Math. Struct. in Comp. Science (2003), *vol.* 13, *pp.* 589–618. © 2003 Cambridge University Press
DOI: 10.1017/S0960129503004006 Printed in the United Kingdom

Embedding untimed into timed process algebra: the case for explicit termination

J. C. M. BAETEN

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, Eindhoven, The Netherlands

Received 10 January 2001; revised 12 May 2002

In ACP-style process algebra the interpretation of a constant atomic action combines action execution with termination. In a setting with timing, different forms of termination can be distinguished: some-time termination, termination before the next clock tick, urgent termination, having terminated. In a setting with the silent action τ, we also have silent termination. This leads to problems with the interpretation of atomic actions in timed theories that involve some form of the empty process or some form of the silent action. Reflection on these problems led to a redesign of basic process algebra, where action execution and termination are separated. Instead of actions as constants, we have action prefix operators. Sequential composition remains a basic operator, and thus we have two basic constants for termination: δ for unsuccessful termination (deadlock) and ϵ for successful termination (skip). We can recover standard ACP-style process algebras as subtheories of the new theory. The new approach has definite advantages over the standard approach. The paper contributes to ongoing work on relationships between algebras with different timing features.

1. Introduction

In ACP-style process algebra (see, for example, Bergstra and Klop (1984b), Baeten and Weijland (1990) and Baeten and Verhoef (1995)), the interpretation of a constant atomic action combines action execution with termination. In a setting with timing, different forms of termination can be distinguished: some-time termination, termination before the next clock tick, urgent termination, having terminated. We will explain these notions in the paper. In the presence of the silent action τ, we also have silent termination.

This leads to problems with the interpretation of atomic actions in timed theories that involve some form of the empty process or some form of the silent action (Baeten *et al.* 2000; Baeten and Reniers 2000).

Reflection on these problems led to a redesign of basic process algebra, where action execution and termination are separated. Instead of actions as constants, we have action prefix operators as in CCS (Milner 1989) or CSP (Hoare 1985). Unlike the CCS case, but as in CSP, we have that sequential composition remains a basic operator. As a consequence, we have two basic constants for termination: δ for unsuccessful termination (deadlock) and ϵ for successful termination (skip) as in Aceto and Hennessy (1992).

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Thus, the syntax of our basic language is pretty close to CSP. However, an important difference from CSP is that bisimulation is our basic notion of equivalence (Park 1981), whereas CSP works with failure(-divergences) equivalence (Hoare 1985). In failure equivalence, applying the abstraction or hiding operator to a term yields a term that does not need the use of an internal event τ (Bergstra *et al.* 1987), except, perhaps, at the root, and with the use of two choice operators, not even there. Put another way, all internal events can be removed from a process expression. Consequently, extra internal events can be added to simplify the giving of semantics, as they are invisible anyway. Thus, sequential composition can be handled easily in CSP by inserting an internal termination event. The same goes for the unfolding of recursion. In bisimulation equivalence, such an event can sometimes be (indirectly) observable, thereby invalidating some laws that we want to hold. In CCS, sequential composition is just an add-on feature, which is also handled by inserting extra internal events. In contrast, in this paper we handle termination in a more faithful way, not as an action, but as a predicate that holds in certain states.

Another principle we have in common with CSP is the interplay of axioms and semantics: the basic ideas that go into the design of the algebra are a set of basic laws that should hold for our operators, and some fundamental ideas about behaviour, that are expressed in the operational rules. In contrast, in CCS, semantics comes first, and the laws are derived afterwards. A good illustration of our meaning is given in the next section, where operational intuition and required laws together determine the design of the algebra.

We can recover standard ACP-style process algebras as subtheories of the new theory. To be more precise, standard BPA, PA, ACP become SRM (Subalgebra of Reduced Model, in the terminology of Baeten and Bergstra (1994)) specifications of the new approach. Thus, all results established earlier in an ACP setting can still be used in our setting. On the other hand, the new approach has definite advantages over the standard approach.

We have better separation of action execution and termination, and, moreover, better separation of atomic actions as a parameter of the theory and as signature elements. We can define a minimal process algebra without sequential composition, and this eases the formulation of concepts such as structural induction, linearity, elimination and guardedness. The difference between the silent step τ and the empty step ϵ also becomes clearer.

In the operational semantics, we have no need for separate terminating action executions. We have a natural restriction of iteration to action prefix iteration, allowing for complete axiomatisations in more cases. Moreover, prefix iteration is sufficient to formulate time iteration and the embedding of untimed into timed theories. In addition, prefix iteration suffices for the formulation of divergent behaviour and fair iteration.

To summarise, this paper gives a redesign of ACP with explicit termination, allowing for improved timing extensions, and at the same time it gives a redesign of CSP in bisimulation semantics.

In the next section, we will analyse the embedding of untimed into timed process algebra. In the following sections, we present our process algebra with explicit termination with its timed extensions. In particular, we have a look at recursion, prefix iteration, projection, renamings, parallel composition and silent step.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

2. Interpretation of untimed into timed

When we design a formalism that incorporates timing aspects of a system it is important that different forms of timing can be considered, such as discrete time versus dense time, or relative time versus absolute time. Also, we should consider the interplay with untimed specifications. It may be that for some parts of the system, timing information is not relevant, or a first draft of a part of a system does not yet take timing considerations into account. In any case, it is relevant to consider the relation between untimed and timed versions of the same formal method. This is also true for process algebra. Untimed process algebras have been around for some 20 years, and timed versions are now becoming well-established. In the final section, we will give a brief survey of related work.

The most important difference between all of this work and our work is that we consider the relationship amongst algebras with different ways of dealing with timing. Bergstra and I (Baeten and Bergstra 1991) have already considered both absolute and relative forms of our dense time theory and their relationship. This relationship was improved in the discrete time theories in Baeten and Bergstra (1996), where we also considered the embedding of the discrete into the dense time theories. It turned out that this embedding can be improved upon when we use so-called urgent actions in the dense time theories, and this is worked out in Baeten and Middelburg (2001). But the embeddings were still not optimal. We found this in Baeten *et al.* (2000) and Baeten and Reniers (2000), where some problems arose with the interpretation of untimed constants in a timed setting. Thus, this article can be seen as a continuation of the work on relating different forms of timing in an algebraic setting. We focus on the embedding of untimed into discrete relative time, and briefly mention the absolute time and dense time variants.

First, consider an untimed process that starts with the execution of an atomic action a. In a timed setting, it is natural to interpret this by saying that a occurs at some unspecified moment of time. Stated more precisely, we interpret the atomic action as a *delayable* action: arbitrary time steps can occur before action execution. Besides these delayable actions, most timed theories will contain some form of actions with timing constraints (such as, in relative discrete time ACP, the action $\underline{a}$ that must execute a before the next clock tick).

Next, we consider the inaction constant δ. In the untimed setting, this is the process that cannot execute any action, and cannot terminate (the process STOP in CSP (Hoare 1985) or LOTOS (Brinksma 1989)). Interpreting this process in a timed setting, we find we have two options:

1 We can interpret it as *time stop*, standing for the *deadlocked* process, that allows for no action execution, no time step and no termination.
2 We can interpret it as *livelock* that allows for no action execution or termination, but does allow for time to pass.

Now we want to argue that we prefer the second interpretation, that is, as livelock. We have (at least) three reasons for prefering this interpretation:

1 δ stands for the blocked atomic action. When a process starts with the atomic action a, and we block the action (called restriction in CCS), the outcome is δ. Restriction, also called encapsulation, is renaming into δ. Since a can let time pass, so should δ.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

2 The way divergence is treated. To give an example, the process that starts with action a, and then diverges, that is, can only perform an unbounded sequence of internal steps, and that cannot terminate (we can denote this process as $a\tau^{\omega}$), equals the process $a\delta$ in weak or branching bisimulation equivalence (van Glabbeek and Weijland 1996). This fits well with an interpretation of δ as livelock.
3 In the axiomatisation of parallel composition $\|$, we have $a \| \delta = a\delta$ (this is a consequence of interleaving, and treating δ in the same way as an action). Since in a parallel composition time can pass only if both components allow it to do so, δ must also be able to let time pass.

Thus, we let δ stand for the livelock process. We need another constant for the deadlocked process (time stop), and the notation $\dot{\delta}$ is used for the deadlocked process in Baeten and Bergstra (1996). It is worth emphasising the difference between the two process expressions: δ represents a process in a state that allows for certain behaviour (that is, passage of time), whereas $\dot{\delta}$ does not allow for any behaviour whatsoever, and does not really represent a state of a system: the process has already deadlocked, and can be said to be in an inconsistent or virtual state (*cf.* Baeten and Reniers (2000)).

In the basic process algebra BPA (with the basic operators $+$ for alternative composition and $\cdot$ for sequential composition), there are two axioms for δ:

$$x + \delta = x \quad (A6) \qquad \delta \cdot x = \delta \quad (A7).$$

In a setting with time, we will interpret alternative composition as *weak choice*, which means that time can progress as soon as one alternative allows it to do so. Using strong choice, both components must be able to delay before a delay can occur. The interpretation as weak choice is more general than an interpretation as strong choice; in the latter case, we need an additional operator to denote a time out (see further on).

The second axiom, A7, also holds in this timed setting, but the first axiom, A6, does not hold for all processes x, due to our interpretation of alternative composition as weak choice. If x has restricted timing, adding δ adds the possibility of arbitrary delay. To give an example, $\underline{\underline{a}}$ must be executed before the first clock tick, and does not allow for further delay, so $\underline{\underline{a}} + \delta \neq \underline{\underline{a}}$. We see A6 only holds for all *delayable* processes x. This is in contrast with the situation for $\dot{\delta}$: the law $x + \dot{\delta} = x$ will hold for all timed processes x, also undelayable ones. The difference between δ and $\dot{\delta}$ can also be observed (in the untimed theory; in Baeten and Bergstra (1996), $\dot{\delta}$ was also introduced in the untimed theory) in the combination with parallel composition: we have the laws

$$x \| \delta = x \cdot \delta \qquad x \| \dot{\delta} = \dot{\delta}.$$

These laws are explained as follows: δ allows for all behaviour of parallel components, as it can wait arbitrarily long, whereas $\dot{\delta}$ has already deadlocked, is in an inconsistent state, and no action or time behaviour of parallel components is allowed.

Now let us consider successful termination. In standard ACP-style process algebra, successful termination is usually *implicit*: the atomic action constant a will execute the action followed by successful termination, there is no constant for the terminated process. In some work, termination is made explicit, see, for example, Vrancken (1997) and Baeten and Weijland (1990), but also Aceto and Hennessy (1992). We denote the successful

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

termination constant by ϵ, and call this the *empty* process or *skip* (*exit* in LOTOS). In the untimed theory, this process is operationally characterised by one rule, *viz*. the termination predicate (denoted $\downarrow$ or $\surd$) holds and no action can be executed. Interpreting ϵ in a timed setting, there are again two options: we either allow for time steps before termination, or do not. We again prefer a delayable interpretation. We can give two arguments for this:

1 The relation of ϵ and parallel composition was investigated in Vrancken (1997). There, it is argued that the equation

$$a \parallel (b + \epsilon) = a \cdot (b + \epsilon) + b \cdot a$$

should hold, and thus there is no summand $\epsilon \cdot a$, and the ϵ cannot be executed before the a. Since a, b are delayable, this also means that ϵ must be delayable.

