Acta Informatica 36, 295–334 (1999)

Reasoning algebraically about loops

R.J.R. Back, J. von Wright

Åbo Akademi University and Turku Centre for Computer Science (TUCS),
FIN-20520 Turku, Finland (e-mail: {backrj,jwright}@abo.fi)

Received: 11 February 1998 / 18 March 1999

Abstract. We show how to formalise different kinds of loop constructs within the refinement calculus, and how to use this formalisation to derive general transformation rules for loop constructs. The emphasis is on using algebraic methods for reasoning about equivalence and refinement of loop constructs, rather than operational ways of reasoning about loops in terms of their execution sequences. We apply the algebraic reasoning techniques to derive a collection of transformation rules for action systems and for guarded loops. These include transformation rules that have been found important in practical program derivations: data refinement and atomicity refinement of action systems; and merging, reordering, and data refinement of loops with stuttering transitions.

1 Introduction

Loops in imperative programming notations are generally defined using recursion. For recursion constructs (greatest and least fixpoints) a simple algebraic theory exists. However, this theory has not been applied to the correctness of transformation rules and refinement rules for guarded loop constructs (such as the ordinary while-loop that occurs in most imperative programming languages) within a total correctness framework. Rules for transformation and refinement of loop constructs have traditionally been proved using operational arguments (unless they have only been justified informally). This means that the proofs are long and hard to follow. Furthermore, they are very difficult to check and (as we shall see) they may make assumptions that are not really needed.

In this paper we define two simple fixpoint-based iteration operators – one is a least and the other a greatest fixpoint. We derive a number of basic properties for the iteration operators and then define the traditional loop construct in terms of iteration. The main part of the paper shows how a comprehensive collection of transformation and refinement rules can be derived from the basic rules for iterations. These rules include a rule for *data refinement with stuttering* and a rule for *refinement of atomicity* in loops. Neither of these has, to our knowledge, been given purely algebraic proofs before.

The algebraic style of reasoning makes proofs short and easy to read and to check. Furthermore, it allows conditions (assumptions) of the rules to be easily identified; they generally arise from the need to commute statements in a proof step. In particular, many conditions arise from a need to propagate free guard statements either forward or backward.

We use lambda notation for functions (e.g., $(\lambda x \bullet x = 1)$) and an infix dot for function application ($f.x$). Quantifiers are given low precedence and their scope is delimited by parentheses. We use standard notation for logical connectives and T for boolean truth. Proofs are written in a calculational style.

The rest of the paper is organised as follows. In Sect. 2 we briefly outline the basic theory of predicate transformers and refinement that is the basis for the paper. Section 3 describes two iteration constructs and their basic properties. In Sect. 6 we show how traditional loop constructs (while- and do-loops) are defined in terms of the iteration constructs. We give algebraic proofs for a number of loop transformation rules, including a rule for data refinement of loops with stuttering. The properties of iterations are applied to action systems in Sect. 4, and in Sect. 5 we consider the more demanding task of atomicity refinement. We then consider basic rules for guarded loops in Sect. 6. In Sect. 7 we derive examples of practically useful transformation rules for loops and we then finish with some concluding remarks in Sect. 8.

2 Predicate transformers as program statements

We assume that the reader is familiar with the weakest precondition semantics of simple imperative programming languages and with the basic notions of program refinement [2, 3, 14, 15, 21, 22]. We here quickly describe the notions of states, predicates and predicate transformers used in this paper, and how they are formalised.

2.1 Predicate transformers

By a predicate we mean a boolean state function, i.e., a function of type $\Sigma \to \mathsf{Bool}$, where the type Σ models the state space. Since a predicate represents a set of states, we freely use set notation (intersection, union, subset) for predicates.

A *predicate transformer* is a function that maps predicates to predicates. We use $\mathsf{Mtran}(\Sigma, \Gamma)$ for the set of *monotonic* functions that map predicates in $\Gamma \to \mathsf{Bool}$ to predicates in $\Sigma \to \mathsf{Bool}$. Here Σ is the *initial state space* and Γ the *final state space*. We take predicate transformers to model programs according to a *weakest precondition* intuition: if S is a predicate transformer and q a predicate over its final state space (a *postcondition*) then the predicate $S.\,q$ describes those initial states from which execution of S is guaranteed to terminate in a state in q. If $\sigma \in S.\,q$, then we say that S *establishes postcondition* q from initial state σ.

The two basic operations on predicate transformers are (sequential) composition and (demonic) choice, defined as follows:

$$(S_1\,;S_2).\,q = S_1.\,(S_2.\,q) \qquad \text{(\textit{sequential composition})}$$
$$(S_1 \sqcap S_2).\,q = S_1.\,q \cap S_2.\,q \qquad \text{(\textit{demonic choice})}$$

Demonic choice is a meet (greatest lower bound) operator. In fact, $\mathsf{Mtran}(\Sigma, \Gamma)$ is a complete lattice (though we will not make use of the join operator here). The bottom element is $\mathsf{abort} = (\lambda q \bullet \mathsf{false})$ and the top element is $\mathsf{magic} = (\lambda q \bullet \mathsf{true})$. The unit of composition is $\mathsf{skip} = (\lambda q \bullet q)$. The ordering on predicate transformers is defined by $S \sqsubseteq S' \equiv (\forall q \bullet S.\,q \subseteq S'.\,q)$, the *refinement ordering*.

In the weakest precondition intuition, the choice $S \sqcap S'$ establishes postcondition q if and only if both S and S' establish q. The bottom element abort establishes no postcondition, not even true. Dually, the top element magic establishes all postconditions, even false.

Predicate transformers can be classified according to basic homomorphism properties (in addition to monotonicity). We will mostly consider conjunctive predicate transformers; we say that S is *conjunctive* if it distributes over nonempty meets, i.e., if $S.\,(\cap i \in I \bullet q_i) = (\cap i \in I \bullet S.\,q_i)$ for arbitrary nonempty collections $\{q_i \mid i \in I\}$ of predicates. Dually, S is *disjunctive* if it distributes over nonempty joins of predicates. Furthermore, S is *strict* if $S.\,\mathsf{false} = \mathsf{false}$ and *terminating* if $S.\,\mathsf{true} = \mathsf{true}$. It is well known that conjunctivity and disjunctivity each implies monotonicity.

We will also make use of continuity. For that, we define a set K to be *directed* (with respect to an ordering $\sqsubseteq$) if

$$(\forall x, y \in K \bullet \exists z \in K \bullet x \sqcup y \sqsubseteq z) \qquad \text{(\textit{directedness})}$$

A function f is said to be *continuous* if it distributes over joins of directed sets. This definition is slightly more general than the traditional definition

in terms of distributivity over limits of sequences. Furthermore, it has the advantage of not involving the natural numbers. Note that continuity is related to disjunctivity: if a predicate transformer is strict and disjunctive, then it is also continuous.

The basic algebraic properties of monotonic predicate transformers can be summarised by stating that $\mathsf{Mtran}(\Sigma, \Sigma)$ is a monoid with composition ; and unit skip and that $\mathsf{Mtran}(\Sigma, \Gamma)$ is a complete lattice with meet $\sqcap$, top magic and bottom abort. Furthermore, these two structures interact as follows, for nonempty index set I:

$$(\sqcap i \in I \bullet S_i)\,;S = (\sqcap i \in I \bullet S_i\,;S)$$
$$S\,;(\sqcap i \in I \bullet S_i) \sqsubseteq (\sqcap i \in I \bullet S\,;S_i)$$
$$S\,;(\sqcap i \in I \bullet S_i) = (\sqcap i \in I \bullet S\,;S_i) \quad \text{if } S \text{ is conjunctive}$$

To avoid excessive parentheses, we give composition higher precedence than choice.

2.2 Guards and assertions

We do not assume any specific notation for constructing predicate transformers that model basic program statements (typically, some kind of notation for assignments to program variables would be used). However, we will use two ways of injecting predicates into predicate transformers, defined as follows:

$$[p].\,q = \neg p \cup q \qquad (\textit{guard})$$
$$\{p\}.\,q = p \cap q \qquad (\textit{assert})$$

Intuitively, the guard statement $[p]$ and the assert statement $\{p\}$ both test whether the condition p holds in the present state. If the condition holds, then both act as skip, i.e., do not change the state. If the condition does not hold, then the guard statement terminates miraculously (acts like magic) while the assert statement acts like abort.

The following basic algebraic properties of guards and assertions will be used (the proofs are straightforward):

$$[p] \sqsubseteq [q] \equiv p \supseteq q \qquad \{p\} \sqsubseteq \{q\} \equiv p \subseteq q$$
$$[p]\,;[q] = [p \cap q] \qquad \{p\}\,;\{q\} = \{p \cap q\}$$
$$[p] \sqcap [q] = [p \cup q] \qquad \{p\} \sqcap \{q\} = \{p \cap q\}$$

2.3 Context information

An assertion $\{p\}$ at a certain point in a program can be seen as asserting that p holds at that point. Dually, a guard $[p]$ corresponds to an assumption that

p holds. A refinement of the form $\{p\} \,;\, S \sqsubseteq S \,;\, \{q\}$ models *propagation of contextual information*; it can be interpreted as saying that if we know p holds in the initial state then execution of S leads to a final state where q holds. Similarly, other refinements including assertions and guards can be interpreted as propagation of information.

The following lemma collects some rules used in this paper.

Lemma 1. *Assertions and guards can be propagated according to the following rules:*

(a) $S \,;\, \{q\} \sqsubseteq \{p\} \,;\, S \;\equiv\; S.\, q \subseteq p$
(b) $[p] \,;\, S \sqsubseteq S \,;\, [q] \;\equiv\; \neg p \subseteq S.\, (\neg q)$
(c) $S \,;\, [q] \sqsubseteq [p] \,;\, S \;\equiv\; S.\, \mathsf{true} \cap p \subseteq S.\, q$ *if S is conjunctive*
(d) $S \,;\, [q] \sqsubseteq [p] \,;\, S \;\equiv\; S.\, (\neg q) \subseteq \neg p \cup S.\, \mathsf{false}$ *if S is disjunctive*

Proof. We show only the proof of part (b), as the proofs of the other parts are similar. We have

$$
\begin{array}{ll}
 & [p] \,;\, S \sqsubseteq S \,;\, [q] \\
\equiv & \{\text{definitions}\} \\
 & (\forall r \bullet \neg p \cup S.\, r \subseteq S.\, (\neg q \cup r)) \\
\Rightarrow & \{\text{specialise } r := \neg q\} \\
 & \neg p \cup S.\, (\neg q) \subseteq S.\, (\neg q) \\
\equiv & \{\text{lattice property}\} \\
 & \neg p \subseteq S.\, (\neg q)
\end{array}
$$

and

$$
\begin{array}{ll}
 & [p] \,;\, S \sqsubseteq S \,;\, [q] \\
\equiv & \{\text{definitions}\} \\
 & (\forall r \bullet \neg p \cup S.\, r \subseteq S.\, (\neg q \cup r)) \\
\Leftarrow & \{\text{general rule } S.\, p \cup S.\, q \subseteq S.\, (p \cup q) \text{ for monotonic } S\} \\
 & (\forall r \bullet \neg p \cup S.\, r \subseteq S.\, (\neg q) \cup S.\, r) \\
\Leftarrow & \{\text{monotonicity of } \cup \text{ with respect to } \subseteq\} \\
 & \neg p \subseteq S.\, (\neg q)
\end{array}
$$

□

Using the notion of *dual predicate transformers*, defined by $S^{\circ}.\, q = \neg\, S.\, (\neg\, q)$ we can reformulate Lemma 1 (d) in the following way which shows the duality with (c) clearly: if S is disjunctive then

$$S \,;\, [q] \sqsubseteq [p] \,;\, S \equiv S^{\circ}.\, \mathsf{true} \cap p \subseteq S^{\circ}.\, q$$

Furthermore, Lemma 1 (c) and (d) have dual formulations in terms of assertions (which we will not use in this paper), using the following general equivalence:

$$S\,;[q] \sqsubseteq [p]\,;S \equiv \{p\}\,;S \sqsubseteq S\,;\{q\}$$

for arbitrary monotonic predicate transformer S.

By combining Lemma 1 (b) and (c) we immediately get the following:

Lemma 2. *Assume that S is conjunctive. Then*

$$[p]\,;S \sqsubseteq S\,;[q] \Rightarrow S\,;[\neg q] \sqsubseteq [\neg p]\,;S$$

3 Iteration constructs

Standard algebraic treatment of recursion and iteration is based on fixpoint theory. Before we define and investigate the iteration constructs, we recall a number of basic results from the fixpoint theory of lattices. These are standard results ("folk theorems") that appear in the literature in many variations [1, 11, 25].

First of all, we assume that the *Knaster-Tarski theorem* is known:

Lemma 3. *Every monotonic function on a complete lattice has a complete lattice of fixpoints.*

In particular, this means that a monotonic function f on a complete lattice has a least fixpoint $\mu.\,f$ and a greatest fixpoint $\nu.\,f$. These are characterised by the following properties:

$$f.(\mu.\,f) = \mu.\,f \qquad f.(\nu.\,f) = \nu.\,f \qquad (\textit{unfolding})$$
$$f.x \sqsubseteq x \Rightarrow \mu.\,f \sqsubseteq x \qquad x \sqsubseteq f.x \Rightarrow x \sqsubseteq \nu.\,f \qquad (\textit{induction})$$

We also assume that the following *rolling rules* for fixpoints are known:

Lemma 4. *Assume that f and g are monotonic functions on a complete lattice. Then*

$$f.(\mu.\,(g \circ f)) \;=\; \mu.\,(f \circ g) \quad \text{and} \quad f.(\nu.\,(g \circ f)) \;=\; \nu.\,(f \circ g)$$

The following *diagonalisation lemma* gives a way of moving between fixpoints over one variable and fixpoints over multiple variables. Here we use binder notation $(\mu\, x \bullet f.x)$ for the least fixpoint $\mu.\,f$ (and similarly for $\nu.\,f$) and the notation $(\mu\, x\; y \bullet t)$ abbreviates $(\mu\, x \bullet (\mu\, y \bullet t))$.

