Computation calculus bridging a formalization gap

Dijkstra, Rutger M

Published in:
Science of computer programming

DOI:
10.1016/S0167-6423(99)00021-0

Publication date:
2000

ELSEVIER

Science of Computer Programming 37 (2000) 3–36

Science of Computer Programming

www.elsevier.nl/locate/scico

Computation calculus bridging a formalization gap

Rutger M. Dijkstra

Rijksuniversiteit Groningen, P.O. Box 800, 9700 AV Groningen, Netherlands

Abstract

We present an algebra that is intended to bridge the gap between programming formalisms that have a high level of abstraction and the operational interpretations these formalisms have been designed to capture. In order to *prove* a high-level formalism sound for its intended operational interpretation, one needs a mathematical handle on the latter. To this end we design the computation calculus. As an expression mechanism, it is sufficiently transparent to avoid begging the question. As an algebra, it is quite powerful and relatively simple.

0. Introduction

For reasoning about (imperative style) programs, lots of extensions of predicate calculus or logic are in circulation: Hoare-logic, *wp*-calculus, temporal logics of various kinds, UNITY-logic, etc. All these extensions enrich the language of predicate calculus with 'primitives' intended to capture some operational aspects of programs, e.g. in *wp*-calculus:

wp.s.q: holds in those initial states for which every execution of program *s* terminates in a state satisfying the predicate *q*,

in linear time temporal logic:

G*s*: holds for those computations (i.e infinite sequences of states) for which every suffix satisfies *s*,

and for a UNITY program:

wlt.q: holds in those (initial) states for which every computation of the program contains at least one state satisfying the predicate *q*.

The verbal right-hand sides above are 'intended operational interpretation' without any formal status: they serve only to translate operational intentions into the formalism. In order to actually prove properties of programs, the formalisms supply rules in the form of postulates, definitions, axioms, inference rules, and so forth. Here is a random sample from each of the aforementioned formalisms:

(a) $wp.(S;T).q = wp.S.(wp.T.q)$,

0167-6423/00/$ - see front matter
PII: S0167-6423(99)00021-0

(b) $\vdash \mathbf{G}(s \Rightarrow \mathbf{X}s) \Rightarrow (s \Rightarrow \mathbf{G}s)$,
(c) $wlt.q = \langle \boldsymbol{\mu} x :: \langle \exists S :: \langle \boldsymbol{\nu} y :: q \vee (\langle \forall T :: wp.T.y \rangle \wedge wp.S.x) \rangle \rangle \rangle$.

This is where the problems start. Although for many of the rules it is quite clear that they are indeed adequate for the intended operational interpretation – see (a) – this is by no means always the case – see (c). This distance between intended interpretation and mathematical formalization is what we call "the formalization gap". If large, the formalization gap can become a serious problem, since it means that the math we write down does not necessarily mean what we think it does.

In this paper we develop an algebra, called "the computation calculus", that is intended to bridge this formalization gap. The idea is that intended operational interpretations can be expressed succinctly in the computation calculus and characterizations on a higher level of abstraction can subsequently be derived from that, e.g. for the predicate transformer *wlt*, we would ideally like to be able to derive the fixpoint expression in (c) from some 'direct' formalization of the intended operational interpretation.

We abstract the computation calculus from a simple model of computations. Thus, we start with a model wherein we *define* some operators satisfying certain properties. Subsequently, we forget the definitions and *postulate* the relevant properties. This ensures that all results in our algebra are at least applicable to the model we abstract it from.

Although the computation calculus is potentially more general than the instance we abstract it from, greater generality is not the purpose of the exercise. The reason for the abstraction is twofold. Firstly, life is simpler at the higher level of abstraction, the landscape less cluttered with detail. Therefore, there is less chance of getting lost or tripping up. Secondly, since results in the algebra are derived from the postulates only, their validity in the model relies only on the postulates being satisfied therein. Thus, the abstraction provides a clear view on the properties we actually use and, hence, the assumptions we rely on.

The procedure we follow is incremental for both the model and the algebra. We begin with a model that is far too general and we extract from this some very elementary insights that we capture in our postulates. As we become more ambitious in the things we want to prove, we may find that our algebra is still not strong enough. We then add what is missing to the algebra and impose, if necessary, the appropriate restrictions on the model.

After the initial exploration of the basics, the first slightly more ambitious target we aim for is the pre-/postcondition semantics of simple sequential programs. The biggest hurdle to overcome here is the treatment of (tail-) recursion. Subsequently, we prepare the ground for higher targets by enriching our algebra with some insights concerning atomic computations. This leaves us with an algebra that is very powerful indeed: subsuming various temporal logics. Finally, we demonstrate the algebra in action by applying all of it to a non-trivial problem: the derivation the fixpoint characterization of *wlt* – see (c) – from its intended operational interpretation.

We build on the predicate calculus of Dijkstra and Scholten [3] augmented with whatever facts from lattice theory we have use for, in particular the fixpoint calculus

as explored by Backhouse et al. [13]. Some highlights of this calculus are listed in the appendix.

Readers familiar with relation algebra in any form, the sequential calculus of von Karger and Hoare [9], or any of various temporal logics may recognize where we got our inspiration from. However, unfamiliarity with these fields should not stand in the way of understanding our algebra.

1. Basics of the model

Our model is in essence taken from Johan Lukkien's Ph.D. Thesis [11].

A "program (fragment)" or "statement" operates on a set of *states*. Starting a statement in some initial state results in a sequence of (atomic) steps each of which ends in a new state. We call the sequence of states traversed during the execution the *computation* generated by the statement. Computations can be finite or infinite but not empty (0 steps means that the initial state is also the final state, i.e. generates a singleton computation). So we have:

$\mathscr{S}$: the state space; a nonempty set, the elements of which we call 'states'.

$\mathscr{C}$: the computation space; a set of nonempty, finite or infinite sequences of states that we call 'computations'.

A specification is a predicate on computations, i.e. a function of the type $\mathscr{C} \to \mathbb{B}$. We also consider programs to be specifications: they specify exactly those computations their execution may result in. Thus, both specifications and statements are elements of the set *CPred*, defined as:

Definition 0. $CPred = \mathscr{C} \to \mathbb{B}$.

This provides a standard model for the predicate calculus as developed in [3] where all boolean operators are lifted: $(s \Rightarrow t).\tau \equiv (s.\tau \Rightarrow t.\tau)$, $\langle \forall i :: s.i \rangle.\tau \equiv \langle \forall i :: s.i.\tau \rangle$, etc., and 'everywhere' quantifies over all computations: $[s] \equiv \langle \forall \tau :: s.\tau \rangle$.

We extend the predicate algebra with sequential composition. Using $\#, \uparrow$, and $\downarrow$ to denote the list operations 'length', 'take', and 'drop' respectively, we define:

Definition 1. For all predicates s and t and computations τ

$$(s;t).\tau \equiv (\#\tau = \infty \wedge s.\tau) \vee \langle \exists n : n < \#\tau : s.(\tau \uparrow n+1) \wedge t.(\tau \downarrow n) \rangle.$$

Note the 'one point overlap' between $\tau \uparrow n+1$ and $\tau \downarrow n$, which reflects that the final state of the s-subcomputation is the initial state of the t-subcomputation.

In order for the right-hand side of (1) to be well typed, the sequences $\tau \uparrow n+1$ and $\tau \downarrow n$ must be computations for all n in the range. We therefore impose on $\mathscr{C}$ the restriction that it satisfies:

Requirement 2. Nonempty segments of computations are computations.

Sequential composition is universally disjunctive in its first argument, positively disjunctive in its second argument, and associative. Moreover, it has the (left- and right-) neutral element $\mathbb{1}$ given by:

Definition 3. $\mathbb{1}.\tau \equiv \#\tau = 1$ for all τ.

Not distinguishing between a singleton computation and the single state it consists of, the singleton computations are a subset of $\mathscr{S}$. We assume that $\mathscr{S}$ does not contain states that do not partake in any computation:

Requirement 4. "no junk" : $\mathscr{S} \subseteq \mathscr{C}$.

This means that $\mathbb{1}$ is the characteristic predicate of our state space and that we can identify predicates on the state space with predicates that hold only for singleton 'computations'.

2. Basics of the algebra

The computation calculus is an algebra on objects of the type *CPred* and it is an extension of the predicate calculus, i.e:

Postulate 5. *CPred* is a predicate algebra.

Postulate 5 means that we import all the postulates for the predicate calculus from [3]. These postulates characterize exactly a complete boolean lattice with the order $[\_ \Rightarrow \_]$ and the boolean operators denoting the various lattice operations.

The first extensions we introduce are a further binary operator and an extra constant:

$$; : CPred^2 \to CPred \text{ and } \mathbb{1} : CPred.$$

Composition $(;)$ binds more strongly than the binary boolean operators but – as all binary operators – more weakly than unary operators.

The additional ingredients satisfy:

Postulate 6.

(i) ; is universally disjunctive in its first and positively disjunctive in its second argument.

(ii) ; is associative and $\mathbb{1}$ is its neutral element.

These postulates amount to little more than that our calculus is what Roland Backhouse calls a "semi-regular (boolean) algebra". This is a very general structure and so there would appear to be little of consequence to be deducible from these postulates alone. For the moment, however, they are all we have got. Some standard consequences

that we do not care to list as separate results are that composition is monotonic in both arguments and has *false* as a left-zero element.

Life becomes more interesting with the introduction of some vocabulary:

Definition 7. p is a state predicate $\equiv [p \Rightarrow \mathbb{1}]$ for all p.

As noted in the previous section, these correspond to predicates on our state space. For state predicates we record the highly useful.

Fact 8 (State-restriction rules)**.** (i) $p\,;s = p\,;\text{true} \wedge s$, *and*
(ii) $s\,;p = s \wedge \text{true}\,;p$ *for all* s, *and all state predicates* p.

Proof. We show only the first. On account of monotonicity it suffices to prove '$\Leftarrow$' and this follows after shunting from the observation that

$$
\begin{array}{ll}
 & p\,;s \vee \neg s \\
\Leftarrow & \quad \{\ p \text{ is a state predicate}\ \} \\
 & p\,;s \vee p\,;\neg s \\
= & \quad \{\ ;\text{over}\ \vee,\ \text{excluded middle}\ \} \\
 & p\,;\mathit{true}. \qquad \square
\end{array}
$$

The state-restriction rules express that by pre-/postfixing a state predicate p to a predicate s, we obtain a characterization of those s computations for which the initial/final state satisfies p. We proceed by exploring the consequences of requiring the final state to satisfy *false*, i.e. be nonexistent. Instantiating state restriction rule (Fact 8(ii)) with $p := \mathit{false}$ yields

$$s\,;\mathit{false} = s \wedge \mathit{true}\,;\mathit{false}.$$

Interpreting $\mathit{true}\,;\mathit{false}$ in the model, we find that this predicate holds exactly for the infinite ("eternal") computations. We introduce names for this constant and for its negation (which of course characterizes the finite computations) and we introduce some more vocabulary.

Definition 9.

$$\mathtt{E}, \mathtt{F} : \mathit{CPred},\ \mathtt{E} = \mathit{true}\,;\mathit{false} \text{ and } \mathtt{F} = \neg\mathtt{E}.$$

$$s \text{ is eternal} \equiv [s \Rightarrow \mathtt{E}] \text{ and } s \text{ is finite} \equiv [s \Rightarrow \mathtt{F}] \quad \text{for all } s.$$

Thus, the preceding observation now reads:

Fact 10. $s\,;\mathit{false} = s \wedge \mathtt{E}$ *for all* s.

