begin{array}{l} \textbf{initially} \;\; \textbf{skip} \\ \textbf{chan} \; c \; \textbf{out} \;\; :- \;\; y := 0 \end{array} \right).$$

It is easy to show that there is a simulation between VP and VP'. However, the soundness of this simulation is only valid because of our view that VP internally chooses between outputting 0 or 1. If VP was understood to offer the choice to the environment between 0 or 1, then it would not be the case that $VP \sqsubseteq VP'$, since VP' refuses to offer a 1.

Comparison of Definitions 4.18 and 6.9, shows that Condition 2 of Definition 4.18 has been replaced by Conditions 2 and 3 in Definition 6.9. Splitting the obligation like this is necessary because

$$P_a * H \quad \preceq_{rep} \quad Q_a * G$$

is not sufficient to show CSP refinement, in the case where P_a and Q_a are output actions. Consider the action systems P and Q as follows:

$$P \;\; \widehat{=} \;\; \left(\begin{array}{lll} \textbf{var} \; f : \textbf{bool} & & \\ \textbf{initially} \;\; f := \mathit{false} & & \\ \textbf{chan} \; a \; \textbf{out} \;\; :- & & (y, f := 0, \mathit{true}) \; [] \; (y, f := 1, \mathit{false}) \\ \textbf{internal} \;\; h :- & & f \rightarrow \textbf{skip} \end{array} \right)$$

$$Q \;\; \widehat{=} \;\; \left(\begin{array}{lll} \textbf{var} \; f : \textbf{bool} & & \\ \textbf{initially} \;\; f := \mathit{false} & & \\ \textbf{chan} \; a \; \textbf{out} \;\; :- & & (y, f := 1, \mathit{true}) \\ \textbf{internal} \;\; g :- & & f \rightarrow \textbf{skip} \end{array} \right).$$

It can easily be shown that P and Q respectively have the same semantics as the CSP processes $\mathcal{P}$ and $\mathcal{Q}$ defined as:

$$\mathcal{P} \;\; \widehat{=} \;\; (a.0 \rightarrow \mathit{CHAOS}) \sqcap (a.1 \rightarrow \mathcal{P}) \qquad \mathcal{Q} \;\; \widehat{=} \;\; (a.1 \rightarrow \mathit{CHAOS}).$$

Since it is not the case that $\mathcal{P} \sqsubseteq \mathcal{Q}$, it cannot be the case that $P \sqsubseteq Q$ either. However, we can show that

$$P_a \; ; \textbf{it} \; P_h \; \textbf{ti} \;\; = \;\; Q_a \; ; \textbf{it} \; Q_g \; \textbf{ti} \;\; = \;\; \textbf{abort}$$

For value-passing action-systems

$$P = (A, v, P_\imath, P_A, \{\}, dir) \quad and \quad Q = (A, w, Q_\imath, Q_A, Q_G, dir)$$

$P \sqsubseteq_{rep} Q$ if the following conditions hold, for representation function rep satisfying (a), (b) and (c) of Theorem 6.10, well-founded set WF, and variant E,

1. $P_\imath \ \preceq'_{rep} \ Q_\imath$
2. $P_a \ \preceq_{rep} \ Q_a$, *each* $a \in A$
3. **skip** $\preceq_{rep} \ Q_g$, *each* $g \in G$
4. $rep(true) \ \Rrightarrow \ E \in WF$
5. $rep(true) \wedge E = e \ \Rrightarrow \ wp(Q_g, E < e)$, *each* $g \in G$
6. $rep(gd(P_a)) \ \Rrightarrow \ gd(Q_a) \vee gd(Q_G)$, *each* $a \in A$.

Figure 6.C: Hiding-refinement rule for value-passing action systems.

so that $P_a * H \preceq_{rep} Q_a * G$.

The upwards simulation rep_u (cf. Section 2.8) used by Woodcock & Morgan [WM90] is not disjunctive. This means their completeness argument cannot be used for simulation in value-passing action-systems as defined here. We leave the completeness of simulation value-passing action-systems for further study.

In the case that the abstract action system P has no internal actions, then a variant function can be used, and the conditions of Definition 6.9 can be simplified. These simplified conditions are listed in Figure 6.C, and are similar to the hiding-refinement conditions of Figure 4.C for normal (non-value-passing) action-systems.

Corollary 6.13 *The simplified conditions of Figure 6.C are sufficient to show that $P \sqsubseteq_{rep} Q$.*

Proof: Conditions 1 and 2 of Definition 6.9 are immediately satisfied. From 3, 4, and 5 of Figure 6.C, we can show

$$\textbf{skip} \preceq_{rep} IT_G$$

as shown in the proof of Corollary 4.21. Then, since $P_H = \{\}$, Condition 3 of Definition 6.9 is satisfied. For input actions, Condition 4 of Definition 6.9 follows from Condition 6 of Figure 6.C and Theorem 4.22 (cf. Corollary 4.23). For output actions, we have for y-predicate ψ,

$$\begin{array}{lll}
& rep(\overline{wp}(IT_H, gd(P_a) \wedge wp(P_a, \psi))) & \\
\Rrightarrow & rep(gd(P_a) \wedge wp(P_a, \psi)) & \textit{since } P_H = \{\} \\
\Rrightarrow & wp(IT_G, rep(gd(P_a) \wedge wp(P_a, \psi))) & \textit{since } \mathbf{skip} \preceq_{rep} IT_G \\
\Rrightarrow & wp(IT_G, rep(gd(P_a)) \wedge rep(wp(P_a, \psi))) & \textit{monotonicity of rep} \\
\Rrightarrow & wp(IT_G, (gd(Q_a) \vee gd(Q_G)) \wedge wp(Q_a, rep(\psi))) & \textit{Conditions 6, 2} \\
\Rrightarrow & \overline{wp}(IT_G, gd(Q_a) \wedge wp(Q_a, rep(\psi))) & \textit{Theorem 4.22} \\
\Rrightarrow & \overline{wp}(IT_G, gd(Q_a) \wedge wp(Q_a, \psi)) & (c) \textit{ of Theorem 6.10.}
\end{array}$$

So, Condition 5 of Definition 6.9 is satisfied.
□

Recall that Section 4.10 describes how the monotonicity of the hiding operator allows us, in certain cases, to use the hiding-refinement rules of Figure 4.C even if the abstract action-system P has internal actions: we simply treat the internal actions of P, and the corresponding internal actions of Q, as visible actions. This also applies to value-passing action-systems, since the hiding operator for value passing action systems is monotonic, as we shall see.

6.8 Channel Hiding

In CSP, a channel c is hidden in a process $\mathcal{P}$ by hiding each event of the form $c.i$ in the alphabet of $\mathcal{P}$. In a value-passing action-system P, the channel c is hidden by internalising the parameterised action P_c. For value-passing action-system $P = (A,\ v,\ P_\imath,\ P_A,\ P_H,\ dir)$, and channel set $C \subseteq A$, $P \setminus C$ denotes hiding of the channels C in P, and is defined by:

Definition 6.14

$$\begin{array}{rcl}
P \setminus C & \widehat{=} & (A - C,\ v,\ P_\imath,\ P_{A-C}, \\
& & P_H \cup \{\ (\mathbf{var}\ x, y \bullet P_c) \mid c \in C\ \},\ C \lhd dir)
\end{array}$$

where x is the input parameter used by input actions of P, and y is the output parameter used by output actions of P.
□

Here, $C \lhd f$ restricts the domain of function f to those elements not in set C. Notice that the parameters x and y are localised in each action to be hidden.

In CSP terms, our intention with this definition is to hide the set of communications $C_{\mathcal{W}} \subseteq A_{\mathcal{W}}$. The following theorem shows that we get the required correspondence with CSP hiding:

Theorem 6.15 *For value-passing action-system P, channel set $C \subseteq A$,*

$$\{\!|P \setminus C|\!\} \quad = \quad \{\!|P|\!\} \setminus C_{\mathcal{W}},$$

where $\{\!|P|\!\} \setminus C_{\mathcal{W}}$ is given by Definition 4.11.

Proof: Proof of this theorem is similar to the proof of Theorem 4.17: Let $Q \mathrel{\widehat{=}} P \setminus C$, $B \mathrel{\widehat{=}} A - C$, $Q_G \mathrel{\widehat{=}} P_H \cup \{\ (\mathbf{var}\ x, y \bullet P_c) \mid c \in C\ \}$. Corresponding to Lemma 4.12 we have (see Appendix D):

Lemma 6.16 *For $b.i \in B_{\mathcal{W}}$,*

$$Q_{b.i} * G \quad = \quad P_{b.i} * H \quad ; \ \mathbf{it}\ ([\!]\ c.i \in C_{\mathcal{W}} \bullet P_{c.i} * H)\ \mathbf{ti}.$$

This leads to lemmas corresponding to Lemmas 4.14 and 4.15 with $A_{\mathcal{W}}, B_{\mathcal{W}}$ instead of A, B. Corresponding to Lemma 4.16 we have (see Appendix D):

Lemma 6.17 *For $X \subseteq B_{\mathcal{W}}$,*

$$\overline{wp}(IT_G, \neg\ commgd_G(Q_X)) \quad \equiv \quad \overline{wp}(IT_G, \neg\ commgd_H(P_{X \cup C_{\mathcal{W}}})).$$

Theorem 6.15 can then be proven in the same manner as Theorem 4.17.
□

Because of Theorem 6.15, we have that the hiding operator for value-passing action-systems enjoys any properties enjoyed by the CSP hiding operator (cf. Section 4.8).

6.9 Parallel Composition – Introduction

Parallel composition will be defined for value-passing action-systems of the form:

$$\begin{aligned} P &= (A,\ v,\ P_\imath,\ P_A,\ P_H,\ dir_P) \\ Q &= (B,\ w,\ Q_\imath,\ Q_B,\ Q_G,\ dir_Q). \end{aligned}$$

As in Chapter 5, we require that the state-variables v and w are distinct. Each input action of P and of Q is parameterised by x, and each output action is parameterised by y. Parallel composition of P and Q is achieved by composing commonly-labelled actions of P and Q, leaving independently-labelled actions unchanged. If c is a channel name common to both P and Q, then there are three possibilities: either both P_c and Q_c are input actions, one is an output action and the other is an input action, or, both P_c and Q_c are output actions.

Consider the case where both P_c and Q_c are input actions. Suppose, for example, that P_c and Q_c are as follows:

$$
\begin{aligned}
P_c &\mathrel{\widehat{=}} x \in X \wedge G_P \rightarrow v := x \\
Q_c &\mathrel{\widehat{=}} x \in X' \wedge G_Q \rightarrow w := x.
\end{aligned}
$$

On its own, P will offer the choice between each value in X on channel c whenever G_P is satisfied, and similarly for Q. The parallel composition of P and Q will only offer the choice between values they both agree on (i.e. values from $X \cap X'$), when both G_P and G_Q are satisfied. To compose P_c and Q_c, the $\otimes$-operator (Definition 5.2) may be used:

$$
\begin{aligned}
P_c \otimes Q_c &= gd(P_c) \;\wedge\; gd(Q_c) \rightarrow P_c \;;\; Q_c \\
&= x \in X \;\wedge\; G_P \;\wedge\; x \in X' \;\wedge\; G_Q \rightarrow v := x \;;\; w := x \\
&= x \in X \cap X' \;\wedge\; G_P \;\wedge\; G_Q \rightarrow v := x \;;\; w := x.
\end{aligned}
$$

Notice that $P_c \otimes Q_c$ is itself parameterised by the input variable x.

Now, consider the case where P_c is an output action, and Q_c is an input action. Suppose, for example, that P_c and Q_c are as follows:

$$
\begin{aligned}
P_c &= G_P \rightarrow y := e \\
Q_c &= x \in X \wedge G_Q \rightarrow w := x.
\end{aligned}
$$

Out intention in composing P_c and Q_c is that the output value produced by P_c should be passed on as the input value for Q_c. To compose P_c and Q_c we may use the $\vec{\otimes}$-operator, defined as follows:

Definition 6.18 *For output action P_a, input action Q_a,*

$$
P_a \mathbin{\vec{\otimes}} Q_a \mathrel{\widehat{=}} P_a \;;\; Q_a[\mathbf{value}\; x \backslash y].
$$

□

In the case of the examples given, we get

$$
\begin{aligned}
P_c \mathbin{\vec{\otimes}} Q_c &= (G_P \rightarrow y := e) \;;\; (x \in X \wedge G_Q \rightarrow w := x)[\mathbf{value}\; x \backslash y] \\
&= (G_P \rightarrow y := e) \;;\; (e \in X \wedge G_Q \rightarrow w := e) \\
&= G_P \;\wedge\; e \in X \;\wedge\; G_Q \rightarrow y := e \;;\; w := e.
\end{aligned}
$$

Note that $P_c \vec{\otimes} Q_c$ is itself parameterised by the output variable y.

However, nondeterministic output-statements can cause problems when composing an output action with an input action. Consider the action systems P and Q defined as follows:

$$P \mathrel{\widehat{=}} \left(\begin{array}{ll} \textbf{initially} & \textbf{skip} \\ c\ \textbf{out} \ :- & y :\in \{0,1\} \end{array} \right) \qquad Q \mathrel{\widehat{=}} \left(\begin{array}{ll} \textbf{initially} & \textbf{skip} \\ c\ \textbf{in} \ :- & x > 0 \rightarrow \textbf{skip} \end{array} \right).$$

Here, P nondeterministically outputs 0 or 1 on channel c, while Q accepts any input value greater than 0 on channel c. P and Q behave respectively as the CSP processes $\mathcal{P}$ and $\mathcal{Q}$ defined as follows:

$$\mathcal{P} \mathrel{\widehat{=}} (\sqcap\, i \in \{0,1\} \bullet c.i \rightarrow \mathcal{P}) \qquad \mathcal{Q} \mathrel{\widehat{=}} ([]\, i > 0 \bullet c.i \rightarrow \mathcal{Q}).$$

Using the algebraic laws of CSP, it can be shown that

$$\mathcal{P} \parallel \mathcal{Q} \quad = \quad \mathit{STOP} \sqcap (c.1 \rightarrow \mathcal{P} \parallel \mathcal{Q}).$$

Here, deadlock occurs when $\mathcal{P}$ is only prepared to offer 0 on channel c. If we compose P_c and Q_c using the $\vec{\otimes}$-operator to form the action system $P \parallel Q$, we get

$$P \parallel Q \quad = \quad \left(\begin{array}{ll} \textbf{initially} & \textbf{skip} \\ c\ \textbf{out} \ :- & y := 1 \end{array} \right).$$

Action system $P \parallel Q$ never deadlocks, but continually outputs the value 1 on channel c, which is not the same as the CSP process $\mathcal{P} \parallel \mathcal{Q}$. This is because the input action Q_c forces the output action P_c to choose a path that sets y to be greater then 0, and so P_c sets y to 1. It is not possible to specify an action system corresponding to $\mathcal{P} \parallel \mathcal{Q}$ that has **skip** as its initialisation. The CSP process $\mathcal{P} \parallel \mathcal{Q}$ does correspond to the action system R as follows:

$$R \quad = \quad \left(\begin{array}{ll} \textbf{var}\ f : \textbf{bool} & \\ \textbf{initially} \ f :\in \{\mathit{true}, \mathit{false}\} & \\ c\ \textbf{out} \ :- & f \rightarrow y := 1\ ;\ f :\in \{\mathit{true}, \mathit{false}\} \end{array} \right).$$

Here, the nondeterministic choice between whether or not deadlock occurs (represented by the setting of flag f), has to be made before action c can be offered. So, action system R cannot be constructed from P and Q by simply composing commonly-labelled actions, but rather requires a more complicated construction.

The case where both actions to be composed are output actions causes a similar problem: deadlock will occur unless both output channels guarantee to output exactly the same value, but constructing an action system that tests for this cannot be achieved by simply composing both output actions.

We shall overcome these problems, firstly, when composing an output action P_c with an input action Q_c, by requiring that P_c is always guaranteed to output a value that is acceptable by Q_c. This restriction is formalised in the next section. Secondly, parallel composition of commonly-labelled output actions will not be allowed.

6.10 Channel Typing

To deal with the parallel composition of an output and an input action, we introduce *type* restrictions on the values that may be communicated on the channels of a value-passing action system. A value-passing action system will now have the form:

$$P \;\; = \;\; (A,\ v,\ P_\imath,\ P_A,\ P_H,\ dir,\ type).$$

The function *type* maps each channel name to some subset of $\mathcal{W}$. For each channel $a \in A$, $type(a)$ represents the range of values that may be communicated along channel a.

The actions of P must obey the restrictions imposed by the function *type*. An input action P_a may only accept input values that are in $type(a)$, and also, if P_a is enabled, then it must accept all input values in $type(a)$. To ensure this, each input action must satisfy the following property:

Property 6.19 *For each $a \in \{a \in A \mid dir(a) = in\}$, P_a must satisfy*

$$gd(P_a) \;\; \equiv \;\; x \in type(a) \wedge gd(\mathbf{var}\ x \bullet P_a).$$

□

For example, if $type(a) = \mathbb{N}$, then the action

$$P_a \;\; \widehat{=} \;\; x \in \mathbb{N} \rightarrow t := t\langle x\rangle$$

satisfies Property 6.19, while the action

$$P_a \;\; \widehat{=} \;\; x \in \mathbb{N} \;\wedge\; x \geq 10 \rightarrow t := t\langle x\rangle$$

does not satisfy Property 6.19, since it won't accept natural numbers less than 10.

An output action P_a should never output a value that is outside $type(a)$. To ensure this, each output action must satisfy the following property:

Property 6.20 *For each $a \in \{a \in A \mid dir(a) = out\}$, P_a must satisfy*

$$wp(P_a, y \in type(a)) \;\; \equiv \;\; true.$$

□

Note that this property implies that output actions are always guaranteed to terminate, a condition we already require in order to ensure that the semantics of a value-passing action-system are well-formed (Theorem 6.8).

Now, when composing an output action P_c of type $type_P(c)$ with an input action Q_c of type $type_Q(c)$, we shall require that $type_P(c) \subseteq type_Q(c)$. We will see that the composite action is also guaranteed to output a value in $type_P(c)$.

If a value-passing action-system P satisfies Properties 6.19 and 6.20, then it can be shown that the type constraints are reflected in the semantic model of P. Firstly, any non-divergent communication, input or output, is always properly typed: for $t \in A_{\mathcal{W}}{}^*$, $a.i \in A_{\mathcal{W}}$,

$$t\langle a.i\rangle \in \mathcal{T}\{\!\!\{P\}\!\!\} \wedge t \notin \mathcal{D}\{\!\!\{P\}\!\!\} \quad \Rightarrow \quad i \in type(a).$$

If P diverges, however, then it no longer obeys this type restriction. Secondly, if P accepts any communication on an input channel a, then all values of $type(a)$ may be communicated, and if P refuses some value in $type(a)$, then it refuses all values on input channel a: for $t \in A_{\mathcal{W}}{}^*$, $a.i \in A_{\mathcal{W}}$, where $dir(a) = in$,

$$\begin{aligned} t\langle a.i\rangle \in \mathcal{T}\{\!\!\{P\}\!\!\} \quad &\Rightarrow \quad (\forall j \in type(a) \bullet t\langle a.j\rangle \in \mathcal{T}\{\!\!\{P\}\!\!\}) \\ (t, \{a.i\}) \in \mathcal{F}\{\!\!\{P\}\!\!\} \wedge i \in type(a) \quad &\Rightarrow \quad (t, \{a.j \mid j \in \mathcal{W}\}) \in \mathcal{F}\{\!\!\{P\}\!\!\}. \end{aligned}$$

6.11 Parallel Composition – Definition

The parallel operator for value-passing action systems is now defined for P and Q of the form

$$\begin{aligned} P &= (A,\ v,\ P_\iota,\ P_A,\ P_H,\ dir_P,\ type_P) \\ Q &= (B,\ w,\ Q_\iota,\ Q_B,\ Q_G,\ dir_Q,\ type_Q). \end{aligned}$$

We do not use the $\otimes$-operator or the $\vec{\otimes}$-operator to define parallel composition of value-passing action-systems formally. Instead, to compose input actions, the $\|$-operator for statements introduced in Section 5.2 is used. In order to use it, we must loosen the constraints on this operator: instead of composition of v-action P_c and w-action Q_c, where v and w are distinct, composition of (x, v)-action P_a and (x, w)-action Q_a will be allowed, provided both P_a and Q_a leave variable x unchanged, i.e. x is an input parameter in both actions. (Note: Property 5.1 will still be concerned only with postconditions in v and in w, as we will not be interested in postconditions containing input parameter x.) The composite action $P_a \parallel Q_a$ is an (x, v, w)-action, with x serving as the input parameter.

To compose an output action P_c with an input action Q_c, we introduce the $\vec{\|}$-operator. We write $P_c \vec{\|} Q_c$ for the composition of P_c and Q_c, where it is understood that the y-value produced by P_c is passed on as the x-value for Q_c. The $\vec{\|}$-operator should preserve monotonicity and positive conjunctivity, and should satisfy the following properties:

Property 6.21 *The $\vec{\|}$-operator satisfies*

(*a*). *For $i \in \mathcal{W}$,* $\quad (P_a \vec{\|} Q_a)[y = i] \;\cong\; (P_a[y = i]) \,\|\, (Q_a[\mathbf{value}\ x \backslash i])$

(*b*). *For y-predicate ψ,* $\quad wp(P_a \vec{\|} Q_a, \psi) \;\equiv\; gd(\mathbf{var}\ x \bullet Q_a) \Rightarrow wp(P_a, \psi)$

(*c*). *For any predicate ϕ, $wp(P_a \vec{\|} Q_a, \phi)$ is independent of y.*

$\square$

For example, it can be shown that the $\vec{\otimes}$-operator satisfies Property 6.21, provided both P_a and Q_a terminate. The composite action $P_a \vec{\|} Q_a$ is a (y, v, w)-action, with y serving as the output parameter. Because of Theorem 6.8, we require that $halt(P_c \vec{\|} Q_c) \equiv true$, and to achieve this it must be the case that $halt(Q_c) \equiv true$, in addition to $halt(P_c) \equiv true$. So any input action to be composed with an output action must always terminate.

For value-passing action-systems

$$\begin{aligned} P &= (A,\ v,\ P_\imath,\ P_A,\ P_H,\ dir_P,\ type_P) \\ Q &= (B,\ w,\ Q_\imath,\ Q_B,\ Q_G,\ dir_Q,\ type_Q) \end{aligned}$$

let the sets IN, OUT_P, and OUT_Q be defined as follows:

$$\begin{aligned} IN &\mathrel{\widehat{=}} \{\ c \in A \cap B \mid dir_P(c) = in \wedge dir_Q(c) = in\ \} \\ OUT_P &\mathrel{\widehat{=}} \{\ c \in A \cap B \mid dir_P(c) = out \wedge dir_Q(c) = in\ \} \\ OUT_Q &\mathrel{\widehat{=}} \{\ c \in A \cap B \mid dir_P(c) = in \wedge dir_Q(c) = out\ \}. \end{aligned}$$

The parallel composition of P and Q, written $P \parallel Q$, is defined if the following conditions are satisfied:

1. *Common actions of P and Q satisfy the type restrictions (Properties 6.19 and 6.20)*
2. *$type_P(c) \subseteq type_Q(c)$, $\quad$ each $c \in OUT_P$*
3. *$type_Q(c) \subseteq type_P(c)$, $\quad$ each $c \in OUT_Q$*
4. *P and Q have no common output-actions*

5. *For input action* Q_c, *where* $c \in OUT_P$, $halt(Q_c) \equiv true$

6. *For input action* P_c, *where* $c \in OUT_Q$, $halt(P_c) \equiv true$.

Parallel composition of P and Q is then given by

Definition 6.22

$$P \parallel Q \;\; \widehat{=} \;\; (A \cup B,\ (v, w),\ P_\imath \parallel Q_\imath,\ par(P_A, Q_B),\ P_H \cup Q_G,\ dir,\ type)$$

where

$$\begin{array}{rcl}
par(P_A, Q_B) & \widehat{=} & P_{A-B} \cup Q_{B-A} \\
& & \cup \; \{ \; P_c \parallel Q_c \mid c \in IN \; \} \\
& & \cup \; \{ \; P_c \vec{\parallel} Q_c \mid c \in OUT_P \; \} \\
& & \cup \; \{ \; Q_c \vec{\parallel} P_c \mid c \in OUT_Q \; \} \\
\\
dir(c) & \widehat{=} & dir_P(c), \qquad \textit{if } c \in A - B \\
& \widehat{=} & dir_Q(c), \qquad \textit{if } c \in B - A \\
& \widehat{=} & in, \qquad \textit{if } c \in IN \\
& \widehat{=} & out, \qquad \textit{if } c \in OUT_P \cup OUT_Q \\
\\
type(c) & \widehat{=} & type_P(c), \qquad \textit{if } c \in A - B \;\vee\; c \in OUT_P \\
& \widehat{=} & type_Q(c), \qquad \textit{if } c \in B - A \;\vee\; c \in OUT_Q \\
& \widehat{=} & type_P(c) \cap type_Q(c), \qquad \textit{if } c \in IN.
\end{array}$$

□

Parallel composition preserves type restrictions:

Theorem 6.23 *If* P *and* Q *satisfy the type restrictions, then composite input actions of* $P \parallel Q$ *satisfy Property 6.19, and composite output-actions of* $P \parallel Q$ *satisfy Property 6.20.*

Proof: In the case that both P_c and Q_c are input actions, we have

$$\begin{array}{ll}
& gd(\mathbf{var}\ x \bullet P_c \parallel Q_c) \\
\equiv & (\exists\, x \bullet gd(P_c) \wedge gd(Q_c)) \qquad\qquad \textit{definition of } \parallel \\
\equiv & (\exists\, x \bullet x \in type_P(c) \wedge gd(\mathbf{var}\ x \bullet P_c) \wedge x \in type_Q(c) \wedge gd(\mathbf{var}\ x \bullet Q_c)) \\
& \qquad\qquad \textit{since } P_c \textit{ and } Q_c \textit{ satisfy Property 6.19} \\
\equiv & (\exists\, x \bullet x \in type_P(c) \wedge x \in type_Q(c)) \wedge gd(\mathbf{var}\ x \bullet P_c) \wedge gd(\mathbf{var}\ x \bullet Q_c)\ .
\end{array}$$

Now, $P_c \parallel Q_c$ satisfies Property 6.19, since

$$
\begin{array}{cll}
 & gd(P_c \parallel Q_c) & \\
\equiv & (gd(P_c) \wedge gd(Q_c)) & \textit{definition of } \parallel \\
\equiv & x \in type_P(c) \wedge x \in type_Q(c) \wedge gd(\mathbf{var}\ x \bullet P_c) \wedge gd(\mathbf{var}\ x \bullet Q_c) & \\
 & & \textit{since } P_c \textit{ and } Q_c \textit{ satisfy Property 6.19} \\
\equiv & x \in type(c) \wedge gd(\mathbf{var}\ x \bullet P_c \parallel Q_c) & \textit{definition of type and from above.}
\end{array}
$$

In the case that P_c is an output action and Q_c is an input action, $P_c \vec{\parallel} Q_c$ satisfies Property 6.20, since

$$
\begin{array}{cll}
 & wp(P_c \vec{\parallel} Q_c, y \in type(c)) & \\
\equiv & wp(P_c \vec{\parallel} Q_c, y \in type_P(c)) & \textit{definition of type} \\
\equiv & gd(\mathbf{var}\ x \bullet Q_c) \Rightarrow wp(P_c, y \in type_P(c)) & \textit{Property 6.21 (b)} \\
\equiv & true & P_c \textit{ satisfies Property 6.20.}
\end{array}
$$

The case where P_c is an input action and Q_c is an output action is similarly proven.
□

The following theorem shows that parallel composition of value-passing action-systems corresponds to CSP parallel-composition:

Theorem 6.24 *For value-passing action-systems P and Q, if $P \parallel Q$ is defined, then*

$$\{\!|P \parallel Q|\!\} \;=\; \{\!|P|\!\} \parallel \{\!|Q|\!\},$$

where $\{\!|P|\!\} \parallel \{\!|Q|\!\}$ is given by Definition 5.4.

Proof: Proof of this theorem is similar that of Theorem 5.17: Let $R \mathrel{\widehat{=}} P \parallel Q$, $C \mathrel{\widehat{=}} A \cup B$, $I \mathrel{\widehat{=}} H \cup G$. Let

$$
\begin{array}{rcll}
\tilde{P}_{c.i} & \widehat{=} & P_{c.i}, & \textit{if } c \in A \\
 & \widehat{=} & \mathbf{skip}, & \textit{otherwise.}
\end{array}
$$

Likewise for $\tilde{Q}_{c.i}$. Corresponding to Lemma 5.10 we get, using Property 6.21 (a) and Lemma 5.7, the following lemma (see Appendix D):

Lemma 6.25 *For $c.i \in C_{\mathcal{W}}$,* $\quad R_{c.i} * I \;\cong\; (\tilde{P}_{c.i} * H) \parallel (\tilde{Q}_{c.i} * G).$

This leads to lemmas corresponding to 5.12 and 5.13. From Property 6.21 (b), the following lemma, corresponding to Lemma 5.14, can be proven (see Appendix D):

Lemma 6.26 *For* $Z \subseteq C_{\mathcal{W}}$*,*

$$\begin{aligned}
&\overline{wp}(IT_I, \neg\ commgd_I(R_Z)) \ \equiv \\
&\quad (\vee\ X \subseteq A_{\mathcal{W}},\ Y \subseteq B_{\mathcal{W}} \mid X \cup Y = Z\ \bullet \\
&\qquad \overline{wp}(IT_I, \neg\ commgd_H(P_X) \wedge \neg\ commgd_G(Q_Y))).
\end{aligned}$$

Theorem 6.24 can then be proven in the same manner as Theorem 5.17.
□

Note: Instead of using $A_{\mathcal{W}}$ as the alphabet for the semantic model of a value-passing action-system (Definition 6.7) we could have used the set

$$\{\ a.i \mid a \in A \wedge i \in type(a)\ \}.$$

However, using $A_{\mathcal{W}}$ simplifies parallel composition, since, if P and Q have channel c in common, then the alphabets of both $\{\!|P|\!\}$ and $\{\!|Q|\!\}$ include the set $\{\ c.i \mid i \in \mathcal{W}\ \}$. Otherwise, the alphabets of $\{\!|P|\!\}$ and $\{\!|Q|\!\}$ would need to be extended, to ensure they had at least $\{\ c.i \mid i \in type_P(c) \cup type_Q(c)\ \}$ in common.

Because of Theorem 6.24, the parallel operator for value-passing action-systems enjoys properties such as monotonicity and associativity. As in Section 5.5, we write $(\parallel\ i \bullet P_i)$ for the parallel composition of a finite collection of action systems P_i. Note that the restrictions imposed on the parallel operator for value-passing action-systems do not hinder associativity, i.e. $(P \parallel Q) \parallel R$ is defined iff $P \parallel (Q \parallel R)$ is defined. The restrictions allow for a straightforward definition of the operator. We shall see in the case studies chapter that the operator is easy to use and that the restrictions reflect natural design decisions.

6.12 Bi-directional Channels

Bi-directional channels may be modelled in CSP with events of the form $c.i.j$ [Len88]. A communication on a bi-directional channel is described using the notation $c?x!e$, and we have

$$(c?x!e \rightarrow \mathcal{P}_x) \ \ \widehat{=}\ \ ([\!]\ i \in values_{\mathcal{P}}(c) \bullet c.i.e \rightarrow \mathcal{P}_i).$$

This is similar to the rendezvous mechanism in the programming language Ada [WWF87] which allows synchronised value-passing in two directions.

In this section, value-passing action-systems are extended to include bi-directional channels. A bi-directional channel is represented by an action that is parameterised by both x and y, where x is the input parameter and y is the output parameter. The value assigned to y by the action may depend on x. An example of such an action is the following:

$$G \wedge x \in T \rightarrow v, y := f(v, x), f'(v, x).$$

Value-passing action-systems now have the form

$$P \;=\; (A,\ v,\ P_\imath,\ P_A,\ P_H,\ dir,\ type),$$

where dir is a function from A to the set $\{in, out, inout\}$, and $type$ has two components when $dir(a) = inout$: $type_{in}(a)$ and $type_{out}(a)$. Also, when $dir(a) = inout$, P_a has input parameter x and output parameter y.

We must extend the CSP semantics of value-passing action-systems. A bi-directional channel a in action system P will be represented by the set of communications $\{\ a.i \mid i \in \mathcal{W}\ \}$. However, P can only communicate $a.i$ if i is a pair, i.e. $i = (j, k)$ for some j, k. So occurrence of $a.(j, k)$ represents input of value j and output of value k on channel a.

Given a bi-directional channel P_a and a value $i \in \mathcal{W}$, the unparameterised action corresponding to the communication $a.i$ (cf. Definitions 6.1 and 6.3) is given by the following definition:

Definition 6.27 *If $dir(a) = inout$, then*

$$P_{a.i} \;\hat{=}\; (\mathbf{var}\ x, y \bullet [(x, y) = i]\ P_a\ [(x, y) = i]\ ).$$

□

Notice that $P_{a.i} = \mathbf{miracle}$ if i is not a pair.

For $S \subseteq \mathcal{W}$, those states in which $a.S$ may be refused by P is represented by $\neg\ commgd(P_{a.S})$ (cf. Definitions 6.2 and 6.4). For bi-directional channels, $commgd(P_{a.S})$ is defined as follows:

Definition 6.28 *If $dir(a) = inout$, then for $S \subseteq \mathcal{W}$,*

$$\begin{aligned} commgd(P_{a.S}) \;&\hat{=}\; (\exists\, x \in fst(S) \bullet gd(P_a) \wedge wp(P_a, y \in S_x)) \\ &\qquad where\ \ fst(S) \;\hat{=}\; \{\ j \mid (\exists\, k \bullet (j, k) \in S)\ \} \\ &\qquad\qquad\qquad S_j \;\hat{=}\; \{\ k \mid (j, k) \in S\ \}. \end{aligned}$$

□

The definition of the CSP semantics of value-passing action-systems is exactly the same as Definition 6.7, though Definitions 6.27 and 6.28 are now also part of the definition. The semantics is well-formed given certain restrictions.

Theorem 6.29 *For a value-passing action-system P with input, output, and bi-directional channels, $\{\!\!\{P\}\!\!\}$ is well-formed in the infinite-traces model, provided $halt(P_a) \equiv true$, for each $a \in A$, where $dir(a) = out$ or $dir(a) = inout$.*

Proof of this theorem is outlined in Appendix D.

Simulation for action systems with bi-directional channels is given by Definition 6.9, with the fifth condition of Definition 6.9 applying to both output and bi-directional channels (see Appendix D for proof). The simplified simulation conditions of Figure 6.C hold for bi-directional channels also.

Hiding for action systems with bi-directional channels is exactly as given in Definition 6.14 (see Appendix D for proof). Parallel composition of bi-directional channels is left for further study.

6.13 Remarks

Our use of parameterised actions was influenced by the parameterised operations of Z [Spi89]. In Z, an operation is described by a predicate on the initial values and the final values of state variables, similar to the specification statements used here. An operation in Z may also have input parameters, usually denoted '$x?$', and output parameters, usually denoted '$y!$'. Both $y!$ and the final values of the state variables may depend on the initial values and on $x?$.

The problem of ensuring that the choice of output value is made internally has been considered by He Jifeng for transition systems [He90]. His approach is to explicitly identify which transitions represent output communications. Then the failures semantics are defined in such a way that the choice between all output transitions is always internal. This is similar to our approach except that, in He Jifeng's case, the internal choice is not on a "per channel" basis, but rather over the entire set of output communications. This means that a system offering output to the environment on more than one output channel cannot be specified using his approach.

The designers of the specification language SL_0 also treat this problem [Old91]. SL_0 is used for specifying communicating systems which are to be implemented in an occam-like programming language. The semantics of SL_0 are a variant of the *readiness* semantics for CSP [OH86]: the semantics $\mathcal{R}$ of a process with alphabet A consists of a set of trace-ready pairs of the form $t \in A^*, S \subseteq A$, and, $(t, S) \in \mathcal{R}$ means that after engaging in event trace t, the process may be ready to engage in each event in S. To model value-passing programs, SL_0 uses pairs consisting of a trace of communication events and a set of channel names, i.e. $t \in A_{\mathcal{W}}{}^*, S \subseteq A$.

Channels are either input or output so the set of values a program is ready to communicate over a channel c is implicit: if c is an input channel, then c in a ready set S means that a process is ready to accept each value on channel c, while, if c is an output channel, then c in a ready set means that a process is ready to output some value on channel c. To ensure this, the semantics must satisfy the following conditions:

$$\begin{aligned} dir(c) = in \wedge (t, S) \in \mathcal{R} \wedge c \in S &\Rightarrow (\forall\, i \in type(c) \bullet t\langle c.i\rangle \in \mathcal{T}) \\ dir(c) = out \wedge (t, S) \in \mathcal{R} \wedge c \in S &\Rightarrow (\exists\, i \in type(c) \bullet t\langle c.i\rangle \in \mathcal{T}). \end{aligned}$$

It is possible to take a similar approach to the failures semantics of value-passing action-systems: instead of having trace-refusal pairs of the form $t \in {A_{\mathcal{W}}}^*, X \subseteq A_{\mathcal{W}}$, we would have pairs of the form $t \in {A_{\mathcal{W}}}^*, X \subseteq A$, satisfying conditions similar to those above. However, in order to use such a model it would be necessary to define the semantics of hiding and parallel composition in the new model and ensure that they result in well-formed processes and satisfy properties such as monotonicity and associativity. By using the infinite traces model of [Ros88] these results are provided for us.

Chapter 7

Case Studies

This chapter demonstrates how the action system techniques developed in previous chapters may be used in practice. Three case studies involving the specification and refinement of action systems are presented. The first case study is an unordered buffer, the second is a distributed message-passing system, and the third is a distributed file-system.

Interspersed with the case studies, a series of design rules for action systems are presented. These rules are specialisations of action system techniques already presented, and of existing specification and refinement techniques. The rules were suggested by the needs of the case studies, but are general enough to be useful elsewhere.

7.1 Specifying Action Systems

In this section, some notational conventions used for specifying action systems are introduced. Each of the examples in this chapter will be specified as a value-passing action-system. Action system specifications will consist of the following:

- declaration of state variable with invariant
- initialisation statement
- specification of channels with associated type and parameterised action
- specification of internal actions.

A state variable v with invariant I (predicate on v) is declared by

$$\mathbf{var}\ v\ \mathbf{where}\ I.$$

By convention, **var** $v : V$ **where** I will be short for **var** v **where** $v \in V \wedge I$.

Initialisations and parameterised actions will be described using specification statements (Definition 2.3). An initialisation will be specified by notation of the form

$$\textbf{initially} \;\; [\; post \;],$$

where *post* may contain state variable v, but is independent of v_0. We adopt the convention that if v has been declared with invariant I, then the above represents the statement

$$v : [\; I \wedge post \;].$$

In order that an action system be well-formed, its initialisation statement must be non-miraculous. This may be checked with the following rule:

Rule 7.1 *Initialisation statement* $[\; post \;]$ *is non-miraculous if*

$$(\exists\, v \bullet post) \;\;\equiv\;\; true.$$

□

In this and subsequent rules, it is assumed that *post* includes the invariant I.

In parameterised actions, input parameters will be represented by variables ending in '?', and output parameters will be represented by variables ending in '!'. An input channel *left*, along with its type T, and its associated input action parameterised by $x?$, will be described by notation of the form

$$\textbf{chan}\; left \;\textbf{in}\; x? : T :\!- \qquad u : [\; post \;],$$

where *post* is independent of $x?_0$ and u is some subset of state variable v. By convention, the specified action will represent the action

$$u : [\; x? \in T \;\;\wedge\;\; I_0 \;\;\wedge\;\; I \;\;\wedge\;\; post \;],$$

where $I_0 \mathrel{\widehat{=}} I[v \backslash v_0]$. Notice that the action is not allowed to change $x?$, so any reference to $x?$ will be to its initial value.

An output channel *right* will be described by notation of the form

$$\textbf{chan}\; right \;\textbf{out}\; y! : T :\!- \qquad u : [\; post \;],$$

where *post* is independent of $y!_0$. By convention, the specified action will represent the action

$$y!, u : [\; y! \in T \;\;\wedge\;\; I_0 \;\;\wedge\;\; I \;\;\wedge\;\; post \;].$$

Here, the action may change $y!$, and, since *post* is independent of $y!_0$, it doesn't refer to the initial value of $y!$.

A bi-directional channel *bi* will be specified by notation of the form

$$\textbf{chan } bi \textbf{ in } x? : S \textbf{ out } y! : T :- \quad u : [\ post\ ],$$

where *post* is independent of $x?_0$ and $y!_0$. By convention, the specified action will represent the action

$$y!, u : [\ x? \in S \ \wedge\ y! \in T \ \wedge\ I_0 \ \wedge\ I \ \wedge\ post\ ].$$

In order that an action system be well-formed, it must be the case that each of its output actions and bi-directional actions terminates (cf. Theorem 6.29). By definition, actions of the form $u : [\ post\ ]$ always terminate. In addition, because of our convention, output actions always satisfy Property 6.20.

An internal action h will be specified by notation of the from

$$\textbf{internal } \ h :- \quad u : [\ post\ ].$$

Again, by convention, the action part will represent the action

$$u : [\ I_0 \ \wedge\ I \ \wedge\ post\ ].$$

Note: Our treatment of types and invariants is similar to that used by Morgan [Mor89] for the guarded command-language. The difference is that in our case, a statement will be miraculous if the invariant is not satisfied initially, while in Morgan's case a statement will abort. Our treatment ensures that the requirement that output actions and bi-directional actions terminate is satisfied.

7.2 Unordered Buffer

Our first case-study is an unordered buffer. An unordered buffer has two channels: *left* and *right*. It is always ready to accept an input on channel *left*, and to output on channel *right* some value that has been input but not yet output. It does not guarantee to output values in the order in which they are input.

We will describe an unordered buffer with an action system that has a *bag* of values as its state variable. A bag is a collection of elements that may have multiple occurrences of any element. We write **bag** T for the set of finite bags of type T. Bags will be enumerated between bag brackets $\prec$ and $\succ$. *Addition* of bags b, c, is written $b + c$, while *subtraction* is written $b - c$.

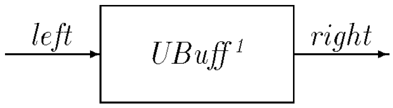
The action system $UBuff^1$ of Figure 7.A describes an unordered buffer that communicates values of type T. The initialisation statement of $UBuff^1$ sets the

$$UBuff^{\mathit{1}} \;\widehat{=}\; \left(\begin{array}{ll} \textbf{var } a : \textbf{bag } T & \\ \textbf{initially } [\, a = \prec\succ \,] & \\ \textbf{chan } \mathit{left} \textbf{ in } x? : T :- & a : [\, a = a_0 + \prec x? \succ \,] \\ \textbf{chan } \mathit{right} \textbf{ out } y! : T :- & a : [\, y! \in a_0 \land a = a_0 - \prec y! \succ \,] \end{array}\right)$$

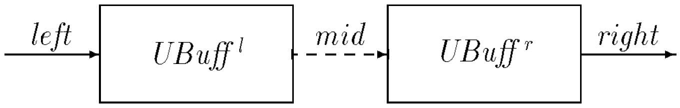
Figure 7.A: Unordered buffer.

bag to be empty, and it can easily be shown by Rule 7.1 that this statement is non-miraculous. The input action *left* accepts input values of type T, adding them to the bag a. Provided a is non-empty, the output action *right* takes some element from a and outputs it.