Lemma 5. *Assume that f is a function of two arguments on a complete lattice and that f is monotonic in each of its arguments. Then*

(a) $(\mu\, x\; y \bullet f.x.y) \;=\; (\mu\, x \bullet f.x.x)$

(b) $(\nu\, x\, y \bullet f.x.y) \;=\; (\nu\, x \bullet f.x.x)$

Finally, we assume the following *fusion lemma* (attributed to Kleene) which links least fixpoints and continuity:

Lemma 6. *Assume that f and g are monotonic functions on complete lattices Σ and Γ and that $h : \Sigma \to \Gamma$ is continuous. Then*

(a) *if $h \circ f \sqsubseteq g \circ h$, then $h.(\mu.f) \sqsubseteq \mu.g$*
(b) *if $h \circ f = g \circ h$, then $h.(\mu.f) = \mu.g$*

There is also a greatest fixpoint version of the fusion theorem, using duals of continuity and directedness. However, we omit this, for brevity.

3.1 Iteration operators

Let us now look at the basic iteration constructs that the rest of this paper builds on. Intuitively, a recursive predicate transformer $(\mu\, X \bullet S)$ is executed by *unfolding*, i.e., by executing S and replacing any occurrence of X encountered with $(\mu\, X \bullet S)$. Thus, recursion leads to iterated execution of a piece of code.

By means of the fixpoint constructs we define two explicit *iteration constructs*:

$$S^\omega \;\hat{=}\; (\mu\, X \bullet S \,;\, X \sqcap \mathsf{skip}) \qquad \text{(\textit{strong iteration})}$$
$$S^* \;\hat{=}\; (\nu\, X \bullet S \,;\, X \sqcap \mathsf{skip}) \qquad \text{(\textit{weak iteration})}$$

Intuitively, S^* is S repeated a (demonically) chosen finite number of times. S^ω is similar, but it allows S to be repeated an infinite number of times (which is semantically equivalent to aborting). These operators themselves are not new [12,24], and some of the properties in Sects. 3.1–3.4 have been published before [9,24].

From the definitions of the iteration operators and the unfolding and induction rules for fixpoints we immediately get the following basic properties of iterations:

$$S^\omega \;=\; S \,;\, S^\omega \sqcap \mathsf{skip} \qquad \text{(\textit{unfold strong iteration})}$$
$$S \,;\, X \sqcap \mathsf{skip} \sqsubseteq X \;\Rightarrow\; S^\omega \sqsubseteq X \qquad \text{(\textit{strong iteration induction})}$$
$$S^* \;=\; S \,;\, S^* \sqcap \mathsf{skip} \qquad \text{(\textit{unfold weak iteration})}$$
$$X \sqsubseteq S \,;\, X \sqcap \mathsf{skip} \;\Rightarrow\; X \sqsubseteq S^* \qquad \text{(\textit{weak iteration induction})}$$

We shall now illustrate the intuition behind the two iteration operators. Both kinds of iterations are *unguarded*, i.e., the termination of an iteration S^ω or S^* is decided by a demonic choice, rather than by evaluation of a guard predicate. To see the difference between the two iterations, consider

iterating skip. Intuitively, repeating skip a finite number of times has the same effect as skip, but repeating skip indefinitely is equivalent to aborting:

$$\mathsf{skip}^* = \mathsf{skip}$$
$$\mathsf{skip}^\omega = \mathsf{abort}$$

The following derivations prove this. First,

$$\begin{array}{ll} & \mathsf{skip}^* \\ = & \{\text{unfold}\} \\ & \mathsf{skip} \,;\, \mathsf{skip}^* \sqcap \mathsf{skip} \\ \sqsubseteq & \{\text{general lattice property } x \sqcap y \sqsubseteq y\} \\ & \mathsf{skip} \end{array}$$

and

$$\begin{array}{ll} & \mathsf{skip} \sqsubseteq \mathsf{skip}^* \\ \Leftarrow & \{\text{induction}\} \\ & \mathsf{skip} \sqsubseteq \mathsf{skip} \,;\, \mathsf{skip} \sqcap \mathsf{skip} \\ \equiv & \{\mathsf{skip} \text{ is unit, meet is idempotent}\} \\ & \mathsf{T} \end{array}$$

which proves $\mathsf{skip}^* = \mathsf{skip}$. Then,

$$\begin{array}{ll} & \mathsf{skip}^\omega \sqsubseteq \mathsf{abort} \\ \Leftarrow & \{\text{induction}\} \\ & \mathsf{skip} \,;\, \mathsf{abort} \sqcap \mathsf{skip} \sqsubseteq \mathsf{abort} \\ \equiv & \{\mathsf{skip} \text{ is unit, } \mathsf{abort} \text{ is bottom element}\} \\ & \mathsf{T} \end{array}$$

which proves $\mathsf{skip}^\omega = \mathsf{abort}$.

Now let S be the guarded command $[x < 2] \,;\, x := x + 1$. If the initial state has $x = 0$, then S^* and S^ω both lead to x being assigned one of the values 0, 1 and 2. Here the guard statement prevents the strong iteration from going into an infinite loop.

3.2 Generalised induction principles

Occasionally, we will need the following generalisation of the definition of iterations.

Lemma 7. *Let S and T be arbitrary monotonic predicate transformers. Then*

$$(\mu\, X \bullet S \,;\, X \sqcap T) = S^\omega \,;\, T$$
$$(\nu\, X \bullet S \,;\, X \sqcap T) = S^* \,;\, T$$

Proof. For strong iteration we apply fusion (Lemma 6 (b)) with $h := (\lambda X \bullet X \,;\, T)$, which is easily proved to be continuous, and with $f := (\lambda X \bullet S \,;\, X \sqcap \mathsf{skip})$ and $g := (\lambda X \bullet S \,;\, X \sqcap T)$. For weak iteration, the proof makes use of the dual version of the fusion theorem (see the comment after Lemma 6). □

Lemma 7 and the general induction principles for fixpoints now give us more general induction rules, of which the induction rules above are special cases:

Corollary 8. *Assume that S and T are monotonic predicate transformers. Then*

$$S \,;\, X \sqcap T \sqsubseteq X \Rightarrow S^\omega \,;\, T \sqsubseteq X$$
$$X \sqsubseteq S \,;\, X \sqcap T \Rightarrow X \sqsubseteq S^* \,;\, T$$

In what follows, the justification "induction" will refer to these more general rules.

3.3 Basic properties of iterations

In the following we list a collection of basic properties of iterations. The proofs are based on induction, unfolding and distributivity properties of predicate transformers.

Lemma 9. *Assume that S and T are monotonic predicate transformers. Then*

(a) $S \sqsubseteq T \;\Rightarrow\; S^\omega \sqsubseteq T^\omega$ *and* $S \sqsubseteq T \;\Rightarrow\; S^* \sqsubseteq T^*$
(b) $S^\omega \sqsubseteq S$ *and* $S^* \sqsubseteq S$
(c) $S^\omega \,;\, S^\omega = S^\omega$ *and* $S^* \,;\, S^* = S^*$
(d) $(S^\omega)^\omega = \mathsf{abort}$ *and* $(S^\omega)^* = S^\omega$
(e) $(S^*)^\omega = \mathsf{abort}$ *and* $(S^*)^* = S^*$

Proof. For (a) we have

$$\begin{array}{ll} & S^\omega \sqsubseteq T^\omega \\ \Leftarrow & \{\text{induction}\} \\ & S \,;\, T^\omega \sqcap \mathsf{skip} \sqsubseteq T^\omega \\ \equiv & \{\text{unfold}\} \\ & S \,;\, T^\omega \sqcap \mathsf{skip} \sqsubseteq T \,;\, T^\omega \sqcap \mathsf{skip} \\ \equiv & \{\text{monotonicity of ; and } \sqcap\} \\ & S \sqsubseteq T \end{array}$$

The argument for weak iteration is similar. Next, (b):

$$
\begin{array}{ll}
& S^\omega \\
= & \{\text{unfold twice}\} \\
& S \mathbin{;} (S \mathbin{;} S^\omega \sqcap \mathsf{skip}) \sqcap \mathsf{skip} \\
\sqsubseteq & \{\text{monotonicity}\} \\
& S \mathbin{;} \mathsf{skip} \sqcap \mathsf{skip} \\
\sqsubseteq & \{\text{monotonicity, } \mathsf{skip} \text{ is unit of sequential composition}\} \\
& S
\end{array}
$$

with a similar argument for S^*. Now consider (c). For weak iteration we have

$$
\begin{array}{ll}
& S^* \mathbin{;} S^* \\
\equiv & \{\text{unfold}\} \\
& S^* \mathbin{;} (S \mathbin{;} S^* \sqcap \mathsf{skip}) \\
\sqsubseteq & \{\text{monotonicity}\} \\
& S^* \mathbin{;} \mathsf{skip} \\
\sqsubseteq & \{\mathsf{skip} \text{ is unit}\} \\
& S^*
\end{array}
$$

and

$$
\begin{array}{ll}
& S^* \sqsubseteq S^* \mathbin{;} S^* \\
\Leftarrow & \{\text{induction}\} \\
& S^* \sqsubseteq S \mathbin{;} S^* \sqcap S^* \\
\equiv & \{\text{general lattice property } x \sqsubseteq x \sqcap y \;\equiv\; x \sqsubseteq y\} \\
& S^* \sqsubseteq S \mathbin{;} S^* \\
\equiv & \{\text{unfold}\} \\
& S \mathbin{;} S^* \sqcap \mathsf{skip} \sqsubseteq S \mathbin{;} S^* \\
\equiv & \{\text{meet is lower bound}\} \\
& \mathsf{T}
\end{array}
$$

For strong iteration, $S^\omega \mathbin{;} S^\omega \sqsubseteq S^\omega$ is proved as for weak iteration. Refinement in the opposite direction is proved differently:

$$
\begin{array}{ll}
& S^\omega \sqsubseteq S^\omega \mathbin{;} S^\omega \\
\Leftarrow & \{\text{induction}\} \\
& S \mathbin{;} S^\omega \mathbin{;} S^\omega \sqcap \mathsf{skip} \sqsubseteq S^\omega \mathbin{;} S^\omega \\
\equiv & \{\text{unfold}\}
\end{array}
$$

$$
\begin{array}{ll}
& S \,;\, S^\omega \,;\, S^\omega \sqcap \mathsf{skip} \sqsubseteq (S \,;\, S^\omega \sqcap \mathsf{skip}) \,;\, S^\omega \\
\equiv & \{\text{distributivity}\} \\
& S \,;\, S^\omega \,;\, S^\omega \sqcap \mathsf{skip} \sqsubseteq S \,;\, S^\omega \,;\, S^\omega \sqcap S^\omega \\
\equiv & \{\text{unfold rightmost } S^\omega\} \\
& S \,;\, S^\omega \,;\, S^\omega \sqcap \mathsf{skip} \sqsubseteq S \,;\, S^\omega \,;\, S^\omega \sqcap S \,;\, S^\omega \sqcap \mathsf{skip} \\
\equiv & \{\text{general lattice property } x \sqcap x = x\} \\
& S \,;\, S^\omega \,;\, S^\omega \sqcap S \,;\, S^\omega \,;\, S^\omega \sqcap \mathsf{skip} \sqsubseteq S \,;\, S^\omega \,;\, S^\omega \sqcap S \,;\, S^\omega \sqcap \mathsf{skip} \\
\equiv & \{\text{monotonicity of ; and } \sqcap\} \\
& S \,;\, S^\omega \,;\, S^\omega \sqsubseteq S \,;\, S^\omega \\
\equiv & \{\text{general rule } S^\omega \sqsubseteq \mathsf{skip} \text{ (by unfolding)}\} \\
& \mathsf{T}
\end{array}
$$

Next, we prove (d). We have

$$
\begin{array}{ll}
& (S^\omega)^\omega \sqsubseteq \mathsf{abort} \\
\Leftarrow & \{\text{general rule } S^\omega \sqsubseteq \mathsf{skip}\text{, part (a) of this lemma}\} \\
& \mathsf{skip}^\omega \sqsubseteq \mathsf{abort} \\
\equiv & \{\text{general rule } \mathsf{skip}^\omega = \mathsf{abort} \text{ (see Sect. 3.1)}\} \\
& \mathsf{T}
\end{array}
$$

and then

$$
\begin{array}{ll}
& S^\omega \sqsubseteq (S^\omega)^* \\
\Leftarrow & \{\text{induction}\} \\
& S^\omega \sqsubseteq S^\omega \,;\, S^\omega \sqcap \mathsf{skip} \\
\equiv & \{\text{part (c) of this lemma}\} \\
& S^\omega \sqsubseteq S^\omega \sqcap \mathsf{skip} \\
\equiv & \{\text{general lattice property } x \sqsubseteq x \sqcap y \equiv x \sqsubseteq y\} \\
& S^\omega \sqsubseteq \mathsf{skip} \\
\equiv & \{\text{general rule } S^\omega \sqsubseteq \mathsf{skip}\ \} \\
& \mathsf{T}
\end{array}
$$

where the converse refinement follows directly from (b). Finally (e) is proved in exactly the same way as (d). □

The monotonicity properties in Lemma 9 (b) are so basic that we will often use them in proofs without explicit reference. We will also need the following property in a number of proofs:

Lemma 10. *Assume that S is a monotonic predicate transformer and g is a predicate. Then*

$$[\neg g] \,;\, ([g] \,;\, S)^\omega = [\neg g]$$

Proof.

$$\begin{aligned}
& [\neg g] \,;\, ([g] \,;\, S)^\omega \\
= & \{\text{unfold, distributivity}\} \\
& [\neg g] \,;\, [g] \,;\, S \,;\, ([g] \,;\, S)^\omega \sqcap [\neg g] \\
= & \{\text{general properties } [p] \,;\, [q] = [p \cap q] \text{ and } [\mathsf{false}] = \mathsf{magic}\} \\
& \mathsf{magic} \,;\, S \,;\, ([g] \,;\, S)^\omega \sqcap [\neg g] \\
= & \{\text{general property } \mathsf{magic} \,;\, S = \mathsf{magic}, \mathsf{magic} \text{ is top element}\} \\
& [\neg g]
\end{aligned}$$

□

3.4 Properties of conjunctive iterations

If we assume conjunctivity, then we can prove two properties that illustrate the correspondence between the iteration operators and the star operator of regular languages. First we have the *leapfrog* property.

