Universal extensions to simulate specifications

Hesselink, Willem

Published in:
Information and Computation

DOI:
10.1016/j.ic.2007.10.003

Publication date:
2008

Available online at www.sciencedirect.com
ScienceDirect
Information and Computation 206 (2008) 108–128
Information and Computation
www.elsevier.com/locate/ic

Universal extensions to simulate specifications

Wim H. Hesselink *

Department of Mathematics and Computing Science, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands

Received 20 September 2006; revised 17 August 2007
Available online 24 October 2007

Abstract

A previous paper introduced eternity variables as an alternative to the prophecy variables of Abadi and Lamport and proved the formalism to be semantically complete: every simulation F: $K \rightarrow\!\!\!\!\triangleright\ L$ that preserves quiescence contains a composition of a history extension, an extension with eternity variables, and a refinement mapping. This result is strengthened here in three ways.

First, the assumption of preservation of quiescence is eliminated. Second, it is shown that the intermediate extension only depends on K, and is independent of L and F. Third, in order to accommodate implementation relations where the concrete specification (occasionally) does fewer steps than the abstract specification, we weaken the concept of simulation, in such a way that it precisely corresponds to the implementation concept of Abadi and Lamport. We add stuttering history extensions to the repertoire of variable extensions, and show that this extended repertoire suffices to factorize an arbitrary (weakened) simulation.

The proofs have been verified with the theorem prover PVS. The methodology of using eternity extensions in correctness proofs is briefly discussed.

Keywords: TLA; Stuttering; Simulation; Semantic completeness; Theorem proving; Auxiliary variable

1. Introduction

Background. Concurrent and reactive systems usually cannot be specified by means of preconditions and postconditions. They are therefore typically specified in terms of the observable behaviours of the system in combination with a well specified but highly nondeterministic environment. The combination can be modelled as a state machine with some fairness or progress properties.

Indeed, in the theory of Abadi and Lamport [l] on the existence of refinement mappings, a specification is defined as a state machine with a supplementary property for fairness aspects. Behaviours of a specification are infinite sequences of states. Behaviours become visible by means of an observation function. A specification is said to implement another one when all visible behaviours of the first one are visible behaviours of the second one.

* Fax: +31 503 633 800.
E-mail address: w.h.hesselink@rug.nl

0890-5401/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2007.10.003

It is proved by Abadi and Lamport [1], under some technical assumptions (finite invisible nondeterminism and internal continuity of L, machine-closedness of K), that, when a specification K implements a specification L, there exists an extension M of K with history variables and prophecy variables together with a refinement mapping from M to L. This is a beautiful result but, when we tried to apply it, we were hampered by the three technical assumptions and by the fact that prophecy variables with *infinite* nondeterminism are unsound.

The implementation concept of Abadi and Lamport [1] is based on observability. While arguing about specifications, it is inconvenient always to be forced to make the observability explicit. In Hesselink [7], we therefore decided to use the term "*simulations of specifications*" to unify forward simulations (history extensions), backward simulations (prophecy extensions), and refinement mappings. Note that there are numerous related notions of simulations and bisimulations for transition systems, automata, process algebra terms, Kripke structures etc., cf. Milner [21], Park [24], Lynch and Vaandrager [18], de Nicola and Vaandrager [3], van Glabbeek and Weijland [26], Attie [2], Manolios [20], Griffioen and Vaandrager [5], Nejati et al. [22], but in none of these papers arbitrary liveness properties are taken care of.

Stuttering. The concept of simulation of Hesselink [7,9] allows the implementing program to take more steps than the abstract program since this is often the case. As Lamport [17, (p. 41)] has argued, however, there are realistic cases where the implementing program occasionally does *fewer* steps than the abstract program (also e.g., Ladkin et al. [16]). In such cases, the simulations of Hesselink [7,9] are inadequate and one needs the coarser abstraction of Abadi and Lamport [1]. This means to admit behaviours of the implementing program that match the specification only after stutterings are added to them. In order to capture this relationship between program and specification, we here weaken the concept of simulation.

Henceforth, the simulations of Hesselink [7,9] are called *strict*, while *simulation* stands for the weaker concept. We write $F : K \rightarrowtriangle L$ for a strict simulation F from specification K to specification L, and $F : K \rightarrow\!\!\!\!\rightarrow L$ for an arbitrary simulation from K to L. We thus distinguish a theory of strict simulations and a "stuttering" theory of general simulations.

The first main result of the stuttering theory is our Theorem 3.1 that every visible behaviour of a specification K can be extended with stutterings to a visible behaviour of a specification L if and only if there is a simulation $K \rightarrow\!\!\!\!\rightarrow L$ that respects the observation functions of K and L.

Generalizing the stuttering variables of Ladkin et al. [16] and following Manolios [20], we introduce stuttering forward simulations as the standard way for behaviours to be simulated in an inductive manner. Whereas forward simulations allow the introduction of history variables, stuttering forward simulations also allow the introduction of additional "abstract steps" in the computations.

Prophecy. Stuttering forward simulations disallow cases where the abstract behaviour takes some nondeterministic choices earlier than the concrete one, so that the simulation would need to be guided by some kind of prophecy. It was such a simulation that we needed in our investigation Hesselink [8] of the serializable database interface of Schneider [25].

Since prophecy variables are only sound under finite nondeterminism, we introduced in Hesselink [7] eternity variables as an alternative. The use of eternity variables is sound: every extension with eternity variables is a simulation which preserves quiescence. In Hesselink [9], we proved that the method is semantically complete in the sense that every simulation that preserves quiescence is a composition of an extension with history variables, an extension with eternity variables, and a refinement mapping.

In this paper, we strengthen the result and simplify the proof. In three ways the result is stronger than before. First, for any specification K, we form a "universal extension" $K \rightarrowtriangle E$ such that every strict simulation $K \rightarrowtriangle L$ factorizes over a refinement mapping $E \rightarrowtriangle L$. In other words, the intermediate specification only depends on K, not on L and F.

Second, we eliminate the assumption of preservation of quiescence, and replace in the proof the "unfolding" of a specification by its "clocking extension". In the unfolding, the whole history is encoded in the state. In the clocking extension only the number of steps is recorded, but the supplementary property is strengthened slightly in the sense that the clock keeps ticking even after termination of the useful activities.

The third contribution is the extension of the completeness result to the stuttering theory: every simulation $K \rightarrow\!\!\!\!\rightarrow L$ factorizes over the "universal extension" $K \rightarrowtriangle E$ mentioned above, followed by a stuttering history extension $E \rightarrow\!\!\!\!\rightarrow T$ and a refinement mapping $T \rightarrowtriangle L$. The technical assumptions of Abadi and Lamport[1] mentioned above are unnecessary.

All results in this paper, including the examples, have been verified with the theorem prover PVS, see Owre et al. [23] and Hesselink [10].

Overview. Section 2 contains concepts and notations on binary relations, infinite sequences, temporal operators, stutterings, and properties. In Section 3, we introduce specifications, refinement mappings, (stuttering) forward simulations, and general simulations. In Section 4, we formalize the concept of extensions, and discuss history extensions and eternity extensions. We give two examples to show how simulations can be decomposed by means of (stuttering) history extensions, eternity extensions, invariant restrictions, and refinement mappings.

Section 5 contains the results on semantic completeness for the strict theory. The semantic completeness of the stuttering theory is dealt with in Section 6. In Section 7, we briefly discuss the methodological issue of how to construct an eternity extension to prove the correctness of an implementation. Conclusions are drawn in Section 8.

Related work. The results of Abadi and Lamport [1] have been extended by several authors, e.g., Jonnson [14], Engelhardt and de Roever [4], Attie [2], Jonsson et al. [15]. In Jonnson [14], Engelhardt and de Roever [4], some of the technical assumptions of Abadi and Lamport [1] are weakened and thus made even more technical.

In Lynch and Vaandrager [18], the results of Abadi and Lamport [1] are transferred to labelled transition systems without supplementary properties. The paper of Attie [2] is restricted to a specific type of liveness properties. The paper of Jonsson et al. [15] replaces prophecy variables by transducers. It does not claim semantic completeness. The FIFO queues in their transducers play a role similar to our eternity variables, but the correspondence is not yet completely clear.

The transition systems of Manolios [20] are not necessarily reflexive and can therefore express a rudimentary way of progress. Since these systems lack initial states and supplementary properties, the completeness result of Manolios [20] is not related to the Abadi–Lamport Theorem. Indeed, its concept of stuttering simulation corresponds to our condition (SF1s) in Section 3.4, whereas the well-founded simulation seems to correspond to (SF1). Its completeness result is a cousin of our Lemma 3.1, which we found independently but later.

2. Technical material

2.1. Binary relations

We use the word *relation* for binary relation. A relation is treated as a set of pairs. So, a relation between sets X and Y is a subset of the Cartesian product $X \times Y$. We use the accessor functions *fst* and *snd* given by $fst(x, y) = x$ and $snd(x, y) = y$. A relation on X is a subset of $X \times X$. The identity relation 1_X on X consists of all pairs (x,x) with $x \in X$. Recall that a relation A on X is called *reflexive* iff $1_X \subseteq A$.

The *inverse* A^{-1} of a relation A is defined by $A^{-1} = \{(x, y) \mid (y,x) \in A\}$. For $y \in Y$, we define $A^{-1}[y] = \{x \mid (x, y) \in A\}$. For relations A and B, we write $(A; B)$ to denote the relational composition, which consists of all pairs (x,z) such that there exists y with $(x, y) \in A$ and $(y,z) \in B$.

A function $f : X \to Y$ is identified with its graph $\{(x, f(x)) \mid x \in X\}$ which is a relation between X and Y. The composition of functions $f : X \to Y$ and $g : Y \to Z$ is a function $g \circ f : X \to Z$, which equals the relational composition $(f; g)$ (note the reversal).

2.2. Infinite sequences and temporal operators

Infinite sequences are used to represent consecutive values during computations. We write X^ω for the set of infinite sequences on X, which are regarded as functions $\mathbb{N} \to X$. For $xs \in X^\omega$, its set of values is denoted by $xs(\mathbb{N}) = \{xs(i) \mid i \in \mathbb{N}\}$. Given $xs \in X^\omega$, a function $f : X \to Y$ induces a sequence $f \circ xs \in Y^\omega$. A relation $F \subseteq X \times Y$ induces a relation $F^\omega \subseteq X^\omega \times Y^\omega$ by the definition:

$$(xs, ys) \in F^\omega \quad \equiv \quad (\forall\, i : (xs(i), ys(i)) \in F)\ .$$

An infinite sequence xs is said to *terminate* iff it is eventually constant, i.e., there is a number n with $xs(i) = xs(n)$ for all $i \geqslant n$. Otherwise it is called *nonterminating*.

Let P be a set of infinite sequences in X, i.e., a subset of X^ω. For a sequence xs, we write $Suf(xs)$ to denote the set of its infinite suffixes. The sets $\Box P$ (always P), and $\Diamond P$ (sometime P) are defined by

$$\begin{aligned} xs \in \Box P &\equiv Suf(xs) \subseteq P\ , \\ xs \in \Diamond P &\equiv Suf(xs) \cap P \neq \emptyset\ . \end{aligned}$$

So, $xs \in \Box P$ means that all suffixes of xs belong to P, and $xs \in \Diamond P$ means that xs has some suffix that belongs to P.