2 The law A6 above is in all accounts a law of untimed process algebra. As argued above, this law only holds for delayable processes. Thus it is desirable that all untimed processes are delayable. Having the process ϵ in untimed process algebra, we need to have ϵ delayable.

Thus we make the choice of having ϵ stand for the process that can terminate at some unspecified moment of time. Analogously to the situation above, we introduced the constant $\dot{\epsilon}$ to stand for the terminated process in Baeten and Reniers (2000). Again, $\dot{\epsilon}$ has already terminated, and is in an inconsistent or virtual state. The terminated process $\dot{\epsilon}$ cannot execute any action, and cannot let time pass.

Now we return to our basic untimed process algebra BPA with explicit termination, containing constants δ, ϵ. Since ϵ denotes skip, the standard theory involves the laws

$$\epsilon \cdot x = x \quad (A8) \qquad x \cdot \epsilon = x \quad (A9).$$

These laws also occur in Aceto and Hennessy (1992) (with *nil* instead of ϵ). Interpreting these laws in a timed setting, we see that A8 expresses the fact that x must be delayable (adding an arbitrary delay at the start does not make any difference), and A9 expresses the fact that an arbitrary delay is possible before termination of x. Thus the interpretation of a constant a must be that action a is executed at some unspecified time, followed by termination some time later (or at the same time). Operationally, we have the rule $a \stackrel{a}{\rightarrow} \epsilon$.

Then we encounter a problem in the timed theory. We must be able to express the process that will execute action a at an unspecified time, followed by execution of b before the next clock tick (for instance, receive a message at some time and send it on within a specified time). We want to express this process as $a \cdot \underline{b}$. But then we have a different interpretation of a, as termination must now follow immediately (or at least, before the next clock tick). Operationally, we have the rule $a \stackrel{a}{\rightarrow} \dot{\epsilon}$ instead of $a \stackrel{a}{\rightarrow} \epsilon$. This implies we are dealing with two different interpretations, which we should keep separate.

We encountered the same phenomenon when dealing with timed process algebra with abstraction in Baeten *et al.* (2000). In order to enforce the first τ-law $x \cdot \tau = x$, an interpretation of an atomic action with some-time termination becomes necessary; on the other hand, working with actions with restricted timing, immediate termination is needed.

The solution we propose in this paper is always to denote explicitly which form of termination we are considering. We will not have the constant process a with implicit

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

termination (as in Baeten and Weijland (1990) and Aceto and Hennessy (1992)), but, instead, replace this with a unary *action prefixing* operator ax of CCS or CSP. Unlike the case for CCS, but as in CSP, we combine this with general sequential composition and constants for both successful and unsuccessful termination. The two constants are called SKIP and STOP in CSP, and EXIT and STOP in LOTOS. All the laws of untimed process algebra still hold in the untimed subtheory, and we have a straightforward interpretation of the untimed into the timed theory – we just add new forms of termination.

To summarise this section, we have discussed a number of design decisions to reach our aim of embedding an untimed algebra into a timed one while keeping the axioms, namely:

— We separate action execution from termination. In ACP-style process algebra, this needs to be motivated. As we use bisimulation equivalence, we need to be much more precise than in CSP concerning the use of internal events.
— We introduce different forms of termination. The use of sequential composition requires a difference between successful and unsuccessful termination. The use of both untimed and timed theories together requires a distinction between delayable and undelayable termination.
— We interpret alternative composition as weak choice. This leads to a more general theory, with an easier formulation of time-outs.

In the following sections, we will demonstrate that with these choices we can indeed reach our aim.

3. Minimal Process Algebra

We start out from the process algebra MPA or Minimal Process Algebra. This acronym was introduced in Fokkink (1994). MPA is a modification of the basic process algebra BPA, in which action constants and sequential composition are replaced by action prefix operators. We assume that we are given a set of actions A. This set, usually finite, is considered a parameter of the theory. The signature elements are:

— Binary operator $+$ denotes *alternative composition* or choice. Process $x + y$ executes either x or y, but not both. The choice is resolved upon execution of the first action.
— Constant δ denotes *inaction*, and is the neutral element of alternative composition. Process δ cannot execute any action, and cannot terminate.
— Constant ϵ denotes the *empty process* or *skip*. It is the neutral element of sequential composition (considered in the next section). Process ϵ cannot execute any action, but terminates successfully.
— We have a unary operator $a.$ for each $a \in A$ called *action prefix*. Process $a.x$, usually written ax, executes action a and then proceeds as x. Putting a constant for x, we have the basic processes $a\delta$ (deadlock upon execution of a) and $a\epsilon$ (successful termination upon execution of a).

The process algebra MPA is axiomatised by Axioms A1,2,3,6 in Table 1. These axioms are well known from, for example, CCS (Milner 1989). This system was called FINTREE

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 1. *Axioms of MPA.*

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$x + \delta = x$	A6

Table 2. *Deduction rules for MPA ($a \in A$).*

$$\frac{}{\epsilon\downarrow} \qquad \frac{}{ax \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \qquad \frac{x\downarrow}{x + y\downarrow} \qquad \frac{y\downarrow}{x + y\downarrow}$$

in Aceto *et al.* (1994). Prefix operators always bind stronger than other operators, $+$ always binds weaker.

Using the axioms, each closed MPA-term t can be written in one of the following two forms:

1 $\mathrm{MPA} \vdash t = \delta + a_1 t_1 + \cdots + a_n t_n$, or
2 $\mathrm{MPA} \vdash t = \epsilon + a_1 t_1 + \cdots + a_n t_n$,

for certain $n \in \mathbb{N}$, $a_i \in A$ and (simpler) MPA-terms t_i ($i \leqslant n$). (In the first case, we can omit δ when $n > 0$. If we let δ stand for empty summation, we can simplify further.)

We present structured operational rules (so-called *SOS rules*) in the style of Plotkin (Plotkin 1983). The rules in Table 2 define the following relations on closed MPA-terms: binary relations $_ \xrightarrow{a} _$ (for $a \in A$) and a unary relation $_\downarrow$. Intuitively, they have the following meaning:

— $x \xrightarrow{a} x'$ means that x evolves into x' by executing atomic action a;
— $x\downarrow$ means that x has an option to terminate successfully (without executing an action).

Thus, the relations concern action execution and termination, respectively; we do not need a mixed relation $_ \xrightarrow{a} \surd$ as in Baeten and Weijland (1990) or Baeten and Verhoef (1995).

The rules provide a transition system for each closed term. We define the equivalence relation of *bisimulation* (notation $\leftrightarrows$) on the resulting transition systems in the standard way (see, for example, Milner (1989) and Baeten and Weijland (1990)).

Definition 3.1. Let R be a binary *symmetric* relation on closed terms. We say R is a *bisimulation* if the following holds:

— whenever $R(x, y)$ and $x \xrightarrow{a} x'$, there is a term y' such that $y \xrightarrow{a} y'$ and $R(x', y')$;
— whenever $R(x, y)$ and $x\downarrow$, we have $y\downarrow$.

We say two closed terms t, s are *bisimulation equivalent* or *bisimilar*, notation $t \leftrightarrows s$ if there is a bisimulation R with $R(s, t)$.

The following results are standard: bisimulation equivalence is a congruence relation on closed MPA-terms (this follows from the format of the deduction rules, see, for example,

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 3. *Axioms of MPA in relative discrete time ($a \in A$).*

$x + \underline{\underline{\delta}} = x$	A6DR	$ax = \underline{\underline{a}}x + \underline{\underline{\sigma}}ax$	DRA
$\underline{\underline{\sigma}}x + \underline{\underline{\sigma}}y = \underline{\underline{\sigma}}(x + y)$	DRTD	$\delta = \underline{\underline{\sigma}}\delta$	DRD
		$\epsilon = \underline{\underline{\epsilon}} + \underline{\underline{\sigma}}\epsilon$	DRE

Baeten and Verhoef (1995)); and the theory MPA is sound and complete for the model of transition systems modulo bisimulation, that is, for all closed terms t, s we have

$$\mathrm{MPA} \vdash t = s \iff t \leftrightarrows s.$$

This is an easy application of the results in Baeten and Verhoef (1995).

So far, we have considered untimed process algebra. Now we consider timing. We interpret untimed processes as processes that can delay an arbitrary amount of time before each action and before termination. It is not necessary to look at the whole framework of discrete and dense timed, absolute and relative timed, time-stamped and two-phase process algebras (Baeten and Middelburg 2001). Instead, we can make the point by considering one member of this family, *viz.* process algebra with discrete time in relative timing in two-phase notation (Baeten and Bergstra 1996).

We have the following syntax in addition to the signature elements of MPA:

— *Current time slice action prefix* $\underline{\underline{a}}.$, where $a \in A$. The process $\underline{\underline{a}}x$ will execute action a in the current time slice and evolve into x.

— *Current slice time stop* $\underline{\underline{\delta}}$ – an extra constant. Time cannot progress beyond the current time slice, and no termination can take place. Note that $\underline{\underline{\delta}}$ does not represent a process that has deadlocked already (so is not $\dot{\delta}$), and activity (of a parallel component) is allowed, as long as no clock tick takes place. In discrete relative time process algebra, $\underline{\underline{\delta}}$ is the neutral element of alternative composition. This changes if we add a constant like $\dot{\delta}$ or $\dot{\epsilon}$.

— *Current slice termination* $\underline{\underline{\epsilon}}$ – also an extra constant. Time cannot progress beyond the current time slice, and termination will take place in the current time slice. Again, take note that $\underline{\underline{\epsilon}}$ has not terminated already (so is not $\dot{\epsilon}$), and allows for parallel activity before the next clock tick. In discrete relative time process algebra, $\underline{\underline{\epsilon}}$ is the neutral element of sequential composition (we add sequential composition in the next section). This changes if we add a constant like $\dot{\delta}$ or $\dot{\epsilon}$.

— *Next time slice prefix* $\underline{\underline{\sigma}}.$. The process $\underline{\underline{\sigma}}x$ will pass to the next time slice and then execute x. We elect to take $\sigma \notin A$ (this decision is rather arbitrary but emphasises the difference between passage of time and execution of an atomic action). In the absence of the next time slice prefix, the theory with $\underline{\underline{a}}.$, $\underline{\underline{\delta}}$, $\underline{\underline{\epsilon}}$ is exactly the same, that is, has the same laws, as the MPA theory built from the operators $a., \delta, \epsilon$.