Lemma 11 (Leapfrog). *Assume that S and T are monotonic predicate transformers and that S is conjunctive. Then*

$$S \,;\, (T \,;\, S)^\omega = (S \,;\, T)^\omega \,;\, S \qquad \text{and} \qquad S \,;\, (T \,;\, S)^* = (S \,;\, T)^* \,;\, S$$

Proof.

$$\begin{aligned}
& S \,;\, (T \,;\, S)^\omega \\
= & \{\text{definition}\} \\
& S \,;\, (\mu\, X \bullet T \,;\, S \,;\, X \sqcap \mathsf{skip}) \\
= & \{\text{rolling (Lemma 4) with } f := (\lambda X \bullet S \,;\, X) \\
& \quad \text{and } g := (\lambda X \bullet T \,;\, X \sqcap \mathsf{skip})\} \\
& (\mu\, X \bullet S \,;\, (T \,;\, X \sqcap \mathsf{skip})) \\
= & \{S \text{ conjunctive}\} \\
& (\mu\, X \bullet S \,;\, T \,;\, X \sqcap S) \\
= & \{\text{Lemma 7}\} \\
& (S \,;\, T)^\omega \,;\, S
\end{aligned}$$

and the derivation for weak iteration is similar. □

By choosing $T := \mathsf{skip}$ in Lemma 11 we get the following as a special case:

$$S \,;\, S^\omega = S^\omega \,;\, S \qquad \text{and} \qquad S \,;\, S^* = S^* \,;\, S$$

Next we have the *decomposition* property.

Lemma 12 (Decomposition). *Assume that S and T are monotonic predicate transformers and that S is conjunctive. Then*

$$(S \sqcap T)^\omega \;=\; S^\omega \,;\, (T \,;\, S^\omega)^\omega \qquad \text{and} \qquad (S \sqcap T)^* \;=\; S^* \,;\, (T \,;\, S^*)^*$$

Proof.

$$\begin{array}{ll} & (S \sqcap T)^\omega \\ = & \{\text{definition}\} \\ & (\mu\, X \bullet (S \sqcap T) \,;\, X \sqcap \mathsf{skip}) \\ = & \{\text{distributivity}\} \\ & (\mu\, X \bullet S \,;\, X \sqcap T \,;\, X \sqcap \mathsf{skip}) \\ = & \{\text{diagonalisation (Lemma 5)}\} \\ & (\mu\, X \bullet (\mu\, Y \bullet S \,;\, Y \sqcap T \,;\, X \sqcap \mathsf{skip})) \\ = & \{\text{Lemma 7}\} \\ & (\mu\, X \bullet S^\omega \,;\, (T \,;\, X \sqcap \mathsf{skip})) \\ = & \{\text{rolling (Lemma 4)}\} \\ & S^\omega \,;\, (\mu\, X \bullet T \,;\, S^\omega \,;\, X \sqcap \mathsf{skip}) \\ = & \{\text{definition of strong iteration}\} \\ & S^\omega \,;\, (T \,;\, S^\omega)^\omega \end{array}$$

Again, the derivation for weak iteration is similar. □

In the conjunctive case there is also a simple connection between the two iterations:

Lemma 13. *Let S be an arbitrary conjunctive predicate transformer. Then $S^\omega = \{\mu.\, S\} \,;\, S^*$.*

Proof. First, we show that $(\lambda q \bullet \{q\} \,;\, T)$ is continuous, for arbitrary monotonic predicate transformer T (we show the stronger fact that it distributes over arbitrary joins of predicates):

$$\begin{array}{ll} & (\lambda q \bullet \{q\} \,;\, T).\, (\cup\, i \in I \bullet q_i) \\ = & \{\beta \text{ reduction}\} \\ & \{\cup\, i \in I \bullet q_i\} \,;\, T \\ = & \{\text{distributivity}\} \end{array}$$

$$
\begin{array}{l}
\quad (\cup\, i \in I \bullet \{q_i\} \,;\, T) \\
= \{\beta \text{ reduction}\} \\
\quad (\cup\, i \in I \bullet (\lambda q \bullet \{q\} \,;\, T).\, q_i)
\end{array}
$$

Then,

$$
\begin{array}{ll}
& \{\mu.\, S\} \,;\, S^* = S^\omega \\
\equiv & \{\beta \text{ reduction, definition of } S^\omega\} \\
& (\lambda p \bullet \{p\} \,;\, S^*).\, (\mu.\, S) = (\mu\, X \bullet S \,;\, X \sqcap \mathsf{skip}) \\
\Leftarrow & \{\text{fusion (Lemma 6 (b)), continuity}\} \\
& (\lambda p \bullet \{p\} \,;\, S^*) \circ S \;=\; (\lambda X \bullet S \,;\, X \sqcap \mathsf{skip}) \circ (\lambda p \bullet \{p\} \,;\, S^*) \\
\equiv & \{\text{definition of composition, pointwise extension, } \beta \text{ reduction}\} \\
& (\forall p \bullet \{S.\, p\} \,;\, S^* \;=\; S \,;\, \{p\} \,;\, S^* \sqcap \mathsf{skip}) \\
\equiv & \{\text{unfold weak iteration}\} \\
& (\forall p \bullet \{S.\, p\} \,;\, (S \,;\, S^* \sqcap \mathsf{skip}) \;=\; S \,;\, \{p\} \,;\, S^* \sqcap \mathsf{skip}) \\
\equiv & \{\text{pointwise extension, statement definitions}\} \\
& (\forall p\, q \bullet S.\, p \cap S.\, (S^*.\, q) \cap q \;=\; S.\, (p \cap S^*.\, q) \cap q) \\
\Leftarrow & \{\text{definition of conjunctivity}\} \\
& S \text{ conjunctive}
\end{array}
$$

□

Intuitively, Lemma 13 provides a decomposition reminiscent of the classical decomposition of total correctness into termination and partial correctness. It is useful for proving properties about strong iteration by first proving a corresponding property for weak iteration (this is used in Lemma 14 below). We could also define an *infinite repetition* by $S^\infty = (\mu\, X \bullet S \,;\, X)$ and find $S^\omega = S^\infty \sqcap S^*$ when S is conjunctive (for a more detailed investigation of the infinite repetition, we refer to [9]).

3.5 Commutativity properties

Later in this paper, commutativity properties of statements will play an important role. The following lemma shows how a generalised commutativity is inherited by assertions and iterations.

Lemma 14. *Assume that S, T and U are monotonic predicate transformers with $S \,;\, T \sqsubseteq U \,;\, S$. Then*

(a) $S \,;\, T^* \sqsubseteq U^* \,;\, S$
(b) $S \,;\, \{\mu.\, T\} \sqsubseteq \{\mu.\, U\} \,;\, S$ *if S is continuous*
(c) $S \,;\, T^\omega \sqsubseteq U^\omega \,;\, S$ *if T and U are conjunctive and S is continuous.*

Proof. First we prove (a):

$$
\begin{array}{ll}
& S \,;\, T^* \sqsubseteq U^* \,;\, S \\
\Leftarrow & \{\text{induction}\} \\
& S \,;\, T^* \sqsubseteq U \,;\, S \,;\, T^* \sqcap S \\
\equiv & \{\text{unfold}\} \\
& S \,;\, (T \,;\, T^* \sqcap \mathsf{skip}) \sqsubseteq U \,;\, S \,;\, T^* \sqcap S \\
\Leftarrow & \{\text{general rule } S \,;\, (T \sqcap U) \sqsubseteq S \,;\, T \sqcap S \,;\, U\} \\
& S \,;\, T \,;\, T^* \sqcap S \sqsubseteq U \,;\, S \,;\, T^* \sqcap S \\
\Leftarrow & \{\text{monotonicity of ; and } \sqcap\} \\
& S \,;\, T \sqsubseteq U \,;\, S
\end{array}
$$

Now, (b)

$$
\begin{array}{ll}
& S \,;\, \{\mu.\, T\} \sqsubseteq \{\mu.\, U\} \,;\, S \\
\equiv & \{\text{definitions}\} \\
& (\forall q \bullet S.\, (\mu.\, T \cap q) \subseteq \mu.\, U \cap S.\, q) \\
\Leftarrow & \{\text{general rule } S.\, (p \cap q) \subseteq S.\, p \cap S.\, q\} \\
& (\forall q \bullet S.\, (\mu.\, T) \cap S.\, q \subseteq \mu.\, U \cap S.\, q) \\
\Leftarrow & \{\text{monotonicity of } \cap\} \\
& S.\, (\mu.\, T) \subseteq \mu.\, U \\
\Leftarrow & \{\text{fusion (Lemma 6 (a))}\} \\
& S \,;\, T \sqsubseteq U \,;\, S
\end{array}
$$

Finally, for (c) we have, assuming $S \,;\, T \sqsubseteq U \,;\, S$

$$
\begin{array}{ll}
& S \,;\, T^\omega \\
= & \{\text{Lemma 13}\} \\
& S \,;\, \{\mu.\, T\} \,;\, T^* \\
\sqsubseteq & \{\text{part (b) of this lemma}\} \\
& \{\mu.\, U\} \,;\, S \,;\, T^* \\
\sqsubseteq & \{\text{part (a) of this lemma}\} \\
& \{\mu.\, U\} \,;\, U^* \,;\, S \\
= & \{\text{Lemma 13}\} \\
& U^\omega \,;\, S
\end{array}
$$

□

We now turn to a more specific kind of commutation. We say that S *commutes over* T if $S \,;\, T \sqsubseteq T \,;\, S$.

Lemma 15. *Assume that S is monotonic, T is conjunctive, and $S;T \sqsubseteq T;S$. Then*

(a) $S^\omega ; T \sqsubseteq T ; S^\omega$
(b) $(S \sqcap T)^\omega = S^\omega ; T^\omega$ *if S is continuous*
(c) $(S \sqcap T)^\omega = S^\omega ; T^\omega$ *if $T^\omega = T^*$.*

Intuitively speaking, Lemma 15 (b) and (c) show under what assumptions we can execute a mix of S and T so that all executions of S come first.

Proof. First, we have

$$
\begin{array}{ll}
& S^\omega ; T \sqsubseteq T ; S^\omega \\
\Leftarrow & \{\text{induction}\} \\
& S ; T ; S^\omega \sqcap T \sqsubseteq T ; S^\omega \\
\equiv & \{\text{unfold, conjunctivity}\} \\
& S ; T ; S^\omega \sqcap T \sqsubseteq T ; S ; S^\omega \sqcap T \\
\Leftarrow & \{\text{monotonicity of ; and } \sqcap\} \\
& S ; T \sqsubseteq T ; S
\end{array}
$$

which proves (a). We then prove (b) and (c) together. First,

$$
\begin{array}{ll}
& (S \sqcap T)^\omega \\
\equiv & \{\text{general rule } S^\omega ; S^\omega = S^\omega \text{ (Lemma 9 (c))}\} \\
& (S \sqcap T)^\omega ; (S \sqcap T)^\omega \\
\sqsubseteq & \{\text{general lattice rule } x \sqcap y \sqsubseteq x, \text{ monotonicity}\} \\
& S^\omega ; T^\omega
\end{array}
$$

and for the converse refinement we have

$$
\begin{array}{ll}
& S^\omega ; T^\omega \sqsubseteq (S \sqcap T)^\omega \\
\Leftarrow & \{\text{induction}\} \\
& S ; (S \sqcap T)^\omega \sqcap T^\omega \sqsubseteq (S \sqcap T)^\omega \\
\equiv & \{\text{decomposition (Lemma 12)}\} \\
& S ; T^\omega ; (S ; T^\omega)^\omega \sqcap T^\omega \sqsubseteq T^\omega ; (S ; T^\omega)^\omega \\
\equiv & \{\text{unfold, } T \text{ conjunctive}\} \\
& S ; T^\omega ; (S ; T^\omega)^\omega \sqcap T^\omega \sqsubseteq T^\omega ; S ; T^\omega ; (S ; T^\omega)^\omega \sqcap T^\omega \\
\Leftarrow & \{(*)\} \\
& S ; T^\omega ; (S ; T^\omega)^\omega \sqcap T^\omega \sqsubseteq S ; T^\omega ; T^\omega ; (S ; T^\omega)^\omega \sqcap T^\omega \\
\Leftarrow & \{\text{monotonicity of ; and } \sqcap\} \\
& T^\omega \sqsubseteq T^\omega ; T^\omega \\
\equiv & \{\text{general rule } S^\omega ; S^\omega = S^\omega \text{ (Lemma 9 (c))}\} \\
& \mathsf{T}
\end{array}
$$

where the justification marked with an asterisk uses the assumption S ; $T \sqsubseteq T$; S and Lemma 14 (c) for (b), and the assumptions $T^\omega = T^*$ and S ; $T \sqsubseteq T$; S and Lemma 14 (a) for (c). □

3.6 Data refinement

Data refinement can at an abstract level be described as a commutativity property: we say that S *is data refined through* D *by* S' if the following condition holds:

$$D \,;\, S \sqsubseteq S' \,;\, D$$

where $D : \mathsf{Mtran}(\Sigma', \Sigma)$ (the *decoding*), $S : \mathsf{Mtran}(\Sigma, \Sigma)$ (the *abstract statement*) and $S' : \mathsf{Mtran}(\Sigma', \Sigma')$ (the *concrete statement*) are monotonic predicate transformers. Intuitively speaking, D models a data abstraction in the sense that it replaces a concrete data structure over Σ' with an abstract data structure over Σ. More details about this algebraic view on data refinement can be found elsewhere [17, 27]. Here we concentrate on the algebraic interaction between data refinement and iterations and loops.

