Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 214 (2008) 51–79
www.elsevier.com/locate/entcs

An Algebraic Approach to Refinement with Fair Choice

Emil Sekerinski[1]

Department of Computing and Software
McMaster University
Hamilton, Ontario, Canada

Abstract

In the analysis and design of concurrent systems, it can be useful to assume fairness among processes. Action systems model a process by a set of atomic actions. Typically, actions are combined by nondeterministic choice, which models minimal progress among processes rather than fairness. Here we define an operator for the fair choice among a set of actions. A refinement rule for action systems with fair choice is derived and applied to the development of the alternating bit protocol. The novelty is the algebraic style in which the fair choice operator is defined and in which formal reasoning is carried out; it avoids an appeal to the operational understanding of fairness.

Keywords: Refinement, fairness, action systems, alternating bit protocol

1 Introduction

In the action system model, a concurrent system is described through a set of atomic actions. Concurrency is modeled through interleaving: two actions that can be executed in any order, can be executed concurrently, and thus can belong to different processes. A concurrent system is understood through the repeated selection and execution of atomic actions. Fairness is a property that restricts this selection: weak fairness requires that a continuously enabled action is infinitely often taken. This is a useful assumption. If two continuously enabled actions belong to different processes, fairness implies that the scheduler must give each process a chance, without specifying the scheduling policy;

[1] Email: emil@mcmaster.ca

doi:10.1016/j.entcs.2008.06.004

by contrast, the minimal progress assumption would only ensure that either one makes progress. If two continuously enabled actions are to be executed on different processors, fairness expresses that each processor is working, without quantifying the relative speed. If two continuously enabled actions model possible behavior of the environment, like successful and failed transmission over a medium, then fairness implies that each behavior is possible, without quantifying the probability. Fairness has become a common assumption in the analysis and development of concurrent systems. Programming theories involving fairness are well worked out, e.g. [8,10,11].

This work is on the stepwise refinement of action systems with fair choice. We define an operator for the fair choice among actions and define an action system as a loop with a body composed by fair choice, following the approach of defining an action system as a loop with a body composed by nondeterministic choice. In particular, we make use of strong and weak iteration constructs of [1,3,7] for defining loops and fair choice, resulting in an algebraic treatment of fairness. No appeal to operational reasoning in terms of traces of executions is needed. A theorem for the data refinement of action systems with fair choice is derived and applied to the development of the alternating bit protocol.

This paper extends the binary fair choice of [14] to an n-ary fair choice and provides refinement rules to complement the verification rules studied there. The rule for data refinement of fair action system is similar to that is suggested in [4]; this paper presents a justification of this rule by reduction to the standard data refinement rule for action systems.

In [15] refinement rules that preserve temporal (leads-to) and fixpoint (termination) properties are studied for fair transitions systems (action systems). Here we restrict ourselves to terminating action systems but consider local variables, allowing a more general notion of refinement.

The approach of [17] is to augment action systems with an explicit specification of unfair non-terminating computations, rather than assuming a fair choice among actions, and to study refinement of such augmented action systems; this allows a wider range of fairness constraints to be expressed compared to the (weak) fairness considered here, though in a different style.

In [13] Dijkstra's calculus (action systems) is extended by a fair choice operator. The approach relies on temporal predicate transformers like "always" and "eventually" and on syntactic substitutions of fair choice by angelic choice, neither of which is needed here.

In [6] Dijkstra's calculus is also extended by a fair choice operator in terms of the dovetail operator $\bigtriangledown$ that models fair parallel execution. The definition of dovetail requires the distinction between possible and definite nontermination, which is done by additionally considering weakest liberal precondi-

tions. The expressiveness of the dovetail operator leads to problems with non-monotonicity and to the need for two ordering relations, both of which is avoided here.

Predicate transformers are used as the model of statements because of their generality. The next section reviews the predicate transformer model of statements and iteration statements defined by fixed points. Data refinement of statements follows common treatment [2,12].

2 Statements

We use typed, higher-order logic for defining statements, following [2]. Function application is written as $f\ x$ and binds tighter than any other operator. Equivalence ($\equiv$) has the same meaning on boolean expressions as equality ($=$), except that $=$, like $\leq$ bind tighter than $\wedge, \vee$, which in turn bind tighter than $\equiv$.

2.1 State Predicates

State predicates of type $\mathcal{P}\Sigma$ are functions from elements of type Σ to *Bool*, i.e. $\mathcal{P}\Sigma = \Sigma \to \mathit{Bool}$. On state predicates, conjunction $\wedge$, disjunction $\vee$, implication $\Rightarrow$, and negation $\neg$ are defined by the pointwise extension of the corresponding operations on *Bool*. Likewise, universal and existential quantification of $p_i : \mathcal{P}\Sigma$ are defined by $(\forall\, i \in I \bullet p_i)\,\sigma \equiv (\forall\, i \in I \bullet p_i\,\sigma)$ and $(\exists\, i \in I \bullet p_i)\,\sigma \equiv (\exists\, i \in I \bullet p_i\,\sigma)$. The entailment ordering $\leq$ is defined by universal implication. The state predicates *true* and *false* represent the universally *true* and *false* predicates, respectively.

2.2 Predicate Transformers

Predicate transformers of type $\Delta \mapsto \Omega$ are functions from predicates over Ω, the postconditions, to predicates over Δ, the preconditions, $\Delta \mapsto \Omega = \mathcal{P}\Omega \to \mathcal{P}\Delta$. A predicate transformers S is called monotonic if $p \leq q$ implies $S\ p \leq S\ q$ for any (state) predicates p and q. We use monotonic predicate transformers to model statements.

The sequential composition of predicate transformers S and T is defined by their functional composition:

$$(S\ ;\ T)\ q \quad \widehat{=} \quad S\ (T\ q)$$

The guard $[p]$ skips if p holds and establishes "miraculously" any postcondition if p does not hold (by blocking execution). The assertion $\{p\}$ skips if p holds

and establishes no postcondition if p does not hold (the system crashes):

$$\begin{array}{rcl} [p]\ q & \widehat{=} & p \Rightarrow q \\ \{p\}\ q & \widehat{=} & p \wedge q \end{array}$$

We define $skip = [true] = \{true\}$ as the identity predicate transformer, $magic = [false]$ as the predicate transformer which always establishes any postcondition, and $abort = \{false\}$ as the predicate transformer which always aborts.

The demonic (nondeterministic) choice $\sqcap$ establishes a postcondition only if both alternatives do. The angelic choice $\sqcup$ establishes a certain postcondition if at least one alternative does.

$$\begin{array}{rcl} (S \sqcap T)\ q & \widehat{=} & S\ q \wedge T\ q \\ (S \sqcup T)\ q & \widehat{=} & S\ q \vee T\ q \end{array}$$

Relations of type $\Delta \leftrightarrow \Omega$ are functions from Δ to predicates over Ω. The relational updates $[R]$ and $\{R\}$ both update the state according to relation R. If several final states are possible, then $[R]$ chooses one demonically and $\{R\}$ chooses one angelically. If R is of type $\Delta \leftrightarrow \Omega$, then $[R]$ and $\{R\}$ are of type $\Delta \mapsto \Omega$:

$$\begin{array}{rcl} [R]\ q\ \delta & \widehat{=} & (\forall\, \omega \bullet R\ \delta\ \omega \Rightarrow q\ \omega) \\ \{R\}\ q\ \delta & \widehat{=} & (\exists\, \omega \bullet R\ \delta\ \omega \wedge q\ \omega) \end{array}$$

The predicate transformers $[p]$, $\{p\}$, $[R]$, $\{R\}$ are all monotonic and the operators $;$, $\sqcap$, $\sqcup$ preserve monotonicity. A predicate transformer S and is called conjunctive if $S(\forall\, i \in I \bullet q_i) = (\forall\, i \in I \bullet S\ q_i)$ for any indexed set $p_i, i \in I$ of predicates and non-empty set I. All conjunctive predicate transformers are monotonic. The predicate transfomers $[p]$, $\{p\}$, $[R]$ are all conjunctive (but not $\{R\}$) and the operators $;$, $\sqcap$ preserve conjunctivity (but not $\sqcup$). For the distributivity of $;$ over $\sqcap$ we have:

$$(\sqcap i \in I \bullet S_i)\ ;\ T = \sqcap i \in I \bullet S_i\ ;\ T \tag{1}$$

$$T\ ;\ (\sqcap i \in I \bullet S_i) \sqsubseteq \sqcap i \in I \bullet T\ ;\ S_i \tag{2}$$

$$T\ ;\ (\sqcap i \in I \bullet S_i) = \sqcap i \in I \bullet T\ ;\ S_i \quad \text{if } T \text{ conjunctive} \tag{3}$$

Other statements can be defined in terms of the above ones. For example the guarded statement $p \rightarrow S$ is defined by $[p]\ ;\ S$ and the conditional by:

$$\textbf{if } p \textbf{ then } S \textbf{ else } T \quad \widehat{=} \quad (p \rightarrow S) \sqcap (\neg p \rightarrow T)$$

The enabledness domain of a statement S is defined as $en\,S = \neg S\,false$ and its termination domain as $tr\,S = S\,true$. We have that:

$$S = [en\,S]\,;\,S \tag{4}$$
$$en(S \sqcap T) = en\,S \vee en\,T \tag{5}$$
$$en\,T = true \quad \Rightarrow \quad en(S\,;\,T) = en\,S \tag{6}$$

2.3 Refinement Ordering

The refinement ordering $\sqsubseteq$ is defined by universal entailment:

$$S \sqsubseteq T \quad \widehat{=} \quad (\forall\, q \bullet S\ q \leq T\ q)$$

With this ordering, the monotonic predicate transformers form a complete boolean lattice, with top *magic*, bottom *abort*, meet $\sqcap$, and join $\sqcup$. Intuitively, refinement can increase the termination domain, decrease the enabledness domain, decrease demonic nondeterminism, and increase angelic nondeterminism. According to the Theorem of Knaster-Tarski, any monotonic function f from predicate transformers to predicate transformers has a unique least fixed point $\mu\,f$ and a unique greatest fixed point $\nu\,f$, also written as $\mu\,x \bullet f\,x$ and $\nu\,s \bullet f\,s$, respectively.