The axiomatisation of MPA_{drt} replaces axiom A6 of MPA by axiom A6DR in Table 3. Further, we have axioms A1,2,3 of MPA in Table 1 and the remaining axioms in Table 3. The *Time Determinism* axiom DRTD (sometimes called *weak time factorisation*) expresses the fact that passage of time cannot introduce non-determinism, so passage of time leads to a unique remaining process. Here we see that passage of time is different from action execution, which can introduce non-determinism. This is consistent with the idea that the

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 4. *Deduction rules for MPA with relative discrete time ($a \in A$).*

$\underline{\underline{a}}x \xrightarrow{a} x$	$\underline{\underline{\epsilon}} \downarrow$	$\underline{\underline{\sigma}}x \overset{1}{\mapsto} x$
$ax \overset{1}{\mapsto} ax$	$\delta \overset{1}{\mapsto} \delta$	$\epsilon \overset{1}{\mapsto} \epsilon$
$\dfrac{x \overset{1}{\mapsto} x',\ y \overset{1}{\mapsto} y'}{x + y \overset{1}{\mapsto} x' + y'}$	$\dfrac{x \overset{1}{\mapsto} x',\ y \overset{1}{\not\mapsto}}{x + y \overset{1}{\mapsto} x'}$	$\dfrac{y \overset{1}{\mapsto} y',\ x \overset{1}{\not\mapsto}}{x + y \overset{1}{\mapsto} y'}$

choice for a certain alternative can only be effected by the execution of an action, not by the mere passage of time. On the other hand, passage of time can block a certain alternative. This is the consequence of interpreting $+$ as weak choice, so that a process $\underline{\underline{a}}+\underline{\underline{\sigma}}(\underline{\underline{b}}+\underline{\underline{c}})$ will evolve into $\underline{\underline{b}}+\underline{\underline{c}}$ by passing to the next time slice, thereby losing the option to execute a. Using strong choice $\underline{\underline{a}}+\underline{\underline{\sigma}}(\underline{\underline{b}}+\underline{\underline{c}})$ equals $\underline{\underline{a}}$, as the first argument does not allow for a delay. This is called *strong time factorisation*. Using weak choice, we immediately have a time-out mechanism. If we add an explicit time-out operator as in Baeten and Middelburg (2001), strong choice can be defined as an additional operator (strong choice equals weak choice, timed out at the earliest ultimate delay time of the components).

Axiom DRA is actually an axiom scheme: we have such an axiom for each action $a \in A$. The delayable processes ax, δ, ϵ are defined recursively. For instance, process ϵ is defined to be the solution of the recursive equation $x = \underline{\underline{\epsilon}} + \underline{\underline{\sigma}}x$. We will study recursion in the following section; for now, we assume that an equation of the form $x = p + \underline{\underline{\sigma}}x$ for a certain term p (it is enough to consider p that cannot execute an initial time step) has a unique solution, that is, there is exactly one process satisfying this equation. This recursion scheme is reminiscent of the delay operator as defined by Milner and Hennessy (Hennessy 1981; Milner 1983; Hennessy 1984). However, it is worth emphasising that their delay has no form of time factorisation.

The definition of an operational semantics by means of SOS deduction rules is as follows. To the relations of Table 2, we add a binary relation $_ \overset{1}{\mapsto} _$ on closed terms. Intuitively, $x \overset{1}{\mapsto} x'$ means that x evolves into x' by passing to the next time slice. We add the rules in Table 4 to the rules of Table 2. Note that $x \overset{1}{\not\mapsto}$ means that x cannot execute a $\overset{1}{\mapsto}$ transition, that is, x cannot pass to the next time slice. Thus, we have here an SOS definition with negative premises. The negative premises are used to enforce time determinism. However, the SOS specification is well-defined, as is shown in Vereijken (1997). Using the technique of *saturation*, it is possible to avoid the negative premises (Baeten and Reniers 2000).

We can now appreciate the advantage of having actions as a prefixing operator instead of a constant: we have the desired embedding of the delayable untimed actions, and we can consider different forms of termination: $a\underline{\underline{\epsilon}}$ will terminate in the same time slice as the execution of a, whereas $a\epsilon$ will terminate some unspecified time after the execution of a. We also have the forms $\underline{\underline{a}}\,\underline{\underline{\epsilon}}$, both action execution and termination in the current time slice, and $\underline{\underline{a}}\epsilon$, action execution in the current time slice, termination at some later time.

To state the argument again, the interpretation of a term $a\underline{\underline{b}}x$ must be that b is executed in the same time slice as a, so the interpretation of the (untimed) constant process a in the timed theory must include immediate termination (or at least, current time slice

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

termination). But then, the equation $a \cdot \epsilon = a$ is not valid anymore, spoiling the equations of the untimed theory (Baeten and Reniers 2000). It is this difficulty, that led us to develop the current paper. Subsequently, several other advantages of action prefix over action constants were found.

All processes in this discrete time theory allow for a delay up to the end of the current time slice before each action, and also before termination. (Compare this characterisation with the characterisation of untimed processes as processes that allow for an arbitrary delay before each action and before termination.) This can be appreciated better if we add processes that allow for no delay whatsoever. This happens, for instance, by adding the constants $\dot{\delta}, \dot{\epsilon}$ denoting inconsistency at the current point of time. They are characterised because they both denote an inconsistent or virtual state: the process has terminated already (either unsuccessfully or successfully (Baeten and Reniers 2000)). In the operational semantics, this is denoted by an extra predicate $\uparrow$. The predicate $\uparrow$ only holds for the inconsistent processes $\dot{\delta}, \dot{\epsilon}$, and in the definition of bisimulation we also require preservation of the predicate $\uparrow$.

In the axiomatisation, law A6DR $x + \underline{\underline{\delta}} = x$ does not hold anymore, as we will have $\dot{\delta} + \underline{\underline{\delta}} = \underline{\underline{\delta}}$. Law A6DR only holds for discretely timed processes, and will be replaced by the axiom $x + \dot{\delta} = x$. For more information, see Baeten and Reniers (2000), Baeten and Middelburg (2001) and Vereijken (1997).

Things work similarly in the absolute time variant, where time is not measured with respect to the current time but with respect to a global clock. We have the syntax elements $\underline{a}$. (execution of a in the first time slice), $\underline{\delta}$ (deadlock at the end of the first time slice), $\underline{\epsilon}$ (termination in the first time slice), $\underline{\sigma}$. (time shift prefix, shifting everything in its scope by one time slice), with similar axioms. In the operational rules, the predicates are not defined on process expressions directly, but on pairs consisting of a process expression and a time slice number. The connection between the absolute time and the relative time variants is as follows: given the (absolute) number of the current time slice, we can convert a relative time expression into an absolute time expression. Thus, relative time processes can be interpreted as functions from natural numbers to absolute time processes. For further details, see Baeten and Middelburg (2001).

There are also counterparts using dense timing of both the discrete time and the absolute time theories. The σ operators get a positive real number as an extra argument, the $\overset{1}{\mapsto}$ relation is generalised to $\overset{t}{\mapsto}$ for any positive real t, and many things work in the same way. However, the embedding of untimed processes can no longer be done using a simple recursive equation. Instead, we use the notion of *integration*, denoting a choice over a subset of the time domain of the non-negative reals. Integration is also used to embed the discrete time theories into the dense time theories; this works best in the absolute time variants, where a discrete timed action allows for a delay until (but not including) the next integer time moment. For further details, see Baeten and Middelburg (2001).

4. Recursion

The axioms for delayable processes are specific instances of a *recursive equation*. It is standard to use recursive equations to specify processes with possibly infinite behaviour,

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

see, for example, Baeten and Weijland (1990) or Baeten and Verhoef (1995). We now proceed to define recursion in our setting.

Let V be a set of variables ranging over processes. A *recursive specification* $E = E(V)$ is a set of equations $E = \{X = t_X \mid X \in V\}$ where each t_X is a term over the signature in question (for instance, MPA) and variables from V. A *solution* of a recursive specification $E(V)$ in our theory is a set of processes $\{\langle X|E\rangle \mid X \in V\}$ in some model of the theory such that the equations of $E(V)$ hold, if for all $X \in V$, X stands for $\langle X|E\rangle$. We are mostly interested in the solution for one particular variable $X \in V$.

We want to use recursive equations to define processes. Thus, we are interested in equations that have unique solutions. The equation $X = X$ is useless, since it is satisfied by every process. The well-known notion of guardedness helps us to identify useful equations. Let t be a term containing a variable X. We call an occurrence of X in t *guarded* if this occurrence of X is in the scope of a prefix operator (that is, in MPA, t has a subterm of the form as, and this X occurs in s). This is an easier formulation of guardedness than in, for example, Baeten and Weijland (1990) and Baeten and Verhoef (1995).

We say a recursive specification is *guarded* if all occurrences of all its variables in the right-hand sides of all its equations are guarded or it can be rewritten to such a recursive specification using the axioms of the theory and the equations of the specification.

Unguarded equations can also lead to other difficulties. For instance, any solution of the specification $\{X_n = a^n\epsilon + X_{n+1} \mid n \in \mathbb{N}\}$ (with the obvious definition of the a^n prefix) will have infinite branching, as it has summands $a^n\epsilon$ for all n. The solution having just these summands, and no others, can be denoted $\sum_{n\in\mathbb{N}} a^n\epsilon$.

We say an equation is *linear* if it is guarded and just uses the signature of MPA. Note that this is a very concise definition of linearity. Every linear specification is equivalent to one in head normal form. What this means is specified in the following theorem.

Theorem 4.1. Every guarded specification over MPA is equivalent to one where all equations have one of the two forms

1 $X = \delta + a_1X_1 + \cdots + a_nX_n$, or
2 $X = \epsilon + a_1X_1 + \cdots + a_nX_n$,

for certain $n \in \mathbb{N}, a_i \in A, X_i \in V$.

Every guarded specification over MPA_{drt} is equivalent to one where all equations have one of the four forms

1 $X = \underline{\underline{\delta}} + \underline{\underline{a}}_1X_1 + \cdots + \underline{\underline{a}}_nX_n$, or
2 $X = \underline{\underline{\epsilon}} + \underline{\underline{a}}_1X_1 + \cdots + \underline{\underline{a}}_nX_n$, or
3 $X = \underline{\underline{\sigma}}X_0 + \underline{\underline{a}}_1X_1 + \cdots + \underline{\underline{a}}_nX_n$, or
4 $X = \underline{\underline{\epsilon}} + \underline{\underline{\sigma}}X_0 + \underline{\underline{a}}_1X_1 + \cdots + \underline{\underline{a}}_nX_n$.

This theorem is called the *Head Normal Form Theorem*. Thus, guardedness will imply finite branching. To analyse this further, we introduce projection operators.

First, we consider the theory without time. For each natural number n, the unary operator π_n will block all further actions, after n actions have been executed. We show the axioms in Table 5 and structured operational rules in Table 6.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 5. *Axioms of projection operators* ($a \in A, n \in \mathbb{N}$).

$\pi_n(\delta) = \delta$	$\pi_0(ax) = \delta$
$\pi_n(\epsilon) = \epsilon$	$\pi_{n+1}(ax) = a\pi_n(x)$
$\pi_n(x + y) = \pi_n(x) + \pi_n(y)$	

Table 6. *Deduction rules for projection* ($a \in A, n \in \mathbb{N}$).