From Lemma 14 we immediately see how data refinement is inherited by iterations:

Theorem 16. *Assume that S, T and D are monotonic predicate transformers such that D ; $S \sqsubseteq T$; D. Then*

(a) D ; $S^* \sqsubseteq T^*$; D
(b) D ; $S^\omega \sqsubseteq T^\omega$; D *if S and T are conjunctive and D is continuous.*

Thus we can say that the weak iteration always *preserves data refinement* while strong (conjunctive) iteration preserves data refinement provided that the decoding is continuous. We return to an interpretation of this result when considering data refinement of action systems (Sect. 4.3) and loops (Sect. 6.3).

4 Action systems

We shall now apply the results for iterations to constructs that are more directly useful in programming. We begin with *action systems* and later (in Sect. 6) move on to loop constructs that correspond directly to the while-loops of traditional sequential programming languages.

4.1 Action systems

An *action system* is defined as follows

$$\begin{aligned}&\mathsf{do}\ A_1 \,[\!]\, \cdots \,[\!]\, A_n\ \mathsf{od}\\ &\quad \triangleq\ (\mu X \bullet A_1 \,;\, X \sqcap \cdots \sqcap A_n \,;\, X \sqcap [\neg \mathsf{gd}.\, A_1 \cap \cdots \cap \neg \mathsf{gd}.\, A_n])\end{aligned}$$

where $A_1, \ldots, A_n$ (the *actions*) are conjunctive predicate transformers. The reason for choosing a least rather than a greatest fixpoint in this definition is that a least fixpoint corresponds to semantically identifying infinite unfolding with abort.

Using iterations we can rewrite the action system in a more convenient form:

$$\mathsf{do}\ A_1 \,[\!]\, \cdots \,[\!]\, A_n\ \mathsf{od} = (A_1 \sqcap \cdots \sqcap A_n)^\omega \,;\, [\neg\, \mathsf{gd}.\, A_1 \cap \cdots \cap \neg\, \mathsf{gd}.\, A_n]$$

Intuitively, the action system is like a strong iteration of the choice $A_1 \sqcap \cdots \sqcap A_n$, but when no action is enabled, then the action system terminates. We say that action A is *enabled* when the *guard* $\mathsf{gd}.\, A = \neg\, A.\, \mathsf{false}$ holds. The guard statement at the end makes sure that the iteration is not terminated until all actions are disabled.

An action system can be seen as modeling a parallel program, in the following way. If the guards of both actions are true and there are no read-write or write-write conflicts between two actions A_i and A_j, then A_i and A_j can be executed in any order. This, in turn, means that we can view them as executed in parallel, with an interleaving semantics for parallelism. This is the essence of the action system approach to parallel algorithms. The action system approach is described in more detail in [7]. It is similar to the UNITY approach [13] but it does not assume fairness and it permits the action bodies to be arbitrarily complex statements.

Here we concentrate on the algebraic properties of action systems. Since the action system can be described in terms of a strong iteration, it should be possible to derive properties for action systems from corresponding properties of strong iterations. How well this works in practice depends on how well we can handle the added guard statement at the end.

Before we consider general properties of action systems, we note the following properties of simple action systems:

Lemma 17. *Assume that A is a monotonic predicate transformer. Then*

(a) $(\mathsf{do}\ A\ \mathsf{od}).\, \mathsf{true} = \mu.\, A$, *and*
(b) $(\mathsf{do}\ A\ \mathsf{od}).\, \mathsf{false} = \mathsf{false}$.

Proof. For (a) we have

$$
\begin{array}{ll}
 & (\mathsf{do}\ A\ \mathsf{od}).\,\mathsf{true} = \mu.\,A \\
\equiv & \{\text{definitions}\} \\
 & (\mu\,X \bullet A\ ;\ X \sqcap [\neg\,\mathsf{gd}.\,A]).\,\mathsf{true} = \mu.\,A \\
\Leftarrow & \{\text{fusion (Lemma 6 (b)) with } h := (\lambda X \bullet X.\,\mathsf{true})\} \\
 & (\forall X \bullet (A\ ;\ X \sqcap [\neg\,\mathsf{gd}.\,A]).\,\mathsf{true} = A.\,(X.\,\mathsf{true})) \\
\equiv & \{\text{definitions, simplification}\} \\
 & (\forall X \bullet A.\,(X.\,\mathsf{true}) \cap (\mathsf{gd}.\,A \cup \mathsf{true}) = A.\,(X.\,\mathsf{true})) \\
\equiv & \{\text{simplification}\} \\
 & \mathsf{T}
\end{array}
$$

and for (b):

$$
\begin{array}{ll}
 & (\mathsf{do}\ A\ \mathsf{od}).\,\mathsf{false} \subseteq \mathsf{false} \\
\equiv & \{\text{definitions}\} \\
 & A^{\omega}\ ;\ [\mathsf{gd}.\,A]\ ;\ \mathsf{abort} \sqsubseteq \mathsf{abort} \\
\Leftarrow & \{\text{induction}\} \\
 & A\ ;\ \mathsf{abort} \sqcap [\mathsf{gd}.\,A]\ ;\ \mathsf{abort} \sqsubseteq \mathsf{abort} \\
\equiv & \{\text{definitions}\} \\
 & A.\,\mathsf{false} \cap \mathsf{gd}.\,A \subseteq \mathsf{false} \\
\equiv & \{\text{definition } \mathsf{gd}.\,A = \neg A.\,\mathsf{false}\} \\
 & \mathsf{T}
\end{array}
$$

□

4.2 Basic properties

We start by lifting general properties of strong iterations to action systems. To keep things simple, we generally consider action systems with one or two actions. In general, the results that we prove can be generalised directly to action systems with three or more actions, because of the general property $\mathsf{do}\ A \parallel B\ \mathsf{od} = \mathsf{do}\ (A \sqcap B)\ \mathsf{od}$.

We first consider the leapfrog property.

Theorem 18 (Action system leapfrog). *Assume that A and B are conjunctive predicate transformers. Then*

$$A\ ;\ \mathsf{do}\ B\ ;\ A\ \mathsf{od} \sqsubseteq \mathsf{do}\ A\ ;\ B\ \mathsf{od}\ ;\ A$$

Proof. We have

$$
\begin{array}{ll}
& A \,;\, \mathsf{do}\ B \,;\, A\ \mathsf{od} \\
= & \{\text{rewrite using iteration}\} \\
& A \,;\, (B \,;\, A)^{\omega} \,;\, [\neg\, \mathsf{gd}.\,(B \,;\, A)] \\
= & \{\text{leapfrog (Lemma 11)}\} \\
& (A \,;\, B)^{\omega} \,;\, A \,;\, [\neg\, \mathsf{gd}.\,(B \,;\, A)] \\
\sqsubseteq & \{\text{see separate derivation below}\} \\
& (A \,;\, B)^{\omega} \,;\, [\neg\, \mathsf{gd}.\,(A \,;\, B)] \,;\, A \\
= & \{\text{rewrite using iteration}\} \\
& \mathsf{do}\ A \,;\, B\ \mathsf{od} \,;\, A
\end{array}
$$

The step that is not an equality is justified as follows:

$$
\begin{array}{ll}
& A \,;\, [\neg\, \mathsf{gd}.\,(B \,;\, A)] \sqsubseteq [\neg\, \mathsf{gd}.\,(A \,;\, B)] \,;\, A \\
\Leftarrow & \{\text{Lemma 1 (c)}\} \\
& A.\,\mathsf{true} \cap \neg\, \mathsf{gd}.\,(A \,;\, B) \subseteq A.\,(\neg\, \mathsf{gd}.\,(B \,;\, A)) \\
\equiv & \{\text{definition of guard}\} \\
& A.\,\mathsf{true} \cap A.\,(B.\,\mathsf{false}) \subseteq A.\,(B.\,(A.\,\mathsf{false})) \\
\equiv & \{A \text{ monotonic, so } A.\,\mathsf{true} \supseteq A.\,(B.\,\mathsf{false})\} \\
& A.\,(B.\,\mathsf{false}) \subseteq A.\,(B.\,(A.\,\mathsf{false})) \\
\equiv & \{A \text{ and } B \text{ are monotonic, } \mathsf{false} \subseteq A.\,\mathsf{false}\} \\
& \mathsf{T}
\end{array}
$$

□

An obvious question is now whether the other half of the leapfrog property holds: $\mathsf{do}\ A \,;\, B\ \mathsf{od} \,;\, A \sqsubseteq A \,;\, \mathsf{do}\ B \,;\, A\ \mathsf{od}$. The proof cannot be easily adjusted, since the refinement

$$[\neg\, \mathsf{gd}.\,(A \,;\, B)] \,;\, A \sqsubseteq A \,;\, [\neg\, \mathsf{gd}.\,(B \,;\, A)]$$

is not valid (if $A = \mathsf{abort}$ then the left-hand side is magic and the right-hand side is abort). On the other hand, we do not have a counterexample, so we leave this as an open question.

Next we consider decomposition.

Theorem 19 (Action system decomposition). *Assume that A and B are conjunctive predicate transformers. Then*

$$\mathsf{do}\ A \,[\!]\, B\ \mathsf{od} = \mathsf{do}\ B\ \mathsf{od} \,;\, \mathsf{do}\ (A \,;\, \mathsf{do}\ B\ \mathsf{od})\ \mathsf{od}$$

provided that $\mathsf{gd}.\,A \cap \mathsf{gd}.\,B = \mathsf{false}$.

Intuitively speaking, the condition $\mathsf{gd}.A \cap \mathsf{gd}.B = \mathsf{false}$ states that the actions A and B *exclude* each other; they cannot be enabled simultaneously.

Proof. First we rewrite the condition $\mathsf{gd}.A \cap \mathsf{gd}.B = \mathsf{false}$ into a more easily used form.

$$
\begin{array}{ll}
& \mathsf{gd}.A \cap \mathsf{gd}.B = \mathsf{false} \\
\equiv & \{\text{general shunting rule } p \cap q \subseteq r \;\equiv\; p \subseteq \neg q \cup r\} \\
& \mathsf{gd}.B \subseteq \neg \mathsf{gd}.A \\
\equiv & \{\text{definition of guard}\} \\
& \mathsf{gd}.B \subseteq A.\mathsf{false} \\
\equiv & \{A \text{ is monotonic}\} \\
& (\forall q \bullet \mathsf{gd}.B \subseteq A.q) \\
\equiv & \{\text{general lattice property } x \sqsubseteq y \;\equiv\; y = x \sqcup y\} \\
& (\forall q \bullet A.q = \mathsf{gd}.B \cup A.q) \\
\equiv & \{\text{definitions}\} \\
& A = [\neg \mathsf{gd}.B] \,;\, A
\end{array}
$$

Now,

$$
\begin{array}{ll}
& \mathsf{do}\ A \parallel B\ \mathsf{od} \\
= & \{\text{rewrite using iteration}\} \\
& (A \sqcap B)^\omega \,;\, [\neg \mathsf{gd}.A \cap \neg \mathsf{gd}.B] \\
= & \{\text{decomposition (Lemma 12)}\} \\
& B^\omega \,;\, (A \,;\, B^\omega)^\omega \,;\, [\neg \mathsf{gd}.A \cap \neg \mathsf{gd}.B] \\
= & \{\text{property of guard statement}\} \\
& B^\omega \,;\, (A \,;\, B^\omega)^\omega \,;\, [\neg \mathsf{gd}.B] \,;\, [\neg \mathsf{gd}.A] \\
= & \{\text{assumption } \mathsf{gd}.A \cap \mathsf{gd}.B = \mathsf{false}\text{, derivation above}\} \\
& B^\omega \,;\, ([\neg \mathsf{gd}.B] \,;\, A \,;\, B^\omega)^\omega \,;\, [\neg \mathsf{gd}.B] \,;\, [\neg \mathsf{gd}.A] \\
= & \{\text{leapfrog (Lemma 11), guard rules}\} \\
& B^\omega \,;\, [\neg \mathsf{gd}.B] \,;\, (A \,;\, B^\omega \,;\, [\neg \mathsf{gd}.B])^\omega \,;\, [\neg \mathsf{gd}.A] \\
= & \{\text{see separate subderivation below}\} \\
& B^\omega \,;\, [\neg \mathsf{gd}.B] \,;\, (A \,;\, B^\omega \,;\, [\neg \mathsf{gd}.B])^\omega \,;\, [\neg \mathsf{gd}.(A \,;\, B^\omega \,;\, [\neg \mathsf{gd}.B])] \\
= & \{\text{rewrite using iteration}\} \\
& \mathsf{do}\ B\ \mathsf{od} \,;\, \mathsf{do}\ (A \,;\, \mathsf{do}\ B\ \mathsf{od})\ \mathsf{od}
\end{array}
$$

where the guard manipulation is justified by the following derivation:

$$\mathsf{gd}.(A \,;\, B^\omega \,;\, [\neg \mathsf{gd}.B]) = \mathsf{gd}.A$$

$$
\begin{array}{ll}
\equiv & \{\text{definitions}\} \\
& \neg A.\,((B^\omega\,;[\neg \mathsf{gd}.\,B]).\,\mathsf{false}) = \neg A.\,\mathsf{false} \\
\Leftarrow & \{\text{functionality, } \mathsf{false} \subseteq p \text{ holds trivially for all } p\} \\
& (B^\omega\,;[\neg \mathsf{gd}.\,B]).\,\mathsf{false} \subseteq \mathsf{false} \\
\equiv & \{\text{Lemma 17 (b), using } B^\omega\,;[\neg \mathsf{gd}.\,B] = \mathsf{do}\ B\ \mathsf{od}\} \\
& \mathsf{T}
\end{array}
$$

□

The proof of Theorem 19 builds on the leapfrog and decomposition rule for iterations, but also on guard manipulations. The main proof directly suggests that the following two assumptions must hold:

$$
\begin{aligned}
A &= [\neg \mathsf{gd}.\,B]\,;A \\
[\neg \mathsf{gd}.\,A] &= [\neg \mathsf{gd}.\,(A\,;B^\omega\,;[\neg \mathsf{gd}.\,B])]
\end{aligned}
$$

Separate derivations then simplify (and possibly discharge) these guard conditions.