In the style of Hoare [Hoa85], we can draw a *connection diagram* for $UBuff^{\mathit{1}}$, with channels represented by arrows of appropriate direction, as follows:



We wish to decompose $UBuff^{\mathit{1}}$ into two parallel buffers, $UBuff^{l}$ and $UBuff^{r}$, that communicate with each other via a hidden channel *mid*. This design is represented by the following connection diagram:



Our goal will be to show that

$$UBuff^{\mathit{1}} \;\sqsubseteq\; (UBuff^{l} \parallel UBuff^{r}) \setminus \{mid\}.$$

To reach this goal, we shall introduce action system $UBuff^{\mathit{2}}$ with internal action *mid*. We will show that $UBuff^{\mathit{1}} \sqsubseteq UBuff^{\mathit{2}}$ (refinement step), and then that $UBuff^{\mathit{2}} = (UBuff^{l} \parallel UBuff^{r}) \setminus \{mid\}$ (parallel decomposition step).

Before proceeding with this case study, we introduce some rules that will be used to prove the correctness of the refinement step. We then introduce some rules that will be used to prove the correctness of the parallel-decomposition step.

7.3 Refining Action Systems

In this section, we introduce some rules for refining action systems which are specialisations of the simulation conditions of Figure 6.C to suit our specification notation. Suppose we wish to show that $P \sqsubseteq Q$ where

$$\begin{aligned} P &= (A,\ v,\ P_\imath,\ P_A,\ \{\},\ dir,\ type) \\ Q &= (A,\ w,\ Q_\imath,\ Q_A,\ Q_G,\ dir,\ type). \end{aligned}$$

The simulation conditions of Figure 6.C require a representation function *rep* to transform predicates in the abstract state variable to predicates in the concrete state variable. If a is the abstract variable, and c is the concrete variable, then for ϕ independent of c, we use a representation function of the form

$$rep(\phi) \ \equiv\ (\exists\, a \bullet AI \wedge \phi).$$

Here, AI is an *abstraction invariant* that relates variables a and c. Usually it will be the case that $AI \Rrightarrow I^C$, where I^C is the state invariant of the concrete action system.

In order for simulation to be valid, Theorem 6.10 requires that *rep* satisfies certain conditions. Provided the abstraction invariant is independent of input and output variables $x?$ and $y!$, then it can be easily shown that the above form of *rep* satisfies these conditions.

The six conditions of Figure 6.C were categorised as *data-refinement* conditions (1, 2 and 3), *non-divergence* conditions (4 and 5), and a *progress* condition (6). For each of these categories, we present rules specific to specification statements.

Data-Refinement Conditions

To check data-refinement conditions, we use the *refinement calculator* for specification statements introduced by Morgan & Gardiner [MG88]. The refinement calculator is given by the following rule:

Rule 7.2 *For post independent of c,*

$$a, z : [\ pre,\ post\ ] \ \ \preceq_{rep} \ \ c, z : [\ (\exists\, a \bullet AI \wedge pre),\ (\exists\, a \bullet AI \wedge post)\ ].$$

□

However, Rule 7.2 does not treat a_0 or z_0 as representations of the initial values of a or z. To apply Rule 7.2, we must regard a_0 and z_0 as logical constants so that $a, z : [\ pre,\ post\ ]$ represents

$$(\mathbf{con}\ a_0, z_0 \bullet a, z : [\ a_0 = a \wedge z_0 = z \wedge pre,\ post]),$$

where **con** is defined by

Definition 7.3 *For ϕ independent of x,*

$$wp(\ (\mathbf{con}\ x \bullet S),\ \phi) \quad \widehat{=} \quad (\exists\, x \bullet wp(S, \phi)).$$

□

In this chapter, we will only be interested in specification statements of the form $z : [\ post\ ]$. From Rule 7.2 and Definition 7.3, the following rule can be derived:

Rule 7.4 *Let $AI_0 \mathrel{\widehat{=}} AI[a, c, z \backslash a_0, c_0, z_0]$. Then for post independent of c, c_0,*

$$a, z : [\ post\ ] \quad \preceq_{rep} \quad (\mathbf{con}\ a_0 \bullet c, z : [\ AI_0 \wedge (\exists\, a \bullet AI \wedge post)\ ]\ ).$$

□

To deal with initialisation statements, we have the following rule:

Rule 7.5 *For post independent of a_0, c, c_0,*

$$a, z : [\ post\ ] \quad \preceq'_{rep} \quad c, z : [\ (\exists\, a \bullet AI \wedge post)\ ].$$

□

Morgan & Gardiner show how the refinement calculator may be simplified in the case that the abstraction invariant is of the form

$$AI \quad \equiv \quad (a = F) \wedge I^C,$$

where both F and I^C are independent of a, a_0. Such an abstraction invariant is said to be *functional*, since for any concrete value, there is at most one corresponding abstract value. With a functional abstraction-invariant, we get $rep(\phi) \equiv I^C \wedge \phi[a \backslash F]$. Also, Rules 7.4 and 7.5 may be simplified as follows:

Rule 7.6 *Let $I_0^C \mathrel{\widehat{=}} I^C[c, z \backslash c_0, z_0]$, $F_0 \mathrel{\widehat{=}} F[c, z \backslash c_0, z_0]$. Then for post independent of c, c_0,*

$$a, z : [\ post\ ] \quad \preceq_{rep} \quad c, z : [\ I_0^C \wedge I^C \wedge post[a, a_0 \backslash F, F_0]\ ].$$

□

Rule 7.7 *For post independent of a_0, c, c_0,*

$$a, z : [\ post\ ] \quad \preceq'_{rep} \quad c, z : [\ I^C \wedge post[a \backslash F]\ ].$$

□

The result of application of any of the above rules can then be algorithmically refined, since for statements S, T, T', we have:

$$S \preceq_{rep} T \;\text{ and }\; T \preceq T' \;\text{ implies }\; S \preceq_{rep} T'$$
$$S \preceq'_{rep} T \;\text{ and }\; T \preceq T' \;\text{ implies }\; S \preceq'_{rep} T'.$$

For specification statements, algorithmic refinement is given by the following rule:

Rule 7.8 *If post $\Leftarrow$ post′, then* $\quad u : [\ post\ ] \;\preceq\; u : [\ post'\ ]$.

Non-divergence Conditions

Satisfaction of simulation conditions 4 and 5 of Figure 6.C ensures that the internal actions Q_G of the concrete action-system do not introduce divergence. We must provide a well-founded set WF and a variant E that is an expression in the concrete variables. We then have the following rule:

Rule 7.9 *Conditions 4 and 5 of Figure 6.C are satisfied if:*

(a). $(\exists\, a \bullet AI) \;\Rightarrow\; E \in WF$

(b). *for each* $g \in G$, *where* $Q_g = v : [\ post_g\ ]$,

$$(\exists\, a \bullet AI)[v \backslash v_0] \;\Rightarrow\; (\forall\, v \bullet post_g \Rightarrow E < E[v \backslash v_0]).$$

□

Validity of part (b) follows easily from Definition 2.3. Note: if the abstraction invariant is of the form $AI \equiv (a = F) \wedge I^C$, then $(\exists\, a \bullet AI)$ is simply I^C.

Progress Condition

The sixth simulation-condition of Figure 6.C involves action guards:

$$rep(gd(P_a)) \;\Rightarrow\; gd(Q_a) \vee gd(Q_G), \qquad \textit{each } a \in A. \tag{i}$$

The following rule will be used to calculate the guard of a specification statement:

Rule 7.10 $gd(v : [\ post\ ]) \;\equiv\; (\exists\, v, y! \bullet post)[v_0 \backslash v]$.

Here, $y!$ is only relevant when $v : [\ post\ ]$ is an output or a bi-directional action.

$$
UBuff^{2} \;\widehat{=}\; \left(\begin{array}{l}
\textbf{var } b, c : \textbf{bag } T \\
\textbf{initially } \;[\; b = c = \prec\succ \;] \\
\textbf{chan } \mathit{left} \textbf{ in } x? : T :- \quad b : [\; b = b_0 + \prec x?\succ \;] \\
\textbf{chan } \mathit{right} \textbf{ out } y! : T :- \quad c : [\; y! \in c_0 \wedge c = c_0 - \prec y!\succ \;] \\
\textbf{internal } \mathit{mid} :- \quad b, c : \left[\begin{array}{l} (\exists\, x \in b_0 \bullet b = b_0 - \prec x\succ \\ \qquad\qquad \wedge\; c = c_0 + \prec x\succ) \end{array} \right]
\end{array} \right)
$$

Figure 7.B: Unordered buffer with internal action.

7.4 Refinement of Buffer

$UBuff^{2}$ is specified in Figure 7.B. In $UBuff^{2}$, the input action places input values in bag b, while the output action takes output values from bag c. Values are moved from b to c by the internal action mid, which is enabled as long as b is non-empty.

To show that $UBuff^{1} \sqsubseteq UBuff^{2}$, we use the functional abstraction invariant

$$AI \;\widehat{=}\; I^{2} \wedge a = b + c,$$

where I^{2} is the state invariant of $UBuff^{2}$, i.e. $b, c \in \textbf{bag } T$.

In the following, we write $init^{i}$ for the initialisation of $UBuff^{i}$, and $left^{i}$ for the $left$ action of $UBuff^{i}$. Similarly for $right$ and mid. We write I^{i} for the state invariant of $UBuff^{i}$, and I_0^{i} for I^{i} with state variables replaced by their 0-subscripted version. We must show that each of the six conditions of Figure 6.C are satisfied.

Data-Refinement Conditions

1. The initialisations are related by $\preceq'_{rep}$:

$$
\begin{array}{rcll}
init^{1} & = & a : [\; a \in \textbf{bag } T \wedge a = \prec\succ \;] & \\
 & \preceq'_{rep} & b, c : [\; I^{2} \wedge (a \in \textbf{bag } T \wedge a = \prec\succ)[a \backslash b + c] \;] & \textit{Rule 7.7} \\
 & \preceq & b, c : [\; b, c \in \textbf{bag } T \wedge b + c = \prec\succ \;] & \textit{Rule 7.8} \\
 & \preceq & b, c : [\; b, c \in \textbf{bag } T \wedge b = c = \prec\succ \;] & \textit{Rule 7.8} \\
 & = & init^{2}. &
\end{array}
$$

2. Commonly-labelled actions are related by $\preceq_{rep}$:

$$\begin{array}{lll}
\mathit{left}^1 & = & a : \left[\begin{array}{l} I_0^1 \wedge I^1 \wedge x? \in T \\ \wedge\ a = a_0 + \prec x? \succ \end{array} \right] \\[2ex]
 & \preceq_{rep} & b, c \left[\begin{array}{l} I_0^2 \wedge I^2 \wedge \\ (b_0 + c_0), (b + c) \in \mathbf{bag}\ T \wedge x? \in T \\ \wedge\ (b + c) = (b_0 + c_0) + \prec x? \succ \end{array} \right] \qquad \mathit{Rule\ 7.6} \\[2ex]
 & \preceq & b, c : \left[\begin{array}{l} b_0, c_0, b, c \in \mathbf{bag}\ T \wedge x? \in T \\ \wedge\ b = b_0 + \prec x? \succ \wedge c = c_0 \end{array} \right] \\[2ex]
 & = & \mathit{left}^2.
\end{array}$$

Similarly, $\mathit{right}^1 \preceq_{rep} \mathit{right}^2$.

3. The internal action of UBuff^2 refines **skip**:

$$\begin{array}{lll}
\mathbf{skip} & = & a : [\ a = a_0\ ] \\[2ex]
 & \preceq_{rep} & b, c : [\ I_0^2 \wedge I^2 \wedge b + c = b_0 + c_0\ ] \qquad \mathit{Rule\ 7.6} \\[2ex]
 & \preceq & b, c : \left[\begin{array}{l} I_0^2 \wedge I^2 \wedge (\exists\, x \in b_0 \bullet b = b_0 - \prec x? \succ \\ \qquad\qquad\qquad\qquad \wedge\ c = c_0 + \prec x? \succ) \end{array} \right] \\[2ex]
 & \preceq & \mathit{mid}^2.
\end{array}$$

Non-divergence Conditions

We use the size of bag b, written $\#b$, as a variant, with $\mathbb{N}$ as a well-founded set. Since b is always a finite bag, $\#b$ is always in $\mathbb{N}$:

$$I^2 \Rrightarrow b \in \mathbf{bag}\ T \Rrightarrow \#b \in \mathbb{N}.$$

It can be seen here why we require I^2 to be part of the abstraction invariant. To show that the internal action of UBuff^2 always decreases $\#b$, we use Rule 7.9 and proceed as follows (we write $\mathit{post}_{\mathit{mid}^2}$ for the postcondition of action mid^2):

$$\begin{array}{ll}
 & (\forall\, b, c \bullet \mathit{post}_{\mathit{mid}^2} \Rightarrow \#b < \#b_0) \\[2ex]
\Lleftarrow & (\forall\, b, c \bullet I_0^2 \wedge I^2 \wedge (\exists\, x \in b_0 \bullet b = b_0 - \prec x \succ \wedge c = c_0 + \prec x \succ) \\
 & \qquad\qquad \Rightarrow \#b < \#b_0) \\[2ex]
\Lleftarrow & (\forall\, b \bullet (b_0, b \in \mathbf{bag}\ T \wedge \#b = \#b_0 - 1) \Rightarrow \#b < \#b_0) \\[2ex]
\Lleftarrow & \mathit{true}.
\end{array}$$

Thus the non-divergence rules of Figure 6.C are satisfied.

Progress Condition

We must demonstrate the following:

$$rep(gd(left^1)) \Rrightarrow gd(left^2) \vee gd(mid^2) \qquad \text{(ii)}$$
$$rep(gd(right^1)) \Rrightarrow gd(right^2) \vee gd(mid^2). \qquad \text{(iii)}$$

We calculate the guard of $left^1$ as follows:

$$\begin{array}{lll} & gd(left^1) & \\ \equiv & (\exists\, a \bullet I_0^1 \wedge I^1 \wedge x? \in T \wedge a = a_0 + \prec x?\succ)[a_0 \backslash a] & \textit{Rule 7.10} \\ \equiv & (I_0^1 \wedge x? \in T)[a_0 \backslash a] & \textit{1-pt rule} \\ \equiv & I^1 \wedge x? \in T. & \end{array}$$

Similarly, $gd(left^2) \equiv I^2 \wedge x? \in T$. So it is easy to see that (ii) is satisfied.

We calculate the guard of $right^1$ as follows:

$$\begin{array}{lll} & gd(right^1) & \\ \equiv & (\exists\, a, y! \bullet I_0^1 \wedge I^1 \wedge y! \in a_0 \wedge a = a_0 - \prec y!\succ)[a_0 \backslash a] & \textit{Rule 7.10} \\ \equiv & (I_0^1 \wedge a_0 \neq \prec\succ)[a_0 \backslash a] & \\ \equiv & I^1 \wedge a \neq \prec\succ. & \end{array}$$

Similarly,

$$\begin{array}{rcl} gd(right^2) & \equiv & I^2 \wedge c \neq \prec\succ \\ gd(mid^2) & \equiv & I^2 \wedge b \neq \prec\succ. \end{array}$$

Now, (iii) is satisfied since

$$\begin{array}{ll} & rep(gd(right^1)) \\ \Rrightarrow & rep(I^1 \wedge a \neq \prec\succ) \\ \Rrightarrow & I^2 \wedge (b + c) \neq \prec\succ \\ \Rrightarrow & (I^2 \wedge b \neq \prec\succ) \;\vee\; (I^2 \wedge c \neq \prec\succ) \\ \Rrightarrow & gd(mid^2) \vee gd(right^2). \end{array}$$

7.5 Parallel-Decomposition Rule

In this section, we introduce a rule for parallel decomposition of action systems. This rule will be used to show that

$$P \;=\; (Q \parallel R) \setminus H,$$

where P has internal actions H, and Q and R have no internal actions and only have the actions H in common.

Assume P, Q, and R are as follows:

$$\begin{array}{lcl} P &=& (A \cup B,\ (v, w),\ P_\imath,\ P_{A\cup B},\ P_H,\ dir_P,\ type_P) \\ Q &=& (A \cup H,\ v,\ Q_\imath,\ Q_{A\cup H},\ \{\},\ dir_Q,\ type_Q) \\ R &=& (B \cup H,\ w,\ R_\imath,\ R_{B\cup H},\ \{\},\ dir_R,\ type_R). \end{array}$$

Assume that A and B are disjoint. Let $IN_Q \mathrel{\hat{=}} \{h \in H \mid dir_Q(h) = in\}$, $OUT_Q \mathrel{\hat{=}} \{h \in H \mid dir_Q(h) = out\}$, $INOUT_Q \mathrel{\hat{=}} \{h \in H \mid dir_Q(h) = inout\}$, and similarly for IN_R, OUT_R, and $INOUT_R$. Assume that $INOUT_Q = \{\}$, $INOUT_R = \{\}$, $IN_Q = OUT_R$, and $OUT_Q = IN_R$. Assume also that the types of OUT_Q and OUT_R are respectively compatible with those of IN_R and IN_Q, and that P, Q, and R agree on the direction and type of channels in $A \cup B$. Based on Definitions 6.14 (hiding) and 6.22 (parallel composition), we then have the following rule for parallel decomposition:

Rule 7.11 $P \;=\; (Q \parallel R) \setminus H$, *provided*

1. $P_\imath = Q_\imath \parallel R_\imath$
2. $P_A = Q_A$ *and* $P_B = R_B$
3. *for each* $h \in IN_Q$, *input action* Q_h *satisfies Property 6.19, and for each* $h \in IN_R$, *input action* R_h *satisfies Property 6.19*
4. $$\begin{array}{lcll} P_h &=& (\mathbf{var}\ y! \bullet Q_h \overrightarrow{\parallel} R_h), & \textit{each } h \in OUT_Q \\ P_h &=& (\mathbf{var}\ y! \bullet R_h \overrightarrow{\parallel} Q_h), & \textit{each } h \in OUT_R. \end{array}$$

□

The parallel-composition operator for statements $\parallel$ must satisfy Property 5.1, while the value-passing version $\overrightarrow{\parallel}$ must satisfy Property 6.21. It can easily be shown that the operators on specification statements defined by the following rules, satisfy those properties respectively:

Rule 7.12 *For u-predicate post1, v-predicate post2, where u and v are distinct, and u' and v' are subsets of u and v respectively,*

$$u' : [\ \mathit{post1}\ ] \ \ \| \ \ v' : [\ \mathit{post2}\ ] \ \ \widehat{=} \ \ u', v' : [\ \mathit{post1} \wedge \mathit{post2}\ ].$$

□

Rule 7.13 *For $(y!, u_0, u)$-predicate post1, $(x?, v_0, v)$-predicate post2, where u and v are distinct, and u' and v' are subsets of u and v respectively,*

$$u' : [\ \mathit{post1}\ ] \ \ \vec{\|} \ \ v' : [\ \mathit{post2}\ ] \ \ \widehat{=} \ \ u', v' : [\ \mathit{post1} \wedge \mathit{post2}[x?\backslash y!]\ ].$$

□

Condition 3 of Rule 7.11 above mentions Property 6.19. Recall this property ensures that an input action accepts exactly those input values in its type whenever it is enabled. To show that an input action satisfies Property 6.19, the following rule will be used:

Rule 7.14 **chan** *left* **in** $x? : T :- \ \ v : [\ \mathit{post}\ ]$ *satisfies Property 6.19 provided*

$$(\exists\, x?, v \bullet \mathit{post}) \wedge x? \in T \ \ \Rrightarrow \ \ (\exists\, v \bullet \mathit{post}).$$

□

To localise input variables, the following rule will be used:

Rule 7.15 $(\mathbf{var}\ y! \bullet v : [\ \mathit{post}\ ]) \ = \ v : [\ (\exists\, y! \bullet \mathit{post})\ ].$

7.6 Parallel Decomposition of Buffer

The action systems $\mathit{UBuff}^{\,l}$ and $\mathit{UBuff}^{\,r}$ are specified in Figure 7.C. To show that

$$\mathit{UBuff}^{\,2} \ = \ (\mathit{UBuff}^{\,l} \parallel \mathit{UBuff}^{\,r}) \setminus \{\mathit{mid}\},$$

we only need to show, by Rule 7.11, that

1. $\mathit{init}^2 = \mathit{init}^l \parallel \mathit{init}^r$
2. $\mathit{left}^2 = \mathit{left}^l$ and $\mathit{right}^2 = \mathit{right}^r$
3. mid^r satisfies Property 6.19
4. $\mathit{mid}^2 = (\mathbf{var}\ y! \bullet \mathit{mid}^l \,\vec{\|}\, \mathit{mid}^r)$.

$$
UBuff^{l} \;\widehat{=}\; \left(\begin{array}{ll}
\mathbf{var}\ b : \mathbf{bag}\ T & \\
\mathbf{initially}\ [\ b = \prec\succ\ ] & \\
\mathbf{chan}\ \mathit{left}\ \mathbf{in}\ x? : T :- & b : [\ b = b_0 + \prec x?\succ\ ] \\
\mathbf{chan}\ \mathit{mid}\ \mathbf{out}\ y! : T :- & b : [\ y! \in b_0 \wedge b = b_0 - \prec y!\succ\ ]
\end{array}\right)
$$

$$
UBuff^{r} \;\widehat{=}\; \left(\begin{array}{ll}
\mathbf{var}\ c : \mathbf{bag}\ T & \\
\mathbf{initially}\ [\ c = \prec\succ\ ] & \\
\mathbf{chan}\ \mathit{mid}\ \mathbf{in}\ x? : T :- & c : [\ c = c_0 + \prec x?\succ\ ] \\
\mathbf{chan}\ \mathit{right}\ \mathbf{out}\ y! : T :- & c : [\ y! \in c_0 \wedge c = c_0 - \prec y!\succ\ ]
\end{array}\right)
$$

Figure 7.C: Parallel buffers.

It is easy to see that 2 is satisfied. It can be shown that 1 is satisfied by Rule 7.12, and that 3 is satisfied using Rule 7.14 with

$$
\mathit{post} \;\equiv\; c_0, c \in \mathbf{bag}\ T \wedge x? \in T \wedge c = c_0 + \prec x?\succ .
$$

Finally, 4 is satisfied since

$$
\begin{array}{cll}
& (\mathbf{var}\ y! \bullet \mathit{mid}^{l} \overrightarrow{\|}\, \mathit{mid}^{r}) & \\
= & (\mathbf{var}\ y! \bullet\ \ b, c\ :\ \left[\begin{array}{l} b_0, b \in \mathbf{bag}\ T \wedge y! \in b_0 \wedge b = b_0 - \prec y!\succ \wedge \\ (c_0, c \in \mathbf{bag}\ T \wedge c = c_0 + \prec x?\succ)[x?\backslash y!] \end{array}\right]) & \mathit{Rule\ 7.13} \\
= & b, c : [\ I_0^2 \wedge I^2 \wedge (\exists\, x \in b_0 \bullet b = b_0 - \prec x\succ \wedge c = c_0 + \prec x\succ)\ ] & \mathit{Rule\ 7.15} \\
= & \mathit{mid}^2. &
\end{array}
$$

7.7 Simple Refinement of Action Systems

In the case that neither the abstract nor the concrete action-system has internal actions, the conditions of Figure 6.C are simplified by the following rule:

Rule 7.16 *When P and Q have no internal actions, $P \sqsubseteq Q$ if*

1. $P_\imath \preceq'_{rep} Q_\imath$

$$Buff^l \;\widehat{=}\; \left(\begin{array}{l} \textbf{var}\ bs : \textbf{seq}\ T \\ \textbf{initially}\ \ [\ bs = \langle\rangle\ ] \\ \textbf{chan}\ \mathit{left}\ \textbf{in}\ x? : T :- \qquad bs : [\ bs = bs_0 \langle x? \rangle\ ] \\ \textbf{chan}\ \mathit{mid}\ \textbf{out}\ y! : T :- \qquad bs : [\ y! = hd(bs_0) \wedge bs = tl(bs_0)\ ] \end{array}\right)$$

Figure 7.D: Ordered buffer.

2. $P_a \preceq_{rep} Q_a$, $\qquad$ *each* $a \in A$

3. $rep(gd(P_a)) \Rrightarrow gd(Q_a)$, $\qquad$ *each* $a \in A$.

□

This rule can be used to refine $UBuff^l$ and $UBuff^r$. $Buff^l$ of Figure 7.D is a buffer that outputs values in the same order in which they are input. A sequence is used to maintain the order of the values.

To show that $UBuff^l \sqsubseteq Buff^l$, we use Rule 7.16 with the abstraction invariant

$$AI \;\equiv\; bs \in \textbf{seq}\ T \wedge b = bag(bs),$$

where *bag* is the function that converts a sequence to a bag, maintaining multiple occurrences of elements. The proof is straightforward, and details are omitted.

It can also be shown that $UBuff^r \sqsubseteq Buff^r$, where $Buff^r$ is an ordered version of $UBuff^r$. Since parallel composition and hiding are monotonic, we then have

$$UBuff^1 \;\sqsubseteq\; (Buff^l \parallel Buff^r) \setminus \{mid\}.$$

7.8 More Specification Techniques

This section presents some more specification techniques that will be used in the second and third case-studies.

Invariants

Z *schemas* [Spi89] will sometimes be used to specify the state variables and invariant of an action system. A schema *Sch* takes the form

$$\begin{array}{|l}
\hline
\textit{Sch} \\
v : T \\
\hline
I \\
\hline
\end{array}$$

where v is a state variable, T is a type, and I is a predicate. If Sch is defined as shown, then the declaration **var** Sch will be short for

$$\textbf{var } v : T \textbf{ where } I.$$

Array Variables

An array of type T with index set D may be represented by a function $f \in D \rightarrow T$. ($D \rightarrow T$ represents the set of total functions from D to T.) A specification statement can be made to update f at index i by containing the formula

$$f \quad = \quad f_0 \oplus \{i \mapsto E\},$$

where $f_0 \oplus \{i \mapsto E\}$ represents the overriding of function f_0 with the function $\{i \mapsto E\}$. For convenience, $f = f_0 \oplus \{i \mapsto E\}$ will be written simply as

$$f(i) \quad = \quad E.$$

Similarly, in substitutions, $[f(i) \backslash E]$ will be short for $[f \backslash f_0 \oplus \{i \mapsto E\}]$.

Since $f \in D \rightarrow T$ is well-defined for each $i \in D$, f may be regarded as a set of independent variables $\{ f_i \mid i \in D \}$. So a statement such as

$$f : [\ f(i) = E\ ]$$

can be written simply as

$$f_i : [\ f_i = E\ ].$$

Indexed Channel-Sets

The action systems of the next case study will contain indexed sets of channels, each one offering similar behaviour. An indexed statement will be used to specify the actions associated with such channel sets. For example, to specify an indexed set of input channels $\{ left_i \mid i \in F \}$, with associated types and actions, the following notation will be used:

$$\textbf{for } i \in F \textbf{ chan } left_i \textbf{ in } x_i? : T_i :- \qquad u_i : [\ post_i\ ].$$

The intention is that the i-indexed statement represents a set of input actions. Similarly for an indexed set of output channels and an indexed set of bi-directional channels. An indexed set of internal actions $\{ h_i \mid i \in F \}$ will be specified by

$$\textbf{for } i \in F \textbf{ internal } \ h_i :- \quad u_i : [\ post_i\ ].$$

Because of the meaning given to internal actions by Definition 6.7, a set of internal actions $\{ h_i \mid i \in F \}$ is the same as the choice over that set $([\!] \ i \in F \bullet h_i)$. So we have the following rule:

Rule 7.17 *An internal action*

$$\textbf{internal } \ h :- \quad u : [\ (\exists\, i \in F \bullet post_i)\ ]$$

is the same as the set of internal actions

$$\textbf{for } i \in F \textbf{ internal } \ h_i :- \quad u : [\ post_i\ ].$$

□

7.9 Message-Passing System

Our second case-study is a message-passing system. We suppose that a message-passing system allows a set of users to exchange messages amongst each other. Each user resides at a node, and each user may engage in either a *send* action, or a *receive* action.

Let $Node$ represent the set of nodes in the system. We shall assume that $Node$ is finite. Let $Mess$ represent the type of messages that may be exchanged, and let Env be the cartesian product of $Node$ and $Mess$, i.e.

$$Env \quad \widehat{=} \quad Node \times Mess.$$

In the pair $(r, m) \in Env$, r is the recipient node, m is the message, and we say that (r, m) is an *envelope*.

The initial specification of the message-passing system, MPS^1, is given in Figure 7.E. Variable $mail$ contains all messages sent but not yet received. Initially $mail$ is empty. For each node n, there is a $send_n$ action and a $receive_n$ action. Action $send_s$ accepts an envelope $(r?, m?)$ at node s and adds it to the bag $mail$. If there is at least one message for recipient r in $mail$, then action $receive_r$ chooses one of these messages and outputs it.

Our goal is to implement MPS^1 as a *store-and-forward* network, where not all nodes are directly connected, and envelopes may pass through a number of

$$
\begin{array}{l}
\mathit{MPS}^1 \;\widehat{=} \\
\left(\begin{array}{l}
\mathbf{var}\ \mathit{mail} : \mathbf{bag}\ \mathit{Env} \\
\mathbf{initially}\ \ [\ \mathit{mail} = \prec\succ\ ] \\
\\
\mathbf{for}\ s \in \mathit{Node}\ \mathbf{chan}\ \mathit{send}_s\ \mathbf{in}\ (r?, m?) : \mathit{Env} :- \\
\qquad \mathit{mail} : [\ \mathit{mail} = \mathit{mail}_0 + \prec(r?, m?)\succ\ ] \\
\\
\mathbf{for}\ r \in \mathit{Node}\ \mathbf{chan}\ \mathit{receive}_r\ \mathbf{out}\ m! : \mathit{Mess} :- \\
\qquad \mathit{mail} : [\ (r, m!) \in \mathit{mail}_0 \wedge \mathit{mail} = \mathit{mail}_0 - \prec(r, m!)\succ\ ]
\end{array}\right)
\end{array}
$$

Figure 7.E: Message-passing system.

intermediate nodes before reaching their recipient. In the first refinement step, we introduce data structures more closely resembling the store-and-forward architecture, and introduce internal actions for passing envelopes between these data structures. Then the system is decomposed into a set of *agents*, one residing at each node, and a set of *media*, by which the agents communicate.

7.10 First Refinement of MPS

In MPS^2, *mail* will be replaced by a set of stores, one per node, and a set of buffers representing direct links. The constant relation $\mathit{net} \in \mathit{Node} \leftrightarrow \mathit{Node}$ will represent the connectivity of the network: $(a, b) \in \mathit{net}$ will mean there is a direct communications link from node a to node b.

Routing relations will be used to determine which intermediate nodes an envelope may pass through. Before defining a *route*, we present some graph theory concepts [Ore62]. We say that a *graph* G is a relation on a set of nodes N (e.g. *net* is a graph on *Node*). A *path* from a to b in G is a non-empty sequence p of nodes from N, such that $p_i G p_{i+1}$, for $0 \leq i < \#p - 1$, and $p_0 = a$ and $p_{\#p-1} = b$. Let G^* be the reflexive, transitive closure of G. Then aG^*b means there is a path from a to b in G. Note that there is always a path from a to a in G. An *arc* from a to b in G is a path from a to b in which all nodes are distinct. If N is finite, then the *elongation* from a to b in G, written $e_G(a, b)$, is the length of the longest arc from a to b in G. Since the only arc from a to a is $\langle a \rangle$, we have $e_G(a, a) = 1$. We introduce *routes* as follows:

Definition 7.18 *Let G be a graph on nodes N. Then $Routes(G)$, the set of* routes *of G, is the set of subgraphs of G such that for all $R \in Routes(G)$, and all $a, b, c \in N$, where $a \neq c$,*

$$aRb \wedge bR^*c \quad \Rightarrow \quad e_R(a, c) > e_R(b, c).$$

□

Here, each $R \in Routes(G)$ is a routing relation, (a, b) is a single step in R, and c is a destination node. The definition says that as we move from node a to node b on the route, the elongation to the destination node decreases.

MPS^2 will contain a fixed set of routes, each one uniquely identified by a tag from a set Tag. These routes will be represented by the constant function

$$route \quad \in \quad Tag \rightarrow Routes(net).$$

On input, each envelope will be assigned one of these routes by being tagged with the route identifier. At any point on its journey the choice of the next node to which an envelope is sent will be determined by its destination and its assigned route. Since a route is a relation, the choice of next node may be nondeterministic. We shall use elongations as a variant to ensure that all envelopes eventually reach their destination.

The state variables of MPS^2, with invariant, are declared by the following schema:

$$\begin{array}{l}
\underline{\quad INV^2 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \\
store \;:\; Node \rightarrow \mathbf{bag}\; Tag \times Env \\
link \;:\; net \rightarrow \mathbf{bag}\; Tag \times Env \\
\hline
(\forall (i, r, m) \in Tag \times Env;\;\; a, b \in Node \bullet \\
\qquad (i, r, m) \in store(a) \Rightarrow (a, r) \in route(i)^* \\
\qquad (a, b) \in net \wedge (i, r, m) \in link(a, b) \Rightarrow (b, r) \in route(i)^*) \\
\hline
\end{array}$$

Corresponding to each node in the network, there is a store (bag) of tagged envelopes. These are modelled by the variable *store*. Corresponding to each direct link in the network, there is an unordered buffer of tagged envelopes. These are modelled by the variable *link*. The predicate part of INV^2 means that there is always a path from the current position of an envelope to its recipient in the assigned route. In order that each distinct pair of nodes be connected by at least one route, we shall assume that the constant function *route* satisfies:

$$(\cup i \in Tag \bullet route(i)^*) \quad = \quad Node \times Node.$$

$$
\begin{array}{l}
MPS^2 \;\widehat{=} \\
\left(\begin{array}{l}
\textbf{var } INV^2 \\
\textbf{initially} \;\; \left[\begin{array}{l} (\forall\, a \in Node \bullet store(a) = \prec\succ) \\ \wedge\, (\forall (a,b) \in net \bullet link(a,b) = \prec\succ) \end{array}\right] \\[2ex]
\textbf{for } s \in Node \textbf{ chan } send_s \textbf{ in } (r?, m?) : Env :- \\
\qquad store \;:\; \left[\begin{array}{l} (\exists\, i \in Tag \bullet (s, r?) \in route(i)^* \\ \qquad \wedge\, store(s) = store_0(s) + \prec(i, r?, m?)\succ\ ) \end{array}\right] \\[2ex]
\textbf{for } r \in Node \textbf{ chan } receive_r \textbf{ out } m! : Mess :- \\
\qquad store \;:\; \left[\begin{array}{l} (\exists\, i \bullet (i, r, m!) \in store_0(r) \\ \qquad \wedge\, store(r) = store_0(r) - \prec(i, r, m!)\succ\ ) \end{array}\right] \\[2ex]
\textbf{internal } \; forward :- \\
\qquad \begin{array}{r} store, \\ link \end{array} \;:\; \left[\begin{array}{l} (\exists (a,b) \in net;\; (i, r, m) \in store_0(a) \bullet \\ \qquad r \neq a \wedge (a,b) \in route(i) \wedge (b, r) \in route(i)^* \\ \qquad \wedge\, store(a) = store_0(a) - \prec(i, r, m)\succ \\ \qquad \wedge\, link(a,b) = link_0(a,b) + \prec(i, r, m)\succ\ ) \end{array}\right] \\[2ex]
\textbf{internal } \; relay :- \\
\qquad \begin{array}{r} store, \\ link \end{array} \;:\; \left[\begin{array}{l} (\exists (a,b) \in net;\; (i, r, m) \in link_0(a,b) \bullet \\ \qquad \wedge\, link(a,b) = link_0(a,b) - \prec(i, r, m)\succ \\ \qquad \wedge\, store(b) = store_0(b) + \prec(i, r, m)\succ\ ) \end{array}\right]
\end{array}\right)
\end{array}
$$

Figure 7.F: Message-passing system with internal actions.

MPS^2 is then specified by Figure 7.F. All stores and links are initially empty. The action $send_s$ accepts an envelope $(r?, m?)$, chooses a route i that connects s to $r?$, and adds $(i, r?, m?)$ to the bag $store(s)$. If there is at least one message for recipient r in $store(r)$, then action $receive_r$ chooses one of those messages and outputs it.

The internal action *forward* takes a tagged envelope that has not yet reached its recipient from some $store(a)$, chooses the next node b to forward the envelope to, and places the envelope in $link(a, b)$. The internal action *relay* simply takes an envelope from some $link(a, b)$ and places it in $store(b)$.

By a sequence of *forward* and *relay* actions, a message sent at node s is eventually delivered to the store of its recipient node r. This is the case since $MPS^1 \sqsubseteq MPS^2$, as we will now show.

Data-Refinement Conditions

The abstract and the concrete variables will be related by equating *mail* with the sum of envelopes in each store and each link. We write $(\Sigma i \bullet b_i)$ for the summation of a set of bags b_i. Let *env* be the function that removes tags from tagged envelopes, i.e. $env(i, r, m) = (r, m)$. If b is a bag of tagged envelopes, then $env(b)$ is the corresponding bag of untagged envelopes. The abstract invariant AI is defined as follows:

$$\begin{aligned} AI \quad \widehat{=} \quad INV^2 \ \wedge \ mail = \ & (\Sigma a \in Node \bullet env(store(a))) \\ & +(\Sigma(a, b) \in net \bullet env(link(a, b))). \end{aligned}$$

The data-refinement obligations are as follows:

$$\begin{aligned} init^1 &\preceq'_{rep} init^2 \\ send^1_s &\preceq_{rep} send^2_s \\ receive^1_r &\preceq_{rep} receive^2_r \\ \mathbf{skip} &\preceq_{rep} forward^2 \\ \mathbf{skip} &\preceq_{rep} relay^2. \end{aligned}$$

These are easily proven using the data-refinement calculators and properties of bags.

Non-divergence Conditions

A variant that is decreased by both $forward^2$ and $relay^2$ must be chosen. We will use the elongation from the current position of each envelope to its destination in its route to define a variant. We will show that the characterising property of routes (Definition 7.18) is necessary to ensure that this variant is decreased by $forward^2$. Let $(\Sigma j \bullet n_j)$ represent the summation of a set of naturals n_j, and let $e_i(a, b)$ be the elongation from a to b on $route(i)$, i.e. $e_{route(i)}(a, b)$. The variant E is then defined as follows:

$$\begin{aligned} E \quad \widehat{=} \quad & (\Sigma a \in Node \bullet (\Sigma(i, r, m) \in store(a) \bullet e_i(a, r) * 2)) \\ & +(\Sigma(a, b) \in net \bullet (\Sigma(i, r, m) \in link(a, b) \bullet (e_i(b, r) * 2) + 1)). \end{aligned}$$

It is easy to show that $INV^2 \Rrightarrow E \in \mathbb{N}$. Let $E_0 \mathrel{\widehat{=}} E[store, link \backslash store_0, link_0]$. Using Rule 7.9 to prove that $forward^2$ decreases E, we proceed as follows:

$$(\forall\, store, link \bullet post_{forward^2} \Rightarrow E < E_0)$$

$$\Leftarrow \quad (\forall\, store, link \bullet \left(\begin{array}{l} \exists (a,b) \in net;\ (i,r,m) \in store(a) \bullet \\ r \neq a \wedge (a,b) \in route(i) \wedge (b,r) \in route(i)^* \\ \wedge\ store(a) = store_0(a) - \prec(i,r,m)\succ \\ \wedge\ link(a,b) = link_0(a,b) + \prec(i,r,m)\succ \end{array}\right) \Rightarrow E < E_0)$$

$$\Leftarrow \quad (\forall\, store, link \bullet (\forall (a,b) \in net;\ (i,r,m) \in store(a) \bullet \left(\begin{array}{l} r \neq a \wedge (a,b) \in route(i) \wedge (b,r) \in route(i)^* \\ \wedge\ store(a) = store_0(a) - \prec(i,r,m)\succ \\ \wedge\ link(a,b) = link_0(a,b) + \prec(i,r,m)\succ \end{array}\right) \Rightarrow E < E_0))$$

$$\Leftarrow \quad (\forall (a,b) \in net, (i,r,m) \in store(a) \bullet (r \neq a \wedge (a,b) \in route(i) \wedge (b,r) \in route(i)^*) \Rightarrow E\left[\begin{array}{l} store(a) \backslash store_0(a) - \prec(i,r,m)\succ, \\ link(a,b) \backslash link_0(a,b) + \prec(i,r,m)\succ \end{array}\right] < E_0)$$

1-pt Rule

$$\Leftarrow \quad (\forall (a,b) \in net, (i,r,m) \in store(a) \bullet (r \neq a \wedge (a,b) \in route(i) \wedge (b,r) \in route(i)^*) \Rightarrow (E_0 - (e_i(a,r) * 2)) + (e_i(b,r) * 2) + 1 < E_0)$$

by definition of E

$$\Leftarrow \quad (\forall\, a, b, r \in Node, i \in Tag \bullet (r \neq a \wedge (a,b) \in route(i) \wedge (b,r) \in route(i)^*) \Rightarrow e_i(b,r) < e_i(a,r)).$$

Now, since $route(i) \in Routes(net)$, this final predicate is equivalent to *true* by Definition 7.18.

Note: Definition 7.18 was derived by firstly attempting the above proof, and then requiring that routes should satisfy the final predicate of the proof.

The proof that $relay^2$ decreases E is a simpler version of the above proof.

Progress Conditions

We must show that

$$rep(gd(send_s^1)) \quad \Rrightarrow \quad gd(send_s^2) \vee gd(FR) \qquad \text{(iv)}$$

$$rep(gd(receive_r^1)) \;\Rrightarrow\; gd(receive_r^2) \vee gd(FR), \tag{v}$$

where $FR \mathrel{\widehat{=}} \{forward^2, relay^2\}$. By calculation, the action guards are as follows:

$$\begin{aligned}
gd(send_s^1) &\equiv mail \in \mathbf{bag}\ Env \wedge (r?, m?) \in Env \\
gd(receive_r^1) &\equiv mail \in \mathbf{bag}\ Env \wedge (\exists\, m \bullet (r, m) \in mail) \\
gd(send_s^2) &\equiv INV^2 \wedge (r?, m?) \in Env \\
gd(receive_r^2) &\equiv INV^2 \wedge (\exists\, i, m \bullet (i, r, m) \in store(r)) \\
gd(forward^2) &\equiv INV^2 \wedge (\exists\, a, i, r, m \bullet a \neq r \wedge (i, r, m) \in store(a)) \\
gd(relay^2) &\equiv INV^2 \wedge (\exists\, a, b, i, r, m \bullet (i, r, m) \in link(a, b)).
\end{aligned}$$

It is easy to see that (iv) is satisfied, while (v) is satisfied since:

$$\begin{aligned}
& rep(gd(receive_r^1)) \\
\Rrightarrow\;& rep(mail \in \mathbf{bag}\ Env \wedge (\exists\, m \bullet (r, m) \in mail)) \\
\Rrightarrow\;& INV^2 \wedge (\exists\, m \bullet (r, m) \in (\Sigma a \bullet env(store(a))) + (\Sigma a, b \bullet env(link(a, b)))) \\
\Rrightarrow\;& INV^2 \wedge (\exists\, a, i, m \bullet (i, r, m) \in store(a)) \\
& \vee\ INV^2 \wedge (\exists\, a, b, i, m \bullet (i, r, m) \in link(a, b)) \\
\Rrightarrow\;& INV^2 \wedge (\exists\, i, m \bullet (i, r, m) \in store(r)) \\
& \vee\ INV^2 \wedge (\exists\, a, i, m \bullet a \neq r \wedge (i, r, m) \in store(a)) \\
& \vee\ INV^2 \wedge (\exists\, a, b, i, m \bullet (i, r, m) \in link(a, b)) \\
\Rrightarrow\;& gd(receive_r^2) \vee gd(forward^2) \vee gd(relay^2).
\end{aligned}$$

Thus all the simulation conditions of Figure 6.C are satisfied and $MPS^1 \sqsubseteq MPS^2$. Our next design step will be to decompose MPS^2. However, before doing this we shall simplify the invariant of MPS^2 in order to simplify the obligations of further design steps.