$\dfrac{x \xrightarrow{a} x'}{\pi_{n+1}(x) \xrightarrow{a} \pi_n(x')}$	$\dfrac{x \downarrow}{\pi_n(x) \downarrow}$	$\dfrac{x \overset{1}{\mapsto} x'}{\pi_{n+1}(x) \overset{1}{\mapsto} \pi_n(x')}$

Table 7. *Axioms of projection operators with discrete time* ($a \in A, n \in \mathbb{N}$).

$\pi_n(\underline{\delta}) = \underline{\delta}$	$\pi_0(\underline{a}x) = \underline{\delta}$
$\pi_n(\underline{\epsilon}) = \underline{\epsilon}$	$\pi_0(\underline{\sigma}x) = \underline{\delta}$
$\pi_n(x + y) = \pi_n(x) + \pi_n(y)$	$\pi_{n+1}(\underline{a}x) = \underline{a}\pi_n(x)$
	$\pi_{n+1}(\underline{\sigma}x) = \underline{\sigma}\pi_n(x)$

Based on the format of the deduction rules, we can prove that the theory of MPA with projection operators is an equational conservative extension of MPA, and that the axiomatisation is sound and complete for the model of transition systems modulo bisimulation.

In the theory with discrete relative timing, we can extend the operator above by just counting actions and not time steps, but for our purposes it is more useful to count both actions and time steps. Then, the projection axioms have to be replaced by their underlined variants, as shown in Table 7. Operationally, we add the last rule in Table 6. Using dense timing, we can count actions, and measure the total amount of delay that has occurred, thus allowing an operator $\pi_{n,t}$ that cuts off a process as soon as n actions have occurred or a total delay of t time has occurred.

For guarded specifications over MPA or MPA_{drt}, the following *projection theorem* follows easily from the Head Normal Form theorem. In theories with time, it is necessary to keep track of both actions and time, due to the presence of so-called *Zeno processes*, which exhibit infinite behaviour in a finite amount of time.

Proposition 4.1. Let E be a guarded recursive specification over the signature of MPA or MPA_{drt} using the variables V. Let p be a solution of E. Then all projections of p can be written as closed terms, independent of the particular solution p.

The projections $\pi_n(p)$ can be considered approximations of the solution p. We can formulate the following principles:

— RDP (the *Recursive Definition Principle*) – each recursive specification has at least one solution;
— RSP (the *Recursive Specification Principle*) – each *guarded* recursive specification has at most one solution;

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 8. *Deduction rules for recursion ($a \in A$).*

$\dfrac{\langle t_X \mid E\rangle \xrightarrow{a} y}{\langle X \mid E\rangle \xrightarrow{a} y}$	$\dfrac{\langle t_X \mid E\rangle \downarrow}{\langle X \mid E\rangle \downarrow}$	$\dfrac{\langle t_X \mid E\rangle \overset{1}{\mapsto} y}{\langle X \mid E\rangle \overset{1}{\mapsto} y}$

— AIP (the *Approximation Induction Principle*) – each process is determined by its finite projections.

Different models of MPA will satisfy none, some or all of these principles. Let us look at the transition system models in particular.

Consider a recursive specification E. For each variable X of E, we can add a new constant $\langle X|E\rangle$ to our syntax. Table 8 provides deduction rules for these constants. They come down to looking upon $\langle X|E\rangle$ as the process $\langle t_X|E\rangle$, which is t_X with, for all $Y \in V$, all occurrences of Y in t_X replaced by $\langle Y|E\rangle$.

Now if we add such constants $\langle X|E\rangle$ for all MPA specifications E to our syntax, we obtain a model $\mathbb{G}^\infty$ for MPA that satisfies RDP and RSP but not AIP (the process $\sum_{n\in\mathbb{N}} a^n\epsilon$ defined above has the same projections as this process with the solution of $X = aX$ as added summand (Baeten and Weijland 1990)). If we add constants $\langle X|E\rangle$ for guarded E only, we obtain a model $\mathbb{G}$ satisfying RSP and AIP but not RDP. It does not satisfy RDP, as the specification $\{X_n = a^n\epsilon + X_{n+1} \mid n \in \mathbb{N}\}$ only has solutions that are infinitely branching, as was argued before. Using guarded recursion, we always get a finitely branching solution (this is a consequence of the head Normal Form Theorem (Theorem 4.1)). As a consequence, the model $\mathbb{G}$ is really smaller than $\mathbb{G}^\infty$. Note that the example requiring infinite branching can only be given by a recursive specification with infinitely many equations.

We see that RDP no longer holds on the second model. The third possibility is to add fewer constants still, adding $\langle X|E\rangle$ for finite and guarded E only. This gives the model $\mathbb{R}$ – the model of *regular* processes. It is equivalent to the model of finite transition systems modulo bisimulation, see, for example, Bergstra and Klop (1984a). Again, we can establish that $\mathbb{R}$ is really smaller than $\mathbb{G}$, simply because an infinite transition system that is not bisimilar to a finite one can only be specified using infinitely many equations.

In the theory with discrete relative timing, the time prefix also serves as a guard. (Using dense timing, any positive delay constitutes a guard.) When we add the last rule in Table 8 as a deduction rule, the same results can be established. The axioms for delayable processes give names to the solution of certain equations. The principle RSP can then be used to derive equalities. For instance, since both processes $\epsilon + \delta$ and ϵ are solutions of the recursive equation $X = \underline{\underline{\epsilon}} + \underline{\underline{\sigma}}X$, it must be that $\epsilon + \delta = \epsilon$. Similarly, we can derive $ax + \delta = ax$.

5. Prefix iteration

A useful special class of guarded recursive equations is given by the iteration prefix. First, we will have a look at the untimed theory. For each action $a \in A$, we have a new unary operator a^* called *iteration prefix*. The process a^*x can execute a a number of times before

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 9. *Axioms of iteration prefix ($a \in A$).*

$a^*x = a(a^*x) + x$	$a^*(a^*x) = a^*x$

Table 10. *Deduction rules for iteration prefix ($a, b \in A$).*

$a^*x \xrightarrow{a} a^*x$	$\dfrac{x \xrightarrow{b} x'}{a^*x \xrightarrow{b} x'}$	$\dfrac{x\downarrow}{a^*x\downarrow}$

starting the execution of x. Note that this construction allows for unbounded behaviour, for instance $a^*\delta$ will execute a an unbounded number of times, that is, infinitely often.

The process algebra MPA extended with prefix iteration is called MPA*, which is axiomatised by the axioms of MPA and the axioms in Table 9. The first axiom says that process a^*x is a solution of the recursive equation $Y = aY + x$. The proof rule RSP now reads

$$y = ay + x \quad \Longrightarrow \quad y = a^*x \qquad \text{RSP}^*.$$

However, the remarkable fact is that this proof rule is not needed in order to achieve a complete axiomatisation of the operational model, based on the rules presented in Table 10 (this result goes back to Hennessy (1981)).

Proposition 5.1. The theory MPA* is a sound and complete axiomatisation of the model of transition systems modulo bisimulation.

Proof. The proof is a straightforward adaptation of the proof in Fokkink (1994). □

Note that here we see an advantage of just having prefix iteration. If we add iteration as a binary operator on MPA, where process x^*y can iterate the behaviour of x until exiting by doing y, then we cannot find a complete finite axiomatisation. This was shown by Sewell (Sewell 1997). On the other hand, there are extensions of action prefix iteration that still have finite axiomatisations (Aceto and Groote 1999; van Glabbeek 1997).

Considering the term model $\mathbb{P}$ of MPA*, this model is even smaller than the model of regular processes $\mathbb{R}$. In Bergstra *et al.* (1994), it is shown that there are regular processes that cannot be defined just using iteration. With prefix iteration only, the difference can be found even more simply, for instance by using a result of Aceto and Ingólfsdóttir (1996): every infinite trace of a process definable over MPA* is eventually constant, so the solution of $X = abX$ is not in $\mathbb{P}$.

Looking at the discrete relative time extension, we consider the time iteration prefix $\underline{\sigma}^*$. The time iteration prefix will prefix the process that follows with an arbitrary number of time steps. Given time iteration, the definition of delayable actions and constants is possible directly, see Table 11. Besides the other iteration prefix axioms as in Table 9, we have two additional axioms when adding time iteration to MPA$_{drt}$. The operational rules are presented in Table 12.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 11. *Axioms for time prefix iteration ($a \in A$).*

$\epsilon = \underline{\sigma}^*\underline{\epsilon}$	$\underline{\sigma}^* x = \underline{\sigma}(\underline{\sigma}^* x) + x$
$\delta = \underline{\sigma}^*\underline{\delta}$	$\underline{\sigma}^*(\underline{\sigma}^* x) = \underline{\sigma}^* x$
$ax = \underline{\sigma}^*(\underline{a}x)$	$\underline{\sigma}^* x + \underline{\sigma}^* y = \underline{\sigma}^*(x + y)$
	$\underline{\sigma}\underline{\sigma}^* x = \underline{\sigma}^*\underline{\sigma} x$

Table 12. *Deduction rules for time prefix iteration ($a \in A$).*

$\dfrac{x \xrightarrow{a} x'}{\underline{\sigma}^* x \xrightarrow{a} x'}$	$\dfrac{x\downarrow}{\underline{\sigma}^* x\downarrow}$	$\dfrac{x \overset{1}{\not\mapsto}}{\underline{\sigma}^* x \overset{1}{\mapsto} \underline{\sigma}^* x}$	$\dfrac{x \overset{1}{\mapsto} x'}{\underline{\sigma}^* x \overset{1}{\mapsto} x' + \underline{\sigma}^* x}$

In dense timing, these things do not work so nicely. Instead of time iteration, we can look at an interval until t delay prefix, which prefixes a process with a delay u chosen from the reals until t, that is, satisfying $0 \leqslant u < t$. Some finite axiomatisations can be found in Baeten and Reniers (2000).

6. Sequential composition

It is now enlightening to add sequential composition to MPA, obtaining the Sequential Process Algebra SPA. SPA is a modification of the basic process algebra $\mathrm{BPA}_{\delta\epsilon}$ or the algebra of Aceto and Hennessy (1992), where action constants are replaced by action prefix operators. Unlike the case for CCS (Milner 1989), we have sequential composition as a basic operator, not a derived operator. In our view, this is needed in view of the central role of sequential composition in all specification and programming languages. As a result, we need the distinction between successful and unsuccessful termination. We will spend some effort in the presentation of the untimed theory in order to emphasise differences compared with ACP or CSP.

The process algebra SPA adds axioms A4,5,7,8,9,10 in Table 13 to the axioms of Table 1.

The following proposition is easy to prove by structural induction.

Proposition 6.1. Every closed instance of axiom A5 is derivable from the other axioms of SPA.