4.3 Data refinement

Now let us consider how the data refinement rules for iterations can be used to derive rules for action systems. The basic rule for data refinement is as follows:

Theorem 20. *Assume that A and A' are conjunctive and D is continuous. Furthermore assume that $D\,;A \sqsubseteq A'\,;D$ and $D\,;[\neg \mathsf{gd}.\,A] \sqsubseteq [\neg \mathsf{gd}.\,A']\,;D$. Then*

$$D\,;\mathsf{do}\ A\ \mathsf{od} \sqsubseteq \mathsf{do}\ A'\ \mathsf{od}\,;D$$

Proof.

$$
\begin{array}{ll}
& D\,;\mathsf{do}\ A\ \mathsf{od} \\
= & \{\text{rewrite using iteration}\} \\
& D\,;A^\omega\,;[\neg \mathsf{gd}.\,A] \\
\sqsubseteq & \{\text{assumption, data refinement rule (Theorem 16 (b))}\} \\
& (A')^\omega\,;D\,;[\neg \mathsf{gd}.\,A] \\
\sqsubseteq & \{\text{assumption}\} \\
& (A')^\omega\,;[\neg \mathsf{gd}.\,A']\,;D \\
= & \{\text{rewrite using iteration}\} \\
& \mathsf{do}\ A'\ \mathsf{od}\,;D
\end{array}
$$

□

Here the guard condition $D\,;[\neg \mathsf{gd}.\,A] \sqsubseteq [\neg \mathsf{gd}.\,A']\,;D$ can be simplified using Lemma 1 (c) or (d). The two basic forms of data refinement are when D is strict and disjunctive (forward data refinement) and when D is strict, terminating, and conjunctive (backward data refinement) [27], so we have the following alternative versions of this condition:

$D.(\mathsf{gd}.\,A) \subseteq \mathsf{gd}.\,A'$ if D is strict and disjunctive
$\neg \mathsf{gd}.\,A' \subseteq D.(\neg \mathsf{gd}.\,A)$ if D is strict, terminating, and conjunctive

Theorem 20 is easily generalised to the case with n actions. There is then one condition for each action ($D\,;A_i \sqsubseteq A_i'\,;D$) and one termination condition ($D\,;[\neg gg] \sqsubseteq [\neg gg']\,;D$), where $gg = \mathsf{gd}.\,A_1 \cup \cdots \cup \mathsf{gd}.\,A_n$.

We can also generalise Theorem 20 to allow *stuttering actions* in a data refinement.

Theorem 21. *Assume that A and B are conjunctive and that D is continuous. Furthermore assume that the following conditions hold:*

(i) $D\,;A \sqsubseteq A'\,;D$ *and* $D\,;\mathsf{skip} \sqsubseteq B\,;D$
(ii) $D\,;[\neg \mathsf{gd}.\,A] \sqsubseteq [\neg \mathsf{gd}.\,A' \cap \neg \mathsf{gd}.\,B]\,;D$
(iii) $D.\mathsf{true} \subseteq \mu.\,B$

Then

$$D\,;\mathsf{do}\ A\ \mathsf{od} \sqsubseteq \mathsf{do}\ A' \parallel B\ \mathsf{od}\,;D$$

Proof. We first note the following:

$$\begin{array}{ll} & B^\omega\,;D \\ = & \{\text{Lemma 13}\} \\ & \{\mu.\,B\}\,;B^*\,;D \\ \sqsupseteq & \{\text{assumption (i), Theorem 16 (a), } \mathsf{skip}^* = \mathsf{skip} \text{ (see Sect. 3.1)}\} \\ & \{\mu.\,B\}\,;D \\ \sqsupseteq & \{\text{assumption (iii), Lemma 1 (a)}\} \\ & D\,;\{\mathsf{true}\} \\ = & \{\{\mathsf{true}\} = \mathsf{skip}\} \\ & D \end{array}$$

We then have

$$\begin{array}{ll} & \mathsf{do}\ A' \parallel B\ \mathsf{od}\,;D \\ = & \{\text{rewrite using iteration}\} \\ & (A' \sqcap B)^\omega\,;[\neg \mathsf{gd}.\,A' \cap \neg \mathsf{gd}.\,B]\,;D \\ \sqsupseteq & \{\text{decomposition (Lemma 12), assumption (ii)}\} \end{array}$$

$$
\begin{array}{ll}
 & B^\omega \,;\, (A' \,;\, B^\omega)^\omega \,;\, D \,;\, [\neg \mathsf{gd}.\, A] \\
\sqsupseteq & \{\text{preceding derivation, Theorem 16 (b)}\} \\
 & B^\omega \,;\, D \,;\, A^\omega \,;\, [\neg \mathsf{gd}.\, A] \\
\sqsupseteq & \{\text{preceding derivation}\} \\
 & D \,;\, A^\omega \,;\, [\neg \mathsf{gd}.\, A] \\
= & \{\text{rewrite using iteration}\} \\
 & D \,;\, \mathsf{do}\ A\ \mathsf{od}
\end{array}
$$

□

The action B in Theorem 21 is called a stuttering action because it corresponds to a skip step on the abstract level. The guard condition is similar to the guard condition of Theorem 20 and can be analysed in the same way. From Lemma 17 (a) we see that the last condition states that execution of do B od must always terminate if the initial state satisfies $D.\,\mathsf{true}$ (if the decoding D is described by an abstraction relation R, then $D.\,\mathsf{true}$ characterises those concrete states for which the relation R is defined, i.e., those states that satisfy the concrete invariant).

5 Atomicity refinement

In the action system approach, the loop notation describes the *atomicity* (the granularity) of the system; the actions are considered to be executed as atomic units, without interference from other actions. A transformation that replaces an action with two or more actions reduces the granularity and is called an *atomicity refinement*. For example, the decomposition rule (Theorem 19) can be seen as a simple rule of atomicity refinement when it is read from right to left.

5.1 Properties of actions

Before we formulate the conditions under which the atomicity is refined, we introduce some intuitively appealing ways of describing conditions:

- A *always disables* B if $A.\,\mathsf{true} \subseteq A.\,(\neg \mathsf{gd}.\, B)$
- A *excludes* B if $\mathsf{gd}.\, A \cap \mathsf{gd}.\, B = \mathsf{false}$
- A *does not enable* B if $\neg \mathsf{gd}.\, B \subseteq A.\,(\neg \mathsf{gd}.\, B)$
- A *does not disable* B if $\mathsf{gd}.\, B \subseteq A.\,(\mathsf{gd}.\, B)$
- *iteration of* A *always terminates* if $(\mathsf{do}\ A\ \mathsf{od}).\,\mathsf{true} = \mathsf{true}$

In order to carry out manipulation exclusively on the predicate transformer level (rather than on the predicate level), we reformulate these conditions in terms of actions directly.

Always disabling A always disables B if and only if $A \,;\, [\mathsf{gd}.\, B] = A$. To see this, we first have

$$\begin{array}{ll} & A \,;\, [\neg \mathsf{gd}.\, B] \sqsubseteq A \\ \equiv & \{\mathsf{skip} = [\mathsf{true}]\} \\ & A \,;\, [\neg \mathsf{gd}.\, B] \sqsubseteq [\mathsf{true}] \,;\, A \\ \equiv & \{\text{Lemma 1 (c)}\} \\ & A.\, \mathsf{true} \cap \mathsf{true} \subseteq A.\, (\neg \mathsf{gd}.\, B) \\ \equiv & \{\text{lattice property}\} \\ & A.\, \mathsf{true} \subseteq A.\, (\neg \mathsf{gd}.\, B) \end{array}$$

The reverse refinement $A \,;\, [\neg \mathsf{gd}.\, B] \sqsupseteq A$ always holds trivially, because of the general rule $\mathsf{skip} \sqsubseteq [p]$.

Exclusion A excludes B if and only if $A = [\neg \mathsf{gd}.\, B] \,;\, A$. This was already shown as part of the proof of Theorem 19.

Nonenabling and nondisabling Lemma 1 (b) gives us

$$[\mathsf{gd}.\, B] \,;\, A \sqsubseteq A \,;\, [\mathsf{gd}.\, B] \equiv \neg \mathsf{gd}.\, B \subseteq A.\, (\neg \mathsf{gd}.\, B)$$

and

$$[\neg \mathsf{gd}.\, B] \,;\, A \sqsubseteq A \,;\, [\neg \mathsf{gd}.\, B] \equiv \mathsf{gd}.\, B \subseteq A.\, (\mathsf{gd}.\, B)$$

Terminating iteration If iteration of A always terminates then $A^\omega = A^*$. This is seen as follows:

$$\begin{array}{ll} & A^\omega = A^* \\ \equiv & \{\text{Lemma 13}\} \\ & \{\mu.\, A\} \,;\, A^* = A^* \\ \Leftarrow & \{\{\mathsf{true}\} = \mathsf{skip}\} \\ & \mu.\, A = \mathsf{true} \\ \Leftarrow & \{\text{Lemma 17 (a)}\} \\ & \mathsf{do}\ A\ \mathsf{od}.\, \mathsf{true} = \mathsf{true} \end{array}$$

Implication in the other direction does not hold in general, but it does hold when A is terminating (i.e., when $A.\, \mathsf{true} = \mathsf{true}$). To see this, we have

$$\begin{array}{ll} & A^\omega = A^* \\ \equiv & \{\text{Lemma 13}\} \\ & \{\mu.\, A\} \,;\, A^* = A^* \end{array}$$

$$
\begin{array}{ll}
\equiv & \{\text{definitions}\} \\
& (\forall q \bullet \mu.\, A \cap A^*.\, q = A^*.\, q) \\
\equiv & \{\text{general lattice property } x \sqcap y = y \equiv y \sqsubseteq x\} \\
& (\forall q \bullet A^*.\, q \subseteq \mu.\, A) \\
\Rightarrow & \{\text{specialise } q := \mathsf{true}\} \\
& A^*.\, \mathsf{true} \subseteq \mu.\, A \\
\equiv & \{A.\, \mathsf{true} = \mathsf{true} \;\Rightarrow\; A^*.\, \mathsf{true} = \mathsf{true} \text{ by derivation below}\} \\
& \mathsf{true} \subseteq \mu.\, A
\end{array}
$$

where the last step is justified by the following derivation

$$
\begin{array}{ll}
& \mathsf{true} \subseteq A^*.\, \mathsf{true} \\
\equiv & \{\text{definitions}\} \\
& \mathsf{magic} \sqsubseteq A^* \,;\, \mathsf{magic} \\
\Leftarrow & \{\text{induction}\} \\
& \mathsf{magic} \sqsubseteq A \,;\, \mathsf{magic} \sqcap \mathsf{magic} \\
\equiv & \{\mathsf{magic} \text{ is top element, definitions}\} \\
& \mathsf{true} \subseteq A.\, \mathsf{true}
\end{array}
$$

5.2 The atomicity refinement theorem

The atomicity refinement setup involves an initialisation I and five actions: the *major action* A, the *minor action* B, the *left mover* L, the *right mover* R, and the *environment action* E. All of these are assumed to be conjunctive. The aim is an atomicity refinement of the form

$$I \,;\, \mathsf{do}\ (A \,;\, \mathsf{do}\ B\ \mathsf{od}) \parallel L \parallel R \parallel E\ \mathsf{od} \sqsubseteq I \,;\, \mathsf{do}\ A \parallel B \parallel L \parallel R \parallel E\ \mathsf{od}$$

Note how execution of $(A \,;\, \mathsf{do}\ B\ \mathsf{od})$ is split up into separate executions of A and B interleaved with the other actions (E, L, and R). For more details about the interpretation of this refinement and its application, we refer to [4].

The conditions for atomicity refinement are the following:

(a) I always disables B
(b) A and E both exclude B
(c) L and E do not enable B and R does not disable B
(d) L commutes over each of A, B and R; and B commutes over R
(e) iteration of R always terminates
(f) L is continuous

We call the actions L and R (left and right) movers because of their commutativity properties. The major and minor actions (A and B) are the ones directly involved in the atomicity refinement, while E models environment actions (which are not involved in the atomicity refinement and do not interfere with it).

The derivations in Sect. 5.1 show that the conditions (a)–(e) give us the following algebraic conditions that can be used in the proof:

$$
\begin{array}{ll}
\text{(a)} & I = I \,;\, [\neg \mathsf{gd}.\, B] \\
\text{(b)} & A = [\neg \mathsf{gd}.\, B] \,;\, A \\
 & E = [\neg \mathsf{gd}.\, B] \,;\, E \\
\text{(c)} & [\mathsf{gd}.\, B] \,;\, L \sqsubseteq L \,;\, [\mathsf{gd}.\, B] \\
 & [\mathsf{gd}.\, B] \,;\, E \sqsubseteq E \,;\, [\mathsf{gd}.\, B] \\
 & [\neg \mathsf{gd}.\, B] \,;\, R \sqsubseteq R \,;\, [\neg \mathsf{gd}.\, B] \\
\text{(d)} & L \,;\, A \sqsubseteq A \,;\, L \\
 & L \,;\, B \sqsubseteq B \,;\, L \\
 & L \,;\, R \sqsubseteq R \,;\, L \\
 & B \,;\, R \sqsubseteq R \,;\, B \\
\text{(e)} & R^{\omega} = R^{*}
\end{array}
$$

Expressed in this way, the conditions can be used efficiently in algebraic manipulations. The theorem is now as follows.