For $U \subseteq X$, we define the subset $[\![U]\!]$ of X^ω to consist of the sequences whose first element is in U. For $A \subseteq X \times X$, we define the subset $[\![A]\!]_2$ of X^ω to consist of the sequences that start with an A-transition. So we have

$$\begin{aligned} xs \in [\![U]\!] &\equiv xs(0) \in U\ , \\ xs \in [\![A]\!]_2 &\equiv (xs(0), xs(1)) \in A\ . \end{aligned}$$

2.3. Stutterings and properties

A sequence ys is defined to be a *stuttering* of a sequence xs, notation $xs \preceq ys$, iff xs can be obtained from ys by replacing some (i.e., a possibly infinite number of) finite nonempty subsequences ss of consecutive equal elements of ys with their first elements $ss(0)$. For example, if, for a finite list vs, we write vs^ω to denote the sequence obtained by concatenating infinitely many copies of vs, the sequence $(aaabbbccb)^\omega$ is a stuttering of $(abbccb)^\omega$.

We use the following formal definition. A function $g : \mathbb{N} \to \mathbb{N}$ is called a *stutter function* iff it is monotonic and surjective. This easily implies that the composition of stutter functions is a stutter function. We use SF to denote the set of the stutter functions. It can be proved that a function g is a stutter function iff it satisfies $g(0) = 0$, and $g(i+1) - g(i) \in \{0,1\}$ for all indices i, and g is unbounded , i.e., for every number n there is an index i with $g(i) \geqslant n$.

The stuttering relation is now defined by $xs \preceq ys$ if and and only if $xs \circ g = ys$ for some stutter function g. Note that, indeed, even if all elements of xs differ, ys stutters when g stutters (i.e., is not injective).

An infinite sequence xs is called *stutterfree* iff it only stutters when terminated: if $xs(n+1) = xs(n)$ then $xs(i) = xs(n)$ for all $i \geqslant n$. Every infinite sequence ys can be "compressed" to a stutterfree infinite sequence xs with $xs \preceq ys$.

A subset P of X^ω is defined to be a *property* iff $xs \preceq ys$ implies that $xs \in P \equiv ys \in P$. This definition is equivalent to the one of Abadi and Lamport [1]. If P and Q are properties, the intersection $P \cap Q$, the complement $X^\omega \setminus P$, and $\Box P$ and $\Diamond P$ are properties. If U is a subset of X then $[\![U]\!]$ is a property. If A is a reflexive relation on X then $\Box[\![A]\!]_2$ is a property, and it consists of the sequences with all transitions belonging to A. It follows that $\Diamond[\![A]\!]_2$ is a property if A is irreflexive.

3. Specifications and simulations

In this section, we introduce the basic concepts of the theory. Following Abadi and Lamport [1], we define specifications in Section 3.1, refinement mappings in Section 3.2, and forward simulations in Section 3.3. In Section 3.4, we introduce stuttering forward simulations. Strict simulations and general simulations are presented in Section 3.5. In Section 3.6, we come to visible specifications and the associated concepts of observability and implementation, which again go back to Abadi and Lamport [1]. We then justify the definition of simulation by its relationship with the notion of implementation.

3.1. Specifications

A *specification* is defined to be a tuple $K = (X, X_0, N, P)$ where X is a set, X_0 is a subset of X, N a reflexive relation on X, and P is a property over X. The set X is called the *state space*, its elements are called *states*, the elements of X_0 are called *initial states*. Relation N is called the *next-state* relation. The set P is called the *supplementary* property by Abadi and Lamport [1].

We define an *initial execution* of K to be a sequence xs over X with $xs(0) \in X_0$ and such that every pair of consecutive elements belongs to N. A *behaviour* of K is an infinite initial execution xs of K with $xs \in P$. The requirements that relation N is reflexive and that set P is a property, are imposed to allow stuttering: if xs is a behaviour of K, any sequence ys obtained from xs by repeating elements of xs or by removing subsequent duplicates is also a behaviour of K. In particular, for every behaviour xs of K, there is a unique stutterfree behaviour xt of K with $xt \preceq xs$.

We write $Beh(K)$ to denote the set of behaviours of K. It is easy to see that $Beh(K) = [\![X_0]\!] \cap \Box[\![N]\!]_2 \cap P$. It follows that $Beh(K)$ is a property.

The components of specification $K = (X, X_0, N, P)$ are denoted $states(K) = X$, $start(K) = X_0$, $step(K) = N$ and $prop(K) = P$. A function on $states(K)$ is also called a state function on K. A numerical state function on K is a function $states(K) \to \mathbb{N}$.

3.2. Refinement mappings

Let K and L be specifications. A function $f : states(K) \to states(L)$ is called a *refinement mapping* (Abadi and Lamport [1]) from K to L iff $f(x) \in start(L)$ for every $x \in start(K)$, and $(f(x), f(x')) \in step(L)$ for every pair $(x, x') \in step(K)$, and $f \circ xs \in prop(L)$ for every $xs \in Beh(K)$. In this situation, we regard L as an abstract specification "implemented" by a concrete specification K. Here and henceforward, "implement" between quotes is used as an informal indication of a relationship between specifications.

Refinement mappings form the simplest way to compare different specifications. It is easy to see that the functional composition of two refinement mappings is a refinement mapping.

Example A. For $m > 1$, let $K(m)$ be the program

var j : $[0 \dots m) := 0$;
do *true* → j := (j + 1) **mod** m **od**;
prop: j changes infinitely often.

Here $[0 \dots m)$ stands for the set of integers i with $0 \leqslant i < m$. We use this program to denote the specification $K(m)$ with $states(K(m)) = [0 \dots m)$, $start(K(m)) = \{0\}$, $prop(K(m)) = \Box\Diamond[\![\neq]\!]_2$, and

$$(j, j') \in step(K(m)) \quad \equiv \quad j' \in \{j, (j+1) \bmod m\} .$$

Note that we omit the stuttering possibility from the program but include it in the next-state relation.

In order to give an example of a refinement mapping, we show that $K(15)$ "implements" $K(7)$. Let f be the function given by $f(j) = \min(j, 6)$. It is easy to verify that f is a refinement mapping from $K(15)$ to $K(7)$. Note that, whenever a concrete behaviour (in $K(15)$) is proceeding from 6 to 14, the corresponding abstract behaviour in $K(7)$ stutters. This example shows that it is useful that the next-state relation is always reflexive.

Let $K(\infty)$ be the program

var j : *Nat* := 0;
do *true* → j := j + 1 **od**;
prop: j becomes arbitrary large.

Function f is not a refinement mapping from $K(\infty)$ to $K(7)$ since it does not preserve behaviours. Function $g : \mathbb{N} \to [0 \dots 7)$ given by $g(j) = j$ **mod** 7 is a refinement mapping from $K(\infty)$ to $K(7)$.

A function $\mathbb{N} \to \mathbb{N}$ is a behaviour of $K(\infty)$ if and only if it is a stutter function. On the other hand, for an arbitrary specification L, a function $\mathbb{N} \to states(L)$ is a behaviour of L if and only if it is a refinement mapping from $K(\infty)$ to L. End Example A.

3.3. Forward simulations

It is well-known that functions, though useful, are often too specific to describe implementation relations. Instead of functions, one may have to use relations that satisfy certain conditions.

The easiest way to prove that one specification simulates (the behaviour of) another is by starting at the beginning and constructing the corresponding behaviour in the other specification inductively. This requires a

condition embodied in so-called forward or downward simulations [6, 18], which go back at least to Milner [21]. They are defined here as follows.

A relation F between $states(K)$ and $states(L)$ is called a *forward simulation* from specification K to specification L iff

(F0) For every $x \in start(K)$, there is $y \in start(L)$ with $(x, y) \in F$.
(F1) For every pair $(x, x') \in step(K)$ and every y with $(x, y) \in F$, there is y' with $(y, y') \in step(L)$ and $(x', y') \in F$.
(F2) Every initial execution ys of L with $(xs, ys) \in F^\omega$ for some $xs \in Beh(K)$ satisfies $ys \in prop(L)$.

If f is a refinement mapping from K to L, the graph of f is easily seen to be a forward simulation.

Example B. Let K be an arbitrary specification. We extend K with an additional component to count the number of non-stuttering steps taken in the behaviour. This is done by forming the specification H with $states(H) = states(K) \times \mathbb{N}$ and

$$
\begin{aligned}
&(x, i) \in start(H) \quad \equiv \quad x \in start(K) \ \wedge \ i = 0\,,\\
&((x, i), (y, j)) \in step(H) \quad \equiv\\
&\qquad (x, y) \in step(K) \quad \wedge \quad ((x = y \ \wedge \ i = j) \ \vee \ (x \neq y \wedge j = i + 1))\,,\\
&vs \in prop(H) \quad \equiv \quad fst \circ vs \in prop(K)\,.
\end{aligned}
$$

Such an additional component in the state is called a history variable. It is easy to verify that function fst is a refinement mapping from H to K and that the inverse relation $ivf = fst^{-1}$ is a forward simulation from K to H. End Example B.

3.4. Stuttering forward simulations

In order to allow the concrete specification to (occasionally) perform fewer steps than the abstract one, we weaken the concept of forward specification as follows.

A relation F between $states(K)$ and $states(L)$ is defined to be a *stuttering forward simulation* from K to L iff it satisfies the conditions (F0), (F2) and

(SF1) For every pair $(x, x') \in step(K)$, there is an integer state function vf on L such that, for every state $y \in states(L)$ with $(x, y) \in F$, there is a state $y' \in states(L)$ with $(y, y') \in step(L)$ such that $(x', y') \in F$, or $(x, y') \in F$ and $vf(y) \geqslant 0$ and $vf(y') < vf(y)$.

Example C. Coming back to the specifications of Example A, we now show that $K(7)$ can also be regarded as an "implementation" of $K(15)$. Indeed, the relation $F = \{(x, y) \mid x = f(y)\}$ is a stuttering forward simulation from $K(7)$ to $K(15)$. Condition (F0) is trivial since 0 is the only start state of both specifications and $0 = f(0)$. Condition (F2) is also trivial, since $(xs, ys) \in F^\omega$ implies that $xs(n) = f(ys(n))$ for all n; therefore $xs(n) \neq xs(n + 1))$ implies $ys(n) \neq ys(n + 1))$.

It remains to verify (SF1). We use (for every pair (x, x')) the integer state function vf on $K(15)$ given by $vf(y) = 15 - y$. Now let $(x, y) \in F$ and $(x, x') \in step(K(7))$. If $x \neq 6$ then $y = x < 6$ and we can take $y' = x'$. If $x' = x = 6$, we can take $y' = y$. It remains to consider the case that $x = 6$ and $x' = 0$. If $y < 14$, we can take $y' = y + 1$ so that $(x, y') \in F$ and $vf(y') < vf(y)$. Finally, if $y = 14$, we can take $y' = 0$. End Example C.