2.4 Iterations

Iteration of a statement S is described through solutions of the equation $X = S\,;\,X \sqcap skip$. We define two fundamental iteration constructs, the strong iteration S^ω and the weak iteration S^* as the smallest and largest such solution (both of which exist as ; and $\sqcap$ are monotonic in both operands). We use the convention that ; binds tighter than $\sqcap$:

$$S^\omega \quad \widehat{=} \quad (\mu\,X \bullet S\,;\,X \sqcap skip)$$
$$S^* \quad \widehat{=} \quad (\nu\,X \bullet S\,;\,X \sqcap skip)$$

Both define a demonically chosen number of repetitions of S. However, with S^* the number of repetitions is always finite whereas with S^ω it can be infinite, which is equivalent to abortion. For example, if S is $a := a + 1$, then the equation $X = a := a + 1\,;\,X \sqcap skip$ has two solutions, *abort* and $skip \sqcap a := a + 1 \sqcap a := a + 2 \sqcap \ldots$. The least solution is given by their demonic choice. As $abort \sqcap Q = abort$ for any Q, we have that $(a := a + 1)^\omega = abort$. The greatest solution is given by their angelic choice. As $abort \sqcup Q = Q$ for any Q, we have that $(a := a + 1)^* = skip \sqcap a := a + 1 \sqcap a := a + 2 \sqcap \ldots$.

From the fixed point definitions we get the following laws for unfolding iterations:

$$S^\omega = S \,;\, S^\omega \sqcap \mathit{skip} \tag{7}$$
$$S^* = S \,;\, S^* \sqcap \mathit{skip} \tag{8}$$

Since S^ω and S^* are defined as the smallest and largest solutions, we have following induction principles:

$$S \,;\, X \sqcap \mathit{skip} \sqsubseteq X \;\Rightarrow\; S^\omega \sqsubseteq X \tag{9}$$
$$X \sqsubseteq S \,;\, X \sqcap \mathit{skip} \;\Rightarrow\; X \sqsubseteq S^* \tag{10}$$

Both weak and strong iteration are monotonic in the sense that $S \sqsubseteq T$ implies $S^\omega \sqsubseteq T^\omega$ and $S^* \sqsubseteq T^*$. Both S^ω and S^* are refined by S itself:

$$S^\omega \sqsubseteq S \tag{11}$$
$$S^* \sqsubseteq S \tag{12}$$

Furthermore, from the two unfolding laws we get immediately (as $S = T \sqcap U$ implies $S \sqsubseteq T$ for any S, T) that both are refined by *skip*:

$$S^\omega \sqsubseteq \mathit{skip} \tag{13}$$
$$S^* \sqsubseteq \mathit{skip} \tag{14}$$

For the nested application of weak and strong iteration we have:

$$(S^\omega)^* \sqsubseteq S^\omega \tag{15}$$
$$(S^*)^* \sqsubseteq S^* \tag{16}$$

However, we note that $(S^\omega)^\omega = \mathit{abort}$ and $(S^*)^\omega = \mathit{abort}$. Intuitively, the inner iteration is refined by *skip*, which then makes $\mathit{skip}^\omega = \mathit{abort}$. For the sequential composition with weak iteration we have [3]:

$$S^* \,;\, S^* = S^* \tag{17}$$
$$S^* \,;\, S = S \,;\, S^* \tag{18}$$

We introduce a derived iteration construct, the positive weak iteration S^+:

$$S^+ \;\widehat{=}\; S \,;\, S^*$$

Positive weak iteration is also monotonic in the sense that $S \sqsubseteq T$ implies $S^+ \sqsubseteq T^+$, which follows from the monotonicity of weak iteration and sequential

composition (in both arguments). Furthermore, S^+ is refined by S itself:

$$S^+ \quad \sqsubseteq \quad S \tag{19}$$

This follows from the definition of S^+ and (14). Weak iteration can also be expressed in terms of positive weak iteration:

$$S^* \quad = \quad S^+ \sqcap \mathit{skip} \tag{20}$$

This follows immediately from the unfolding law (8) and the definition of S^+. A consequence of this is that S^* is refined by S^+:

$$S^* \quad \sqsubseteq \quad S^+ \tag{21}$$

For the nested applications of positive weak iterations with weak iteration and strong iteration we get:

$$(S^+)^* \quad = \quad S^* \tag{22}$$
$$(S^+)^\omega \quad = \quad S^\omega \tag{23}$$

We show the first one by mutual refinement: $(S^+)^* \sqsubseteq S^*$ holds by (19) and monotonicity of weak iteration. For the refinement $S^* \sqsubseteq (S^+)^*$ we note that the left side is equal to $(S^*)^*$ by (16), hence this is implied by (21). For the sequential composition with positive weak iteration we have:

$$S^+ ; S \quad = \quad S ; S^+ \tag{24}$$

This follows directly from the definition of S^+ and (18). For the enabledness domain of the iteration constructs we get:

$$\mathit{en}\ S^\omega \quad = \quad \mathit{true} \tag{25}$$
$$\mathit{en}\ S^* \quad = \quad \mathit{true} \tag{26}$$
$$\mathit{en}\ S^+ \quad = \quad \mathit{en}\ S \tag{27}$$

The first two follow immediately from the unfolding laws (7) and (8) as $\mathit{en}\ \mathit{skip} = \mathit{true}$. The last one follows easily from the definition of S^+, (26) and (6). For the weak iteration of guards and asserts we have:

$$[p]^* \quad = \quad [p] \sqcap \mathit{skip} \tag{28}$$
$$\{p\}^* \quad = \quad \{p\} \sqcap \mathit{skip} \tag{29}$$
$$[p]^+ \quad = \quad [p] \tag{30}$$
$$\{p\}^+ \quad = \quad \{p\} \tag{31}$$

We prove the first two by mutual refinement. The direction $[p]^* \sqsubseteq [p] \sqcap \mathit{skip}$ follows from (12). For $[p] \sqcap \mathit{skip} \sqsubseteq [p]^*$ we have:

$$
\begin{array}{ll}
 & [p] \sqcap \mathit{skip} \sqsubseteq [p]^* \\
\Leftarrow & \quad \langle (10) \rangle \\
 & [p] \sqcap \mathit{skip} \sqsubseteq [p] \,;\, ([p] \sqcap \mathit{skip}) \sqcap \mathit{skip} \\
\equiv & \quad \langle [p] \text{ conjunctive} \rangle \\
 & [p] \sqcap \mathit{skip} \sqsubseteq [p] \,;\, [p] \sqcap [p] \,;\, \mathit{skip} \sqcap \mathit{skip} \\
\equiv & \quad \langle \text{for any } p\text{: } [p] \,;\, [p] = [p] \rangle \\
 & \mathit{true}
\end{array}
$$

The proof of (29) is analogous. Property (27) follows from the definition of S^+ and (27), and (31) follows similarly.

The following property is known as decomposition [3]. Let S, T be monotonic predicate transformer assume T is conjunctive:

$$(S \sqcap T)^\omega = S^\omega \,;\, (T \,;\, S^\omega)^\omega \tag{32}$$
$$(S \sqcap T)^* = S^* \,;\, (T \,;\, S^*)^* \tag{33}$$

A statement S disables itself if executing it once leads to a state in which S is disabled, formally $S(\neg\, en\, S) = \mathit{true}$. This can be also expressed "more algebraically" without referring to pre- and postconditions:

$$S(\neg\, en\, S) = \mathit{true} \quad \equiv \quad S \,;\, [en\, S] = \mathit{magic} \tag{34}$$

We prove this in an equational style:

$$
\begin{array}{ll}
 & S \,;\, [en\, S] = \mathit{magic} \\
\equiv & \quad \langle \text{equality of functions, definition of } ; \rangle \\
 & \forall q \bullet S([en\, S]q) = \mathit{magic} q \\
\equiv & \quad \langle \text{definitions of } \mathit{magic}, \text{ guard} \rangle \\
 & \forall q \bullet S(en\, S \Rightarrow q) = \mathit{true} \\
\equiv & \quad \langle \text{as } S \text{ monotonic} \rangle \\
 & S(\neg\, en\, S) = \mathit{true}
\end{array}
$$

If S disables itself, then S^ω and S^* execute S at most once, and S^+ executes S exactly once. Assume S is continuous:

$$S \,;\, [en\, S] = \mathit{magic} \quad \Rightarrow \quad S^\omega = S \sqcap \mathit{skip} \tag{35}$$
$$S \,;\, [en\, S] = \mathit{magic} \quad \Rightarrow \quad S^* = S \sqcap \mathit{skip} \tag{36}$$
$$S \,;\, [en\, S] = \mathit{magic} \quad \Rightarrow \quad S^+ = S \tag{37}$$