Theorem 22. *Assume that I, A, B, E, L and R are actions satisfying the conditions (a)–(f) above. Then the following refinement holds:*

$$I \,;\, \mathsf{do}\ (A \,;\, \mathsf{do}\ B\ \mathsf{od}) \parallel L \parallel R \parallel E\ \mathsf{od} \sqsubseteq I \,;\, \mathsf{do}\ A \parallel B \parallel L \parallel R \parallel E\ \mathsf{od}$$

Proof. We abbreviate $\neg \mathsf{gd}.\, A \cap \neg \mathsf{gd}.\, L \cap \neg \mathsf{gd}.\, R \cap \neg \mathsf{gd}.\, E$ by q and we omit the semicolon for sequential composition, to make formulas easier to handle. We have

$$
\begin{array}{ll}
 & I \,;\, \mathsf{do}\ A \parallel B \parallel L \parallel R \parallel E\ \mathsf{od} \\
= & \{\text{definition of loop, hide semicolons}\} \\
 & I(A \sqcap B \sqcap L \sqcap R \sqcap E)^{\omega}[\neg \mathsf{gd}.\, B \cap q] \\
= & \{\text{decomposition (Lemma 12)}\} \\
 & I(A \sqcap B \sqcap L \sqcap R)^{\omega}(E(A \sqcap B \sqcap L \sqcap R)^{\omega})^{\omega}[\neg \mathsf{gd}.\, B \cap q] \\
= & \{\text{commutativity assumptions for } L\text{, Lemma 15}\} \\
 & IL^{\omega}(A \sqcap B \sqcap R)^{\omega}(EL^{\omega}(A \sqcap B \sqcap R)^{\omega})^{\omega}[\neg \mathsf{gd}.\, B \cap q] \\
= & \{\text{decomposition (Lemma 12)}\} \\
 & IL^{\omega}(B \sqcap R)^{\omega}(A(B \sqcap R)^{\omega})^{\omega}(EL^{\omega}(B \sqcap R)^{\omega}(A(B \sqcap R)^{\omega})^{\omega})^{\omega}
\end{array}
$$

$[\neg \mathsf{gd}.\, B \cap q]$

$= \{BR \sqsubseteq RB \text{ and } R^\omega = R^* \text{ and Lemma 15}\}$
$IL^\omega B^\omega R^\omega (AB^\omega R^\omega)^\omega (EL^\omega B^\omega R^\omega (AB^\omega R^\omega)^\omega)^\omega [\neg \mathsf{gd}.\, B \cap q]$

$\sqsupseteq \{\text{Lemma 23 (a)}\}$
$IL^\omega B^\omega R^\omega (AB^\omega R^\omega)^\omega ([\neg \mathsf{gd}.\, B] EL^\omega R^\omega (AB^\omega R^\omega)^\omega)^\omega [\neg \mathsf{gd}.\, B \cap q]$

$= \{[\neg \mathsf{gd}.\, B \cap q] = [\neg \mathsf{gd}.\, B]\,;\, [q]\text{, leapfrog (Lemma 11 (b))}\}$
$IL^\omega B^\omega R^\omega (AB^\omega R^\omega)^\omega [\neg \mathsf{gd}.\, B] (EL^\omega R^\omega (AB^\omega R^\omega)^\omega [\neg \mathsf{gd}.\, B])^\omega [q]$

$\sqsupseteq \{\text{assumption } A = [\neg \mathsf{gd}.\, B]A\text{, leapfrog (Lemma 11 (b))}\}$
$IL^\omega B^\omega R^\omega [\neg \mathsf{gd}.\, B] (AB^\omega R^\omega [\neg \mathsf{gd}.\, B])^\omega$
$\quad (EL^\omega R^\omega [\neg \mathsf{gd}.\, B] (AB^\omega R^\omega [\neg \mathsf{gd}.\, B])^\omega)^\omega [q]$

$\sqsupseteq \{\text{assumption } R[\neg \mathsf{gd}.\, B] \sqsupseteq [\neg \mathsf{gd}.\, B]R\text{, Lemma 14 (a),}$
$\quad \text{assumption } R^\omega = R^*,\ [p] \sqsupseteq \mathsf{skip}\}$
$IL^\omega B^\omega R^\omega (AB^\omega [\neg \mathsf{gd}.\, B] R^\omega)^\omega (EL^\omega R^\omega (AB^\omega [\neg \mathsf{gd}.\, B] R^\omega)^\omega)^\omega [q]$

$\sqsupseteq \{\text{Lemma 23 (b) }\}$
$IL^\omega R^\omega (AB^\omega [\neg \mathsf{gd}.\, B] R^\omega)^\omega (EL^\omega R^\omega (AB^\omega [\neg \mathsf{gd}.\, B] R^\omega)^\omega)^\omega [q]$

$= \{\text{decomposition (Lemma 12)}\}$
$IL^\omega (AB^\omega [\neg \mathsf{gd}.\, B] \sqcap R)^\omega (EL^\omega (AB^\omega [\neg \mathsf{gd}.\, B] \sqcap R)^\omega)^\omega [q]$

$= \{\text{Lemma 23 (c) }\}$
$I(AB^\omega [\neg \mathsf{gd}.\, B] \sqcap L \sqcap R)^\omega (E(AB^\omega [\neg \mathsf{gd}.\, B] \sqcap L \sqcap R)^\omega)^\omega [q]$

$= \{\text{decomposition (Lemma 12)}\}$
$I(AB^\omega [\neg \mathsf{gd}.\, B] \sqcap L \sqcap R \sqcap E)^\omega [q]$

$= \{\text{Lemma 23 (d), definition of loop}\}$
$I\,;\, \mathsf{do}\ (A\,;\, \mathsf{do}\ B\ \mathsf{od}) \parallel L \parallel R \parallel E\ \mathsf{od} \quad \square$

The lemmas used in the proof are then proved as follows.

Lemma 23. *Under the assumptions that are made for the atomicity refinement theorem,*

(a) $EL^\omega B^\omega \sqsupseteq [\neg \mathsf{gd}.\, B] EL^\omega$
(b) $IL^\omega B^\omega \sqsupseteq IL^\omega$
(c) $L^\omega (AB^\omega [\neg \mathsf{gd}.\, B] \sqcap R)^\omega = (AB^\omega [\neg \mathsf{gd}.\, B] \sqcap L \sqcap R)^\omega$
(d) $\mathsf{gd}.\, (A\,;\, \mathsf{do}\ B\ \mathsf{od}) = \mathsf{gd}.\, A$

Proof. For (a) we have

$EL^\omega B^\omega$
$= \{\text{assumption (b)}\}$
$[\neg \mathsf{gd}.\, B] EL^\omega B^\omega$

$\sqsupseteq$ {$[p] = [p]$; $[p]$, assumption (c) and Lemma 2}
$[\neg \mathsf{gd}.\ B]E[\neg \mathsf{gd}.\ B]L^\omega B^\omega$
$\sqsupseteq$ {assumption (c) and (f), Lemmas 2 and 14 (c)}
$[\neg \mathsf{gd}.\ B]EL^\omega[\neg \mathsf{gd}.\ B]B^\omega$
$\sqsupseteq$ {general rules $[\neg \mathsf{gd}.\ B]B^\omega = [\neg \mathsf{gd}.\ B]$ (Lemma 10) and $[p] \sqsupseteq \mathsf{skip}$}
$[\neg \mathsf{gd}.\ B]EL^\omega$

Next, (b):

$IL^\omega B^\omega$
$=$ {assumption (a)}
$I[\neg \mathsf{gd}.\ B]L^\omega B^\omega$
$\sqsupseteq$ {assumption (c) and (f), Lemmas 2 and 14 (c)}
$IL^\omega[\neg \mathsf{gd}.\ B]B^\omega$
$\sqsupseteq$ {general rule $B = [\mathsf{gd}.\ B]\ ;\ B$}
$IL^\omega[\neg \mathsf{gd}.\ B]([\mathsf{gd}.\ B]B)^\omega$
$\sqsupseteq$ {Lemma 10, general rule $[p] \sqsupseteq \mathsf{skip}$}
IL^ω

Now, (c) We have

$L(AB^\omega[\neg \mathsf{gd}.\ B] \sqcap R)$
$\sqsubseteq$ {L conjunctive}
$LAB^\omega[\neg \mathsf{gd}.\ B] \sqcap LR$
$\sqsubseteq$ {assumption (d), Lemma 14}
$AB^\omega L[\neg \mathsf{gd}.\ B] \sqcap RL$
$\sqsubseteq$ {assumption (c), Lemma 2}
$AB^\omega[\neg \mathsf{gd}.\ B]L \sqcap RL$
$=$ {distributivity}
$(AB^\omega[\neg \mathsf{gd}.\ B] \sqcap R)L$

so $L^\omega(AB^\omega[\neg \mathsf{gd}.\ B] \sqcap R)^\omega = (AB^\omega[\neg \mathsf{gd}.\ B] \sqcap L \sqcap R)^\omega$ follows by assumption (f) and Lemma 15 (b).

Finally, we prove (d):

$\mathsf{gd}.\ (A\ ;\ \mathsf{do}\ B\ \mathsf{od})$
$=$ {definitions}
$\neg A.\ ((\mathsf{do}\ B\ \mathsf{od}).\ \mathsf{false})$
$=$ {Lemma 17 (b)}

$$
\begin{array}{ll}
& \neg A.\ \mathsf{false} \\
= & \{\text{definition of action guard}\} \\
& \mathsf{gd}.\ A
\end{array}
$$

□

6 Loops

As noted before, the iteration constructs are unguarded, i.e., the termination of an iteration S^ω or S^* is decided by a demonic choice, rather than by evaluation of a guard predicate. In an action system the iteration is guarded, but the guard is implicit in the action. We now consider the traditional *loop* construct where the guard is explicit in the syntax.

We use the traditional definition of a loop as a least fixpoint:

$$
\begin{array}{l}
\mathsf{do}\ g_1 \to S_1 \ [\!]\ \cdots\ [\!]\ g_n \to S_n\ \mathsf{od} \\
\quad \stackrel{\wedge}{=}\ (\mu X \bullet [g_1]\ ;\ S_1\ ;\ X \sqcap \cdots \sqcap [g_n]\ ;\ S_n\ ;\ X \sqcap [\neg g_1 \cap \cdots \cap \neg g_n])
\end{array}
$$

Intuitively, the loop is executed in the following way. First, all the guards g_i are evaluated. If all guards are false, then the loop has terminated.

Otherwise, one of the bodies S_i for which the guard g_i was true, is executed. If execution of this body terminates normally, then the guards are again evaluated etc. Exactly as for action systems, infinite execution corresponds to aborting, since the definition uses a least fixpoint.

Using iterations, we can rewrite the definition of the loop as follows:

$$
\begin{array}{l}
\mathsf{do}\ g_1 \to S_1 \ [\!]\ \cdots\ [\!]\ g_n \to S_n\ \mathsf{od} \\
\quad =\ ([g_1]\ ;\ S_1 \sqcap \cdots \sqcap [g_n]\ ;\ S_n)^\omega\ ;\ [\neg g_1 \cap \cdots \cap \neg g_n]
\end{array}
$$

This means that we can use the rules for strong iterations to derive rules for loops. Before we do that, we consider the relationship between action systems and loops in some more detail.

6.1 Loops and action systems

The most important difference between a loop and an action system is that the guard is explicit in the loop. If the body S in the loop $\mathsf{do}\ g \to S\ \mathsf{od}$ is strict, then we have the following:

$$
\begin{array}{ll}
& \neg([g]\ ;\ S).\ \mathsf{false} \\
= & \{\text{definitions}\} \\
& g \cap \neg S.\ \mathsf{false} \\
= & \{S \text{ assumed strict}\} \\
& g
\end{array}
$$

This shows that $\mathsf{gd}.([g];S) = g$, which means that the loop $\mathsf{do}\ g \to S\ \mathsf{od}$ and the action system $\mathsf{do}\ [g]\,;S\ \mathsf{od}$ are the same predicate transformer. However, if the loop body is not strict, then the loop can terminate miraculously which is something no action system can. A loop with a nonstrict body cannot be written as an action system, which means that loops are more general that action systems. However, this does not mean that the rules for action systems are direct consequences of the corresponding rules for loops, since some rules for loops are more detailed (they include separate assumptions on the guard predicate).

6.2 Leapfrog and decomposition for loops

For action systems, only a weak version of the leapfrog rule was proved (Theorem 18). For loops, however, we can prove a stronger version.

Theorem 24 (Loop leapfrog). *Assume that S and T are conjunctive predicate transformers.*

(a) *If $g \subseteq S.\,h$ and $\neg g \subseteq S.\,(\neg h)$ then*

$$\mathsf{do}\ g \to S\,;T\ \mathsf{od}\,;S \sqsubseteq S\,;\mathsf{do}\ h \to T\,;S\ \mathsf{od}$$

(b) *If $S.\,\mathsf{true} \cap g \subseteq S.\,h$ and $S.\,\mathsf{true} \cap \neg g \subseteq S.\,(\neg h)$ then*

$$\mathsf{do}\ g \to S\,;T\ \mathsf{od}\,;S \sqsupseteq S\,;\mathsf{do}\ h \to T\,;S\ \mathsf{od}$$

Proof. For (a) we have

$$\begin{array}{ll} & \mathsf{do}\ g \to S\,;T\ \mathsf{od}\,;S \\ = & \{\text{rewrite using iteration}\} \\ & ([g]\,;S\,;T)^{\omega}\,;[\neg g]\,;S \\ \sqsubseteq & \{\text{Lemma 1 (b), assumptions (*)}\} \\ & (S\,;[h]\,;T)^{\omega}\,;S\,;[\neg h] \\ = & \{\text{leapfrog (Lemma 11)}\} \\ & S\,;([h]\,;T\,;S)^{\omega}\,;[\neg h] \\ = & \{\text{rewrite using iteration}\} \\ & S\,;\mathsf{do}\ h \to T\,;S\ \mathsf{od} \end{array}$$

For (b) the derivation is similar, but in the step marked (*) we get refinement in the opposite direction from Lemma 1 (c). □

The proof of Theorem 24 builds on the leapfrog rule for iterations and on propagating guards. The assumptions in both (a) and (b) are derived

directly from the need to propagate guards and Lemma 1. We can also give an intuitive interpretation to the conditions. For (a) we require that if g ($\neg g$) holds before S is executed, then h ($\neg h$) holds afterwards. For (b) the conditions are similar, but they do not require that S is terminating. Note that for action systems we had refinement in one direction only (Theorem 18), but then without guard conditions.

Note that the assumption in Theorem 24 (a) is stronger than the one in (b). Thus we have an equality rule as an immediate consequence:

Corollary 25. *Assume that S and T are conjunctive predicate transformers. If $g \subseteq S.\, h$ and $\neg g \subseteq S.\, (\neg h)$ then*

$$\mathsf{do}\ g \to S\ ;\ T\ \mathsf{od}\ ;\ S = S\ ;\ \mathsf{do}\ h \to T\ ;\ S\ \mathsf{od}$$

The decomposition rule can be directly generalised to loops, in the same way as for action systems.