We now explore to what extent our modest postulates allow us to confirm our expectations concerning finite and infinite behaviour. The first things to note are that, on account of Fact 10 and elementary predicate calculus, we have for all s

(a) $[s \Rightarrow \mathtt{E}] \equiv s\,;\, \mathit{false} = s$ and (b) $[s \Rightarrow \mathtt{F}] \equiv s\,;\, \mathit{false} = \mathit{false}$.

Composition being universally disjunctive in its first argument, *false* is left-zero thereof. Consequently, it follows from (a) that the eternal predicates are precisely the left-zeroes of composition:

Fact 11 (Preemption theorem)**.**

$$[s \Rightarrow \mathtt{E}] \equiv \langle \forall t :: s\,;\, t = s \rangle \quad \textit{for all } s.$$

Since composition is positively disjunctive in its second argument and $s\,;\, \mathit{false} = \mathit{false}$ means that $s\,;\_$ distributes over existential quantifications with an empty range as well, we get from (b) the useful:

Fact 12. $[s \Rightarrow \mathtt{F}] \equiv s\,;\_$ *is universally disjunctive for all* s.

We conclude this section with a couple of exercises. In spite of the heading "Etude" we freely use the following facts as theorems.

Etude 13. Leaving universal quantification over the free variables understood:
(i) $[\mathbb{1} \Rightarrow \mathtt{F}],\ \mathtt{F}\,;\mathtt{F} = \mathtt{F}$,
(ii) $s\,;t = (s \wedge \mathtt{E}) \vee (s \wedge \mathtt{F})\,;t$,
(iii) $s\,;t \wedge \mathtt{F} = (s \wedge \mathtt{F})\,;(t \wedge \mathtt{F})$, and
(iv) $s\,;t \wedge \mathtt{E} = (s \wedge \mathtt{E}) \vee (s \wedge \mathtt{F})\,;(t \wedge \mathtt{E})$.

3. Hoare-triples and 'Leads-to'

Let s be some statement; let p and q be state predicates. The Hoare-triple $\{p\}s\{q\}$ asserts that every s-computation for which the initial state satisfies p terminates in a final state satisfying q. The s-computations with an initial state satisfying p are given by $p\,;s$ and the computations that terminate in a state satisfying q are given by $\mathtt{F}\,;q$. Thus, the Hoare-triple can be translated into our algebra as:

(a) $[p\,;s \Rightarrow \mathtt{F}\,;q]$.

Now, let u be some UNITY program. In UNITY – where the program under consideration is left implicit – the assertion $p \mapsto q$ expresses that every u-computation for which the initial state satisfies p has some state satisfying q. Since the computations that have a state satisfying q are those that satisfy $\mathtt{F}\,;q\,;\mathit{true}$, this can be rendered in our algebra as

(b) $[p\,;u \Rightarrow \mathtt{F}\,;q\,;\mathit{true}]$.

Considered as equations in the unknown p of the type 'state predicate', the weakest solution of (a) is $wp.s.q$ and the weakest solution of (b) is $wlt.q$. So we would like to get some grip on these weakest solutions.

4. State predicates and computation quantifiers

We denote some special attention to the universe of state predicates. The first thing to do is to name this universe:

Definition 14. $p \in SPred \equiv [p \Rightarrow \mathbb{1}]$ for all predicates p.

One immediate consequence of this definition is that $(SPred, [\_ \Rightarrow \_])$ is also a complete boolean lattice. From the encompassing lattice, it inherits all the lattice operations that it is closed under, i.e. existential quantification, disjunction, and conjunction. The other lattice operations on $SPred$ – we use only negation and universal quantification – are easily defined in terms of the corresponding ones on $CPred$:

Definition 15. $\sim : SPred \to SPred$ and $\overset{s}{\forall} : \mathscr{P}.SPred \to SPred$,

$$\sim p = \neg p \wedge \mathbb{1} \text{ and } \langle \overset{s}{\forall} i :: p.i \rangle = \langle \forall i :: p.i \rangle \wedge \mathbb{1}.$$

With these operators (and $\mathbb{1}$ playing the rôle of *true*) $SPred$ is yet another predicate algebra.

For a state predicate p, the computations with an initial state satisfying p are characterized by $p\,;true$. For the restriction of $\_\,;true$ to state predicates we introduce special notation, the 'initially operator':

Definition 16. $\cdot : SPred \to CPred$, $\cdot p = p\,;true$.

'Initially' enjoys some properties not enjoyed by $\_\,;true$ in general. In particular:

Fact 17. $\cdot \sim p = \neg \cdot p$ *for all* p.

Proof. By ping-pong argument.

Ping: $[\cdot \sim p \Rightarrow \neg \cdot p]$
$= \quad$ { shunt, definition (16) }
$[p\,;true \wedge \sim p\,;true \Rightarrow false]$
$= \quad$ { state restriction (8) (thrice) }
$[(p \wedge \sim p)\,;true \Rightarrow false]$
$= \quad$ { false is left-zero }
$true.$

Pong: $[\neg \cdot p \Rightarrow \cdot \sim p]$
$= \quad$ { shunt, definition (16) }
$[true \Rightarrow p\,;true \vee \sim p\,;true]$
$= \quad$ { ;over $\vee$ }
$[true \Rightarrow (p \vee \sim p)\,;true]$
$= \quad$ { neutrality of $\mathbb{1}$ }
$true.$ □

Combining the fact that the initially operator inherits universal disjunctivity from $\_\,;true$ with Fact 17, we now find that it is universally conjective as well. Consequently,

the initially operator has both an upper adjoint and a lower adjoint. We introduce operators denoting these adjoints:

Definition 18. $\mathscr{A}, \mathscr{E} : CPred \to SPred$,

$[\cdot p \Rightarrow s] \equiv [p \Rightarrow \mathscr{A} s]$ and

$[\mathscr{E} s \Rightarrow p] \equiv [s \Rightarrow \cdot p]$ for all s and all state predicates p.

These operators are, in essence, the "trace quantifiers" from the branching-time temporal logic CTL* [7]. Their interpretation is elementary:

- $\mathscr{A}$'s holds in those initial states for which every computation satisfies s and
- $\mathscr{E}$'s holds in those initial states for which at least one computation satisfies s.

We call these operators the universal and existential computation quantifier, respectively.

Using the universal computation quantifier, we can now lay our hands on the weakest solution of the Hoare-triple equation (a) from the previous section:

$$
\begin{array}{ll}
 & [p\,;s \Rightarrow \mathtt{F}\,;q] \\
= & \quad \{ \text{ state restriction } \} \\
 & [p\,;\mathit{true} \wedge s \Rightarrow \mathtt{F}\,;q] \\
= & \quad \{ \text{ Definition 16 and shunting } \} \\
 & [\cdot p \Rightarrow (s \Rightarrow \mathtt{F}\,;q)] \\
= & \quad \{ \text{ Definition 18 } \} \\
 & [p \Rightarrow \mathscr{A}(s \Rightarrow \mathtt{F}\,;q)].
\end{array}
$$

Since $\mathscr{A}(s \Rightarrow \mathtt{F}\,;q)$ is a state predicate, it is the weakest solution for p. As a succinct description of the intended operational interpretation of $wp.s.q$, we consider this a highly satisfactory solution: $\mathscr{A}(s \Rightarrow \mathtt{F}\,;q)$ expresses quite clearly that "every s-computation terminates in a state satisfying q", which is indeed what $wp.s.q$ is supposed to mean.

The computation quantifiers are not only in name and notation suggestive of quantifiers, but also in their algebraic properties:

Etude 19. Leaving universal quantification over the free variables understood:

(i) "de Morgan": $\sim \mathscr{E} s = \mathscr{A} \neg s$.
(ii) $\mathscr{A} \langle \forall i :: s_i \rangle = \langle \forall i :: \mathscr{A} s_i \rangle$.
(iii) $\mathscr{E} \langle \exists i :: s_i \rangle = \langle \exists i :: \mathscr{E} s_i \rangle$.
(iv) $\mathscr{A} \mathit{false} = \mathit{false}$ and $\mathscr{E} \mathit{true} = \mathbb{1}$.
(v) $\mathscr{A} \cdot p = p = \mathscr{E} \cdot p$.
(vi) $p \wedge \mathscr{A} s = \mathscr{A}(\cdot p \vee s)$ and $p \wedge \mathscr{E} s = \mathscr{E}(\cdot p \wedge s)$.
(vii) $p \wedge \mathscr{A} s = \mathscr{A}(\cdot p \wedge s)$ and $p \vee \mathscr{E} s = \mathscr{E}(\cdot p \vee s)$.

For the restriction of $true\,;\_$ to *SPred*, on might expect properties that are similar to those of the initially operator. However, since composition is only positively disjunctive in its second argument, "finally" is slightly more complicated than "initially". Adapting (the mirror image of) the proof of (17) to cater for these complications, we obtain:

Fact 20. *$true\,;\sim p = \neg(\mathtt{F}\,;p)$ for all state predicates p.*

Thus, as functions of the type $SPred \to CPred$, the prefixes $true\,;\_$ and $\mathtt{F}\,;\_$ are each others dual. It follows that the one is as conjuctive as the other is disjunctive. Now, $true\,;\_$ is positively disjunctive while $\mathtt{F}\,;\_$ is, on account of $\mathtt{F}$ being finite, universally disjunctive; hence

Fact 21. *As functions of the type $SPred \to CPred$*
(i) *$true\,;\_$ is universally conjunctive, and*
(ii) *$\mathtt{F}\,;\_$ is positively conjunctive.*

5. Weakest preconditions

In the precondition semantics of Dijkstra and Scholten [3] a statement s is characterized by two state-predicate transformers, *wp.s* and *wlp.s*. We introduce these functions into our algebra via their intended interpretation.

Definition 22. For all s and state predicates q
(i) $wp.s.q = \mathscr{A}(s \Rightarrow \mathtt{F}\,;q)$, and
(ii) $wlp.s.q = \mathscr{A}(s \Rightarrow true\,;q)$.

Now that we have included the weakest preconditions in our algebra, we may seek to *prove*, as theorems in our algebra, facts that Dijkstra and Scholten *postulate*.

Since $\mathscr{A}$ is universally conjunctive (Etude 19(ii)), it follows from, respectively, Fact 21 and state-restriction rule (Fact 8(ii)) that:

Fact 23. *For all s*
(i) *$wp.s$ is positively conjunctive and $wlp.s$ is universally conjunctive.*
(ii) *$wp.s.q = wp.s.\mathbb{1} \wedge wlp.s.q$ for all state predicates q.*

Dijkstra and Scholten impose these as "healthiness conditions": properties that a pair $(wp.s, wlp.s)$ should satisfy in order to qualify as a reasonable definition of statement s. Since we can prove these properties from the intended interpretations (22), anything else would, indeed, be quite unhealthy.

From the universal conjunctivity of $\mathscr{A}$ and predicate calculus, we also get:

Fact 24 (Nondeterministic choice)**.** *For all functions s and state predicates q*
(i) *$wp.\langle\exists i :: s.i\rangle.q = \langle\forall :: wp.(s.i).q\rangle$ and*
(ii) *$wlp.\langle\exists i :: s.i\rangle.q = \langle\forall i :: wlp.(s.i).q\rangle$.*

These equalities are what Hesselink [8] uses to define "nondeterministic choice". A moment of reflection should suffice to see that, in our model, nondeterministic choice is, indeed, captured by existential quantification.