For the proof of (35) we assume $S \,;\, [en\, S] = \mathit{magic}$ and continue:

$$
\begin{array}{ll}
 & S^\omega \\
= & \quad \langle (7) \text{ twice} \rangle \\
 & S \,;\, (S \,;\, S^\omega \sqcap \mathit{skip}) \sqcap \mathit{skip} \\
= & \quad \langle S \text{ continuous, } ; \text{ distributes over } \sqcap, \mathit{skip} \text{ unit of } ; \rangle \\
 & S \,;\, S \,;\, S^\omega \sqcap S \sqcap \mathit{skip} \\
= & \quad \langle (4) \rangle
\end{array}
$$

$$
\begin{array}{ll}
 & S \,;\, [\mathit{en}\, S] \,;\, S \,;\, S^\omega \sqcap S \sqcap \mathit{skip} \\
= & \quad \langle \text{assumption} \rangle \\
 & \mathit{magic} \,;\, S \,;\, S^\omega \sqcap S \sqcap \mathit{skip} \\
= & \quad \langle \text{for any } S\text{: } \mathit{magic} \,;\, S = \mathit{magic}, \text{ lattice property} \rangle \\
 & S \sqcap \mathit{skip}
\end{array}
$$

The proof of (36) is analogous. For the proof of (37) we assume $S \,;\, [\mathit{en}\, S] = \mathit{magic}$ and continue:

$$
\begin{array}{ll}
 & S^+ \\
= & \quad \langle \text{definition of } S^+ \rangle \\
 & S \,;\, S^* \\
= & \quad \langle \text{assumption, } (36) \rangle \\
 & S \,;\, (S \sqcap \mathit{skip}) \\
= & \quad \langle S \text{ continuous, } ; \text{ distributes over } \sqcap, \mathit{skip} \text{ unit of } ; \rangle \\
 & S \,;\, S \sqcap S \\
= & \quad \langle (4) \rangle \\
 & S \,;\, [\mathit{en}\, S] \,;\, S \sqcap S \\
= & \quad \langle \text{assumption} \rangle \\
 & \mathit{magic} \,;\, S \sqcap S \\
= & \quad \langle \text{for any } S\text{: } \mathit{magic} \,;\, S = \mathit{magic}, \text{ lattice property} \rangle \\
 & S
\end{array}
$$

A statement S is always enabled if its guard is always true, formally $\mathit{true} = \mathit{en}\, S$. This can be also expressed "more algebraically" without referring to pre- and postconditions:

$$\mathit{en}\, S = \mathit{true} \quad \equiv \quad S \,;\, \mathit{abort} = \mathit{abort} \tag{38}$$

Intuitively, as *magic* is a left zero of ;, *abort* can be a right zero only if the left operand is not miraculous. This is shown by:

$$
\begin{array}{ll}
& S \,;\, abort = abort \\
\equiv & \quad \langle \text{equality of functions, definition of } ; \rangle \\
& \forall q \bullet S(abort q) = abort q \\
\equiv & \quad \langle \text{definition of } abort \rangle \\
& S(false) = false \\
\equiv & \quad \langle \text{definition of } en \rangle \\
& en\ S = true
\end{array}
$$

A variation of above observation arises when considering that statement S does not disable itself, formally $S(\neg\, en\ S) = false$:

$$S(\neg\, en\ S) = false \quad \equiv \quad S \,;\, \{\neg\, en\ S\} = \{\neg\, en\ S\} \tag{39}$$

The proof is:

$$
\begin{array}{ll}
& S \,;\, \{\neg\, en\ S\} = \{\neg\, en\ S\} \\
\equiv & \quad \langle \text{equality of functions, definition of } ; \rangle \\
& \forall q \bullet S(\{\neg\, en\ S\} q) = \{\neg\, en\ S\} q \\
\equiv & \quad \langle \text{definition of } \{p\},\ en \rangle \\
& \forall q \bullet S(S\ false \wedge q) = S\ false \wedge q \\
\equiv & \quad \langle (*) \rangle \\
& S(S\ false) = false \\
\equiv & \quad \langle \text{definition of } en \rangle \\
& S(\neg\, en\ S) = false
\end{array}
$$

The step (*) is shown by mutual implication: for "$\Rightarrow$" we instantiate q with $false$, which gives $S\ false = false$, and we instantiate q with $true$, which gives $S(S\ false) = S\ false$, which together give $S(S\ false) = false$. For "$\Leftarrow$" we show that $S\ false = false$ assuming $S(S\ false) = false$. Without loss of generality, we let $S\ false = r$. With the assumption, we have $S(r) = false$. Now, by monotonicity of S, from $false \leq r$ we get $r \leq$ false, and hence $r = false$ and therefore $S\ false = false$, from which $\forall q \bullet S(S\ false \wedge q) = S\ false \wedge q$ follows.

The loop $\mathbf{do}\ S\ \mathbf{od}$ executes its body as long as it is enabled, possibly not terminating (i.e. aborting). This is formally expressed as a strong iteration, followed by a guard which ensures that the guard of the body will not hold at exit:

$$\mathbf{do}\ S\ \mathbf{od} \quad \widehat{=} \quad S^\omega \,;\, [\neg\, en\ S]$$

Strong iteration S^ω and weak iteration S^* are the same if S eventually disables itself [3]:

$$S^\omega = \{tr(\mathbf{do}\ S\ \mathbf{od})\} \,;\, S^* \tag{40}$$

Our interest is in terminating loops; all non-terminating loops are equal to *abort*. In particular, if the body is always enabled, the loop is aborting:

$$S\,;\,abort = abort \quad \Rightarrow \quad \mathbf{do}\,S\,\mathbf{od} = abort \tag{41}$$

A generalization of above observation arises if the loop body does not disable itself. In that case, if the body of the loop is initially disabled, the loop terminates immediately. If the body is initially enabled, the loop does not terminate:

$$S\,;\,\{\neg\, en\, S\} = \{\neg\, en\, S\} \quad \Rightarrow \quad \mathbf{do}\,S\,\mathbf{od} = \{\neg\, en\, S\} \tag{42}$$

Assuming $S\,;\{\neg\, en\, S\} = \{\neg\, en\, S\}$, we show the consequence by mutual refinement. For $\{\neg\, en\, S\} \sqsubseteq \mathbf{do}\,S\,\mathbf{od}$ we make a case analysis: if $en\, S$ holds initially, then $\{\neg\, en\, S\} = abort$ and the refinement holds as *abort* is the bottom of the lattice. If $\neg\, en\, S$ holds initially, then $\{\neg\, en\, S\} = skip$ and by definition of $\mathbf{do}\,S\,\mathbf{od}$ and (7) the loop also simplifies to *skip*, hence refinement holds. For the other direction we assume $S\,;\,\{\neg\, en\, S\} = \{\neg\, en\, S\}$ and continue:

$$\begin{array}{ll}
 & \mathbf{do}\,S\,\mathbf{od} \sqsubseteq \{\neg\, en\, S\} \\
\equiv & \quad \langle \text{definition of } \mathbf{do}\,S\,\mathbf{od} \rangle \\
 & S^\omega\,;\,[\neg\, en\, S] \sqsubseteq \{\neg\, en\, S\} \\
\Leftarrow & \quad \langle \text{monotonicity of ;, for any } p\text{: } \{p\}\,;\,[p] = \{p\} \rangle \\
 & S^\omega \sqsubseteq \{\neg\, en\, S\} \\
\Leftarrow & \quad \langle (9) \rangle \\
 & S\,;\,\{\neg\, en\, S\} \sqcap skip \sqsubseteq \{\neg\, en\, S\} \\
\equiv & \quad \langle \text{assumption} \rangle \\
 & \{\neg\, en\, S\} \sqcap skip \sqsubseteq \{\neg\, en\, S\} \\
\equiv & \quad \langle \text{lattice structure} \rangle \\
 & true
\end{array}$$

Refining the body of a loop leads to the loop being refined, provided that the enabledness domain is not decreased, as this would otherwise lead to nontermination that was not originally present:

$$S \sqsubseteq T \quad \wedge \quad en\, S \leq en\, T \quad \Rightarrow \quad \mathbf{do}\,S\,\mathbf{od} \sqsubseteq \mathbf{do}\,T\,\mathbf{od} \tag{43}$$

This follows from the definition of $\mathbf{do}\,S\,\mathbf{od}$, monotonicity of S^ω, and the property that $[p] \sqsubseteq [q] \equiv p \geq q$. The while loop $\mathbf{while}\,b\,\mathbf{do}\,S$ can be defined as $\mathbf{do}\,b \rightarrow S\,\mathbf{od}$, provided S is always enabled. An action system is loop of the form:

$$\mathbf{do}\,S_1 \sqcap \cdots \sqcap S_n\,\mathbf{od}$$

The statements S_i are called the actions and are typically of the form $g_i \to B_i$, where g_i is the guard and B_i is the (always enabled) body. In this form, the choice among the actions is nondeterministic (demonic); no fairness in the selection of the actions is guaranteed.