Proof. In order to prove $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ for all closed SPA-terms x, y, z, we use structural induction on term x. When we prove the equation for a certain triple x, y, z, we

Table 13. *Axioms of SPA ($a \in A$).*

$(x + y) \cdot z = x \cdot z + y \cdot z$	A4	$\delta \cdot x = \delta$	A7
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5	$\epsilon \cdot x = x$	A8
$ax \cdot y = a(x \cdot y)$	A10	$x \cdot \epsilon = x$	A9

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 14. *Deduction rules for SPA ($a \in A$).*

$\dfrac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y}$	$\dfrac{x\downarrow,\ y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'}$	$\dfrac{x\downarrow,\ y\downarrow}{x \cdot y\downarrow}$

can assume as the induction hypothesis that for all subterms x' of x, and for all terms v, w, we have $(x' \cdot v) \cdot w = x' \cdot (v \cdot w)$. Now we have the following case distinction:

1 $x = \delta$. Then $(\delta \cdot y) \cdot z = \delta \cdot z = \delta = \delta \cdot (y \cdot z)$.
2 $x = \epsilon$. Then $(\epsilon \cdot y) \cdot z = y \cdot z = \epsilon \cdot (y \cdot z)$.
3 x has the form of an action prefix, say $x = ax'$ for certain $a \in A$ and subterm x'. Then $(ax' \cdot y) \cdot z = a(x' \cdot y) \cdot z = a((x' \cdot y) \cdot z) = a(x' \cdot (y \cdot z)) = ax' \cdot (y \cdot z)$.
4 x has the form of a sum, say $x = x' + x''$ for certain subterms x', x''. Then $((x' + x'') \cdot y) \cdot z = (x' \cdot y + x'' \cdot y) \cdot z = (x' \cdot y) \cdot z + (x'' \cdot y) \cdot z = x' \cdot (y \cdot z) + x'' \cdot (y \cdot z) = (x' + x'') \cdot (y \cdot z)$.
5 x has the form of a product, say $x = x' \cdot x''$ for certain subterms x', x''. Then $((x' \cdot x'') \cdot y) \cdot z = (x' \cdot (x'' \cdot y)) \cdot z = x' \cdot ((x'' \cdot y) \cdot z) = x' \cdot (x'' \cdot (y \cdot z)) = (x' \cdot x'') \cdot (y \cdot z)$.

□

Thus, associativity of sequential composition (A5) holds in the initial algebra model of the theory. This could be a reason not to include this axiom in our theory. We do so, nevertheless, since it is such a basic result that we will always assume that it holds for all processes. Its status is comparable to the axioms of standard concurrency of Bergstra and Tucker (1984).

The next proposition shows that each closed SPA-term can be reduced to a closed MPA-term, so sequential composition can be eliminated from closed terms. This is the so-called *elimination theorem*. This allows us to use structural induction in proofs and axiomatisations without considering the case of sequential composition. The separation of action prefixing and sequential composition allows easier formulations of the elimination theorem and structural induction.

Being able to reduce each closed term to a term without sequential composition does not imply that sequential composition is not important. In fact, by adding (restricted forms of) recursion we find that the addition of sequential composition adds expressive power to the algebra.

Proposition 6.2. Let t be a closed SPA-term. Then there is a closed MPA-term s such that $\mathrm{SPA} \vdash t = s$.

Proof. We can turn the axioms A3–10 of SPA into rewrite rules, by orienting them from left to right. We obtain a confluent and terminating term rewrite system modulo A1–2. We find s by reducing t to normal form. This proof is like several examples in Baeten and Weijland (1990) and Baeten and Verhoef (1995). □

This proof shows more than the statement of the theorem. From it also follows the fact that SPA is an equational conservative extension of MPA.

Operational rules for sequential composition are given in Table 14.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 15. *Axiom of iteration prefix and sequential composition ($a \in A$).*

$a^*(x \cdot y) = a^* x \cdot y$

Bisimulation equivalence is still a congruence relation on closed SPA-terms, and the theory SPA is sound and complete for the model of transition systems modulo bisimulation, that is, for all closed terms t, s we have

$$\mathrm{SPA} \vdash t = s \quad \Longleftrightarrow \quad t \leftrightarrows s.$$

We can recover the original formulation of the process algebra found in Bergstra and Klop (1984b), Baeten and Weijland (1990) and Baeten and Verhoef (1995) as follows. We define new constants a by the equation $a = a.\epsilon$ for each $a \in A$. Then we reduce the signature by deleting the prefix operators. The subalgebra of the initial algebra that is obtained by this reduced signature is now completely axiomatised by the theory $\mathrm{BPA}_{\delta\epsilon}$ of Baeten and Weijland (1990) and Baeten and Verhoef (1995). Following Baeten and Bergstra (1994), we call $\mathrm{BPA}_{\delta\epsilon}$ an SRM-specification (Subalgebra of Reduced Model Specification) of SPA.

Then we can reduce further by deleting ϵ, or, also, δ, and obtain the SRM specifications BPA_{δ} or BPA, respectively, of Bergstra and Klop (1984b), Baeten and Weijland (1990) and Baeten and Verhoef (1995). This means that all specifications and verifications of systems that have been obtained in recent decades remain valid.

The process algebra SPA extended with prefix iteration is called SPA^*, which is axiomatised by the axioms of SPA, the axioms in Table 9 and in addition the axiom in Table 15. We still obtain a complete axiomatisation of the operational model.

Proposition 6.3. The theory SPA^* is a sound and complete axiomatisation of the model of transition systems modulo bisimulation.

Proof. First, we extend the elimination theorem, showing that each closed SPA^*-term can be reduced to a closed MPA^*-term. Then, we show that all additional axioms are sound, and invoke the corresponding proposition for MPA^*. □

Note that associativity of sequential composition still follows from the other axioms for all closed SPA^*-terms; it is sufficient to add the following case in the induction proof:

$$(a^* x \cdot y) \cdot z = (a^*(x \cdot y)) \cdot z = a^*((x \cdot y) \cdot z) = a^*(x \cdot (y \cdot z)) = a^* x \cdot (y \cdot z)).$$

Extending prefix iteration to full recursion, a lot of things change with respect to the MPA case. However, we do have that the projection theorem is still valid.

Proposition 6.4. Let E be a guarded recursive specification over SPA over variables V. Let p be a solution of E. Then all projections $\pi_n(p)$ can be written as closed MPA-terms, independent of the particular solution.

Proof. In a setting with ϵ, this proof is more difficult than in the standard case, as, for example, in Baeten and Weijland (1990). First, by adding extra variables, we show that

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 16. *Axioms of SPA in relative discrete time ($a \in A$).*

$\underline{\underline{\delta}} \cdot x = \underline{\underline{\delta}}$	A7DR	$\underline{\underline{a}}x \cdot y = \underline{\underline{a}}(x \cdot y)$	A10DR
$\underline{\underline{\epsilon}} \cdot x = x$	A8DR	$\underline{\underline{\sigma}}x \cdot y = \underline{\underline{\sigma}}(x \cdot y)$	DRSEQ
$x \cdot \underline{\underline{\epsilon}} = x$	A9DR		

Table 17. *Deduction rules for sequential composition and time.*

$\dfrac{x \overset{1}{\mapsto} x',\ x \not\Downarrow}{x \cdot y \overset{1}{\mapsto} x' \cdot y}$	$\dfrac{x \downarrow,\ x \overset{1}{\not\mapsto},\ y \overset{1}{\mapsto} y'}{x \cdot y \overset{1}{\mapsto} y'}$	$\dfrac{x \downarrow,\ x \overset{1}{\mapsto} x',\ y \overset{1}{\mapsto} y'}{x \cdot y \overset{1}{\mapsto} x' \cdot y + y'}$

each guarded recursive specification can be brought into a form where each equation has one of the following two forms:

1 $X = \delta + a_1\vec{X}_1 + \cdots + a_n\vec{X}_n$, or
2 $X = \epsilon + a_1\vec{X}_1 + \cdots + a_n\vec{X}_n$,

for certain $n \in \mathbb{N}, a_i \in A, X \in V$ and sequences $\vec{X}_i \in V^*$. By induction on the length of a sequence, we can extend this result so that each sequence of variables can be written in one of these two forms. The projection theorem then follows. □

So, the projection theorem goes through, and thus finite guarded recursion can only define finite branching. However, many other things change. For example, finite unguarded recursion can induce infinite branching. An example is the single equation

$$X = \epsilon + X \cdot a\epsilon,$$

which has summands $a^n\epsilon$ for each $n \in \mathbb{N}$ (Baeten and Weijland 1990).

In SPA, finite guarded recursion allows for the specification of non-regular processes. An example is the counter process C defined as follows:

$$C = T \cdot C \qquad T = pS \qquad S = m\epsilon + T \cdot S.$$

There is no finite guarded specification over MPA that has the counter as a solution, as the counter has infinitely many different states. Thus, we see sequential composition does add expressive power.

A difference from the standard case is that finite guarded recursion allows for the specification of a process with unbounded branching. The following example of this is based on Bosscher (1997):

$$X = aX \cdot Y + aZ \qquad Y = aZ + \epsilon \qquad Z = \epsilon.$$

The transition system for X generated by the rules above has states $Z \cdot Y^n$ with summands $a^i\epsilon$ for each $i \leqslant n$.

Following Vereijken (1997), we can now obtain the discrete relative time extension. Laws A6 and A8 hold only for all processes that can delay an arbitrary amount of time initially, and A9 holds only for processes with any time termination (termination in ϵ). These laws are replaced by law A6DR of Table 3 and the laws A8DR, A9DR in Table 16. Additional operational rules are shown in the first line of Table 17. Use of RSP allows

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 18. *Axioms of renaming and encapsulation operators* ($a \in A, f : A \to A, H \subseteq A$).

$\rho_f(\delta) = \delta$	$\partial_H(\delta) = \delta$
$\rho_f(\epsilon) = \epsilon$	$\partial_H(\epsilon) = \epsilon$
$\rho_f(ax) = f(a)\rho_f(x)$	$\partial_H(ax) = a\partial_H(x)$ if $a \notin H$
	$\partial_H(ax) = \delta$ if $a \in H$
$\rho_f(x \cdot y) = \rho_f(x) \cdot \rho_f(y)$	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$
$\rho_f(x + y) = \rho_f(x) + \rho_f(y)$	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$

Table 19. *Deduction rules for renaming and encapsulation* ($a \in A, f : A \to A, H \subseteq A$).

$\dfrac{x \xrightarrow{a} x'}{\rho_f(x) \xrightarrow{f(a)} \rho_f(x')}$	$\dfrac{x\downarrow}{\rho_f(x)\downarrow}$	$\dfrac{x \overset{1}{\mapsto} x'}{\rho_f(x) \overset{1}{\mapsto} \rho_f(x')}$
$\dfrac{x \xrightarrow{a} x',\ a \notin H}{\partial_H(x) \xrightarrow{a} \partial_H(x')}$	$\dfrac{x\downarrow}{\partial_H(x)\downarrow}$	$\dfrac{x \overset{1}{\mapsto} x'}{\partial_H(x) \overset{1}{\mapsto} \partial_H(x')}$

us to derive several equations such as $\epsilon \cdot \underline{\underline{\delta}} = \delta = \epsilon \cdot \delta$ and $\epsilon \cdot \underline{\underline{a}}x = ax = \epsilon \cdot ax$ and $\epsilon \cdot (x + y) = \epsilon \cdot x + \epsilon \cdot y$.