7.11 Invariant Simplification

A statement S is said to *establish* invariant I if

$$true \;\Rrightarrow wp(S, I).$$

A statement S is said to *preserve* invariant I if

$$I \;\Rrightarrow wp(S, I).$$

If an invariant is established by the initialisation statement of an action system, and preserved by each action, then that invariant can be eliminated, as the following rule shows:

Rule 7.19 *Assume action system P has been defined to have invariant $I \wedge J$. Assume A is the channel set of P, and H is the label set of the internal actions of P. Let action system P' be the same as P but without the invariant I. Then $P \sqsubseteq P'$ provided*

1. P'_i *establishes* I
2. P'_α *preserves* I, $\qquad$ *each* $\alpha \in A \cup H$.

□

Soundness of this rule is proven by using simulation, with the representation function $rep(\phi) \mathrel{\widehat{=}} I \wedge \phi$.

Rule 7.19 can be used to simplify the state invariant of MPS^2. The specification of MPS^3 is the same as that of MPS^2, except that its state invariant is simply:

$$\begin{array}{|l}
\hline INV^3 \\
store : Node \rightarrow \mathbf{bag}\ Tag \times Env \\
link : net \rightarrow \mathbf{bag}\ Tag \times Env \\
\hline
\end{array}$$

It can easily be shown that the initialisation of MPS^3 establishes INV^2, and that each action of MPS^3 preserves INV^2. Hence, by Rule 7.19, with INV^2 for I and INV^3 for J, we have

$$MPS^2 \sqsubseteq MPS^3.$$

7.12 Parallel Decomposition of MPS

In this step, MPS^3 is decomposed into two parallel action-systems: *Agents* and *Media*, both of which are specified in Figure 7.G. *Agents* represents the behaviour of all the nodes of the network, and has a *send*, *receive*, *forward*, and *relay* channel for each network node. *Agents* only has the state variable *store*. *Media* represents the communications links of the network, and has a *forward* and a *relay* channel for each network node. *Media* only has the state variable *link*. *Agents* and *Media* communicate via *forward* and *relay* channels, and we have that

$$MPS^3 = (Agents \parallel Media) \setminus \{ forward_a \mid a \in Node \} \cup \{ relay_b \mid b \in Node \}.$$

$$
\begin{array}{l}
Agents \;\widehat{=} \\
\left(\begin{array}{l}
\textbf{var } store : Node \rightarrow \textbf{bag } Tag \times Env \\
\textbf{initially } \left[\, (\forall\, a \in Node \bullet store(a) = \prec\succ) \,\right] \\
\\
\textbf{for } s \in Node \textbf{ chan } send_s \textbf{ in } (r?, m?) : Env :- \\
\qquad store \;:\; \left[\begin{array}{l} (\exists\, i \in Tag \bullet (s, r?) \in route(i)^* \\ \qquad \wedge\ store(s) = store_0(s) + \prec(i, r?, m?)\succ\ ) \end{array}\right] \\
\\
\textbf{for } r \in Node \textbf{ chan } receive_r \textbf{ out } m! : Mess :- \\
\qquad store \;:\; \left[\begin{array}{l} (\exists\, i \bullet (i, r, m!) \in store_0(r) \\ \qquad \wedge\ store(r) = store_0(r) - \prec(i, r, m!)\succ\ ) \end{array}\right] \\
\\
\textbf{for } a \in Node \textbf{ chan } forward_a \textbf{ out } b! : Node;\ i! : Tag;\ (r!, m!) : Env :- \\
\qquad store \;:\; \left[\begin{array}{l} (i!, r!, m!) \in store_0(a) \wedge r! \neq a \\ \wedge\ (a, b!) \in route(i!) \wedge (b!, r!) \in route(i!)^* \\ \wedge\ store(a) = store_0(a) - \prec(i!, r!, m!)\succ \end{array}\right] \\
\\
\textbf{for } b \in Node \textbf{ chan } relay_b \textbf{ in } a? : Node;\ i? : Tag;\ (r?, m?) : Env :- \\
\qquad store \;:\; \left[\begin{array}{l} (a?, b) \in net \Rightarrow \\ \qquad store(b) = store_0(b) + \prec(i?, r?, m?)\succ \end{array}\right]
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
Media \;\widehat{=} \\
\left(\begin{array}{l}
\textbf{var } link : net \rightarrow \textbf{bag } Tag \times Env \\
\textbf{initially } \left[\, (\forall (a, b) \in net \bullet link(a, b) = \prec\succ) \,\right] \\
\\
\textbf{for } a \in Node \textbf{ chan } forward_a \textbf{ in } b? : Node;\ i? : Tag;\ (r?, m?) : Env :- \\
\qquad link \;:\; \left[\begin{array}{l} (a, b?) \in net \Rightarrow \\ \qquad link(a, b?) = link_0(a, b?) + \prec(i?, r?, m?)\succ \end{array}\right] \\
\\
\textbf{for } b \in Node \textbf{ chan } relay_b \textbf{ out } a! : Node;\ i! : Tag;\ (r!, m!) : Env :- \\
\qquad link \;:\; \left[\begin{array}{l} (a!, b) \in net \wedge (i!, r!, m!) \in link_0(a!, b) \\ \wedge\ link(a!, b) = link_0(a!, b) - \prec(i!, r!, m!)\succ \end{array}\right]
\end{array}\right)
\end{array}
$$

Figure 7.G: Network agents and network media.

This decomposition step is an example of what Hoare [Hoa85] terms *subordination*: the channel set of *Media* is a subset of the channel set of *Agents*, and the common channels to be hidden are exactly those of *Media*. *Media* is said to be subordinate to *Agents*, since its behaviour is exactly under the control of *Agents*.

To prove this decomposition step correct, we first replace the internal actions of MPS^3 with corresponding indexed sets of internal actions using Rule 7.17. For example, **internal** *relay* is replaced in MPS^3 by

$$\textbf{for } b \in Node \textbf{ internal } \ relay_b :- \\ \begin{array}{r} store, \\ link \end{array} : \left[\begin{array}{l} (\exists\, a \in Node;\ (i, r, m) \in link \mid (a, b) \in net \bullet \\ \qquad \wedge\ link(a, b) = link_0(a, b) - \prec(i, r, m)\succ \\ \qquad \wedge\ store(b) = store_0(b) + \prec(i, r, m)\succ\ ) \end{array} \right].$$

Then Rule 7.11 is applied as shown in Section 7.6.

Note 1: The first attempted specification of the $relay_b$ action of *Agents* was

$$store\ : \left[\begin{array}{l} (a?, b) \in net\ \wedge \\ store(b) = store_0(b) + \prec(i?, r?, m?)\succ \end{array} \right].$$

However, this action violates Property 6.19, since it only accepts input values $a?$ if $(a?, b) \in net$, and therefore, it cannot be placed in parallel with the $relay_b$ action of *Media*. The $relay_b$ action of *Agents* shown in Figure 7.G accepts all input values $a? \in Node$, though it's behaviour is unspecified if $(a?, b) \notin net$. It could be refined, for example, by

$$store\ : \left[\begin{array}{l} (a?, b) \in net \Rightarrow store(b) = store_0(b) + \prec(i?, r?, m?)\succ \\ (a?, b) \notin net \Rightarrow store = store_0 \end{array} \right].$$

The same is true of the $forward_a$ actions of *Media*.

Note 2: The $relay_b$ action of *Agents* doesn't check if b is on the route of the tagged envelope $(i?, r?, m?)$ that it accepts as input. So, on its own, *Agents* could accept a tagged envelope on channel $relay_b$ for which it has no routing information, i.e. INV^2 could be violated. Similarly, *Media* never checks tagged envelopes for violation of INV^2. However, since MPS^3 preserves INV^2, the combined system $Agents \parallel Media$ also preserves it, and so envelopes for which there is no routing information can never be communicated between *Agents* and *Media*.

In the next section, we introduce a rule that will allow us to decompose *Agents* into individual parallel-agents.

7.13 Another Parallel-Decomposition Rule

We introduce a parallel-decomposition rule in which the constituent action-systems of the parallel composition do not communicate with each other. This rule will be used to show that

$$P \;=\; (\| \; i \in F \bullet Q_i)$$

where P has alphabet A, each Q_i has alphabet A_i, and $\langle\, A_i \mid i \in F \,\rangle$ is a partition of A. The rule will only be valid for finite F. The rule will be used to distribute individual members of the same indexed set of channels on to separate, parallel action-systems.

Assume that P and each Q_i are of the form

$$\begin{aligned} P &= (A,\ v,\ P_\imath,\ P_A,\ P_H,\ dir,\ type) \\ Q_i &= (A_i,\ v_i,\ Q^i_\imath,\ Q^i_{A_i},\ Q^i_{H_i},\ dir_i,\ type_i). \end{aligned}$$

Assume that $\langle\, A_i \mid i \in F \,\rangle$ partitions A, $\langle\, v_i \mid i \in F \,\rangle$ partitions v, and $\langle\, H_i \mid i \in F \,\rangle$ partitions H. Assume also that $(\cup i \in F \bullet dir_i) = dir$ and $(\cup i \in F \bullet type_i) = type$. The decomposition rule is then as follows:

Rule 7.20 $P = (\| \; i \in F \bullet Q_i)$ *provided*

1. $(\| \; i \in F \bullet Q^i_\imath) = P_\imath$
2. $Q^i_a = P_a$, *each* $i \in F,\ a \in A_i$
3. $Q^i_h = P_h$, *each* $i \in F,\ h \in H_i$.

□

Soundness of the rule is proven by successive application of the binary parallel-operator, hence the requirement that F be finite.

The following rule may be used to show that Condition 1 above is satisfied:

Rule 7.21 *Let* $\langle\, v_i \mid i \in F \,\rangle$ *be a partition of* v. *Then*

$$(\| \; i \in F \bullet v_i : [\ post_i\ ]) \;=\; v : [\ (\forall\, i \in F \bullet post_i)\ ]$$

provided each $post_i$ *is independent of* v_j, *for* $j \neq i$.
□

7.14 Parallel Decomposition of Agents

In this step, *Agents* is decomposed into a set of parallel action-systems, each one representing the behaviour of an individual node of the network. We have

$$Agents \;=\; (\| \; n \in Node \bullet Agent_n).$$

$$
\begin{array}{l}
Agent_n \;\widehat{=} \\
\left(\begin{array}{l}
\mathbf{var}\ store_n : \mathbf{bag}\ Tag \times Env \\
\mathbf{initially}\ \left[\ store_n = \prec\succ\ \right] \\
\\
\mathbf{chan}\ send_n\ \mathbf{in}\ (r?, m?) : Env :- \\
\qquad store_n\ : \left[\begin{array}{l} (\exists\, i \in Tag \bullet (n, r?) \in route(i)^* \\ \qquad \wedge\ store_n = (store_n)_0 + \prec(i, r?, m?)\succ\ ) \end{array}\right] \\
\\
\mathbf{chan}\ receive_n\ \mathbf{out}\ m! : Mess :- \\
\qquad store_n\ : \left[\begin{array}{l} (\exists\, i \bullet (i, n, m!) \in (store_n)_0 \\ \qquad \wedge\ store_n = (store_n)_0 - \prec(i, n, m!)\succ\ ) \end{array}\right] \\
\\
\mathbf{chan}\ forward_n\ \mathbf{out}\ b! : Node;\ i! : Tag;\ (r!, m!) : Env :- \\
\qquad store_n\ : \left[\begin{array}{l} (i!, r!, m!) \in (store_n)_0 \wedge r! \neq n \\ \wedge\ (n, b!) \in route(i) \wedge (b!, r!) \in route(i)^* \\ \wedge\ store_n = (store_n)_0 - \prec(i!, r!, m!)\succ \end{array}\right] \\
\\
\mathbf{chan}\ relay_n\ \mathbf{in}\ a? : Node;\ i? : Tag;\ (r?, m?) : Env :- \\
\qquad store_n\ : \left[\begin{array}{l} (a?, n) \in net \Rightarrow \\ \qquad store_n = (store_n)_0 + \prec(i?, r?, m?)\succ \end{array}\right]
\end{array}\right)
\end{array}
$$

Figure 7.H: Individual agent.

$Agent_n$ is specified in Figure 7.H. Each $Agent_n$ only has the state variable $store_n$, which is simply a bag of tagged envelopes. Rule 7.20 can be used to prove the correctness of this decomposition step.

The fixed routing-relations *route* may be replaced by fixed routing-tables in a subsequent refinement-step. For $n, r \in Node$, $i \in Tag$, we define $table_n(r, i)$ as follows:

$$table_n(r, i) \;\widehat{=}\; \{\ b \in Node \mid (n, b) \in route(i) \wedge (b, r) \in route(i)^*\ \}.$$

At each node n, $table_n(r, i)$ is the set of next nodes to which messages for recipient r on $route(i)$ may be sent. The $forward_n$ action of $Agent_n$ may then be replaced by the following:

$$\textbf{chan}\ forward_n\ \textbf{out}\ b! : Node;\ i! : Tag;\ (r!, m!) : Env :-$$

$$store_n \ : \left[\begin{array}{l} (i!, r!, m!) \in (store_n)_0 \wedge r! \neq n \\ \wedge\ b! \in table_n(r!, i!) \\ \wedge\ store_n = (store_n)_0 - \prec(i!, r!, m!)\succ \end{array}\right].$$

The choice of which next node to pass a message to is nondeterministic if $table_n(r, i)$ contains more than one element. A possible further refinement would be to introduce an ordering on elements of $table_n(r, i)$ and choose elements from $table_n(r, i)$ for successive messages on a "round-robin" basis.

In Section 7.16, an adaptive routing mechanism will be introduced, where congestion information is passed around the network, and the node chosen from $table_n(i, r)$ is on the "least-congested" path to r.

7.15 Superposition of Action Systems

In this section, we introduce a technique for action-system refinement which is similar to UNITY *superposition* [CM88]. In UNITY superposition, variables and statements are added to an existing program. The additional statements may only make assignments to the additional variables, though they may read the existing variables. A statement S is added to the existing program P in one of two ways: S is simply added to the statement set of P (*union* rule), or, a statement T of P is transformed into a statement $S \parallel T$ that executes S and T simultaneously (*augmentation* rule). In UNITY, each property of the existing program is preserved by superposition.

We shall regard superposition as a non-symmetric operator between two action systems: an action system Q is superposed on to an existing action-system P to result in the action system $P \setminus\!\setminus Q$. If P is of the form

$$P \ = \ (A,\ v,\ P_\imath,\ P_A,\ P_H,\ dir,\ type)$$

then Q should be a simple action system of the form:

$$Q \ = \ (B,\ (v', w),\ Q_\imath,\ Q_B,\ Q_G)$$

where $B \subseteq A$, $v' \subseteq v$, H and G are disjoint, and no action of Q may write to variable v. $P \setminus\!\setminus Q$ is then given by the following definition:

Definition 7.22 $P \setminus\!\setminus Q \ \widehat{=} \ (A,\ (v, w),\ (Q_\imath\ ;\ P_\imath),$
$\qquad P_{A-B} \cup \{\ Q_b\ ;\ P_b \mid b \in B\ \},\ P_H \cup Q_G, dir, type).$

□

Superposing Q on to P has the effect of adding variables and statements to P. Superposing the actions Q_B is similar to using the UNITY augmentation-rule[1], while superposing the actions Q_G is similar to using the UNITY union-rule. Provided Q satisfies certain conditions, then superposition preserves all properties of the existing system P in the sense that $P \sqsubseteq (P \setminus\setminus Q)$. The conditions on Q are listed in the following rule:

Rule 7.23 *Let I be an invariant that is established by $(Q_\imath \; ; \; P_\imath)$ and preserved by each $P_\alpha, \alpha \in A \cup H$, and each $Q_\alpha, \alpha \in B \cup G$. Then for action systems P and Q as shown above, $P \sqsubseteq (P \setminus\setminus Q)$ provided the following conditions hold:*

1. *for each $\alpha \in \{\imath\} \cup B \cup G$, predicate ϕ independent of w,*

$$I \wedge \phi \;\Rrightarrow\; wp(Q_\alpha, \phi)$$

2. *for each $b \in B$,* $\qquad I \;\Rrightarrow gd(Q_b)$

3. *for some expression E, well-founded set WF,* $\qquad I \;\Rrightarrow\; E \in WF,$ *and for each $g \in G$,*

$$I \wedge E = e \;\Rrightarrow\; wp(Q_g, E < e).$$

□

Condition 1 says that provided I is satisfied, then each action of Q must leave all variables except w unchanged, and must terminate. Condition 2 says that each action in Q_B must be enabled whenever I is satisfied. This ensures that Q does not reduce the choice of actions offered by P. Condition 3 ensures that the internal actions Q_G do not introduce divergence. The soundness of Rule 7.23 can be proven using the hiding-refinement rule with the representation function $rep(\phi) \mathrel{\hat=} I \wedge \phi$, for ϕ independent of w.

Condition 1 of Rule 7.23 may be checked syntactically using the following rule:

Rule 7.24 *For predicate ϕ independent of w,* $\qquad \phi \;\Rrightarrow\; wp(w : [post], \phi).$

Remarks On Superposition

UNITY superposition does not have any equivalent of Condition 3. Because of the fairness assumptions in UNITY, where each statement is executed infinitely often, possible infinite execution of actions introduced by the UNITY union rule will not

[1]Since Q_b does not make assignments to the variables of P_b, executing Q_b followed by P_b has the same effect as executing P_b and Q_b simultaneously.

prevent execution of existing statements. However, Condition 3 is necessary in our case since the extra actions Q_G are internal and therefore outside the control of the environment, and we make no fairness assumptions about internal actions.

Our decision to treat superposition as an operator between two action systems is similar to the approach taken by Francez and Forman [FF90]. They define a superposition operator for programs of the IP (*interacting processes*) language. The semantics of IP programs and of the superposition operator are given in terms of state transitions. Goldman [Gol91] takes a similar approach to superposition in I/O-automata [LT87]. It does not seem possible to give a truly abstract definition of a superposition operator, i.e. a definition in terms of the CSP failures-divergences-infinites model. This is because, unlike the CSP parallel operator, interaction between P and Q in $P \setminus\!\setminus Q$ is not based on event synchronisation alone, but rather Q is allowed to read the state of P.

The only correspondence our superposition operator has with CSP is that it is a special case of hiding refinement. Back also treats superposition as a special case of his stuttering refinement [BS92].

7.16 Adaptive Routing

In this section, we introduce an adaptive-routing mechanism to MPS^2 of Figure 7.F. The mechanism chooses the "least-congested" path to pass messages on, and is introduced in two stages: firstly, *Detect*, a system that detects congestion at node stores and propagates that information through the network, is superposed on to MPS^2, and secondly, MPS^2 is refined so that it makes use of this congestion information.

7.16.1 Superposition Step

We say that a node is congested if the number of messages stored at that node exceeds the fixed value *upper*. When a node becomes congested, *Detect* will notify all other nodes of this fact. A node becomes decongested when the number of messages at that node drops below the fixed value *lower*, where $lower \leq upper$. Again, *Detect* will notify other nodes when some node becomes decongested.

The state variables of *Detect* are given by the following schema:

$$
\begin{array}{|l}
\hline
\textit{DT} \\
\textit{store} : \textit{Node} \rightarrow \textbf{bag}\ \textit{Tag} \times \textit{Env} \\
\textit{cong} : \mathbb{P}\ \textit{Node} \\
\textit{notices} : \textbf{bag}\ \textit{Node} \times \textit{Node} \times \textbf{bool} \\
\textit{localcong} : \textit{Node} \rightarrow \mathbb{P}\ \textit{Node} \\
\hline
\end{array}
$$

The variable *store* is included as *Detect* will need to read it. The set *cong* will contain those nodes that have been detected as being congested. All notifications are passed via the bag *notices*: $(a, b, \mathit{true}) \in \mathit{notices}$ notifies node a that node b has become congested, while $(a, b, \mathit{false}) \in \mathit{notices}$ notifies node a that node b has become decongested. Based on the notices it receives, each node will record those nodes it believes to be congested: $\mathit{localcong}(n)$ contains those nodes that node n believes to be congested.

The action system *Detect* is specified in Figure 7.I. The action *detcong* detects when a node becomes congested and sends a notice to all other nodes. Similarly, the action *detdecong* detects when a node becomes decongested. The action *note* reads notices and updates *localcong* accordingly.

We let $\mathit{DMPS} \mathrel{\widehat{=}} (\mathit{MPS}^2 \mathbin{\backslash\backslash} \mathit{Detect})$, and use Rule 7.23 to show that $\mathit{MPS}^2 \sqsubseteq \mathit{DMPS}$. By Rule 7.24 we can see that each action of *Detect* satisfies Condition 1 of Rule 7.23, and doesn't change the variables of MPS^2. Condition 2 is trivially satisfied since *Detect* only has internal actions. In order to show that Condition 3 of Rule 7.23 is satisfied, we must define a variant that is decreased by each of the internal actions of *Detect*. We use the following expression as a variant:

$$
\begin{aligned}
E \;\widehat{=}\; & (C + D, \#\mathit{notices}) \\
& \quad \mathit{where} \;\; C \;\widehat{=}\; \#\{\, n \in \mathit{Node} \mid \#\mathit{store}(n) > \mathit{upper} \wedge n \notin \mathit{cong} \,\} \\
& \qquad\qquad\;\; D \;\widehat{=}\; \#\{\, n \in \mathit{Node} \mid \#\mathit{store}(n) < \mathit{lower} \wedge n \in \mathit{cong} \,\}.
\end{aligned}
$$

The ordering we use on E is the usual lexicographic ordering on tuples. Although the *detcong* action increases the second component of E, it is easy to show that it decreases the first component of E. Similarly for the *detdecong* action. It is also easy to show that the *note* action decreases the second component of E, while leaving the first component unchanged. So each action of *Detect* does decrease E, and the conditions of Rule 7.23 are satisfied.

In *DMPS*, *localcong* need not be a true reflection of *cong*, since there is no guarantee that notices will be received by nodes before envelopes are passed between nodes. For this reason we say that $\mathit{localcong}(n)$ represents what node n believes to be true.

$$
\begin{array}{l}
Detect \ \widehat{=} \\
\left(\begin{array}{l}
\textbf{var } DT \\
\textbf{initially} \quad \begin{array}{r} cong, localcong, \\ notices \end{array} : \left[\begin{array}{l} cong = \{\} \land notices = \prec\succ \\ \land (\forall n \in Node \bullet localcong(n) = \{\}) \end{array}\right] \\
\\
\textbf{internal} \ \ detcong :- \\
\quad \begin{array}{r} cong, \\ notices \end{array} : \left[\begin{array}{l} (\exists n \in Node \mid n \notin cong_0 \bullet \\ \qquad \#store(n) > upper \\ \qquad \land cong = cong_0 \cup \{n\} \\ \qquad \land notices = \\ \qquad\qquad notices_0 + \prec (r, n, true) \mid r \in Node \setminus \{n\} \succ) \end{array}\right] \\
\\
\textbf{internal} \ \ detdecong :- \\
\quad \begin{array}{r} cong, \\ notices \end{array} : \left[\begin{array}{l} (\exists n \in Node \mid n \in cong_0 \bullet \\ \qquad \#store(n) < lower \\ \qquad \land cong = cong_0 \setminus \{n\} \\ \qquad \land notices = \\ \qquad\qquad notices_0 + \prec (r, n, false) \mid r \in Node \setminus \{n\} \succ) \end{array}\right] \\
\\
\textbf{internal} \ \ note :- \\
\quad \begin{array}{r} localcong, \\ notices \end{array} : \left[\begin{array}{l} (\exists (a, b, f) \in notices_0 \bullet \\ \qquad notices = notices_0 - \prec(a, b, f)\succ \\ \qquad \land f \Rightarrow localcong(a) = localcong_0(a) \cup \{b\} \\ \qquad \land \neg f \Rightarrow localcong(a) = localcong_0(a) \setminus \{b\}) \end{array}\right]
\end{array}\right)
\end{array}
$$

Figure 7.I: Congestion detection mechanism.

7.16.2 Refinement Step

In this section, *DMPS* is refined so that it makes use of the congestion information in *localcong* for adaptive routing. Each node a will give a congestion weighting $w_a(b)$ to each other node b, depending on whether it believes it to be congested or not, as follows:

$$
\begin{array}{rcll}
w_a(b) & \widehat{=} & 1 & \text{, if } b \in localcong(a) \\
 & \widehat{=} & 0 & \text{, if } b \notin localcong(a).
\end{array}
$$

A node will give a weighting to a path by adding the weights of all its intermediate nodes: for $p \in Node^*$,

$$w_a(p) \;\; \widehat{=} \;\; (\Sigma i \mid 1 \leq i < \#p - 1 \bullet w_a(p_i)).$$

We define *AMPS* (*MPS* with adaptive routing) to be the same as *DMPS*, except that the *forward* action is replaced by the following:

$$\textbf{internal} \;\; \mathit{forward} :- \\ \begin{array}{c} \mathit{store}, \\ \mathit{link} \end{array} : \left[\begin{array}{l} (\exists (a,b) \in net;\; (i,r,m) \in store_0(a) \bullet \\ \qquad r \neq a \wedge (a,b) \in route(i) \wedge (b,r) \in route(i)^* \\ \qquad \wedge\, BestPath \\ \qquad \wedge\, store(a) = store_0(a) - \prec(i,r,m)\succ \\ \qquad \wedge\, link(a,b) = link_0(a,b) + \prec(i,r,m)\succ\;) \end{array} \right].$$

The predicate *BestPath* ensures that the chosen b is on a path that has the least congestion-weighting, and is defined as follows:

$$BestPath \;\; \widehat{=} \;\; (\exists\, p \in bestpaths(a,r,i) \bullet snd(p) = b)$$

where

$$\begin{array}{rcl} paths(a,r,i) & \widehat{=} & \{\; p \in Node^* \mid p \;\; is\; an\; arc\; from\; a\; to\; r\; in\; route(i) \;\; \} \\ bestpaths(a,r,i) & \widehat{=} & \{\; p \in paths(a,r,i) \mid \\ & & \qquad w_a(p) = min\{\; w_a(p') \mid p' \in paths(a,r,i) \;\}\; \} \\ snd(p) & \widehat{=} & second\;\, element\;\, of\; p\;\; . \end{array}$$

To show that $DMPS \sqsubseteq AMPS$ using simulation, we need to show

1. $DMPS_{forward} \;\preceq\; AMPS_{forward}$
2. $gd(DMPS_{forward}) \;\Rrightarrow\; gd(AMPS_{forward})$.

The first obligation follows easily by Rule 7.8, while the second follows from:

$$\begin{array}{ll} & (a,r) \in route(i)^* \wedge a \neq r \\ \Rrightarrow & (\exists\, p \in paths(a,r,i)) \wedge a \neq r \\ \Rrightarrow & (\exists\, p \in bestpaths(a,r,i) \wedge \#p > 1) \\ \Rrightarrow & (\exists\, b \bullet BestPath). \end{array}$$

That is, if the *forward* action of *DMPS* could choose a next node b, then so could the *forward* action of *AMPS*.

The fact that *DMPS* is refined by *AMPS*, is independent of the chosen weighting (provided it is well-founded). So we could instead use a weighting function such as

$$\begin{array}{lcll} w'_a(b) & \mathrel{\hat{=}} & 10 & \text{, if } b \in \mathit{localcong}(a) \\ & \mathrel{\hat{=}} & 1 & \text{, if } b \notin \mathit{localcong}(a). \end{array}$$

With this weighting the routing would be based on a mixture of least-congested and shortest path.

Further refinements of AMPS are not presented here. It is possible to refine *AMPS* in such a way that there is a $\mathit{detcong}_n$ and a $\mathit{detdecong}_n$ action for each node n that only reads $\mathit{store}(n)$, and the congestion notices get passed around the network on the same communications links as envelopes. In this way, the adaptive-routing mechanism will also be a distributed system.

The techniques presented here allow us to ensure that the introduction of the adaptive-routing mechanism preserves correctness with respect to the original specification MPS^1. However, quite different techniques would be required to measure the effectiveness of the adaptive-routing mechanism in terms of increasing the performance of the message-passing system.

7.17 Distributed File-System

In this section, we build a distributed file-system using the message-passing system of Figure 7.E. We suppose that a distributed file-system consists of a set of file servers, in which files are stored, and a set of terminals through which users may access files. The location of files will be transparent to users, though known to terminals. Terminals and servers will communicate via the message-passing system.

The approach we take is to model the file system initially as a single action system, with a single file store, and an indexed set of channels through which users access files. This action system is then refined several times, and eventually decomposed into a set of terminals, a set of servers, and a message-passing system.

7.17.1 Initial Specification

All files will be of type *File* and will be identified by names of type *Fid*. For convenience, we assume that $\perp$ is an element of *File* representing the "undefined" file, and we represent the abstract file store as a total function from Fid to $File$:

$$fstore : Fid \rightarrow File.$$

An *operation* on a file updates that file and produces a reply of type Rep. Rather than defining a set of file operations, we shall simply assume that Op, the set of all file operations, satisfies

$$Op \quad \subseteq \quad File \rightarrow (File \times Rep).$$

If a user requests that an operation p be performed on a file identified by n, then the file store will be updated as follows:

$$\begin{aligned} fstore \quad = \quad & fstore_0 \oplus \{\ n \mapsto f\ \} \\ & \quad where\ (f, r) = p(fstore_0(n)), \end{aligned}$$

and the reply r will be passed back to the user. Formal specifications of several file operations can be found in [MS87].

The terminals of the file system are identified by the set Tid. We assume that Tid is finite. Corresponding to each terminal, there is a *request* channel and a *response* channel. Each request channel is bi-directional: it accepts a file identifier and an operation as input, and immediately outputs a unique request identifier of type Rid. The request identifier allows the user to associate an operation request with the reply to that operation. Once an operation has been performed on the file store, its reply is ready to be output on the response channel of the terminal through which it entered, along with its associated request identifier.

The state variables of the file system are as follows:

$$\begin{array}{|l} \hline FS^1 \\ \hline fstore\ :\ Fid \rightarrow File \\ wait\ :\ Rid \rightarrow\!\!\!\!\!\!+ \ (Tid \times Fid \times Op) \\ comp\ :\ Rid \rightarrow\!\!\!\!\!\!+ \ (Tid \times Rep) \\ unused\ :\ \mathbb{P}\ Rid \\ \hline disjoint\ \langle dom(wait), dom(comp), unused \rangle \\ \hline \end{array}$$

Operation requests that have been accepted but not yet performed are held in *wait*, while replies from completed operations that have not yet been output are held in *comp*. Both *wait* and *comp* are partial functions with disjoint domains, so that a request identifier can be associated with at most one request or reply. Request identifiers that are free to be used are held in *unused*.

$$
\mathit{FileSys}^{1} \;\widehat{=}
$$

$$
\left(
\begin{array}{l}
\textbf{var } \mathit{FS}^{1} \\[4pt]
\textbf{initially } \left[\begin{array}{l} \mathit{fstore} = (\lambda\, n : \mathit{Fid} \bullet \bot) \\ \wedge\; \mathit{wait} = \mathit{comp} = \{\} \wedge \mathit{unused} = \mathit{Rid} \end{array}\right] \\[12pt]
\textbf{for } t \in \mathit{Tid} \textbf{ chan } \mathit{req}_t \textbf{ in } n? : \mathit{Fid};\; p? : \mathit{Op} \textbf{ out } i! : \mathit{Rid} :- \\[4pt]
\qquad \begin{array}{r} \mathit{wait}, \\ \mathit{unused} \end{array} : \left[\begin{array}{l} i! \in \mathit{unused}_0 \\ \wedge\; \mathit{unused} = \mathit{unused}_0 \setminus \{i!\} \\ \wedge\; \mathit{wait} = \mathit{wait}_0 \cup \{(i!, t, n?, p?)\} \end{array}\right] \\[12pt]
\textbf{for } t \in \mathit{Tid} \textbf{ chan } \mathit{resp}_t \textbf{ out } i! : \mathit{Rid};\; r! : \mathit{Rep} :- \\[4pt]
\qquad \mathit{comp} : \left[\begin{array}{l} (i!, t, r!) \in \mathit{comp}_0 \\ \wedge\; \mathit{comp} = \{i!\} \lhd \mathit{comp}_0 \end{array}\right] \\[12pt]
\textbf{internal } \mathit{service} :- \\[4pt]
\qquad \begin{array}{r} \mathit{wait}, \\ \mathit{comp}, \\ \mathit{fstore} \end{array} : \left[\begin{array}{l} (\exists\, i : \mathit{Rid};\; t : \mathit{Tid};\; n : \mathit{Fid};\; p : \mathit{Op};\; f : \mathit{File};\; r : \mathit{Rep} \bullet \\ \qquad (i, t, n, p) \in \mathit{wait}_0 \\ \qquad \wedge\; (f, r) = p(\mathit{fstore}_0(n)) \\ \qquad \wedge\; \mathit{wait} = \{i\} \lhd \mathit{wait}_0 \\ \qquad \wedge\; \mathit{comp} = \mathit{comp}_0 \cup \{(i, t, r)\} \\ \qquad \wedge\; \mathit{fstore} = \mathit{fstore}_0 \oplus \{n \mapsto f\}) \end{array}\right]
\end{array}
\right)
$$

Figure 7.J: File system.

The initial specification of the file system, *FileSys*[1], is given in Figure 7.J. A req_t action accepts $n?$ and $p?$ as input and immediately chooses a request identifier from *unused* as output. Then $n?$ and $p?$, along with the request identifier and terminal identifier are placed in *wait*. We assume *Rid* is infinite, so that a request action can always choose an identifier from *unused*. If there is at least one reply for terminal t in *comp*, then action resp_t chooses one of those replies and outputs it along with its request identifier. File operations are performed by the internal action *service*, which takes requests from *wait*, updates the file store appropriately, and places the reply in *comp*.

Since the *service* action chooses elements from *wait* nondeterministically, the order in which file operations are performed will be nondeterministic. This is

$$
\begin{array}{l}
\mathit{ReqRespSys} \mathrel{\widehat{=}} \\
\quad \left(\begin{array}{l}
\mathbf{var}\ \mathit{requests} : \mathbf{bag}\ \mathit{Tid} \\
\mathbf{initially}\ \left[\ \mathit{requests} = \prec\succ\ \right] \\
\\
\mathbf{for}\ t \in \mathit{Tid}\ \mathbf{chan}\ \mathit{req}_t\ \mathbf{in}\ n? : \mathit{Fid};\ p? : \mathit{Op}\ \mathbf{out}\ i! : \mathit{Rid} :- \\
\qquad \mathit{requests}\ : \left[\ \mathit{requests} = \mathit{requests}_0 + \prec t \succ\ \right] \\
\\
\mathbf{for}\ t \in \mathit{Tid}\ \mathbf{chan}\ \mathit{resp}_t\ \mathbf{out}\ i! : \mathit{Rid};\ r! : \mathit{Rep} :- \\
\qquad \mathit{requests}\ : \left[\ t \in \mathit{requests}_0 \wedge \mathit{requests} = \mathit{requests}_0 - \prec t \succ\ \right]
\end{array} \right)
\end{array}
$$

Figure 7.K: Request-response system.

intended to reflect the fact that the system may perform file operations in any order. Although *service* is outside the control of the environment, we can show that it never causes divergence, and it never blocks communication at an output channel resp_t if there are outstanding requests from terminal t in *wait*. We do this by showing that $\mathit{FileSys}^1$ is a refinement of *ReqRespSys* of Figure 7.K. *ReqRespSys* has the same channels as $\mathit{FileSys}^1$. For each req_t that it accepts, *ReqRespSys* offers a resp_t, though it ignores input values, and produces nondeterministic output-values. It can be shown that $\mathit{ReqRespSys} \sqsubseteq \mathit{FileSys}^1$ using the abstraction invariant

$$
\begin{array}{rcl}
\mathit{requests} & = & \prec t \mid (\exists\, i, n, p \bullet (i, t, n, p) \in \mathit{wait}) \succ \\
& & + \prec t \mid (\exists\, i, r \bullet (i, t, r) \in \mathit{comp}) \succ,
\end{array}
$$

and variant $E = \#\mathit{dom}(\mathit{wait})$.

7.17.2 Distributed Implementation

Several successive refinements and decompositions of $\mathit{FileSys}^1$ are presented in Appendix E. The correctness of these transformations can be proven using techniques already presented. In this section, we describe the final transformation of $\mathit{FileSys}^1$, which is its distributed implementation.

Each terminal t is represented by an action system $\mathit{Terminal}_t$. To identify the servers, we introduce the (finite) set *Sid*. Each server s is represented by an action system Server_s. Each $\mathit{Terminal}_t$ has a req_t channel and a resp_t channel corresponding to those of $\mathit{FileSys}^1$. Each $\mathit{Terminal}_t$ also has a send_t and a $\mathit{receive}_t$

channel to interface with the message-passing system. Each $Server_s$ has a $send_s$ and a $receive_s$ to interface with the message-passing system. We equate the nodes of the message-passing system with terminals and servers:

$$Node \;\widehat{=}\; Tid \cup Sid.$$

Two types of message may be passed on the message-passing system. The first is a request from a terminal to a server, which consists of a request identifier, a terminal identifier, a file identifier, and an operation. The second message type is a reply from a server to a terminal, which consists of a request identifier and the reply. So we define messages as follows:

$$\begin{aligned} Mess \;\widehat{=}\; & mq(Rid \times Tid \times Fid \times Op) \\ & \mid mp(Rid \times Rep). \end{aligned}$$

Here we are using a "free type" definition from the Z notation [Spi89]. The definition declares mq to be an injective function from $Rid \times Tid \times Fid \times Op$ to $Mess$, mp to be an injective function from $Rid \times Rep$ to $Mess$, and mq and mp to have disjoint ranges.

To distribute the file store amongst the servers, we assume the existence of a fixed function $which$, that maps file identifiers to servers:

$$which : Fid \rightarrow Sid.$$

A terminal uses the function $which$ when deciding which server to pass a request to. To distribute request identifiers amongst the terminals, we assume that $init\_unused$ is a partition of Rid, such that for each $t \in Tid$, $init\_unused_t$ is an infinite subset of Rid.

The action system $Terminal_t$ is specified in Figure 7.L. The set $wait_t$ contains those requests that have been received by terminal t, but not yet passed on to the appropriate server. The set $comp_t$ contains those replies that have been returned by a server to terminal t and have yet to be passed to the user. The $send_t$ action chooses some request from $wait_t$, determines which server $s!$ to send it to, and generates the appropriate message $m!$. The action $receive_t$ accepts a message and, provided it is of the correct form, it extracts the i and r components, placing them in $comp_t$.

The action system $Server_s$ is specified in Figure 7.M. The file store of $Server_s$ is represented by the function $fstore_s$. The only part of $fstore_s$ of interest is

$$\{\ n \mapsto f \mid fstore_s(n) = f \wedge which(n) = s\ \}.$$

$Server_s$ has two control states represented by the boolean variable b_s. If b_s is $false$, then $Server_s$ is ready to receive a request, and, if b_s is true, then $Server_s$ is ready

$$
\begin{array}{l}
Terminal_t \ \widehat{=} \\
\left(\begin{array}{l}
\textbf{var} \left[\begin{array}{l} wait_t : \mathbb{P}(Rid \times Fid \times Op) \\ comp_t : \mathbb{P}(Rid \times Rep) \\ unused_t : \mathbb{P}\ Rid \end{array}\right] \\[2ex]
\textbf{initially} \ \left[\begin{array}{l} wait_t = comp_t = \{\} \\ \wedge\ unused_t = init\_unused_t \end{array}\right] \\[2ex]
\textbf{chan}\ req_t\ \textbf{in}\ n? : Fid;\ p? : Op\ \textbf{out}\ i! : Rid :- \\
\quad \begin{array}{c} wait_t, \\ unused_t \end{array} : \left[\begin{array}{l} i! \in (unused_t)_0 \\ \wedge\ unused_t = (unused_t)_0 \setminus \{i!\} \\ \wedge\ wait_t = (wait_t)_0 \cup \{(i!, n?, p?)\} \end{array}\right] \\[2ex]
\textbf{chan}\ resp_t\ \textbf{out}\ i! : Rid;\ r! : Rep :- \\
\quad comp_t \ : \left[\begin{array}{l} (i!, r!) \in (comp_t)_0 \\ \wedge\ comp_t = (comp_t)_0 \setminus \{(i!, r!)\} \end{array}\right] \\[2ex]
\textbf{chan}\ send_t\ \textbf{out}\ s! : Node;\ m! : Mess :- \\
\quad wait_t \ : \left[\begin{array}{l} (\exists\, i : Rid;\ n : Fid;\ p : Op \bullet \\ \qquad (i, n, p) \in (wait_t)_0 \\ \qquad \wedge\ s! = which(n) \\ \qquad \wedge\ m! = mq(i, t, n, p) \\ \qquad \wedge\ wait_t = (wait_t)_0 \setminus \{(i, n, p)\}) \end{array}\right] \\[2ex]
\textbf{chan}\ receive_t\ \textbf{in}\ m? : Mess :- \\
\quad comp_t \ : \left[\begin{array}{l} m? \in ran(mp) \Rightarrow \\ \qquad (\exists\, i : Rid;\ r : Rep \mid m? = mp(i, r) \bullet \\ \qquad\qquad comp_t = (comp_t)_0 \cup \{(i, r)\}) \end{array}\right]
\end{array}\right)
\end{array}
$$

Figure 7.L: Individual terminal.

to output a reply. When the $receive_s$ action accepts a request, it updates $fstore_s$, and assigns appropriate values to the state variables b_s, i_s, t_s, r_s. The values in i_s, t_s, and r_s are then used by the $send_s$ action to determine what message should be sent where.

$$
\begin{array}{l}
\mathit{Server}_s \;\widehat{=}\\
\left(\begin{array}{l}
\textbf{var}\left[\begin{array}{l} \mathit{fstore}_s : \mathit{Fid} \rightarrow \mathit{File}\\ b_s : \textbf{bool};\; i_s : \mathit{Rid};\; t_s : \mathit{Tid};\; r_s : \mathit{Rep}\end{array}\right]\\[2ex]
\textbf{initially}\;\left[\begin{array}{l} \mathit{fstore}_s = (\lambda\, n : \mathit{Fid} \bullet \perp)\\ \wedge\; b_s = \mathit{false}\end{array}\right]\\[2ex]
\textbf{chan}\; \mathit{receive}_s\; \textbf{in}\; m? : \mathit{Mess} :\!-\\
\quad\begin{array}{c} b_s,\\ i_s,\\ t_s,\\ r_s,\\ \mathit{fstore}_s\end{array} : \left[\begin{array}{l}
(b_s)_0 = \mathit{false} \wedge\\
m? \in ran(mq) \Rightarrow\\
\quad (\exists\, n : \mathit{Fid};\; p : \mathit{Op};\; f : \mathit{File} \bullet\\
\qquad m? = mq(i_s, t_s, n, p)\\
\qquad \wedge\; (f, r_s) = p((\mathit{fstore}_s)_0(n))\\
\qquad \wedge\; \mathit{fstore}_s = (\mathit{fstore}_s)_0 \oplus \{n \mapsto f\}\\
\qquad \wedge\; b_s = \mathit{true})
\end{array}\right]\\[2ex]
\textbf{chan}\; \mathit{send}_s\; \textbf{out}\; t! : \mathit{Node};\; m! : \mathit{Mess} :\!-\\
\quad b_s : \left[\begin{array}{l}
(b_s)_0 = \mathit{true}\\
\wedge\; b_s = \mathit{false}\\
\wedge\; t! = t_s\\
\wedge\; m! = mp(i_s, r_s)
\end{array}\right]
\end{array}\right)
\end{array}
$$

Figure 7.M: Individual file server.