Theorem 26 (Loop decomposition). *Assume that S and T are conjunctive predicate transformers. Then*

$$\begin{aligned}&\mathsf{do}\ g \cap \neg h \to S \parallel h \to T\ \mathsf{od}\\ &\quad = \ \mathsf{do}\ h \to T\ \mathsf{od}\ ;\ \mathsf{do}\ g \to (S\ ;\ \mathsf{do}\ h \to T\ \mathsf{od})\ \mathsf{od}\end{aligned}$$

Proof.

$$\begin{aligned}&\mathsf{do}\ h \to T\ \mathsf{od}\ ;\ \mathsf{do}\ g \to (S\ ;\ \mathsf{do}\ h \to T\ \mathsf{od})\ \mathsf{od}\\ &= \{\text{rewrite using iteration}\}\\ &\quad ([h]\ ;\ T)^\omega\ ;\ [\neg h]\ ;\ ([g]\ ;\ S\ ;\ ([h]\ ;\ T)^\omega\ ;\ [\neg h])^\omega\ ;\ [\neg g]\\ &= \{\text{leapfrog (Lemma 11), guard rules}\}\\ &\quad ([h]\ ;\ T)^\omega\ ;\ ([g \cap \neg h]\ ;\ S\ ;\ ([h]\ ;\ T)^\omega)^\omega\ ;\ [\neg g \cap \neg h]\\ &= \{\text{decomposition (Lemma 12)}\}\\ &\quad ([g \cap \neg h]\ ;\ S \sqcap [h]\ ;\ T)^\omega\ ;\ [\neg g \cap \neg h]\\ &= \{\text{rewrite using iteration}\}\\ &\quad \mathsf{do}\ g \cap \neg h \to S \parallel h \to T\ \mathsf{od}\end{aligned}$$

□

It is interesting to note that no restrictions on the guards (g and h) are needed in Theorem 26. Another interesting fact is that in the proof, both the leapfrog and the decomposition rules for iterations are used.

A direct consequence of Theorem 26 is the following result, which has also been proved by Manasse and Nelson [20] and by van de Snepscheut [24].

Corollary 27. *Assume that S and T are conjunctive predicate transformers and that $g \cap h = \mathsf{false}$. Then*

$$\mathsf{do}\ g \to S \parallel h \to T\ \mathsf{od}\ =\ \mathsf{do}\ h \to T\ \mathsf{od}\ ;\ \mathsf{do}\ g \to (S\ ;\ \mathsf{do}\ h \to T\ \mathsf{od})\ \mathsf{od}$$

Manasse and Nelson have a very long and complex proof and they comment that "the labor involved in the proof seems excessive". The proof method of van de Snepscheut is closer to ours, but it involves an auxiliary notion of the *lowest meet closure* of a function.

6.3 Data refinement of loops

To end this section, we show how the data refinement rules for iterations can be used to derive rules for data refinement of loops. We begin with the basic case [27]:

Theorem 28. *Assume that D is continuous and that $D ; S \sqsubseteq S' ; D$ and $D ; [g] \sqsubseteq [g'] ; D$ and $D ; [\neg g] \sqsubseteq [\neg g'] ; D$. Then*

$$D ; \mathsf{do}\ g \to S\ \mathsf{od} \sqsubseteq \mathsf{do}\ g' \to S'\ \mathsf{od} ; D$$

The proof follows the same line of argument as the proof of the corresponding rule for action systems (Theorem 20), so we omit it. The guard conditions $D ; [g] \sqsubseteq [g'] ; D$ and $D ; [\neg g] \sqsubseteq [\neg g'] ; D$ can be simplified using Lemma 1 (c) or (d). Since the two basic forms of data refinement are when D is strict and disjunctive (forward data refinement) and when D is strict, terminating, and conjunctive (backward data refinement), we have the following versions of this condition: $D.\, g \subseteq g'$ and $D.\, (\neg g) \subseteq \neg g'$ when D is strict and disjunctive, and $g' \subseteq D.\, g$ and $\neg g' \subseteq D.\, (\neg g)$ when D is strict, terminating, and conjunctive.

Intuitively the conditions can be justified as follows: $D ; [\neg g] \sqsubseteq [\neg g'] ; D$ states that the concrete loop must be able to continue whenever the abstract loop can and $D ; [g] \sqsubseteq [g'] ; D$ states that the concrete loop must be able to terminate whenever the abstract loop can. Thus, together they say that termination of the two loops must happen at corresponding points.

Exactly as for action systems we can allow stuttering actions in a data refinement.

Theorem 29. *Assume that S and T are conjunctive and D is continuous. Furthermore assume that the following conditions hold:*

(i) $D ; S \sqsubseteq S' ; D$ *and* $D ; \mathsf{skip} \sqsubseteq [h] ; T ; D$
(ii) $D ; [g] \sqsubseteq [g'] ; D$ *and* $D ; [\neg g] \sqsubseteq [\neg g' \cap \neg h] ; D$
(iii) $D.\, \mathsf{true} \subseteq \mu.\, ([h] ; T)$

Then

$$D ; \mathsf{do}\ g \to S\ \mathsf{od} \sqsubseteq \mathsf{do}\ g' \to S' \,[\!]\, h \to T\ \mathsf{od} ; D$$

Again the proof is similar to the corresponding rule for action systems (Theorem 21) so we omit it.

7 Loop transformations

We now show how the basic loop rules are used to derive transformation rules that allow loop constructs in programs to be manipulated. A collection of basic rules that are useful from a practical program transformation point of view can be derived in this way. We here consider only two such derived rules for loops: *removing vacuous loops* and *splitting and merging loops*. These exemplify the general technique for deriving interesting loop transformation rules from the basic properties of iterations.

We begin with a rule that removes a vacuous loop. This rule can be seen as corresponding to the rules for iterations in Lemma 9 (c) (although that lemma cannot be used directly in the proof).

Theorem 30 (Remove vacuous loop). *Assume that S is conjunctive. Then*

$$\mathsf{do}\ g \cap h \to S\ \mathsf{od}\ ;\ \mathsf{do}\ g \to S\ \mathsf{od} = \mathsf{do}\ g \to S\ \mathsf{od}$$

Proof. We first prove the special case when $h = \mathsf{true}$:

$$\begin{aligned}
&\mathsf{do}\ g \to S\ \mathsf{od}\ ;\ \mathsf{do}\ g \to S\ \mathsf{od} \\
&= \{\text{rewrite using iteration}\} \\
&\quad ([g]\ ;\ S)^\omega\ ;\ [\neg g]\ ;\ ([g]\ ;\ S)^\omega\ ;\ [\neg g] \\
&= \{\text{Lemma 10}\} \\
&\quad ([g]\ ;\ S)^\omega\ ;\ [\neg g]\ ;\ [\neg g] \\
&= \{\text{general guard property } [p]\ ;\ [p] = [p]\} \\
&\quad ([g]\ ;\ S)^\omega\ ;\ [\neg g] \\
&= \{\text{rewrite using iteration}\} \\
&\quad \mathsf{do}\ g \to S\ \mathsf{od}
\end{aligned}$$

Furthermore we have the following:

$$\begin{aligned}
&\mathsf{do}\ g \to S\ \mathsf{od} \\
&= \{\text{rewrite using iteration}\} \\
&\quad ([g]\ ;\ S)^\omega\ ;\ [\neg g] \\
&= \{\text{lattice properties}\} \\
&\quad ([(g \cap h) \cup (g \cap \neg h)]\ ;\ S)^\omega\ ;\ [\neg(g \cap h) \cap \neg(g \cap \neg h)] \\
&= \{\text{homomorphism and distributivity properties}\} \\
&\quad ([g \cap h]\ ;\ S \sqcap [g \cap \neg h]\ ;\ S)^\omega\ ;\ [\neg(g \cap h) \cap \neg(g \cap \neg h)] \\
&= \{\text{rewrite using iteration}\} \\
&\quad \mathsf{do}\ g \cap h \to S \parallel g \cap \neg h \to S\ \mathsf{od} \\
&= \{\text{decomposition with } T := S \text{ and } h := g \cap h\} \\
&\quad \mathsf{do}\ g \cap h \to S\ \mathsf{od}\ ;\ \mathsf{do}\ g \to (S\ ;\ \mathsf{do}\ g \cap h \to S\ \mathsf{od})\ \mathsf{od}
\end{aligned}$$

Now,

$$
\begin{aligned}
& \mathsf{do}\ g \cap h \to S\ \mathsf{od}\ ;\ \mathsf{do}\ g \to S\ \mathsf{od} \\
= & \ \{\text{second derivation}\} \\
& \mathsf{do}\ g \cap h \to S\ \mathsf{od}\ ;\ \mathsf{do}\ g \cap h \to S\ \mathsf{od}\ ;\ \mathsf{do}\ g \\
& \quad \to (S\ ;\ \mathsf{do}\ g \cap h \to S\ \mathsf{od})\ \mathsf{od} \\
= & \ \{\text{first derivation}\} \\
& \mathsf{do}\ g \cap h \to S\ \mathsf{od}\ ;\ \mathsf{do}\ g \to (S\ ;\ \mathsf{do}\ g \cap h \to S\ \mathsf{od})\ \mathsf{od} \\
= & \ \{\text{second derivation}\} \\
& \mathsf{do}\ g \to S\ \mathsf{od}
\end{aligned}
$$

□

We can now prove a more general rule for *splitting and merging loops*.

Theorem 31 (Split/Merge loops). *Assume that S and T are conjunctive predicate transformers with $T.\mathsf{true} \cap \neg g \subseteq T.(\neg g)$. Then*

$$\mathsf{do}\ g \to S \parallel \neg g \cap h \to T\ \mathsf{od} = \mathsf{do}\ g \to S\ \mathsf{od}\ ;\ \mathsf{do}\ h \to T\ \mathsf{od}$$

Proof. We first note that by Lemma 1 (c), the assumption $T.\mathsf{true} \cap \neg g \subseteq T.(\neg g)$ can be restated as $T\ ;\ [\neg g] \sqsubseteq [\neg g]\ ;\ T$ Now we have

$$
\begin{aligned}
& \mathsf{do}\ g \to S \parallel \neg g \cap h \to T\ \mathsf{od} \\
= & \ \{\text{decomposition (Theorem 26)}\} \\
& \mathsf{do}\ g \to S\ \mathsf{od}\ ;\ \mathsf{do}\ h \to (T\ ;\ \mathsf{do}\ g \to S\ \mathsf{od})\ \mathsf{od} \\
= & \ \{\text{rewrite using iteration}\} \\
& ([g]\ ;\ S)^\omega\ ;\ [\neg g]\ ;\ ([h]\ ;\ T\ ;\ ([g]\ ;\ S)^\omega\ ;\ [\neg g])^\omega\ ;\ [\neg h] \\
\sqsubseteq & \ \{\text{general property } \mathsf{skip} \sqsubseteq [p]\} \\
& ([g]\ ;\ S)^\omega\ ;\ [\neg g]\ ;\ ([h]\ ;\ T\ ;\ [\neg g]\ ;\ ([g]\ ;\ S)^\omega\ ;\ [\neg g])^\omega\ ;\ [\neg h] \\
= & \ \{\text{Lemma 10, general rule } [p]\ ;\ [p] = [p]\} \\
& ([g]\ ;\ S)^\omega\ ;\ [\neg g]\ ;\ ([h]\ ;\ T\ ;\ [\neg g])^\omega\ ;\ [\neg h] \\
\sqsubseteq & \ \{\text{see separate derivation below}\} \\
& ([g]\ ;\ S)^\omega\ ;\ [\neg g]\ ;\ ([h]\ ;\ T)^\omega\ ;\ [\neg h] \\
= & \ \{\text{rewrite using iteration}\} \\
& \mathsf{do}\ g \to S\ \mathsf{od}\ ;\ \mathsf{do}\ h \to T\ \mathsf{od}
\end{aligned}
$$

Here the fifth step is justified by the following derivation:

$$
\begin{aligned}
& [\neg g]\ ;\ ([h]\ ;\ T\ ;\ [\neg g])^\omega \sqsubseteq [\neg g]\ ;\ ([h]\ ;\ T)^\omega \\
\equiv & \ \{\text{general rule } [p]\ ;\ [p] = [p]\}
\end{aligned}
$$

$$
\begin{array}{ll}
& [\neg g] \,;\, ([h] \,;\, T \,;\, [\neg g])^\omega \sqsubseteq [\neg g] \,;\, [\neg g] \,;\, ([h] \,;\, T)^\omega \\
\Leftarrow & \{\text{monotonicity}\} \\
& ([h] \,;\, T \,;\, [\neg g])^\omega \sqsubseteq [\neg g] \,;\, ([h] \,;\, T)^\omega \\
\Leftarrow & \{\text{induction}\} \\
& [h] \,;\, T \,;\, [\neg g] \,;\, [\neg g] \,;\, ([h] \,;\, T)^\omega \sqcap \mathsf{skip} \sqsubseteq [\neg g] \,;\, ([h] \,;\, T)^\omega \\
\Leftarrow & \{\text{general rule } [p] \,;\, [p] = [p], \text{ assumption } T \,;\, [\neg g]; \sqsubseteq [\neg g] \,;\, T\} \\
& [h] \,;\, [\neg g] \,;\, T \,;\, ([h] \,;\, T)^\omega \sqcap \mathsf{skip} \sqsubseteq [\neg g] \,;\, ([h] \,;\, T)^\omega \\
\equiv & \{\text{general rules } [p] \,;\, [q] = [q] \,;\, [p] \text{ and } \mathsf{skip} \sqsubseteq [p]\} \\
& [\neg g] \,;\, [h] \,;\, T \,;\, ([h] \,;\, T)^\omega \sqcap [\neg g] \sqsubseteq [\neg g] \,;\, ([h] \,;\, T)^\omega \\
\equiv & \{\text{distributivity}\} \\
& [\neg g] \,;\, ([h] \,;\, T \,;\, ([h] \,;\, T)^\omega \sqcap \mathsf{skip}) \sqsubseteq [\neg g] \,;\, ([h] \,;\, T)^\omega \\
\equiv & \{\text{unfolding}\} \\
& \mathsf{T}
\end{array}
$$