Further exploration of the preconditions is greatly simplified by the following two observations:

Fact 25. $wlp.s = wp.(s \wedge \mathtt{F})$ *for all* s.

Fact 26 (wp-dualization). $\sim wp.s.q = \mathscr{E}(s ; \sim q)$ *for all* s *and state predicates* q.

The first of these follows immediately from the definitions and the fact that, according to Etude 13(ii), $true ; q = \mathtt{E} \vee \mathtt{F} ; q$. The second is established using state restriction and dualities (Etude 19(i) and Fact 20).

Our next natural target consists of the preconditions of sequential compositions. This is the point where we finally reach the limit of our very modest initial postulates. Following [3], we expect:

Fact 27 (Compositionality of the preconditions). (i) $wp.(s;t).q = wp.s.(wp.t.q)$ *and* (ii) $wlp.(s;t).q = wlp.s.(wlp.t.q)$ *for all* s,t *and state predicates* q.

Using Fact (25) and Etude 13(iii), we find the second of these to be equal to the first instantiated with $s,t := (s \wedge \mathtt{F}), (t \wedge \mathtt{F})$, so we need to consider the first only. Attempting to verify Fact 27(i) we observe – with the types of the dummies understood – that

$$
\begin{array}{ll}
 & \langle \forall s,t,q :: \ wp.(s;t).q = wp.s.(wp.t.q) \rangle \\
= & \quad \{ \text{ negating both sides and (26) (thrice) } \} \\
 & \langle \forall s,t,q :: \ \mathscr{E}(s ; t ; \sim q) = \mathscr{E}(s ; \mathscr{E}(t ; \sim q)) \rangle \\
= & \quad \{ \text{ mutual instantiation: } q := \sim \mathbb{1} \text{ and } t := t ; \sim q, \text{ respectively } \} \\
 & \langle \forall s,t :: \ \mathscr{E}(s ; t) = \mathscr{E}(s ; \mathscr{E} t) \rangle
\end{array}
$$

and here we get stuck. The last line is not provable because, as we will see, it need not – yet – be true in our model.

We make the compositionality of the preconditions a theorem of our algebra by adding the last line in the preceding calculation to our postulates.

Postulate 28 (*Composition rule*). $\mathscr{E}(s ; t) = \mathscr{E}(s ; \mathscr{E} t)$ for all s and t.

In the next section we check what this postulate means for our model, thereby establishing both that it is reasonable and that it is independent of the previous postulates. However, we gave the composition rule in its simplest, rather than in its weakest form. In order to facilitate the analysis of the next section, we request that the reader verify that Postulate 28 follows from the evidently weaker:

Fact 28′. $[\mathscr{E}(s;\mathscr{E}t) \Rightarrow \mathscr{E}(s;t)]$ *for all finite* s *and all* t.

This is the rule that we now have a closer look at in our model.

6. Composition in the model

In our model of computations, the meaning of the existential computation quantifier is that, for any predicate s and any state π:

$$(\mathscr{E}s).\pi = \langle \exists \sigma : \pi = \sigma.0 : s.\sigma \rangle.$$

In order to keep the analysis of the composition rule in our model palatable, we introduce two abbreviations:

$$\sigma \bowtie \tau \equiv \#\sigma < \infty \wedge \text{last}.\sigma = \tau.0 \quad (\text{where } \text{last}.\sigma = \sigma.(\#\sigma - 1)),$$

$$\rho \simeq \sigma \cdot \tau \equiv \langle \exists n : n < \#\rho : \sigma = \rho \uparrow n + 1 \wedge \tau = \rho \downarrow n \rangle$$

("σ fits τ" and "ρ consists of σ followed by τ"). Now, Fact 28′ states that

$$(\mathscr{E}(s;\mathscr{E}t)).\pi \Rightarrow (\mathscr{E}(s;t)).\pi \quad \text{for all finite } s, \text{ all } t, \text{ and all states } \pi,$$

and spelling out the antecedent and consequent in isolation yields – after a considerable amount of juggling:

$$(\mathscr{E}(s;\mathscr{E}t)).\pi \equiv \langle \exists \sigma, \tau : \pi = \sigma.0 \wedge s.\sigma \wedge t.\tau : \sigma \bowtie \tau \rangle,$$

$$(\mathscr{E}(s;t)).\pi \equiv \langle \exists \sigma, \tau : \pi = \sigma.0 \wedge s.\sigma \wedge t.\tau : \langle \exists \rho :: \rho \simeq \sigma \cdot \tau \rangle \rangle.$$

Comparing the two right-hand sides we find that the question of whether Fact 28′ holds in our model boils down to the question of whether

$$(*) \qquad \sigma \bowtie \tau \Rightarrow \langle \exists \rho :: \rho \simeq \sigma \cdot \tau \rangle \quad \text{for all } \sigma \text{ and } \tau.$$

Is this reasonable? Can we prove it? Well, since $\sigma \bowtie \tau$ means that the final state of the finite computation σ coincides with the initial state of the computation τ, it is easily seen that there is a (unique) sequence of states ρ such that $\rho \simeq \sigma \cdot \tau$: just "paste" σ and τ together with one point overlap. However, the dummy ρ in $(*)$ ranges over our computation space $\mathscr{C}$ rather than over arbitrary sequences of states; thus, $(*)$ expresses that our computation space is closed under "pasting".

The current restrictions on our computation space do not allow us to conclude this. Hence, if we find it reasonable, we have to require it explicitly. We do find it reasonable and so we impose on $\mathscr{C}$ the further restriction:

Requirement 29. $\mathscr{C}$ is closed under "pasting", i.e.

$$\sigma \bowtie \tau \Rightarrow \langle \exists \rho : \rho \in \mathscr{C} : \rho \simeq \sigma \cdot \tau \rangle \quad \text{for all } \sigma \text{ and } \tau \text{ in } \mathscr{C}.$$

7. Iterators

So far, the expressive power of our algebra is rather limited. We increase the expressive power – considerably – by the introduction of two more unary operators, $\_^*$ and $\_^\infty$, respectively. The operational interpretation of these operators is that s^* corresponds to "s repeated any finite number of times" and s^∞ corresponds – roughly – to "s repeated forever". We do not, however, try to formalize these operational intentions. Instead, we define the operators by fixpoint expressions that have proven to be fundamental in similar algebraic contexts. Subsequently, we have a closer look at the operational significance of these operators in the current context. For a glossary on fixpoints, we refer the reader to the appendix.

Definition 30. $\_^*, \_^\infty : CPred \to CPred$,
(i) $s^* = \langle \boldsymbol{\mu} x :: s ; x \vee \mathbb{1} \rangle$ and
(ii) $s^\infty = \langle \boldsymbol{\nu} x :: s ; x \rangle$ for all s.

We call these operators the "finite iterator" and "infinite iterator", respectively, and they bind stronger than negation and the computation quantifiers.

Arguably the single most important aspect of these operators is that they give us a handle on both extreme solutions of the 'tail-recursion equation' $x = s ; x \vee t$.

Fact 31 (Tail-recursion theorem)**.** *For all s and t*
(i) $\langle \boldsymbol{\mu} x :: s ; x \vee t \rangle = s^* ; t$ *and*
(ii) $\langle \boldsymbol{\nu} x :: s ; x \vee t \rangle = s^* ; t \vee s^\infty$.

Proof. Along with a lot of other properties of 'Kleene-stars in general', a proof of (i) can be found in [13]. For the sake of completeness, we repeat it here.

$$
\begin{array}{ll}
 & s^* ; t = \langle \boldsymbol{\mu} x :: s ; x \vee t \rangle \\
= & \quad \{ \text{ Definition 30(i), dummy renaming } \} \\
 & \langle \boldsymbol{\mu} x :: s ; x \vee \mathbb{1} \rangle ; t = \langle \boldsymbol{\mu} y :: s ; y \vee t \rangle \\
\Leftarrow & \quad \{ \ \_ ; t \text{ is universally disjunctive, fixpoint fusion (see the appendix) } \} \\
 & \langle \forall x, y : x ; t = y : (s ; x \vee \mathbb{1}) ; t = s ; y \vee t \rangle \\
= & \quad \{ \ ; \text{over } \vee, \text{neutrality of } \mathbb{1}, \text{ Leibniz } \} \\
 & \mathit{true}.
\end{array}
$$

The proof of (ii) is similar:

$$
\begin{array}{ll}
 & s^* ; t \vee s^\infty = \langle \boldsymbol{\nu} x :: s ; x \vee t \rangle \\
= & \quad \{ \text{ Definition 30(ii), dummy renaming } \}
\end{array}
$$

$$
\begin{array}{ll}
 & s^*;t \vee \langle \boldsymbol{\nu} x :: s;x\rangle = \langle \boldsymbol{\nu} y :: s;\, y \vee t\rangle \\
\Leftarrow & \quad \{\ s^*;t \vee \_ \text{ is universally conjunctive, fixpoint fusion } \} \\
 & \langle \forall x, y : s^*;t \vee x = y : s^*;t \vee s;x = s;\, y \vee t\rangle \\
= & \quad \{\ \text{predicate calculus, } ;\text{over } \vee\ \} \\
 & \langle \forall x :: s^*;t \vee s;x = s;s^*;t \vee s;x \vee t\rangle \\
= & \quad \{\ \text{from (i) by unfolding: } s^*;t = s;s^*;t \vee t\ \} \\
 & \mathit{true}. \qquad \square
\end{array}
$$

After these preliminaries, we now have a closer look at each of our iterators in isolation.

7.1. The finite iterator

With exponentiation defined as usual: $s^0 = \mathbb{1}$ and $s^{n+1} = s;s^n$, the operational interpretation of s^n is clearly "s repeated n times". Now, one readily verifies that

Etude 32. $s^* = \langle \exists n :: s^n\rangle$ for all s and this substantiates the interpretation we gave for the finite iterator, since the right-hand side expresses "s repeated any finite number of times" almost literally.

In fact, Etude 32 is the more traditional definition of a Kleene-star and all familiar algebraic properties of Kleene-stars are, indeed, enjoyed by ours. Since re-exploring these properties is not all that exciting, we refrain from doing so; we will just state the properties that we use when and where the need arises. For a fine treatment of general Kleene-stars based on fixpoint calculus we refer the reader to [13].

We do take a closer look at the interaction between the finite iterator and the constants $\mathtt{F}$ and $\mathtt{E}$, since these are rather specific for our algebra. For any s, we calculate:

$$
\begin{array}{ll}
 & s^* \\
= & \quad \{\ \text{Definition 30(i)}\ \} \\
 & \langle \boldsymbol{\mu} x :: s;x \vee \mathbb{1}\rangle \\
= & \quad \{\ \text{13(ii)}\ \} \\
 & \langle \boldsymbol{\mu} x :: (s \wedge \mathtt{F});x \vee (s \wedge \mathtt{E}) \vee \mathbb{1}\rangle \\
= & \quad \{\ \text{tail-recursion Theorem 31(i), ;over } \vee,\ \text{neutrality of } \mathbb{1}\ \} \\
 & (s \wedge \mathtt{F})^*;(s \wedge \mathtt{E}) \vee (s \wedge \mathtt{F})^*.
\end{array}
$$

The first of these disjuncts is eternal and, since $\mathtt{F}^* = \mathtt{F}$ (etude), the second disjunct is finite. Thus, the eternal s^*-computations are those that eventually get stuck in an

eternal s-subcomputation and the finite ones are those wherein every s-subcomputation is finite. This is highly unsurprising but it is good to note that finite iteration does not, in itself, introduce any 'new' nonterminating computations: if s is finite, so is s^*.