Finally, from the transformations presented in Appendix E, we have that

$$
\begin{array}{l}
\mathit{FileSys}^1 \;\sqsubseteq\\
\quad\left(\begin{array}{l}
(\parallel t \in \mathit{Tid} \bullet \mathit{Terminal}_t)\\
\parallel (\parallel s \in \mathit{Sid} \bullet \mathit{Server}_s)\\
\parallel \mathit{MPS}^1
\end{array}\right) \setminus \left(\begin{array}{l}
\{\; \mathit{send}_n \mid n \in \mathit{Node}\;\}\\
\cup\; \{\mathit{receive}_n \mid n \in \mathit{Node}\;\}
\end{array}\right).
\end{array}
$$

Like the decomposition of *MPS* into *Agents* and *Media*, this is another example of subordination. Here, the servers and the message-passing system are subordinate to the terminals.

7.18 Remarks

Distributed systems, similar in nature to the systems presented in this chapter, have been developed using various state-based approaches.

Staskaukas [Sta88] has developed a distributed funds-transfer system in UNITY. This system maintains a set of accounts, and allows money to be transferred between them through "point-of-sale" terminals. Sere [Ser91] has developed a reactive processor-farm using Back's action system formalism. This accepts tasks from its environment and distributes them amongst slave processors via a communications network. In both of these case studies, the system is specified initially as a single system that interacts with its environment. Then the system is refined and decomposed into interacting subsystems by a series of transformations. In the case of [Sta88], transformation is achieved using UNITY logic proof rules, while in the case of [Ser91], transformation is achieved using Back's refinement and parallel composition rules for action systems [Bac90b]. One difference between both these case studies and the case studies presented in this chapter concerns the form of interaction used. In our case, interaction with the environment and between sub-systems is via shared actions, while, in the case of both [Sta88] and [Ser91], interaction is via shared variables. For example, in the processor farm of [Ser91], the environment offers a task to the processor farm by appending it to a variable representing a queue.

Kurki-Suonio & Kankaanpää [KSK88] have used Back's action-system formalism to develop a telephone exchange. In this case, interaction is via shared actions rather than shared variables. For example, the environment initiates a telephone call by engaging in a 'lift-receiver' action jointly with the telephone exchange. The state-traces model (with transitions labelled by actions) is used to reason about interaction via shared actions.

The structure of the file-system implementation presented in this chapter resembles the *open-systems interconnect* (OSI) reference model used to describe standards for telecommunications services [HS88]. In the OSI reference model, communications services are divided into seven hierarchical layers. Each layer uses the services of the lower layers to provide more enhanced services. The lowest layer of the model (*physical* layer) is concerned with physical communications links, while the highest layer (*application* layer) is concerned with end-user services such as electronic mail and file transfer. In the case of the file-system implementation, we have three layers rather then seven. The bottom layer is the action system *Media*, which resembles the OSI physical-layer. The middle layer is the message-passing system, which is implemented using *Media*. The service provided by the message-passing system resembles that provided by the OSI *network*-layer, which controls end-to-end routing. Finally, the top layer is the distributed file-system, which is implemented using the message-passing system.

The language LOTOS [vEVD89] has been used to specify several OSI layers, including the *transport* layer and the *session* layer. LOTOS consists of a process algebra based on CCS [Mil89], and an abstract-data-type algebra based on ACT-ONE [EM85]. The process algebra is used to describe ordering of communication events, while the abstract-data-type algebra is used to specify data structures and operations on them. Usually the event parts and the data parts of LOTOS specifications are refined separately [vEVD89]. This contrasts with our approach, where the event parts and data parts are not specified separately, and refinement is more uniform.

Chapter 8

Discussion

In this chapter, we draw some conclusions, make some comparisons with related work, and suggest future developments.

8.1 Conclusions

The starting point for our CSP approach to action systems was Morgan's CSP failures-divergences semantics for action systems [Mor90a]. We developed this in a number of ways and demonstrated the usefulness of our developments with a series of case studies.

In Chapter 3, we extended the action-system semantics to the CSP infinite-traces model in order to deal properly with unbounded nondeterminism. Unbounded nondeterminism can arise quite naturally in specification; for example, the req_t actions of *FileSys* (Figure 7.J) are unboundedly nondeterministic in their choice of identifier $i!$. The predicate $\overline{inf}(\Sigma)$ was introduced to define the infinite traces of action systems. Using properties of wp and $\overline{inf}$, the infinite traces semantics of action systems were shown to satisfy the well-formedness conditions of the infinite traces model given by Roscoe [Ros88]. It was shown that the simulation proof technique for refinement of action systems is sound in the infinite traces model, though completeness is lost for unboundedly-nondeterministic action-systems.

Chapter 4 introduced internal actions in order to define a hiding operator for action systems. The nondeterministic-iterate construct was used in the definition of the CSP infinite traces semantics of action systems with internal actions, and it was shown that the hiding operator for action systems corresponds exactly to the CSP hiding-operator in the infinite-traces model. Therefore, any properties enjoyed by the CSP operator, such as monotonicity, are also enjoyed by the action-system operator. It was shown, using a simple example, that the

CSP failures-divergences model is inappropriate for unboundedly-nondeterministic action-systems, since the failures-divergences correspondence may be lost after application of the action-system hiding-operator.

Also in Chapter 4, simulation was extended to deal with internal actions. A special case of simulation, called hiding refinement, was developed, which may be used to introduce internal actions representing hidden communication between sub-systems. In hiding refinement, only the concrete action system has internal actions. These actions are "stuttering steps", since they cause no change to the corresponding abstract state and no infinite sequence of internal actions is possible. A well-foundedness argument may be used to check that no infinite sequence of internal actions is possible. It was shown how hiding refinement may also be used when the abstract action-system has internal actions, and the concrete action-system has an internal action corresponding to each abstract internal-action.

Chapter 5 introduced an operator for composing two action systems in parallel, in which interaction is based on synchronisation over commonly-labelled actions. The operator was shown to correspond to the CSP parallel-operator in the infinite traces model, so that any properties enjoyed by the CSP parallel-operator are enjoyed by the action-system parallel-operator. The operator may be used to show that the parallel composition of two or more action systems is a valid implementation of some single action-system (parallel-decomposition step).

In Chapter 6, value-passing action-systems were defined, in which each action is parameterised by an input variable and/or an output variable. Value-passing action-systems were given a CSP semantics such that each action corresponds to a CSP channel, and the parameter variables of an action represent the values that may be communicated on that channel. The CSP semantics was defined so that an output value for a channel is always chosen internally by an action system. This corresponds to our operational understanding of output communication and means that a refinement of an action system may be more deterministic in its choice of output values. The simulation technique was extended to value-passing action-systems, though the restrictions on representation functions meant that simulation is incomplete even for boundedly-nondeterministic action-systems. The hiding operator and parallel operator were also extended to value-passing action-systems and shown to correspond to the respective CSP operators. As well as involving synchronisation over commonly-labelled channels, the parallel operator allows passing of values between action systems, just as in CSP value-communication. However, parallel composition of bi-directional channels was left for further study.

Chapter 7 demonstrated the use of the action-system techniques through a series of case studies. Example action systems were described using specification statements. Several design rules for refinement and parallel composition of action systems were developed and used to refine and decompose example action systems into parallel interacting sub-systems. These design rules were based on our definitions of hiding refinement and parallel composition for action systems and were intended to be general enough for use elsewhere. Some of the design rules were also based on rules of the refinement calculus, which was possible since our definition of action refinement is the same as that used in the refinement calculus [Bac80, Bac90a, Mor90b, MRG88, Mrr87]. The case studies demonstrate that our approach is suited to the design of distributed systems, including telecommunications systems, in which interaction is based on value-passing communication.

In several places, restrictions were imposed on action systems in order to maintain the correspondence with CSP. In Chapter 2, initialisation statements were required to be non-miraculous. This ensures that the traces of any action system contain the empty trace at least. In Chapter 5, parallel action-systems were not allowed to share variables. This ensures that interaction between the systems is based on shared actions alone. In Chapter 6, output actions were required to be terminating. This was because the value to be output by an action can only be determined when that action has terminated. Also in Chapter 6, typing restrictions were imposed on input and output actions. These ensure that the case where an output action tries to offer a value not acceptable by the corresponding input action never arises.

8.2 Related Work

The structure of our action systems is based on that introduced by Back & Kurki-Suonio [BKS83]. This structure allows the actions of a single action-system to interact via shared variables. For example, the *send* and *receive* actions of MPS^1 (Figure 7.E) interact through the shared variable *mail*. Such interaction is useful when specifying complex systems. However, in our CSP approach, although the actions of a single action system may interact via shared variables, parallel action systems may only interact via shared actions. In order to decompose an action system into parallel sub-systems, we must introduce internal actions representing interaction between the sub-systems, and refine the state variables so that they can be partitioned amongst the sub-systems. In contrast, Back's formalism allows parallel action systems to interact via shared variables. This provides more flexibility, since action systems may be decomposed by an arbitrary partitioning of actions.

Back gives a state-trace semantics to action systems, whereas we give a CSP semantics. The basic-refinement rule used by Back (see Page 12) is similar to our simulation rule, and, Back's stuttering refinement is similar to our hiding refinement. Back's refinement rules ensure state-trace refinement in the manner of Abadi & Lamport [AL88], whereas our simulation rules ensure CSP refinement. Back's parallel and (variable) hiding operators are defined on the structure of action systems, and properties of the operators are derived from these definitions. For example, the monotonicity of Back's parallel-operator w.r.t. action-system refinement is proven from the way in which the parallel composition of two action systems is constructed. In our case, although the parallel and hiding operators are defined on the structure of action systems, their properties are simply inherited from the corresponding CSP operators rather than being proven separately.

The state-traces model for action systems makes no distinction between internal and external choice of action. This is not important when interaction between systems is based on shared variables. However, the distinction is important when interaction is based on shared actions. The case study by Kurki-Suonio & Kankaanpää [KSK88], mentioned in Chapter 7, uses Back's action system formalism to describe a telephone exchange in which interaction between users and the system is based on shared actions. After lifting the receiver, a user may engage in a dial-action or a hangup-action. In the state-traces model, this could be implemented by a system that internally chooses between offering a dial-action or a hangup-action, thus possibly preventing the user from dialling. One way of disallowing this is to place fairness constraints on the system — the system should be weakly fair w.r.t. the dial-action and w.r.t. the hangup-action. Then, an implementation satisfying these fairness constraints couldn't prevent a user from eventually dialling. This is the approach taken with I/O-automata by Lynch & Tuttle [LT87] and by Jonsson [Jss90]. In our CSP approach to action systems, the distinction between internal and external choice is modelled, so it is not necessary to use fairness constraints in this case. With CSP refinement, an implementation of the telephone exchange cannot choose internally between offering a dial-action or a hangup-action, but must allow the user to choose. Of course, we cannot specify more general fairness requirements on action systems in our CSP approach.

Hofstee et al [HMvdS90, Hof92] have developed a distributed sorting-algorithm using an approach similar to ours. They start with a sequential program whose structure resembles that of Back's sequential action-systems, i.e. an initialisation followed by a DO-loop. They then decompose that program onto an array of processes with interaction based on synchronised value-passing. The value-passing is introduced by replacing an assignment statement $x := E$ with a pair of communications primitives $out!E \parallel in?x$. These communications primitives were introduced to the guarded-command language by Martin [Mar81] and they provide synchronisation based on a notion of process suspension, along with distributed assignment. In the approach of Hofstee et al, the communications primitives are embedded within the actions of a program and communication primitives for the same channel may appear in more than one place in a program. This can make reasoning about parallel compositions more difficult. In contrast, with our approach, all communications on a channel are represented by a single action, and when composing systems, we only have to reason about pairs of corresponding actions. Also, they lack a formal basis for communication and concurrency in their approach. It should be possible to develop their distributed algorithms using our approach.

The distributed implementation of the sorting algorithm described by Hofstee et al in [HMvdS90] is based on a linear array of processes. Since each process has to communicate with at most two neighbours, and since these communications are easily interleaved, there is no need for processes to offer external choice between channels. In [Hof92], the implementation is a more general graph-like array of processes, and a process may have to communicate with any number of neighbours, so that external choice is required. This is achieved by using *probes* [Mar85]. A probe is boolean flag associated with a channel that can be read by a process to check the readiness of its neighbour to communicate on that channel. A probe can be read without having to communicate on a channel. In our approach, external choice between channels is achieved simply by having actions with overlapping guards. It is worth noting that probes allow a process to test if its neighbour is not ready to communicate. Martin [Mar85] shows how this test can be used to describe a system offering fair choice between channels. Such a test cannot be described in our approach, nor in CSP.

Our CSP approach to action systems is similar to the CSP approach to transition systems of He Jifeng [He89] and Josephs [Jos88]. They use relations to define the CSP semantics of transition systems. We, on the other hand, use weakest-precondition formulae which provides a more uniform treatment of divergence. This can be seen, for example, in our hiding operator which is simpler than He Jifeng's hiding operator since he has to deal separately with the introduction of divergence. Both He Jifeng and Josephs only use the CSP failures-divergences model, whereas we use the CSP infinite-traces model and thus we treat unbounded nondeterminism properly. Also, our value-passing action systems allow for straightforward specification of value communication. From the practical point of view, an advantage of our approach over that of He Jifeng and Josephs is our closeness to the refinement calculus which allows us to apply its specification statements and refinement rules to the development of action systems.

The ability to use specification statements to describe actions also means that our approach is related to the Z [Spi89] and VDM [Jon86] methods. *Operations* in Z and VDM are described using predicates on the initial and final values of state variables, just like specification statements. A system in Z or VDM is specified by a set of state variables and a set of operations on those variables, which is the same as the structure of an action system. (The examples of Chapter 7 could easily be rewritten using either the Z or VDM notations.) Houston & Josephs [HJ92] have described a technique for parallel composition of systems specified in Z which turns out to be similar to our way of composing action systems. Their approach is intended to be based on CSP parallel composition [Jos91], though their link between CSP and Z specifications is informal. By using specification statements in action systems, we provide a formal link between Z and CSP, and as well as providing a sound parallel-operator, we provide a sound notion of internal action.

8.3 Future Work

As the case studies of Chapter 7 demonstrate, the structure of action systems provides much flexibility in refinement and parallel decomposition. However, the action-system approach does not cover the entire life-cycle of system development.

At early stages of system development it may be convenient to specify action systems more abstractly using, for example, temporal-logic formulae as in Pnueli [Pnu86] and UNITY [CM88], or using trace-refusal specifications as in CSP [Hoa85]. A set of proof rules would then be required to show that an action system satisfies its specification.

Later stages of system development involve implementation in programming languages. Action systems cannot be written directly in any programming language, since they contain no explicit flow of control, but rather any action may be executed if it is enabled. However, it should be possible to implement certain action systems as sequential programs in languages that provide synchronised communication such as occam [JG88] and Ada [WWF87]. For example, Figure 8.A contains an action system that accepts a value on channel *left* and then outputs that value on channel *right*. Figure 8.A also contains an occam program that implements the action system $COPY$. In the action system, the flow of control is determined by the boolean variable and the action guards, whereas in the occam program it is determined by the WHILE-loop and the order in which the communication events are written. In this case, our justification for saying that the occam program implements the action system is intuitive. It would be useful to have a

$$COPY \mathrel{\widehat{=}} \left(\begin{array}{ll} \mathbf{var}\ f : \mathbf{bool};\ n : T & \\ \mathbf{initially}\ \ f := \mathit{false} & \\ \mathbf{chan}\ \mathit{left}\ \mathbf{in}\ x? : T :- & \neg f \rightarrow f, n := \mathit{true}, x? \\ \mathbf{chan}\ \mathit{right}\ \mathbf{out}\ y! : T :- & f \rightarrow f, y! := \mathit{false}, n \end{array} \right)$$

$$\begin{array}{l} PROC\ copy(CHAN\ left, right : T)\ = \\ \quad WHILE\ true \\ \quad\quad VAR\ n : T \\ \quad\quad\quad SEQ \\ \quad\quad\quad\quad left?n \\ \quad\quad\quad\quad right!n \end{array}$$

Figure 8.A: Action system and its occam implementation.

set of compositional proof-rules for checking that an occam program implements an action system.

The designers of the specification language SL_0 [Old91] have attempted to achieve aims similar to those mentioned in the previous paragraph. In SL_0, a combination of CSP algebraic-notation and the transition-system approach of [He89, Jos88] is used to specify a communicating system. A set of compositional proof-rules are provided for transforming SL_0 specifications into occam programs. Back & Sere [BS90] have also investigated the transformation of action systems into occam programs.

Reed & Roscoe [RR86] have developed a timed model for CSP, and Seidel [Sei92] has developed a probabilistic model for CSP. It may be possible to treat timing and probability in action systems using these models. The timed model may allow us to specify that an action must terminate within a certain time. The probabilistic model may allow us to attach probability measures to internal and external choices. Timing would be useful in the design of safety-critical systems, for example, while probability would be useful in the design of fault-tolerant communications networks.

As Roscoe points out [Ros88], it should be possible to specify fairness requirements in CSP by using the infinite-traces model. If notions of fairness were developed for the infinte-traces model, then it should be possible to apply them to our approach to action systems.

Appendix A

Proofs for Chapter 3

Proof of Theorem 3.8

Theorem 3.8 $\overline{inf}(S\ ;\ \Sigma)\ \equiv\ \overline{wp}(S, \overline{inf}(\Sigma))$.

Proof by mutual entailment:

(a). Let $\Phi\ \mathrel{\widehat=}\ (\overline{\mu}\ X \bullet \overline{wp}_{\infty}(S\ ;\ \Sigma, X))$. Now $\Phi[1..]$ is a solution of

$$X \ \Rrightarrow\ \overline{wp}_{\infty}(\Sigma, X).$$

Hence, by Knaster-Tarski, $\Phi[1..]\ \Rrightarrow\ (\overline{\mu}\ X \bullet \overline{wp}_{\infty}(\Sigma, X))$, and, by Definition 3.6,

$$\Phi_1 \ \Rrightarrow\ \overline{inf}(\Sigma). \qquad \text{(i)}$$

Finally,

$$\begin{array}{lll} \overline{inf}(S\ ;\ \Sigma) & \Rrightarrow\ \overline{wp}(S, \Phi_1) & \textit{defn of } \Phi \\ & \Rrightarrow\ \overline{wp}(S, \overline{inf}(\Sigma)) & \textit{monotonicity,}\ \text{(i)}. \end{array}$$

(b). Let $\Psi\ \mathrel{\widehat=}\ (\overline{\mu}\ X \bullet \overline{wp}_{\infty}(\Sigma, X))$, $\quad \Psi'\ \mathrel{\widehat=}\ \langle \overline{wp}(S, \overline{inf}(\Sigma))\rangle\Psi$, $\quad \Sigma'\ \mathrel{\widehat=}\ S\ ;\ \Sigma$. Now Ψ' is a solution of

$$X \ \Rrightarrow\ \overline{wp}_{\infty}(\Sigma', X).$$

Hence, by Knaster-Tarski, $\Psi'\ \Rrightarrow\ (\overline{\mu}\ X \bullet \overline{wp}_{\infty}(\Sigma', X))$, and, by Definition 3.6, $\Psi'_0\ \Rrightarrow\ \overline{inf}(\Sigma')$, that is,

$$\overline{wp}(S, \overline{inf}(\Sigma)) \ \Rrightarrow\ \overline{inf}(S\ ;\ \Sigma).$$

Proof of Theorem 3.10

Theorem 3.10 $\overline{inf}(S^{\infty})\ \equiv\ (\overline{\mu}\ X \bullet \overline{wp}(S, X))$.

Proof by mutual entailment:

(a). By Theorem 3.8, $\overline{inf}(S^{\infty})$ is a solution of

$$X \;\Rrightarrow\; \overline{wp}(S, X).$$

Hence, by Knaster-Tarski, $\overline{inf}(S^{\infty}) \;\Rrightarrow\; (\overline{\mu}\ X \bullet \overline{wp}(S, X))$.

(b). Let $\Phi \;\hat{=}\; \langle\, (\overline{\mu}\ X \bullet \overline{wp}(S, X)) \mid i \geq 0 \,\rangle$. Now Φ is a solution of

$$X \;\Rrightarrow\; \overline{wp}_{\infty}(S^{\infty}, X).$$

Hence, by Knaster-Tarski, $\Phi \;\Rrightarrow\; (\overline{\mu}\ X \bullet \overline{wp}_{\infty}(S^{\infty}, X))$, and, by Definition 3.6, $(\overline{\mu}\ X \bullet \overline{wp}(S, X)) \;\Rrightarrow\; \overline{inf}(S^{\infty})$.

Proof of Lemma 3.14

Lemma 3.14 *For $u \in A^{\omega}$,* $\qquad \overline{inf}(Q_u) \;\Rrightarrow\; \overline{rep}(\overline{inf}(P_u))$.

Proof: Let $\overline{F} \;\hat{=}\; \overline{wp}_{\infty}(Q_u, \_)$, $\overline{G} \;\hat{=}\; \overline{wp}_{\infty}(P_u, \_)$.
For infinite predicate $\Phi \;\equiv\; \langle\, \Phi_i \mid i \in \mathbb{N} \,\rangle$, let $\overline{rep}(\Phi) \;\hat{=}\; \langle\, \overline{rep}(\Phi_i) \mid i \in \mathbb{N} \,\rangle$, (unlike Definition 3.4). From Condition 2 of Definition 2.17 we can show

$$\overline{F}(\overline{rep}(\Phi)) \;\Rrightarrow\; \overline{rep}(\overline{G}(\Phi)) \qquad \text{(ii)}$$

Then by ordinal induction we show $\overline{F}^{\alpha} \;\Rrightarrow\; \overline{rep}(\overline{G}^{\alpha})$ as follows: assume holds for $\beta < \alpha$, then

$$\begin{array}{cll}
\overline{F}^{\alpha} & \equiv & (\wedge\, \beta < \alpha \bullet \overline{F}(\overline{F}^{\beta})) \\
 & \Rrightarrow & (\wedge\, \beta < \alpha \bullet \overline{F}(\overline{rep}(\overline{G}^{\beta}))) \quad \textit{induction hypothesis, monotonicity} \\
 & \Rrightarrow & (\wedge\, \beta < \alpha \bullet \overline{rep}(\overline{G}(\overline{G}^{\beta}))) \quad \textit{by}\ \text{(ii)} \\
 & \Rrightarrow & \overline{rep}(\wedge\, \beta < \alpha \bullet \overline{G}(\overline{G}^{\beta})) \quad \textit{since}\ \overline{rep}\ \textit{is}\ \wedge\textit{-continuous} \\
 & \equiv & \overline{rep}(\overline{G}^{\alpha})
\end{array}$$

Thus, by Theorem 3.2

$$(\overline{\mu}\ X \bullet \overline{F}(X)) \;\Rrightarrow\; \overline{rep}(\overline{\mu}\ Y \bullet \overline{G}(Y)),$$

and by Definition 3.6, $\overline{inf}(Q_u) \;\Rrightarrow\; \overline{rep}(\overline{inf}(P_u))$.

Proof of Lemma 3.16

Lemma 3.16 $(\wedge\, s < u \bullet \overline{wp}(P_{\langle\imath\rangle s}, \mathit{true})) \;\Rrightarrow\; \overline{inf}(P_{\langle\imath\rangle u})$.

Proof: Let $\Phi \;\widehat{=}\; \langle\, (\wedge\; s < u[i..] \bullet \overline{wp}(P_s, true)) \mid i \geq 0 \,\rangle$. Note that $(\wedge\; s < u[i..] \bullet \overline{wp}(P_s, true))$ is a chain-conjunction. Then

$$
\begin{array}{cl}
 & \overline{wp}_\infty(P_u, \Phi) \\
\equiv & \langle\, \overline{wp}(P_{u_i}, (\wedge\; s < u[i+1..] \bullet \overline{wp}(P_s, true))) \mid i \geq 0 \,\rangle \qquad \textit{Definition 3.4} \\
\equiv & \langle\, (\wedge\; s < u[i+1..] \bullet \overline{wp}(P_{\langle u_i \rangle s}, true)) \mid i \geq 0 \,\rangle \\
 & \qquad\qquad\qquad\qquad \textit{each } P_a \textit{ is boundedly nondeterministic} \\
\equiv & \langle\, (\wedge\; s < u[i..] \bullet \overline{wp}(P_s, true)) \mid i \geq 0 \,\rangle \\
\equiv & \Phi
\end{array}
$$

Hence, by Knaster-Tarski, $\Phi \;\Rrightarrow\; (\overline{\mu}\; X \bullet \overline{wp}_\infty(P_u, X))$, and by Definition 3.6, $\Phi_0 \;\Rrightarrow\; \overline{inf}(P_u)$, that is,

$$(\wedge\; s < u \bullet \overline{wp}(P_s, true)) \;\;\Rrightarrow\;\; \overline{inf}(P_u).$$

Lemma now follows since $P_\imath$ is also boundedly nondeterministic.

Satisfaction of Well-formedness Condition C9

A process $\mathcal{P} = (A, F, D, I)$ is pre-deterministic if it satisfies:

$$
\begin{array}{rcll}
(s, X) \in F & \Rightarrow & s \in D \;\vee\; (\forall\, a \in X \bullet (s\langle a \rangle, \{\}) \notin F), \text{ each } s \in A^*,\ X \subseteq A & \text{(iii)} \\
I & = & \{u \in A^\omega \mid (\forall\, s < u \bullet (s, \{\}) \in F)\}. & \text{(iv)}
\end{array}
$$

Note: for infinite trace u, $s < u$ means that s is a finite prefix of u. The set of pre-deterministic implementations of a process, $imp(\mathcal{P})$, is defined as:

$$imp(\mathcal{P}) \;\;\widehat{=}\;\; \{\; \mathcal{Q} \mid \mathcal{P} \sqsubseteq \mathcal{Q} \wedge \mathcal{Q} \textit{ is pre-deterministic } \}.$$

Well-formedness condition **C9** is written as:

C9 $\qquad (A, F, D, I) = (\sqcap\, imp(A, F, D, I)).$

Roscoe [Ros88] shows that **C9** can be replaced by

$$F \;\;=\;\; \bigcup\{\; F' \mid (A, F', D', I') \in imp(A, F, D, I)\; \}. \qquad \text{(v)}$$

Our task then is to show: (a) that any pre-deterministic action system satisfies (iii) and (iv), and, (b) that for any action system P, $\{\!\!\{P\}\!\!\}$ satisfies (v).

(a). A pre-deterministic action system is one in which the initialisation and each labelled action is pre-deterministic. A statement S is pre-deterministic if

$$\overline{wp}(S, \phi) \;\;\Rrightarrow\;\; \neg\; halt(S) \vee wp(S, \phi).$$

From this we get, for pre-deterministic action system $P = (A, V, P_\imath, P_A)$, $s \in A^*$,

$$\overline{wp}(P_{\langle \imath \rangle s}, \phi) \;\;\Rrightarrow\;\; \neg\; halt(P_{\langle \imath \rangle s}) \vee wp(P_{\langle \imath \rangle s}, \phi).$$

Thus, for $\{\} \subset X \subseteq A$,

$$
\begin{array}{lll}
 & (s,X) \in \mathcal{F}[\![P]\!] & \\
\Rrightarrow & \overline{wp}(P_{\langle\imath\rangle s}, \neg\, gd(P_X)) & \textit{Definition 2.15} \\
\Rrightarrow & \neg\, halt(P_{\langle\imath\rangle s}) \vee wp(P_{\langle\imath\rangle s}, \neg\, gd(P_X)) & P \textit{ pre-deterministic} \\
\Rrightarrow & \neg\, halt(P_{\langle\imath\rangle s}) \vee (\forall\, a \in X \bullet \neg\, \overline{wp}(P_{\langle\imath\rangle s}, gd(P_a))) & \textit{positive conjunctivity} \\
\Rrightarrow & s \in \mathcal{D}[\![P]\!] \vee (\forall\, a \in X \bullet (s\langle a\rangle, \{\}) \notin \mathcal{F}[\![P]\!]). & \textit{Definition 2.15}
\end{array}
$$

That is, $[\![P]\!]$ satisfies (iii) (note: if $X = \{\}$, then (iii) is trivially satisfied).

If statement S is pre-deterministic, then it is easy to show that $\overline{wp}(S, \_)$ is $\wedge$-continuous. If the initialisation and each labelled action of an action system P is continuous, then it is easy to show, for $u \in A^*$, (cf. Lemma 3.16)

$$\overline{inf}(P_{\langle\imath\rangle u}) \;\equiv\; (\wedge\, t < u \bullet gd(P_{\langle\imath\rangle t})).$$

Hence, for pre-deterministic P, $[\![P]\!]$ satisfies (iv).

(b). Equation (v) is equivalent to

$$(s,X) \in F \;\Rightarrow\; (\exists\, F', D', I' \bullet (s,X) \in F' \wedge (A, F', D', I') \in imp(A, F, D, I)).$$

To show that any action system $P = (A, V, P_\imath, P_A)$ satisfies this, we construct, for each $(s,X) \in \mathcal{F}[\![P]\!]$, a pre-deterministic implementation of P that also has (s,X) as a failure. Firstly we have the following lemma about a trace $s \in A^*$ and an action system Q which is constructed from P:

Lemma A.1 *For trace $s \in \mathcal{T}[\![P]\!]$, let action system Q be such that*

$$
\begin{array}{rcll}
Q_\imath & = & M_{\langle\rangle}\ ;\ t := \langle\rangle & \\
Q_a & = & t\langle a\rangle \le s \rightarrow M_{t\langle a\rangle}\ ;\ t := t\langle a\rangle & \\
 & & [\!]\ t\langle a\rangle \not\le s \rightarrow N_a & \textit{each } a \in A
\end{array}
$$

where M_t is defined for $t \le s$, N_a is defined for $a \in A$, and M and N satisfy:

1. $P_\imath \preceq M_{\langle\rangle}$
2. $gd(M_{\langle\rangle}) \equiv true$
3. $P_a \preceq M_{t\langle a\rangle}$, *each* $t\langle a\rangle \le s$
4. $gd(P_a) \Rrightarrow gd(M_{t\langle a\rangle})$, *each* $t\langle a\rangle \le s$
5. M_t *is pre-deterministic,* *each* $t \le s$
6. $P_a \preceq N_a$, *each* $a \in A$
7. $gd(P_a) \Rrightarrow gd(N_a)$, *each* $a \in A$
8. N_a *is pre-deterministic,* *each* $a \in A$.

Then $[\![Q]\!] \in imp(P)$.

Proof: Since each action of Q is pre-deterministic, $\{\!|Q|\!\}$ is pre-deterministic. To show that $P \sqsubseteq Q$, use simulation with the ($\vee$-continuous) representation function $rep(\phi) \mathrel{\widehat{=}} \phi$, where ϕ is independent of state variable t.
□

This lemma provides a way of constructing a pre-deterministic implementation of P: simply construct M and N satisfying the conditions listed in the lemma. We will see shortly that this is always possible. Furthermore, for any $(s, X) \in \mathcal{F}\{\!|P|\!\}$, we can construct M in such a way as to ensure that $(s, X) \in \mathcal{F}\{\!|Q|\!\}$. This is achieved as follows: for a given $(s, X) \in \mathcal{F}\{\!|P|\!\}$, we define a set of predicates ϕ_t, for each $t \leq s$, where

$$\begin{array}{lcll} \phi_s & \mathrel{\widehat{=}} & \neg\, gd(P_X) & \\ \phi_t & \mathrel{\widehat{=}} & \overline{wp}(P_a, \phi_{t\langle a\rangle}), & \quad \textit{each } t\langle a\rangle \leq s, \end{array}$$

so that $\phi_{\langle\rangle} \equiv \overline{wp}(P_s, \neg\, gd(P_X))$, and, since $(s, X) \in \mathcal{F}\{\!|P|\!\}$, we have that

$$\overline{wp}(P_\imath, \phi_{\langle\rangle}) \quad \equiv \quad \textit{true}.$$

Then, construct M such that

$$\begin{array}{rcll} \textit{true} & \Rrightarrow & \overline{wp}(M_{\langle\rangle}, \phi_{\langle\rangle}) & \qquad \text{(vi)} \\ \phi_t & \Rrightarrow & \overline{wp}(M_{t\langle a\rangle}, \phi_{t\langle a\rangle}), & \quad \textit{for } t \neq \langle\rangle,\ t\langle a\rangle \leq s. \qquad \text{(vii)} \end{array}$$

If M satisfies (vi) and (vii), then it is easy to show that Q satisfies:

$$\begin{array}{rcl} \textit{true} & \Rrightarrow & \overline{wp}(Q_{\langle\imath\rangle s}, \neg\, gd(Q_X)) \\ & \Rrightarrow & (s, X) \in \mathcal{F}\{\!|Q|\!\}. \end{array}$$

Now well-formedness condition **C9** is satisfied, provided, for each $(s, X) \in \mathcal{F}\{\!|P|\!\}$, Q can be constructed such that M satisfies (vi) and (vii), and M and N satisfy the conditions of Lemma A.1. Lemma A.2 (see below) ensures that M and N can always be constructed as required. For example, given

$$\phi_t \quad \Rrightarrow \quad \overline{wp}(P_a, \phi_{t\langle a\rangle})$$

Lemma A.2 says we can construct $M_{t\langle a\rangle}$ such that

$$\phi_t \quad \Rrightarrow \quad \overline{wp}(M_{t\langle a\rangle}, \phi_{t\langle a\rangle})$$

and $M_{t\langle a\rangle}$ satisfies 3, 4, 5 of Lemma A.1.

Lemma A.2 *Given a statement S, predicates ϕ and ψ, such that $\phi \Rrightarrow \overline{wp}(S, \psi)$ there is an S' satisfying:*

1. $S \preceq S'$
2. $gd(S) \Rrightarrow gd(S')$
3. S' *is pre-deterministic*
4. $\phi \Rrightarrow \overline{wp}(S', \psi)$.

Proof: For any statement S, we construct its equivalent specification statement SP, then construct SP' satisfying 1..4. Assume S is a v-statement, then SP is defined as:

$$SP \ \widehat{=} \ v : [\ halt(S),\ \overline{wp}(S, v = v')[v \backslash v_0][v' \backslash v]\ ]$$

where v' is some fresh variable independent of v. Using the definition of variable independence (Definition 2.9) and the definition of specification statements (Definition 2.3), it can be shown that $SP = P$. (Note: construction of SP is similar to the correspondence between predicate transformers and predicates described by Nelson [Nel89, Theorem 1].) Now SP is of the form $SP = v : [pre, post]$, and predicates ϕ and ψ are such that $\phi \Rrightarrow \overline{wp}(SP, \psi)$. Construct relations R, R', as follows:

$$R \ \widehat{=} \ \{v_0, v \mid post \wedge \psi\} \qquad\qquad R' \ \widehat{=} \ \{v_0, v \mid post\}.$$

Let f and f' be functions such that

$$\begin{array}{rcl} f & \subseteq & R \\ dom(f) & = & dom(R) \end{array} \qquad \begin{array}{rcl} f' & \subseteq & R' \\ dom(f') & = & dom(R') \end{array}$$

Note that f and f' exist by the Axiom of Choice [End77]:

$$(\forall\, relation\ R \bullet (\exists\, function\ f \bullet f \subseteq R \wedge dom(f) = dom(R))).$$

Finally, construct statement SP':

$$SP' \ \widehat{=} \ v : \left[\ pre,\ \begin{array}{rcl} \phi[v \backslash v_o] & \Rightarrow & v = f(v_0) \\ \neg\, \phi[v \backslash v_o] & \Rightarrow & v = f'(v_0) \end{array}\ \right]$$

SP' is pre-deterministic since the new value for v is chosen by a function. It is also easy to show that

$$\begin{array}{rcl} SP & \preceq & SP' \\ gd(SP) & \Rrightarrow & gd(SP') \\ \phi & \Rrightarrow & \overline{wp}(SP', \psi). \end{array}$$

Appendix B

Proofs for Chapter 4

Proof of Theorem 4.2

Let $g(Y) \;\widehat{=}\; \phi \wedge wp(S, Y)$. From Definition 4.1 we have

$$wp(\mathbf{it}\ S\ \mathbf{ti}, \phi) \;\equiv\; (\mu\, X \bullet g(X)). \tag{i}$$

Let $f(Y) \;\widehat{=}\; wp(S, Y)$. From Theorem 3.10, we have

$$\overline{inf}(S^{\infty}) \;\equiv\; (\overline{\mu}\ X \bullet \overline{f}(X)).$$

Using the Knaster-Tarski Theorem, we can prove the following lemma:

Lemma B.1 *For monotonic f,* $\quad (\mu\, X \bullet f(X)) \equiv \neg\, (\overline{\mu}\ X \bullet \overline{f}(X)).$

So we have

$$\neg\, \overline{inf}(S^{\infty}) \;\equiv\; (\mu\, X \bullet f(X)). \tag{ii}$$

Let $C \equiv (\wedge\ i \in \mathbb{N} \bullet wp(S^i, \phi))$, and let g^{α} and f^{α} be as defined in Theorem 3.1. Then we have lemmas as follows:

Lemma B.2 *For any ordinal α,* $\quad g(C \wedge f^{\alpha}) \;\equiv\; C \wedge f^{\alpha+1}.$

Proof:

$$\begin{array}{cll}
 & g(C \wedge f^{\alpha}) & \\
\equiv & \phi \wedge wp(S, (\wedge\ i \in \mathbb{N} \bullet wp(S^i, \phi)) \wedge f^{\alpha}) & \textit{definition of } g \textit{ and } C \\
\equiv & \phi \;\wedge\; wp(S, (\wedge\ i : \mathbb{N} \bullet wp(S^i, \phi))) \;\wedge\; wp(S, f^{\alpha}) & \textit{conjunction} \\
\equiv & wp(S^0, \phi) \wedge (\wedge\ i \in \mathbb{N} \bullet wp(S^{i+1}, \phi)) \wedge f^{\alpha+1} & \textit{conjunction, definition of } f \\
\equiv & C \wedge f^{\alpha+1} & \textit{join conjuncts}
\end{array}$$

□

Lemma B.3 *For each ordinal α,* $\quad g^{\alpha} \;\equiv\; C \wedge f^{\alpha}.$

Proof by ordinal induction:

$$\begin{aligned}
g^{\alpha} &\equiv (\vee\, \beta \mid \beta < \alpha \bullet g(g^{\beta})) && \textit{definition of } g^{\alpha}\\
&\equiv (\vee\, \beta \mid \beta < \alpha \bullet g(C \wedge f^{\beta})) && \textit{induction hypothesis}\\
&\equiv (\vee\, \beta \mid \beta < \alpha \bullet C \wedge f^{\beta+1}) && \textit{Lemma B.2}\\
&\equiv C \wedge (\vee\, \beta \mid \beta < \alpha \bullet f^{\beta+1}) && C \textit{ is independent of } \beta\\
&\equiv C \wedge f^{\alpha} && \textit{definition of } f^{\alpha}
\end{aligned}$$

□

Lemma B.4 *For ordinal α, let f^{α} be a fixed-point of f. Then g^{α} is a fixed-point of g.*

Proof: Follows from Lemma B.3.

Theorem 4.2 *Let S^i be i copies of S composed sequentially ($S^0 =$ **skip**) and let S^{∞} be the infinite sequential composition of S (Definition 3.9). Then for any predicate ϕ*

$$wp(\mathbf{it}\ S\ \mathbf{ti}, \phi) \equiv (\wedge\, i \in \mathbb{N} \bullet wp(S^i, \phi)) \wedge \neg\, \overline{inf}(S^{\infty}).$$

Proof:

$$\begin{aligned}
& C \wedge \neg\, \overline{inf}(S^{\infty})\\
\equiv\ & C \wedge f^{\alpha} && \textit{by } \mathrm{(i)} \textit{ and Theorem 3.1, where } f^{\alpha} \equiv f^{\alpha+1}\\
\equiv\ & g^{\alpha} && \textit{Lemma B.3}\\
\equiv\ & wp(\mathbf{it}\ S\ \mathbf{ti}, \phi) && \textit{by Lemma B.4, Theorem 3.1 and } \mathrm{(ii)}.
\end{aligned}$$

Proof of Lemma 4.3

Firstly, we have Lemmas B.5 and B.6:

Lemma B.5 ${P_B}^{i} = ([]\, t \in B^* \mid \#t = i \bullet P_t)$.

Proof by induction on i: Case $i = 0$ is trivial. For case $i+1$, we have

$$\begin{aligned}
{P_B}^{i+1} &= ([]\, t \in B^* \mid \#t = i \bullet P_t) \,;\, ([]\, b \in B \bullet P_b)\\
&= ([]\, t \in B^* \mid \#t = i \bullet ([]\, b \in B \bullet P_{t\langle b\rangle})\\
&= ([]\, t' \in B^* \mid \#t = i+1 \bullet P_{t'})
\end{aligned}$$

□

Lemma B.6 $\overline{inf}({P_B}^{\infty}) \equiv (\vee\, u \in B^{\omega} \bullet \overline{inf}(P_u))$.

Proof: Let $R \mathrel{\widehat{=}} (\vee\, u \in B^{\omega} \bullet \overline{inf}(P_u))$.
(a). It can be shown that R is a solution of

$$X \Rrightarrow \overline{wp}(P_B, X) \qquad \text{(iii)}$$

and since $\overline{inf}({P_B}^{\infty})$ is the greatest solution of (iii), we get $R \Rrightarrow \overline{inf}({P_B}^{\infty})$.