2.5 Program Variables

The state space is made up of a number of program variables. Thus the state space is of the form $\Gamma = \Gamma_1 \times \ldots \times \Gamma_n$ and states are tuples $\gamma = (x_1, \ldots, x_n)$. The variable names serve for selecting components of the state. Guards and assertions can be written with boolean expressions, like $[x > 0]$, instead of state predicates, if the state space is understood from the context:

$$\begin{array}{rcl} [b] & \widehat{=} & [p] \quad \text{where} \quad p\,\gamma \equiv b \\ \{b\} & \widehat{=} & \{p\} \quad \text{where} \quad p\,\gamma \equiv b \end{array}$$

The assignment $x := e$ updates x and leaves all other variables unchanged. The nondeterministic assignment $x :\in q$ assigns x an arbitrary element such that $q\,x$ holds and leaves all other variables unchanged. For example, if x, y are the only program variables, then:

$$\begin{array}{rcl} x := e & \widehat{=} & [R] \quad \text{where} \quad R\,(x,y)\,(x',y') \equiv x' = e \wedge y' = y \\ x :\in q & \widehat{=} & [R] \quad \text{where} \quad R\,(x,y)\,(x',y') \equiv q\,x' \wedge y' = y \end{array}$$

The declaration of a local variable $y : \Delta$ with boolean expression b extends the state space and sets y to any value for which b holds. Suppose the state space consists only of $x : \Gamma$:

$$\begin{array}{rcl} \mathbf{var}\, y \mid b \bullet S & \widehat{=} & [\mathit{Enter}]\,;\, S\,;\, [\mathit{Exit}] \;\text{ where } \mathit{Enter}\,x\,(x',y') \equiv x = x' \wedge b[y \backslash y'] \\ & & \qquad\qquad \text{and } \mathit{Exit}\,(x,y)\,x' \equiv x = x' \end{array}$$

Leaving out the initialization predicate as in $\mathbf{var}\, y \bullet S$ means initializing the variable arbitrarily, $\mathbf{var}\, y \mid \mathit{true} \bullet S$, and $\mathbf{var}\, y = y_0 \bullet S$ means setting y initially to y_0. For brevity, we leave out the type of the introduced variable. Since $\Gamma \times (\Delta \times \Omega)$ is isomorphic to $(\Gamma \times \Delta) \times \Omega$, we can always find functions which transform an expression of one to the other type. Hence we simply write $\Gamma \times \Delta \times \Omega$. For example, if $\Gamma = \Gamma_1 \times \cdots \times \Gamma_n$ then S above would have the type $\Gamma_1 \times \cdots \times \Gamma_n \times \Delta \mapsto \Gamma_1 \times \cdots \times \Gamma_n \times \Delta$.

We use following properties to move guards and assertions over assignments and to eliminate local variable declarations; $f[x \backslash e]$ stands for expression f with

variable x substituted by expression e:

$$x := e\,;[b] \quad = \quad [b[x\backslash e]]\,;x := e \tag{44}$$
$$x := e\,;\{b\} \quad = \quad \{b[x\backslash e]\}\,;x := e \tag{45}$$
$$\mathbf{var}\ y \mid q \bullet x, y := e, f \quad = \quad x := e \tag{46}$$

2.6 Data Refinement

Data refinement $S \sqsubseteq_R T$ generalizes (algorithmic) refinement by relating the initial and final state of S and T with relation R. We allow R to refine only part of the state, i.e. if the (initial and final) state space of S can be partitioned into $\Delta \times \Gamma$ and R to relates values of Δ to values of Ω, then the state space of T is $\Omega \times \Gamma$. We write Id for the identity relation and $\times$ for the parallel composition of relations:

$$S \sqsubseteq_R T \quad \widehat{=} \quad S\,;[R \times Id] \sqsubseteq [R \times Id]\,;T$$

Sequential composition and nondeterministic choice preserved data refinement in the following sense:

$$S_1 \sqsubseteq_R T_1 \ \wedge\ S_2 \sqsubseteq_R T_2 \ \Rightarrow\ S_1\,;S_2 \sqsubseteq_R T_1\,;T_2 \tag{47}$$
$$S_1 \sqsubseteq_R T_1 \ \wedge\ S_2 \sqsubseteq_R T_2 \ \Rightarrow\ S_1 \sqcap S_2 \sqsubseteq_R T_1 \sqcap T_2 \tag{48}$$

Strong, weak, and positive iteration preserve data refinement. Let S and T be conjunctive predicate transformers:

$$S \sqsubseteq_R T \ \Rightarrow\ S^\omega \sqsubseteq_R T^\omega \tag{49}$$
$$S \sqsubseteq_R T \ \Rightarrow\ S^* \sqsubseteq_R T^* \tag{50}$$
$$S \sqsubseteq_R T \ \Rightarrow\ S^+ \sqsubseteq_R T^+ \tag{51}$$

The image of predicate p under relation R is denoted by $R[p]$. We have for any R, p, q:

$$R[p] \le q \quad \equiv \quad R\,x\,y \wedge p\,x \Rightarrow q\,y \tag{52}$$
$$\tag{53}$$

For the data refinement of guards and asserts we have:

$$[p] \sqsubseteq_R [q] \quad \equiv \quad R[\neg p] \le \neg q \tag{54}$$
$$\{p\} \sqsubseteq_R \{q\} \quad \equiv \quad R[p] \le q \tag{55}$$

We give selected theorems about data refining assignments; they naturally generalize when only a specific component of a larger state space is refined.

Assume that relation R relates X to Y and the state space includes Z. Variables x, y, z refer to the corresponding state components:

$$b \to x, z := e, g \sqsubseteq_R c \to y, z := f, h \quad \equiv \quad c \wedge R\,x\,y \Rightarrow b \wedge R\,e\,f \wedge g = h \tag{56}$$

$$\mathit{skip} \sqsubseteq_R c \to y := f \quad \equiv \quad c \wedge R\,x\,y \Rightarrow R\,x\,f \tag{57}$$

3 Binary Fair Choice

For statements S and T, the loop $\mathbf{do}\, S \diamond T\, \mathbf{od}$ repeatedly executes S or T, whichever is enabled, and terminates when neither one is enabled. If both S and T are enabled, the choice is arbitrary, except that if one is continuously enabled, it will be repeatedly taken, a criterion known as weak fairness. The loop $\mathbf{do}\, S \rhd T\, \mathbf{od}$ is only fair only to T: if T is continuously enabled, it will be repeatedly taken, and S may be neglected forever. Both $\diamond$, read "fair choice" and $\rhd$, read "right fair choice" are defined in isolation, such that the meaning of a loop containing those is given in a compositional manner.

We introduce an operator $\overline{S}$, read "try S", for a predicate transformer S. If S is enabled, $\overline{S}$ behaves as S, otherwise as *skip*:

$$\overline{S} \quad \widehat{=} \quad S \sqcap [\neg\, en\, S]$$

In the fair choice between S and T we may take S or T arbitrarily but finitely often, and then have to give T and S a chance, respectively. This is expressed in terms of positive weak iterations:

$$S \diamond T \quad \widehat{=} \quad S^+ ; \overline{T} \sqcap T^+ ; \overline{S}$$
$$S \rhd T \quad \widehat{=} \quad S^+ ; \overline{T} \sqcap T^+$$

The "left fair choice" operator $S \lhd T$ is defined by $T \rhd S$. For reasons of symmetry, we continue only with $S \rhd T$. To support our confidence in these definitions, we study two examples. The first one is an abstract view of transmission over an unreliable medium. The specification calls for copying data *in* to variable *out*. The implementation tries to do that repeatedly, and will either succeed or fail, with success given a fair chance. We assume that we

can detect successful reception and indicate this by setting variable r to *true*:

$$
\begin{array}{lcl}
T_0 & \widehat{=} & out := in \\
T_1 & \widehat{=} & \mathbf{var}\ r \bullet r := false\ ; \\
 & & \mathbf{do}\ \neg r \rightarrow skip \\
 & & \ \rhd \neg r \rightarrow out, r := in, true \\
 & & \mathbf{od}
\end{array}
$$

We prove that $T_0 = T_1$ by first simplifying the body of the loop:

$$
\begin{array}{ll}
 & \neg r \rightarrow skip \rhd \neg r \rightarrow out, r := in, true \\
= & \quad \langle \text{definition of } \rightarrow, \textit{skip} \text{ unit of } ; \rangle \\
 & [\neg r] \rhd [\neg r]\ ;\ out, r := in, true \\
= & \quad \langle \text{definition of } \rhd \rangle \\
 & [\neg r]^+\ ;\ \overline{[\neg r]\ ;\ out, r := in, true} \sqcap ([\neg r]\ ;\ out, r := in, true)^+ \\
= & \quad \langle [\neg r]\ ;\ out, r := in, true \text{ disables itself, } (37) \rangle \\
 & [\neg r]^+\ ;\ \overline{[\neg r]\ ;\ out, r := in, true} \sqcap [\neg r]\ ;\ out, r := in, true \\
= & \quad \langle (27) \rangle \\
 & [\neg r]\ ;\ \overline{[\neg r]\ ;\ out, r := in, true} \sqcap [\neg r]\ ;\ out, r := in, true \\
= & \quad \langle \text{definition of } \overline{S} \text{ and } en([\neg r]\ ;\ out, r := in, true) = \neg r \text{ by } (6) \rangle \\
 & [\neg r]\ ;\ ([\neg r]\ ;\ out, r := in, true \sqcap [r]) \sqcap [\neg r]\ ;\ out, r := in, true \\
= & \quad \langle ;\ \text{distributes over } \sqcap, \text{ for any } p, q\text{: } [p]\ ;\ [q] = [p \wedge q] \rangle \\
 & [\neg r]\ ;\ out, r := in, true \sqcap [false] \sqcap [\neg r]\ ;\ out, r := in, true \\
= & \quad \langle \text{definition of } \textit{magic}, \textit{magic} \text{ top of lattice, } \sqcap \text{ idempotent} \rangle \\
 & [\neg r]\ ;\ out, r := in, true
\end{array}
$$