For the refinement in the opposite direction we have

$$
\begin{array}{ll}
& \mathsf{do}\ g \to S \parallel \neg g \cap h \to T\ \mathsf{od} \\
= & \{\text{decomposition (Theorem 26)}\} \\
& \mathsf{do}\ g \to S\ \mathsf{od} \,;\, \mathsf{do}\ h \to (T \,;\, \mathsf{do}\ g \to S\ \mathsf{od})\ \mathsf{od} \\
= & \{\text{rewrite using iteration, guard property}\} \\
& ([g] \,;\, S)^\omega \,;\, [\neg g] \,;\, [\neg g] \,;\, ([h] \,;\, T \,;\, ([g] \,;\, S)^\omega \,;\, [\neg g])^\omega \,;\, [\neg h] \\
= & \{\text{leapfrog (Lemma 11)}\} \\
& ([g] \,;\, S)^\omega \,;\, [\neg g] \,;\, ([\neg g] \,;\, [h] \,;\, T \,;\, ([g] \,;\, S)^\omega)^\omega \,;\, [\neg g] \,;\, [\neg h] \\
= & \{\text{general rule } [p] \,;\, [q] = [q] \,;\, [p]\} \\
& ([g] \,;\, S)^\omega \,;\, [\neg g] \,;\, ([h] \,;\, [\neg g] \,;\, T \,;\, ([g] \,;\, S)^\omega)^\omega \,;\, [\neg g] \,;\, [\neg h] \\
\sqsupseteq & \{\text{Lemma 1 (c), assumption } T \,;\, [\neg g]; \sqsubseteq [\neg g] \,;\, T\} \\
& ([g] \,;\, S)^\omega \,;\, [\neg g] \,;\, ([h] \,;\, T \,;\, [\neg g] \,;\, ([g] \,;\, S)^\omega)^\omega \,;\, [\neg g] \,;\, [\neg h] \\
= & \{\text{Lemma 10}\} \\
& ([g] \,;\, S)^\omega \,;\, [\neg g] \,;\, ([h] \,;\, T \,;\, [\neg g])^\omega \,;\, [\neg g] \,;\, [\neg h] \\
= & \{\text{see mutual refinement proof below}\} \\
& ([g] \,;\, S)^\omega \,;\, [\neg g] \,;\, ([h] \,;\, T)^\omega \,;\, [\neg h] \\
= & \{\text{rewrite using iteration}\} \\
& \mathsf{do}\ g \to S\ \mathsf{od} \,;\, \mathsf{do}\ h \to T\ \mathsf{od}
\end{array}
$$

where the mutual refinement proof mentioned in the middle consists of

$$[\neg g] \,;\, ([h] \,;\, T)^\omega \ \sqsubseteq\ [\neg g] \,;\, ([h] \,;\, T \,;\, [\neg g])^\omega \,;\, [\neg g]$$

$$\equiv \{\text{general rule } \mathsf{skip} \sqsubseteq [q]\}$$
$$\mathsf{T}$$

and

$$[\neg g] ; ([h] ; T ; [\neg g])^\omega ; [\neg g] \;\sqsubseteq\; [\neg g] ; ([h] ; T)^\omega$$
$$\equiv \{\text{leapfrog (Lemma 11), general rule } [q] ; [q] = [q]\}$$
$$([\neg g] ; [h] ; T)^\omega ; [\neg g] \;\sqsubseteq\; [\neg g] ; ([h] ; T)^\omega$$
$$\Leftarrow \{\text{induction}\}$$
$$[\neg g] ; [h] ; T ; [\neg g] ; ([h] ; T)^\omega \sqcap [\neg g] \;\sqsubseteq\; [\neg g] ; ([h] ; T)^\omega$$
$$\equiv \{\text{unfold, distributivity } ([q] \text{ is conjunctive})\}$$
$$[\neg g] ; [h] ; T ; [\neg g] ; ([h] ; T)^\omega \sqcap [\neg g]$$
$$\sqsubseteq\; [\neg g] ; [h] ; T ; ([h] ; T)^\omega \sqcap [\neg g]$$
$$\Leftarrow \{\text{componentwise refinement}\}$$
$$[\neg g] ; [h] ; T ; [\neg g] \;\sqsubseteq\; [\neg g] ; [h] ; T$$
$$\equiv \{\text{general rule } [p] ; [q] = [q] ; [p]\}$$
$$[h] ; [\neg g] ; T ; [\neg g] \;\sqsubseteq\; [h] ; [\neg g] ; T$$

$$\equiv \{\text{assumption } T ; [\neg g]; \sqsubseteq [\neg g] ; T, \text{ general rule } [q] ; [q] = [q]\}$$
$$\mathsf{T}$$

□

Manasse and Nelson [20] prove a special case of Theorem 31, stating that

$$\mathsf{do}\ g \to S \,[\!]\, h \to T\ \mathsf{od} = \mathsf{do}\ g \to S\ \mathsf{od}\ ;\ \mathsf{do}\ h \to T\ \mathsf{od}$$

if $g \cap h = \mathsf{false}$ and $\mathsf{T}.\,\mathsf{true} \subseteq T.\,(\neg g)$. The intuition of the second assumption is that if T terminates, then it establishes $\neg g$ (Manasse and Nelson formulate this assumption in terms of weakest liberal preconditions, but the intuition is the same). We can give a very short proof of this. First we note that $\mathsf{T}.\,\mathsf{true} \subseteq T.\,(\neg g)$ can be restated as $T = T ; [\neg g]$ (using Lemma 1 (c)). Then

$$\mathsf{do}\ g \to S \,[\!]\, h \to T\ \mathsf{od}$$
$$= \{\text{decomposition (Lemma 12), noting } h = \neg g \cap h\}$$
$$\mathsf{do}\ g \to S\ \mathsf{od}\ ;\ \mathsf{do}\ h \to (T\ ;\ \mathsf{do}\ g \to S\ \mathsf{od})\ \mathsf{od}$$
$$= \{\text{rewrite using iteration}\}$$
$$([g] ; S)^\omega ; [\neg g] ; ([h] ; T ; ([g] ; S)^\omega ; [\neg g])^\omega ; [\neg h]$$
$$= \{\text{assumption}\}$$
$$([g] ; S)^\omega ; [\neg g] ; ([h] ; T ; [\neg g] ; ([g] ; S)^\omega ; [\neg g])^\omega ; [\neg h]$$

$$
\begin{array}{l}
= \{\text{Lemma 10, general rule } [p]\,;\,[p] = [p]\} \\
\quad ([g]\,;\,S)^\omega\,;\,[\neg g]\,;\,([h]\,;\,T\,;\,[\neg g])^\omega\,;\,[\neg h] \\
= \{\text{assumption}\} \\
\quad ([g]\,;\,S)^\omega\,;\,[\neg g]\,;\,([h]\,;\,T)^\omega\,;\,[\neg h] \\
= \{\text{rewrite using iteration}\} \\
\quad \mathsf{do}\ g \to S\ \mathsf{od}\,;\,\mathsf{do}\ h \to T\ \mathsf{od}
\end{array}
$$

From the rules in Theorems 30 and 31 it is possible to derive further, more specialised rules. An example of such a rule is the following explicit merge of two loops using flag variables:

$$
\begin{array}{l}
\mathsf{do}\ g \to S\ \mathsf{od}\,;\,\mathsf{do}\ h \to T\ \mathsf{od} \\
\sqsubseteq \\
\mathsf{begin\ var}\ f := \mathsf{T}\,; \\
\quad \mathsf{do}\ f \wedge g \to S \\
\quad [\!]\ f \wedge \neg g \to f := \mathsf{F} \\
\quad [\!]\ \neg f \wedge h \to T \\
\quad \mathsf{od} \\
\mathsf{end}
\end{array}
$$

8 Conclusion

We have shown how one can reason about iterations and loops in a purely algebraic setting, based on just the lattice theoretic properties of loops, as formalised in the refinement calculus. We have described the strong and weak iteration operators and their basic properties in more detail elsewhere [9]. Here we apply the iteration operators to loops and in particular to the derivation of advanced loop transformation rules. The rules that we have derived are central ones, which can be used as stepping stones for more detailed and specific transformation rules. We draw our inspiration from regular expressions and the way they are formalised in regular algebras. This has directed our attention to ways of deriving analogous results for iteration statements and traditional (guarded) loop constructs. The contribution of the paper is best visible in the proofs of the rules for data refinement with stuttering (Theorem 29) and the atomicity refinement theorem (Theorem 22). Although neither theorem is new as such, we have proved them in a purely algebraic style and with weaker assumptions than in previous proofs.

The idea of iteration operators is old, and goes back to applications of regular algebra to program transformation [10, 16, 23]. Back introduced a version of weak and strong iteration using action sequences [5]. Weak iteration (called it...ti) was also used in a predicate transformer setting by

Butler and Morgan [12] while van de Snepscheut has used strong iteration [24]. Least and greatest fixpoint constructs similar to the iteration operators described here have also been investigated thoroughly by the Eindhoven Mathematics of Program Construction group [1]. They make heavy use of Galois Connections, and do not apply results to refinement or to guarded loops.

The transformation rules for loops derived in this paper are not new in themselves. However, they have generally been justified only informally or given operationally oriented proofs (an exception is the paper by van de Snepscheut where the leapfrog rule for loops is proved [24]). The rule for data refinement with stuttering is the basis for the *superposition* method of refinement [8]. It was proved by von Wright [26] but under unnecessarily strong assumptions and using a number of ad hoc lemmas. The roots of the atomicity refinement rule go back to Lipton [19] and Lamport [18] and its practical use in program refinement is demonstrated by Back and Sere [6, 7]. Back has given a purely operational proof of the theorem [4] and one in a more algebraic style, but still in terms of execution sequences and with unnecessarily strong assumptions (in the form of extra nonenabling and nondisabling assumptions) [5]. Here the combination of a purely algebraic approach and a structured calculational proof style combine to give proofs that are elegant and easy to check and that show clearly at what points assumptions are needed for the proof to go through.

Acknowledgements. We wish to thank the anonymous referees for the detailed comments, which led to a number of improvements to the original version of this paper.

References

1. C. Aarts et al.: Fixpoint calculus. Information Processing Letters 53(3), February 1995
2. R.J. Back: Correctness Preserving Program Refinements: Proof Theory and Applications, volume 131 of Mathematical Centre Tracts. Mathematical Centre, Amsterdam, 1980
3. R.J. Back: A calculus of refinements for program derivations. Acta Informatica 25, 593–624 (1988)
4. R.J. Back: Refining atomicity in parallel algorithms. In PARLE Conference on Parallel Architectures and Languages Europe, Eindhoven, the Netherlands. Berlin Heidelberg New York: Springer 1989
5. R.J. Back: Atomicity refinement in a refinement calculus framework. Reports on computer science and mathematics 141, Åbo Akademi, 1992
6. R.J. Back, K. Sere: Stepwise refinement of parallel algorithms. Science of Computer Programming 13, 133–180 (1990)
7. R.J. Back, K. Sere: Stepwise refinement of action systems. Structured Programming 12, 17–30 (1991)
8. R.J. Back, K. Sere: Superposition refinement of parallel algorithms. In K.R. Parker, G.A. Rose, editors, Formal Description Techniques IV, pages 475–493. North-Holland: Elsevier Science Publishers 1992

9. R.J. Back, J. von Wright: Refinement Calculus: A Systematic Introduction. Berlin Heidelberg New York: Springer 1998
10. R.C. Backhouse, B.A. Carré: Regular algebra applied to path finding problems. Journal Inst. Math. Appl. 15, 161–186 (1975)
11. G. Birkhoff: Lattice Theory. American Mathematical Society, Providence, 1961
12. M.J. Butler, C.C. Morgan: Action systems, unbounded nondeterminism and infinite traces. Formal Aspects of Computing 7(1), 37–53 (1995)
13. K.M. Chandy, J. Misra: Parallel Program Design: A Foundation. Addison–Wesley 1988
14. E.W. Dijkstra: A Discipline of Programming. Prentice-Hall 1976
15. E.W. Dijkstra, C.S. Scholten: Predicate Calculus and Program Semantics. Berlin Heidelberg New York: Springer 1990
16. R.M. Dijkstra: Relational calculus and relational program semantics. Tech. Rpt. 9408, Rijksuniversiteit Groningen, 1994
17. P.H. Gardiner, C.C. Morgan: Data refinement of predicate transformers. Theoretical Computer Science 87(1), 143–162 (1991)
18. L. Lamport: A theorem on atomicity in destributed algorithms. Distributed Computing 4, 59–68 (1990)
19. R.J. Lipton: Reduction: A method of proving properties of parallel programs. Communications of the ACM 18(12), 717–721 (1975)
20. M.S. Manasse, C.G. Nelson: Correct compilation of control structures. Techn. Memo. 11271-840909-09TM, AT&T Bell Laboratories, September 1984
21. C.C. Morgan: The specification statement. ACM Transactions on Programming Languages and Systems 10(3), 403–419 (1988)
22. J.M. Morris: A theoretical basis for stepwise refinement and the programming calculus. Science of Computer Programming 9, 287–306 (1987)
23. S. Rönn: On the Regularity Calculus and its Role in Distributed Programming. PhD thesis, Helsinki University of technology, Helsinki, Finland, 1992
24. J.L.A. van de Snepscheut: On lattice theory and program semantics. Technical Report CS-TR-93-19, Caltech, Pasadena, California, USA, 1993
25. A. Tarski: A lattice theoretical fixed point theorem and its applications. Pacific J. Mathematics 5, 285–309 (1955)
26. J. von Wright: Data refinement with stuttering. Reports on computer science and mathematics 137, Åbo Akademi, 1992
27. J. von Wright: The lattice of data refinement. Acta Informatica 31, 105–135 (1994)