If we extend our language with the terminated processes $\dot{\delta}, \dot{\epsilon}$ as we indicated at the end of Section 3, the laws A8DR, A9DR have to be discarded again, and $\dot{\epsilon}$ becomes the neutral element of sequential composition.

Note that in the case of absolute timing, we are almost required to include the inconsistent process $\dot{\delta}$, since a process can exhibit inconsistent timing. This happens, for instance, if we follow an a in the second time slice by a b in the first time slice. We say this is equal to an a in the second time slice followed by $\dot{\delta}$.

7. Renaming and encapsulation

We will now consider renaming operators. For each unary function f on A, we have a unary renaming operator ρ_f that renames each action a into $f(a)$, and does nothing else. The axiomatisation in Table 18 follows the structure of SPA-terms, and the deduction rules are given in Table 19. In standard ACP, a stands for both an element of the set of actions A, and the process that executes action a followed by termination. We do not have this overloading, as there is a clear separation between the set of atomic actions A and the action prefix operators. Thus, there can also be no confusion between the renaming *function* f (given as a parameter) and the renaming *operator* ρ_f (in the signature of the algebra). Also, in presentations of standard ACP, an a occurring in an axiom can also stand for δ, but an a occurring in an operational rule cannot. We also avoid this confusion, and this simplifies the deduction rules.

Based on the format of the deduction rules, we can prove as, for example, in Baeten and Verhoef (1995) that the extension with renaming operators is a conservative extension of SPA and that the axiomatisation is sound and complete for the model of transition systems modulo bisimulation. Notice that the axiom $\rho_f(x \cdot y) = \rho_f(x) \cdot \rho_f(y)$ is derivable

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 20. *Axioms of initial time blocking ($a \in A$).*

$v(\underline{\delta}) = \underline{\delta}$	$v(\underline{\sigma}x) = \underline{\delta}$
$v(\underline{\epsilon}) = \underline{\epsilon}$	$v(\underline{\sigma}^* x) = v(x)$
$v(\underline{a}x) = \underline{a}x$	$v(x + y) = v(x) + v(y)$

Table 21. *Deduction rules for initial time blocking ($a \in A$).*

$\dfrac{x \xrightarrow{a} x'}{v(x) \xrightarrow{a} x'}$	$\dfrac{x \downarrow}{v(x) \downarrow}$

from the other axioms for all closed terms (using structural induction on x). In a setting with discrete relative timing, we add underlined variants of the axioms and the additional axiom $\rho_f(\underline{\sigma}x) = \underline{\sigma}\rho_f(x)$, and add the time rule in Table 19. In absolute or dense timing, things work similarly.

We can look at the combination of prefix iteration and renaming. Here, the axioms we have are definitely not complete, and we need the following extra axiom:

$$\rho_f(a^* x) = f(a)^* \rho_f(x).$$

Using this axiom, the renaming operator can be eliminated from all closed terms, and previous results go through.

The renaming operators are clearly distinguished from the blocking or encapsulation operator ∂_H that prohibits execution of actions from H, a subset of A. Nonetheless, axiomatization and deduction rules follow the same pattern (see Tables 18 and 19), and the same results concerning conservativity and completeness can be established. In a setting with discrete relative timing, we again add the axioms with underlining and $\partial_H(\underline{\sigma}x) = \underline{\sigma}\partial_H(x)$, and the time rule in Table 19.

Particularly useful in the further development of the theory with discrete timing is the 'now' operator v, which blocks any initial time step of a process (Baeten and Middelburg 2001; Baeten and Bergstra 1996). Later, we will use this operator in the axiomatisation of silent step with time. The axioms are presented in Table 20, and the operational rules in Table 21.

This 'now' operator is a simple case of the time out operator used in dense time, which blocks any further delay after a delay of t time units. A counterpart of the time out operator is the time initialisation operator, which starts a process after a delay of t. The latter operator plays a special role in the embedding of the relative time in the absolute time theory (Baeten and Middelburg 2001).

8. Parallel composition

We now extend the theory SPA with parallel composition and communication by adding the following signature elements:

— First, we assume that on the set of actions A, we are given a partial binary function γ that is commutative and associative (when defined). For actions $a, b \in A$, if $\gamma(a, b) = c$

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 22. *Axioms of IPA* $(a, b, c \in A)$.

$x \parallel y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x + x \mid y$	MC		
$\delta \lfloor\!\lfloor x = \delta$	LM1	$x \mid y = y \mid x$	CM1
$\epsilon \lfloor\!\lfloor \delta = \delta$	LM2	$\delta \mid x = \delta$	CM2
$\epsilon \lfloor\!\lfloor \epsilon = \epsilon$	LM3	$\epsilon \mid x = \delta$	CM3
$\epsilon \lfloor\!\lfloor ax = \delta$	LM4		
$\epsilon \lfloor\!\lfloor (x + y) = \epsilon \lfloor\!\lfloor x + \epsilon \lfloor\!\lfloor y$	LM5	$ax \mid by = c(x \parallel y)$ if $\gamma(a, b) = c$	CM4
$ax \lfloor\!\lfloor y = a(x \parallel y)$	LM6	$ax \mid by = \delta$ if $\gamma(a, b)$ undefined	CM5
$(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	LM7	$(x + y) \mid z = x \mid z + y \mid z$	CM6

is defined, this means that the simultaneous execution of actions a and b (by parallel components) yields the communication c (between the components). The *communication function* γ is a parameter of the theory.

— We have the binary operator $\parallel$ denoting parallel composition or *merge*. In order to provide a finite axiomatisation, we have the auxiliary operators $\lfloor\!\lfloor$ *left merge* and $\mid$ *communication merge*. The merge axiom MC expresses the fact that $x \parallel y$ can start with a step from x, a step from y or a communication between x and y.

— The left merge $x \lfloor\!\lfloor y$ behaves just like $x \parallel y$, with the restriction that the first step is a step from x. It is also used to express termination (if both components can terminate).

— The communication merge $x \mid y$ behaves just like $x \parallel y$, with the restriction that the first step is a communication step between x and y.

— The *encapsulation operator* ∂_H from Section 7 will block the execution of actions from the set of actions H. It is used to enforce communication between components.

The axioms of the resulting theory IPA (Interaction Process Algebra) consist of the axioms in Table 13, the encapsulation axioms in Table 18 and the axioms in Table 22. If we delete the last summand of axiom MC, the encapsulation axioms and axioms CM1–6, we obtain the theory CPA (Concurrent Process Algebra) containing parallel composition but without communication.

The axiomatisation follows the structure of MPA-terms. For the left-merge, we look at the form of the left-hand argument. When this is ϵ, we have to look at the form of the right-hand argument. The axioms for $\epsilon \lfloor\!\lfloor x$ together code the termination behaviour of merge: a parallel composition can only terminate if both arguments can terminate, that is, both arguments have an ϵ summand. As a result, we have

$$\epsilon \lfloor\!\lfloor x = \begin{cases} \epsilon & \text{if } x = x + \epsilon \\ \delta & \text{otherwise.} \end{cases}$$

An example derivation is $(a\epsilon + \epsilon) \parallel b\epsilon = ab\epsilon + b(a\epsilon + \epsilon)$ (no communication defined). The absence of a summand $b\epsilon$ is necessary to guarantee associativity of the merge (Vrancken 1997) (if we treat an ϵ summand as just an action summand, we see that $(a\epsilon + \epsilon) \parallel (b\epsilon \parallel c\epsilon)$ is not the same as $((a\epsilon + \epsilon) \parallel b\epsilon) \parallel c\epsilon)$.

For the axioms of communication merge, we first assume commutativity. This implies the commutativity of merge, a property that we want to have in any case. Then, we

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 23. *Deduction rules for IPA* $(a,b,c \in A)$.

$\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y}$	$\dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}$	$\dfrac{x\downarrow,\ y\downarrow}{x \parallel y\downarrow}$
$\dfrac{x \xrightarrow{a} x'}{x \lfloor\!\lfloor y \xrightarrow{a} x' \parallel y}$		$\dfrac{x\downarrow,\ y\downarrow}{x \lfloor\!\lfloor y\downarrow}$
$\dfrac{x \xrightarrow{a} x',\ y \xrightarrow{b} y',\ \gamma(a,b)=c}{x \parallel y \xrightarrow{c} x' \parallel y'}$		$\dfrac{x \xrightarrow{a} x',\ y \xrightarrow{b} y',\ \gamma(a,b)=c}{x \mid y \xrightarrow{c} x' \parallel y'}$

have an axiom for when one of the arguments is a constant or a sum. The remaining case is where both arguments have the form of an action prefix, and then we have a case distinction, depending on whether the communication between the two actions is defined. Again, due to our modifications of the standard theory, we have a clear separation between the communication *function* and the communication *operator*. We could have also included an axiom for the associativity of the merge, as we want this to hold in every model we consider. The following proposition shows that associativity holds on the initial algebra.

Proposition 8.1. For all closed IPA-terms x,y,z, we have the following equations (called the axioms of *Standard Concurrency*):

$$x \parallel (y \parallel z) = (x \parallel y) \parallel z$$
$$x \lfloor\!\lfloor (y \parallel z) = (x \lfloor\!\lfloor y) \lfloor\!\lfloor z$$
$$x \mid (y \lfloor\!\lfloor z) = (x \mid y) \lfloor\!\lfloor z$$
$$x \mid (y \mid z) = (x \mid y) \mid z.$$

The proof is quite involved, due to the intricate design of the axiomatisation (a proof is presented in full detail in Vrancken (1997), which can be straightforwardly adapted to the present setting). As a consequence of this proposition, we can formulate an expansion theorem.

The deduction rules for IPA add the rules in Table 23 and the encapsulation rules in Table 19 to the deduction rules of SPA (see Table 14). Deleting the rules in the last line of Table 23 yields the rules for CPA. Due to the absence of the $. \xrightarrow{a} \surd$ predicate, there are fewer deduction rules than for standard PA or ACP.

We have an elimination theorem for both CPA and IPA as follows.

Theorem 8.1. Every closed term over the signature of CPA or IPA is derivably equal to a closed term over MPA.