Condition (SF1) has some equivalent alternatives that look rather different:

Lemma 3.1. *Condition (SF1) is equivalent to either of the following equivalent conditions*:
(SF1h) For every pair $(x, x') \in step(K)$ and every state y of L with $(x, y) \in F$, there is a finite sequence of states $(y_0, \ldots, y_n)$ of L such that $y_0 = y$, and $(x', y_n) \in F$, and $(x, y_i) \in F$ and $(y_i, y_{i+1}) \in step(L)$ for all $i < n$.
(SF1s) For every sequence $xs \in \Box(step(K))$ and every state y of L with $(xs(0), y) \in F$, there is a sequence $ys \in \Box(step(L))$ and a stutter function h such that $y_0 = y$ and $(xs \circ h, ys) \in F^\omega$.

This Lemma is proved by cyclic implication (SF1)⇒(SF1s)⇒(SF1h)⇒(SF1). The proof is spelled out in PVS in Hesselink [10]. The equivalence between (SF1) and (SF1s) seems to be contained in Theorem 3 of Manolios [20].

One may replace the integers in (SF1) by a well-founded partial order as in Ladkin et al. [16, Fig. 21], Manolios [20], and Griffioen and Vaandrager [5]. This gives somewhat more flexibility, but it is also semantically equivalent.

Condition (SF1h) is less convenient for program verification than (SF1), but it is semantically simpler. It shows a clear analogy with the properties of branching bisimulations of van Glabbeek and Weijland [26]. Condition (SF1s) easily implies that (SF1) is preserved under relational composition.

Remark. Since it mentions executions and behaviours, condition (F2) is offensive to the ideals of assertional reasoning. In Hesselink [12,11], we therefore developed splitting simulations in which condition (F2) is combined with (F1) or (SF1) for specifications where the supplementary property is expressed in terms of weak and strong fairness properties only. This is useful for practical correctness proofs, but we do not claim semantic completeness.

3.5. Simulations

Forward simulations and stuttering forward simulations form a constructive way to relate the behaviours of the implementing specification with those of the abstract specification. Since this is not the only way, however, it is useful to give a nonconstructive characterization.

A relation F between the state spaces of specifications K and L is defined to be a *strict simulation* from K to L, notation $F : K \rightarrowtriangle L$, if for every $xs \in \mathit{Beh}(K)$ there exists $ys \in \mathit{Beh}(L)$ with $(xs, ys) \in F^\omega$.

A relation F between the state spaces of specifications K and L is defined to be a *simulation* from K to L, notation $F : K \rightarrow\!\!\!\!\triangleright\!\!\triangleright L$, if for every $xs \in \mathit{Beh}(K)$ there is a behaviour $ys \in \mathit{Beh}(L)$ and a stutter function h with $(xs \circ h, ys) \in F^\omega$.

Every strict simulation $K \rightarrowtriangle L$ is a simulation $K \rightarrow\!\!\!\!\triangleright\!\!\triangleright L$ since we can take for h the identity function. If $G \subseteq F$ and G is a [strict] simulation $K \rightarrow\!\!\!\!\triangleright\!\!\triangleright L$, then F is a [strict] simulation $K \rightarrow\!\!\!\!\triangleright\!\!\triangleright L$ since $G^\omega \subseteq F^\omega$. The next result justifies the terminology of Sections 3.3 and 3.4. It asserts the soundness of (stuttering) forward simulations.

Lemma 3.2. *Let K and L be specifications.*
(*a*) *Every forward simulation from K to L is a strict simulation $K \rightarrowtriangle L$.*
(*b*) *Every stuttering forward simulation from K to L is a simulation $K \rightarrow\!\!\!\!\triangleright\!\!\triangleright L$.*

We refer to Hesselink [9] for the proof of part (a). Part (b) easily follows from Lemma 3.1 using condition (SF1s).

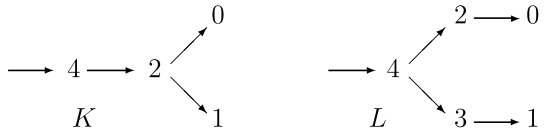
Because of this lemma, we are henceforth justified to write $F : K \rightarrowtriangle L$ when F is a forward simulation from K to L, or a refinement mapping from $\mathit{states}(K)$ to $\mathit{states}(L)$, and to write $F : K \rightarrow\!\!\!\!\triangleright\!\!\triangleright L$ when F is a stuttering forward simulation from K to L.

It is important that simulations and strict simulations are compositional. This is expressed by the next lemma, the proof of which is straightforward.

Lemma 3.3. *Let $F : K \rightarrow\!\!\!\!\triangleright\!\!\triangleright L$ and $G : L \rightarrow\!\!\!\!\triangleright\!\!\triangleright M$ be simulations. Then the relational composition $(F; G)$ is a simulation $K \rightarrow\!\!\!\!\triangleright\!\!\triangleright M$. If F and G are strict then $(F; G)$ is also strict.*

Note, however, that a composition of [stuttering] forward simulations need not be a [stuttering] forward simulation, since condition (F2) is not compositional. This alone might be a reason to introduce simulations.

Example D. The standard example of a simulation that is not a (stuttering) forward simulation uses specifications K and L, both with state space $X = \{0,1,2,3,4\}$, initial set $X_0 = \{4\}$, and property $\Diamond [\![\{0,1\}]\!]$. The next-state relations are given by the diagrams



in which the identity steps have been omitted. One can easily verify that relation $F = 1_X \cup \{(2,3)\}$ is a strict simulation $F : K \rightarrowtriangle L$. It is not a (stuttering) forward simulation since conditions (F1) and (SF1h) do not hold for $(x, x') = (2,0)$ and $y = 3$. End Example D.

3.6. Implementations

In order to justify the above definition of simulation, we relate it to the implementation concept of Abadi and Lamport [1], which is based on the idea of observable behaviours.

Recall from Abadi and Lamport [1] that a *visible specification* is defined to be a pair (K, f) where K is a specification and f is a state function on K, regarded as an observation function. The set of observations of (K, f) is defined by

$$Obs(K, f) \quad = \quad \{f \circ xs \mid xs \in Beh(K)\} .$$

Let (K, f) and (L, g) be visible specifications where f and g are functions to the same set. Then (K, f) is defined to be an *implementation* of (L, g) iff for every $zs \in Obs(K, f)$ there exists $zt \in Obs(L, g)$ with $zs \preceq zt$. This is the concept of implementation of Abadi and Lamport [1]. It is easy to prove that it is transitive.

Completely analogous to Theorem 2.6 of Hesselink [9], we have

Theorem 3.1. *Consider visible specifications (K, f) and (L, g) where f and g are functions to the same set. Then (K, f) is an implementation of (L, g) if and only if there is a simulation $F : K \rightarrowtriangle\!\!\!\!\rightarrow L$ with $(F; g) \subseteq f$.*

Proof. (if) Let $F : K \rightarrowtriangle\!\!\!\!\rightarrow L$ be a simulation with $(F; g) \subseteq f$. Let $zs \in Obs(K, f)$. We have to provide $zt \in Obs(L, g)$ with $zs \preceq zt$. By the definition of Obs, there is $xs \in Beh(K)$ with $zs = f \circ xs$. Since F is a simulation, there is $ys \in Beh(L)$ and a stutterfunction h with $(xs \circ h, ys) \in F^\omega$. Since $(F; g) \subseteq f$, we easily obtain $f \circ xs \circ h = g \circ ys$. This implies that $zs = f \circ xs \preceq f \circ xs \circ h \in Obs(L, g)$. We can thus choose $zt = f \circ xs \circ h$.

(only if) Now, let (K, f) be an implementation of (L, g). Just as in Hesselink [9], we use relation $F = \{(x, y) \mid f(x) = g(y)\}$, which satisfies $(F; g) \subseteq f$. It remains to prove that F is a simulation $K \rightarrowtriangle\!\!\!\!\rightarrow L$. Let $xs \in Beh(K)$. Since (K, f) is an implementation of (L, g), there is $ys \in Beh(L)$ with $f \circ xs \preceq g \circ ys$. There exists a stutter function h with $f \circ xs \circ h = g \circ ys$ and hence $(xs \circ h, ys) \in F^\omega$. This proves that F is a simulation $K \rightarrowtriangle\!\!\!\!\rightarrow L$. □

4. Extensions

The basic method of refinement calculus is to form a simulation $K \rightarrowtriangle\!\!\!\!\rightarrow M$ step by step, i.e., by constructing an intermediate specification L with simulations $K \rightarrowtriangle\!\!\!\!\rightarrow L$ and $L \rightarrowtriangle\!\!\!\!\rightarrow M$, followed by an application of Lemma 3.3. After having formed some simulation $K \rightarrowtriangle\!\!\!\!\rightarrow L$, it may happen however that the targetted simulation $K \rightarrowtriangle\!\!\!\!\rightarrow M$ does not factorize over L since some information from K is lost in L. Extensions are simulations $K \rightarrowtriangle\!\!\!\!\rightarrow L$ where this does not happen. This is expressed in the factorization lemma presented in Section 4.1.

In Section 4.2, we define (stuttering) history extensions, where the specification is extended with history variables or stuttering variables, cf. Abadi and Lamport [1], Ladkin et al. [16]. In the Sections 4.3 and 4.4, we treat invariant restrictions and eternity extensions introduced in Hesselink [8,9].

Section 4.5 contains an example of the use of these extensions to prove the correctness of a strict simulation. In Section 4.6, we treat a general simulation by means of a stuttering history extension. In Section 4.7 we define the clocking extension which plays a small but crucial role in the proof of semantic completeness.

4.1. Extensions and factorization lemma

An *extension* of specification K is defined to be a simulation $F : K \rightarrowtriangle\!\!\!\!\rightarrow L$ that is contained in the inverse of a refinement mapping $L \rightarrowtriangle K$. The extension F is called *strict* iff it is a strict simulation $F : K \rightarrowtriangle L$.

It easily follows that the relational composition of two [strict] extensions is a [strict] extension. If one wants to construct a simulation $K \rightarrowtriangle\!\!\!\!\rightarrow M$ and one has an extension $K \rightarrowtriangle\!\!\!\!\rightarrow L$, it suffices to construct a simulation $L \rightarrowtriangle\!\!\!\!\rightarrow M$. As shown by the next lemma, this is possible whenever a simulation $K \rightarrowtriangle\!\!\!\!\rightarrow M$ exists.

Lemma 4.1. *Let $F : K \rightarrowtriangle\!\!\!\!\rightarrow L$ be an extension. Let $G : K \rightarrowtriangle\!\!\!\!\rightarrow M$ be a simulation. Then there is a simulation $G' : L \rightarrowtriangle\!\!\!\!\rightarrow M$ with $(F; G') \subseteq G$. Moreover, if simulation G is strict, then G' can be chosen strict.*

Proof. Since F is an extension, we can choose a refinement mapping $f : L \twoheadrightarrow K$ with $F \subseteq f^{-1}$. Since f is a refinement mapping, it is a strict simulation. By compositionality, $G' = (f; G)$ is a simulation $L \twoheadrightarrow M$. Moreover, if G is strict, then G' is strict.