For later reference we list two insights we just obtained:

Fact 33. *For all* s
(i) $[s \Rightarrow \mathtt{F}] \Rightarrow [s^* \Rightarrow \mathtt{F}]$ *and*
(ii) $s^* \wedge \mathtt{F} = (s \wedge \mathtt{F})^*$.

7.2. The infinite iterator

Although the infinite iterator is less well known than the Kleene-star, we are equally brief about its algebraic properties. These properties are similar to those of the finite iterator and, on the whole, unsurprising, e.g. the familiar "leap-frog" and "loop nesting" rules for Kleene-stars have obvious counterparts for the infinite iterator:

$$s\,;(t\,;s)^* = (s\,;t)^*\,;s \text{ and } s\,;(t\,;s)^\infty = (s\,;t)^\infty,$$
$$(s \vee t)^* = (s^*\,;t)^*\,;s^* \text{ and } (s \vee t)^\infty = (s^*\,;t)^*\,;s^\infty \vee (s^*\,;t)^\infty.$$

An exhaustive treatment of $\_^\infty$ in relational calculus can be found in [5].

The operational interpretation of s^∞ is somewhat more subtle than that of s^*. In order to get a tighter grip on the operational significance of the infinite iterator we begin with a taxonomy of s^∞-computations.

For any s, let $s_1 = s \wedge \mathbb{1}$, $s_f = s \wedge \neg\mathbb{1} \wedge \mathtt{F}$, and $s_e = s \wedge \mathtt{E}$, i.e. we partition the s-computations into singletons, finite non-singletons, and eternal computations. Then

$$\begin{aligned}
& s^\infty \\
= \quad & \{\ \text{Definition 30(ii)},\ s = s_1 \vee s_f \vee s_e\ \} \\
& \langle \boldsymbol{v}x :: (s_1 \vee s_f \vee s_e)\,;x\rangle \\
= \quad & \{\ ;\text{ over }\vee,\ \text{Preemption 11, and diagonal rule (see the appendix)}\ \} \\
& \langle \boldsymbol{v}x :: \langle \boldsymbol{v}y :: s_1\,;y \vee s_f\,;x \vee s_e\rangle\rangle \\
= \quad & \{\ \text{tail-recursion theorem (Fact 31(ii))}\ \} \\
& \langle \boldsymbol{v}x :: s_1^*;(s_f;x \vee s_e) \vee s_1^\infty\rangle \\
= \quad & \{\ s_1 \text{ is a state predicate, hence } s_1^* = \mathbb{1} \text{ and } s_1^\infty = s_1\,;\mathit{true}\ \text{(etude)}\ \} \\
& \langle \boldsymbol{v}x :: s_f;x \vee s_e \vee s_1\,;\mathit{true}\rangle \\
= \quad & \{\ \text{tail-recursion theorem (Fact 31(ii)),}\,;\text{over}\ \vee\ \} \\
& s_f^*\,;s_e \vee s_f^*\,;s_1\,;\mathit{true} \vee s_f^\infty.
\end{aligned}$$

So we find that s^∞-computations come in three flavours and now explicate each of these in the model.

Inner eternal $s_f^*;s_e$: After a finite number of finite s-executions, the entire remainder of the computation consists of a single infinite s-execution. It is this type of s^∞-computation that motivates reading $\_^\infty$ as "forever" rather than "infinitely often" (though even "forever" is misleading in the presence of short circuits).

Short circuits $s_f^*;s_1;true$: A finite number of finite s-computations ending with a singleton s-execution and followed by *anything if at all*. This term is not even eternal (unless equal to *false*) and so these computations may be rather unexpected for "s forever".

The thing to note is that singleton computations involve no computation steps and state predicates characterize just single states without any actual computation. Stretching the imagination, one can see state predicates as statements that "take no time". The equality $s_1^\infty = s_1;true$ can then be read as expressing that s_1^∞ results in a "short-circuit loop": s_1 is executed infinitely often but without any accumulation of time and, subsequently, anything may happen. Elsewhere, this is known as "unguarded recursion" or "causal loop".

Alternatively, we may just conclude that "s forever" is not an adequate interpretation of s^∞ if short circuits are possible ($s_1 \neq false$).

Outer eternal s_f^∞: Infinite computations consisting of infinitely many finite s-computations pasted together. More formally, we have for any computation τ that

$(s_f^\infty).\tau \equiv$ there is an infinite sequence *cut* of indices (naturals), such that

- $cut_0 = 0$ and $cut_i < cut_{i+1} < \#\tau$ for all i (hence $\#\tau = \infty$)
- $s.(\tau[cut_i \ldots cut_{i+1}])$ for all i (where $\tau[n \ldots k] = (\tau\uparrow k+1)\downarrow n$).

Thus, s_f^∞ characterizes "infinitely often s".

The only s^∞-computations that may unexpectedly be finite are the short circuits and these are excluded from the range of possibilities if $s_1 = false$, i.e. $[s \Rightarrow \neg\mathbb{1}]$. We introduce an adjective for the latter property of s and record the insight that infinite loops do not, ordinarily, terminate.

Definition 34. s is active $\equiv [s \Rightarrow \neg\mathbb{1}]$ for all s.

Fact 35. $[s^\infty \Rightarrow \mathtt{E}]$ *for all active s.*

Alas, our current postulates do not allow us to confirm the latter fact. So far, all our postulates are also valid in relational calculus and in this context (Fact 35) is not true. So we add this to our postulates but first eliminate the infinite iterator:

$\langle \forall s : [s \Rightarrow \neg\mathbb{1}] : [s^\infty \Rightarrow \mathtt{E}] \rangle$ i.e. Fact 35

$=$ { $\_^\infty$ is monotonic, predicate calculus }

$[(\neg\mathbb{1})^\infty \Rightarrow \mathtt{E}]$

$=$ { Definition 30(ii) and Knaster–Tarski (see the appendix) }

$$[\langle \exists s : [s \Rightarrow \neg \mathbb{1}\,;s]\,:s\rangle \Rightarrow \mathtt{E}]$$

$$= \quad \{ \text{ predicate calculus } \}$$

$$\langle \forall s : [s \Rightarrow \neg \mathbb{1}\,;s]\,:[s \Rightarrow \mathtt{E}]\rangle \text{ (Postulate 36) below}$$

and this is what we take for our new postulate.

Postulate 36. *Accumulation rule.* $[s \Rightarrow \neg \mathbb{1}\,;s] \Rightarrow [s \Rightarrow \mathtt{E}]$ for all s.

8. Tail-recursion, a.k.a. repetition

We now have a look at repetition or, slightly more generally, tail recursion. We consider recursive declarations that are, for some active s and some t, of the form

$$(*) \qquad r = s\,;r \vee t,$$

i.e. r is a choice ($\vee$) between a recursive case ($s\,;r$) and a base case (t).

Although we generally feel that a recursive procedure declaration characterizes the procedure in question uniquely, the equality above actually fails to do so. The tail-recursion theorem (Fact 31) gives the two extreme candidates as $s^*\,;t$ and $s^*\,;t \vee s^\infty$, respectively, and the latter is in general properly weaker than the former. So we have to decide what solution of a recurrence equation we expect from a computing mechanism.

We claim that recursion operationally leads to *weakest* fixpoints, i.e. every computation that is not inconsistent with the equation should be considered possible. The case of tail-recursion presents corroborative evidence for this. Spelling out the weakest solution for r from $(*)$ using the taxonomies from the previous section gives

$$r = (s \wedge \mathtt{F})^*\,;t \vee (s \wedge \mathtt{F})^*\,;(s \wedge \mathtt{E}) \vee (s \wedge \mathtt{F})^\infty.$$

This sums up the possibilities nicely: either the base case after a finite number of unfoldings, an eternal s-computation after a finite number of unfoldings, or infinite recursion.

Remark. The repetition $r =$ **while** b **do** s **od** is characterized by its first unfolding:

$$r = \textbf{if } b \textbf{ then } s\,;r \textbf{ fi} = b\,;s\,;r \vee \sim b.$$

The weakest solution for r from this is $(b\,;s)^*\,;\sim b \vee (b\,;s)^\infty$, which one would also get straight from the operational understanding of the repetition.

The restriction to *active* s ensures that at least one atomic step separates the starts of any two incarnations of r. Operationally, this is inescapable, because the repetition involves at least one jump or subroutine call. So this requirement involves no significant loss of generality. Without it, some of the things below would come apart as a result of short-circuit loops.

The restriction to active s means that the recursion in $(*)$ is "guarded". In our algebra, this does *not* imply that $(*)$ has a unique solution, as it does in many process algebras. The price one pays for having unique solutions to guarded recurrence equations is the inability to distinguish between "unbounded" and "infinite".

Finally, note that nondeterminism has nothing to do with the fixpoint not being unique. The point is that the strongest fixpoint fails to capture the possibility of infinite unfolding.

Now, that we have the pieces together for dealing with tail recursion we take a look at the preconditions. Generalizing Dijkstra and Scholten's treatment of the repetition or applying Hesselink's theory of general recursion leads us to expect

Fact 37. *For all active s, all t, and all state predicates q*
(i) $wp.\langle \boldsymbol{\nu} x :: s\,; x \vee t\rangle.q = \langle \boldsymbol{\mu} y :: wp.s.y \wedge wp.t.q\rangle$ *and*
(ii) $wlp.\langle \boldsymbol{\nu} x :: s\,; x \vee t\rangle.q = \langle \boldsymbol{\nu} y :: wlp.s.y \wedge wlp.t.q\rangle$.

Let us see whether we can confirm this. The closed form of the tail-recursion theorem suggests that we look at the iterators in isolation first and we begin with the finite iterator. For any s and state predicate q, we calculate:

$$
\begin{array}{ll}
 & wp.s^*.q \\
= & \quad \{\ wp\text{-dualization (Fact 26)}\ \} \\
 & \sim\mathscr{E}(s^*\,; \sim q) \\
= & \quad \{\ \text{tail recursion theorem (Fact 31(i))}\ \} \\
 & \sim\mathscr{E}\langle \boldsymbol{\mu} x :: s\,; x \vee \sim q\rangle \\
= & \quad \{\ \mathscr{E} \text{ is universally disjunctive. Heading for fixpoint fusion, we calculate for} \\
 & \qquad \text{any } x \text{ and } y \text{ such that } \mathscr{E}x = y \\
 & \qquad \mathscr{E}(s\,; x \vee \sim q) \\
 & \qquad = \quad \{\ \mathscr{E} \text{ over } \vee\text{, composition rule (Postulate 28)}\ \} \\
 & \qquad \mathscr{E}(s\,; \mathscr{E}x) \vee \mathscr{E}\sim q \\
 & \qquad = \quad \{\ \mathscr{E}x = y \text{ (given) and } \mathscr{E}\sim q = \sim q \text{ (etude)}\ \} \\
 & \qquad \mathscr{E}(s\,; y) \vee \sim q. \\
 & \qquad \text{Thus, we can now apply fixpoint fusion.}\} \\
 & \sim\langle \boldsymbol{\mu} y :: \mathscr{E}(s\,; y) \vee \sim q\rangle
\end{array}
$$

$=$ { fixpoint dualisation, de Morgan, and *wp*-dualization (Fact 26)}

$\langle \boldsymbol{\nu} y :: wp.s.y \wedge q \rangle$.