(b). Assume $\overline{inf}(P_B{}^\infty)$. By the definition of $\overline{inf}$ we have an infinite sequence of predicates Φ such that $\Phi_0 \equiv \overline{inf}(P_B{}^\infty)$ and, for each i,

$$\begin{aligned} \Phi_i &\Rrightarrow \overline{wp}(P_B,\ \Phi_{i+1}) \\ &\Rrightarrow (\vee\ b \in B \bullet \overline{wp}(P_b,\ \Phi_{i+1})). \end{aligned}$$

So for each i we can choose some $b \in B$ to construct an infinite sequence $u \in B^\omega$ such that

$$\Phi_i \ \Rrightarrow\ \overline{wp}(P_{u_i},\ \Phi_{i+1}).$$

Therefore $\Phi_0 \Rrightarrow \overline{inf}(P_u)$ for such a u, hence, $\overline{inf}(P_B{}^\infty) \Rrightarrow (\vee\ u \in B^\omega \bullet \overline{inf}(P_u))$.
□

Lemma 4.3 *If* $P_B = ([\!]\ b \in B \bullet P_b)$ *then*

$$wp(\textbf{it}\ P_B\ \textbf{ti}, \phi)\ \equiv\ (\wedge\ t \in B^* \bullet wp(P_t, \phi))\ \wedge\ (\wedge\ u \in B^\omega \bullet \neg\ \overline{inf}(P_u)).$$

Proof:

$$\begin{aligned} & wp(\textbf{it}\ P_B\ \textbf{ti}, \phi) \\ \equiv\ & (\wedge\ i \in \mathbb{N} \bullet wp(P_B{}^i, \phi)) \wedge \neg\ \overline{inf}(P_B{}^\infty) && \textit{Theorem 4.2} \\ \equiv\ & (\wedge\ i \in \mathbb{N}, t \in B^* \mid \#t = i \bullet wp(P_t, \phi)) \\ & \wedge \neg\ (\vee\ u \in B^\omega \bullet \overline{inf}(P_u)) && \textit{Lemmas B.5 and B.6} \\ \equiv\ & (\wedge\ t \in B^* \bullet wp(P_t, \phi)) \\ & \wedge (\wedge\ u \in B^\omega \bullet \neg\ \overline{inf}(P_u)) && \textit{predicate calculus.} \end{aligned}$$

Proof of Lemma 4.4

Lemma 4.4 $\quad$ **it** S **ti** ; **it** S **ti** $=$ **it** S **ti**.

Proof: Firstly, assuming $i, j \in \mathbb{N}$, we have

$$\begin{aligned} & wp(S^i, wp(\textbf{it}\ S\ \textbf{ti}, \phi)) \\ \equiv\ & wp(S^i, (\wedge\ j \bullet wp(S^j, \phi)))\ \wedge\ wp(S^i, \neg\ \overline{inf}(S^\infty)) && \textit{Theorem 4.2, conjunction} \\ \equiv\ & (\wedge\ j \bullet wp(S^{i+j}, \phi))\ \wedge\ \neg\ \overline{inf}(S^i\ ;\ S^\infty) && \textit{conjunction, Theorem 3.8} \\ \equiv\ & (\wedge\ j \bullet wp(S^{i+j}, \phi))\ \wedge\ \neg\ \overline{inf}(S^\infty) && \textit{definition of } S^\infty. \end{aligned}$$

So that

$$\begin{aligned} & wp(\textbf{it}\ S\ \textbf{ti}\ ;\ \textbf{it}\ S\ \textbf{ti}, \phi) \\ \equiv\ & (\wedge\ i \bullet wp(S^i, wp(\textbf{it}\ S\ \textbf{ti}, \phi)))\ \wedge\ \neg\ \overline{inf}(S^\infty) && \textit{Theorem 4.2} \\ \equiv\ & (\wedge\ i \bullet (\wedge\ j \bullet wp(S^{i+j}, \phi)) \wedge \neg\ \overline{inf}(S^\infty))\ \wedge\ \neg\ \overline{inf}(S^\infty) && \textit{from above} \\ \equiv\ & (\wedge\ i \bullet wp(S^i, \phi))\ \wedge\ \neg\ \overline{inf}(S^\infty) && \textit{pred. calc.} \\ \equiv\ & wp(\textbf{it}\ S\ \textbf{ti}, \phi) && \textit{Theorem 4.2.} \end{aligned}$$

Proof of Lemma 4.5

Lemma 4.5 *If* $P_B = ([\!]\, b \in B \bullet P_b)$ *and* $P_C = ([\!]\, c \in C \bullet P_c)$ *then*

$$\begin{array}{rcll} \textbf{it } P_B \,[\!]\, P_C \textbf{ ti} & = & \textbf{it } P_B \textbf{ ti} \,;\, \textbf{it } (P_C \,;\, \textbf{it } P_B \textbf{ ti}) \textbf{ ti} & \text{(iv)} \\ & = & \textbf{it } (\textbf{it } P_B \textbf{ ti} \,;\, P_C) \textbf{ ti} \,;\, \textbf{it } P_B \textbf{ ti}. & \text{(v)} \end{array}$$

Proof: We prove case (iv). Proof of case (v) is similar. Firstly we have lemmas as follows:

Lemma B.7 *For* $i > 0$,

$$\begin{array}{ll} & wp((P_C \,;\, \textbf{it } P_B \textbf{ ti})^i, \phi) \\ \equiv & (\wedge\, t \in (B \cup C)^* \mid \#(t \restriction C) = i \wedge \mathit{fst}(t) \in C \bullet wp(P_t, \phi)) \\ & (\wedge\, u \in (B \cup C)^\omega \mid 0 < \#(u \restriction C) \leq i \wedge \mathit{fst}(u) \in C \bullet \neg\, \overline{\mathit{inf}}(P_u)). \end{array}$$

Proof: By induction on i, using Lemma 4.3.

Lemma B.8

$$\overline{\mathit{inf}}((P_C \,;\, \textbf{it } P_B \textbf{ ti})^\infty) \;\equiv\; (\vee\, u \in (B \cup C)^\omega \mid \mathit{fst}(u) \in C \bullet \neg\, \overline{\mathit{inf}}(P_u)).$$

Proof: Similar to the proof of Lemma B.6.

Lemma B.9

$$\begin{array}{rcl} wp(\textbf{it } (P_C \,;\, \textbf{it } P_B \textbf{ ti}) \textbf{ ti}, \phi) & \equiv & \phi \\ & & \wedge\, (\wedge\, t \in (B \cup C)^* \mid \mathit{fst}(t) \in C \bullet wp(P_t, \phi)) \\ & & \wedge\, (\wedge\, u \in (B \cup C)^\omega \mid \mathit{fst}(u) \in C \bullet \neg\, \overline{\mathit{inf}}(P_u)). \end{array}$$

Proof: From Lemmas B.7 and B.8.

Finally we prove case (iv) of Lemma 4.5:

$$\begin{array}{lll} & wp(\textbf{it } P_B \textbf{ ti} \,;\, \textbf{it } (P_C \,;\, \textbf{it } P_B \textbf{ ti}) \textbf{ ti}, \phi) & \\ \equiv & (\wedge\, t \in B^* \bullet wp(P_t, wp(\textbf{it } (P_C \,;\, \textbf{it } P_B \textbf{ ti}) \textbf{ ti}, \phi))) & \\ & \wedge\, (\wedge\, u \in B^\omega \bullet \neg\, \overline{\mathit{inf}}(P_u)) & \textit{Lemma 4.3} \\ \equiv & (\wedge\, t \in B^* \bullet wp(P_t, \phi)) & \\ & \wedge\, (\wedge\, t \in B^*, s \in (B \cup C)^* \mid \mathit{fst}(s) \in C \bullet wp(P_{ts}, \phi)) & \\ & \wedge\, (\wedge\, t \in B^*, v \in (B \cup C)^\omega \mid \mathit{fst}(v) \in C \bullet \neg\, \overline{\mathit{inf}}(P_{tv})) & \\ & \wedge\, (\wedge\, u \in B^\omega \bullet \neg\, \overline{\mathit{inf}}(P_u)) & \textit{Lemma B.9} \\ \equiv & (\wedge\, t \in (B \cup C)^* \mid t \restriction C = \langle\rangle \bullet wp(P_t, \phi)) & \\ & \wedge\, (\wedge\, t \in (B \cup C)^* \mid t \restriction C \neq \langle\rangle \bullet wp(P_t, \phi)) & \\ & \wedge\, (\wedge\, u \in (B \cup C)^\omega \mid u \restriction C \neq \langle\rangle \bullet \neg\, \overline{\mathit{inf}}(P_u)) & \\ & \wedge\, (\wedge\, u \in (B \cup C)^\omega \mid u \restriction C = \langle\rangle \bullet \neg\, \overline{\mathit{inf}}(P_u)) & \textit{pred. calc.} \\ \equiv & (\wedge\, t \in (B \cup C)^* \bullet wp(P_t, \phi)) & \\ & \wedge\, (\wedge\, u \in (B \cup C)^\omega \bullet \neg\, \overline{\mathit{inf}}(P_u)) & \textit{pred. calc.} \\ \equiv & wp(P_B \,[\!]\, P_C, \phi) & \textit{Lemma 4.3.} \end{array}$$

Proof of Lemma 4.12

Lemma 4.12 *For action* $b \in B \cup \{\imath\}$,

$$Q_b * G \quad = \quad P_b * H \ \ ; \ \mathbf{it}\ ([\!]\, c \in C \bullet P_c * H)\ \mathbf{ti}.$$

Proof:

$$\begin{array}{cll}
 & Q_b * G & \\
= & P_b \ ; IT_{H \cup C} & \textit{Definition 4.6} \\
= & P_b \ ; IT_H \ ; \mathbf{it}\ ([\!]\, c \in C \bullet P_c) \ ; IT_H \ \mathbf{ti} & \textit{Lemma 4.5} \\
= & P_b * H \ ; \mathbf{it}\ ([\!]\, c \in C \bullet P_c * H)\ \mathbf{ti} & \textit{distribution of } ; , \textit{ Definition 4.6.}
\end{array}$$

Proof of Lemma 4.13

Lemma 4.13 *For action* $b \in B \cup \{\imath\}$, *predicate* ϕ,

$$\begin{array}{rcl}
\overline{wp}(Q_b * G, \phi) & \equiv & (\vee\, s \in C^* \bullet \overline{wp}(P_{\langle b \rangle s} * H, \phi)) \\
 & & \vee\, (\vee\, u \in C^\omega \bullet \overline{inf}(P_{\langle b \rangle u} * H)).
\end{array}$$

Proof:

$$\begin{array}{cll}
 & \overline{wp}(Q_b * G, \phi) & \\
\equiv & \overline{wp}(P_b * H, \overline{wp}(\mathbf{it}\ P_C * H\ \mathbf{ti}, \phi)) & \textit{Lemma 4.12} \\
\equiv & \overline{wp}(P_b * H, (\vee\, s \in C^* \bullet \overline{wp}(P_s * H, \phi)) & \\
 & \qquad \vee\, (\vee\, u \in C^\omega \bullet \overline{inf}(P_u * H))\ ) & \\
 & \qquad\qquad \textit{Lemma 4.3 with } P_B = ([\!]\, c \in C \bullet P_c * H) & \\
\equiv & (\vee\, s \in C^* \bullet \overline{wp}(P_{\langle b \rangle s} * H, \phi)) & \\
 & \vee\, (\vee\, u \in C^\omega \bullet \overline{inf}(P_{\langle b \rangle u} * H)) & \textit{disjunction.}
\end{array}$$

Proof of Lemma 4.14

Lemma 4.14 *For action trace* $r \in B^*$, *predicate* ϕ,

$$\begin{array}{rcl}
\overline{wp}(Q_{\langle \imath \rangle r} * G, \phi) & \equiv & (\vee\, s \in A^* \mid s \upharpoonright B = r \bullet \overline{wp}(P_{\langle \imath \rangle s} * H, \phi)) \\
 & & \vee\, (\vee\, u \in A^\omega \mid u \upharpoonright B \le r \bullet \overline{inf}(P_{\langle \imath \rangle u} * H)).
\end{array}$$

Proof by induction on r:
Case $\langle\rangle$:

$$\begin{array}{cll}
 & \overline{wp}(Q_{\langle \imath \rangle} * G, \phi) & \\
\equiv & (\vee\, s \in C^* \bullet \overline{wp}(P_{\langle \imath \rangle s} * H, \phi)) & \\
 & \vee\, (\vee\, u \in C^\omega \bullet \overline{inf}(P_{\langle \imath \rangle u} * H)) & \textit{Lemma 4.13} \\
\equiv & (\vee\, s \in A^* \mid s \upharpoonright B = \langle\rangle \bullet \overline{wp}(P_{\langle \imath \rangle s} * H, \phi)) & \\
 & \vee\, (\vee\, u \in A^\omega \mid u \upharpoonright B \le \langle\rangle \bullet \overline{inf}(P_{\langle \imath \rangle u} * H)) & \textit{pred. calc.}
\end{array}$$

Case $r\langle b\rangle, \qquad b \in B$:

$$
\begin{array}{cll}
 & \overline{wp}(Q_{\langle \imath\rangle r\langle b\rangle} * G, \phi) & \\
\equiv & \overline{wp}(Q_{\langle \imath\rangle r} * G,\ \overline{wp}(Q_b * G, \phi)\ ) & \\
\equiv & \overline{wp}(Q_{\langle \imath\rangle r} * G, (\vee\ s' \in C^* \bullet \overline{wp}(P_{\langle b\rangle s'} * H, \phi))) & \\
 & \vee\ \overline{wp}(Q_{\langle \imath\rangle r} * G, (\vee\ u \in C^\omega \bullet \overline{inf}(P_{\langle b\rangle u} * H))) & \textit{Lemma 4.13} \\
\equiv & (\vee\ s \in A^* \mid s\restriction B = r \bullet \overline{wp}(P_{\langle \imath\rangle s} * H, (\vee\ s' \in C^* \bullet \overline{wp}(P_{\langle b\rangle s'} * H, \phi)))) & \\
 & \vee\ (\vee\ u \in A^\omega \mid u\restriction B \leq r \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)) & \\
 & \vee\ (\vee\ s \in A^* \mid s\restriction B = r \bullet \overline{wp}(P_{\langle \imath\rangle s} * H, (\vee\ u \in C^\omega \bullet \overline{inf}(P_{\langle b\rangle u} * H)))) & \textit{ind. hyp.} \\
\equiv & \text{(vi)} \vee \text{(vii)} \vee \text{(viii)} &
\end{array}
$$

$$
\begin{array}{ccll}
\text{(vi)} & \equiv & (\vee\ s \in A^*, s' \in C^* \mid s\restriction B = r \bullet \overline{wp}(P_{\langle \imath\rangle s\langle b\rangle s'} * H, \phi)) & \textit{disjunction} \\
 & \equiv & (\vee\ s \in A^*, s' \in C^* \mid (s\langle b\rangle s')\restriction B = r\langle b\rangle \bullet \overline{wp}(P_{\langle \imath\rangle s\langle b\rangle s'} * H, \phi)) & \textit{pred. calc.} \\
 & \equiv & (\vee\ s \in A^* \mid s\restriction B = r\langle b\rangle \bullet \overline{wp}(P_{\langle \imath\rangle s} * H, \phi)) & \textit{pred. calc.}
\end{array}
$$

$$
\begin{array}{ccll}
\text{(vii)} & \equiv & (\vee\ u \in A^\omega \mid u\restriction B \leq r \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)) & \\
 & \equiv & (\vee\ u \in A^\omega \mid u\restriction B < r\langle b\rangle \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)) & \textit{pred. calc.}
\end{array}
$$

$$
\begin{array}{ccll}
\text{(viii)} & \equiv & (\vee\ s \in A^*, u \in C^\omega \mid s\restriction B = r \bullet \overline{inf}(P_{\langle \imath\rangle s\langle b\rangle u} * H)) & \textit{disjunction} \\
 & \equiv & (\vee\ s \in A^*, u \in C^\omega \mid (s\langle b\rangle u)\restriction B = r\langle b\rangle \bullet \overline{inf}(P_{\langle \imath\rangle s\langle b\rangle u} * H)) & \textit{pred. calc.} \\
 & \equiv & (\vee\ u \in A^\omega \mid u\restriction B = r\langle b\rangle \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)) & \textit{pred. calc.}
\end{array}
$$

$$
\begin{array}{cll}
 & \text{(vi)} \vee \text{(vii)} \vee \text{(viii)} & \\
\equiv & (\vee\ s \in A^* \mid s\restriction B = r\langle b\rangle \bullet \overline{wp}(P_{\langle \imath\rangle s} * H, \phi)\ ) & \\
 & \vee\ (\vee\ u \in A^\omega \mid u\restriction B < r\langle b\rangle \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)\ ) & \\
 & \vee\ (\vee\ u \in A^\omega \mid u\restriction B = r\langle b\rangle \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)\ ) & \\
\equiv & (\vee\ s \in A^* \mid s\restriction B = r\langle b\rangle \bullet \overline{wp}(P_{\langle \imath\rangle s} * H, \phi)\ ) & \\
 & \vee\ (\vee\ u \in A^\omega \mid u\restriction B \leq r\langle b\rangle \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)\ ) & \textit{pred. calc.}
\end{array}
$$

Proof of Lemma 4.15

Lemma 4.15 *For action trace* $v \in B^\omega$,

$$\overline{inf}(Q_{\langle \imath\rangle v} * G) \ \equiv\ (\vee\ u \in A^\omega \mid u\restriction B \leq v \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)).$$

Proof by mutual entailment:
(a). Let $v' = \langle \imath\rangle v$ and $\Phi \equiv (\mu\ Y \bullet \overline{wp}_\infty(Q_{v'} * G, Y))$. So for each $i \geq 0$,

$$
\begin{array}{cll}
\Phi_i & \Rrightarrow & \overline{wp}(Q_{v'_i} * G, \Phi_{i+1}) \\
 & \Rrightarrow & (\vee\ s \in C^* \bullet \overline{wp}(P_{\langle v'_i\rangle s} * H, \Phi_{i+1})) \\
 & & \vee\ (\vee\ u \in C^\omega \bullet \overline{inf}(P_{\langle v'_i\rangle u} * H)) \qquad \textit{Lemma 4.13.}
\end{array}
$$

Now, either for each $i \geq 0$,

$$\Phi_i \;\Rrightarrow\; (\vee\, s \in C^* \bullet \overline{wp}(P_{\langle v'_i\rangle s} * H, \Phi_{i+1})) \tag{ix}$$

or, there is a $j \geq 0$ such that

$$\begin{array}{lll} \Phi_i & \Rrightarrow & (\vee\, s \in C^* \bullet \overline{wp}(P_{\langle v'_i\rangle s} * H, \Phi_{i+1})) \quad \text{each } i < j \\ \text{and} & & \\ \Phi_j & \Rrightarrow & (\vee\, u \in C^\omega \bullet \overline{inf}(P_{\langle v'_i\rangle u} * H)). \end{array} \tag{x}$$

Given (ix) we can construct an infinite sequence $u \in A^\omega$ such that

$$\langle \imath\rangle(u\restriction B) = v' = \langle \imath\rangle v$$

and

$$\begin{array}{lll} \Phi_0 & \Rrightarrow & \overline{inf}(P_{\langle \imath\rangle u} * H) \\ & \Rrightarrow & (\vee\, u \in A^\omega \mid u\restriction B \leq v \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)). \end{array}$$

Given (x) with $j = 0$, we immediately have

$$\begin{array}{lll} \Phi_0 & \Rrightarrow & (\vee\, u \in C^\omega \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)) \\ & \Rrightarrow & (\vee\, u \in A^\omega \mid u\restriction B \leq v \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)). \end{array}$$

Given (x) with $j > 0$, then there exists some $t \in A^*$ such that

$$\langle \imath\rangle(t\restriction B) = v'[0..j-1]$$

and

$$\begin{array}{lll} \Phi_0 & \Rrightarrow & \overline{wp}(P_{\langle \imath\rangle t} * H, \Phi_j) \\ & \Rrightarrow & \overline{wp}(P_{\langle \imath\rangle t} * H, (\vee\, u \in C^\omega \bullet \overline{inf}(P_{\langle v'_j\rangle u} * H))) \\ & \Rrightarrow & (\vee\, u \in C^\omega \bullet \overline{inf}(P_{\langle \imath\rangle t\langle v'_j\rangle u} * H)) \\ & \Rrightarrow & (\vee\, u \in A^\omega \mid u\restriction B \leq v \bullet \overline{inf}(P_{\langle \imath\rangle u} * H)). \end{array}$$

(b). Assume we have a trace $u \in A^\omega$ such that $\overline{inf}(P_{\langle \imath\rangle u} * H)$.

Case $u\restriction B = v$: Let $u' = \langle \imath\rangle u$ and $v' = \langle \imath\rangle v$. Since $u\restriction B = v$ there exist $s_0, s_1, \ldots$ such that

$$u' = \langle v'_0\rangle s_0 \langle v'_1\rangle s_1 \ldots$$

There is an infinite sequence of predicates Ψ such that $\Psi_0 \equiv \overline{inf}(P_{u'} * H)$ and for each $i \geq 0$

$$\begin{array}{llll} \Psi_i & \Rrightarrow & \overline{wp}(P_{\langle v'_i\rangle s_i} * H, \Psi_{i+1}) & \\ & \Rrightarrow & (\vee\, s \in C^* \bullet \overline{wp}(P_{\langle v'_i\rangle s} * H, \Psi_{i+1})) & \\ & \Rrightarrow & \overline{wp}(Q_{v'_i} * G, \Psi_{i+1}) & \textit{Lemma 4.13} \end{array}$$

so that

$$\Psi_0 \quad \Rrightarrow \quad \overline{\mathit{inf}}(Q_{v'} * G).$$

Case $u \restriction B < v$:

$$\begin{array}{ll}
 & u \restriction B < v \wedge \overline{\mathit{inf}}(P_{\langle\imath\rangle u} * H) \\
\Rrightarrow & (\vee\, s \in A^*, u' \in C^\omega \mid s \restriction B < v \bullet \\
 & \qquad \overline{\mathit{inf}}(P_{\langle\imath\rangle s u'} * H)) \qquad\qquad \textit{pred. calc.} \\
\Rrightarrow & (\vee\, s \in A^* \mid s \restriction B < v \bullet \\
 & \qquad \overline{wp}(P_{\langle\imath\rangle s} * H, (\vee\, u' \in C^\omega \bullet \overline{\mathit{inf}}(P_{u'} * H)))) \qquad \textit{disjunction} \\
\Rrightarrow & (\vee\, s \in A^* \mid s \restriction B < v \bullet \\
 & \qquad \overline{wp}(P_{\langle\imath\rangle s} * H, \overline{wp}(\mathbf{it}\ P_C * H\ \mathbf{ti}, \mathit{false}))) \qquad \textit{Lemma 4.3} \\
\Rrightarrow & (\vee\, s \in A^*, t \in B^* \mid t = s \restriction B \wedge t < v \bullet \\
 & \qquad \overline{wp}(P_{\langle\imath\rangle s} * H, \overline{wp}(IT_G, \mathit{false}))) \\
\Rrightarrow & (\vee\, t \in B^* \mid t < v \bullet \overline{wp}(Q_{\langle\imath\rangle t} * G, \mathit{false})) \qquad \textit{Lemma 4.14} \\
\Rrightarrow & (\vee\, t \in B^*, v' \in B^\omega \mid t v' = v \bullet \\
 & \qquad \overline{wp}(Q_{\langle\imath\rangle t} * G, \overline{\mathit{inf}}(Q_{v'} * G))) \qquad \textit{monotonicity} \\
\Rrightarrow & \overline{\mathit{inf}}(Q_{\langle\imath\rangle v} * G).
\end{array}$$

Proof of Lemma 4.16

Let $P_C \mathrel{\widehat{=}} ([]\, c \in C \bullet P_c)$. Then we have lemmas as follows:

Lemma B.10 $IT_G \;=\; \mathbf{it}\ IT_H\ ;\ P_C\ \mathbf{ti}\ ;\ IT_H$.

Proof: Follows easily from Lemma 4.5.

Lemma B.11 $\overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \neg\, gd(S)) \;\equiv\; \overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \mathit{true})$.

Proof: $\overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \neg\, gd(S))$ is the greatest solution of

$$X \quad \Rrightarrow \quad \neg\, gd(S) \vee \overline{wp}(S, X) \tag{xi}$$

We can easily show that *true* is a solution of (xi), and since $\overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \mathit{true}) \equiv \mathit{true}$, the lemma follows.
□

Lemma B.12 *For predicate* ϕ, $\quad \phi \;\Rrightarrow\; \overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \phi)$.

Proof: $\phi \;\Rrightarrow\; \phi \vee \overline{wp}(S, \overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \phi)) \;\Rrightarrow\; \overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \phi)$
□

Lemma B.13 $\overline{wp}(IT_G, \mathit{true}) \;\equiv\; \overline{wp}(IT_G, \neg\, gd_H(P_C))$.

Proof: If $C = \{\}$ then lemma is trivially true. If $C \neq \{\}$ then

$$
\begin{array}{lll}
 & \overline{wp}(IT_G, true) & \\
\equiv & \overline{wp}(\mathbf{it}\ IT_H\ ;\ P_C\ \mathbf{ti}\ ;\ IT_H, true) & \textit{Lemma B.10} \\
\equiv & \overline{wp}(\mathbf{it}\ IT_H\ ;\ P_C\ \mathbf{ti}, true) & \textit{since } \overline{wp}(IT_H, true) \equiv true \\
\equiv & \overline{wp}(\mathbf{it}\ IT_H\ ;\ P_C\ \mathbf{ti}, \neg\ gd(IT_H\ ;\ P_C)) & \textit{Lemma B.11} \\
\equiv & \overline{wp}(\mathbf{it}\ IT_H\ ;\ P_C\ \mathbf{ti}, \neg\ gd_H(P_C)) & \textit{since } c \neq \{\} \\
\equiv & \overline{wp}(IT_G, \neg\ gd_H(P_C)) & \textit{Lemmas B.12,B.10}
\end{array}
$$

Also, $\overline{wp}(IT_G, \neg\ gd_H(P_C)) \Rrightarrow \overline{wp}(IT_G, true)$ by monotonicity.
□

Lemma B.14 *For* $\{\} \subset X \subseteq B$,

$$\neg\ gd_H(P_X) \wedge \neg\ gd(IT_H\ ;\ P_C) \quad \equiv \quad \neg\ gd_H(P_{X \cup C}).$$

Proof:

$$
\begin{array}{lll}
 & \neg\ gd_H(P_X) \wedge \neg\ gd(IT_H\ ;\ P_C) & \\
\equiv & (\wedge\ a \in X \bullet wp(IT_H, \neg\ gd(P_a))) \wedge wp(IT_H, \neg\ gd(P_C)) & \textit{Definition 4.7} \\
\equiv & wp(IT_H, \neg\ gd(P_X)) \wedge wp(IT_H, \neg\ gd(P_C)) & \textit{since } X \neq \{\} \\
\equiv & wp(IT_H, \neg\ gd(P_{X \cup C})) & \textit{conjunction} \\
\equiv & \neg\ gd_H(P_{X \cup C}) & \textit{since } X \neq \{\}
\end{array}
$$

□

Lemma B.15 *For predicate* ϕ,

$$wp(\mathbf{it}\ S\ \mathbf{ti}, \phi) \ \Rrightarrow\ \overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \phi \wedge \neg\ gd(S)).$$

Proof: Let $C \mathrel{\hat=} \overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \phi \wedge \neg\ gd(S))$. Can show that C is a solution of

$$\phi \wedge wp(S, X) \quad \Rrightarrow \quad X \tag{xii}$$

Now, since $wp(\mathbf{it}\ S\ \mathbf{ti}, \phi)$ is the least solution of (xii), we have $wp(\mathbf{it}\ S\ \mathbf{ti}, \phi) \Rrightarrow C$.
□

Lemma B.16 $\phi \wedge \neg\ gd(S) \ \Rrightarrow\ wp(\mathbf{it}\ S\ \mathbf{ti}, \phi)$.

Proof:

$$
\begin{array}{lll}
 & \phi \wedge \neg\ gd(S) & \\
\Rrightarrow & \phi \wedge wp(S, false) & \textit{Definition 2.12} \\
\Rrightarrow & \phi \wedge wp(S, wp(\mathbf{it}\ S\ \mathbf{ti}, \phi)) & \textit{monotonicity} \\
\Rrightarrow & wp(\mathbf{it}\ S\ \mathbf{ti}, \phi) & \textit{Definition 4.1}
\end{array}
$$

□

Lemma B.17 *For* $\{\} \subset X \subseteq B$, $\quad \neg\ gd_G(Q_X) \ \Rrightarrow\ \overline{wp}(IT_G, \neg\ gd_H(P_{H \cup C}))$.

Proof:

$$
\begin{array}{lll}
 & \neg\, gd_G(Q_X) & \\
\Rrightarrow & wp(IT_G, \neg\, gd(P_X)) & \textit{conjunction, } P_X = Q_X \\
\Rrightarrow & wp(\textbf{it } IT_H \;;\; P_C \textbf{ ti}, wp(IT_H, \neg\, gd(P_X))) & \textit{Lemma B.10} \\
\Rrightarrow & \overline{wp}(\textbf{it } IT_H \;;\; P_C \textbf{ ti}, wp(IT_H, \neg\, gd(P_X)) \wedge \neg\, gd(IT_H \;;\; P_C)) & \textit{Lemma B.15} \\
\Rrightarrow & \overline{wp}(\textbf{it } IT_H \;;\; P_C \textbf{ ti}, \neg\, gd_H(P_{X\cup C})) & \textit{Lemma B.14} \\
\Rrightarrow & \overline{wp}(\textbf{it } IT_H \;;\; P_C \textbf{ ti} \;;\; IT_H, \neg\, gd_H(P_{X\cup C})) & \textit{Lemma B.12} \\
\Rrightarrow & \overline{wp}(IT_G, \neg\, gd_H(P_{X\cup C})) & \textit{Lemma B.10}
\end{array}
$$

□

Lemma B.18 *For* $\{\} \subset X \subseteq B$, $\qquad \neg\, gd_H(P_{X\cup C}) \Rrightarrow \neg\, gd_G(Q_X)$

Proof:

$$
\begin{array}{lll}
 & \neg\, gd_H(P_{X\cup C}) & \\
\Rrightarrow & \neg\, gd_H(P_X) \wedge \neg\, gd(IT_H \;;\; P_C) & \textit{Lemma B.14} \\
\Rrightarrow & wp(\textbf{it } IT_H \;;\; P_C \textbf{ ti}, \neg\, gd_H(P_X)) & \textit{Lemma B.16} \\
\Rrightarrow & wp(\textbf{it } IT_H \;;\; P_C \textbf{ ti}, wp(IT_H, \neg\, gd(P_X))) & \textit{since } X \neq \{\} \\
\Rrightarrow & wp(IT_G, \neg\, gd(Q_X)) & \textit{Lemma B.10, } P_X = Q_X \\
\Rrightarrow & \neg\, gd_G(Q_X) & \textit{since } X \neq \{\}
\end{array}
$$

□

Lemma 4.16 *For action set* $X \subseteq B$,

$$\overline{wp}(IT_G, \neg\, gd_G(Q_X)) \equiv \overline{wp}(IT_G, \neg\, gd_H(P_{X\cup C})).$$

Proof: Case $X = \{\}$ follows from Lemma B.13. Case $X \neq \{\}$ follows from Lemmas B.17 and B.18.

Proof of Theorem 4.20

Theorem 4.20 *If* WF *is some well-founded set ordered by* $<$, E *is some expression in the state-variables, and for some predicate* ϕ

$$
\begin{array}{rcl}
\phi & \Rrightarrow & E \in WF \\
E = e \wedge \phi & \Rrightarrow & wp(S, E < e \wedge \phi)
\end{array}
$$

then $\qquad \phi \Rrightarrow wp(\textbf{it } S \textbf{ ti}, \phi).$

Proof: It can be shown that induction over a set is valid precisely when that set is well-founded (e.g. [DS90, Chapter 9]). Let

$$Y \quad \equiv \quad wp(\mathbf{it}\ S\ \mathbf{ti}, \phi)$$

By induction over $x \in WF$, we can show that $E = x \wedge \phi \ \Rrightarrow\ Y$:

$$\begin{array}{ll@{\qquad}r}
 & E = x \wedge \phi & \\
\Rrightarrow & \phi \wedge wp(S, E < x \wedge \phi) & \textit{antecedent of theorem} \\
\Rrightarrow & \phi \wedge wp(S, (\vee\ x' < x \bullet E = x' \wedge \phi)) & \textit{where } x' \textit{ ind. of } E,\ \phi,\ S \\
\Rrightarrow & \phi \wedge wp(S, (\vee\ x' < x \bullet Y)) & \textit{induction hypothesis, monotonicity} \\
\Rrightarrow & \phi \wedge wp(S, Y) & Y \textit{ independent of } x' \\
\Rrightarrow & Y & Y \textit{ a soln. of } X \equiv \phi \wedge wp(S, X)
\end{array}$$

Finally

$$\begin{array}{lll@{\qquad}r}
\phi & \Rrightarrow & E \in WF \wedge \phi & \textit{antecedent of theorem} \\
 & \Rrightarrow & (\vee\ x \in WF \bullet E = x \wedge \phi) & \textit{predicate calculus} \\
 & \Rrightarrow & (\vee\ x \in WF \bullet Y) & \textit{from above} \\
 & \Rrightarrow & Y & Y \textit{ independent of } x.
\end{array}$$

Appendix C

Proofs for Chapter 5

Proof of Lemma 5.7

Ordinal addition is used in order to prove Lemma 5.7. Ordinal addition is not, in general, commutative, but does satisfy the following properties (α, β represent any ordinal, λ represents limit ordinals, m, n represent ordinals less than ω) [End77]:

Lemma C.1 $m + n = n + m$.

Lemma C.2 $\alpha, \beta < \lambda \;\Rightarrow\; \alpha + \beta < \lambda$.

Lemma C.3 $\alpha < \alpha' \wedge \beta < \beta' \;\Rightarrow\; \alpha + \beta < \alpha' + \beta'$.

Lemma C.4 $\alpha < \alpha' \wedge \beta < \beta' \;\Rightarrow\; \beta + \alpha < \alpha' + \beta'$.

Lemma C.5 $\beta + n = \lambda \;\Leftrightarrow\; \beta = \lambda \wedge n = 0$.

The following lemma about $S \,[]\, T$ is proven using Definition 2.9:

Lemma C.6 *For v-statement S, w-statement T, v-predicate ϕ, w-predicate ψ,*

$$\begin{array}{rcl} \overline{wp}(S \,[]\, T, \phi \wedge \psi) & \equiv & \overline{wp}(S, \phi) \wedge \psi \\ & & \vee \;\; \phi \wedge \overline{wp}(T, \psi) \\ & & \vee \;\; \overline{wp}(S, \mathit{false}) \;\vee\; \overline{wp}(T, \mathit{false}) \end{array}$$

□

Let C be defined as $C \;\widehat{=}\; (\vee\, i \bullet \overline{wp}(S^i, \phi)) \wedge (\vee\, i \bullet \overline{wp}(T^i, \psi))$. For predicate X, define predicate transformers $\overline{f}$, $\overline{g}$, $\overline{h}$ as:

$$\begin{array}{rcl} \overline{f}(X) & \widehat{=} & \phi \wedge \psi \;\vee\; \overline{wp}(S \,[]\, T, X) \\ \overline{g}(X) & \widehat{=} & \overline{wp}(S, X) \\ \overline{h}(X) & \widehat{=} & \overline{wp}(T, X). \end{array}$$

By ordinal induction we get

Lemma C.7 *For each ordinal α,*

$$\overline{f}^{\alpha} \;\equiv\; C \;\vee\; (\vee\, \beta \leq \alpha, n < \omega \mid \beta + n = \alpha \bullet \overline{g}^{\beta} \wedge \overline{h}^{n} \;\vee\; \overline{g}^{n} \wedge \overline{h}^{\beta}).$$

Proof: zero case is trivial.
Case successor ordinal : using Lemmas C.6, C.1 and properties of $\overline{wp}$ it can be shown that, if the equation is true of $\overline{f}^{\alpha}$, then it is also true of $\overline{f}^{\alpha+1}$.
Case limit ordinal :

$$\begin{array}{rcll} \overline{f}^{\lambda} & \equiv & (\wedge\, \alpha < \lambda \bullet C \vee (\vee\, \beta, n \mid \beta + n = \alpha \bullet \overline{g}^{\beta} \wedge \overline{h}^{n} \ \vee\ \overline{g}^{n} \wedge \overline{h}^{\beta})) & \\ & \equiv & C \vee & \\ & & (\wedge\, \alpha < \lambda \bullet (\vee\, \beta, n \mid \beta + n = \alpha \bullet \overline{g}^{\beta} \wedge \overline{h}^{n} \ \vee\ \overline{g}^{n} \wedge \overline{h}^{\beta})) & \text{(i)} \end{array}$$

By predicate calculus

$$\text{(i)} \ \Rrightarrow\ \overline{g}^{\lambda} \wedge \text{(i)} \ \vee\ \neg\, \overline{g}^{\lambda} \wedge \text{(i)} \qquad \text{(ii)}$$

Now,

$$\begin{array}{cll} & \neg\, \overline{g}^{\lambda} \wedge \text{(i)} & \\ \Rrightarrow & \neg\, (\wedge\, \gamma < \lambda \bullet \overline{g}^{\gamma}) & \\ & \wedge\, (\wedge\, \alpha < \lambda \bullet (\vee\, \beta, n \mid \beta + n = \alpha \bullet \overline{g}^{\beta} \wedge \overline{h}^{n} \ \vee\ \overline{g}^{n} \wedge \overline{h}^{\beta})) & \textit{defn. of } \overline{g}^{\gamma} \\ \Rrightarrow & (\vee\, \gamma < \lambda \bullet \neg\, \overline{g}^{\gamma} & \\ & \quad \wedge\, (\wedge\, \alpha < \lambda \bullet (\vee\, \beta, n \mid \beta + n = \alpha \bullet \overline{g}^{\beta} \wedge \overline{h}^{n} \ \vee\ \overline{g}^{n} \wedge \overline{h}^{\beta}))) & \\ & & \textit{negation, distribution} \\ \Rrightarrow & (\vee\, \gamma < \lambda \bullet \neg\, \overline{g}^{\gamma} & \\ & \quad \wedge\, (\wedge\, \alpha < \lambda \bullet (\vee\, \beta, \beta' \mid (\beta + \beta' = \alpha \vee \beta' + \beta = \alpha) \bullet \overline{g}^{\beta'} \wedge \overline{h}^{\beta}))) & \\ \Rrightarrow & (\vee\, \gamma < \lambda \bullet & \\ & \quad \wedge\, (\wedge\, \alpha < \lambda \bullet (\vee\, \beta, \beta' \mid \beta' < \gamma \wedge (\beta + \beta' = \alpha \vee \beta' + \beta = \alpha) \bullet \overline{h}^{\beta}))) & \\ & & \textit{since } \beta' \geq \gamma \wedge \neg\, \overline{g}^{\gamma} \ \Rrightarrow\ \neg\, \overline{g}^{\beta'} \\ \Rrightarrow & (\wedge\, \alpha < \lambda \bullet \overline{h}^{\alpha}) & \textit{see below} \\ \Rrightarrow & \overline{h}^{\lambda}. & \end{array}$$

Take any $\alpha' < \lambda$, show $\overline{h}^{\alpha'}$:

$$\begin{array}{cll} & \alpha' < \lambda & \\ \Rrightarrow & \alpha' + \gamma < \lambda & \textit{Lemma C.2} \\ \Rrightarrow & (\vee\, \beta, \beta' \mid \beta' < \gamma \wedge (\beta + \beta' = \alpha' + \gamma \vee \beta' + \beta = \alpha' + \gamma) \bullet \overline{h}^{\beta}) & \textit{from above} \\ \Rrightarrow & (\vee\, \beta, \beta' \mid \alpha' \leq \beta \bullet \overline{h}^{\beta}) & \textit{Lemmas C.3, C.4} \\ \Rrightarrow & \overline{h}^{\alpha'} & \alpha' \leq \beta \wedge \overline{h}^{\beta} \ \Rrightarrow\ \overline{h}^{\alpha'} \end{array}$$

so we get $(\wedge\, \alpha' < \lambda \bullet \overline{h}^{\alpha'})$. Finally, from (ii), and since $\overline{g}^{\lambda} \Rrightarrow \text{(i)}$, $\overline{h}^{\lambda} \Rrightarrow \text{(i)}$, $\neg\, \overline{g}^{\lambda} \wedge \text{(i)} \Rrightarrow \overline{h}^{\lambda}$, we get

$$\text{(i)} \ \equiv\ \overline{g}^{\lambda} \vee \overline{h}^{\lambda}$$

so that

$$\begin{array}{rcll} \overline{f}^{\lambda} & \equiv & C \vee \overline{g}^{\lambda} \vee \overline{h}^{\lambda} & \\ & \equiv & C \vee (\vee\, \beta, n \mid \beta + n = \lambda \bullet \overline{g}^{\beta} \wedge \overline{h}^{n} \ \vee\ \overline{g}^{n} \wedge \overline{h}^{\beta}) & \textit{Lemma C.5} \end{array}$$

□

Lemma C.8 *For $\alpha, \rho, \sigma \geq \omega$, let $\overline{g}^\rho$ be a fixed-point of $\overline{g}$, $\overline{h}^\sigma$ be a fixed-point of $\overline{h}$, $\alpha \geq \rho + \sigma$, and $\alpha \geq \sigma + \rho$. Then $\overline{f}^\alpha$ is a fixed-point of $\overline{f}$, and*

$$\overline{f}^\alpha \quad \equiv \quad C \vee \overline{g}^\rho \vee \overline{h}^\sigma .$$

Proof: Firstly,

$$\begin{array}{lll}
\overline{f}^\alpha & \equiv & C \\
& & \vee \ \ (\vee\, \beta, n \mid \beta + n = \alpha \bullet \overline{g}^\beta \wedge \overline{h}^n) \\
& & \vee \ \ (\vee\, \beta, n \mid \beta + n = \alpha \bullet \overline{g}^n \wedge \overline{h}^\beta) \qquad\qquad \textit{Lemma C.7} \\
& \equiv & C \\
& & \vee \ \ (\vee\, \beta, n \mid \beta + n = \alpha \bullet \overline{g}^\rho \wedge \overline{h}^n) \qquad \textit{since } \overline{g}^\rho \equiv \overline{g}^\beta \textit{ for } \beta \geq \rho \\
& & \vee \ \ (\vee\, \beta, n \mid \beta + n = \alpha \bullet \overline{g}^n \wedge \overline{h}^\sigma) \qquad \textit{since } \overline{h}^\sigma \equiv \overline{h}^\beta \textit{ for } \beta \geq \sigma \\
& \equiv & C \vee \overline{g}^\rho \vee \overline{h}^\sigma .
\end{array}$$

Secondly,

$$\begin{array}{lll}
& & \overline{f}(\overline{f}^\alpha) \\
& \equiv & \phi \wedge \psi \ \vee \ \overline{wp}(S \mathbin{[\!]} T, C \vee \overline{g}^\rho \vee \overline{h}^\sigma) \\
& \equiv & C \ \vee \ \overline{g}^{\rho+1} \ \vee \ \overline{g}^\rho \wedge \overline{h}^1 \ \vee \ \overline{g}^1 \wedge \overline{h}^\sigma \ \vee \ \overline{h}^{\sigma+1} \quad \textit{Lemma C.6, defn. of } \overline{g}^\alpha, \overline{h}^\alpha \\
& \equiv & C \ \vee \ \overline{g}^\rho \ \vee \ \overline{g}^\rho \wedge \overline{h}^1 \ \vee \ \overline{g}^1 \wedge \overline{h}^\sigma \ \vee \ \overline{h}^\sigma \qquad \textit{since } \overline{g}^\rho \equiv \overline{g}^{\rho+1}, \overline{h}^\sigma \equiv \overline{h}^{\sigma+1} \\
& \equiv & C \ \vee \ \overline{g}^\rho \ \vee \ \overline{h}^\sigma \\
& \equiv & \overline{f}^\alpha
\end{array}$$

□

Lemma 5.7 *For v-statement S, w-statement T,* **it** S **ti** $\|$ **it** T **ti** $\cong$ **it** $S \mathbin{[\!]} T$ **ti***.*

Proof: Let $\overline{g}^\rho$ be a fixed-point of $\overline{g}$, $\overline{h}^\sigma$ be a fixed-point of $\overline{h}$, and let $\alpha \geq \rho + \sigma, \sigma + \rho$. Then

$$\begin{array}{lll}
& & \overline{wp}(\textbf{it } S \mathbin{[\!]} T \textbf{ ti}, \phi \wedge \psi) \\
& \equiv & \overline{f}^\alpha \qquad\qquad\qquad\qquad \textit{Lemma C.8} \\
& \equiv & C \vee \overline{g}^\rho \vee \overline{h}^\sigma \qquad\qquad \textit{Lemma C.8} \\
& \equiv & C \vee \overline{\mathit{inf}}(S^\infty) \vee \overline{\mathit{inf}}(T^\infty) \qquad \textit{Theorem 3.10} \\
& \equiv & \overline{wp}(\textbf{it } S \textbf{ ti}, \phi) \wedge \overline{wp}(\textbf{it } T \textbf{ ti}, \psi) \\
& & \vee\, \overline{\mathit{inf}}(S^\infty) \vee \overline{\mathit{inf}}(T^\infty) \qquad \textit{Theorem 4.2} \\
& \equiv & \overline{wp}(\textbf{it } S \textbf{ ti}, \phi) \wedge \overline{wp}(\textbf{it } T \textbf{ ti}, \psi) \\
& & \vee\, \overline{wp}(\textbf{it } S \textbf{ ti}, \mathit{false}) \\
& & \vee\, \overline{wp}(\textbf{it } T \textbf{ ti}, \mathit{false}) \qquad \textit{since } \overline{wp}(\textbf{it } S \textbf{ ti}, \mathit{false}) \equiv \overline{\mathit{inf}}(S^\infty) \\
& \equiv & \overline{wp}(\textbf{it } S \textbf{ ti} \,\|\, \textbf{it } T \textbf{ ti}, \phi \wedge \psi) \qquad \textit{Lemma 5.11.}
\end{array}$$

Proof of Lemma 5.10

Lemma 5.10 *For action $c \in C_i$,* $\quad R_c * I \cong (\tilde{P}_c * H) \parallel (\tilde{Q}_c * G)$.