Using this theorem we can show that axiom CM1 can be derived for all closed terms from the other axioms.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 24. *Axioms of IPA_{drt} ($a \in A$).*

$\underline{\sigma}x \mathbin{\underline{\lfloor\!\lfloor}} (\underline{a}y + z) = \underline{\sigma}x \mathbin{\underline{\lfloor\!\lfloor}} z$	$\underline{\epsilon} \mathbin{\underline{\lfloor\!\lfloor}} \underline{\sigma}x = \underline{\delta}$
$\underline{\sigma}x \mathbin{\underline{\lfloor\!\lfloor}} \underline{\delta} = \underline{\delta}$	
$\underline{\sigma}x \mathbin{\underline{\lfloor\!\lfloor}} \underline{\epsilon} = \underline{\sigma}x$	$\underline{\sigma}x \mid \underline{a}y = \underline{\delta}$
$\underline{\sigma}x \mathbin{\underline{\lfloor\!\lfloor}} \underline{\sigma}y = \underline{\sigma}(x \mathbin{\lfloor\!\lfloor} y)$	$\underline{\sigma}x \mid \underline{\sigma}y = \underline{\sigma}(x \mid y)$
$\underline{\sigma}x \mathbin{\underline{\lfloor\!\lfloor}} (\underline{\epsilon} + \underline{\sigma}y) = \underline{\sigma}(x \mathbin{\lfloor\!\lfloor} y)$	

Table 25. *Deduction rules for IPA_{drt}.*

$\dfrac{x \overset{1}{\mapsto} x',\ y \overset{1}{\mapsto} y'}{x \parallel y \overset{1}{\mapsto} x' \parallel y'}$	$\dfrac{x \overset{1}{\mapsto} x',\ y\downarrow,\ y \overset{1}{\not\mapsto}}{x \parallel y \overset{1}{\mapsto} x'}$	$\dfrac{x\downarrow,\ x \overset{1}{\not\mapsto},\ y \overset{1}{\mapsto} y'}{x \parallel y \overset{1}{\mapsto} y'}$
$\dfrac{x \overset{1}{\mapsto} x',\ y \overset{1}{\mapsto} y'}{x \mathbin{\underline{\lfloor\!\lfloor}} y \overset{1}{\mapsto} x' \mathbin{\underline{\lfloor\!\lfloor}} y'}$	$\dfrac{x \overset{1}{\mapsto} x',\ y\downarrow,\ y \overset{1}{\not\mapsto}}{x \mathbin{\underline{\lfloor\!\lfloor}} y \overset{1}{\mapsto} x'}$	$\dfrac{x \overset{1}{\mapsto} x',\ y \overset{1}{\mapsto} y'}{x \mid y \overset{1}{\mapsto} x' \mid y'}$

Next we can show that CPA and IPA are both equational conservative extensions of SPA, and that the axiomatisations are sound and complete for the model of transition systems modulo bisimulation equivalence.

Theorem 8.2. The equational theories CPA and IPA are equational conservative extensions of the theory SPA.

Theorem 8.3. The axiomatisations of CPA and IPA are sound and complete for the model of transition systems modulo bisimulation equivalence.

We can add prefix iteration to CPA or to IPA, giving a conservative extension of these theories, but then we can no longer obtain a complete axiomatisation: we can reduce the expression $\partial_H(a^*x)$, but have problems with an expression of the form $a^*x \mathbin{\underline{\lfloor\!\lfloor}} y$. We can formulate an expansion theorem, nevertheless, when we extend action prefix iteration to a prefix iteration operator of a *sum* of actions – the so-called *flat* iteration of van Glabbeek (1997).

In the setting with relative discrete timing, we use Vereijken (1997). The deduction rules remain valid, and in the axioms, the constants and action prefixes are replaced by their double underlined variants. In order to axiomatise timing, we add the axioms in Table 24. Deduction rules for timing are shown in Table 25.

A sample derived equation is $\underline{\epsilon} \parallel x = x$. If we add the terminated processes $\dot{\delta}, \dot{\epsilon}$, this equation, and the equations $\underline{a}x \mathbin{\underline{\lfloor\!\lfloor}} y = \underline{a}(x \parallel y)$ and $\underline{\epsilon} \mathbin{\underline{\lfloor\!\lfloor}} \underline{\sigma}x = \underline{\delta}$ are no longer valid in full generality, and have to be adapted. Also, the three first action rules in Table 23 are no longer valid in full generality, and have to be adapted. For more details, see Baeten and Middelburg (2001).

Things get more complicated, but not very much so, in an absolute time or dense time setting. Many details can be found in Baeten and Middelburg (2001).

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 26. *Axiom of silent step ($a \in A$).*

$a(\tau(x+y)+x) = a(x+y)$	B

9. Silent step

In this section we consider the silent step τ. We choose to treat the silent step as an action, so we take $\tau \in A$. This means we have the action prefix τx as a particular prefix operator. This emphasises the fact that τ is an action (whose execution cannot be observed), and as such has nothing to do with (some form of) termination – τ and ϵ can be clearly distinguished. The process $\tau\epsilon$ will terminate without executing a visible action, the process $\tau\delta$ can be called deadlock: without executing a visible action, a state will be reached where the process is stuck.

As the semantical treatment of the silent step we choose rooted branching bisimulation (van Glabbeek and Weijland 1996) rather than Milner's original weak bisimulation (see, for example, Milner (1989)), as the former is closer to an action interpretation, since all axioms of IPA that hold for all actions also hold for τ.

In a setting with ϵ, there is only one extra axiom for τ, the branching axiom B shown in Table 26. Taking $x = \delta$, we obtain $a\tau y = ay$ (in particular, $\tau\tau y = \tau y$), so every τ-step that is not the first step and is not part of a sum can be removed.

The theories MPA_τ, SPA_τ, CPA_τ, IPA_τ are obtained from the theories MPA, SPA, CPA and IPA, respectively, by having a special element $\tau \in A$ and adding axiom B. We can obtain an elimination theorem as before – all closed terms can be reduced to MPA-terms. In the semantics, as we are using branching bisimulation, we cannot capture the special behaviour of the silent step in terms of deduction rules (this is possible in weak bisimulation (van Glabbeek 1987)). Rather, we have to divide out a different equivalence relation on the transition systems generated by the rules we have defined previously.

Definition 9.1. For closed terms s, t, we define $s \Rightarrow t$ if t can be reached from s by doing a number of τ-steps (0 or more). Moreover, we put $s \stackrel{(a)}{\rightarrow} t$ if either $a = \tau$ and $s = t$ or $s \stackrel{a}{\rightarrow} t$.

Then, let R be a binary *symmetric* relation on closed terms. We say R is a *branching bisimulation* if the following holds:

— Whenever $R(x, y)$ and $x \stackrel{a}{\rightarrow} x'$, there are terms y'', y' such that $y \Rightarrow y'' \stackrel{(a)}{\rightarrow} y'$ and $R(x, y'')$ and $R(x', y')$.
— Whenever $R(x, y)$ and $x \downarrow$, there is a term y'' such that $y \Rightarrow y'' \downarrow$ and $R(x, y'')$.

If R is a branching bisimulation relating terms s, t, we say s, t satisfy the *root condition* (for R) if the following holds:

— Whenever $s \stackrel{a}{\rightarrow} x$, there is a term y such that $t \stackrel{a}{\rightarrow} y$ and $R(x, y)$.
— Whenever $t \stackrel{a}{\rightarrow} y$, there is a term x such that $s \stackrel{a}{\rightarrow} x$ and $R(x, y)$.
— $s \downarrow$ iff $t \downarrow$.

We say two closed terms t, s are *rooted branching bisimulation equivalent* or *rooted branching bimilar*, notation $t \leftrightarrows_{rb} s$, if there is a branching bisimulation R with $R(s, t)$ that satisfies the root condition for s, t.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 27. *Axiom for divergence.*

$\tau\tau^* x = \tau x$	FI

We can prove that rooted branching bisimulation equivalence is a congruence relation on IPA_τ-terms. We obtain models with complete axiomatisations.

Theorem 9.1. Let X be one of the theories $MPA_\tau, SPA_\tau, CPA_\tau, IPA_\tau$, and let s, t be closed X-terms.

Then $X \vdash s = t \iff s \leftrightarrows_{rb} t$.

Proof. The proof follows from van Glabbeek and Weijland (1996), see also Baeten and Weijland (1990). □

Next, we consider the other extensions. First, we cannot maintain the axioms of the projection operators. Since sequences of τ's can be compressed, silent steps have no depth, and we cannot count them in projections. Thus, the axiom $\pi_{n+1}(ax) = a\pi_n(x)$ can only hold for $a \neq \tau$. For τ, we have to put the following equation:

$$\pi_n(\tau x) = \tau\pi_n(x).$$

When we consider renamings, we have to keep τ fixed, that is, we can only allow for renaming functions $f : A \to A$ that satisfy $f(\tau) = \tau$. There is one renaming operator that is particularly useful – the *abstraction operator*, usually denoted τ_I, which renames all actions from $I \subseteq A$ into τ, and is based on the following function f:

$$f(a) = \begin{cases} \tau & \text{if } a \in I \\ a & \text{otherwise.} \end{cases}$$

Considering prefix iteration, we get the following. The τ prefix iteration can be called the *divergence* prefix, and we can formulate the *fair iteration* axiom FI. See Table 27. This axiom is a slight variation of the fair iteration axiom given in Bergstra *et al.* (1994) (we adapt their axiom to the setting of branching bisimulation congruence). The axiom can also be found in Aceto *et al.* (1996). The divergence prefix comes closest in the present setting to the delay prefix of Hennessy (1981), Milner (1983) and Hennessy (1984).

Note that the fair iteration axiom is equivalent to $\tau^* x = \tau x + x$. It is valid on the model of transition systems modulo rooted branching bisimulation equivalence. Note that this is a very compact form of the fairness principle, as compared with a rule like KFAR in Baeten and Weijland (1990).

The time extension is based on Baeten *et al.* (2000). Operationally, we have the obvious generalisation of branching bisimulation (where we have to keep in mind that the root condition must also hold after initial time steps), and we show the axioms in Table 28. We do not consider terminated processes here – Baeten *et al.* (2000) includes the constant $\dot{\delta}$. Note that the axiomatisation is a bit awkward due to time determinism: the law

$$\underline{a}(\underline{\tau}(\underline{\underline{\sigma b}} + \underline{\underline{\sigma c}}) + \underline{\underline{\sigma b}}) = \underline{a}(\underline{\underline{\sigma b}} + \underline{\underline{\sigma c}})$$

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Table 28. *Axioms of silent step and time ($a \in A$).*

$\tau x = \underline{\underline{\sigma}}^* \underline{\underline{\tau}} x$	$\underline{\underline{a}}(\underline{\underline{\sigma}}\underline{\underline{\tau}} x + v(y)) = \underline{\underline{a}}(\underline{\underline{\sigma}} x + v(y))$
$\underline{\underline{a}}(\underline{\underline{\tau}}(v(x) + y) + v(x)) = \underline{\underline{a}}(v(x) + y)$	$\underline{\underline{a}}(\tau \underline{\underline{\sigma}}^* v(x + y) + \underline{\underline{\sigma}}^* v(x)) = \underline{\underline{a\sigma}}^* v(x + y)$
$\underline{\underline{a}}(\underline{\underline{\tau}}(v(x) + y) + y) = \underline{\underline{a}}(v(x) + y)$	

does not hold, since on the left-hand side, the choice to do b and not c in the next time slice can be made by passing to the next time slice, whereas on the right-hand side, the choice not to do c in the next time slice can only be made by executing b in the next time slice. For further details, refer to Baeten *et al.* (2000).