Using Facts 25 and 33(ii), the equality we just derived can be transformed into the corresponding one for the liberal precondition. So we have:

Fact 38. *For all s and state predicates q*:
(i) $wp.s^*.q = \langle \boldsymbol{\nu} y :: wp.s.y \wedge q \rangle$ *and*
(ii) $wlp.s^*.q = \langle \boldsymbol{\nu} y :: wlp.s.y \wedge q \rangle$.

In essence, this takes care of Fact 37(ii). Using the closed form from the tail-recursion theorem and the rules for the liberal preconditions of disjunctions (Fact 24), compositions (Fact 27), and finite iterations (above), we get

$$wlp.\langle \boldsymbol{\nu} x :: s ; x \vee t \rangle.q = \langle \boldsymbol{\nu} y :: wlp.s.y \wedge wlp.t.q \rangle \wedge wlp.s^{\infty}.q.$$

Since s is active, s^{∞} is eternal by Fact 35 and hence (using Fact 25) $wlp.s^{\infty}.q = \mathbb{1}$. Consequently, the final conjunct above vanishes. Note that this final conjunct would not vanish, and indeed Fact 37(ii) would be false, if s were not active.

The equality of Fact 37(i) is less straightforward. Using the same recipe as above, we get for the left-hand side of Fact 37(i).

$$wp.\langle \boldsymbol{\nu} x :: s ; x \vee t \rangle.q = \langle \boldsymbol{\nu} y :: wp.s.y \wedge wp.t.q \rangle \wedge wp.s^{\infty}.q,$$

whereas, using fixpoint fusion, we get for the right-hand side of Fact 37(i)

$$\langle \boldsymbol{\mu} y :: wp.s.y \wedge wp.t.q \rangle = \langle \boldsymbol{\nu} y :: wp.s.y \wedge wp.t.q \rangle \wedge \langle \boldsymbol{\mu} y :: wp.s.y \rangle.$$

Thus, we would be done if we can establish the equality of the two final conjuncts on the right (which amounts to our original proof obligation (Fact 37(i)) with $t := \mathit{false}$). Now,

$wp.s^{\infty}.q = \langle \boldsymbol{\mu} y :: wp.s.y \rangle$

$=$ { dualization, $s^{\infty} ; {\sim}q = s^{\infty}$ by (11) since s^{∞} is eternal }

$\mathscr{E} s^{\infty} = \langle \boldsymbol{\nu} y :: \mathscr{E}(s ; y) \rangle$

$=$ { '$\Rightarrow$' is an immediate consequence of fixpoint induction }

$[\langle \boldsymbol{\nu} y :: \mathscr{E}(s ; y) \rangle \Rightarrow \mathscr{E} s^{\infty}]$

$=$ { Knaster–Tarski and predicate calculus }

$\langle \forall y : [y \Rightarrow \mathscr{E}(s ; y)] : [y \Rightarrow \mathscr{E} s^{\infty}] \rangle$.

Sadly, this is not – yet – provable because it need not – yet – be true in our model. We did leave room in our model for the *possibility* of computations being eternal, but

nothing in our current assumptions allows the conclusion that our computation space *actually contains* eternal computations. If all computations were finite, the term in the last line above would almost always be false, but the same would not hold for the range.

The point is that the nonliberal precondition distinguishes itself from the liberal one also by guaranteeing termination. If certain eternal computations are a priori excluded from the range of possibilities, it may well be that a precondition that is weaker than the strongest fixpoint in Fact 37(i) would be sufficient.

We do not believe in magical termination and we make our scepticism explicit by adding the final line of the preceding calculation as a postulate to our algebra.

Postulate 39 (*Cycle rule*). $[p \Rightarrow \mathscr{E}(s\,;p)] \Rightarrow [p \Rightarrow \mathscr{E}s^{\infty}]$ for all p and s.

(This brings the total of our postulates up to Postulates 5, 6, 28, 36, and 39.)

Before we check what this means for our model, we record the fixpoint equality we had rewritten to obtain the cycle rule (see preceding calculation):

Fact 40. $\mathscr{E}s^{\infty} = \langle \boldsymbol{v}\, y :: \mathscr{E}(s\,;y)\rangle$ *for all* s.

And now we have a look at the model.

9. Cycles in the model

Let p and s satisfy $[p \Rightarrow \mathscr{E}(s\,;p)]$, i.e. the antecedent of the cycle rule. We would like to establish the consequent of the cycle rule: $[p \Rightarrow \mathscr{E}s^{\infty}]$, i.e. every p-state is initial for some s^{∞}-computation. Roughly, the reasoning goes as follows.

For every initial state satisfying p we have, on account of $[p \Rightarrow \mathscr{E}(s\,;p)]$ an s-computation that, if finite, ends with a state that satisfies p again. If existent, this final state is initial for another such s-computation which fits the previous one and, hence can be pasted to it. Repeating the process indefinitely or until an infinite s-computation is encountered, now yields our s^{∞}-computation.

Well, almost but not quite. Our 'computation-under-construction' converges to a sequence of states that would, indeed, be an s^{∞}-computation *if only it were a computation*. But nothing in the current assumptions about $\mathscr{C}$ allows us to conclude the latter: $\mathscr{C} = \mathscr{S}^{+}$ would satisfy all restrictions well.

We remedy the situation by imposing yet another restriction on our model.

Requirement 41. $\mathscr{C}$ is "limit closed", i.e. every nonempty subset of $\mathscr{C}$ that is linear w.r.t the prefix-order has an upper bound in $\mathscr{C}$ (upper bound w.r.t to the prefix-order, naturally).

It should be noted that the limit closedness is stronger than necessary for ensuring the validity of the cycle rule. Putting it the other way around: the cycle rule is too

weak to capture limit closedness in its full generality. We make do with the cycle rule because it is simple and it suffices for almost everything we want. Notable exceptions to the latter are general (mutual) recursion and UNITY programs with infinite assign sections.

10. Temporal logic and atomic steps

Having dealt to our satisfaction with the pre-/postcondition semantics of simple imperative programs, we now raise our sights. In pre-/postcondition semantics, the only things that interest us in a program are, for every initial state, the reachable final states and the possibility of nontermination. This is OK as long as these are the only things that matter.

There are, however, situations in which what happens in the course of program execution is also relevant or is all that is relevant. In particular, this is the case if programs are supposed to interact (with a user or with other programs). In such cases, we want to be able to specify run-time behaviour.

Various temporal logics have been designed precisely for this purpose. We have already mentioned the "branching time" temporal logic CTL* from Emerson and Srinivasan [7], which is the logic from which we have borrowed our computation quantifiers. CTL* is an extension of another logic: the "linear time" temporal logic LTL from Manna and Pnueli [12]. In fact, the trace quantifiers that we borrowed are exactly what CTL* adds to LTL.

The language of LTL extends standard logic with a handful of temporal operators intended to be used to specify behaviour in time. The single most important of these is the unary "next-time operator" **X**: a computation satisfies $\mathbf{X}s$ if the computation that follows the first atomic step satisfies s.

There are two reasons why this is an operator of fundamental importance. Firstly, all the other temporal operators of LTL can be expressed in terms of it. Hence, **X** is all we need to obtain the full expressive power of LTL. Secondly and more importantly, this operator makes the grain of atomicity explicit and doing so is essential for reasoning about parallel programs with interleaving semantics.

So we add the next-operation to our algebra. There is, however, a slight difference between LTL and our algebra. Where LTL is designed to be sound for models containing eternal computations only, the model from which we are abstracting our algebra contains finite computations as well. In particular, our computation space contains atomic computations. Consequently, we can add "next" as a constant instead of an operator.

An atomic computation is a computation of length 2 and, after introducing the predicate $\mathtt{X}$ that holds for exactly these:

Definition 42. $\mathtt{X}.\tau \equiv \#\tau = 2$,
we can express the next-operation of LTL as $\mathbf{X}s = \mathtt{X}\,;s$.

To add $\mathtt{X}$ to the computation calculus, we need some algebraic characterization of it. The first thing we note is that $\mathtt{X};\mathit{true}$ holds exactly for the nonsingleton computations:

(a) $[\mathbb{1} \not\equiv \mathtt{X};\mathit{true}]$.

From this fact it can be inferred that singleton computations do not satisfy $\mathtt{X}$ and that atomic steps do. However, (a) also holds for $\mathtt{X} := \neg\mathbb{1}$ – it follows from (a) that this is so – thus, we still have to capture the fact that computations of length greater than 2 do not satisfy $\mathtt{X}$. We do this by excluding that possibility of one computation having more than one prefix satisfying $\mathtt{X}$:

(b) $[\mathtt{X};s \wedge \mathtt{X};\neg s \Rightarrow \mathit{false}]$ for all s.

Seen as equation in the unknown $\mathtt{X}$, (a) $\wedge$ (b) has a unique solution in our model which is given by Definition 42. But, models for our current postulates can be constructed wherein there are no solutions at all. Hence, we have to add this to our postulates.

Right at the beginning, we required that our state space be contained in our computation space. This does not exclude the possibility of states that occur *only* in a singleton computation. In fact, a computation space consisting of nothing else but singleton computations satisfies all the restrictions we imposed so far.

Since we consider this an undesirable degeneration of our model, we replace the no-junk property by the stronger requirement:

Requirement 43. An atomic step is possible for every initial state, which can be rendered algebraically as

(c) $\mathscr{E}\mathtt{X} = \mathbb{1}$.

11. Atomicity and temporal logic

We extend our algebra with a new constant $\mathtt{X}$ of type *CPred* satisfying the following properties:

Postulate 44 (*Atomicity rules*).
(i) $[\mathbb{1} \not\equiv \mathtt{X};\mathit{true}]$.
(ii) $[\mathtt{X};s \wedge \mathtt{X};\neg s \Rightarrow \mathit{false}]$ *for all* s.
(iii) $\mathscr{E}\mathtt{X} = \mathbb{1}$.