It remains to prove $(F; G') \subseteq G$. Since $G' = (f; G)$ and relational composition is associative and preserves inclusions, it suffices to prove that $(F; f)$ is contained in the identity relation. Well, if $(x, y) \in (F; f)$, there exists z with $(x, z) \in F$ and $f(z) = y$. Now $F \subseteq f^{-1}$ implies that $x = f(z) = y$. □

4.2. History extensions

We define a *history extension* to be a strict extension that is also a forward simulation. The standard method of extending a specification with a history variable always leads to a history extension. For example, the strict simulation $ivf : K \twoheadrightarrow H$ of Example B above is a history extension.

A stuttering forward simulation that is contained in the inverse of a refinement mapping is called a *stuttering history extension*. As far as we can see, this is the same as an extension with *stuttering variables* of Ladkin et al. [16, Fig. 21] (their weak fairness condition is needed to compensate for TLA's indifference to stuttering).

As we shall show by example in Section 4.6, a stuttering history extension means the introduction of some auxiliary variables and some auxiliary commands, while some guards of original commands are strengthened so that these commands can be temporarily disabled.

4.3. Invariants and invariant restrictions

The theory of invariants is most easily expressed in terms of subsets of the state space, but for programming purposes it is more convenient to work with predicates. To combine both points of view, we identify a predicate with the set of states where it holds.

Let $K = (X, X_0, N, P)$ be a specification. A state $x \in X$ is called *occurring* iff it is an element of a behaviour of K; it is called *reachable* iff it is an element of an initial execution. A subset D of X is called an *invariant* iff it contains all occurring states; it is called a *forward invariant* iff it contains all reachable states. Since every behaviour is an initial execution, every occurring state is reachable and every forward invariant is an invariant.

There are two principal ways to prove that a subset D of X is invariant. The standard way is by means of inductivity. Recall that a set D is called *inductive* (e.g., Manna and Pnueli [19]) iff it contains all start states and is preserved in every step:

$$\text{(ind)} \qquad X_0 \subseteq D \quad \wedge \quad (\forall\, x, y : (x, y) \in N \ \wedge\ x \in D \ \Rightarrow\ y \in D)\,.$$

Let us call D *backwards inductive* iff it occurs infinitely often in every behaviour and is preserved in the inverse of every step, as formalized in

$$\text{(b-ind)} \qquad \mathit{Beh}(K) \subseteq \Box\Diamond[\![\, D \,]\!] \quad \wedge \quad (\forall\, x, y : (x, y) \in N \ \wedge\ y \in D \ \Rightarrow\ x \in D)\,.$$

If D is inductive, it contains all reachable elements, and is therefore a forward invariant and an invariant. If D is backwards inductive, every state in a behaviour xs is eventually followed in xs by a state of D. Using induction backward along xs, it follows that every element of xs belongs to D. Therefore D is an invariant. This proves that all inductive and all backwards inductive subsets of X are invariants. The first point is of course well known. Below in 4.5, we give an example of a backwards inductive set.

If D is a subset of X, the subspace restriction K_D of K is defined as the tuple $(D, X_0 \cap D, N \cap D^2, P \cap D^\omega)$. It is easy to verify that K_D is a specification and that the identity function 1_D is a refinement mapping from K_D to K. It is also easy to verify that the inverse relation 1_D is a strict simulation $K \twoheadrightarrow K_D$ if and only if D is an invariant. In that case, the strict simulation $K \twoheadrightarrow K_D$ is a strict extension, technically speaking, but we prefer to use the term "invariant restriction".

4.4. Eternity extensions

Let K be a specification. Let M be an arbitrary set, to be called the *eternity type*. A relation R between $states(K)$ and M is called a *behaviour restriction* of K at M iff, for every behaviour xs of K, there exists an $m \in M$ with $xs(\mathbb{N}) \subseteq R^{-1}[m]$ (see Sections 2.1 and 2.2 for the definitions):

$$\text{(BR)} \qquad xs \in Beh(K) \quad \Rightarrow \quad (\exists\, m : xs(\mathbb{N}) \subseteq R^{-1}[m])\ .$$

Note that, for given values of xs and m, the set $R^{-1}[m]$ serves as a kind of invariant: all states of the behaviour belong to it.

If R is a behaviour restriction of K at M, we define the *eternity extension* $W = et(K,R)$ as the specification W given by

$$\begin{array}{l} states(W) \quad = \quad R\ , \\ start(W) \quad = \quad R \cap (\ alphaK \times M)\ , \\ ((x,m),(x',m')) \in step(W) \quad \equiv \\ \qquad (x,x') \in step(K) \quad \wedge \quad m = m'\ , \\ ys \in prop(W) \quad \equiv \quad fst \circ ys \in prop(K)\ . \end{array}$$

It is clear that $step(W)$ is reflexive and that $prop(W)$ is a property. Therefore W is a specification. The component $m \in M$ is called an eternity variable since it does not change during the entire behaviour.

It is easy to verify that $fst : states(W) \to states(K)$ is a refinement mapping. Let relation ivf between $states(K)$ and $states(W)$ be defined as the inverse of fst. We now justify the term "eternity extension" and prove soundness (see Hesselink [9]):

Theorem 4.1. *Let R be a behaviour restriction of K at M. Then relation ivf is a strict extension $K \mathrel{-\!\!\triangleright} W$.*

Proof. We first prove that ivf is a strict simulation. Let $xs \in Beh(K)$. We have to construct $ys \in Beh(W)$ with $(xs, ys) \in ivf^{\omega}$. By (BR), we can choose m with $xs(\mathbb{N}) \subseteq (R^{-1}[m])$. Define $ys(i) = (xs(i), m)$. A trivial verification shows that the sequence ys constructed in this way is a behaviour of W with $(xs, ys) \in ivf^{\omega}$. This proves that ivf is a strict simulation. Since ivf is contained in the inverse of the refinement mapping fst, it is a strict extension. □

The strict simulation $ivf : K \mathrel{-\!\!\triangleright} et(K,R)$ of Theorem 4.1 is called the *eternity extension* of K corresponding to behaviour restriction R.

In general, condition (BR) is a heavy proof obligation. It requires to invent a relation R and then, for every behaviour, it requires to invent an element m. In our practice Hesselink [8,9,13], m is always some kind of limit of some abstraction of the states in xs, and relation R expresses this fact. Moreover, in order to find a suitable eternity extension, we almost always have to first form a history extension so that the states hold enough information to form limits, usually infinite sequences approximated by initial segments. We come back to this in Section 7. One may note, that once adequate R and m have been found, the verification of $xs(\mathbb{N}) \subseteq R^{-1}[m]$ is completely analogous to the verification of an invariant.

4.5. Example of strict extensions

In this section, we present an example to show how the correctness of a simulation can be proved by means of a history extension, an eternity extension, an invariant restriction, and a refinement mapping. The example is chosen as easy as possible and is therefore clearly unrealistic. The example is only conceptual, not intended to represent a methodology.

Let the abstract specification L be given by the program

$$\begin{array}{l} \textbf{var}\ t : Nat := 0\ ,\ p : Nat\,; \\ \textbf{do}\ \ t = p \quad \to \quad t := 0\ ;\ p := \textbf{some}\ p'\,; \\ []\quad t < p \quad \to \quad t := \textbf{some}\ t'\ \textbf{with}\ t < t' \leqslant p\,; \\ \textbf{od}\,; \\ \textbf{prop:}\ \ true\ . \end{array}$$

We thus have $states(L) = \mathbb{N} \times \mathbb{N}$ and $start(L) = \{0\} \times \mathbb{N}$ and

$$((t,p),(t',p')) \in step(L) \quad \equiv$$
$$(t = p \ \wedge \ t' = 0) \ \vee \ (t < t' \leqslant p = p') \ \vee \ (t = t' \ \wedge \ p = p') \, .$$

There is no supplementary property (other than *true*).

We propose to implement L by specification K given by

var $i : Nat := 0$;
do $true \quad \rightarrow \quad i := i + 1 \quad [] \quad true \quad \rightarrow \quad i := 0$ **od**;
prop: $i = 0$ infinitely often.

It is clear that variable i of specification K plays a similar role as t in L. For a given value of i, the choice of p in L is open. We therefore define relation F between the state spaces of K and L by

$$(i,(t,p)) \in F \quad \equiv \quad i = t \, .$$

We prove that F is a strict simulation $K \mathrel{-\!\!\triangleright} L$. This is done by means of standard extensions and a refinement mapping. The choice of p in the first alternative of L takes place when t is reset to 0, and determines the next value where t will jump back. From the point of view of K, variable p serves as a prophecy of the next jumping value. The choice of p has infinite nondeterminacy. Since prophecy variables with infinite nondeterminacy are unsound, we use an eternity variable to determine the jumping values. The value of this eternity variable is to be a list that contains all jumping values.

We first introduce a history variable *ps* to approximate this eternity variable. Variable *ps* holds a finite list of numbers, initially the empty list ε. We thus extend K to a specification K_1 given by

var $i : Nat := 0 \ ; \quad ps : Nat^* := \varepsilon$;
do $true \quad \rightarrow \quad i := i + 1$;
[] $true \quad \rightarrow \quad add(ps, i) \ ; \quad i := 0$;
od ;
prop: $i = 0$ infinitely often.

Here, *add* is the operation to extend the first argument, a finite list, with the second argument, a number. It is clear that the projection function *fst* is a refinement mapping $K_1 \mathrel{-\!\!\triangleright} K$. It is easy to verify that the inverse relation *ivf* is a forward simulation and hence a history extension $K \mathrel{-\!\!\triangleright} K_1$.

We extend K_1 with an eternity variable *ms* of type $\mathbb{N}^\omega$ with the behaviour restriction *Pref* that *ps* is always a prefix of *ms* (notation $ps \sqsubseteq ms$):

$$((i,ps),ms) \in Pref \quad \equiv \quad ps \sqsubseteq ms \, .$$

We need to prove condition (BR) that, for every behaviour *xs* of K_1, the eternity variable *ms* has some value that always satisfies behaviour restriction *Pref*. Indeed, in every behaviour *xs* of K_1, the finite list *ps* only grows at its end. It therefore approximates some finite or infinite list, say *qs*, which depends on *xs*. If *qs* is infinite, we take $ms = qs$. Otherwise, we extend *qs* to an infinite list *ms* in an arbitrary way. All values of *ps* in behaviour *xs* are prefixes of *qs* and hence of *ms*. This proves (BR).

Since (BR) holds, we can form specification $K_2 = et(K_1, Pref)$ with its eternity extension $K_1 \mathrel{-\!\!\triangleright} K_2$. The program of K_2 looks very much like the program of K_1, but has an additional program variable *ms* with an arbitrary initial value, which is never modified. Moreover, the states of K_2 are subject to restriction *Pref*.

Since *ps* is always a prefix of *ms*, the number $ms(\#ps)$, where $\#ps$ is the length of *ps*, is a prophecy of the next number to be added to *ps*. We therefore define function $f : states(K_2) \rightarrow states(L)$ by $f(i,ps,ms) = (i, ms(\#ps))$. Unfortunately, f is not a refinement mapping $K_2 \mathrel{-\!\!\triangleright} L$, since K_2 can always do steps that increment i. We can however eliminate these steps since they do not occur in behaviours. This is done by means of an invariant restriction.