Proof:

$$
\begin{array}{ll}
 & \overline{wp}(R_c * I, \phi \wedge \psi) \\
\equiv & \overline{wp}(\tilde{P}_c \parallel \tilde{Q}_c, \overline{wp}(IT_I, \phi \wedge \psi)) \\
\equiv & \overline{wp}(\tilde{P}_c \parallel \tilde{Q}_c, \overline{wp}(IT_H, \phi) \wedge \overline{wp}(IT_G, \psi) \\
 & \qquad\qquad \vee\ \neg\ halt(IT_H) \\
 & \qquad\qquad \vee\ \neg\ halt(IT_G)) \qquad\qquad\qquad \textit{Lemma 5.7, } gd(IT) \equiv true \\
\equiv & \overline{wp}(\tilde{P}_c \parallel \tilde{Q}_c, \overline{wp}(IT_H, \phi) \wedge \overline{wp}(IT_G, \psi)) \\
 & \vee\ \overline{wp}(\tilde{P}_c \parallel \tilde{Q}_c, \neg\ halt(IT_H)) \\
 & \vee\ \overline{wp}(\tilde{P}_c \parallel \tilde{Q}_c, \neg\ halt(IT_G)) \qquad\qquad\qquad \textit{disjunction} \\
\equiv & \overline{wp}(\tilde{P}_c, \overline{wp}(IT_H, \phi)) \wedge \overline{wp}(\tilde{Q}_c, \overline{wp}(IT_G, \psi)) \\
 & \vee\ \overline{wp}(\tilde{P}_c, \neg\ halt(IT_H)) \wedge gd(\tilde{Q}_c) \\
 & \vee\ gd(\tilde{P}_c) \wedge \overline{wp}(\tilde{Q}_c, \neg\ halt(IT_G)) \\
 & \vee\ \neg\ halt(\tilde{P}_c) \wedge gd(\tilde{Q}_c) \\
 & \vee\ gd(\tilde{P}_c) \wedge \neg\ halt(\tilde{Q}_c) \qquad\qquad\qquad \textit{Property 5.1} \\
\equiv & \overline{wp}(\tilde{P}_c * H, \phi) \wedge \overline{wp}(\tilde{Q}_c * G, \psi) \\
 & \vee\ \neg\ halt(\tilde{P}_c * H) \wedge gd(\tilde{Q}_c * G) \\
 & \vee\ gd(\tilde{P}_c * H) \wedge \neg\ halt(\tilde{Q}_c * G) \\
 & \qquad\qquad \textit{since } \neg\ halt(S) \Rrightarrow \neg\ halt(S\ ;\ T),\ gd(S) \equiv gd(S\ ;\ IT) \\
\equiv & \overline{wp}(\tilde{P}_c * H \parallel \tilde{Q}_c * G, \phi \wedge \psi) \qquad\qquad\qquad \textit{Property 5.1.}
\end{array}
$$

Proof of Lemma 5.12

Lemma 5.12 *For action trace $t \in C_i^*$, v-predicate ϕ, w-predicate ψ,*

$$
\begin{array}{lll}
\overline{wp}(R_t * I, \phi \wedge \psi) & \equiv & \overline{wp}(\tilde{P}_t * H, \phi) \wedge \overline{wp}(\tilde{Q}_t * G, \psi) \\
 & & \vee\ (\vee\ s \leq t \bullet\ \ \overline{wp}(\tilde{P}_s * H, false) \wedge \overline{wp}(\tilde{Q}_s * G, true) \\
 & & \qquad\qquad\qquad \vee\ \ \overline{wp}(\tilde{P}_s * H, true) \wedge \overline{wp}(\tilde{Q}_s * G, false)).
\end{array}
$$

Proof by induction over t: Base case is trivially true.

Case $t\langle c \rangle$:

$$
\begin{array}{ll}
& \overline{wp}(R_t * I, \overline{wp}(R_c * I, \phi \wedge \psi)) \\
\equiv & \overline{wp}(R_t * I, \overline{wp}(\tilde{P}_c * H, \phi) \wedge \overline{wp}(\tilde{Q}_c * G, \psi)) \\
& \vee \;\; \overline{wp}(R_t * I, \overline{wp}(\tilde{P}_c * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_c * G, \mathit{true})) \\
& \vee \;\; \overline{wp}(R_t * I, \overline{wp}(\tilde{P}_c * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_c * G, \mathit{false})) \qquad \textit{Lemma 5.10} \\
\equiv & \overline{wp}(\tilde{P}_t * H, \overline{wp}(\tilde{P}_c * H, \phi)) \wedge \overline{wp}(\tilde{Q}_t * G, \overline{wp}(\tilde{Q}_c * G, \psi)) \\
& \vee \;\; \overline{wp}(\tilde{P}_t * H, \overline{wp}(\tilde{P}_c * H, \mathit{false})) \wedge \overline{wp}(\tilde{Q}_t * G, \overline{wp}(\tilde{Q}_c * G, \mathit{true})) \\
& \vee \;\; \overline{wp}(\tilde{P}_t * H, \overline{wp}(\tilde{P}_c * H, \mathit{true})) \wedge \overline{wp}(\tilde{Q}_t * G, \overline{wp}(\tilde{Q}_c * G, \mathit{false})) \\
& \vee \;\; (\vee\, s \leq t \bullet \overline{wp}(\tilde{P}_s * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{true}) \\
& \qquad\qquad \vee \;\; \overline{wp}(\tilde{P}_s * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{false})) \qquad \textit{induction hypothesis} \\
\equiv & \overline{wp}(\tilde{P}_{t\langle c \rangle} * H, \phi) \wedge \overline{wp}(\tilde{Q}_{t\langle c \rangle} * G, \psi) \\
& \vee \;\; \overline{wp}(\tilde{P}_{t\langle c \rangle} * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_{t\langle c \rangle} * G, \mathit{true}) \\
& \vee \;\; \overline{wp}(\tilde{P}_{t\langle c \rangle} * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_{t\langle c \rangle} * G, \mathit{false}) \\
& \vee \;\; (\vee\, s \leq t \bullet \overline{wp}(\tilde{P}_s * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{true}) \\
& \qquad\qquad \vee \;\; \overline{wp}(\tilde{P}_s * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{false})) \\
\equiv & \overline{wp}(\tilde{P}_{t\langle c \rangle} * H, \phi) \wedge \overline{wp}(\tilde{Q}_{t\langle c \rangle} * G, \psi) \\
& \vee \;\; (\vee\, s \leq t\langle c \rangle \bullet \overline{wp}(\tilde{P}_s * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{true}) \\
& \qquad\qquad \vee \;\; \overline{wp}(\tilde{P}_s * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{false})).
\end{array}
$$

Proof of Lemma 5.13

Let

$$
\mathrm{R} = \overline{wp}_\infty(R_u * I, \_), \quad \mathrm{P} = \overline{wp}_\infty(\tilde{P}_u * H, \_), \quad \mathrm{Q} = \overline{wp}_\infty(\tilde{Q}_u * G, \_),
$$

so that for suitable ordinal α,

$$
\overline{\mathit{inf}}(R_u * I, \_) \equiv (\mathrm{R}^\alpha)_0, \quad \overline{\mathit{inf}}(\tilde{P}_u * H, \_) \equiv (\mathrm{P}^\alpha)_0, \quad \overline{\mathit{inf}}(\tilde{Q}_u * G, \_) \equiv (\mathrm{Q}^\alpha)_0.
$$

Also, let $C \;\equiv\; \langle c_i \mid i \geq 0 \rangle$, where

$$
\begin{array}{lll}
c_i & \equiv & (\vee\, s < u[i..] \bullet \;\; \overline{wp}(\tilde{P}_s, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_s, \mathit{true}) \\
& & \qquad\qquad\qquad\; \vee \;\; \overline{wp}(\tilde{P}_s, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_s, \mathit{false}))
\end{array}
$$

Lemma C.9 *For* $i \in \mathbb{N}$, $\qquad \overline{wp}(R_{u_i} * I, c_{i+1}) \;\equiv\; c_i.$

Proof:

$$
\begin{array}{ll}
& \overline{wp}(R_{u_i} * I, c_{i+1}) \\
\equiv & (\vee\ s < u[i+1..] \bullet \\
& \quad \overline{wp}(R_{u_i} * I, \overline{wp}(\tilde{P}_s * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{true})) \\
& \quad \vee\ \ \overline{wp}(R_{u_i} * I, \overline{wp}(\tilde{P}_s * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{false}))) \qquad \textit{disjunction} \\
\equiv & (\vee\ s < u[i+1..] \bullet \\
& \quad \overline{wp}(\tilde{P}_{u_i} * H, \overline{wp}(\tilde{P}_s * H, \mathit{false})) \wedge \overline{wp}(\tilde{Q}_{u_i} * G, \overline{wp}(\tilde{Q}_s * G, \mathit{true})) \\
& \quad \vee\ \ \overline{wp}(\tilde{P}_{u_i} * H, \overline{wp}(\tilde{P}_s * H, \mathit{true})) \wedge \overline{wp}(\tilde{Q}_{u_i} * G, \overline{wp}(\tilde{Q}_s * G, \mathit{false})) \\
& \quad \vee\ \ \overline{wp}(\tilde{P}_{u_i} * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_{u_i} * G, \mathit{true}) \\
& \quad \vee\ \ \overline{wp}(\tilde{P}_{u_i} * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_{u_i} * G, \mathit{false})) \qquad \textit{Lemma 5.10} \\
\equiv & (\vee\ s < u[i..] \bullet \\
& \quad \overline{wp}(\tilde{P}_s * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{true}) \\
& \quad \vee\ \ \overline{wp}(\tilde{P}_s * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{false})) \\
\equiv & c_i
\end{array}
$$

□

Lemma C.10 *For $i \in \mathbb{N}$,*

$$\overline{wp}(R_{u_i} * I, (\ (\mathrm{P}^\alpha)_{i+1} \wedge (\mathrm{Q}^\alpha)_{i+1}\ ) \vee c_{i+1}) \quad \equiv \quad (\ (\mathrm{P}^\alpha)_i \wedge (\mathrm{Q}^\alpha)_i\ ) \vee c_i).$$

Proof:

$$
\begin{array}{ll}
& \overline{wp}(R_{u_i} * I, (\ (\mathrm{P}^\alpha)_{i+1} \wedge (\mathrm{Q}^\alpha)_{i+1}\ ) \vee c_{i+1}) \\
\equiv & \overline{wp}(R_{u_i} * I, (\mathrm{P}^\alpha)_{i+1} \wedge (\mathrm{Q}^\alpha)_{i+1}) \\
& \vee\ \ \overline{wp}(R_{u_i} * I, c_{i+1}) \qquad \textit{disjunction} \\
\equiv & \overline{wp}(\tilde{P}_{u_i} * H, (\mathrm{P}^\alpha)_{i+1}) \wedge \overline{wp}(\tilde{Q}_{u_i} * G, (\mathrm{Q}^\alpha)_{i+1}) \\
& \vee\ \ \overline{wp}(\tilde{P}_{u_i} * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_{u_i} * G, \mathit{true}) \\
& \vee\ \ \overline{wp}(\tilde{P}_{u_i} * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_{u_i} * G, \mathit{false}) \\
& \vee\ \ c_i \qquad \textit{Lemmas 5.10,C.9} \\
\equiv & (\mathrm{P}^{\alpha+1})_i \wedge (\mathrm{Q}^{\alpha+1})_i \\
& \vee\ \ \overline{wp}(\tilde{P}_{u_i} * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_{u_i} * G, \mathit{true}) \qquad \text{(iii)} \\
& \vee\ \ \overline{wp}(\tilde{P}_{u_i} * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_{u_i} * G, \mathit{false}) \qquad \text{(iv)} \\
& \vee\ \ c_i \\
\equiv & (\mathrm{P}^{\alpha+1})_i \wedge (\mathrm{Q}^{\alpha+1})_i \\
& \vee\ \ c_i \qquad \textit{since } \text{(iii)} \Rrightarrow c_i,\ \text{(iv)} \Rrightarrow c_i
\end{array}
$$

□

Lemma C.11 *For any ordinal α,*

$$\mathrm{R}(\ (\mathrm{P}^\alpha \wedge \mathrm{Q}^\alpha) \vee C) \quad \equiv \quad (\mathrm{P}^{\alpha+1} \wedge \mathrm{Q}^{\alpha+1}) \vee C.$$

Proof:

$$
\begin{array}{ll}
& \mathrm{R}(\ (\mathrm{P}^\alpha \wedge \mathrm{Q}^\alpha) \vee C) \\
\equiv & \overline{wp}_\infty(R_u * G, (\mathrm{P}^\alpha \wedge \mathrm{Q}^\alpha) \vee C) \\
\equiv & \langle\ \overline{wp}(R_{u_i} * G, (\ (\mathrm{P}^\alpha)_{i+1} \wedge (\mathrm{Q}^\alpha)_{i+1}\ ) \vee c_{i+1}) \mid i \geq 0\ \rangle \\
\equiv & \langle\ (\ (\mathrm{P}^{\alpha+1})_i \wedge (\mathrm{Q}^{\alpha+1})_i\ ) \vee c_i \mid i \geq 0\ \rangle \qquad \textit{Lemma C.10} \\
\equiv & (\mathrm{P}^{\alpha+1} \wedge \mathrm{Q}^{\alpha+1}) \vee C
\end{array}
$$

□

Lemma C.12 *For each ordinal* α, $\quad \mathrm{R}^{\alpha} \;\equiv\; (\mathrm{P}^{\alpha} \wedge \mathrm{Q}^{\alpha}) \vee C$

Proof by ordinal induction:

$$
\begin{array}{cll}
 & \mathrm{R}^{\alpha} & \\
\equiv & (\wedge\, \beta \mid \beta < \alpha \bullet \mathrm{R}(\mathrm{R}^{\beta})) & \\
\equiv & (\wedge\, \beta \mid \beta < \alpha \bullet \mathrm{R}(\, (\mathrm{P}^{\beta} \wedge \mathrm{Q}^{\beta}) \vee C)) & \textit{induction hypothesis} \\
\equiv & (\wedge\, \beta \mid \beta < \alpha \bullet (\mathrm{P}^{\beta+1} \wedge \mathrm{Q}^{\beta+1}) \vee C) & \textit{Lemma C.11} \\
\equiv & (\wedge\, \beta \mid \beta < \alpha \bullet \mathrm{P}^{\beta+1} \wedge \mathrm{Q}^{\beta+1}) \vee C & C \textit{ independent of } \beta \\
\equiv & (\, (\wedge\, \beta \mid \beta < \alpha \bullet \mathrm{P}^{\beta+1}) \wedge (\wedge\, \beta \mid \beta < \alpha \bullet \mathrm{Q}^{\beta+1})\,) \vee C & \textit{pred. calc.} \\
\equiv & (\mathrm{P}^{\alpha} \wedge \mathrm{Q}^{\alpha}) \vee C &
\end{array}
$$

□

Lemma 5.13 *For action trace* $u \in C_\iota^{\omega}$,

$$
\begin{array}{rcl}
\overline{\mathit{inf}}(R_u * I) & \equiv & \overline{\mathit{inf}}(\tilde{P}_u * H) \wedge \overline{\mathit{inf}}(\tilde{Q}_u * G) \\
 & & \vee \;\; (\vee\, s < u \bullet \;\; \overline{wp}(\tilde{P}_s * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{true}) \\
 & & \qquad\qquad\qquad \vee \;\; \overline{wp}(\tilde{P}_s * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{false})).
\end{array}
$$

Proof:

$$
\begin{array}{cll}
 & \overline{\mathit{inf}}(R_u * I) & \\
\equiv & (\mathrm{R}^{\alpha})_0 & \\
\equiv & (\, (\mathrm{P}^{\alpha})_0 \wedge (\mathrm{Q}^{\alpha})_0\,) \vee c_0 & \textit{Lemma C.12} \\
\equiv & \overline{\mathit{inf}}(\tilde{P}_u * H) \wedge \overline{\mathit{inf}}(\tilde{Q}_u * G) & \\
 & \vee \;\; (\vee\, s < u \bullet \;\; \overline{wp}(\tilde{P}_s * H, \mathit{false}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{true}) & \\
 & \qquad\qquad\qquad \vee \;\; \overline{wp}(\tilde{P}_s * H, \mathit{true}) \wedge \overline{wp}(\tilde{Q}_s * G, \mathit{false})). &
\end{array}
$$

Proof of Lemma 5.14

Firstly we have lemmas as follows:

Lemma C.13 $\mathit{halt}(IT_I) \;\equiv\; \mathit{halt}(IT_H) \wedge \mathit{halt}(IT_G)$.

Proof:

$$
\begin{array}{cll}
 & \mathit{halt}(IT_I) & \\
\equiv & \mathit{halt}(IT_H \parallel IT_G) & \textit{Lemma 5.7} \\
\equiv & \mathit{gd}(IT_H) \wedge \mathit{gd}(IT_G) \Rightarrow \mathit{halt}(IT_H) \wedge \mathit{halt}(IT_G) & \textit{Property 5.1} \\
\equiv & \mathit{halt}(IT_H) \wedge \mathit{halt}(IT_G) & \textit{since } \mathit{gd}(IT) \equiv \mathit{true}
\end{array}
$$

□

Lemma C.14 *For v-predicate* ϕ, $\quad wp(IT_I, \phi) \equiv \mathit{halt}(IT_I) \wedge wp(IT_H, \phi)$.

Proof:

$$\begin{array}{lll}
& wp(IT_I, \phi) & \\
\equiv & wp(IT_H, \phi) \wedge halt(IT_G) & \textit{Lemma 5.7, } gd(IT) \equiv true \\
\equiv & wp(IT_H, \phi) \wedge halt(IT_H) \wedge halt(IT_G) & \textit{since } wp(IT, \phi) \Rrightarrow halt(IT) \\
\equiv & wp(IT_H, \phi) \wedge halt(IT_I) & \textit{Lemma C.13}
\end{array}$$

□

Lemma C.15 *For $Z \subseteq (A - B)$,*

$$halt(IT_I) \;\wedge\; \neg\, gd_I(R_Z) \;\equiv\; halt(IT_I) \;\wedge\; \neg\, gd_H(P_Z).$$

Proof:

$$\begin{array}{lll}
& halt(IT_I) \;\wedge\; \neg\, gd_I(R_Z) & \\
\equiv & halt(IT_I) \;\wedge\; (\wedge\, c \in Z \bullet wp(IT_I, \neg\, gd(R_c))) & \textit{Definition 4.7} \\
\equiv & halt(IT_I) \;\wedge\; (\wedge\, c \in Z \bullet wp(IT_I, \neg\, gd(P_c))) & \textit{since } c \notin B \\
\equiv & halt(IT_I) \;\wedge\; (\wedge\, c \in Z \bullet wp(IT_H, \neg\, gd(P_c))) & \textit{Lemma C.14} \\
\equiv & halt(IT_I) \;\wedge\; \neg\, gd_H(P_Z) & \textit{Definition 4.7}
\end{array}$$

□

Similarly we have:

Lemma C.16 *For $Z \subseteq (B - A)$,*

$$halt(IT_I) \;\wedge\; \neg\, gd_I(R_Z) \;\equiv\; halt(IT_I) \;\wedge\; \neg\, gd_G(Q_Z)$$

□

Lemma C.17 *For set Z, predicates ϕ_i, ψ_i,*

$$(\wedge\, i \in Z \bullet \phi_i \vee \psi_i) \;\equiv\; (\vee\, X, Y \subseteq Z \mid X \cup Y = Z \bullet (\wedge\, i \in X \bullet \phi_i) \wedge (\wedge\, i \in Y \bullet \psi_i)).$$

Proof by mutual entailment:

(a). Assume $(\wedge\, i \in Z \bullet \phi_i \vee \psi_i)$. Let $X \mathrel{\widehat{=}} \{i \in Z \mid \phi_i\}$, $Y \mathrel{\widehat{=}} \{i \in Z \mid \psi_i\}$. Then

$$\begin{array}{lll}
X \cup Y & = & \{i \in Z \mid \phi_i \vee \psi_i\} \\
& = & Z \qquad \textit{since } \phi_i \vee \psi_i\textit{, each } i \in Z
\end{array}$$

hence $(\vee\, X, Y \subseteq Z \mid X \cup Y = Z \bullet (\wedge\, i \in X \bullet \phi_i) \wedge (\wedge\, i \in Y \bullet \psi_i))$

(b).

$$\begin{array}{ll}
& (\vee\, X, Y \subseteq Z \mid X \cup Y = Z \bullet (\wedge\, i \in X \bullet \phi_i) \wedge (\wedge\, i \in Y \bullet \psi_i)) \\
\Rrightarrow & (\vee\, X, Y \subseteq Z \mid X \cup Y = Z \bullet (\wedge\, i \in X \bullet \phi_i \vee \psi_i) \wedge (\wedge\, i \in Y \bullet \phi_i \vee \psi_i)) \\
\Rrightarrow & (\vee\, X, Y \subseteq Z \mid X \cup Y = Z \bullet (\wedge\, i \in (X \cup Y) \bullet \phi_i \vee \psi_i)) \\
\Rrightarrow & (\wedge\, i \in Z \bullet \phi_i \vee \psi_i)
\end{array}$$

□

Lemma C.18 *For $Z \subseteq (A \cap B)$,*

$$halt(IT_I) \ \wedge \ \neg\, gd_I(R_Z) \ \equiv$$
$$halt(IT_I) \ \wedge \ (\vee\, X, Y \subseteq Z \mid X \cup Y = Z \bullet \neg\, gd_H(P_X) \wedge \neg\, gd_G(Q_Y)).$$

Proof:

$$
\begin{array}{lll}
 & halt(IT_I) \ \wedge \ \neg\, gd_I(R_Z) & \\
\equiv & halt(IT_I) \ \wedge \ (\wedge\, c \in Z \bullet wp(IT_I, \neg\, gd(R_c))) & \textit{Definition 4.7} \\
\equiv & halt(IT_I) \ \wedge \ (\wedge\, c \in Z \bullet wp(IT_H \parallel IT_G, \neg\, gd(P_c) \vee \neg\, gd(Q_c))) & \textit{since } c \in A \cap B \\
\equiv & halt(IT_I) \ \wedge \ (\wedge\, c \in Z \bullet wp(IT_H, \neg\, gd(P_c)) & \\
 & \qquad\qquad \vee\, wp(IT_G, \neg\, gd(Q_c))) & \textit{Property 5.1, Lemma C.13} \\
\equiv & halt(IT_I) \ \wedge \ (\vee\, X, Y \subseteq Z \mid X \cup Y = Z \bullet & \\
 & \quad (\wedge\, c \in X \bullet wp(IT_H, \neg\, gd(P_c))) & \\
 & \quad \wedge \ (\wedge\, c \in Y \bullet wp(IT_G, \neg\, gd(Q_c)))) & \textit{Lemma C.17} \\
\equiv & halt(IT_I) \ \wedge \ (\vee\, X, Y \subseteq Z \mid X \cup Y = Z \bullet & \\
 & \quad \neg\, gd_H(P_X) \wedge \neg\, gd_G(Q_Y)) & \textit{Definition 4.7}
\end{array}
$$

□

Lemma C.19 *For $Z \subseteq C$,*

$$halt(IT_I) \ \wedge \ \neg\, gd_I(R_Z) \ \equiv$$
$$halt(IT_I) \ \wedge \ (\vee\, X \subseteq A, Y \subseteq B \mid X \cup Y = Z \bullet \neg\, gd_H(P_X) \wedge \neg\, gd_G(Q_Y)).$$

Proof:

$$
\begin{array}{lll}
 & halt(IT_I) \ \wedge \ \neg\, gd_I(R_Z) & \\
\equiv & halt(IT_I) \ \wedge \ \neg\, gd_I(R_{Z-B}) \wedge \neg\, gd_I(R_{Z-A}) \wedge \neg\, gd_I(R_{Z \cap A \cap B}) & \textit{conjunction} \\
\equiv & halt(IT_I) \wedge \neg\, gd_H(P_{Z-B}) \wedge \neg\, gd_G(Q_{Z-A}) \wedge & \\
 & (\vee\, X, Y \subseteq Z \mid X \cup Y = Z \cap A \cap B \bullet & \\
 & \qquad \neg\, gd_H(P_X) \wedge \neg\, gd_G(Q_Y)) & \textit{C.15, C.16, C.18} \\
\equiv & halt(IT_I) \wedge & \\
 & (\vee\, X, Y \subseteq Z \mid X \cup Y = Z \cap A \cap B \bullet & \\
 & \qquad \neg\, gd_H(P_{X \cup (Z-B)}) \wedge \neg\, gd_G(Q_{Y \cup (Z-A)})) & \textit{conjunction} \\
\equiv & halt(IT_I) \wedge & \\
 & (\vee\, X, Y \subseteq Z \mid X \cup Y = Z \cap A \cap B \bullet & \\
 & \quad (\vee\, X' \subseteq A, Y' \subseteq B \mid X' = X \cup (Z - B) \ \wedge \ Y' = Y \cup (Z - A) \bullet & \\
 & \qquad \neg\, gd_H(P_{X'}) \wedge \neg\, gd_G(Q_{Y'}))) & \\
\equiv & halt(IT_I) \wedge & \\
 & (\vee\, X' \subseteq A, Y' \subseteq B \mid X' \cup Y' = (Z \cap A \cap B) \cup (Z - B) \cup (Z - A) \bullet & \\
 & \qquad \neg\, gd_H(P_{X'}) \wedge \neg\, gd_G(Q_{Y'})) & \\
\equiv & halt(IT_I) \wedge & \\
 & (\vee\, X' \subseteq A, Y' \subseteq B \mid X' \cup Y' = Z \bullet & \\
 & \qquad \neg\, gd_H(P_{X'}) \wedge \neg\, gd_G(Q_{Y'})) &
\end{array}
$$

□

Lemma C.20 *For any action S, predicate ϕ,*

$$\overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \phi) \;\equiv\; \overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \mathit{halt}(\mathbf{it}\ S\ \mathbf{ti}) \wedge \phi).$$

Proof: We have $LHS \Leftarrow RHS$ by monotonicity. It is easy to show that $\overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \phi)$ is a solution of

$$X \;\Rrightarrow\; \mathit{halt}(\mathbf{it}\ S\ \mathbf{ti}) \wedge \phi \;\vee\; \overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, X) \tag{v}$$

and since $\overline{wp}(\mathbf{it}\ S\ \mathbf{ti}, \mathit{halt}(\mathbf{it}\ S\ \mathbf{ti}) \wedge \phi)$ is the greatest solution of (v), $LHS \Rrightarrow RHS$.
□

Lemma 5.14 *For $Z \subseteq C$,*

$$\begin{aligned}\overline{wp}(IT_I, \neg\, gd_I(R_Z)) \;\equiv\; & (\vee\, X \subseteq A, Y \subseteq B \mid X \cup Y = Z \bullet \\ & \quad \overline{wp}(IT_I, \neg\, gd_H(P_X) \wedge \neg\, gd_G(Q_Y))).\end{aligned}$$

Proof:

$$\begin{array}{ll}
& \overline{wp}(IT_I, \neg\, gd_I(R_Z)) \\
\equiv & \overline{wp}(IT_I, \mathit{halt}(IT_I) \wedge \neg\, gd_I(R_Z)) \qquad\qquad \textit{Lemma C.20} \\
\equiv & \overline{wp}(IT_I, \mathit{halt}(IT_I) \wedge \\
& \qquad (\vee\, X \subseteq A, Y \subseteq B \mid X \cup Y = Z \bullet \\
& \qquad\quad \neg\, gd_H(P_X) \wedge \neg\, gd_G(Q_Y))) \qquad\qquad \textit{Lemma C.19} \\
\equiv & (\vee\, X \subseteq A, Y \subseteq B \mid X \cup Y = Z \bullet \\
& \quad \overline{wp}(IT_I, \neg\, gd_H(P_X) \wedge \neg\, gd_G(Q_Y))) \qquad \textit{disjunction, Lemma C.20.}
\end{array}$$

Appendix D

Proofs for Chapter 6

Proof of Theorem 6.8

Given $P = (A, V, P_\imath, P_A, P_H)$, construct P^+ and show that $\{\!\!\{P\}\!\!\} = \{\!\!\{P^+\}\!\!\}$. Before constructing P^+, we present some definitions. For statement S, $S[\mathbf{fix}\ v]$ behaves as S on all variables except v which it leaves unchanged:

Definition D.1 *$wp(S[\mathbf{fix}\ v], \phi) \ \widehat{=}\ wp(S[v\backslash l], \phi)[l\backslash v]$, where l is some fresh variable independent of v, S and ϕ.*

For example, $(v, y := E, F)[\mathbf{fix}\ v] \ = \ y := F$. For statement S, $\mathit{feasible}(S)$ behaves as S in those states in which S is enabled, otherwise it behaves as **skip**:

Definition D.2 *$\mathit{feasible}(S) \ \widehat{=}\ S \mathbin{[\!]} [\neg\, gd(S)]$.*

Note that $\mathit{feasible}(S)$ is always non-miraculous.

For each output statement P_a of P, construct a statement CHS_a as follows:

$$CHS_a \ \widehat{=}\ \mathit{feasible}(P_a[y\backslash y_a][\mathbf{fix}\ v]).$$

CHS_a assigns to y_a what P_a would assign to y, and leaves v unchanged. Because $halt(P_a) \equiv \mathit{true}$ (see Theorem 6.8), CHS_a always terminates. For example, if

$$P_a \ = \ (v \neq \langle\rangle \rightarrow y := hd(v)\ ;\ v := tl(v))$$

then $CHS_a = (v \neq \langle\rangle \rightarrow y_a := hd(v) \mathbin{[\!]} v = \langle\rangle \rightarrow \mathbf{skip})$.

We introduce a generalised parallel operator over a possibly infinite set of predicates. Generalised composition over the set $S_i, i \in I$ is written $(\| \ i \in I \bullet S_i)$, and satisfies the following property:

Property D.3 *Let $\{\ v_i \mid i \in I\ \}$ be a set of distinct variables. For each $i \in I$, let S_i be a v_i-statement, and ϕ_i be a v_i-predicate. Then the generalised parallel operator for statements satisfies:*

$$\begin{array}{l} wp(\ (\|\ i \in I \bullet S_i),\ (\vee\ i \in I \bullet \phi_i)\ ) \\ \quad \equiv \ \ (\wedge\ i \in I \bullet gd(S_i)) \ \Rightarrow\ (\wedge\ i \in I \bullet halt(S_i)) \wedge (\vee\ j \in I \bullet wp(S_j, \phi_j)) \end{array}$$

□

Note: For infinite I, $(\| \; i \in I \bullet S_i)$ cannot be constructed using sequential composition. However, we can define an operator on specification statements that satisfies Property D.3:

$$(\| \; i \bullet v_i : [\; pre_i, \; post_i \;]) \quad \widehat{=} \quad (\cup i \bullet v_i) : [\; (\wedge \; i \bullet pre_i), \; (\wedge \; i \bullet post_i) \;].$$

Using the "predicate correspondence" (cf. construction of SP in Lemma A.2), this operator can be applied to any set of statements.

Construct the statement CHS as follows:

$$CHS \quad \widehat{=} \quad (\| \; a \in OUT \bullet CHS_a).$$

Now construct P^+ as follows:

$$P^+ \quad \widehat{=} \quad (A_{\mathcal{W}}, \; v \cup \{\; y_a \mid a \in OUT \;\}, \; P_\imath^+, \; \{P_{a.i}^+ \mid a.i \in A_{\mathcal{W}}\}, \; \{P_h^+ \mid h \in H\})$$

where

$$\begin{array}{lcll} P_\imath^+ & \widehat{=} & P_\imath \; ; \; CHS, & \\ P_{a.i}^+ & \widehat{=} & P_a[\mathbf{value}\; x \backslash i] \; ; \; CHS, & \quad \textit{if } dir(a) = in, \\ P_{a.i}^+ & \widehat{=} & y_a = i \rightarrow (\mathbf{var}\; y \bullet P_a[y = i]) \; ; \; CHS, & \quad \textit{if } dir(a) = out, \\ P_h^+ & \widehat{=} & P_h \; ; \; CHS. & \end{array}$$

P^+ is a normal action system with non-parameterised actions and by Theorem 4.9, $\{\!\![P]\!\!\}$ is well-formed. Before showing that $\{\!\![P]\!\!\} = \{\!\![P^+]\!\!\}$, we present some lemmas about CHS:

Lemma D.4 $CHS \; ; \; CHS \;=\; CHS.$

This is a consequence of the fact that each output action is independent of the initial value of y (Condition 2 of theorem). The following disjunctivity results follow from Property D.3:

Lemma D.5 *Let ϕ_a represent any predicate that may depend on y_a, but is independent of each $y_{a'}$, for $a' \neq a$. Then for $X \subseteq OUT$,*

$$wp(CHS, (\vee \; a \in X \bullet \phi_a)) \quad \equiv \quad (\vee \; a \in X \bullet wp(CHS_a, \phi_a))$$

□

Lemma D.6 *For ψ independent of each y_a, any ϕ,*

$$wp(CHS, \phi \vee \psi) \quad \equiv \quad wp(CHS_a, \phi) \vee \psi$$

□

For $a.i \in A_{\mathcal{W}}$, $P_{a.i}$ is defined by Definitions 6.1 and 6.3. The following lemma gives a relationship between $P_{a.i}$ and $P_{a.i}^+$:

Lemma D.7 *For v-predicate ϕ,*

$$\overline{wp}(P_{a.i}, \phi) \quad \equiv \quad \overline{wp}(CHS \; ; \; P_{a.i}^+, \phi).$$

Proof: If $dir(a) = in$ then proof is trivial. If $dir(a) = out$ then,

$$\begin{array}{lll} & \overline{wp}(CHS \; ; \; P^+_{a.i}, \phi) & \\ \equiv & \overline{wp}(CHS, y_a = i \wedge \overline{wp}(P_a, y = i \wedge \phi)) & \textit{definition of } P^+_{a.i} \\ \equiv & \overline{wp}(CHS_a, y_a = i \wedge \overline{wp}(P_a, y = i \wedge \phi)) & \textit{Lemma D.5} \\ \equiv & \overline{wp}(CHS_a, y_a = i) \wedge \overline{wp}(P_a, y = i \wedge \phi) & \textit{Lemma D.6} \\ \equiv & \overline{wp}(P_a, y = i) \wedge \overline{wp}(P_a, y = i \wedge \phi) & \textit{definition of } CHS_a \\ \equiv & \overline{wp}(P_{a.i}, \phi) & \textit{definition of } CHS_a \end{array}$$

□

Because $CHS \; ; \; CHS = CHS$ and each action of P^+ (visible and internal) ends in CHS, it is easy to prove the following lemma by induction on traces:

Lemma D.8 *For v-predicate ϕ, $s \in A_{\mathcal{W}}{}^*$,*

$$\overline{wp}(P^+_{\langle i \rangle s} * H^+, \phi) \;\; \equiv \;\; \overline{wp}(P_{\langle i \rangle s} * H, \phi)$$

□

Similarly, it is easy to prove the following lemma using a fixed-point argument:

Lemma D.9 *For $u \in A_{\mathcal{W}}{}^\omega$,*

$$\overline{inf}(P^+_{\langle i \rangle s} * H^+) \;\; \equiv \;\; \overline{inf}(P_{\langle i \rangle s} * H)$$

□

Lemma D.8 allows us to compare the effect of finite traces of communication events on P and on P^+, and Lemma D.9 allows to compare the infinite traces of P and of P^+. So to show that $\{\!|P|\!\} = \{\!|P^+|\!\}$ we only need to be able to compare the conditions under which P and P^+ may refuse sets of communications. This will be provided by Lemma D.23, but before giving this lemma, we present a series of lemmas leading to it.

Lemma D.10 *For $S \neq \{\}$, $a \in OUT$,*

$$commgd(P_{a.S}) \;\Rrightarrow\; (\vee \; i \in S \bullet \overline{wp}(P_a, y = i)).$$

Proof: From Theorem 2.8 we get $gd(P_a) \wedge wp(P_a, y \in S) \Rrightarrow \overline{wp}(P_a, y \in S)$, and lemma follows by disjunction.

□

Lemma D.11 *For $S \neq \{\}$, $a \in OUT$,*

$$commgd(P_{a.S}) \;\;\Rrightarrow\;\; (\wedge \; i \in S' \bullet \overline{wp}(P_a, y = i)) \;\wedge\; wp(P_a, y \in S')$$

where S' is some non-empty subset of S.

Proof: Assume $commgd(P_{a.S})$, then

$$\begin{array}{ll} & true \\ \equiv & (\wedge \; i \in S \bullet \overline{wp}(P_a, y = i) \vee wp(P_a, y \neq i)) \\ \Rrightarrow & (\wedge \; i \in S \bullet \overline{wp}(P_a, y = i) \vee wp(P_a, y \neq i)) \;\wedge\; (\vee \; i \in S \bullet \overline{wp}(P_a, y = i)) \\ & \hfill \textit{Lemma D.10} \\ \Rrightarrow & (\wedge \; i \in S' \bullet \overline{wp}(P_a, y = i)) \;\wedge\; (\wedge \; i \in (S - S') \bullet wp(P_a, y \neq i)) \\ \Rrightarrow & (\wedge \; i \in S' \bullet \overline{wp}(P_a, y = i)) \;\wedge\; (\wedge \; i \in (S - S') \bullet wp(P_a, y \neq i)) \;\wedge\; wp(P_a, y \in S) \\ & \hfill \textit{since } commgd(P_{a.S}) \Rrightarrow wp(P_a, y \in S) \\ \Rrightarrow & (\wedge \; i \in S' \bullet \overline{wp}(P_a, y = i)) \;\wedge\; wp(P_a, y \in S') \end{array}$$

□

Lemma D.12 *For* $a \in OUT$, $\quad wp(CHS_a, gd(P^+_{a.S})) \equiv commgd(P_{a.S})$.

Proof: For $S = \{\}$, proof is trivial. For $S \neq \{\}$, by mutual entailment:
(a).

$$
\begin{array}{lll}
 & wp(CHS_a, gd(P^+_{a.S})) & \\
\Rrightarrow & wp(CHS_a, (\vee\ i \in S \bullet y_a = i \wedge \overline{wp}(P_a, y = i))) & \textit{definition of } P^+_{a.i} \\
\Rrightarrow & wp(CHS_a, (\vee\ i \in S \bullet y_a = i) \wedge gd(P_a)) & \\
\Rrightarrow & wp(CHS_a, y_a \in S \wedge gd(P_a)) & \\
\Rrightarrow & wp(P_a, y \in S) \wedge gd(P_a) & \textit{definition of } CHS_a \\
\Rrightarrow & commgd(P_{a.S}) &
\end{array}
$$

(b).

$$
\begin{array}{lll}
 & commgd(P_{a.S}) & \\
\Rrightarrow & (\wedge\ i \in S' \bullet \overline{wp}(P_a, y = i)) \wedge wp(P_a, y \in S') & \textit{Lemma D.11} \\
\Rrightarrow & (\wedge\ i \in S' \bullet \overline{wp}(P_a, y = i)) \wedge wp(CHS_a, y_a \in S') & \textit{definition of } CHS_a \\
\Rrightarrow & wp(CHS_a, y_a \in S' \wedge (\wedge\ i \in S' \bullet \overline{wp}(P_a, y = i))) & \textit{Lemma D.6} \\
\Rrightarrow & wp(CHS_a, (\vee\ i \in S \bullet y_a = i \wedge \overline{wp}(P_a, y = i))) & S' \textit{ non-empty} \\
\Rrightarrow & wp(CHS_a, gd(P^+_{a.S})) & \textit{definition of } P^+_{a.i}
\end{array}
$$

□

Lemma D.13 *For* $a \in \{a \in A \mid dir(a) = in\}$, $\quad gd(P^+_{a.S}) \equiv commgd(P_{a.S})$.

Proof: Follows easily from definition of P^+ and $commgd$.

Let $P_H = ([]\ h \in H \bullet P_h)$, $P^+_H \mathrel{\widehat{=}} ([]\ h \in H \bullet P^+_h)$, $IT_H \mathrel{\widehat{=}}$ **it** P_H **ti**, $IT^+_H \mathrel{\widehat{=}}$ **it** P^+_H **ti**. The following two lemmas are true since CHS is independent of any previous execution of CHS:

Lemma D.14 $CHS\ ;\ IT_{H+} \ = \ IT_H\ ;\ CHS$.

Lemma D.15 $P^+_H\ ;\ IT_{H+} \ = \ P_H\ ;\ IT_H\ ;\ CHS$.

We also have

Lemma D.16 *For v-predicate* ϕ, $\quad wp(IT^+_H, \phi) \equiv wp(IT_H, \phi)$.