We list some elementary consequences of these postulates:

Fact 45.
(i) $[\mathtt{X} \Rightarrow \neg\mathbb{1}]$ *and* $[\mathtt{X} \Rightarrow \mathtt{F}]$
(ii) $\mathtt{X}^{\infty} = \mathtt{E}$ *and* $\mathtt{X}^{*} = \mathtt{F}$.
(iii) $\neg(\mathtt{X};s) = \mathtt{X};\neg s \vee \mathbb{1}$ *for all* s.
(iv) $\mathtt{X};\_$ *is positively conjunctive.*

Proof. That $\mathtt{X}$ is active and finite – i.e. (i) – follows, respectively, from Postulate 44(i) and from Postulate 44(ii) with $s := \mathit{true}$. From (i), and Facts 35 and 33(i), we immediately obtain '$\Rightarrow$' for the two equalities in (ii) and the reverse implications are established by

$$
\begin{array}{ll}
\quad [\mathtt{E} \Rightarrow \mathtt{X}^{\infty}] & \quad [\mathtt{X}^{*} \Leftarrow \mathtt{F}] \\
\Leftarrow \quad \{ \text{ fixpoint induction } \} & = \quad \{ \text{ predicate calculus } \} \\
\quad [\mathtt{E} \Rightarrow \mathtt{X}\,;\mathtt{E}] & \quad [\mathtt{X}^{*} \vee \mathtt{E}] \\
= \quad \{ \text{ Definition 9 of } \mathtt{E} \} & = \quad \{ \text{ neutralities and } \mathtt{E} = \mathtt{X}^{\infty} \} \\
\quad [\mathtt{E} \Rightarrow \mathtt{X}\,;\mathit{true}\,;\mathit{false}] & \quad [\mathit{true} \Rightarrow \mathtt{X}^{*}\,;\mathbb{1} \vee \mathtt{X}^{\infty}] \\
= \quad \{ \text{ Fact 10 and Postulate 44(i) } \} & \Leftarrow \quad \{ \text{ Fact 31(ii) and fixpoint induction } \} \\
\quad [\mathtt{E} \Rightarrow \neg\mathbb{1} \wedge \mathtt{E}] & \quad [\mathit{true} \Rightarrow \mathtt{X}\,;\mathit{true} \vee \mathbb{1}] \\
= \quad \{ \mathtt{E} \text{ is active } \} & = \quad \{ \text{ Postulate 44(i), predicate calculus } \} \\
\quad \mathit{true}. & \quad \mathit{true}.
\end{array}
$$

The proofs of (iii) and its immediate consequence (iv) are left to the reader. □

With the constant $\mathtt{X}$, we have the full expressive power of LTL at our disposal (the full expressive power of CTL* in fact). The other temporal operators of LTL are "some-time", "always", and "until". Since we have no use for "until" we refrain from introducing it. The "some-time" operator is something we already have: it is the prefix $\mathtt{F}\,;\_$. In order to be in accordance with the axiomatization of LTL, this operator should satisfy:

Fact 46. $\mathtt{F}\,;s = \langle \boldsymbol{\mu} x :: \mathtt{X}\,;x \vee s \rangle$ *for all* s.

This fact follows immediately from Fact 45(ii) and the tail-recursion theorem (Fact 31(i)). The most salient algebraic properties of "some time" are captured by:

Fact 47. $\mathtt{F}\,;\_$ *is a universally disjunctive closure* (see the appendix).

Which follows from Fact 12 and Etude 13(i).

We define "always" as the dual of "some time":

Definition 48. $\mathtt{G} : \mathit{CPred} \rightarrow \mathit{CPred}$, $\mathtt{G}s = \neg(\mathtt{F}\,;\neg s)$ for all s.

On account of being the dual of a universally disjunctive closure:

Fact 49. $\mathtt{G}$ *is a universally conjunctive interior* (see the appendix).

The axiomatization of LTL for the always-operator boils down to the fixpoint characterization $\mathtt{G}s = \langle \boldsymbol{\nu} x :: \mathtt{X}\,;x \wedge s \rangle$, but – since our computations may be finite – this does not hold in our algebra. Instead, dualizing Fact 46 and simplifying the result with Fact 45(iii) yields:

Fact 50. $\mathtt{G}s = \langle \nu x :: (\mathtt{X}\,;x \vee \mathbb{1}) \wedge s \rangle$ *for all* s.

If s is eternal, $(\mathtt{X}\,;x \vee \mathbb{1}) \wedge s = \mathtt{X}\,;x \wedge s$ since $\mathbb{1}$ is finite. So, in that case, the above fixpoint characterization reduces to the one of LTL.

Taking care of the cosmetic differences necessary to cater for the existence of finite computations, the entire axiomatization of LTL can now be proved correct; so our algebra subsumes this logic completely. The reader may wonder how we fare with CTL*, but there is nothing to subsume here since CTL* has no axiomatization that we know of. Emerson and Srinivasan [7] do give an axiomatization for a (small) segment of CTL* and those axioms and inference rules are, indeed, all provable in our algebra (Postulate 44(iii) is essential here).

12. Persistence

The fixpoints of the interior operator G are of particular interest: they occur frequently and are pleasant to work with. We call these predicates 'persistent':

Definition 51. s is persistent $\equiv s = \mathtt{G}s$ for all s.

Since G is strengthening, the mathematical content of this equality consists of the implication $[s \Rightarrow \mathtt{G}s]$.

If a computation has some persistent property, all of its suffixes have the same property. This inheritance is something we frequently exploit and the following rule makes it algebraically explicit.

Fact 52 (Persistence rule)**.**

$$[t\,;u \wedge s \Rightarrow t\,;(u \wedge s)] \quad \textit{for all persistent } s \textit{ and all } t \textit{ and } u.$$

Proof. This follows after shunting from the observation that:

$$\begin{array}{ll} & t\,;u \\ = & \quad \{\ \text{Etude 13(ii)}\ \} \\ & (t \wedge \mathtt{E}) \vee (t \wedge \mathtt{F})\,;u \\ = & \quad \{\ \text{excluded middle and distribution}\ \} \\ & (t \wedge \mathtt{E}) \vee (t \wedge \mathtt{F})\,;(s \wedge u) \vee (t \wedge \mathtt{F})\,;(\neg s \wedge u) \\ \Rightarrow & \quad \{\ \text{Etude 13(ii), monotonicity}\ \} \\ & t\,;(s \wedge u) \vee \mathtt{F}\,;\neg s \\ = & \quad \{\ \mathtt{F}\,;\neg s = \neg \mathtt{G}s,\ s \text{ is persistent}\ \} \\ & t\,;(s \wedge u) \vee \neg s. \qquad \square \end{array}$$

Using the persistence rule in any particular circumstance requires that we establish the persistence of the particular instance of s involved. Obviously, a direct proof of $[s \Rightarrow \mathtt{G}s]$ would do, but there are cheaper alternatives. The first and cheapest is induction

on the syntax; from the idempotency, universal conjunctivity, and monotonicity of G respectively, it follows that

Fact 53. *For all s and functions t*
(i) $\mathtt{G}s$ *is persistent, and*
(ii) $\langle \forall i :: t.i \text{ is persistent} \rangle \Rightarrow \langle \forall i :: t.i \rangle$ *and* $\langle \exists i :: t.i \rangle$ *are persistent.*

The second cheap way for establishing persistence consists in weakening the proof obligation $[s \Rightarrow \mathtt{G}s]$ using Fact 50 and fixpoint induction to $[s \Rightarrow \mathtt{X}; s \vee \mathbb{1}]$. In the – usual – case where s is active, the final '$\ldots \vee \mathbb{1}$' vanishes and this strategy boils down to

Fact 54. s *is persistent* $\Leftarrow [s \Rightarrow \mathtt{X}\,; s]$ *for all* s.

As an easy consequence of the persistence rule, we get a relation between the always-operator G and the infinite iterator $\_^{\infty}$. For any s we have

$$
\begin{array}{ll}
 & \mathtt{G}(s\,; true) \\
= & \quad \{\ \mathtt{G} \text{ is strengthening } \} \\
 & s\,; true \wedge \mathtt{G}(s\,; true) \\
\Rightarrow & \quad \{\ \mathtt{G}(s\,; true) \text{ is persistent, persistence rule (52) } \} \\
 & s\,; \mathtt{G}(s\,; true),
\end{array}
$$

from which we conclude – by fixpoint induction – that $[\mathtt{G}(s\,; true) \Rightarrow s^{\infty}]$. Asking ourselves the question under which conditions the reverse implication – and hence equality – would hold, we first note that $\mathtt{G}(s\,; true) = s^{\infty}$ implies that s^{∞} is persistent. Thus the latter is a necessary condition for the reverse implication to hold. It is also sufficient:

$$
\begin{array}{ll}
 & [\mathtt{G}(s\,; true) \Leftarrow s^{\infty}] \\
= & \quad \{\ s^{\infty} \text{ is persistent and unfolding } \}
\end{array}
$$

$$
\begin{array}{ll}
 & [\mathtt{G}(s\,; true) \Leftarrow \mathtt{G}(s\,; s^{\infty})] \\
= & \quad \{\ \text{monotonicity } \} \\
 & true.
\end{array}
$$

So we have found

Fact 55. *For all s*
(i) $[\mathtt{G}(s\,; true) \Rightarrow s^{\infty}]$
(ii) $\mathtt{G}(s\,; true) \Rightarrow s^{\infty} \equiv s^{\infty}$ *is persistent.*

13. Atomic actions

An atomic action is a statement generating only single-step computations:

Definition 56. a is atomic $\equiv [a \Rightarrow \mathtt{X}]$ for all a.

Atomic actions satisfy a property that is similar to the state restriction rule (Fact 8(i)) for state predicates.

Fact 57 (Atomic split). $a\,;s = a\,;true \wedge \mathtt{X}\,;s$ *for all atomic* a *and all* s.

Proof. On account of monotonicity it sufficies to prove '$\Leftarrow$' and this follows after shunting from the observation that:

$$
\begin{array}{ll}
 & a\,;s \vee \neg(\mathtt{X}\,;s) \\
\Leftarrow & \{\ \text{Postulate 44(ii)}\ \} \\
 & a\,;s \vee \mathtt{X}\,;\neg s \\
\Leftarrow & \{\ a \text{ is atomic}, ; \text{over } \vee,\ \text{excluded middle}\ \} \\
 & a\,;true. \qquad \square
\end{array}
$$

When reasoning on the atomic level, we occassionally need the following stronger version of the atomic split.

Fact 58 (Abide rule). *For all atomic* a *and* b *and all* s *and* t

$$a\,;s \wedge b\,;t = (a \wedge b)\,;(s \wedge t).$$

Since our algebra is by now quite powerful, one might expect the verification of the abide rule to be unproblematic. Sadly, it is not. Using the atomic split, one readily verifies that it suffices to prove the abide rule for s and t both equal to *true*, but then we get stuck.

The problem is that the validity of the abide rule in the model depends upon a property of the prefix order $\preccurlyeq$ that our current postulates do not address:

$$\langle \exists \rho :: \sigma \preccurlyeq \rho \wedge \tau \preccurlyeq \rho \rangle \Rightarrow \sigma \preccurlyeq \tau \vee \tau \preccurlyeq \sigma \text{ for all computations } \sigma \text{ and } \tau.$$

In [9], something akin to this is called "local linearity".

This brings us to the final postulate of our algebra.

Postulate 59 (*Linearity*). For all s and t

$$[s\,;true \wedge t\,;true \Rightarrow (s\,;true \wedge t)\,;true \vee (s \wedge t\,;true)\,;true].$$

We leave it to the reader to prove the abide rule from this.

The growing number of postulates is making it increasingly difficult to find alternative models and we have not been able to construct a model that establishes that our final postulate does not follow from the rest. Still, we are reasonably confident that it does not.

14. Unity

The need for formal support for operational reasoning was driven home to us in the course of our explorations of UNITY. State-predicate transformers for UNITY with a straightforward operational interpretation often have an intractable mathematical characterization, which makes the correctness of such a characterization nontrivial. As proof of the pudding, we apply all the machinery we have developed to progress in UNITY.

A small warning may be in order. Using the algebra does not make operational reasoning much less cumbersome than it otherwise would be, only much less error-prone. Spelling out the derivations below in meticulous detail would require a prohibitive number of pages. Therefore, the calculations are rather coarse grained, with many of the steps motivated by an 'etude'.

A UNITY program is given by a nonempty finite set A of atomic actions. Execution of the program results in an infinite sequence of steps such that

– each step consists of the execution of some action from A, and
– every action in A is executed infinitely often.

Defining s by $s = \langle \exists a\colon a \in A : a \rangle$, the computations described above are those that

– consist of an infinite sequence of s-steps, and
– contain, for every $a \in A$, an infinite number of a-steps.