We claim that $D : i \leqslant ms(\#ps)$ is an invariant of K_2. This is proved by verifying that D is backwards inductive. First, $Beh(K_2) \subseteq \Box\Diamond[\![D]\!]$ holds since the supplementary property says that $i = 0$ holds infinitely often. Second, for a transition where i is added to ps and jumps back to 0, the behaviour restriction $Pref$ in the post-state implies that the pre-state satisfies $i = ms(\#ps)$ and hence D. Third, all other steps only increment i, so that D in the post-state implies D in the pre-state.

Let $K_3 = (K_2)_D$ be the subspace restriction with respect to subset D. Since D is an invariant, we have a strict extension $K_2 \rightarrowtriangle K_3$. By composition this yields a strict extension $K \rightarrowtriangle K_3$. It is straightforward to verify that f restricted to D is a refinement mapping $K_3 \rightarrowtriangle L$ and that the relational composition of the strict simulation $K \rightarrowtriangle K_3$ and (the graph of) function f is a subset of F. Since a composition of strict simulations is a strict simulation, it follows that F is a strict simulation.

4.6. Example of a stuttering history extension

We take specification K_1 of Section 4.5 as our abstract specification L:

var $i : Nat := 0$; $ps : Nat^* := \varepsilon$;
do $true \;\rightarrow\; i := i + 1$;
[] $true \;\rightarrow\; add(ps, i)$; $i := 0$;
od;
prop: $i = 0$ infinitely often.

We propose to implement L by specification K given by

var $t : Nat$; $ts : Nat^* := \varepsilon$;
do $true \;\rightarrow\; add(ts, t)$; $t :=$ **some** t' **od**;
prop: ts changes infinitely often.

Note that variable t is initialized nondeterministically. The idea is that ts plays the role of ps, but K needs fewer steps than L to fill its list. We define relation F between the state spaces of K and L by

$$((t, ts), (i, ps)) \in F \quad \equiv \quad ts = ps\,.$$

In order to show that F is a simulation $K \rightarrowtriangle\!\!\rightarrowtriangle L$, we extend K with an auxiliary variable i that in small steps climbs from 0 to t. We thus form the specification K_1 given by

var $t : Nat$; $ts : Nat^* := \varepsilon$; $i : Nat := 0$;
do $t \leqslant i \;\rightarrow\; add(ts, t)$; $t :=$ **some** t' ; $i := 0$;
[] $i < t \;\rightarrow\; i := i + 1$;
od;
prop: ts changes infinitely often.

We claim that the natural relation id_{12} between the state spaces of K and K_1 is a stuttering history extension $K \rightarrowtriangle\!\!\rightarrowtriangle K_1$. Indeed, the inverse relation is the function that forgets i, which is easily seen to be a refinement mapping. Also, id_{12} satisfies conditions (F0) and (F2) of 3.3. We define the state function vf on K_1 by $vf(t, ts, i) = i - t$. Condition (SF1) holds since, for every state (t, ts, i) of K_1 with $t \leqslant i$, the step of K is mimicked by the corresponding step of K_1, whereas otherwise $vf > 0$ and vf can be decremented by incrementing i and this does not change the projection on K.

Let $f : states(K_1) \rightarrow states(L)$ be given by $f(t, ts, i) = (i, ts)$. Just as in 4.5, function f is not yet a refinement mapping since, in this case, the steps with $t < i$ are not treated correctly. The remedy is to use the subset $D : i \leqslant t$ of $states(K_1)$. It is easy to see that D is inductive and hence an invariant. We therefore form the invariant restriction $1_D : K_1 \rightarrowtriangle K_2 = (K_1)_D$. Indeed, the restriction of f to D is a refinement mapping $K_2 \rightarrowtriangle L$. By Lemma 3.3, the composition $(id_{12}; 1_D; f)$ is a simulation. Since F contains this composition, F is also a simulation.

4.7. The clocking extension

The completeness proofs of Abadi and Lamport [1], Lynch and Vaandrager [18], Hesselink [9] rely on the unfolding of specification K, which is an extension of K with the complete history. In our setting, this is more than is necessary. Moreover, the unfolding is only strong enough to deal with strict simulations that "preserve quiescence". In this paper, we show that we only need to record the number of steps taken and that preservation of quiescence can be eliminated if we force this number of steps to increase indefinitely. For this purpose, we introduce the clocking extension, which is a minor variation of Example B in Section 3.3.

Let K be an arbitrary specification. We augment K with an integer variable that is incremented with 1 in every nontrivial step, and also infinitely often. Formally, let $W = \mathit{cl}(K)$ be the specification defined by

$$\begin{aligned}
&\mathit{states}(W) = \mathit{states}(K) \times \mathbb{N}\,,\\
&\mathit{start}(W) = \mathit{start}(K) \times \{0\}\,,\\
&((x,i),(y,j)) \in \mathit{step}(W) \quad \equiv\\
&\quad (x,y) \in \mathit{step}(K) \quad \wedge \quad (j = i+1 \ \vee \ (x = y \ \wedge \ i = j))\,,\\
&ys \in \mathit{prop}(W) \quad \equiv \quad \mathit{fst} \circ ys \in \mathit{prop}(K) \ \wedge \ (\forall\, n : \exists\, i : \mathit{snd}(ys(i)) \geqslant n)\,.
\end{aligned}$$

It is easy to verify that $\mathit{step}(W)$ is reflexive and that $\mathit{prop}(W)$ is a property. So, indeed, W is a specification. Just as in Example B, the function fst is a refinement mapping $W \to K$. Its inverse relation $\mathit{ivf} = \mathit{fst}^{-1}$ is a strict simulation $K \rightarrowtriangle W$ since, for every behaviour xs of K, the sequence $ys = \lambda i : (xs(i), i)$ is a behaviour of W with $(xs, ys) \in \mathit{ivf}^{\omega}$. This proves that $\mathit{ivf} : K \rightarrowtriangle \mathit{cl}(K)$ is a strict extension. It is called the *clocking extension* of K.

If K has a behaviour that terminates, relation ivf is no forward simulation since condition (F2) fails. Indeed, every behaviour ys of $\mathit{cl}(K)$ is nonterminating.

5. Semantic completeness for strict simulations

We prove semantic completeness by constructing what may be regarded as a universal eternity extension of a specification.

In Section 5.1, we define "logical clocks" and "origin functions", and show that these functions are useful to construct a refinement mapping. In Section 5.2, we define several properties of specifications and show how some of these are used to construct a logical clock and an origin function. In Section 5.3, we show that, for a specification K that satisfies the conjunction of five properties, every strict simulation $K \rightarrowtriangle L$ can be replaced by a refinement mapping. This conjunction is established in section 5.4 by means of an eternity extension. In Section 5.5, we show how to eliminate the Axiom of Choice from the proof of semantic completeness.

5.1. Clocks, origin functions, and never-termination

We define a *logical clock* on specification K to be a numerical state function c that satisfies

$$\begin{aligned}
&z \in \mathit{start}(K) \quad \Rightarrow \quad c(z) = 0\,,\\
&(x,y) \in \mathit{step}(K) \ \wedge \ x \neq y \quad \Rightarrow \quad c(y) = c(x) + 1\,.
\end{aligned}$$

Not every specification K allows a logical clock. For example, the specifications $K(m)$ with $m < \infty$ of Example A in Section 3.2 have no logical clocks. For any K, the clocking extension $\mathit{cl}(K)$ has the logical clock snd. The importance of logical clocks stems from the following result.

Lemma 5.1. *Let K be a specification with a logical clock c. Let xs be a nonterminating initial execution of K.*
(a) If xs is stutterfree, then $c \circ xs$ is the identity function $1_{\mathbb{N}}$.
(b) In any case, $c \circ xs$ is a stutter function.

Proof. (a) Since xs is nonterminating and stutterfree, subsequent elements of xs are always different. We then use that xs is an initial execution and that c is a logical clock to prove by induction that $c(xs(i)) = i$ for all i.

(b) Let ys be the stutterfree sequence with $ys \preceq xs$. Then ys is a nonterminating stutterfree initial execution. Part (a) therefore implies that $c \circ ys = 1_{\mathbb{N}}$. Let g be a stutter function with $xs = ys \circ g$. Then $c \circ xs = g$. □

A function $q : \mathit{states}(K) \to \mathit{start}(K)$ is called an *origin function* iff $q(x) = x$ for every $x \in \mathit{start}(K)$ and $q(x) = q(y)$ for all pairs $(x, y) \in \mathit{step}(K)$. Specification K is defined to be *never-terminating* iff all its behaviours are nonterminating.

The next result shows how a logical clock and an origin function can be used to construct a refinement mapping from a never-terminating specification.

Lemma 5.2. *Let K and L be specifications. Assume that K is never-terminating and has a logical clock c and an origin function q. Let $h : \mathit{start}(K) \to \mathit{Beh}(L)$ be a given function. Then function f given by $f(x) = h(q(x))(c(x))$ is a refinement mapping $K \twoheadrightarrow L$.*

Proof. We verify the three conditions for refinement mappings. First, let $x \in \mathit{start}(K)$. Then $c(x) = 0$ and $q(x) = x$. Therefore, $f(x) = h(x)(0) \in \mathit{start}(L)$, since $h(x) \in \mathit{Beh}(L)$.

Second, let $(x, y) \in \mathit{step}(K)$ with $x \neq y$. Let $z = q(x)$. Then $f(x) = h(z)(c(x))$. We also have $z = q(y)$ and $c(y) = c(x) + 1$ and hence $f(y) = h(z)(c(x) + 1)$. Since $h(z)$ is a behaviour of L, this implies that $(f(x), f(y)) = (h(z)(c(x)), h(z)(c(x) + 1)) \in \mathit{step}(L)$.

Third, let xs be a behaviour of K. Then $z = xs(0)$ satisfies $z = q(xs(i))$ for all i. Therefore, $f \circ xs = h(z) \circ c \circ xs$. Since K is never-terminating, xs does not terminate. By Lemma 5.1, $c \circ xs$ is a stutter function. It follows that $h(z) \preceq f \circ xs$. Since $h(z)$ is a behaviour of L, this proves that $f \circ xs$ is a behaviour of L. □

5.2. Determinacy conditions

A specification K is called *deterministic* iff the next-state relation is functional in the sense that, for all states x, y, z,

$$(x, y) \in \mathit{step}(K) \ \wedge \ (x, z) \in \mathit{step}(K) \ \wedge \ y \neq x \neq z \quad \Rightarrow \quad y = z \, .$$

Specification K is called *backwards-deterministic* iff the inverse of the next-state relation is functional in the sense that, for all states x, y, z,

$$(x, z) \in \mathit{step}(K) \ \wedge \ (y, z) \in \mathit{step}(K) \ \wedge \ x \neq z \neq y \quad \Rightarrow \quad x = y \, .$$

Specification K is called *fresh* iff a nontrivial step never ends in a start state, i.e., iff for all states x and y,

$$(x, y) \in \mathit{step}(K) \ \wedge \ y \in \mathit{start}(K) \quad \Rightarrow \quad x = y \, .$$

Specification K is called *full* iff every state of K occurs in some behaviour of K.