Lemma D.17 *For* $X \subseteq A_{\mathcal{W}}$, $\quad gd_H(P^+_X) \Rrightarrow gd(P^+_X) \vee gd(P_H)$.

Proof:

$$
\begin{array}{lll}
 & gd_H(P^+_X) & \\
\Rrightarrow & (\vee\ a.i \in X \bullet \overline{wp}(IT^+_H, gd(P^+_{a.i}))) & \textit{Definition 4.7} \\
\Rrightarrow & (\vee\ a.i \in X \bullet gd(P^+_{a.i}) \vee \overline{wp}(P^+_H\ ;\ IT^+_H, gd(P^+_{a.i}))) & \textit{property of } \textbf{it}\ S\ \textbf{ti} \\
\Rrightarrow & (\vee\ a.i \in X \bullet gd(P^+_{a.i})) \vee gd(P^+_H) & \overline{wp}(s, \phi) \Rrightarrow gd(S) \\
\Rrightarrow & gd(P^+_X) \vee gd(P_H) & gd(P^+_H) \equiv gd(P_H)
\end{array}
$$

□

Lemma D.18 *For $X \subseteq A_{\mathcal{W}}$,* $wp(CHS, gd_{H^+}(P_X^+)) \Rrightarrow commgd(P_X) \vee gd(P_H)$.

Proof:

$$
\begin{array}{ll}
 & wp(CHS, gd_{H^+}(P_X^+)) \\
\Rrightarrow & wp(CHS, gd(P_X^+) \vee gd(P_H)) \qquad \textit{Lemma D.17} \\
\Rrightarrow & wp(CHS, gd(P_X^+)) \vee gd(P_H) \\
\Rrightarrow & wp(CHS, (\vee\ a \in chan(\!|X|\!) \mid dir(a) = in \bullet gd(P^+_{X\restriction a})) \\
 & \qquad \vee (\vee\ a \in chan(\!|X|\!) \mid dir(a) = out \bullet gd(P^+_{X\restriction a}))) \ \vee \ gd(P_H) \\
\Rrightarrow & (\vee\ a \in chan(\!|X|\!) \mid dir(a) = in \bullet gd(P^+_{X\restriction a})) \\
 & \vee (\vee\ a \in chan(\!|X|\!) \mid dir(a) = out \bullet wp(CHS_a, gd(P^+_{X\restriction a}))) \ \vee \ gd(P_H) \\
 & \qquad \textit{Lemma D.5} \\
\Rrightarrow & (\vee\ a \in chan(\!|X|\!) \mid dir(a) = in \bullet commgd(P_{X\restriction a})) \\
 & \vee (\vee\ a \in chan(\!|X|\!) \mid dir(a) = out \bullet commgd(P_{X\restriction a})) \ \vee \ gd(P_H) \\
 & \qquad \textit{Lemmas D.12 and D.13} \\
\Rrightarrow & commgd(P_X) \vee gd(P_H)
\end{array}
$$

□

Lemma D.19 *For $X \neq \{\}$,*

$$wp(IT_H \ ; \ CHS, \ gd_{H^+}(P_X^+)) \ \Rrightarrow \ commgd_H(P_X).$$

Proof:

$$
\begin{array}{lll}
 & wp(IT_H, wp(CHS, gd_{H^+}(P_X^+))) & \\
\Rrightarrow & wp(IT_H, commgd(P_X) \vee gd(P_H)) & \textit{Lemma D.18} \\
\Rrightarrow & \overline{wp}(IT_H, commgd(P_X)) & \textit{Theorem 4.22} \\
\Rrightarrow & commgd_H(P_X) & \textit{disjunction}
\end{array}
$$

□

Lemma D.20 *For $X \subseteq A_{\mathcal{W}}$,*

$$wp(IT_H \ ; \ CHS, gd_{H^+}(P_X^+)) \ \Rrightarrow \ wp(IT_H, commgd_H(P_X)).$$

Proof: For $X = \{\}$, proof is trivial. For $X \neq \{\}$ we have:

$$
\begin{array}{lll}
 & wp(IT_H \ ; \ CHS, gd_{H^+}(P_X^+))) & \\
\Rrightarrow & wp(IT_H \ ; \ IT_H \ ; \ CHS, gd_{H^+}(P_X^+))) & IT_H \ ; \ IT_H = IT_H \\
\Rrightarrow & wp(IT_H, commgd_H(P_X)) & \textit{Lemma D.19}
\end{array}
$$

□

Lemma D.21 *For $a \in OUT$,*

$$\overline{wp}(IT_H, commgd(P_{a.S})) \Rrightarrow wp(CHS_a, \overline{wp}(IT_H^+, gd(P^+_{a.S})).$$

Proof:

$$
\begin{array}{lll}
& \overline{wp}(IT_H, commgd(P_{a.S})) & \\
\Rrightarrow & \overline{wp}(IT_H, wp(CHS_a, gd(P^+_{a.S}))) & \textit{Lemma D.12} \\
\Rrightarrow & wp(CHS_a, gd(P^+_{a.S})) \vee \overline{wp}(P_H \ ; \ IT_H, wp(CHS_a, \ gd(P^+_{a.S}))) & \\
& & \textit{property of } \mathbf{it} \ S \ \mathbf{ti} \\
\Rrightarrow & wp(CHS_a, gd(P^+_{a.S})) \vee \overline{wp}(P_H \ ; \ IT_H \ ; \ CHS, \ gd(P^+_{a.S})) & \textit{Lemma D.5} \\
\Rrightarrow & wp(CHS_a, gd(P^+_{a.S})) \vee \overline{wp}(P^+_H \ ; \ IT^+_H, \ gd(P^+_{a.S})) & \textit{Lemma D.15} \\
\Rrightarrow & wp(CHS_a, gd(P^+_{a.S}) \vee \overline{wp}(P^+_H \ ; \ IT^+_H, \ gd(P^+_{a.S}))) & \textit{Lemma D.6} \\
\Rrightarrow & wp(CHS_a, \overline{wp}(IT^+_H, \ gd(P^+_{a.S}))) &
\end{array}
$$

□

Lemma D.22 $commgd_H(P_X) \Rrightarrow wp(CHS, gd_{H^+}(P^+_X))$.

Proof:

$$
\begin{array}{ll}
& commgd_H(P_X) \\
\Rrightarrow & (\vee \ a \in chan(\!|X|\!) \mid dir(a) = in \bullet \overline{wp}(IT_H, commgd(P_{X\restriction a}))) \\
& \vee (\vee \ a \in chan(\!|X|\!) \mid dir(a) = out \bullet \overline{wp}(IT_H, commgd(P_{X\restriction a}))) \\
& \hfill \textit{Definiton 6.6} \\
\Rrightarrow & (\vee \ a \in chan(\!|X|\!) \mid dir(a) = in \bullet \overline{wp}(IT^+_H, gd(P^+_{X\restriction a}))) \\
& \vee (\vee \ a \in chan(\!|X|\!) \mid dir(a) = out \bullet wp(CHS_a, \overline{wp}(IT^+_H, gd(P^+_{X\restriction a})))) \\
& \hfill \textit{Lemmas D.16, D.13, D.21} \\
\Rrightarrow & wp(CHS, (\vee \ a \in chan(\!|X|\!) \mid dir(a) = in \bullet \overline{wp}(IT^+_H, gd(P^+_{X\restriction a}))) \\
& \quad \vee (\vee \ a \in chan(\!|X|\!) \mid dir(a) = out \bullet \overline{wp}(IT^+_H, gd(P^+_{X\restriction a})))) \\
& \hfill \textit{Lemma D.5} \\
\Rrightarrow & wp(CHS, gd_{H^+}(P^+_X))
\end{array}
$$

□

Lemma D.23 $wp(CHS \ ; \ IT^+_H, \ gd_{H^+}(P^+_X)) \ \equiv \ wp(IT_H, commgd_H(P_X))$.

Proof: By Lemma D.14 only need to show

$$wp(IT_H \ ; \ CHS, \ gd_{H^+}(P^+_X)) \ \equiv \ wp(IT_H, commgd_H(P_X))$$

and this follows from Lemma D.20, and Lemma D.22 and monotonicity.

□

Finally to show that $[\![P]\!] = [\![P^+]\!]$, we have firstly failures equality: for $a \in A^*_{\mathcal{W}}$, $X \subseteq A_{\mathcal{W}}$,

$$
\begin{array}{lll}
& (s, X) \in \mathcal{F}[\![P]\!] & \\
\equiv & \overline{wp}(P_{\langle\imath\rangle s} * H, \neg \ commgd_H(P_X)) & \\
\equiv & \overline{wp}(P^+_{\langle\imath\rangle s} * H^+, \neg \ commgd_H(P_X)) & \textit{Lemma D.8} \\
\equiv & \overline{wp}(P^+_{\langle\imath\rangle s} * H^+, \neg \ gd_H(P^+_X)) & \textit{Lemma D.23} \\
\equiv & (s, X) \in \mathcal{F}[\![P^+]\!] &
\end{array}
$$

Similarly, divergences equality follows from Lemma D.8 with $\phi \equiv \mathit{false}$, and infinites equality follows from Lemma D.9.

Proof of Lemma 6.11

Lemma 6.11 *For each $a.i \in A_{\mathcal{W}}$,* $\quad P_{a.i} \preceq_{rep} Q_{a.i}$.

Proof:
Case $dir(a) = in$:

$$
\begin{array}{lll}
 & rep(wp(P_{a.i}, \phi)) & \\
\Rrightarrow & rep(wp(P_a, \phi)[x \backslash i]) & \textit{Definition 6.1} \\
\Rrightarrow & rep(wp(P_a, \phi))[x \backslash i] & (b) \textit{ of Theorem 6.10} \\
\Rrightarrow & wp(Q_a, rep(\phi))[x \backslash i] & \textit{since } P_a \preceq_{rep} Q_a \\
\Rrightarrow & wp(Q_{a.i}, rep(\phi)) & \textit{Definition 6.1}
\end{array}
$$

Case $dir(a) = out$:

$$
\begin{array}{lll}
 & rep(wp(P_{a.i}, \phi)) & \\
\Rrightarrow & rep(wp((\mathbf{var}\ y \bullet P_a[y = i]), \phi)) & \textit{Definition 6.3} \\
\Rrightarrow & rep(wp(P_a, y = i \Rightarrow \phi)) & wp(P_a, \phi) \textit{ independent of } y \\
\Rrightarrow & wp(Q_a, rep(y = i \Rightarrow \phi)) & \textit{since } P_a \preceq_{rep} Q_a \\
\Rrightarrow & wp(Q_a, y = i \Rightarrow rep(\phi)) & \textit{disjunction, } (c) \textit{ of Theorem 6.10} \\
\Rrightarrow & wp(Q_{a.i}, rep(\phi)) & \textit{Definition 6.3.}
\end{array}
$$

Proof of Lemma 6.12

Firstly we have lemmas as follows:

Lemma D.24 $rep(\exists\, x \in S \bullet \phi) \Rrightarrow (\exists\, x \in S \bullet rep(\phi))$.

Proof: From (a) and (b) of Theorem 6.10.

Lemma D.25 *For* $\{\} \subset S \subseteq \mathcal{W}$,

$$rep(\overline{wp}(IT_H, commgd(P_{a.S}))) \ \Rrightarrow\ \overline{wp}(IT_G, commgd(Q_{a.S})).$$

Proof: Case $dir(a) = in$:

$$
\begin{array}{lll}
 & rep(\overline{wp}(IT_H, commgd(P_{a.S}))) & \\
\Rrightarrow & rep(\overline{wp}(IT_H, (\exists\, x \in S \bullet gd(P_a))) & \textit{Definition 6.2} \\
\Rrightarrow & rep(\exists\, x \in S \bullet \overline{wp}(IT_H, gd(P_a))) & IT_H \textit{ ind. of } x\textit{,disjunction} \\
\Rrightarrow & (\exists\, x \in S \bullet rep(\overline{wp}(IT_H, gd(P_a)))) & \textit{Lemma D.24} \\
\Rrightarrow & (\exists\, x \in S \bullet \overline{wp}(IT_G, gd(Q_a))) & \textit{Condition 4 of Definition 6.9} \\
\Rrightarrow & \overline{wp}(IT_G, commgd(Q_{a.S})) & \textit{disjunction, Definition 6.2}
\end{array}
$$

Case $dir(a) = out$:

$$\begin{array}{lll} & rep(\overline{wp}(IT_H, commgd(P_{a.S}))) & \\ \Rrightarrow & rep(\overline{wp}(IT_H, gd(P_a) \wedge wp(P_a, y \in S))) & \textit{Definition 6.4} \\ \Rrightarrow & \overline{wp}(IT_G, gd(Q_a) \wedge wp(Q_a, y \in S)) & \textit{Condition 5 of Definition 6.9} \\ \Rrightarrow & \overline{wp}(IT_G, commgd(Q_{a.S})) & \textit{Definition 6.4} \end{array}$$

□

Lemma 6.12 *For* $X \subseteq A_{\mathcal{W}}$, $\quad rep(commgd_H(P_X)) \Rrightarrow commgd_G(Q_X)$.

Proof: Follows from Lemma D.25 and disjunction.

Proof of Lemma 6.16

In the following, we assume that $x, y \in \mathcal{W} \equiv true$. This can always be ensured by making $x, y \in \mathcal{W}$ an invariant of each action (cf. Section 7.1). We have the following lemma:

Lemma D.26 *For* $c \in C$, $\quad (\mathbf{var}\ x, y : \mathcal{W} \bullet P_c) = ([]\ i \in \mathcal{W} \bullet P_{c.i})$.

Proof: For ϕ independent of x, y, i, we have
Case $dir(c) = in$:

$$\begin{array}{lll} & wp(([]\ i \in \mathcal{W} \bullet P_{c.i}), \phi) & \\ \equiv & (\wedge\ i \in \mathcal{W} \bullet wp(P_{c.i}, \phi)) & \textit{Definition 2.2} \\ \equiv & (\wedge\ i \in \mathcal{W} \bullet wp(P_c, \phi)[x \backslash i]) & \textit{Definition 6.1} \\ \equiv & (\forall\, x \in \mathcal{W} \bullet wp(P_c, \phi)) & P_c,\ \phi \textit{ ind. of } i \\ \equiv & (\forall\, x, y \in \mathcal{W} \bullet wp(P_c, \phi)) & P_c \textit{ ind. of } y \\ \equiv & wp((\mathbf{var}\ x, y : \mathcal{W} \bullet P_c), \phi) & \textit{Definition 2.5} \end{array}$$

Case $dir(c) = out$:

$$\begin{array}{lll} & wp(([]\ i \in \mathcal{W} \bullet P_{c.i}), \phi) & \\ \equiv & (\wedge\ i \in \mathcal{W} \bullet wp(P_c, y = i \Rightarrow \phi)) & \textit{Definitions 2.2 and 6.3} \\ \equiv & wp(P_c, (\wedge\ i \in \mathcal{W} \bullet y = i \Rightarrow \phi)) & \textit{conjunction} \\ \equiv & wp(P_c, y \notin \mathcal{W} \vee \phi) & \phi \textit{ ind. of } i \\ \equiv & wp(P_c, \phi) & y \in \mathcal{W} \equiv true \\ \equiv & (\forall\, x, y \in \mathcal{W} \bullet wp(P_c, \phi)) & wp(P_c, \phi) \textit{ ind. of } x, y \\ \equiv & wp((\mathbf{var}\ x, y : \mathcal{W} \bullet P_c), \phi) & \textit{Definition 2.5} \end{array}$$

□

Lemma 6.16 *For* $b.i \in B_{\mathcal{W}}$,

$$Q_{b.i} * G \;=\; P_{b.i} * H\ ;\ \mathbf{it}\ ([]\ c.i \in C_{\mathcal{W}} \bullet P_{c.i} * H)\ \mathbf{ti}.$$

Proof:

$$
\begin{array}{lll}
& Q_{b.i} \ ; IT_G & \\
= & P_{b.i} \ ; \textbf{it } P_H \, [] ([] \, c \in C \bullet (\textbf{var } x, y : \mathcal{W} \bullet P_c)) \textbf{ ti} & \textit{Definition 6.14} \\
= & P_{b.i} \ ; IT_H \ ; \textbf{it } ([] \, c \in C \bullet (\textbf{var } x, y : \mathcal{W} \bullet P_c)) \ ; IT_H \textbf{ ti} & \textit{Lemma 4.5} \\
= & P_{b.i} \ ; IT_H \ ; \textbf{it } ([] \, c \in C \bullet ([] \, i \in \mathcal{W} \bullet P_{c.i})) \ ; IT_H \textbf{ ti} & \textit{Lemma D.26} \\
= & P_{b.i} \ ; IT_H \ ; \textbf{it } ([] \, c.i \in C_{\mathcal{W}} \bullet P_{c.i}) \ ; IT_H \textbf{ ti} & \textit{join choices} \\
= & P_{b.i} * H \ ; \textbf{it } ([] \, c.i \in C_{\mathcal{W}} \bullet P_{c.i} * H) \textbf{ ti} & \textit{defn. of } *H.
\end{array}
$$

Proof of Lemma 6.17

Firstly, we have the following lemma:

Lemma D.27 $commgd(P_{c.\mathcal{W}}) \ \equiv \ gd(\textbf{var } x, y : \mathcal{W} \bullet P_c)$.

Proof:
Case $dir(c) = in$:

$$
\begin{array}{lll}
& commgd(P_{c.\mathcal{W}}) & \\
\equiv & (\exists \, x \in \mathcal{W} \bullet gd(P_c)) & \textit{Definition 6.2} \\
\equiv & gd(\textbf{var } x : \mathcal{W} \bullet P_c) & \textit{Definition 2.5} \\
\equiv & gd(\textbf{var } x, y : \mathcal{W} \bullet P_c) & \textit{since } P_c \textit{ independent of } y
\end{array}
$$

Case $dir(c) = out$:

$$
\begin{array}{lll}
& commgd(P_{c.\mathcal{W}}) & \\
\equiv & gd(P_c) \wedge wp(P_c, y \in \mathcal{W}) & \textit{Definition 6.4} \\
\equiv & gd(P_c) & wp(P_c, y \in \mathcal{W}) \equiv \textit{true} \\
\equiv & gd(\textbf{var } x, y : \mathcal{W} \bullet P_c) & \textit{since } \phi, \ wp(P_c, \phi) \textit{ independent of } x, y
\end{array}
$$

□

Given this lemma we can show

Lemma D.28 *For* $C \subseteq A$,

$$commgd(P_{C_{\mathcal{W}}}) \ \equiv \ gd([] \, c \in C \bullet (\textbf{var } x, y : \mathcal{W} \bullet P_c))$$

□

We also have lemma as follows

Lemma D.29 *For* $X \subseteq B_{\mathcal{W}}$,

$$commgd(P_{X \cup C_{\mathcal{W}}}) \ \equiv \ commgd(P_X) \ \vee \ commgd(P_{C_{\mathcal{W}}}).$$

Proof: Follows easily from Definititon 6.6 and disjunction.

Lemma 6.17 *For* $X \subseteq B$,

$$\overline{wp}(IT_G, \neg \, commgd_G(Q_X)) \ \equiv \ \overline{wp}(IT_G, \neg \, commgd_H(P_{X \cup C_{\mathcal{W}}})).$$

Proof: Given Lemmas D.28, D.29, proof is similar to proof of Lemma 4.16.

Proof of Lemma 6.25

Firstly, we have lemmas as follows:

Lemma D.30 *For input action P_c, output action Q_c,*

$$(P_c[\textbf{value } x\backslash i]) \parallel (Q_c[\textbf{value } x\backslash i]) \;\cong\; (P_c \parallel Q_c)[\textbf{value } x\backslash i].$$

Proof: Follows easily from Property 5.1 and Definition 2.6.

Lemma D.31 *For $c.i \in C_{\mathcal{W}}$,* $\qquad R_{c.i} \cong \tilde{P}_{c.i} \parallel \tilde{Q}_{c.i}$.

Proof: In the case that $c \in A - B$, proof is trivial since $\tilde{Q}_{c.i} = \textbf{skip}$. Similarly for $c \in B - A$. In the case that $c \in IN$, proof follows from Lemma D.30. In the case that $c \in OUT_P$, we have

$$\begin{array}{lll}
R_{c.i} & = & R_c[y = i] & \textit{Definition 6.1} \\
& = & (P_c \vec{\parallel} Q_c)[y = i] & \textit{Definition 6.22} \\
& \cong & (P_c[y = i]) \parallel (Q_c[\textbf{value } x\backslash i]) & \textit{Property 6.21 (a)} \\
& = & P_{c.i} \parallel Q_{c.i} & \textit{Definitions 6.3, 6.1} \\
& = & \tilde{P}_{c.i} \parallel \tilde{Q}_{c.i} & \textit{definition of } \tilde{P}, \tilde{Q}
\end{array}$$

Similarly for $c \in OUT_Q$.
□

Lemma 6.25 *For $c.i \in C_{\mathcal{W}}$,* $\qquad R_{c.i} * I \;\cong\; (\tilde{P}_{c.i} * H) \parallel (\tilde{Q}_{c.i} * G)$

Given Lemma D.31, proof of Lemma 6.25 is similar to proof of Lemma 5.10.

Proof of Lemma 6.26

Lemma D.32 *For $c \in OUT_P \cup OUT_Q$, $S \subseteq \mathcal{W}$,*

$$\textit{commgd}(R_{c.S}) \;\equiv\; \textit{commgd}(P_{c.S}) \wedge \textit{commgd}(Q_{c.S}).$$

Proof: In the case where $c \in OUT_P$, we have firstly

$$\begin{array}{lll}
& \textit{commgd}(Q_{c.S}) & \\
\equiv & (\exists\, x \in S \bullet \textit{gd}(Q_c)) & \textit{Definition 6.2} \\
\equiv & (\exists\, x \in S \bullet x \in \textit{type}_Q(c) \wedge \textit{gd}(\textbf{var } x \bullet Q_c)) & \textit{Property 6.19} \\
\equiv & S \cap \textit{type}_Q(c) \neq \{\} \wedge \textit{gd}(\textbf{var } x \bullet Q_c) & \textit{pred. calc.}
\end{array}$$

secondly

$$\begin{array}{lll}
& \textit{commgd}(P_{c.S}) & \\
\equiv & \textit{gd}(P_c) \wedge \textit{wp}(P_c, y \in S) & \textit{Definition 6.4} \\
\equiv & \textit{gd}(P_c) \wedge \textit{wp}(P_c, y \in S \wedge y \in \textit{type}_Q(c)) & \\
& & \textit{Property 6.20, } \textit{type}_P(c) \subseteq \textit{type}_Q(c) \\
\equiv & \textit{gd}(P_c) \wedge \textit{wp}(P_c, y \in S \wedge S \cap \textit{type}_Q(c) \neq \{\}) & \textit{pred. calc.} \\
\equiv & \textit{gd}(P_c) \wedge \textit{wp}(P_c, y \in S) \wedge S \cap \textit{type}_Q(c) \neq \{\} & \textit{since } \textit{halt}(P_c) \equiv \textit{true}
\end{array}$$

so that

$$\begin{array}{lll} & commgd(R_{c.S}) & \\ \equiv & gd(R_c) \wedge wp(R_c, y \in S) & \textit{Definition 6.4} \\ \equiv & gd(P_c \vec{\|} Q_c) \wedge wp(P_c \vec{\|} Q_c, y \in S) & \textit{Definition 6.22} \\ \equiv & gd(P_c) \wedge gd(\mathbf{var}\ x \bullet Q_c) \wedge wp(P_c, y \in S) & \textit{Property 6.21 (b)} \\ \equiv & gd(P_c) \wedge wp(P_c, y \in S) \wedge S \cap type_Q(c) \neq \{\} \wedge gd(\mathbf{var}\ x \bullet Q_c) & \textit{see above} \\ \equiv & commgd(P_{c.S}) \wedge commgd(Q_{c.S}) & \textit{see above} \end{array}$$

Proof is similar for the case where $c \in OUT_Q$.
□

Lemma D.33 *For $c \in OUT_P$,*

$$commgd(Q_{c.S}) \quad \Rrightarrow \quad gd(\mathbf{var}\ x \bullet Q_c) \wedge S \cap type_P(c) \neq \{\}.$$

Proof: Follows from Property 6.19 and $type_P(c) \subseteq type_Q(c)$. There is a similar lemma for $c \in OUT_Q$.

Lemma D.34 *For $c \in OUT_P$, $J, K \subseteq \mathcal{W}$, where $K \cap type_P(c) = \{\}$,*

$$\overline{wp}(P_c, y \notin J) \ \Rrightarrow \overline{wp}(P_c, y \notin (J \cup K)).$$

Proof:

$$\begin{array}{lll} & \overline{wp}(P_c, y \notin J) & \\ \Rrightarrow & \overline{wp}(P_c, y \notin J) \wedge wp(P_c, y \in type_P(c)) & \textit{Property 6.20} \\ \Rrightarrow & \overline{wp}(P_c, y \notin J) \wedge wp(P_c, y \notin K) & K \cap type_P(c) = \{\} \\ \Rrightarrow & \overline{wp}(P_c, y \notin J \wedge y \notin K) & \textit{Theorem 2.8} \end{array}$$

□

Lemma D.35 *For $S \subseteq \mathcal{W}$, $c \in A \cap B$,*

$$\begin{array}{ll} halt(IT_I) \wedge \neg\ commgd_I(R_{c.S}) & \text{(i)} \\ \quad \equiv \ halt(IT_I) \wedge (\vee\ J, K \mid J \cup K = S \bullet \neg\ commgd_H(P_{c.J}) \wedge \neg\ commgd_G(Q_{c.K})). & \text{(ii)} \end{array}$$

Proof: In the case that $c \in IN$, we have

$$\begin{array}{lll} & halt(IT_I) \wedge \neg\ commgd_I(R_{c.S}) & \\ \equiv & halt(IT_I) \wedge wp(IT_I, (\forall\, x \in S \bullet \neg\ gd(R_c))) & \textit{Defn. 6.2, negation} \\ \equiv & halt(IT_I) \wedge wp(IT_I, (\forall\, x \in S \bullet \neg\ gd(P_c) \vee \neg\ gd(Q_c))) & \textit{Property 5.1} \\ \equiv & halt(IT_I) \wedge (\forall\, x \in S \bullet wp(IT_I, \neg\ gd(P_c) \vee \neg\ gd(Q_c))) & \textit{conjunction} \\ \equiv & halt(IT_I) \wedge (\forall\, x \in S \bullet wp(IT_H, \neg\ gd(P_c)) \vee wp(IT_G, \neg\ gd(Q_c))) & \textit{Lemma 5.7} \\ \equiv & halt(IT_I) \wedge (\vee\ J, K \mid J \cup K = S \bullet & \\ & \qquad (\forall\, x \in J \bullet wp(IT_H, \neg\ gd(P_c))) & \\ & \qquad \wedge (\forall\, x \in K \bullet wp(IT_G, \neg\ gd(Q_c)))) & \textit{Lemma C.17} \\ \equiv & halt(IT_I) \wedge (\vee\ J, K \mid J \cup K = S \bullet & \\ & \qquad \neg\ commgd_H(P_{c.J}) \wedge \neg\ commgd_G(Q_{c.K})) & \textit{conjunction.} \end{array}$$

Now, consider the case where $c \in OUT_P$. To show (i) $\Rrightarrow$ (ii), use Lemma D.32 and take $J = K = S$. To show (ii) $\Rrightarrow$ (i), we have for $J \cup K = S$,

$$
\begin{array}{lll}
 & halt(IT_I) \wedge \neg\, commgd_H(P_{c.J}) \wedge \neg\, commgd_G(Q_{c.K}) & \\
\Rrightarrow & halt(IT_I) \wedge wp(IT_H, \neg\, gd(P_c) \vee \overline{wp}(P_c, y \notin J)) & \\
 & \wedge\, wp(IT_G, \neg\, gd(\mathbf{var}\ x \bullet Q_c) \vee K \cap type_P(C) = \{\}) & \textit{Lemma D.33} \\
\Rrightarrow & halt(IT_I) \wedge wp(IT_H, \neg\, gd(P_c) \vee \overline{wp}(P_c, y \notin J)) & \\
 & \wedge\, (wp(IT_G, \neg\, gd(\mathbf{var}\ x \bullet Q_c)) \ \vee\ K \cap type_P(C) = \{\}) & \textit{move const. out} \\
\Rrightarrow & halt(IT_I) \wedge & \\
 & (wp(IT_G, \neg\, gd(\mathbf{var}\ x \bullet Q_c)) & \\
 & \vee\ wp(IT_H, \neg\, gd(P_c) \vee \overline{wp}(P_c, y \notin J)) \wedge K \cap type_P(C) = \{\}) & \textit{pred. calc.} \\
\Rrightarrow & halt(IT_I) \wedge & \\
 & (wp(IT_G, \neg\, gd(\mathbf{var}\ x \bullet Q_c)) & \\
 & \vee\ wp(IT_H, \neg\, gd(P_c) \vee \overline{wp}(P_c, y \notin S))) & \textit{Lemma D.34} \\
\Rrightarrow & halt(IT_I) \wedge (\neg\, commgd_H(P_{c.S}) \vee \neg\, commgd_G(Q_{c.S})) & \textit{Defs. 6.2,6.4} \\
\Rrightarrow & halt(IT_I) \wedge \neg\, commgd_I(R_{c.S}) & \textit{Lemma D.32}
\end{array}
$$

The case where $c \in OUT_Q$ is similarly proven.
□

Lemma D.36 *For* $Z \subseteq (A \cup B)_{\mathcal{W}}$,

$$
\begin{array}{l}
halt(IT_I) \wedge \neg\, commgd_I(R_Z) \\
\quad \equiv \quad halt(IT_I) \wedge (\vee\, X, Y \mid X \cup Y = Z \bullet \neg\, commgd_H(P_X) \wedge \neg\, commgd_G(Q_y)).
\end{array}
$$

Proof: By deductive argument along the lines of Lemma C.17, using Lemma D.35.

Lemma 6.26 *For* $Z \subseteq C_{\mathcal{W}}$,

$$
\begin{array}{l}
\overline{wp}(IT_I, \neg\, commgd_I(R_Z)) \\
\quad \equiv \quad (\vee\, X \subseteq A_{\mathcal{W}}, Y \subseteq B_{\mathcal{W}} \mid X \cup Y = Z \bullet \overline{wp}(IT_I, \neg\, commgd_H(P_X) \wedge \neg\, commgd_G(Q_y))).
\end{array}
$$

Proof: Follows from Lemma D.36, as shown in proof of Lemma 5.14.

Bi-directional Channels and Well-formedness

We show that Theorem 6.8 holds for action systems with bi-directional actions. Firstly, we assume that any bi-directional action leaves the input parameter unchanged, so that the following property is satisfied:

Property D.37 *For inout action* P_a, *x-predicate* ψ, *any* ϕ,

$$wp(P_a, \psi \vee \phi) \quad \equiv \quad \psi \vee wp(P_a, \phi)$$

□

Assume that $\mathcal{W}$ satisfies the following property:

Property D.38 $(j, k) \in \mathcal{W} \;\equiv\; j \in \mathcal{W} \wedge k \in \mathcal{W}$.

Let $\mathcal{W}_0 \;\widehat{=}\; \{\; i \in \mathcal{W} \mid i \text{ is not a pair } \}$. We then have the following lemma:

Lemma D.39 *For inout action P_a, value $i \in (\mathcal{W} - \mathcal{W}_0)$, let $j \mathrel{\widehat{=}} fst(i)$ and $k \mathrel{\widehat{=}} snd(i)$. Then*

$$P_{a.i} \;=\; P_a[\textbf{value } x \backslash j][y = k].$$

For $i \in \mathcal{W}_0$, $\qquad P_{a.i} \;=\; \textbf{miracle}.$

Proof: In the case that $i \in (\mathcal{W} - \mathcal{W}_0)$, we have:

$$\begin{array}{cll}
 & wp(P_{a.i}, \phi) & \\
\equiv & wp((\textbf{var } x, y \bullet [(x, y) = (j, k)] P_a[(x, y) = (j, k)]), \phi) & \textit{Definition 6.27} \\
\equiv & (\forall\, x, y \bullet x = j \wedge y = k \Rightarrow wp(P_a, x = j \wedge y = k \Rightarrow \phi)) & \textit{wp-calc.} \\
\equiv & wp(P_a, x = j \wedge y = k \Rightarrow \phi)[x \backslash j] & wp(P_a, \phi) \textit{ ind. of } y \\
\equiv & (x = j \Rightarrow wp(P_a, y = k \Rightarrow \phi))[x \backslash j] & \textit{Property D.37} \\
\equiv & wp(P_a[\textbf{value } x \backslash j][y = k], \phi) & \textit{wp-calc.}
\end{array}$$

In the case that $i \in \mathcal{W}_0$ then trivially $P_{a.i} = \textbf{miracle}$.
□

The proof that Theorem 6.8 holds with bi-directional channels is similar to the proof already given for mono-directional channels: construct an action system P^+ and show that $\{\!\!\{P^+\}\!\!\} = \{\!\!\{P\}\!\!\}$. For input and output actions, the construction of the corresponding actions of P^+ is as already given. For each bi-directional channel P_a of P, add the set of distinct variables $\{\; y_{a.i} \mid i \in \mathcal{W} \;\}$ to P^+. A value for each $y_{a.i}$ is chosen by

$$CHS_{a.i} \;\;\widehat{=}\;\; \textit{feasible}(P_a[\textbf{value } x \backslash i][y \backslash y_{a.i}][\textbf{fix } v]).$$

For each bi-directional channel P_a of P, add the set of actions $\{\; P_{a.i} \mid i \in \mathcal{W} \;\}$ as follows: for $i \in (\mathcal{W} - \mathcal{W}_0)$, where $i = (j, k)$,

$$P^+_{a.i} \;\;\widehat{=}\;\; y_{a.j} = k \rightarrow (\textbf{var } y \bullet P_a[\textbf{value } x \backslash j][y = k]) \;;\; CHS,$$

while, for $i \in \mathcal{W}_0$, $P_{a.i} \;\widehat{=}\; \textbf{miracle}$.

Using lemma D.39, it can be shown that $\{\!\!\{P^+\}\!\!\} = \{\!\!\{P\}\!\!\}$ in the manner already given for Theorem 6.8. Since $\{\!\!\{P^+\}\!\!\}$ is a normal action system, it is well-formed, thus $\{\!\!\{P\}\!\!\}$ is also well-formed.

Bi-directional Channels and Refinement

We show that Theorem 6.10 holds for action systems with bi-directional channels. Note: Condition 5 of Definition 6.9 now applies to bi-directional actions as well as output actions. Firstly, we have that Lemma 6.11 is valid for bi-directional channels:

Lemma 6.11 *For $a.i \in A_{\mathcal{W}}$,* $\qquad P_{a.i} \preceq_{rep} Q_{a.i}$.

Proof: In the case that $i \in \mathcal{W}_0$, proof is trivial. In the case that $i \in (\mathcal{W} - \mathcal{W}_0)$, we have $i = (j, k)$ for some j, k, and

$$
\begin{array}{lll}
 & rep(wp(P_{a.i}, \phi)) & \\
\Rrightarrow & rep(wp(P_a[\mathbf{value}\ x \backslash j][y = k], \phi)) & \textit{Lemma D.39} \\
\Rrightarrow & rep(wp(P_a, y = k \Rightarrow \phi)[x \backslash j]) & \\
\Rrightarrow & rep(wp(P_a, y = k \Rightarrow \phi))[x \backslash j] & (b) \textit{ of Theorem 6.8} \\
\Rrightarrow & wp(Q_a, rep(y = k \Rightarrow \phi))[x \backslash j] & P_a \preceq_{rep} Q_a \\
\Rrightarrow & wp(Q_a, y = k \Rightarrow rep(\phi))[x \backslash j] & \textit{disjunction,}(c) \textit{ of Theorem 6.8} \\
\Rrightarrow & wp(Q_{a.i}, rep(\phi)) & \textit{Lemma D.39}
\end{array}
$$

□

Secondly, Lemma D.25 is valid for bi-directional actions:

Lemma D.25 *For $\{\} \subset S \subseteq \mathcal{W}$,*

$$rep(\overline{wp}(IT_H, commgd(P_{a.S}))) \ \Rrightarrow\ \overline{wp}(IT_G, commgd(Q_{a.S})).$$

Proof: Case $fst(S) = \{\}$:

$$
\begin{array}{lll}
 & rep(\overline{wp}(IT_H, commgd(P_{a.S}))) & \\
\Rrightarrow & rep(\overline{wp}(IT_H, false)) & \textit{Def. 6.28} \\
\Rrightarrow & rep(\overline{wp}(IT_H, gd(P_a) \wedge wp(P_a, false))) & \textit{monotonicity} \\
\Rrightarrow & \overline{wp}(IT_G, gd(Q_a) \wedge wp(Q_a, false)) & \textit{5 of Def. 6.9} \\
\Rrightarrow & \overline{wp}(IT_G, false) & \textit{since } wp(Q_a, false) \equiv \neg\, gd(Q_a) \\
\Rrightarrow & \overline{wp}(IT_G, commgd(Q_{a.S})) & \textit{Def. 6.28}
\end{array}
$$

Case $fst(S) \neq \{\}$:

$$
\begin{array}{lll}
 & rep(\overline{wp}(IT_H, commgd(P_{a.S}))) & \\
\Rrightarrow & rep(\overline{wp}(IT_H, (\exists\, x \in fst(S) \bullet gd(P_a) \wedge wp(P_a, y \in S_x)))) & \textit{Def. 6.28} \\
\Rrightarrow & (\exists\, x \in fst(S) \bullet rep(\overline{wp}(IT_H, gd(P_a) \wedge wp(P_a, y \in S_x)))) & \textit{disjunction} \\
\Rrightarrow & (\exists\, x \in fst(S) \bullet \overline{wp}(IT_G, gd(Q_a) \wedge wp(Q_a, y \in S_x))) & \textit{5 of Def. 6.9} \\
\Rrightarrow & \overline{wp}(IT_G, (\exists\, x \in fst(S) \bullet gd(Q_a) \wedge wp(Q_a, y \in S_x))) & \textit{disjunction} \\
\Rrightarrow & \overline{wp}(IT_G, commgd(Q_{a.S})) & \textit{Def. 6.27}
\end{array}
$$

□

This means that Lemma 6.12 also holds for bi-directional actions (see proof of Lemmma 6.12). Finally, $P \sqsubseteq_{rep} Q$ implies $P \sqsubseteq Q$ as described in the proof of Theorem 6.10.

Bi-directional Channels and Hiding

Hiding of bi-directional channnels is as described in Definition 6.14. We have that

$$\{\!\!\{P \setminus C\}\!\!\} \;=\; \{\!\!\{P\}\!\!\} \setminus C_{\mathcal{W}}.$$

Proof is as in the proof of Theorem 6.15. Firstly, Lemma D.26 holds for inout actions:

Lemma D.26 $(\mathbf{var}\ x, y : \mathcal{W} \bullet P_c) \;=\; ([\!]\ i \in \mathcal{W} \bullet P_{c.i})$

Proof:

$$\begin{array}{lll}
& wp(\ ([\!]\ i \in \mathcal{W} \bullet P_{c.i}), \phi) & \\
\equiv & (\wedge\ i \in \mathcal{W} \bullet wp(P_{c.i}, \phi)) & \\
\equiv & (\wedge\ j, k \in \mathcal{W} \bullet wp(P_{c.(j,k)}, \phi)) & \\
& \wedge\ (\wedge\ i \in \mathcal{W}_0 \bullet wp(P_{c.i}, \phi)) & \\
\equiv & (\wedge\ j, k \in \mathcal{W} \bullet wp(P_c, k = y \Rightarrow \phi)[x \backslash j]) & \\
& \wedge\ (\wedge\ i \in \mathcal{W}_0 \bullet true) & \textit{Lemma D.39} \\
\equiv & (\wedge\ j \in \mathcal{W} \bullet wp(P_c, (\wedge\ k \in \mathcal{W} \bullet k = y \Rightarrow \phi))[x \backslash j]) & \textit{conjunction} \\
\equiv & (\forall\, x, y \in \mathcal{W} \bullet wp(P_c, \phi)) & \phi,\ wp(P_c, \phi)\ \textit{ind. of}\ y \\
\equiv & wp((\mathbf{var}\ x, y \in \mathcal{W} \bullet P_c), \phi)) &
\end{array}$$

□

Thus, Lemma 6.16 holds for bi-directional actions.

Secondly, Lemma D.27 holds for bi-directional actions:

Lemma D.27 $commgd(P_{c.\mathcal{W}}) \;\equiv\; gd(\mathbf{var}\ x, y : \mathcal{W} \bullet P_c).$

Proof:

$$\begin{array}{lll}
& commgd(P_{c.\mathcal{W}}) & \\
\equiv & (\exists\, x \in \mathit{fst}(\mathcal{W}) \bullet gd(P_c) \wedge wp(P_c, y \in \mathcal{W}_x)) & \textit{Definition 6.28} \\
\equiv & (\exists\, x \in \mathcal{W} \bullet gd(P_c) \wedge wp(P_c, y \in \mathcal{W})) & \textit{Property D.38} \\
\equiv & (\exists\, x \in \mathcal{W} \bullet gd(P_c)) & halt(P_c) \equiv true \\
\equiv & gd(\mathbf{var}\ x, y : \mathcal{W} \bullet P_c) & gd(P_c)\ \textit{ind. of}\ y
\end{array}$$

□

Thus Lemma 6.17 holds for bi-directional actions.

Finally, $\{\!\!\{P \setminus C\}\!\!\} \;=\; \{\!\!\{P\}\!\!\} \setminus C_{\mathcal{W}}$ is proven as described in the proof of Theorem 6.15.