From the definition of s it follows that s is atomic and that $[a \Rightarrow s]$ for all $a \in A$. These consequences are all we ever need and, therefore, all that we require in the formal characterization of UNITY computations. We give some equivalent characterizations; the reader may choose the one that he/she considers to be most convincing as *the* definition.

Definition and Fact 60. *For any atomic s and nonempty finite set A such that $[a \Rightarrow s]$ for all $a \in A$, we define – with the range $a \in A$ understood – that*

$$\begin{aligned}
&\text{(i)} && \mathit{unity.s.A} = \mathtt{G}(s\,;\mathit{true}) \wedge \langle \forall a :: \mathtt{G}(\mathtt{F}\,;a\,;\mathit{true})\rangle,\\
&\text{(ii)} && = s^{\infty} \wedge \langle \forall a :: (\mathtt{F}\,;a)^{\infty}\rangle,\\
&\text{(iii)} && = \langle \boldsymbol{\nu} x :: \langle \forall a :: s^{*}\,;a\,;x\rangle\rangle,\\
&\text{(iv)} && = \langle \forall a :: (s^{*}\,;a)^{\infty}\rangle.
\end{aligned}$$

Proof. The first two are direct translations into mathematics of the English description given above. The verification that these two are equal is reasonably straightforward:

$$\begin{aligned}
&\mathtt{G}(s\,;\mathit{true}) \wedge \langle \forall a :: \mathtt{G}(\mathtt{F}\,;a\,;\mathit{true})\rangle = s^{\infty} \wedge \langle \forall a :: (\mathtt{F}\,;a)^{\infty}\rangle\\
&\Leftarrow \quad \{\ \text{Leibniz and Fact 55}\ \}\\
&\quad s^{\infty} \text{ is persistent} \wedge \langle \forall a :: (\mathtt{F}\,;a)^{\infty} \text{ is persistent}\rangle\\
&\Leftarrow \quad \{\ \text{Fact 54}\ \}
\end{aligned}$$

$$[s^{\infty} \Rightarrow \mathtt{X}\,;s^{\infty}] \wedge \langle \forall a :: [(\mathtt{F}\,;a)^{\infty} \Rightarrow \mathtt{X}\,;(\mathtt{F}\,;a)^{\infty}]\rangle$$

$=$ { etude }

true.

Now, let $u = unity.s.A$ as defined by the first two. Then

$$[u \Rightarrow \langle \boldsymbol{\nu} x :: \langle \forall a :: s^{*}\,;a\,;x\rangle\rangle]$$

$\Leftarrow$ { fixpoint induction, predicate calculus }

$$\langle \forall a :: [u \Rightarrow s^{*}\,;a\,;u]\rangle$$

$=$ { from (i) it follows that u is persistent, persistence rule (Fact 52) }

$$\langle \forall a :: [u \Rightarrow s^{*}\,;a\,;\mathit{true}]\rangle$$

$\Leftarrow$ { $[b^{\infty} \wedge \mathtt{F}\,;t \Rightarrow b^{*}\,;t]$ for all atomic b and all t (etude) }

$$\langle \forall a :: [u \Rightarrow s^{\infty} \wedge \mathtt{F}\,;a\,;\mathit{true}]\rangle$$

$=$ { immediate from (ii) }

true.

And we close the cycle by observing that

(iii) $\langle \boldsymbol{\nu} x :: \langle \forall a :: s^{*}\,;a\,;x\rangle\rangle$

$\Rightarrow$ { monotonicity }

$$\langle \forall a :: \langle \boldsymbol{\nu} x :: s^{*}\,;a\,;x\rangle\rangle$$

$\Rightarrow$ { Definition 30(ii) }

(iv) $\langle \forall a :: (s^{*}\,;a)^{\infty}\rangle$

$\Rightarrow$ { range nonempty, $[a \Rightarrow s], [s \Rightarrow \mathtt{X}]$ }

$$(s^{*}\,;s)^{\infty} \wedge \langle \forall a :: (\mathtt{X}^{*}\,;a)^{\infty}\rangle$$

$=$ { $(s^{*}\,;s)^{\infty} = s^{\infty}$ (etude), $\mathtt{X}^{*} = \mathtt{F}$ (Fact 45(ii)) }

(ii) $s^{\infty} \wedge \langle \forall a :: (\mathtt{F}\,;a)^{\infty}\rangle$. □

For the rest of this section we assume that s and A satisfy the premises of Definition 60 and that $u = unity.s.A$.

In UNITY logic, as it was designed by Chandy and Misra [1], progress of a unity program is captured by the relation $\mapsto$ ("leads to") on state predicates. Operationally, $p \mapsto q$ means that in any u-computation, every state satisfying p is succeeded eventually

– possibly simultaneously – by some state satisfying q. Carefully transcribing this into our algebra yields

Definition 61. $p \mapsto q \equiv [u \Rightarrow \mathtt{G}(\cdot p \Rightarrow \mathtt{F}\,;\cdot q)]$ for all p and q.

For any q, we calculate the weakest p that leads to q:

$$
\begin{array}{ll}
 & p \mapsto q \\
= & \quad \{ \text{ Definition 61 } \} \\
 & [u \Rightarrow \mathtt{G}(\cdot p \Rightarrow \mathtt{F}\,;\cdot q)] \\
= & \quad \{ \text{ } u \text{ is persistent, i.e. “open” w.r.t. to the interior operator } \mathtt{G} \text{ } \} \\
 & [u \Rightarrow (\cdot p \Rightarrow \mathtt{F}\,;\cdot q)] \\
= & \quad \{ \text{ shunt twice } \} \\
 & [\cdot p \Rightarrow (u \Rightarrow \mathtt{F}\,;\cdot q)] \\
= & \quad \{ \text{ Definition 18 of } \mathscr{A} \text{ } \} \\
 & [p \Rightarrow \mathscr{A}(u \Rightarrow \mathtt{F}\,;\cdot q)].
\end{array}
$$

So, we now get:

Fact 62. *$p \mapsto q \equiv [p \Rightarrow wlt.q]$ for all p and q, where the state-predicate transformer wlt is given by*

Definition 63. $wlt.q = \mathscr{A}(u \Rightarrow \mathtt{F}\,;\cdot q)$ for all state predicates q.

Spelling out the right-hand side in English, we get

$wlt.q$: holds in those initial states for which every u-computation contains at least one state satisfying q,

which is essentially what we wrote in the introduction.

The formal definition of the predicate transformer wlt given by Jutla et al. [10] is a double fixpoint expression which, in [6], we showed to be equivalent to formula (c) of the introduction. Since the universal quantification in this formula corresponds to the weakest precondition of nondeterministic choice (Fact 24(i)) this characterization is the special case with $s = \langle \exists a :: a \rangle$ of the following:

Fact 64. $wlt.q = \langle \boldsymbol{\mu} x :: \langle \exists a :: \langle \boldsymbol{\nu} y :: q \vee (wp.s.y \wedge wp.a.x) \rangle \rangle \rangle$ *for all q.*

The computation calculus has been designed to enable us to derive characterizations such as Fact 64 from definitions such as Definition 63.

Remark. One might wonder why a characterization as involved as Fact 64 is even worth considering. However, replacing in the right-hand side of Definition 63 u by one

of its possible Definition 60(iii), and subsequently eliminating the finite iterator yields

$$\mathscr{A}(\langle \boldsymbol{\nu} x :: \langle \forall a :: \langle \boldsymbol{\mu} y :: s\,;\, y \vee a\,;\, x\rangle\rangle\rangle \Rightarrow \mathtt{F}\,;\cdot q).$$

Something like this is what Fact 64 should be compared with.

Characterization Fact 64 has two big advantages. Firstly, the dummies of the fixpoint expressions are of the type state predicate rather than computation predicate. Secondly, the computation quantifiers – hidden in *wp*-quantify over atomic steps only. Consequently, Fact 64 serves as a basis for progress considerations conducted solely in terms of state predicates and single computation steps.

Starting from Definition 63, the objective is to push the computation quantifier as deep into the syntax tree as we can manage (all the way down to the atomic actions). We begin with dualization and elimination of q from our considerations:

$$\begin{aligned}
& wlt.q \\
= \;& \{\ \text{Definition 63, dualization, and Definition 60(i)}\ \} \\
& \sim \mathscr{E}(\mathtt{G}\cdot \sim q \wedge \mathtt{G}(s\,;\mathit{true}) \wedge \langle \forall a :: \mathtt{G}(\mathtt{F}\,;a\,;\mathit{true})\rangle) \\
= \;& \{\ \text{etude}\ \} \\
& \sim \mathscr{E}(\mathtt{G}(\sim q\,;s\,;\mathit{true}) \wedge \langle \forall a :: \mathtt{G}(\mathtt{F}\,;\sim q\,;a\,;\mathit{true})\rangle) \\
= \;& \{\ \sim q\,;s \text{ and } \{\ a :: \sim q\,;a\ \} \text{ satisfy the premises of Definition 60}\ \} \\
& \sim \mathscr{E}(\mathit{unity}.(\sim q\,;s).\{\ a :: \sim q\,;a\ \}).
\end{aligned}$$

Thus, $wlt.q$ is just the complement of the domain of another UNITY program

Fact 65. $wlt.q = \sim \mathscr{E}(\mathit{unity}.(\sim q\,;s).\{\ a :: \sim q\,;a\ \})$.

Since our current UNITY program is completely general, we can do the instantiation with $s, A := \sim q\,;s, \{\ a :: \sim q\,;a\ \}$ whenever we want and all that remains to be done is determining $\mathscr{E}u$.

Now, according to Definition 60(iii), u is the weakest solution of

$$(*) \qquad x : [x \Rightarrow s^*\,;a\,;x] \quad \text{for all } a \in A$$

and, from monotonicity of $\mathscr{E}$ and composition rule Fact 28 it follows that, for any solution x of $(*)$, $\mathscr{E}x$ is a solution of

$$(**) \qquad p : [p \Rightarrow \mathscr{E}(s^*\,;a\,;p)] \quad \text{for all } a \in A.$$

Since u is the *weakest* solution of $(*)$, it is not entirely unreasonable to expect, correspondingly, $\mathscr{E}u$ to be the *weakest* solution of $(**)$. This leads us to conjecture:

Fact 66. $\mathscr{E}u = \langle \boldsymbol{\nu} x :: \langle \overset{\boldsymbol{\forall}}{a} :: \mathscr{E}(s^*\,;a\,;x)\rangle\rangle$.

We leave it to the reader to verify whether combining Facts 66 with 65 yields the fixpoint characterization Fact 64 of *wlt.q*; so proving Fact 66 is all that is left to be done.

Proof. The right-hand side is the weakest solution of $(**)$ and, since we just noted that $\mathscr{E}u$ is a solution thereof, it suffices to show that $[p \Rightarrow \mathscr{E}u]$ for any p solving $(**)$. To this end we first observe

$$
\begin{array}{ll}
 & [p \Rightarrow \mathscr{E}u] \\
= & \{ \text{ Definition 60(iv), } (s^*;a)^\infty = (s^*;a;s^*)^\infty \text{ (etude) } \} \\
 & [p \Rightarrow \mathscr{E}\langle \forall a :: (s^*;a;s^*)^\infty \rangle] \\
\Leftarrow & \{ \text{ monotonicity } \} \\
 & [p \Rightarrow \mathscr{E}\langle \forall a :: s^*;a;s^* \rangle^\infty] \\
\Leftarrow & \{ \text{ cycle rule (Postulate 39) } \} \\
 & [p \Rightarrow \mathscr{E}(\langle \forall a :: s^*;a;s^* \rangle ; p)]. \\
\Leftarrow & \{ \text{ instantiation } \} \\
 & [p \Rightarrow \mathscr{E}(\langle \forall b : b \in B : s^*;b;s^* \rangle ; p)] \quad \text{for all nonempty subsets } B \text{ of } A.
\end{array}
$$

The purpose of the last step is that, A being finite, the last line above is amenable to induction over the cardinality of B.