Specification K is called *bi-deterministic* iff it is never-terminating, deterministic, backwards-deterministic, fresh, and full (this is clearly a very strong condition).

Lemma 5.3. *Let K be a fresh, full, and backwards-deterministic specification. Then K has a logical clock and an origin function.*

Proof. Since K is fresh and backwards-deterministic, there is a unique "predecessor" function $p : \mathit{states}(K) \to \mathit{states}(K)$ with

$$\begin{aligned} &(x, y) \in \mathit{step}(K) \quad \equiv \quad x = p(y) \ \vee \ x = y \, , \\ &x \in \mathit{start}(K) \quad \equiv \quad p(x) = x \, . \end{aligned}$$

Since K is full, every state x occurs in a behaviour xs, say $x = xs(n)$, and we may assume that $xs(i) \neq xs(i + 1)$ for all $i < n$. It then follows that n is the smallest number with $p^n(x) \in \mathit{start}(K)$, where p^n refers to repeated application of function p. We can therefore define the state functions c and q by $c(x) = n$ and $q(x) = p^n(x)$ where n is the smallest number with $p^n(x) \in \mathit{start}(K)$. It is easy to verify that c is a logical clock and that q is an origin function. □

5.3. Bi-deterministic specifications

If specification K is deterministic, there is a unique "successor" function s with, for all x and y,

$$(x, y) \in \mathit{step}(K) \quad \equiv \quad y = s(x) \ \vee \ y = x \ .$$

For any state x, let $s^*(x)$ stand for the sequence xs with $xs(n) = s^n(x)$. It is easy to see that $s^*(x)$ is an initial execution for every $x \in \mathit{start}(K)$.

Lemma 5.4. *Let K be a bi-deterministic specification with successor function s, logical clock c, and origin function q.*
(a) If xs is a stutterfree behaviour, then $xs = s^*(z)$ for $z = xs(0) \in \mathit{start}(K)$.
(b) For every state x, we have $s^{c(x)}(q(x)) = x$.
(c) For every $x \in \mathit{start}(K)$, $s^*(x)$ is a stutterfree behaviour.

Proof. (a) Let xs be a stutterfree behaviour. Since K is never-terminating, Lemma 5.1 implies that $c(xs(i)) = i$ for all i. Since K is deterministic, it follows that $xs(i+1) = s(xs(i))$ for all i, and hence that $xs(i) = s^i(xs(0))$ for all i. This proves $xs = s^*(xs(0))$. Since xs is a behaviour, $xs(0) \in \mathit{start}(K)$.

(b) Let a state x be given. Since K is full, state x occurs in a behaviour and hence in a stutterfree behaviour, say $x = xs(n)$ for some index n and some stutterfree behaviour xs. Since q is an origin function, we have $q(xs(i)) = xs(0)$ for all i. Therefore $x = s^n(xs(0)) = s^{c(x)}(q(x))$.

(c) We proceed with the situation in the proof of (b). Now $n = c(x) = 0$, so that $q(x) = x$. Now use part (a). □

Theorem 5.1. *Let specification K be bi-deterministic and let $F : K \rightarrowtriangle L$ be a strict simulation. Then there is a refinement mapping $f : K \rightarrowtriangle L$ with $f \subseteq F$.*

Proof. Let s be the successor function of K. By Lemma 5.3, K has a logical clock c and an origin function q. Since F is a strict simulation, every behaviour xs of K has a behaviour ys of L such that $(xs, ys) \in F^\omega$. This applies in particular for the stutterfree behaviours of K, as analysed in Lemma 5.4(c). By the Axiom of Choice, there exists a function $\varepsilon : \mathit{start}(K) \to \mathit{Beh}(L)$ such that $(s^*(z), \varepsilon(z)) \in F^\omega$ for all start states z of K.

We use Lemma 5.2 to define the refinement mapping $f : K \rightarrowtriangle L$ by $f(x) = \varepsilon(q(x))(c(x))$. We prove that $f \subseteq F$ by observing, for every $x \in \mathit{states}(K)$, that

$$\begin{array}{ll} & (x, f(x)) \in F \\ \equiv & \quad \{ \text{ let } z = q(x), \text{ Lemma 5.4(b), definition of } f \ \} \\ & (s^*(z)(c(x)), \varepsilon(z)(c(x))) \in F \\ \Leftarrow & \quad \{ \text{ definition } F^\omega \text{ in 2.2 } \} \\ & (s^*(z), \varepsilon(z)) \in F^\omega \\ \equiv & \quad \{ \text{ choice of } \varepsilon \ \} \\ & \mathit{true}. \qquad \square \end{array}$$

Most specifications are not bi-deterministic. Theorem 5.1 is therefore seldom directly applicable. The next result turns out to be more useful.

Theorem 5.2. *Let $e : K \rightarrowtriangle E$ be an extension of specifications, and assume that E is bi-deterministic. For any strict simulation $F : K \rightarrowtriangle L$, there exists a refinement mapping $f : E \rightarrowtriangle L$ with $(e; f) \subseteq F$.*

Proof. By Lemma 4.1, there is a strict simulation $G : E \rightarrowtriangle L$ with $(e; G) \subseteq F$. Since E is bi-deterministic, Theorem 5.1 implies that G contains a refinement mapping $f : E \rightarrowtriangle L$. Then $(e; f) \subseteq F$. □

The condition of bi-determinism in Theorem 5.1 is not only sufficient for the existence of g, but also necessary, in the sense that K is bi-deterministic if, for every specification L, every strict simulation $K \rightarrowtriangle L$ contains a refinement mapping. This is shown in Corollary 5.2 below. For the moment, we only prove a very special case of it.

Lemma 5.5. *Let $F : K \rightarrowtriangle E$ be a strict extension of specifications, and assume that E is bi-deterministic. Assume that F contains a refinement mapping. Then K is bi-deterministic.*

Proof. We verify the five constituents of bi-determinism. Let $f : K \rightarrowtriangle E$ be a refinement mapping contained in F. Since f is a refinement mapping, a terminating behaviour of K would yield a terminating behaviour of E, contradicting that E is never-terminating. Therefore, K is never-terminating.

Since F is a strict extension, it is contained in the inverse of a refinement mapping $g : E \rightarrowtriangle K$. Consequently, f is contained in the inverse of g. Since f and g are functions, it follows that $g \circ f$ is the identity function of $\mathit{states}(K)$. In particular, function f is injective. It is easy to verify that, since f is an injective refinement mapping, and E is deterministic, and backwards-deterministic, and fresh, K is also deterministic, and backwards-deterministic, and fresh.

It remains to prove that K is full. Let x be a state of K. Since E is full, E has a behaviour ys that contains $f(x)$. Therefore $g \circ ys$ is a behaviour of K that contains $g(f(x)) = x$. □

5.4. The universal eternity extension

In order to use Theorem 5.2, we have to construct a strict extension towards a bi-deterministic specification. For this purpose, we use the eternity variables of Section 4.4. Formally, the most natural way to construct a behaviour restriction is to use for M the set of behaviours and to choose the behaviour restriction R in a convenient way that guarantees $xs(\mathbb{N}) \subseteq R^{-1}[xs]$. Since the set $xs(\mathbb{N})$ only depends on the occurring states and not on their multiplicities, we restrict the attention to the stutterfree behaviours. We thus take $M = \mathit{SBeh}(K)$ where $\mathit{SBeh}(K)$ stands for the set of stutterfree behaviours of specification K. We define the relation R between $\mathit{states}(K)$ and $\mathit{SBeh}(K)$ by

$$(x, zs) \in R \quad \equiv \quad x \in zs(\mathbb{N}) \,.$$

Since, for every behaviour xs, there is a stutterfree behaviour zs with $zs \preceq xs$ and hence $zs(\mathbb{N}) = xs(\mathbb{N})$, every behaviour xs has an element $zs \in \mathit{SBeh}(K)$ with $xs(\mathbb{N}) \subseteq zs(\mathbb{N})$. This implies that R is a behaviour restriction.

We define the universal eternity extension $\mathit{UEt}(K) = \mathit{et}(K, R)$. Relation ivf between $\mathit{states}(K)$ and $\mathit{states}(\mathit{UEt}(K))$ consists of the pairs $(x, (y, zs))$ with $x = y \in zs(\mathbb{N})$. By Theorem 4.1, we now have the extension $\mathit{ivf} : K \rightarrowtriangle \mathit{UEt}(K)$.

Theorem 5.3. *Assume that K is never-terminating and has a logical clock. Then specification $\mathit{UEt}(K)$ is bi-deterministic.*

Proof. We abbreviate $\mathit{UEt}(K)$ to W. Let c be a logical clock of K. We prove that W is bi-deterministic by verifying the five defining conditions.

Let w be a state of W. Then $w = (x, xs)$ for some stutterfree behaviour xs of K, say with $x = xs(n)$. Since K is never-terminating and c is a logical clock of K, we have $c(xs(i)) = i$ for all i, and in particular $c(x) = n$. It follows that the only nontrivial step in W from w goes to the pair $(xs(n+1), xs)$, and that the only nontrivial step towards w comes from $(xs(n-1), xs)$. Moreover, the latter is only available if $w \notin \mathit{start}(W)$. This shows that W is deterministic, backwards-deterministic and fresh. W is full since w occurs in the behaviour $\lambda i : (xs(i), xs)$. Since $\mathit{fst} : W \rightarrowtriangle K$ is a refinement mapping and K is never-terminating, W is never-terminating. □

Given an arbitrary specification K, we now apply Lemma 5.3 to the clocking extension $\mathit{cl}(K)$ of Section 4.7, which by construction is never-terminating and has the projection snd as a logical clock. Since the composition of strict extensions is a strict extension, this yields the next result.

Theorem 5.4. *Let K be a specification. There is a bi-deterministic specification E with a strict extension $\mathsf{eb} : K \rightarrowtriangle E$ which is a composition of the clocking extension $K \rightarrowtriangle \mathsf{cl}(K)$ and an eternity extension $\mathsf{cl}(K) \rightarrowtriangle E$.*

Corollary 5.1. *Let K be a specification. Let $\mathsf{eb} : K \rightarrowtriangle E$ be as in Theorem 5.4. Let $F : K \rightarrowtriangle L$ be a strict simulation. Then Theorem 5.2 yields the existence of a refinement mapping $f : E \rightarrowtriangle L$ with $(\mathsf{eb}; f) \subseteq F$.*

In Hesselink [9], we only gave a direct proof of a weak version of Corollary 5.1. Here, we have separated the most difficult part of the proof in Theorem 5.1, while the details of the eternity extension are deferred to Theorem 5.3. Also, the clocking extension used in Theorem 5.4 is much simpler than the unfolding used previously.

Is is now easy to prove that the condition of bi-determinism in Theorem 5.1 is necessary.