We continue:

$$
\begin{array}{ll}
 & \mathbf{var}\ r \bullet r := \mathit{false}\ ;\ \mathbf{do}\ \neg r \rightarrow \neg r \rightarrow \mathit{skip} \rhd \mathit{out}, r := \mathit{in}, \mathit{true}\ \mathbf{od} \\
= & \quad \langle \text{above calculation} \rangle \\
 & \mathbf{var}\ r \bullet r := \mathit{false}\ ;\ \mathbf{do}\ [\neg r]\ ;\ \mathit{out}, r := \mathit{in}, \mathit{true}\ \mathbf{od} \\
= & \quad \langle \text{definition of } \mathbf{do}\ S\ \mathbf{od} \text{ and } \mathit{en}([\neg r]\ ;\ \mathit{out}, r := \mathit{in}, \mathit{true}) = \neg r \text{ by (6)} \rangle \\
 & \mathbf{var}\ r \bullet r := \mathit{false}\ ;\ ([\neg r]\ ;\ \mathit{out}, r := \mathit{in}, \mathit{true})^{\omega}\ ;\ [r] \\
= & \quad \langle [\neg r]\ ;\ \mathit{out}, r := \mathit{in}, \mathit{true} \text{ disables itself, (35)} \rangle \\
 & \mathbf{var}\ r \bullet r := \mathit{false}\ ;\ [\neg r]\ ;\ \mathit{out}, r := \mathit{in}, \mathit{true}\ ;\ [r] \\
= & \quad \langle (44) \text{ twice} \rangle \\
 & \mathbf{var}\ r \bullet [\mathit{true}]\ ;\ r := \mathit{false}\ ;\ [\mathit{true}]\ ;\ \mathit{out}, r := \mathit{in}, \mathit{true} \\
= & \quad \langle \text{definition of } \mathit{skip} \text{ and } \mathit{skip} \text{ unit of } ; \rangle \\
 & \mathbf{var}\ r \bullet \mathit{out}, r := \mathit{in}, \mathit{true} \\
= & \quad \langle (46) \rangle \\
 & \mathit{out} := \mathit{in}
\end{array}
$$

Thus we have $T_0 = T_1$. On the other hand, if we replace the right fair choice by nondeterministic choice, as in T_2 below, the second alternative may be continuously selected, leading to nontermination.

$$
\begin{array}{lll}
T_2 & \widehat{=} & \mathbf{var}\ r \bullet r := \mathit{false}\ ; \\
 & & \mathbf{do}\ \neg r \rightarrow \mathit{skip} \\
 & & \ \sqcap\ \neg r \rightarrow \mathit{out}, r := \mathit{in}, \mathit{true} \\
 & & \mathbf{od}
\end{array}
$$

Formally, we note that the body of the loop is enabled when $\neg r$ holds. The body of the loop does not disable itself. Hence by (42) the loop is equal to $\{\neg r\}$. From (45) we get that $r := \mathit{false}; \{\neg r\} = \mathit{abort}$. We have $\mathbf{var}\ r \bullet \mathit{abort} = \mathit{abort}$, therefore $T_2 = \mathit{abort}$.

The second example illustrates that the fair choice operator ensures only weak fairness, not strong fairness. Consider a loop with two boolean variables, b and c:

$$
\begin{array}{lll}
U_0 & \widehat{=} & \mathbf{do}\ b \rightarrow c := \neg c \\
 & & \ \Diamond\ b \wedge c \rightarrow b := \mathit{false} \\
 & & \mathbf{od}
\end{array}
$$

If the first alternative is continuously taken, the second alternative is repeatedly enabled and disabled, but is not continuously enabled. With strong fairness it, will eventually be taken, with weak fairness not. Hence we expect $U_0 = \{\neg b\}$. We sketch the proof. Consider the body of the loop:

$$
\begin{array}{ll}
 & b \to c := \neg c \diamond b \wedge c \to b := \mathit{false} \\
= & \quad \langle \text{definition of } \diamond \rangle \\
 & (b \to c := \neg c)^+ \,;\, \overline{b \wedge c \to b := \mathit{false}} \sqcap (b \wedge c \to b := \mathit{false})^+ \,;\, \overline{b \to c := \neg c} \\
= & \quad \langle b \wedge c \to b := \mathit{false} \text{ disables itself, (37)} \rangle \\
 & (b \to c := \neg c)^+ \,;\, \overline{b \wedge c \to b := \mathit{false}} \sqcap b \wedge c \to b := \mathit{false} \,;\, \overline{b \to c := \neg c} \\
= & \quad \langle (b \to c := \neg c)^+ \text{ simplifies to } b \to c := \mathit{true} \sqcap b \to c := \mathit{false} \rangle \\
 & b \to c := \mathit{true} \,;\, \overline{b \wedge c \to b := \mathit{false}} \sqcap \\
 & b \to c := \mathit{false} \,;\, \overline{b \wedge c \to b := \mathit{false}} \sqcap \\
 & b \wedge c \to b := \mathit{false} \,;\, \overline{b \to c := \neg c} \\
= & \quad \langle \text{definition of } \overline{S}, \text{ simplifications} \rangle \\
 & b \to c := \mathit{true} \,;\, b := \mathit{false} \sqcap b \to c := \mathit{false} \sqcap b \wedge c \to b := \mathit{false} \,;\, c := \neg c
\end{array}
$$

Due to $b \to c := \mathit{false}$, the action in the last line does not disable itself, hence $U_0 = \{\neg b\}$ by (42). Intuitively, $b \to c := \mathit{false}$ arises from repeated executions of $b \to c := \neg c$ in U_0.

Following theorem, taken from [14], states that nondeterministic choice can be implemented by symmetric and asymmetric fair choice.

Theorem 3.1 *Let S and T be monotonic predicate transformers:*

$$\mathbf{do}\ S \sqcap T\ \mathbf{od} \quad \sqsubseteq \quad \mathbf{do}\ S \diamond T\ \mathbf{od} \qquad \text{(a)}$$

$$\mathbf{do}\ S \sqcap T\ \mathbf{od} \quad \sqsubseteq \quad \mathbf{do}\ S \rhd T\ \mathbf{od} \qquad \text{(b)}$$

4 Generalized Fair Choice

Consider the fair choice among tree alternatives, expressed using binary fair choice in two different ways:

$$L = (S \diamond T) \diamond U \qquad\qquad R = S \diamond (T \diamond U)$$

Assume that S, T, U are always enabled. If L starts with repeating T a finite number of times, S will be tried and taken. If R starts with repeating T a finite number of times, U will be tried and taken. Hence the sequence $T\,;U$ is impossible for L but possible for R, and dually for the sequence $T\,;S$. Thus L and R are different and $\diamond$ is not associative. This necessitates a more general fair choice operator over a set of alternatives. It also implies that we will not consider nested applications of fair choice.

We intend to allow combinations of unfair and fair actions. As nondeterministic choice $\sqcap$ is associative, several unfair alternatives can be combined to a single one. It is sufficient to consider an operator with a single unfair

and an number of a fair alternatives, which we write as $S \rhd i \in I \bullet S_i$ for a finite and non-empty set I. The idea of the definition is that no matter which combinations of statements S_j is selected, each statement S_i has to be tried; which S_i is tried at the end is arbitrary:

$$\begin{array}{rcll} S \rhd i \in \{j\} \bullet S_i & \widehat{=} & S \rhd S_j & \\ S \rhd i \in I \bullet S_i & \widehat{=} & \sqcap\, i \in I \bullet (S \rhd j \in I - \{i\} \bullet S_j)^+ \,;\, \overline{S_i} & \text{if } |S| > 1 \end{array}$$

If there are only fair alternatives, we write simply $\diamond\, i \in I \bullet S_i$, defined by:

$$\diamond\, i \in I \bullet S_i \quad \widehat{=} \quad \mathit{magic} \rhd i \in I \bullet S_i$$

Binary fair choice emerges as a special case of general fair choice, in the sense that for $k \neq j$ we have $\diamond\, i \in \{j,k\} \bullet S_i = S_j \diamond S_k$. Given a fixed finite index set $I = \{1,..,n\}$, we write $S \rhd S_1 \diamond S_2 \diamond \cdots \diamond S_n$. The order in which the alternatives appear does not matter by definition. We also allow a mixture of notations as in $S \rhd i \in I \bullet S_i \diamond j \in J \bullet T_j$. To illustrate the definition, for the choice among three alternatives we have:

$$S \rhd T \diamond U \quad = \quad (T \diamond U)^+ \sqcap (S \rhd U)^+ \,;\, \overline{T} \sqcap (S \rhd T)^+ \,;\, \overline{U} \tag{58}$$

$$S \diamond T \diamond U \quad = \quad (T \diamond U)^+ \,;\, \overline{S} \sqcap (S \diamond U)^+ \,;\, \overline{T} \sqcap (S \diamond T)^+ \,;\, \overline{U} \tag{59}$$

We state some basic properties of fair choice. Fair choice is enabled when any one of the alternatives is:

Theorem 4.1 *Let I be a non-empty index set and S, S_i for $i \in I$ be monotonic predicate transformers:*

$$\mathit{en}(S \rhd i \in I \bullet S_i) \quad = \quad \mathit{en}\, S \vee (\vee i \in I \bullet \mathit{en}\, S_i)$$

Proof. The proof proceeds by induction over the size of I. For $|I| = 1$, the base case, we have:

$$\begin{array}{rl} & \mathit{en}(S \rhd T) \\ = & \quad \langle \text{definition of } \rhd \rangle \\ & \mathit{en}(S^+ \,;\, \overline{T} \sqcap T^+) \\ = & \quad \langle (5), (27), (6), \text{ for any } S\text{: } \mathit{en}\, \overline{S} = \mathit{true} \rangle \\ & \mathit{en}\, S \vee \mathit{en}\, T \end{array}$$

Now assume $\mathit{en}(S \rhd i \in J \bullet S_i) = \mathit{en}\, S \vee (\vee i \in J \bullet \mathit{en}\, S_i)$ for set J with $|J| \geq 1$. We show that $\mathit{en}(S \rhd i \in I \bullet S_i) = \mathit{en}\, S \vee (\vee i \in I \bullet \mathit{en}\, S_i)$ holds for set I with $|I| = |J| + 1$:

$$
\begin{array}{ll}
 & en(S \rhd i \in I \bullet S_i) \\
= & \quad \langle \text{definition of generalized } \rhd \rangle \\
 & en(\sqcap i \in I \bullet (S \rhd j \in I - \{i\} \bullet S_j)^+ \,;\, \overline{S_i}) \\
= & \quad \langle \text{for any } S_i\text{: } en(\sqcap i \in I \bullet S_i) = \vee i \in I \bullet en\, S_i \rangle \\
 & \vee i \in I \bullet en((S \rhd j \in I - \{i\} \bullet S_j)^+ \,;\, \overline{S_i}) \\
= & \quad \langle (27), (6), \text{for any } S\text{: } en\, \overline{S} = true \rangle \\
 & \vee i \in I \bullet en(S \rhd j \in I - \{i\} \bullet S_j) \\
= & \quad \langle \text{induction assumption} \rangle \\
 & \vee i \in I \bullet en\, S \vee (\vee j \in I - \{i\} \bullet en\, S_i) \\
= & \quad \langle \text{logic} \rangle \\
 & en\, S \vee (\vee i \in I \bullet en\, S_i)
\end{array}
$$

□

Nondeterministic choice $\sqcap$ is monotonic in both operands. By comparison, fair choice is monotonic provided that additionally the enabledness of each fair alternative is not decreased.