Singleton B: Let $B = \{a\}$ for some $a \in A$. Then

$$
\begin{array}{ll}
 & \mathscr{E}(\langle \forall b : b \in B : s^*;b;s^* \rangle ; p) \\
= & \{ \ B = \{a\} \ \} \\
 & \mathscr{E}(s^*;a;s^* ; p) \\
\Leftarrow & \{ \ [\mathbb{1} \Rightarrow s^*] \ \} \\
 & \mathscr{E}(s^*;a ; p) \\
\Leftarrow & \{ \ p \text{ solves } (**) \ \} \\
 & p.
\end{array}
$$

Non-singeleton B: Let $B = C \cup D$ with C and D (necessarily nonempty) proper subsets of B. With b, c and d understood to range over B, C, and D, respectively, we observe

$$
\begin{array}{ll}
 & p \\
\Rightarrow & \{ \text{ induction hypothesis for } C \ \}
\end{array}
$$

$$\mathscr{E}(\langle \forall c :: s^* ; c ; s^* \rangle ; p)$$

$\Rightarrow$ { induction hypothesis for D }

$$\mathscr{E}(\langle \forall c :: s^* ; c ; s^* \rangle ; \mathscr{E}(\langle \forall d :: s^* ; d ; s^* \rangle ; p))$$

$=$ { composition rule (Fact 28) }

$$\mathscr{E}(\langle \forall c :: s^* ; c ; s^* \rangle ; \langle \forall d :: s^* ; d ; s^* \rangle ; p)$$

$\Rightarrow$ { monotonicity }

$$\mathscr{E}(\langle \forall c, d :: s^* ; c ; s^* ; s^* ; d ; s^* \rangle ; p)$$

$\Rightarrow$ { $[c \Rightarrow s], [d \Rightarrow s], [s \Rightarrow s^*]$, and $[s^* ; s^* \Rightarrow s^*]$ }

$$\mathscr{E}(\langle \forall c, d :: s^* ; c ; s^* \wedge s^* ; d ; s^* \rangle ; p)$$

$=$ { predicate calculus, understood ranges of c, d, and b }

$$\mathscr{E}(\langle \forall b :: s^* ; b ; s^* \rangle ; p). \quad \square$$

The predicate transformer *wlt* is not the only tool of the trade in UNITY theory. Two others state-predicate transformers capturing temporal properties are:

Definition 67. For any state-predicate q
(i) $wst.q = \mathscr{A}(u \Rightarrow \mathtt{G}\cdot q)$ and
(ii) $wta.q = \mathscr{A}(u \Rightarrow \mathtt{F} ; \mathtt{G}\cdot q)$.

The predicate transformer 'weakest stable' (*wst*) is characterized by Chandy and Sanders [2] via

Fact 68. $wst.q = \langle \boldsymbol{\nu} y :: wp.s.y \wedge q \rangle$ *for all* q.

Upon trying to prove Fact 68 from Definition 67(i), we found that we need to assume $\mathscr{E}u = \mathbb{1}$ for that. This follows from $\mathscr{E}a = \mathbb{1}$ for all a and, since totality of all the actions is a running assumption in UNITY, this is not a serious restriction; however, that it is essential for Fact 68 to be correct had hitherto escaped our notice entirely.

The predicate transformer 'weakest to-always' (*wta*) is introduced by Sanders and the present author in [4] via

Fact 69. $wta.q = \langle \boldsymbol{\mu} x :: wlt.(wst.(x \vee q)) \rangle$.

There, a verbal argument is given for the correspondence with the operational interpretation Definition 67(ii). This verbal argument can be rendered algebraically as a proof in the computation calculus.

15. Summary and concluding remarks

We developed the computation calculus as we went along and, consequently, the ingredients are scattered throughout the text. So let us summarize. The sum total of the postulates that make up the computation calculus is:

the basic Postulates 5 and 6,
the composition rule (Postulate 28),
the accumulation rule (Postulate 36),
the cycle rule (Postulate 39),
the atomicity rules (Postulate 44), and
linearity (Postulate 59).

That is it. While, in general, we would like the algebras we use to be less complex, we feel that this is reasonably modest in view of what it achieves. We covered the precondition semantics for sequential programs of Dijkstra and Scholten. We have subsumed the linear time temporal logic of Manna and Pnueli. We have subsumed the language of CTL* and, while this "logic" has no proof system – that we know of – our algebra does give a handle on these expressions. Finally, the calculus as a whole is powerful enough for a mathematically explicit analysis of UNITY, something which no other formalism has enabled us to do.

Also, scattered throughout the text are the restriction on our computation space ensuring that it provides a model for the computation calculus. We rendered these restrictions informally as:

Nonempty segments of computations are computations.
$\mathscr{C}$ is closed under "pasting".
$\mathscr{C}$ is limit closed.
$\mathscr{C}$ contains an atomic step for every initial state.

(Note that Requirement 4, i.e. that every state constitutes a singleton computation, is subsumed by the first and last of this list.) Putting the other way around, the computation calculus is 'sound' for the class of computation spaces satisfying these four 'healthiness properties'.

We consider these healthiness requirements to be quite reasonable and so we can live with the fact that the applicability of our algebra is restricted to computation spaces satisfying them. In fact, now that we have this list explicitly in front of us, we can see that the list constitutes assumptions that we used to make implicitly and without always being fully aware of it.

Although our algebra is quite powerful, there is some room to make it stronger still. Burghard von Karger noted that the composition rule (Postulate 28) and the linearity Postulate 59 can be replaced by a single postulate ("local linearity") that appears to be stronger than the conjunction of these two. Unfortunately, the formulation of this alternative requires the introduction of an operator from sequential calculus that we were careful to avoid since we felt that it did not go well with the objective of "expressive transparency". Moreover, we do not immediately see any use for the added formal strength.

A strengthening that would definitely serve a purpose concerns the cycle rule. This rule fails to capture limit closedness in its full generality and, consequently, there are some significant problems that we cannot tackle. Unfortunately in this case, we have been unable to come up with a workable rule that captures limit closedness better.

Appendix A. On fixpoint calculus, closures and interiors

In our explorations of the computation calculus, we make heavy use of fixpoint calculus [13]. For a monotonic predicate transformer f we use $\langle \boldsymbol{\mu} x :: f.x \rangle$ to denote the strongest fixpoint and $\langle \boldsymbol{\nu} x :: f.x \rangle$ to denote the weakest fixpoint. We list the essential rules (for $\boldsymbol{\mu}$ only).

Leaving universal quantification over the free variables and monotonicity of the functions understood, we have

Theorem of Knaster–Tarski

$$\langle \boldsymbol{\mu} x :: f.x \rangle = \langle \forall x : [f.x \Rightarrow x] : x \rangle.$$

(Un-)folding

$$\langle \boldsymbol{\mu} x :: f.x \rangle = f.\langle \boldsymbol{\mu} x :: f.x \rangle.$$

Fixpoint induction

$$[f.x \Rightarrow x] \Rightarrow [\langle \boldsymbol{\mu} x :: f.x \rangle \Rightarrow x].$$

Monotonicity

$$\langle \forall x :: [f.x \Rightarrow g.x] \rangle \Rightarrow [\langle \boldsymbol{\mu} x :: g.(f.x) \rangle \Rightarrow \langle \boldsymbol{\mu} x :: g.x \rangle].$$

Fixpoint rolling

$$f.\langle \boldsymbol{\mu} x :: g.(f.x) \rangle = \langle \boldsymbol{\mu} x :: f.(g.x) \rangle.$$

Dualisation

$$\neg \langle \boldsymbol{\mu} x :: f.x \rangle = \langle \boldsymbol{\nu} x :: \neg f.(\neg x) \rangle.$$

Fixpoint fusion For universally disjunctive f

$$f.\langle \boldsymbol{\mu} x :: g.x \rangle = \langle \boldsymbol{\mu} y :: h.y \rangle \Leftarrow \langle \forall x, y : f.x = y : f.(g.x) = h.y \rangle.$$

Diagonal rule

$$\langle \boldsymbol{\mu} x :: \langle \boldsymbol{\mu} y :: f.(x, y) \rangle \rangle = \langle \boldsymbol{\mu} x :: f.(x, x) \rangle.$$

We also make some use of familiarity with 'closures' and 'interiors'. These are predicate transformers enjoying a cocktail of properties:

$$f \text{ is a closure} \equiv f \text{ is monotonic, idempotent and weakening,}$$

$$f \text{ is an interior} \equiv f \text{ is monotonic, idempotent and strengthening.}$$

The most frequently exploited algebraic properties thereof are

$$[f.x \Rightarrow f.y] \equiv [x \Rightarrow f.y] \quad \text{if } f \text{ is a closure, and}$$

$$[f.x \Rightarrow f.y] \equiv [f.x \Rightarrow y] \quad \text{if } f \text{ is an interior.}$$

References

[1] K.M. Chandy, J. Misra, Parallel Program Design, a Foundation, Addison-Wesley, Reading, MA, 1988.
[2] K.M. Chandy, B.A. Sanders, Conjunctive predicate transformers for reasoning about concurrent computation, Sci. Comput. Programming 24 (1995) 129–148.
[3] E.W. Dijkstra, C.S. Scholten, Predicate Calculus and Program Semantics, Texts and Monographs in Computer Science, Springer, New York, 1990.
[4] R.M. Dijkstra, B.A. Sanders, A predicate transformer for the progress property 'to-always', Formal Aspects Comput. 9 (1997) 270–282.
[5] R.M. Dijkstra, Relational calculus and relational program semantics, CS-R9408, University of Groningen, 1994.
[6] R.M. Dijkstra, DUALITY: a simple formalism for the analysis of UNITY, Formal Aspects Comput. 7 (1995) 353–388.
[7] E.A. Emerson, J. Srinivasan, Branching time temporal logic, in: W.P. de Roever, J.W. de Bakker, G. Rozenberger (Eds.), Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency, Lecture Notes in Computer Science, Vol. 354, Springer, Berlin, 1989, pp. 123–172.
[8] W.H. Hesselink, Programs, Recursion and Unbounded Choice, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press. Cambridge, 1992.
[9] B. von Karger, C.A.R. Hoare, Sequential calculus, Inform. Process. Lett. 53 (1995) 123–130.
[10] E. Knapp, C.S. Jutla, J.R. Rao, A predicate transformer approach to semantics of parallel programs, in: Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing, ACM Press, New York, 1989, pp. 249–263.
[11] J.J. Lukkien, Parallel program design and generalized weakest preconditions, Ph.D. Thesis, Rijksuniversiteit, Groningen, 1990.
[12] Z. Manna, A. Pnueli, Verification of concurrent programs: a temporal proof system, in: J.W. de Bakker, J. van Leeuwen (Eds.), Foundations of Computer Science, Distributed Systems: Part 2, Semantics and Logic, Mathematical Centre Tracts, Vol. 159, Mathematisch Centrum, Amsterdam, 1983, pp. 163–255.
[13] R.C. Backhouse et al., Mathematics of program construction group, Fix-point calculus, Inform. Process. Lett. 53 (1995) 131–136.