Corollary 5.2. *Let K be a specification such that, for every specification L, every strict simulation $F : K \rightarrowtriangle L$ contains a refinement mapping. Then K is bi-deterministic.*

Proof. By Theorem 5.4, there is a bi-deterministic specification E with a strict extension $eb : K \rightarrowtriangle E$. The assumption implies that eb contains a refinement mapping. Therefore, Lemma 5.5 implies that K is bi-deterministic. □

5.5. Elimination of the axiom of choice

The proof of Theorem 5.1 uses the Axiom of Choice. The reader who does not want to rely on this axiom has the following alternative. Define a relation F between the state spaces of specifications K and L to be a *constructive strict simulation* iff there exists a function $\gamma : Beh(K) \to Beh(L)$ such that $(xs, \gamma(xs)) \in F^\omega$ for every behaviour xs of K. It is clear that a constructive strict simulation is a strict simulation, and that conversely the Axiom of Choice implies that every strict simulation is a constructive strict simulation.

The Axiom of Choice is now eliminated from the proof of Theorem 5.1 by requiring that F is a constructive strict simulation. Of course, the strengthened assumption also affects the descendant results: we need to assume that F is a constructive strict simulation in Theorem 5.2 and Corollary 5.1.

6. Stuttering universality

In this section, we prove the completeness result that, if $F : K \rightarrowtriangle E$ is a strict extension towards a bi-deterministic specification E, an arbitrary simulation $K \rightarrowtriangle\!\!\!\rightarrow L$ can be factorized over F, followed by a stuttering history extension $E \rightarrowtriangle\!\!\!\rightarrow T$, and a refinement mapping $T \rightarrowtriangle L$.

We first prepare the ground by a mathematical analysis of stuttering in Section 6.1. In Section 6.2 we introduce temporization, a general construction of stuttering history extensions. The completeness results are treated in Section 6.3.

6.1. Deeper into Stuttering

As a preparation for some of the technicalities in the next sections, we now develop three different views of stuttering. We represent the stuttering relation by means of functions in $\mathbb{N} \to \mathbb{N}$, or in the set SF of the stutter functions, or in the set $Inc0$ of the increasing functions $f : \mathbb{N} \to \mathbb{N}$ with $f(0) = 0$ (increasing means that $f(i) < f(i+1)$ for all $i \in \mathbb{N}$).

We introduce operators $\#$ and $\bullet$ to translate between the three domains. Both operators are denoted as postfix operators, either above or below the line, i.e., we construct a pair of inverse functions $(\_)^\# : (\mathbb{N} \to \mathbb{N}) \to Inc0$ and $(\_)_\# : Inc0 \to (\mathbb{N} \to \mathbb{N})$, and a pair of inverse functions $(\_)^\bullet : Inc0 \to SF$, and $(\_)_\bullet : SF \to Inc0$.

The simplest view consists of the set of all functions $d : \mathbb{N} \to \mathbb{N}$, regarded as delay specifiers where the i-th genuine step is preceeded by $d(i)$ stuttering steps. The second view is obtained from the first view by accumulating all steps of the first view into

$$d^\#(k) = \big(\sum i : i < k : d(i) + 1\big) .$$

Function $d^\#$ clearly belongs to the set $Inc0$. Conversely, if $f \in Inc0$, then $f_\#$ given by $f_\#(i) = f(i+1) - f(i) - 1$ is a delay specifier with $(f_\#)^\# = f$. Also, $(d^\#)_\# = d$ for every $d : \mathbb{N} \to \mathbb{N}$.

The third view consists of the set SF of the stutter functions. A function $f \in Inc0$ induces a stutter function $f^\bullet \in SF$ by $f^\bullet(i) = \sup\{k \mid f(k) \leqslant i\}$. This function is well-defined since f is unbounded and $f(0) = 0$. To prove that $f^\bullet$ is a stutter function, we first observe that increasingness of f implies the Galois correspondence

$$(\forall\, i,k : f(k) \leqslant i \quad \equiv \quad k \leqslant f^\bullet(i)) .$$

It follows that $f^\bullet(i) = k$ iff $f(k) \leqslant i < f(k+1))$. Therefore, $f^\bullet$ is surjective. Monotonicity of $f^\bullet$ follows from

$$
\begin{array}{ll}
& f^\bullet(i) \leqslant f^\bullet(j) \\
\equiv & (\forall\, k : k \leqslant f^\bullet(i) \;\Rightarrow\; k \leqslant f^\bullet(j)) \\
\equiv & (\forall\, k : f(k) \leqslant i \;\Rightarrow\; f(k) \leqslant j) \\
\Leftarrow & i \leqslant j\ .
\end{array}
$$

This proves that $f^\bullet \in SF$ for every $f \in Inc0$.

Conversely, for $g \in SF$ we define $g_\bullet \in Inc0$ by $g_\bullet(k) = \inf\{i \mid k \leqslant g(i)\}$. This is well-defined since g is unbounded. The proof of $g_\bullet \in Inc0$ is based on the Galois correspondence

$$(\forall\, i,k : g_\bullet(k) \leqslant i \;\equiv\; k \leqslant g(i))\ .$$

The combined Galois correspondences also imply that $(f^\bullet)_\bullet = f$ for every $f \in Inc0$, and that $(g_\bullet)^\bullet = g$ for every $g \in SF$.

So, the three domains SF, $Inc0$, and $\mathbb{N} \to \mathbb{N}$ present three equivalent views on stuttering, and we have the operators $\#$ and $\bullet$ to move between them.

6.2. Temporization

Temporization is a general construction of a stuttering history extension, that closely resembles the extension $K \rightarrowtriangle\!\!\!\!\rightarrow K_1$ of Section 4.6.

Let K be a specification with a numerical state function tmp. We define the specification $W = Tm(K, tmp)$ by

$$
\begin{array}{l}
states(W) \;=\; \{(x,i) \mid x \in states(K) \;\wedge\; i \in [0\,..\,tmp(x)]\}\ , \\
start(W) \;=\; start(K) \times \{0\}\ , \\
((x,i),(y,j)) \in step(W) \;\equiv \\
\quad (x = y \;\wedge\; j \in \{i, i+1\}) \\
\quad \vee\ ((x,y) \in step(K) \;\wedge\; x \neq y \;\wedge\; i = tmp(x) \;\wedge\; j = 0)\ , \\
ws \in prop(W) \;\equiv\; fst \circ ws \in prop(K)\ .
\end{array}
$$

This definition has the effect that every step (x, y) of K can be mimicked by W, but only after $tmp(x)$ internal steps of W. It is easy to see that fst is a refinement mapping from W to K. We define the relation $ivft$ between $states(K)$ and $states(W)$ to be its inverse fst^{-1}. We now claim soundness:

Lemma 6.1. *Let K be a specification with a numerical state function tmp. Then $ivft : K \rightarrowtriangle\!\!\!\!\rightarrow Tm(K, tmp)$ is a stuttering history extension.*

In view of this lemma, the specification $W = Tm(K, tmp)$ together with the extension $ivft : K \rightarrowtriangle\!\!\!\!\rightarrow W$ is called the *temporizing extension* of K with respect to tmp.

The reason for introducing temporizing extensions is that they preserve bi-determinism while inserting stutterings in a controlled fashion. This is expressed in the next two lemmas.

Lemma 6.2. *Let $W = Tm(K, tmp)$ for a specification K. If K is deterministic, or backwards-deterministic, or fresh, or full, or never-terminating, or bi-deterministic, then W has the same property.*

Proof. The first three cases are trivial. The fourth case is proved by means of a behaviour construction also used in the proof of Lemma 3.2. Since the refinement mapping $fst : W \rightarrowtriangle K$ maps every terminating behaviour of W to a terminating behaviour of K, we have that W is never-terminating if K is never-terminating. The sixth case follows from the other ones. □

Lemma 6.3. *Let K be bi-deterministic. Let s denote the successor functions of K and $W = Tm(K, tmp)$. Let $z \in start(K)$, so that $(z, 0) \in start(W)$. Then $fst \circ s^*(z, 0) = s^*(z) \circ d^{\#\bullet}$ where $d = tmp \circ s^*(z)$.*

Proof. Since K is never-terminating, the successor function s of K satisfies $s^{k+1}(z) \neq s^k(z)$ for all k. We now write $ac = d^\#$ and $g = ac^\bullet$, and then use induction to prove that $s^i(z, 0) = (s^{g(i)}(z), i - ac(g(i)))$ for all indices i. This implies that $fst(s^i(z, 0)) = s^{g(i)}(z)$ and hence $fst \circ s^*(z, 0) = s^*(z) \circ g$. □

6.3. Stuttering completeness

This section contains three versions of the completeness result for the stuttering theory. We first prove the main technical result, which is a stuttering analogue of Theorem 5.1.

Theorem 6.1. *Let K be a bi-deterministic specification. Let $F : K \rightarrowtriangle\!\!\!\!\rightarrow L$ be a simulation. Then there is a numerical state function tmp on K and a refinement mapping $f : Tm(K, tmp) \rightarrowtriangle L$ with $(ivft\,;\, f) \subseteq F$.*

Proof. We first use that F is a simulation. It follows that, for every behaviour xs of K, there is a behaviour ys of L and a stutter function h such that $(xs \circ h, ys) \in F^\omega$. In particular, by Lemma 5.4(c), every $z \in start(K)$ has a behaviour ys of L and a stutter function h such that $(s^*(z) \circ h, ys) \in F^\omega$.

By the Axiom of Choice, it follows that there exist choice functions ε from $start(K)$ to behaviours of L and η from $start(K)$ to the stutter functions such that $(s^*(z) \circ \eta(z), \varepsilon(z)) \in F^\omega$ for every $z \in start(K)$.

For any state x of K, recall that $x = s^{c(x)}(q(x))$, the $c(x)$-th element of behaviour $s^*(q(x))$. This behaviour is slowed down by means of the stutter function $\eta(q(x))$. According to Section 6.1, this defines delay specifiers $dinv(x) = \eta(q(x))_{\bullet\#} : \mathbb{N} \to \mathbb{N}$. Since state x is the $c(x)$-th state of its behaviour, we define the temporizing state function tm by $tm(x) = dinv(x)(c(x))$.

Function tm is used to form a temporizing specification $W = Tm(K, tm)$. Lemma 6.2 implies that specification W is bi-deterministic. We use the symbols s, q, and c also to denote the successor function, the origin function, and the clock function of W.

For every $z \in start(K)$, Lemma 6.3 implies $fst \circ s^*(z, 0) = s^*(z) \circ d^{\#\bullet}$ where $d = tm \circ (s^*(z))$. For every index i, we have $d(i) = tm(s^i(z)) = dinv(s^i(z))(c(s^i(z))) = dinv(z)(i)$, since z is the origin of all states in $s^*(z)$. This implies that $d = dinv(z)$ and hence $d^{\#\bullet} = dinv(z)^{\#\bullet} = (\eta(z)_{\bullet\#})^{\#\bullet} = \eta(z)$. This implies that, for every $z \in start(K)$, the successor function s of W satisfies $fst \circ s^*(z, 0) = s^*(z) \circ \eta(z)$.

Let the function $h : start(W) \to Beh(L)$ be defined by $h(z, 0) = \varepsilon(z)$ for all $z \in start(K)$. By Lemma 5.2, the function $f : states(W) \to states(L)$ given by $f(w) = h(q(w))(c(w))$ is a refinement mapping $W \rightarrowtriangle L$.