Theorem 4.2 *Let I be a non-empty index set and S, T, S_i, T_i for $i \in I$ be monotonic predicate transformers. If*

(a) $S \sqsubseteq T, \quad \wedge i \in I \bullet S_i \sqsubseteq T_i,$

(b) $\wedge i \in I \bullet en\, S_i \leq en\, T_i$

then:

$$S \rhd i \in I \bullet S_i \quad \sqsubseteq \quad T \rhd i \in I \bullet T_i$$

Proof. The proof proceeds by induction over the size of I. For $|I| = 1$, the base case, we have to show that $S \sqsubseteq T$, $U \sqsubseteq V$, and $en\, U \leq en\, V$ imply $S \rhd U \sqsubseteq T \rhd V$. Assuming $S \sqsubseteq T$, $U \sqsubseteq V$, and $en\, U \leq en\, V$ we have:

$$
\begin{array}{ll}
 & S \rhd U \\
= & \quad \langle \text{definition of } \diamond \rangle \\
 & S^+ \,;\, \overline{U} \sqcap U^+ \\
\sqsubseteq & \quad \langle \text{monotonicity of } S^+, (*) \text{ below, monotonicity of } ; \text{ and } \sqcap \rangle \\
 & T^+ \,;\, \overline{V} \sqcap V^+ \\
= & \quad \langle \text{definition of } \diamond \rangle \\
 & T \rhd V
\end{array}
$$

The step (*) relies on the property that $U \sqsubseteq V \wedge (en\, U \leq en\, V)$ implies $\overline{U} \sqsubseteq \overline{V}$. This follows from the definition of $\overline{S}$, monotonicity of $\sqcap$ and the fact that $[p] \sqsubseteq [q] \equiv p \geq q$. For the induction step, the hypothesis is that $S \sqsubseteq T$, $\wedge i \in J \bullet S_i \sqsubseteq T_i$, and $\wedge i \in J \bullet en\, S_i \leq en\, T_i$ imply $S \rhd i \in J \bullet S_i \sqsubseteq T \rhd i \in$

$I \bullet T_i$ for set J with $|J| \geq 1$. We show that the theorem holds for set I with $|I| = |J| + 1$. Assuming $S \sqsubseteq T$, $\wedge i \in I \bullet S_i \sqsubseteq T_i$, and $\wedge i \in I \bullet en\, S_i \leq en\, T_i$) we have:

$$
\begin{array}{ll}
 & S \rhd i \in I \bullet S_i \\
= & \quad \langle \text{definition of } \rhd,\ |I| > 1 \rangle \\
 & \sqcap i \in I \bullet (S \rhd j \in I - \{i\} \bullet S_j)^+ \,;\, \overline{S_i} \\
\sqsubseteq & \quad \langle \text{hypothesis, monotonicity of } S^+, (*) \text{ above, monotonicity of } ; \text{ and } \sqcap \rangle \\
 & \sqcap i \in I \bullet (T \rhd j \in I - \{i\} \bullet T_j)^+ \,;\, \overline{T_i} \\
= & \quad \langle \text{definition of } \diamond \rangle \\
 & T \rhd i \in I \bullet T_i
\end{array}
$$

□

To see that enabledness in the fair operand must not be decreased, consider the statements:

$$S_0 \ \widehat{=}\ b \to b := \mathit{false} \diamond b \to \mathit{skip} \qquad S_1 \ \widehat{=}\ \mathit{magic} \diamond b \to \mathit{skip}$$

Clearly we have $b \to b := \mathit{false} \sqsubseteq \mathit{magic}$, but if S_0 were refined by S_1, then **do** S_0 **od** would be refined by **do** S_1 **od** according to (43), as $en\, S_0 = en\, S_1$. However, **do** S_0 **od** always terminates, by setting b to *false* it if is *true* initially, but **do** S_1 **od** does not terminate if b is *true* initially.

We study further basic properties. Nondeterministic choice has *abort* as zero, as does fair choice:

Theorem 4.3 *Let I be a non-empty index set and S, S_i for $i \in I$ be monotonic predicate transformers:*

$$\mathit{abort} \rhd i \in I \bullet S_i \ = \ \mathit{abort} \qquad \text{(a)}$$

$$S \rhd i \in I \bullet S_i \diamond \mathit{abort} \ = \ \mathit{abort} \qquad \text{(b)}$$

Proof. The proofs are straightforward by induction over the size of I. We only give the base case of (a):

$$
\begin{array}{ll}
 & \mathit{abort} \rhd S \\
= & \quad \langle \text{definition of } \rhd \rangle \\
 & \mathit{abort}^+ \sqcap S^+ \,;\, \mathit{abort} \\
= & \quad \langle \mathit{abort}^+ = \mathit{abort},\ \mathit{abort} \text{ zero of } \sqcap \rangle \\
 & \mathit{abort}
\end{array}
$$

□

Nondeterministic choice has *magic* as unit, $S \sqcap \mathit{magic} = S$ and is idempotent, $S \sqcap S = S$. For fair choice we have instead $S \rhd \mathit{magic} = S^+$,

$magic \rhd S = S^+$, and $S \diamond S = S^+ ; \overline{S}$. However, in the context of a loop, fair choice has *magic* as unit and is idempotent. For simplicity, we formalize this only for binary choice:

Theorem 4.4 *Let S be a monotonic predicate transformer:*

$$\begin{array}{rcll} \mathbf{do}\ S \diamond magic\ \mathbf{od} & = & \mathbf{do}\ S\ \mathbf{od} & \text{(a)} \\ \mathbf{do}\ S \diamond S\ \mathbf{od} & = & \mathbf{do}\ S\ \mathbf{od} & \text{(c)} \end{array}$$

Proof. For (a) we have:

$$\begin{array}{ll} & \mathbf{do}\ S \diamond magic\ \mathbf{od} \\ = & \quad \langle \text{definition of } \mathbf{do}\ S\ \mathbf{od} \rangle \\ & (S \diamond magic)^\omega ; [\neg\, en(S \diamond magic)] \\ = & \quad \langle \text{Theorem 4.1}, \ en\ magic = false \rangle \\ & (S \diamond magic)^\omega ; [\neg\, en\ S] \\ = & \quad \langle \text{definition of } \diamond \rangle \\ & (S^+ ; \overline{magic} \sqcap magic^+ ; \overline{S})^\omega ; [\neg\, en\ S] \\ = & \quad \langle \overline{magic} = skip, magic^+ = magic \rangle \\ & (S^+ ; skip \sqcap magic ; \overline{S})^\omega ; [\neg\, en\ S] \\ = & \quad \langle \text{for any } S\text{: } magic ; S = magic,\ S \sqcap magic = S,\ S ; skip = S \rangle \\ & (S^+)^\omega ; [\neg\, en\ S] \\ = & \quad \langle (23), \text{ definition of } \mathbf{do}\ S\ \mathbf{od} \rangle \\ & \mathbf{do}\ S\ \mathbf{od} \end{array}$$

For (b) we have:

$$\begin{array}{ll} & \mathbf{do}\ S \diamond S\ \mathbf{od} = \mathbf{do}\ S\ \mathbf{od} \\ = & \quad \langle \text{definition of } \mathbf{do}\ S\ \mathbf{od}, \text{ Theorem 4.1} \rangle \\ & (S \diamond S)^\omega ; [\neg\, en\ S] = S^\omega ; [\neg\, en\ S] \\ = & \quad \langle \text{definition of } \diamond, \text{ simplifications} \rangle \\ & (S^+ ; \overline{S})^\omega ; [\neg\, en\ S] = S^\omega ; [\neg\, en\ S] \end{array}$$

This is shown by mutual refinement. For brevity, we give only the proof in one direction:

$$
\begin{array}{ll}
 & (S^+ \,;\, \overline{S})^\omega \,;\, [\neg\, en\ S] \sqsupseteq S^\omega \,;\, [\neg\, en\ S] \\
\Leftarrow & \quad \langle (23), \text{monotonicity of } ;,\ S^\omega \rangle \\
 & S^+ \,;\, \overline{S} \sqsupseteq S^+ \\
\Leftarrow & \quad \langle \text{definition of } \overline{S}, \text{ distributivity} \rangle \\
 & S^+ \,;\, S \sqcap S^+ \,;\, [\neg\, en\ S] \sqsupseteq S^+ \\
\Leftarrow & \quad \langle \text{property of } \sqcap \rangle \\
 & S^+ \,;\, S \sqsupseteq S^+ \wedge S^+ \,;\, [\neg\, en\ S] \sqsupseteq S^+
\end{array}
$$

From the definition of S^+ and (8) we have that $S^+ = S \,;\, S^+ \sqcap S$. Hence the left conjunct can be rewritten as $S^+ \,;\, S \sqsupseteq S \,;\, S^+ \sqcap S$, which follows from (24).□

5 Data Refinement

We state theorems that allow additional alternatives to be introduced when data refining a loop. First we give a theorem for nondeterministic choice among the alternatives, then a theorem for both nondeterministic and fair choice.