Appendix E

Refinement of File System

First Refinement of File System

Replace *wait* with *twait* and *mwait*:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\;FS^2\;}}\\
fstore : Fid \rightarrow File \\
twait : Rid \rightarrowtail\!\!\!\!\!\!\!+ \; (Tid \times Fid \times Op) \\
mwait : Rid \rightarrowtail\!\!\!\!\!\!\!+ \; (Sid \times Tid \times Fid \times Op) \\
comp : Rid \rightarrowtail\!\!\!\!\!\!\!+ \; (Tid \times Rep) \\
unused : \mathbb{P}\, Rid \\
\hline
disjoint\ \{dom(twait), dom(mwait), dom(comp), unused\} \\
\hline
\end{array}
$$

FS^1 and FS^2 are related by the following abstraction invariant:

$$
\begin{aligned}
{}^1AI^2 \;\widehat{=}\; & FS^2 \land \\
& wait = twait \cup \{\, (i, t, n, p) \mid (\exists\, s \bullet (i, s, t, n, p) \in mwait \,\}
\end{aligned}
$$

$FileSys^2$ is defined as follows:

$$
\begin{array}{ll}
\textbf{var}\ FS^2 & \\
\textbf{initially} & \left[\begin{array}{l} fstore = (\lambda\, n : Fid \bullet \bot) \\ \land\ twait = mwait = comp = \{\} \\ \land\ unused = Rid \end{array}\right]
\end{array}
$$

$$
\begin{array}{l}
\textbf{for}\ t \in Tid\ \textbf{chan}\ req_t\ \textbf{in}\ n? : Fid;\ p? : Op\ \textbf{out}\ i! : Rid :- \\
\begin{array}{r} twait, \\ unused \end{array} : \left[\begin{array}{l} i! \in unused_0 \\ \land\ unused = unused_0 \setminus \{i!\} \\ \land\ twait = twait_0 \cup \{(i!, t, n?, p?)\} \end{array}\right]
\end{array}
$$

$$
\begin{array}{l}
\textbf{for}\ t \in Tid\ \textbf{chan}\ resp_t\ \textbf{out}\ i! : Rid;\ r! : Rep :- \\
comp\ : \left[\begin{array}{l} (i!, t, r!) \in comp_0 \\ \land\ comp = \{i!\} \lhd comp_0 \end{array}\right]
\end{array}
$$

Requests are moved from *twait* to *mwait* by *treq*:

$$\textbf{internal}\ \ treq :-$$

$$\begin{array}{c} twait, \\ mwait \end{array} : \left[\begin{array}{l} (\exists\, i : Rid;\ s : Sid;\ t : Tid;\ n : Fid;\ p : Op \bullet \\ \qquad (i, t, n, p) \in twait_0 \\ \qquad \wedge\ s = which(n) \\ \qquad \wedge\ twait = \{i\} \lhd twait_0 \\ \qquad \wedge\ mwait = mwait_0 \cup \{(i, s, t, n, p)\}) \end{array}\right]$$

$$\textbf{internal}\ \ sreq :-$$

$$\begin{array}{c} mwait, \\ comp, \\ fstore \end{array} : \left[\begin{array}{l} (\exists\, i : Rid;\ s : Sid;\ t : Tid;\ n : Fid;\ p : Op;\ f : File;\ r : Rep \bullet \\ \qquad (i, s, t, n, p) \in mwait_0 \\ \qquad \wedge\ (f, r) = p(fstore_0(n)) \\ \qquad \wedge\ mwait = \{i\} \lhd mwait_0 \\ \qquad \wedge\ comp = comp_0 \cup \{(i, t, r)\} \\ \qquad \wedge\ fstore = fstore_0 \oplus \{n \mapsto f\}) \end{array}\right]$$

To show $FileSys^1 \sqsubseteq FileSys^2$, firstly rename $service^1$ to $sreq^1$. This refinement step then consists of introducing *treq*, so the hiding refinement rule is applied.

Second Refinement of File System

Replace *comp* with *scomp*, *mcomp* and *tcomp*:

$$\begin{array}{|l} \hline FS^3 \\ fstore : Fid \rightarrow File \\ twait : Rid \nrightarrow (Tid \times Fid \times Op) \\ mwait : Rid \nrightarrow (Sid \times Tid \times Fid \times Op) \\ scomp : Sid \nrightarrow (Rid \times Tid \times Rep) \\ mcomp, tcomp : Rid \nrightarrow (Tid \times Rep) \\ unused : \mathbb{P}\, Rid \\ \hline ran(scomp) \in Rid \nrightarrow (Tid \times Rep) \\ disjoint\ \{dom(twait), dom(mwait), \{\ i \mid (i, t, r) \in ran(scomp)\ \}, \\ \qquad\qquad dom(mcomp), dom(tcomp), unused\} \\ \hline \end{array}$$

$$\begin{array}{rcl} {}^2AI^3 & \widehat{=} & FS^3 \wedge \\ & & comp = mcomp \cup tcomp \cup ran(scomp) \end{array}$$

$FileSys^3$ is defined as follows:

$$\textbf{var}\ FS^3$$

$$\textbf{initially} \quad \left[\begin{array}{l} fstore = (\lambda\, n : Fid \bullet \perp) \\ \wedge\ twait = mwait = \{\} \\ \wedge\ scomp = mcomp = tcomp = \{\} \\ \wedge\ unused = Rid \end{array}\right]$$

for $t \in Tid$ **chan** req_t **in** $n? : Fid;\ p? : Op$ **out** $i! : Rid$:–
(body as $FileSys^2$)

for $t \in Tid$ **chan** $resp_t$ **out** $i! : Rid;\ r! : Rep$:–

$$tcomp \ : \left[\begin{array}{l} (i!, t, r!) \in tcomp_0 \\ \land\ tcomp = \{i!\} \lhd tcomp_0 \end{array} \right]$$

internal $treq$:– (body as $FileSys^2$)

internal $sreq$:–

$$\begin{array}{c} mwait, \\ scomp, \\ fstore \end{array} : \left[\begin{array}{l} (\exists\, s : Sid;\ i : Rid;\ t : Tid;\ n : Fid;\ p : Op;\ f : File;\ r : Rep \bullet \\ \qquad s \notin dom(scomp_0) \\ \qquad \land\ (i, s, t, n, p) \in mwait_0 \\ \qquad \land\ (f, r) = p(fstore_0(n)) \\ \qquad \land\ mwait = \{i\} \lhd mwait_0 \\ \qquad \land\ scomp = scomp_0 \cup \{(s, i, t, r)\} \\ \qquad \land\ fstore = fstore_0 \oplus \{n \mapsto f\}) \end{array} \right]$$

Responses are moved from *scomp* to *mcomp* by *sresp*:

internal $sresp$:–

$$\begin{array}{c} scomp, \\ mcomp \end{array} : \left[\begin{array}{l} (\exists\, s : Sid;\ i : Rid;\ t : Tid;\ r : Rep \bullet \\ \qquad (s, i, t, r) \in scomp_0 \\ \qquad \land\ scomp = \{s\} \lhd scomp_0 \\ \qquad \land\ mcomp = mcomp_0 \cup \{(i, t, r)\}) \end{array} \right]$$

Responses are moves from *mcomp* to *tcomp* by *tresp*:

internal $tresp$:–

$$\begin{array}{c} mcomp, \\ tcomp \end{array} : \left[\begin{array}{l} (\exists\, i : Rid;\ t : Tid;\ r : Rep \bullet \\ \qquad (i, t, r) \in mcomp_0 \\ \qquad \land\ mcomp = \{i\} \lhd mcomp_0 \\ \qquad \land\ tcomp = tcomp_0 \cup \{(i, t, r)\}) \end{array} \right]$$

In this step, two internal actions, *sresp* and *tresp*, have been introduced, so the hiding refinement rule is used to show $FileSys^2 \sqsubseteq FileSys^3$.

Third Refinement of File System

Simplify the invariant:

$$\begin{array}{|l}
\multicolumn{1}{l}{FS^4} \\ \hline
fstore : Fid \rightarrow File \\
twait : \mathbb{P}(Rid \times Tid \times Fid \times Op) \\
mwait : \mathbb{P}(Rid \times Sid \times Tid \times Fid \times Op) \\
scomp : \mathbb{P}(Sid \times Rid \times Tid \times Rep) \\
mcomp, tcomp : \mathbb{P}(Rid \times Tid \times Rep) \\
unused : \mathbb{P}\ Rid \\ \hline
\end{array}$$

Use Rule 7.19 to show $FileSys^3 \sqsubseteq FileSys^4$.

Fourth Refinement of File System

Replace $mwait$ and $mcomp$ with $mail$:

$$\begin{array}{|l}
\multicolumn{1}{l}{FS^5} \\ \hline
fstore : Fid \rightarrow File \\
twait : \mathbb{P}(Rid \times Tid \times Fid \times Op) \\
scomp : \mathbb{P}(Sid \times Rid \times Tid \times Rep) \\
tcomp : \mathbb{P}(Rid \times Tid \times Rep) \\
mail : \mathbf{bag}\ Env \\
unused : \mathbb{P}\ Rid \\ \hline
(\forall s : Sid, t : Tid \bullet \\
\quad (s, m) \in mail \Rightarrow m \in ran(mq) \\
\quad \wedge\ (t, m) \in mail \Rightarrow m \in ran(mp)) \\ \hline
\end{array}$$

Here, the invariant ensures that messages for servers are requests and messages for terminals are responses.

$$\begin{array}{rcl}
{}^4AI^5 & \widehat{=} & FS^5 \wedge \\
& & mwait = \{\ (i, s, t, n, p) \mid (s, mq(i, t, n, p)) \in mail\ \} \\
& & mcomp = \{\ (i, t, r) \mid (t, mp(i, r)) \in mail\ \}
\end{array}$$

$FileSys^5$ is defined as follows:

var FS^5

initially $\left[\begin{array}{l} fstore = (\lambda\, n : Fid \bullet \perp) \\ \wedge\ twait = scomp = tcomp = \{\} \\ \wedge\ mail = \prec\succ \\ \wedge\ unused = Rid \end{array}\right]$

for $t \in Tid$ **chan** req_t **in** $n? : Fid;\ p? : Op$ **out** $i! : Rid$:–
(body as $FileSys^2$)

for $t \in Tid$ **chan** $resp_t$ **out** $i! : Rid;\ r! : Rep$:–
(body as $FileSys^3$)

$$\textbf{internal}\ \ \mathit{treq} :-$$

$$\begin{array}{c} \mathit{twait}, \\ \mathit{mail} \end{array} : \left[\begin{array}{l} (\exists\, i : \mathit{Rid};\ s : \mathit{Sid};\ t : \mathit{Tid};\ n : \mathit{Fid};\ p : \mathit{Op};\ m : \mathit{Mess} \bullet \\ \qquad (i, t, n, p) \in \mathit{twait}_0 \\ \qquad \wedge\ s = \mathit{which}(n) \\ \qquad \wedge\ m = \mathit{mq}(i, t, n, p) \\ \qquad \wedge\ \mathit{twait} = \{i\} \lhd \mathit{twait}_0 \\ \qquad \wedge\ \mathit{mail} = \mathit{mail} + \prec\!(s, m)\!\succ) \end{array} \right]$$

$$\textbf{internal}\ \ \mathit{sreq} :-$$

$$\begin{array}{c} \mathit{mail}, \\ \mathit{scomp}, \\ \mathit{fstore} \end{array} : \left[\begin{array}{l} (\exists\, s : \mathit{Sid};\ i : \mathit{Rid};\ t : \mathit{Tid};\ n : \mathit{Fid};\ p : \mathit{Op}; \\ \quad f : \mathit{File};\ r : \mathit{Rep};\ m : \mathit{Mess} \bullet \\ \qquad s \notin \mathit{dom}(\mathit{scomp}_0) \\ \qquad \wedge\ (s, m) \in \mathit{mail}_0 \\ \qquad \wedge\ m = \mathit{mq}(i, t, n, p) \\ \qquad \wedge\ (f, r) = p(\mathit{fstore}_0(n)) \\ \qquad \wedge\ \mathit{mail} = \mathit{mail}_0 - \prec\!(s, m)\!\succ \\ \qquad \wedge\ \mathit{scomp} = \mathit{scomp}_0 \cup \{(s, i, t, r)\} \\ \qquad \wedge\ \mathit{fstore} = \mathit{fstore}_0 \oplus \{n \mapsto f\}) \end{array} \right]$$

$$\textbf{internal}\ \ \mathit{sresp} :-$$

$$\begin{array}{c} \mathit{scomp}, \\ \mathit{mail} \end{array} : \left[\begin{array}{l} (\exists\, s : \mathit{Sid};\ i : \mathit{Rid};\ t : \mathit{Tid};\ r : \mathit{Rep};\ m : \mathit{Mess} \bullet \\ \qquad (s, i, t, r) \in \mathit{scomp}_0 \\ \qquad \wedge\ \mathit{scomp} = \{s\} \lhd \mathit{scomp}_0 \\ \qquad \wedge\ m = \mathit{mp}(i, r) \\ \qquad \wedge\ \mathit{mail} = \mathit{mail}_0 + \prec\!(t, m)\!\succ) \end{array} \right]$$

$$\textbf{internal}\ \ \mathit{tresp} :-$$

$$\begin{array}{c} \mathit{mail}, \\ \mathit{tcomp} \end{array} : \left[\begin{array}{l} (\exists\, i : \mathit{Rid};\ t : \mathit{Tid};\ r : \mathit{Rep};\ m : \mathit{Mess} \bullet \\ \qquad (t, m) \in \mathit{mail}_0 \\ \qquad \wedge\ m = \mathit{mp}(i, r) \\ \qquad \wedge\ \mathit{mail} = \mathit{mail}_0 - \prec\!(t, m)\!\succ \\ \qquad \wedge\ \mathit{tcomp} = \mathit{tcomp}_0 \cup \{(i, t, r)\}) \end{array} \right]$$

Fifth Refinement of File System

$$\mathit{FS}^6\ \widehat{=}\ \mathit{FS}^5$$

Replace *sreq* with:

$$\textbf{internal}\ \ sreq :- \\
\begin{array}{c} mail, \\ scomp, \\ fstore \end{array} : \left[\begin{array}{l}
(\exists\, s : Sid;\ i : Rid;\ t : Tid;\ n : Fid;\ p : Op; \\
\quad f : File;\ r : Rep;\ m : Mess \bullet \\
\qquad s \notin dom(scomp_0) \\
\qquad \wedge\ (s, m) \in mail_0 \\
\qquad \wedge\ mail = mail_0 - \prec(s, m)\succ \\
\qquad \wedge\ m \in ran(mq) \Rightarrow \\
\qquad\qquad (m = mq(i, t, n, p) \\
\qquad\qquad \wedge\ (f, r) = p(fstore_0(n)) \\
\qquad\qquad \wedge\ scomp = scomp_0 \cup \{(s, i, t, r)\} \\
\qquad\qquad \wedge\ fstore = fstore_0 \oplus \{n \mapsto f\}))
\end{array}\right]$$

Replace *tresp* with:

$$\textbf{internal}\ \ tresp :- \\
\begin{array}{c} mail, \\ tcomp \end{array} : \left[\begin{array}{l}
(\exists\, i : Rid;\ t : Tid;\ r : Rep;\ m : Mess \bullet \\
\qquad (t, m) \in mail_0 \\
\qquad \wedge\ mail = mail_0 - \prec(t, m)\succ \\
\qquad \wedge\ m \in ran(mp) \Rightarrow \\
\qquad\qquad (m = mp(i, r) \\
\qquad\qquad \wedge\ tcomp = tcomp_0 \cup \{(i, t, r)\}))
\end{array}\right]$$

These refinements mean that *sreq* and *tresp* will not refuse to take messages of the wrong type from *mail*. This will be important for the parallel decomposition step.

Sixth Refinement of File System

Simplify the invariant:

FS^7

$$\begin{array}{|l}
fstore : Fid \rightarrow File \\
twait : \mathbb{P}(Rid \times Tid \times Fid \times Op) \\
scomp : \mathbb{P}(Rid \times Sid \times Tid \times Rep) \\
tcomp : \mathbb{P}(Rid \times Tid \times Rep) \\
mail : \textbf{bag}\ Env \\
unused : \mathbb{P}\ Rid \\
\hline
\end{array}$$

Decomposition of File System

Decompose $FileSys^7$ to $(Terminals^1 \parallel Servers^1 \parallel MPS^1)$:

Terminals

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{TMS}^{1}} \\
\hline
\mathit{twait} : \mathbb{P}(\mathit{Rid} \times \mathit{Tid} \times \mathit{Fid} \times \mathit{Op}) \\
\mathit{tcomp} : \mathbb{P}(\mathit{Rid} \times \mathit{Tid} \times \mathit{Rep}) \\
\mathit{unused} : \mathbb{P}\ \mathit{Rid} \\
\hline
\end{array}
$$

$\mathit{Terminals}^{1}$ is defined as follows:

$$
\begin{array}{l}
\textbf{var } \mathit{TMS}^{1} \\
\textbf{initially} \left[\begin{array}{l} \mathit{twait} = \mathit{tcomp} = \{\} \\ \wedge\ \mathit{unused} = \mathit{Rid} \end{array} \right]
\end{array}
$$

$$
\begin{array}{l}
\textbf{for } t \in \mathit{Tid} \textbf{ chan } \mathit{req}_t \textbf{ in } n? : \mathit{Fid};\ p? : \mathit{Op} \textbf{ out } i! : \mathit{Rid} :- \\
\quad \begin{array}{c} \mathit{twait}, \\ \mathit{unused} \end{array} : \left[\begin{array}{l} i! \in \mathit{unused}_0 \\ \wedge\ \mathit{unused} = \mathit{unused}_0 \setminus \{i!\} \\ \wedge\ \mathit{twait} = \mathit{twait}_0 \cup \{(i!, t, n?, p?)\} \end{array} \right]
\end{array}
$$

$$
\begin{array}{l}
\textbf{for } t \in \mathit{Tid} \textbf{ chan } \mathit{resp}_t \textbf{ out } i! : \mathit{Rid};\ r! : \mathit{Rep} :- \\
\quad \mathit{comp} : \left[\begin{array}{l} (i!, t, r!) \in \mathit{tcomp}_0 \\ \wedge\ \mathit{tcomp} = \{i!\} \lhd \mathit{tcomp}_0 \end{array} \right]
\end{array}
$$

$$
\begin{array}{l}
\textbf{for } t \in \mathit{Tid} \textbf{ chan } \mathit{send}_t \textbf{ out } s! : \mathit{Node};\ m! : \mathit{Mess} :- \\
\quad \mathit{twait} : \left[\begin{array}{l} (\exists\, i : \mathit{Rid};\ n : \mathit{Fid};\ p : \mathit{Op} \bullet \\ \qquad (i, t, n, p) \in \mathit{twait}_0 \\ \qquad \wedge\ s! = \mathit{which}(n) \\ \qquad \wedge\ m! = \mathit{mq}(i, t, n, p) \\ \qquad \wedge\ \mathit{twait} = \{i\} \lhd \mathit{twait}_0) \end{array} \right]
\end{array}
$$

$$
\begin{array}{l}
\textbf{for } t \in \mathit{Tid} \textbf{ chan } \mathit{receive}_t \textbf{ in } m? : \mathit{Mess} :- \\
\quad \mathit{tcomp} : \left[\begin{array}{l} m? \in \mathit{ran}(\mathit{mp}) \Rightarrow \\ \qquad (\exists\, i : \mathit{Rid};\ r : \mathit{Rep} \mid m? = \mathit{mp}(i, r) \bullet \\ \qquad\qquad \mathit{tcomp} = \mathit{tcomp}_0 \cup \{(i, t, r)\}) \end{array} \right]
\end{array}
$$

Servers

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SVS}^{1}} \\
\hline
\mathit{fstore} : \mathit{Fid} \rightarrow \mathit{File} \\
\mathit{scomp} : \mathbb{P}(\mathit{Sid} \times \mathit{Rid} \times \mathit{Tid} \times \mathit{Rep}) \\
\hline
\end{array}
$$

$\mathit{Servers}^{1}$ is defined as follows:

$$
\begin{array}{l}
\textbf{var } \mathit{SVS}^{1} \\
\textbf{initially} \left[\begin{array}{l} \mathit{fstore} = (\lambda\, n : \mathit{Fid} \bullet \perp) \\ \wedge\ \mathit{scomp} = \{\} \end{array} \right]
\end{array}
$$

for $s \in Sid$ **chan** $receive_s$ **in** $m? : Mess$:–

$$\begin{array}{l} scomp, \\ fstore \end{array} : \left[\begin{array}{l} s \notin dom(scomp_0) \wedge \\ m? \in ran(mq) \Rightarrow \\ \quad (\exists\, i : Rid;\ t : Tid;\ n : Fid;\ p : Op;\ f : File;\ r : Rep \bullet \\ \qquad m? = mq(i, t, n, p) \\ \qquad \wedge\ (f, r) = p(fstore_0(n)) \\ \qquad \wedge\ scomp = scomp_0 \cup \{(s, i, t, r)\} \\ \qquad \wedge\ fstore = fstore_0 \oplus \{n \mapsto f\}) \end{array} \right]$$

for $s \in Sid$ **chan** $send_s$ **out** $t! : Node;\ m! : Mess$:–

$$scomp\ : \left[\begin{array}{l} (\exists\, i : Rid;\ r : Rep \bullet \\ \quad (s, i, t!, r) \in scomp_0 \\ \quad \wedge\ scomp = \{s\} \lhd scomp_0 \\ \quad \wedge\ m! = mp(i, r)) \end{array} \right]$$

Rename **internal** $treq$ of $FileSys^7$ to
for $t \in Tid$**internal** $send_t$

using Rule 7.17. Similarly rename $tresp$ to $receive_t$, $sreq$ to $receive_s$, and $sresp$ to $send_s$. Use Rule 7.11 to show

$$FileSys^7\ =\ (Terminals^1 \parallel Servers^1 \parallel MPS^1) \setminus \{send, receive\}.$$

Refinement of Terminals

Replace $twait, tcomp, unused$ with $ltwait, ltcomp, lunused$:

TMS^2

$$\begin{array}{l} ltwait : Tid \rightarrow \mathbb{P}(Rid \times Fid \times Op) \\ ltcomp : Tid \rightarrow \mathbb{P}(Rid \times Rep) \\ lunused : Tid \rightarrow \mathbb{P}\ Rid \\ \hline disjoint\ ran(lunused) \\ \wedge\ (\forall\, t : Tid \bullet lunused(t)\ \text{ is infinite}) \end{array}$$

TMS^1 and TMS^2 are related by:

$$\begin{array}{rcl} {}^1TAI^2 & \widehat{=} & TMS^2 \wedge \\ & & twait = \{\ (i, t, n, p) \mid (i, n, p) \in ltwait(t)\ \} \\ & & tcomp = \{\ (i, t, r) \mid (i, r) \in ltcomp(t)\ \} \\ & & unused = (\cup t : Tid \bullet lunused(t)) \end{array}$$

$Terminals^2$ is defined as follows:

var TMS^2

initially $\left[\begin{array}{l} (\forall\, t \in Tid \bullet ltwait(t) = ltcomp(t) = \{\}) \\ \wedge\ ran(lunused)\ partitions\ Rid \end{array} \right]$

for $t \in \mathit{Tid}$ **chan** req_t **in** $n? : \mathit{Fid};\ p? : \mathit{Op}$ **out** $i! : \mathit{Rid}$:–

$$\begin{array}{c} \mathit{ltwait}, \\ \mathit{lunused} \end{array} : \left[\begin{array}{l} i! \in \mathit{lunused}_0(t) \\ \wedge\ \mathit{lunused}(t) = \mathit{lunused}_0(t) \setminus \{i!\} \\ \wedge\ \mathit{ltwait}(t) = \mathit{ltwait}_0(t) \cup \{(i!, n?, p?)\} \end{array} \right]$$

for $t \in \mathit{Tid}$ **chan** resp_t **out** $i! : \mathit{Rid};\ r! : \mathit{Rep}$:–

$$\mathit{ltcomp} : \left[\begin{array}{l} (i!, r!) \in \mathit{ltcomp}_0(t) \\ \wedge\ \mathit{ltcomp}(t) = \mathit{ltcomp}_0(t) \setminus \{(i!, r!)\} \end{array} \right]$$

for $t \in \mathit{Tid}$ **chan** send_t **out** $s! : \mathit{Node};\ m! : \mathit{Mess}$:–

$$\mathit{ltwait} : \left[\begin{array}{l} (\exists\, i : \mathit{Rid};\ n : \mathit{Fid};\ p : \mathit{Op} \bullet \\ \qquad (i, n, p) \in \mathit{ltwait}_0(t) \\ \qquad \wedge\ s! = \mathit{which}(n) \\ \qquad \wedge\ m! = \mathit{mq}(i, t, n, p) \\ \qquad \wedge\ \mathit{ltwait}(t) = \mathit{ltwait}_0(t) \setminus \{(i, n, p)\}) \end{array} \right]$$

for $t \in \mathit{Tid}$ **chan** $\mathit{receive}_t$ **in** $m? : \mathit{Mess}$:–

$$\mathit{ltcomp} : \left[\begin{array}{l} m? \in \mathit{ran}(\mathit{mp}) \Rightarrow \\ \qquad (\exists\, i : \mathit{Rid};\ r : \mathit{Rep} \mid m? = \mathit{mp}(i, r) \bullet \\ \qquad\qquad \mathit{ltcomp}(t) = \mathit{ltcomp}_0(t) \cup \{(i, r)\}) \end{array} \right]$$

Assume *init_lunused* satisfies TMS^2. Then the invariant of $\mathit{Terminals}^2$ is simplified as follows: Replace TMS^2 with:

$$\begin{array}{|l} \hline \mathit{TMS}^3 \\ \mathit{ltwait} : \mathit{Tid} \rightarrow \mathbb{P}(\mathit{Rid} \times \mathit{Fid} \times \mathit{Op}) \\ \mathit{ltcomp} : \mathit{Tid} \rightarrow \mathbb{P}(\mathit{Rid} \times \mathit{Rep}) \\ \mathit{lunused} : \mathit{Tid} \rightarrow \mathbb{P}\ \mathit{Rid} \\ \hline \end{array}$$

Replace initialisation of $\mathit{Terminals}^2$ with:

$$\textbf{initially} \left[\begin{array}{l} (\forall\, t \in \mathit{Tid} \bullet \mathit{ltwait}(t) = \mathit{ltcomp}(t) = \{\} \\ \qquad \wedge\ \mathit{lunused}(t) = \mathit{init\_lunused}(t)) \end{array} \right]$$

Decomposition of Terminals

Decompose $\mathit{Terminals}^3$ to $(\|\ t \in \mathit{Tid} \bullet \mathit{Terminal}_t^1)$:

$$\begin{array}{|l} \hline \mathit{TM}_t^1 \\ \mathit{ltwait}_t : \mathbb{P}(\mathit{Rid} \times \mathit{Fid} \times \mathit{Op}) \\ \mathit{ltcomp}_t : \mathbb{P}(\mathit{Rid} \times \mathit{Rep}) \\ \mathit{lunused}_t : \mathbb{P}\ \mathit{Rid} \\ \hline \end{array}$$

For each $t \in Tid$, $Terminal^1_t$ is defined as follows:

$$\textbf{var } TM^1_t$$

$$\textbf{initially} \left[\begin{array}{l} ltwait_t = ltcomp_t = \{\} \\ \wedge\ lunused_t = init\_lunused(t) \end{array}\right]$$

$$\textbf{chan } req_t \textbf{ in } n? : Fid;\ p? : Op \textbf{ out } i! : Rid :-$$

$$\begin{array}{c} ltwait_t, \\ lunused_t \end{array} : \left[\begin{array}{l} i! \in (lunused_t)_0 \\ \wedge\ lunused_t = (lunused_t)_0 \setminus \{i!\} \\ \wedge\ ltwait_t = (ltwait_t)_0 \cup \{(i!, n?, p?)\} \end{array}\right]$$

$$\textbf{chan } resp_t \textbf{ out } i! : Rid;\ r! : Rep :-$$

$$ltcomp_t : \left[\begin{array}{l} (i!, r!) \in (ltcomp_t)_0 \\ \wedge\ ltcomp_t = (ltcomp_t)_0 \setminus \{(i!, r!)\} \end{array}\right]$$

$$\textbf{chan } send_t \textbf{ out } s! : Node;\ m! : Mess :-$$

$$ltwait_t : \left[\begin{array}{l} (\exists\, i : Rid;\ n : Fid;\ p : Op \bullet \\ \qquad (i, n, p) \in (ltwait_t)_0 \\ \qquad \wedge\ s! = which(n) \\ \qquad \wedge\ m! = mq(i, t, n, p) \\ \qquad \wedge\ ltwait_t = (ltwait_t)_0 \setminus \{(i, n, p)\}) \end{array}\right]$$

$$\textbf{chan } receive_t \textbf{ in } m? : Mess :-$$

$$ltcomp_t : \left[\begin{array}{l} m? \in ran(mp) \Rightarrow \\ \qquad (\exists\, i : Rid;\ r : Rep \mid m? = mp(i, r) \bullet \\ \qquad\qquad ltcomp_t = (ltcomp_t)_0 \cup \{(i, r)\}) \end{array}\right]$$

Use Rule 7.20 to show $Terminals^3 = (\| \ i \in Tid \bullet Terminal^1_t)$. Note that $ltwait_t, ltcomp_t, lunused_t$ are written respectively as $wait_t, comp_t, unused_t$ in Figure 7.L.

Refinement of Servers

Replace SVS^1 with:

$$\begin{array}{|l} \hline SVS^2 \\ \hline lfstore : Sid \rightarrow (Fid \rightarrow File) \\ b : Sid \rightarrow \textbf{bool} \\ i : Sid \rightarrow Rid \\ t : Sid \rightarrow Tid \\ r : Sid \rightarrow Rep \\ \hline \end{array}$$

SVS^1 and SVS^2 are related by:

$$\begin{aligned} {}^1SAI^2 \ \widehat{=}\ & SVS^2 \wedge \\ & fstore = (\cup s : Sid \bullet (which^{-1}(s) \lhd lfstore(s))) \\ & scomp = \{\ (s, i(s), t(s), r(s)) \mid b(s) = true\ \} \end{aligned}$$

$Servers^2$ is defined as follows:

$$\textbf{var } SVS^2$$

$$\textbf{initially} \left[\begin{array}{l} (\forall\, s \in Sid \bullet lfstore(s) = (\lambda\, n : Fid \bullet \bot) \\ \qquad \wedge\ b(s) = false) \end{array} \right]$$

for $s \in Sid$ **chan** $receive_s$ **in** $m? : Mess$:–

$$\begin{array}{r} b_s, \\ i_s, \\ t_s, \\ r_s, \\ lfstore \end{array} : \left[\begin{array}{l} (b_s)_0 = false\ \wedge \\ m? \in ran(mq) \Rightarrow \\ \qquad (\exists\, n : Fid;\ p : Op;\ f : File \bullet \\ \qquad\qquad m? = mq(i_s, t_s, n, p) \\ \qquad\qquad \wedge\ (f, r_s) = p(lfstore_0(s, n)) \\ \qquad\qquad \wedge\ lfstore(s) = lfstore_0(s) \oplus \{n \mapsto f\} \\ \qquad\qquad \wedge\ b_s = true) \end{array} \right]$$

for $s \in Sid$ **chan** $send_s$ **out** $t! : Node;\ m! : Mess$:–

$$b_s \ : \left[\begin{array}{l} (b_s)_0 = true \\ \wedge\ b_s = false \\ \wedge\ t! = t_s \\ \wedge\ m! = mp(i_s, r_s) \end{array} \right]$$

Decomposition of Servers

Decompose $Servers^2$ to $(\|\ s \in Sid \bullet Server^1_s)$:

SV^1_s

$lfstore_s : Fid \rightarrow File$
$b_s :$ **bool**
$i_s : Rid$
$t_s : Tid$
$r_s : Rep$

$Server^1_s$ is defined as follows:

$$\textbf{var } SV^1_s$$

$$\textbf{initially} \left[\begin{array}{l} lfstore_s = (\lambda\, n : Fid \bullet \bot) \\ \wedge\ b_s = false \end{array} \right]$$

chan $receive_s$ **in** $m? : Mess$:–

$$\begin{array}{r} b_s, \\ i_s, \\ t_s, \\ r_s, \\ lfstore_s \end{array} : \left[\begin{array}{l} (b_s)_0 = false\ \wedge \\ m? \in ran(mq) \Rightarrow \\ \qquad (\exists\, n : Fid;\ p : Op;\ f : File \bullet \\ \qquad\qquad m? = mq(i_s, t_s, n, p) \\ \qquad\qquad \wedge\ (f, r_s) = p((lfstore_s)_0(n)) \\ \qquad\qquad \wedge\ lfstore_s = (lfstore_s)_0 \oplus \{n \mapsto f\} \\ \qquad\qquad \wedge\ b_s = true) \end{array} \right]$$

chan $send_s$ **out** $t! : Node;\ m! : Mess$:–

$$b_s \ : \left[\begin{array}{l} (b_s)_0 = true \\ \wedge\ b_s = false \\ \wedge\ t! = t_s \\ \wedge\ m! = mp(i_s, r_s) \end{array}\right]$$

Use Rule 7.20 to show $Servers^2 \ = \ (\| \ i \in Sid \bullet Servers^1_s)$. Note that $lfstore_s$ is written $fstore_s$ in Figure 7.M.

Bibliography

[AL88] M. Abadi and L. Lamport. The existence of refinement mappings. In *3rd IEEE Symposium on LICS*, 1988.

[AL90] M. Abadi and L. Lamport. Composing specifications. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume LNCS 430. Springer–Verlag, 1990.

[AS87] B. Alpern and F.B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.

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

[Bac81] R.J.R. Back. Proving total correctness of nondeterministic programs in infinitary logic. *Acta Informatica*, 15:233–249, 1981.

[Bac90a] R.J.R. Back. Refinement calculus I: Sequential and nondeterministic programs. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume LNCS 430. Springer–Verlag, 1990.

[Bac90b] R.J.R. Back. Refinement calculus II: Parallel and reactive systems. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume LNCS 430. Springer–Verlag, 1990.

[Bac92a] R.J.R. Back. Refinement calculus, lattices and higher order logic. In *Marktoberdorf International Summer School — Program Design Calculi*, July 1992.

[Bac92b] R.J.R. Back. Refinement of parallel and reactive programs. In *Marktoberdorf International Summer School — Program Design Calculi*, July 1992.

[BK85] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[BKS83] R.J.R. Back and R. Kurki-Suonio. Decentralisation of process nets with centralised control. In *2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, pages 131–142, 1983.

[BKS90] R.J.R. Back and R. Kurki-Sounio. Superposition and fairness in reactive system refinement. In *Jerusalem Conference on Information Technology*, October 1990.

[Boo82] H.J. Boom. A weaker precondition for loops. *ACM Trans. Prog. Lang. and Sys.*, 4:668–677, Oct 1982.

[BS89] R.J.R. Back and K. Sere. Stepwise refinement of parallel algorithms. *Science of Computer Programming*, 13:133–180, 1989.

[BS90] R.J.R. Back and K. Sere. Deriving an occam implementation of action systems. In C.C. Morgan and J.C.P. Woodcock, editors, *3rd BCS-FACS Refinement Workshop*. Springer–Verlag, 1990.

[BS92] R.J.R. Back and K. Sere. Superposition refinement of parallel algorithms. In K.A. Parker and G.A. Rose, editors, *FORTE'91*. North–Holland, 1992.

[BvG92] R.C. Backhouse and A.J.M. van Gasteren. Calculating a Path Algorithm. In *Mathematics of Program Construction*, Oxford, 1992.

[CM88] K.M. Chandy and J. Misra. *Parallel Program Design – A Foundation*. Addison–Wesley, 1988.

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

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

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification I*. Springer–Verlag, 1985.

[End77] H.B. Enderton. *Elements of Set Theory*. Academic Press, 1977.

[FF90] N. Francez and I.R. Forman. Superimposition for interacting processes. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR'90*, volume LNCS 458. Springer–Verlag, 1990.

[Fra86] N. Francez. *Fairness*. Springer–Verlag, 1986.

[GM88] P.H.B. Gardiner and C.C. Morgan. Data refinement of predicate transformers. Submitted to *Sci. Comp. Prog.*. Reprinted in [MRG88], February 1988.

[GM89] P.H.B. Gardiner and C.C. Morgan. A single complete rule for data refinement. Technical Report TR-7-89, Programming Research Group, Oxford University, November 1989.

[Gol91] K.J. Goldman. A compositional model for layered distributed systems. In J.C.M. Baeten and J.F. Groote, editors, *CONCUR'91*, volume LNCS 527. Springer–Verlag, 1991.

[Ham78] A.G. Hamilton. *Logic for Mathematicians*. Cambridge University Press, 1978.

[He89] J. He. Process refinement. In J. McDermid, editor, *The Theory and Practice of Refinement*. Butterworths, 1989.

[He90] J. He. Various simulations and refinements. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume LNCS 430. Springer–Verlag, 1990.

[HHS86] J. He, C.A.R. Hoare, and J.W. Sanders. Data refinement refined. In *European Symposium on Programming*, volume LNCS 213. Springer–Verlag, 1986.

[HJ92] I.S.C. Houston and M.B. Josephs. Specifying distributed CICS in Z: accessing local and remote resources. Submitted to *Formal Aspects of Computing*, 1992.

[HMvdS90] H.P. Hofstee, A.J. Martin, and J.L.A. van de Snepscheut. Distributed sorting. *Science of Computer Programming*, 15:119–133, 1990.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice–Hall, 1985.

[Hof92] H.P. Hofstee. Distributing a Class of Sequential Programs. In *Mathematics of Program Construction*, Oxford, 1992.

[HP72] P. Hitchcock and D. Park. Induction rules and termination proofs. In *IRIA Conference on Automata Languages and Programming Theory*, France, 1972.

[HS88] J. Henshall and S. Shaw. *OSI Explained*. Ellis–Horwood, 1988.

[JG88] G. Jones and M. Goldsmith. *Programming in occam 2*. Prentice–Hall, 1988.

[Jon83] C.B. Jones. Specification and design of (parallel) programs. In R.E.A. Mason, editor, *Information Processing '83*. IFIP, North Holland, 1983.

[Jon86] C.B. Jones. *Systematic Software Development using VDM*. Prentice–Hall, 1986.

[Jos88] M.B. Josephs. A state-based approach to communicating sequential processes. *Distributed Computing*, 3:9–18, 1988.

[Jos91] M.B. Josephs. Specifying Reactive Systems in Z. Technical Report PRG-TR-19-91, Programming Research Group, University of Oxford, 1991.

[Jss90] B. Jonsson. On decomposing and refining specifications of distributed systems. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume LNCS 430. Springer–Verlag, 1990.

[KSK88] R. Kurki-Suonio and T. Kankaanpää. On the design of reactive systems. *BIT*, 28:581–604, 1988.

[Lam83] L. Lamport. What good is temporal logic? In R.E.A. Mason, editor, *Information Processing '83*. IFIP, North Holland, 1983.

[Lam89] L. Lamport. A simple approach to specifying concurrent systems. *Comms. of the ACM*, 32(1):32–45, 1989.

[Len88] P.M. Lenders. A generalised message-passing mechanism for CSP. *IEEE Trans. Comp.*, 37(6):646–651, June 1988.

[LT87] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *6th ACM Symp. on Principles of Distributed Computing*, pages 137–151, 1987.

[Lyn90] N.A. Lynch. Multivalued possibilities mappings. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume LNCS 430. Springer–Verlag, 1990.

[Mar81] A.J. Martin. An axiomatic definition of synchronisation primitives. *Acta Informatica*, 16:219–235, 1981.

[Mar85] A.J. Martin. The probe: An addition to communications primitives. *Information Processing Letters*, 20:125–130, 1985.

[Mce91] C.E. Martin. *Preordered Categories and Predicate Transformers.* D.Phil. Thesis, Programming Research Group, Oxford University, 1991.

[MG88] C.C. Morgan and P.H.B. Gardiner. Data refinement by calculation. Submitted to *Acta Informatica.* Reprinted in [MRG88], July 1988.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice–Hall, 1989.

[Min67] M.L. Minsky. *Computation: Finite and Infinite Machines.* Prentice–Hall, 1967.

[Mor79] C.C. Morgan. *Parallel Programming Without Synchronisation.* Ph.D. Thesis, University of Sydney, 1979.

[Mor88a] C.C. Morgan. Procedures, parameters, and abstraction: Separate concerns. *Sci. Comp. Prog.*, 11, 1988. Reprinted in [MRG88].

[Mor88b] C.C. Morgan. The specification statement. *ACM Trans. Prog. Lang. and Sys.*, 10(3), 1988. Reprinted in [MRG88].

[Mor89] C.C. Morgan. Types and invariants in the refinement calculus. In J.L.A. van de Snepsheut, editor, *Mathematics of Program Construction*, volume LNCS 375. Springer-Verlag, 1989.

[Mor90a] C.C. Morgan. Of wp and CSP. In W.H.J. Feijen, A.J.M. van Gasteren, D. Gries, and J. Misra, editors, *Beauty is our business: a birthday salute to Edsger W. Dijkstra.* Springer–Verlag, 1990.

[Mor90b] C.C. Morgan. *Programming from Specifications.* Prentice–Hall, 1990.

[MP84] Z. Manna and A. Pnueli. Adequate proof principles for invariances and liveness. *Science of Computer Programming*, 4:257–289, 1984.

[MRG88] C.C. Morgan, K.A. Robinson, and P.H.B. Gardiner. *On the Refinement Calculus.* Technical Monograph PRG–70, Programming Research Group, Oxford University, October 1988.

[Mrr87] J.M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Sci. Comp. Prog.*, 9(3):298–306, 1987.

[Mrr89] J.M. Morris. Laws of data refinement. *Acta Informatica*, 26:287–308, 1989.

[MS87] C.C. Morgan and B.A. Sufrin. Specification of the UNIX filing system. In I.J. Hayes, editor, *Specification Case Studies.* Prentice–Hall, 1987.

[Nel89] G. Nelson. A generalization of Dijkstra's calculus. *ACM Trans. Prog. Lang. Sys.*, 11(4):517–561, 1989.

[Nel92] G. Nelson. Notes on complier correctness. Unpublished presentation to *Mathematics of Program Construction* conference, Oxford, July 1992.

[OG76] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6:319–340, 1976.

[OH86] E.-R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986.

[Old91] E.-R. Olderog. Towards a design calculus for communicating programs. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings CONCUR '91*, volume LNCS 527. Springer-Verlag, 1991.

[Ore62] O. Ore. *Theory of Graphs*, volume XXXVIII of *American Math. Soc. Colloquium Publications*. American Mathematical Society, 1962.

[Pnu86] A. Pnueli. Specification and development of reactive systems. In H.-J. Kugler, editor, *Information Processing '86*. IFIP, North Holland, 1986.

[Ros88] A.W. Roscoe. *Unbounded Nondeterminism in CSP*. Technical Monograph PRG-67, Programming Research Group, Oxford University, July, 1988.

[RR86] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. In E. Shapiro, editor, *Automata, Languages and Programming*, volume LNCS 226. Springer–Verlag, 1986.

[Sei92] K. Seidel. *Probabilistic Communicating Processes*. D.Phil. Thesis, Programming Research Group, Oxford University, 1992.

[Ser91] K. Sere. Stepwise refinement of reactive processor farms. Åbo Akademi University, Finland, January 1991.

[Spi89] J.M. Spivey. *The Z Notation - A Reference Manual*. Prentice–Hall, 1989.

[Sta88] M.G. Staskauskas. The formal specification and design of a distributed electronic funds-transfer system. *IEEE Trans. on Comp.*, 37(12):1515–1528, 1988.

[Sti88] C. Stirling. A Generalisation of Owicki-Gries's Hoare Logic for a Concurrent While Langauge. *Theoretical Computer Science*, 58:347–359, 1988.

[Stø91] K. Stølen. An attempt to reason about shared-state concurrency in the style of VDM. In S. Prehn and W.J. Toetenel, editors, *VDM '91*, volume LNCS 551. Springer–Verlag, October 1991.

[Tar55] A. Tarski. A lattice-theoretic fixpoint theorem and its applications. *Pacific Journal Math.*, 5, 1955.

[vEVD89] P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors. *The Formal Description Technique LOTOS*. North–Holland, 1989. Results of ESPRIT/SEDOS Project.

[vG92] A.J.M. van Gasteren. Private communication, July 1992.

[WM90] J.C.P. Woodcock and C.C. Morgan. Refinement of state-based concurrent systems. In D. Bjørner, C.A.R. Hoare, and H. Langmaack, editors, *VDM '90*, volume LNCS 428. Springer–Verlag, 1990.

[WWF87] D.A. Watt, B.A. Wichmann, and W. Findlay. *ADA: Language and Methodology.* Prentice–Hall, 1987.

[XH91] Q. Xu and J. He. A theory of state-based parallel programming by refinement: Part I. In J. Morris and R.C. Shaw, editors, *4th BCS-FACS Refinement Workshop.* Springer–Verlag, 1991.