It remains to verify $(ivft;\, f) \subseteq F$. Let $(x, y) \in (ivft;\, f)$. Then there is $w \in states(W)$ with $x = fst(w)$ and $f(w) = y$. We can write $w = s^n(z, 0)$ with $n = c(w)$ and $(z, 0) = q(w)$. Then we have $x = fst(s^n(z, 0)) = (s^*(z) \circ \eta(z))(n)$, and $y = h(z, 0)(n) = \varepsilon(z)(n)$. This implies $(x, y) = (s^*(z) \circ \eta(z)(n), \varepsilon(z)(n)) \in F$ since $(s^*(z) \circ h, ys) \in F^\omega$. □

By Theorem 5.4, every specification K has a strict extension $K \rightarrowtriangle E$ with E bi-deterministic. The next result shows that such an extension is in a certain sense also "universal" for general simulations.

Theorem 6.2. *Let $e : K \rightarrowtriangle E$ be an extension of specifications such that E is bi-deterministic as in Theorem 5.4. Let $F : K \rightarrowtriangle\!\!\!\!\rightarrow L$ be a simulation. Then E has a temporizing extension $t : E \rightarrowtriangle\!\!\!\!\rightarrow T$ with a refinement mapping $f : T \rightarrowtriangle L$ such that $(e\,;\, t\,;\, f) \subseteq F$.*

Proof. Lemma 4.1 yields a simulation $G : E \rightarrowtriangle\!\!\!\!\rightarrow L$ with $(e\,;\, G) \subseteq F$. Theorem 6.1 then gives a temporizing extension $t : E \rightarrowtriangle\!\!\!\!\rightarrow T$ and a refinement mapping $f : T \rightarrowtriangle L$ with $(t\,;\, f) \subseteq G$. It follows that $(e\,;\, t\,;\, f) \subseteq F$. □

For practical purposes, this result means that proofs of stuttering simulations only need clocking extensions, eternity extensions, stuttering history extensions, and refinement mappings, as expressed in the following corollary.

Corollary 6.1. *Let $F : K \rightarrowtriangle\!\!\!\!\rightarrow L$ be a simulation. Then there is a stuttering extension $G : K \rightarrowtriangle\!\!\!\!\rightarrow T$ which is a composition of a clocking extension, an eternity extension, and a stuttering history extension, and a refinement mapping $f : T \rightarrowtriangle L$, such that $(G;\, f) \subseteq F$.*

7. How to find a suitable eternity extension

In the field of Concurrency Verification, most researchers agree that assertional methods are to be preferred over behavioural ones. In other words, one should try to reduce the verification to arguments about states and the next-state relation, and should eliminate consideration of complete behaviours as much as possible. Indeed, we developed the theory presented here with precisely this aim.

The verifier of a specific algorithm should not look inside the proofs of Theorems 5.1 and 5.4 and Corollary 6.1 to find a suitable eternity extension. Indeed, the universal eternity extension is very inadequate for that purpose. The theorems only serve as a reassurance: if there is a simulation, it can be decomposed in this way.

In our experience Hesselink [8,9,13], we first introduce history variables in such a way that the states hold enough information about how they have been reached. If at that point the specification cannot be proved since the intended simulation does not satisfy condition (SF1) of Section 3.4, one may feel forced to introduce prophecies (if the prophecy required only concerns the immediately next step of the specification, it may be possible to avoid it by using a gliding simulation as in Hesselink[13]).

The prophecy variables of Abadi and Lamport [1] Jonnson [14], Engelhardt and de Roever [4] are only sound under complicated technical conditions. Our alternative is to combine the required subsequent values of a prophecy variable in a single eternity variable, with an additional counter variable to indicate the "current prophecy". In this way, indeed, the construction used in the proof of Theorem 5.1 reemerges, but it is not the complete state that is recorded in the eternity variable but only those parts that are needed for prophecies.

Since the order of the events to be prophesied may differ in different behaviours, we prefer not to enumerate them. We let every event consist of a choice of a new value. So, in general, we introduce a set E of events and a function $\mathtt{rec} : \mathit{states}(K) \to E$, which indicates which events occur. Every behaviour xs of K determines a set of events $\mathtt{rec}(xs) = \{\mathtt{rec}(xs(n)) \mid n \in \mathbb{N}\}$. Note that subsequent states may have the same $\mathtt{rec}$ values, in which case they do not generate new elements of $\mathtt{rec}(xs)$. Finally, let M be a set of subsets of E such that, for every behaviour xs of K, there is $m \in M$ with $\mathtt{rec}(xs) \subseteq m$. Let relation Rec between K and M be defined by $(x, m) \in \mathit{Rec}$ if and only if $\mathtt{rec}(x) \in m$. It is easy to verify that Rec is a behaviour restriction. Indeed, if $\mathtt{rec}(xs) \subseteq m$, then $(xs(n), m) \in \mathit{Rec}$ for all indices n.

In general, we propose to design eternity extensions in this way: determine a projection function $\mathtt{rec}$ that selects the aspects of the state space needed for prophecies. Select a set M of subsets of E such that, for every behaviour xs of K, there is $m \in M$ with $\mathtt{rec}(xs) \subseteq m$. Finally, construct the eternity extension corresponding to the behaviour restriction Rec.

8. Conclusions

We have weakened the concept of simulation to get closer agreement with the concept of implementation of Abadi and Lamport [1]. So, now, the simulations of Hesselink [7,9] are called *strict* simulations.

We have strengthened our previous semantic completeness result in three different ways. In Theorem 5.2, we show that, if $e : K \rightarrowtriangle E$ is a strict extension to a bi-deterministic specification E, every strict simulation $K \rightarrowtriangle L$ factorizes over e and a refinement mapping $E \rightarrowtriangle L$. In Theorem 6.2, we prove with the same assumptions on F, that every simulation $K \rightarrowtriangle\!\!\rightarrowtriangle L$ factorizes over e, a temporizing extension $E \rightarrowtriangle\!\!\rightarrowtriangle T$, and a refinement mapping $T \rightarrowtriangle L$.

For this to be theoretically useful, we need enough extensions to bi-deterministic specifications. Theorem 5.4 says that every specification has a strict extension to a bi-deterministic specification, which is obtained as a composition of a clocking extension with an eternity extension.

In comparison with Abadi and Lamport [1], Jonnson [14], Engelhardt and de Roever [4], we eliminate the assumptions of finite invisible nondeterminism, internal continuity, and machine-closedness, as well as the finiteness conditions needed for the soundness of the prophecy variables. In comparison with Hesselink [9], we eliminate the assumption of preservation of quiescence and we add the treatment of nonstrict simulations. The concept of stuttering forward simulations is a variation of the stuttering variables of Ladkin et al. [16].

The methodological question of how to find a suitable eternity extension is similar to the old question of how to find invariants. It requires imagination and experience. Most of us know how to find invariants but, as yet, our only experience with finding eternity extensions seems to be contained in Hesselink [8,9,13].

References

[1] M. Abadi, L. Lamport, The existence of refinement mappings, Theor. Comput. Sci. 82 (1991) 253–284.

[2] P.C. Attie, Liveness-preserving simulation relations, in: *ACM Symposium on the Principles of Distributed Computing, PODC'99*, 1999, pp. 63–72.
[3] R. de Nicola, F. Vaandrager, Three logics for branching time bisimulation, J. ACM 42 (1995) 458–487.
[4] K. Engelhardt, W.P. de Roever, Generalizing Abadi & Lamport's method to solve a problem posed by A. Pnueli, in: J.C.P. Woodcock, P.G. Larsen (Eds.), FME'93: Industrial-Strength Formal Methods, Lecture Notes in Computer Science, vol. 670, Springer, 1993, pp. 294–313.
[5] D. Griffioen, F. Vaandrager, A theory of normed simulations, ACM Trans. Comp. Logic 5 (2003) 1–33.
[6] J. He, C.A.R. Hoare, J.W. Sanders, Data refinement refined, in: B. Robinet, R. Wilhelm (Eds.), ESOP 86, LNCS, vol. 213, Springer, New York, 1986, pp. 187–196.
[7] W.H. Hesselink, Eternity variables to simulate specifications, in: E.A. Boiten, B. Moeller (Eds.), MPC 2002, LNCS, vol. 2386, Springer, New York, 2002, pp. 117–130.
[8] W.H. Hesselink, Using eternity variables to specify and prove a serializable database interface, Sci. Comput. Program. 51 (2004) 47–85.
[9] W.H. Hesselink, Eternity variables to prove simulation of specifications, ACM Trans. Comp. Logic 6 (2005) 175–201.
[10] W.H. Hesselink, PVS proof script of "universal extensions to simulate specifications", http://www.cs.rug.nl/∼wim/mechver/eternity, 2005b.
[11] W.H. Hesselink, Refinement verification of the lazy caching algorithm, Acta Inf. 43 (2006) 195–222.
[12] W.H. Hesselink, Splitting forward simulations to cope with liveness, Acta Inf. 42 (2006) 583–602.
[13] W.H. Hesselink, A criterion for atomicity revisited, Acta Inf. 44 (2007) 123–151.
[14] B. Jonnson, Simulations between specifications of distributed systems, in: J.C.M. Baeten, J.F. Groote (Eds.), CONCUR '91, LNCS, vol. 527, Springer, New York, 1991, pp. 346–360.
[15] B. Jonsson, A. Pnueli, C. Rump, Proving refinement using transduction, Distrib. Comput. 12 (1999) 129–149.
[16] P. Ladkin, L. Lamport, B. Olivier, D. Roegel, Lazy caching in TLA, Distrib. Comput. 12 (1999) 151–174.
[17] L. Lamport, A simple approach to specifying concurrent systems, Commun. ACM 32 (1989) 32–45.
[18] N. Lynch, F. Vaandrager, Forward and backward simulations, part I: untimed systems, Inf. Comput. 121 (1995) 214–233.
[19] Z. Manna, A. Pnueli, Temporal Verification of Reactive Systems: Safety, Springer, New York, 1995.
[20] P. Manolios, A compositional theory of refinement for branching time, in: D. Geist and E. Tronci (Eds.), Correct Hardware Design and Verification Methods, 12th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2003, Proceedings, volume 2860 of *LNCS*, pp. 304–318. Springer, 2003. ISBN 3-540-20363-X.
[21] R. Milner, An algebraic definition of simulation between programs, in: Proc. 2nd Int. Joint Conf. on Artificial Intelligence, Br. Comp. Soc., 1971, pp. 481–489.
[22] S. Nejati, A. Gurfinkel, M. Chechik, Stuttering abstraction for model checking, www.cs.toronto.edu/fm, 2004.
[23] S. Owre, N. Shankar, J.M. Rushby, D.W.J. Stringer-Calvert, *PVS Version 2.4, System Guide, Prover Guide, PVS Language Reference*, 2001. http://pvs.csl.sri.com.
[24] D. Park, Concurrency and automata on infinite sequences, in: Theor. Comput. Sci., LNCS, vol. 104, Springer, Berlin, 1981, pp. 167–183.
[25] F.B. Schneider, Introduction, Distrib. Comput. 6 (1) (1992) 1–3.
[26] R.J. van Glabbeek, W.P. Weijland, Branching time and abstraction in bisimulation semantics, J. ACM 43 (1996) 555–600.