Theorem 5.1 *Let S, T, H be conjunctive predicate transformers and R a relation. If*

(a) $S \sqsubseteq_R T, \quad skip \sqsubseteq_R H$

(b) $R[en\ S] \leq en\ T \vee en\ H$

(c) $R[true] \leq tr(\mathbf{do}\ H\ \mathbf{od})$

then:

$$\mathbf{do}\ S\ \mathbf{od} \quad \sqsubseteq_R \quad \mathbf{do}\ T \sqcap H\ \mathbf{od}$$

Condition (a) requires that concrete action T data refines abstract action S and that H is a stuttering action, i.e. it's effect is not observable abstractly. Condition (b) requires that T and H must be enabled whenever S is, i.e. the concrete loop will not terminate if the abstract loop does not. Condition (c) requires that H eventually disables itself, provided $R[true]$, the concrete invariant, holds. That is, H cannot introduce nontermination. The proof is given in [3].

Theorem 5.1 is applied to loops with multiple actions by taking $S = S_1 \sqcap \ldots \sqcap S_n$ and $T = T_1 \sqcap \ldots \sqcap T_m$. For $S \sqsubseteq_R T$ it is sufficient that each T_j data refines some S_i, formally $S_i \sqsubseteq_R T_j$. In general, each T_j can refine any subset of $S_1 \sqcap \ldots \sqcap S_n$ and not each S_i needs to be refined. Likewise, multiple stuttering actions can be introduced by taking $H = H_1 \sqcap \ldots \sqcap H_k$. Actions T_i and H_j are called the main and auxiliary actions.

For extending above theorem to fairness, we allow refinement to introduce both fair and unfair stuttering actions. All unfair actions can again be "merged" into one action.

Theorem 5.2 *Let S, H, G be conjunctive predicate transformers and let R be a relation. If*

(a) $S \sqsubseteq_R T, \quad skip \sqsubseteq_R H, \quad skip \sqsubseteq_R G$

(b) $R[en\ S] \leq en\ T \vee en\ H \vee en\ G$

(c) $R[true] \leq tr(\mathbf{do}\ H \rhd G\ \mathbf{od})$

then:

$$\mathbf{do}\ S\ \mathbf{od} \quad \sqsubseteq_R \quad \mathbf{do}\ T \sqcap H \rhd G\ \mathbf{od}$$

Condition (a) requires that the concrete unfair action data refines the abstraction action and that the introduced actions are stuttering actions. Condition (b) requires that the body of the concrete loop must be enabled when the body of the abstract loop is, according to Theorem 4.1. Condition (c) requires that the auxiliary computation, consisting of all auxiliary actions, eventually disables itself.

Proof. Assuming (a), (b), and (c), we have:

$$\begin{array}{ll}
 & \mathbf{do}\ S\ \mathbf{od} \sqsubseteq_R \mathbf{do}\ T \sqcap H \rhd G\ \mathbf{od} \\
\equiv & \quad \langle \text{definition of } \mathbf{do}\ S\ \mathbf{od}, \text{Theorem 4.1} \rangle \\
 & S^\omega\ ;\ [\neg\ en\ S] \sqsubseteq_R (T \sqcap H \rhd G)^\omega\ ;\ [\neg\ en\ T \wedge \neg\ en\ H \wedge \neg G] \\
\Leftarrow & \quad \langle (47), (54), \text{(b)} \rangle \\
 & S^\omega \sqsubseteq_R (T \sqcap H \rhd G)^\omega \\
\equiv & \quad \langle \text{definition of } \rhd \rangle \\
 & S^\omega \sqsubseteq_R ((T \sqcap H)^+\ ;\ \overline{G} \sqcap G^+)^\omega \\
\equiv & \quad \langle (23), \text{definition of } S^+, \text{distributivity} \rangle \\
 & (S^+)^\omega \sqsubseteq_R (T\ ;\ (T \sqcap H)^*\ ;\ \overline{G} \sqcap H\ ;\ (T \sqcap H)^*\ ;\ \overline{G} \sqcap G^+)^\omega \\
\Leftarrow & \quad \langle (47), \text{(a)}, S^+ \sqsubseteq_R T\ ;\ (T \sqcap H)^*\ ;\ \overline{G} \rangle \\
 & (S^+)^\omega \sqsubseteq_R (H\ ;\ (T \sqcap H)^*\ ;\ \overline{G} \sqcap G^+)^\omega \\
\equiv & \quad \langle (33), \text{definition of } S^+ \rangle \\
 & (S^+)^\omega \sqsubseteq_R (H^+\ ;\ (T\ ;\ H^*)^*\ ;\ \overline{G} \sqcap G^+)^\omega \\
\equiv & \quad \langle (20), \text{distributivity} \rangle \\
 & (S^+)^\omega \sqsubseteq_R (H^+\ ;\ \overline{G} \sqcap H^+\ ;\ (T\ ;\ H^*)^+\ ;\ \overline{G} \sqcap G^+)^\omega
\end{array}$$

$\Leftarrow$ $\langle$(47), (a), $S^+ \sqsubseteq_R H^+ ; (T ; H^*)^+ ; \overline{G}\rangle$
$(S^+)^\omega \sqsubseteq_R (H^+ ; \overline{G} \sqcap G^+)^\omega$
$\Leftarrow$ $\langle$(23), definition of $\rhd\rangle$
$S^\omega \sqsubseteq_R (H \rhd G)^\omega$
$\Leftarrow$ $\langle$(40)$\rangle$
$S^\omega \sqsubseteq_R \{tr(\mathbf{do}\, H \rhd G\, \mathbf{od})\} ; (H \rhd G)^*$
$\Leftarrow$ $\langle$(c), (55), $\{true\} = skip\rangle$
$S^\omega \sqsubseteq_R (H \rhd G)^*$
$\Leftarrow$ $\langle S^\omega \sqsubseteq S$, (a), $skip^* = skip\rangle$
true

□

A further generalization is to allow the abstract action system to have both unfair and fair actions and an arbitrary numer of fair stuttering actions to be introduced.

Theorem 5.3 *Let I, J be non-empty index sets, let S, S_i, T, T_i, H, H_j for $i \in I, j \in J$ be conjunctive predicate transformers, and let R be a relation. If*

(a) $S \sqsubseteq_R T, \quad \wedge i \in I \bullet S_i \sqsubseteq_R T_i, \quad skip \sqsubseteq_R H, \quad \wedge j \in J \bullet skip \sqsubseteq_R H_j$

(b) $R[en\, S \vee (\vee i \in I \bullet en\, S_i)] \leq en\, T \vee en\, H \vee (\vee i \in I \bullet en\, T_i) \vee (\vee j \in J \bullet en\, H_j)$,
$\wedge i \in I \bullet R[en\, S_i] \leq en\, T_i$

(c) $R[true] \leq tr(\mathbf{do}\, H \rhd j \in J \bullet H_j\, \mathbf{od})$

then:

$$\mathbf{do}\, S \rhd i \in I \bullet S_i\, \mathbf{od} \quad \sqsubseteq_R \quad \mathbf{do}\, T \sqcap H \rhd i \in I \bullet T_i \diamond j \in J \bullet H_j\, \mathbf{od}$$

Condition (a) requires that the concrete actions data refine the abstraction actions and that the introduced actions are stuttering actions. However, there has to be a one-to-one correspondence among the fair actions, $S_i \sqsubseteq_R T_i$ for all $i \in I$ and every S_i has to be refined; the unfair actions S, T, and H can be decomposed as previously. Condition (b) requires that the body of the concrete loop must be enabled when the body of the abstract loop is. Additionally, each concrete fair action T_i must be enabled when the corresponding abstract action S_i. Condition (c) requires that the auxiliary computation, consisting of all auxiliary actions, eventually disables itself. The proof proceeds by induction over the size of both I and J; it is left out for brevity.

6 Alternating Bit Protocol

The alternating bit protocol [5], a protocol for reliable communication over unreliable channels, has been repeatedly formalized. Our treatment is inspired by that of [8,16]. Channels are modeled as variables, as in [9], rather than as sequences. Let a, b be sequences of data items. We write $|a|$ for the length of a sequence, $a[i]$ for the i-th element, and $a[i..j)$ for the subsequence starting at index i of length $j-i$. We develop the alternating bit protocol as a refinement of A in a sequence of steps.

6.1 Specification

In its most abstract form, a transmission copies sequence a to sequence b:

$$A \quad \widehat{=} \quad b := a$$

6.2 Copying Data Items Successively

The first refinement step introduces a loop that copies the data items successively. Its body is:

$$S \quad \widehat{=} \quad k < |a| \to b, k := b^\frown\langle a[k]\rangle, k+1$$

The whole program, B, initializes index k to zero. The refinement $A \sqsubseteq B$ can be shown using standard refinement rules:

$$B \quad \widehat{=} \quad \mathbf{var}\, k \bullet b, k := \langle\rangle, 0 \,;\, \mathbf{do}\, S\, \mathbf{od}$$

6.3 Splitting into Sender and Receiver

The second refinement steps decomposes S into an abstract sender, T_1, and an abstract receiver, T_2. Sender and receiver communicate via common variable s. They synchronize by a common boolean variable m in a ping-pong fashion:

$$\begin{array}{lcl} T_1 & \widehat{=} & m \wedge x < |a| \to m, s, x := \mathit{false}, a[x], x+1 \\ T_2 & \widehat{=} & \neg m \to b, m := b^\frown\langle s\rangle, \mathit{true} \end{array}$$

The whole program, C, initializes m such that the sender starts:

$$C \quad \widehat{=} \quad \mathbf{var}\, m, s, x \bullet m, x := \mathit{true}, 0 \,;\, \mathbf{do}\, T_1 \sqcap T_2\, \mathbf{od}$$

To establish the correctness of this refinement, we show **do** S **od** $\sqsubseteq_P$ **do** $T_1 \sqcap T_2$ **od** by Theorem 5.1 using as the refinement relation:

$$P(k)(m,s,x) \quad \widehat{=} \quad 0 \le x \le |a| \wedge b[0..k) = a[0..k) \wedge ((m \wedge x = k) \vee (\neg m \wedge k = x-1 \wedge s = a[x-1]))$$

That is, this refinement step replaces k by m, s, x and keeps s. As T_2 modifies variable b, as S does, T_2 is the main action and T_1 the auxiliary action. The resulting conditions are:

(a) $S \sqsubseteq_P T_2, \quad skip \sqsubseteq_P T_1$

(b) $P[en\ S] \le en\ T_1 \vee en\ T_2$

(c) $P[true] \le tr(\mathbf{do}\ T_1\ \mathbf{od})$

The first part of condition (a) follows by (56) and the second part by (57). Condition (b) follows by (52). For condition (c) we observe that T_1 always disables itself, hence **do** T_1 **od** always terminates.

6.4 Introducing Faulty Channels

In the third refinement step faulty channels are placed between the sender and receiver. The sender keeps one private bit, c, that is attached to every transmission and flipped on that occasion, such that messages of the sender have alternating bits. Variable t is used to start and stop the data transmission. The sender becomes:

$$U_1 \quad \widehat{=} \quad c = g \wedge x < |a| \rightarrow c, s, t, x := \neg c, a[x], true, x+1$$

Once data transmission is started, the data channel may successfully transmit the bit and the data by copying c, s to f, p and stop; we assume that the correct transmission of the bit and the data can be detected. The channel may also keep transmitting c, s, or may loose the message, i.e. do nothing. These three possibilities are given by:

$$\begin{array}{lcl} U_2 & \widehat{=} & t \rightarrow f, p, t := c, s, false \\ U_3 & \widehat{=} & t \rightarrow f, p := c, s \\ U_4 & \widehat{=} & t \rightarrow skip \end{array}$$

The receiver keeps a bit with the "parity" of the expected message. If the received message matches, the data is appended to b and the bit is flipped. The transmission of an acknowledgement message, consisting of a single bit,

g, is initiated by setting u to *true*.

$$U_5 \quad \widehat{=} \quad d = f \to b, d, u := b^\frown\langle p\rangle, \neg b, \mathit{true}$$

The acknowledgement channel may either transmit the bit properly and stop, may keep transmitting the bit, or may loose the message:

$$\begin{array}{lcl} U_6 & \widehat{=} & u \to g, u := \neg d, \mathit{false} \\ U_7 & \widehat{=} & u \to g := \neg d \\ U_8 & \widehat{=} & u \to \mathit{skip} \end{array}$$

The sender detects proper transmission by comparing the received bit, g, with the expected on c. The initialization of the protocol allows the sender to start. This protocol copies a to b provided that correct transmission of data, action U_2, and correct transmission of acknowledgements, action U_6, have a fair chance:

$$\begin{array}{lcl} D & \widehat{=} & \mathbf{var}\, s, d, f, g, p, s, t, u, x \bullet \\ & & c, d, f, g, t, u := \mathit{true}, \mathit{false}, \mathit{true}, \mathit{true}, \mathit{false}, \mathit{false}\ ; \\ & & \mathbf{do}\ U_1 \sqcap U_3 \sqcap U_4 \sqcap U_5 \sqcap U_7 \sqcap U_8 \rhd U_2 \diamond U_6\ \mathbf{od} \end{array}$$

The correctness of the refinement of the loops is established by Theorem 5.3 with refinement relation:

$$\begin{array}{lcl} Q(m)(c, d, f, g, p, t, u) & \widehat{=} & (c = g \Rightarrow m) \wedge (m \Rightarrow u \vee c = g) \wedge \\ & & (d = f \Rightarrow \neg m) \wedge (\neg m \Rightarrow t \vee d = f) \wedge \\ & & (c = g \Rightarrow g \neq d) \wedge (d = f \Rightarrow f = c \wedge p = s) \end{array}$$

The first line states that if the concrete sender, U_1, is ready to transmit, $c = g$, the abstract sender, T_1 must be ready as well; on the other hand, if T_1 is ready to transmit, then either U_1 must be ready to transmit or the actions of the acknowledgement channel, U_6, U_7, U_8, are enabled. The second line states an analogous property about the receiver. The third line is the concrete invariant and expresses what the sender, U_1 and the receiver, U_5, can expect: if U_1 is ready to transmit, the receiver must have had acknowledged and if U_5 is ready to receive, the data must have been correctly transmitted. The refinement replaces m and keeps s and x. The data channel actions and the acknowledgement channel actions modify only the introduced variables and become auxiliary actions. The resulting conditions are:

(a) $T_1 \sqcap T_2 \sqsubseteq_Q U_1 \sqcap U_5$, $\mathit{skip} \sqsubseteq_Q U_3 \sqcap U_4 \sqcap U_7 \sqcap U_8$, $\mathit{skip} \sqsubseteq_Q U_2$, $\mathit{skip}_Q \sqsubseteq U_6$

(b) $Q[en\ T_1 \vee en\ T_2] \leq en\ U_1 \vee ... \vee en\ T_8$

(c) $Q[true] \leq tr(\mathbf{do}\ U_3 \sqcap U_4 \sqcap U_7 \sqcap U_8 \rhd U_2 \diamond U_6\ \mathbf{od})$

The first part of condition (a) can be split into $T_1 \sqsubseteq_Q U_1$ and $T_2 \sqsubseteq_Q U_5$, each of which follows by (56). The remaining parts follow by (57), after splitting the second one into four parts. Condition (b) follows by (52). For condition (c) we observe that $U_3 \sqcap U_4 \sqcap U_7 \sqcap U_8 \rhd U_2 \diamond U_6$ always disables itself, hence the loop always terminates.

7 Discussion

Theorem 5.3 is similar to the forward simulation of fair action systems in [4], but differs in three ways. First, the condition for the auxiliary computation in [4] is weaker as the auxiliary computation either has to terminate or has to enable and execute a main action. The second difference is that a more general mapping between abstract and concrete main actions is considered, compared to the one-to-one correspondence that is assumed here for simplicity. The third difference is that only unfair auxiliary actions can be introduced, whereas Theorem 5.3 gives the conditions for both unfair and fair auxiliary actions. The development of the alternating bit protocol relies on fair auxiliary actions being introduced for representing unreliable channels.

Theorem 5.3 has a stronger condition for fair actions than for unfair actions, as fair actions have to be refined individually but unfair actions can be refined jointly, suggesting that unfair actions are to be preferred. On the other hand, introducing fair auxiliary actions compared to unfair auxiliary actions makes the condition for the termination of the auxiliary computation weaker. Thus the methodological consequence is that the introduction of fair actions should be postponed until needed for ensuring the termination of the auxiliary computation. The development of the alternating bit protocol follows this scheme.

It would be interesting to see if strong fairness can also be treated using iteration statements. If binary fair choice were associative, the formalism could be significantly simplified. The definition of an associative binary fair choice operator has remained elusive is left as an open problem.

References

[1] C. Aarts, R. Backhouse, E. Boiten, H. Doorndijk, N. van Gasteren, R. van Geldrop, P. Hoogendijk, T. Voermans, and J. van der Woude. Fixed-point calculus. *Information Processing Letters*, 53(3):131–136, 1995.

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

[3] R.J.R. Back and J. von Wright. Reasoning algebraically about loops. *Acta Informatica*, 36(4):295–334, 1999.

[4] R.J.R. Back and Q. Xu. Refinement of fair action systems. *Acta Informatica*, 35(2):131–165, 1998.

[5] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, 1969.

[6] M. Broy and G. Nelson. Adding fair choice to Dijkstra's calculus. *ACM Transactions on Programming Languages and Systems*, 16(3):924–938, 1994.

[7] M. J. Butler and C. C. Morgan. Action systems, unbounded nondeterminism and infinite traces. *Formal Aspects of Computing*, 7(1):37–53, 1995.

[8] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[9] W. H. J. Feijen and A. J. M. van Gasteren. *On a Method of Multiprogramming*. Springer-Verlag, 1999.

[10] N. Francez. *Fairness*. Texts and Monographs in Computer Science. Springer-Verlag, 1986.

[11] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[12] Carroll C. Morgan. *Programming from Specifications*. Prentice Hall, 2nd edition, 1994.

[13] J. M. Morris. Temporal predicate transformers and fair termination. *Acta Informatica*, 27(4):287–313, 1990.

[14] E. Sekerinski. On guarded commands with fair choice. In R. Backhouse and J. Oliveira, editors, *5th International Conference on the Mathematics of Program Construction, MPC 2000*, Lecture Notes in Computer Science 1837, pages 127–139, Ponte de Lima, Portugal, 2000. Springer-Verlag.

[15] A. K. Singh. Program refinement in fair transition systems. *Acta Informatica*, 30:503–535, 1993.

[16] A. Wabenhorst. A stepwise development of the alternating bit protocol. Technical Report PRG-TR-12-97, Oxford University Computing Laboratory, March 1997.

[17] A. Wabenhorst. Stepwise development of fair distributed systems. *Acta Informatica*, 39:233271, 2003.