In an absolute time or dense time setting, the theory with abstraction is not fully worked out yet. This will be the topic of ongoing research.

10. Related work

Closest in syntax elements to the work presented here is the work on CSP and its timed extensions, see, for example, Reed and Roscoe (1986) and Reed and Roscoe (1988). A very nice recent textbook is Schneider (2000). Timed CSP has action and time prefixing, and sequential composition. The main difference is the use of failure-divergences semantics instead of bisimulation semantics. In combination with an extra internal choice operator, this allows one to use extra internal steps freely. Strong time factorisation is used in combination with an explicit time out operator. Timing is dense and relative, and no other variants are considered.

There are many timed extensions of CCS, see Wang (1990), Moller and Tofts (1990), Moller and Tofts (1991), Hennessy and Regan (1995), Hansson (1991), Chen (1992), Jeffrey (1992), Daniels (1991) and Fidge (1993). The first four of these are compared in Corradini *et al.* (1999). All these theories use action and time prefixing, but do not have sequential composition as a basic operator, and only one mode of termination. Some are dense time, some are discrete time (but never both!), and all are relative time (none are absolute time). Usually there is strong time factorisation (occasionally, there are two choice operators).

Very similar in nature is the timed process algebra ATP (Nicollin *et al.* 1990; Nicollin and Sifakis 1994). Here, also, we have action and time prefixing, one mode of termination and no sequential composition. We have discrete relative timing only, strong time factorisation, and explicit time out operators.

There are also many timed extensions of LOTOS, see Quemada *et al.* (1993), Bolognesi and Lucidi (1992), Leduc (1992), Leduc and Leonard (1993), Davies *et al.* (1996), Miguel *et al.* (1993) and Nakata *et al.* (1994). An overview is given in Bolognesi *et al.* (1994). They all have action prefixing, and either time-stamped actions or time prefixing. Most are discrete time, some are dense time, but none are both. All, except the last mentioned, are relative time only; the last mentioned is absolute time only. Furthermore, there are operators like in some timed versions of CCS or in ATP. We do find successful termination in some of these process algebras.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Other timed process algebras we can mention are: a dense time algebra where actions have a duration in Murphy (1989); a dense time algebra related to CCS in Cerans (1995); a dense time algebra that can express sharing of resources in Bremond-Gregoire *et al.* (1993); a generalisation of a number of other timed process algebras in Jeffrey (1991); a timed algebra related to ATP with multiple clocks in Andersen and Mendler (1994); and, finally, perhaps the earliest timed process algebra in Cardelli (1982), which uses dense time and timestamping.

More extensive overviews of related work can be found in Nicollin and Sifakis (1992), Klusener (1993) and Vereijken (1997). A general remark that we want to emphasise yet again is that there is hardly any work on embedding one variant of process algebra into another, for example, untimed into timed, relative time into absolute time, or discrete time into dense time. One exception (apart from our own work mentioned earlier) is Corradini (2000), where a translation is given from an absolute time theory to a relative time theory. Both are only discrete time, however. The present paper contributes to the work on interrelating theories.

11. Conclusion

We have presented a redesign of ACP-style process algebra, where action execution and termination are separated. This allows for an improved embedding of untimed into timed process algebra.

The separation of action execution and termination also entails better separation of atomic actions as a parameter of the theory and as signature elements. We can define a minimal process algebra without sequential composition, and this eases formulation of concepts such as structural induction, linearity, elimination and guardedness. The difference between the silent step τ and the empty step ϵ becomes clearer, as τ becomes an action prefix operator and ϵ a termination constant.

In the operational semantics, we have no need for separate terminating action executions. We have a natural restriction of iteration to action prefix iteration, allowing complete axiomatisations in more cases. Moreover, we can formulate time iteration as a form of prefix iteration, and we can formulate divergent behaviour as a form of prefix iteration. With this, we get a concise expression of fair iteration.

Acknowledgements

The author wishes to acknowledge suggestions and helpful remarks by the anonymous referees, and Jan Bergstra and Alban Ponse (University of Amsterdam), Leszek Holenderski, Sjouke Mauw, Kees Middelburg, Michel Reniers, Jeroen Voeten, Marc Voorhoeve and Tim Willemse (all of Eindhoven University of Technology), and Kim Larsen (BRICS Denmark). Special thanks also go to Luca Aceto, who played an important role in realising this paper.

References

Aceto, L., Bloom, B. and Vaandrager, F.W. (1994) Turning SOS rules into equations. *Information and Computation* **111** (1) 1–52.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Aceto, L., Fokkink, W. J., van Glabbeek, R. J. and Ingólfsdóttir, A. (1996) Axiomatizing prefix iteration with silent steps. *Information and Computation* **127** (1) 26–40.

Aceto, L. and Groote, J. F. (1999) A complete equational axiomatization for MPA with string iteration. *Theoretical Computer Science* **211** (1/2) 339–374.

Aceto, L. and Hennessy, M. (1992) Termination, deadlock and divergence. *Journal of the ACM* **39** 147–187.

Aceto, L. and Ingólfsdóttir, A. (1996) An equational axiomatisation of observation congruence for prefix iteration. In: *Proceedings AMAST'96*, 195–209.

Andersen, H. R. and Mendler, M. (1994) An asynchronous process algebra with multiple clocks. In: Sannella, D. (ed.) Proceedings ESOP'94. *Springer-Verlag Lecture Notes in Computer Science* **788** 58–73.

Baeten, J. C. M. and Bergstra, J. A. (1991) Real time process algebra. *Formal Aspects of Computing* **3** (2) 142–188.

Baeten, J. C. M. and Bergstra, J. A. (1994) On sequential composition, action prefixes and process prefix. *Formal Aspects of Computing* **6** (3) 250–268.

Baeten, J. C. M. and Bergstra, J. A. (1996) Discrete time process algebra. *Formal Aspects of Computing* **8** (2) 188–208.

Baeten, J. C. M., Bergstra, J. A. and Reniers, M. A. (2000) Discrete time process algebra with silent step. In: Plotkin, G., Stirling, C. and Tofte, M. (eds.) *Proof, Language, and Interaction (Essays in Honour of Robin Milner)*, Foundations of Computing, MIT Press, Chapter 18, 535–569.

Baeten, J. C. M. and Middelburg, C. A. (2001) Process algebra with timing: Real time and discrete time. In: Bergstra, J. A., Ponse, A. and Smolka, S. A. (eds.) *Handbook of Process Algebra*, Elsevier Science 627–684.

Baeten, J. C. M. and Reniers, M. A. (2000) Termination in timed process algebra. Technical Report CSR 00/13, Technische Universiteit Eindhoven, Computer Science Division. (See `http://www.win.tue.nl/fm/pubbaeten.html`.)

Baeten, J. C. M. and Verhoef, C. (1995) Concrete process algebra. In: Abramsky, S., Gabbay, D. M. and Maibaum, T. S. E. (eds.) *Handbook of Logic in Computer Science*, Volume **4**, Oxford University Press 149–269.

Baeten, J. C. M. and Weijland, W. P. (1990) *Process Algebra*, Cambridge Tracts in Theoretical Computer Science **18**, Cambridge University Press.

Bergstra, J. A., Bethke, I. and Ponse, A. (1994) Process algebra with iteration and nesting. *The Computer Journal* **37** (4) 243–258.

Bergstra, J. A. and Klop, J. W. (1984a) The algebra of recursively defined processes and the algebra of regular processes. In: Paredaens, J. (ed.) Proceedings 11th ICALP. *Springer-Verlag Lecture Notes in Computer Science* **172** 82–95.

Bergstra, J. A. and Klop, J. W. (1984b) Process algebra for synchronous communication. *Information and Control* **60** (1/3) 109–137.

Bergstra, J. A., Klop, J. W. and Olderog, E.-R. (1987) Failures without chaos: a new process semantics for fair abstraction. In: Wirsing, M. (ed.) *Proc. IFIP Conf. on Formal Description of Programming Concepts – III*, Ebberup, Denmark, North-Holland 77–103.

Bergstra, J. A. and Tucker, J. V. (1984) Top down design and the algebra of communicating processes. *Science of Computer Programming* **5** 171–199.

Bolognesi, T. and Lucidi, F. (1992) LOTOS-like process algebras with urgent or timed interactions. In: Parker, K. R. and Rose, G. A. (eds.) *Proceedings FORTE'91*, North-Holland 249–264.

Bolognesi, T., Lucidi, F. and Trigila, S. (1994) Converging towards a timed LOTOS standard. *Computer Standards and Interfaces* **16** (2) 87–118.

Bosscher, D. J. B. (1997) *Grammars Modulo Bisimulation*, Ph.D. thesis, University of Amsterdam.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Bremond-Gregoire, P., Lee, I. and Gerber, R. (1993) ACSR: an algebra of communicating shared resources with dense time and priorities. In: E. Best (ed.) Proceedings CONCUR'93. *Springer-Verlag Lecture Notes in Computer Science* **715** 417–431.

Brinksma, E. (ed.) (1989) Information Processing Systems, Open Systems Interconnection, LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. *International Standard* **IS-8807**, ISO, Geneva.

Cardelli, L. (1982) Real time agents. In: Nielsen, M. and Schmidt, E. M. (eds.) Proceedings ICALP'82. *Springer-Verlag Lecture Notes in Computer Science* **140** 94–106.

Cerans, K. (1995) A calculus of timed refinement. In: Lee, I. and Smolka, S. (eds.) Proceedings CONCUR'95. *Springer-Verlag Lecture Notes in Computer Science* **962** 516–530.

Chen, L. (1992) *Timed Processes: Models, Axioms and Decidability*, Ph.D. thesis, University of Edinburgh.

Corradini, F. (2000) Absolute versus relative time in process algebras. *Information and Computation* **156** (1) 122–172.

Corradini, F., D'Ortenzio, D. and Inverardi, P. (1999) On the relationships among four timed process algebras. *Fundamenta Informaticae* **38** (4) 377–395.

Daniels, M. (1991) Modeling real-time behavior with an interval time calculus. In: J. Vytopil (ed.) Proceedings FTRTFTS'92. *Springer-Verlag Lecture Notes in Computer Science* **571** 53–71.

Davies, J., Bryans, J. W. and Schneider, S. A. (1996) Real-time LOTOS and timed observations. In: von Bochmann, G., Dssouli, R. and Rafiq, O. (eds.) *Proceedings FORTE'95*, North-Holland 383–397.

Fidge, C. (1993) A constraint oriented real-time process calculus. In: Diaz, M. and Groz, R. (eds.) *Proceedings FORTE'92*, North-Holland 363–378.

Fokkink, W. J. (1994) A complete equational axiomatisation for prefix iteration. *Information Processing Letters* **52** (6) 333–337.

van Glabbeek, R. J. (1987) Bounded nondeterminism and the approximation induction principle in process algebra. In: Brandenburg, F. J., Vidal-Naquet, G. and Wirsing, M. (eds.) *Proceedings STACS'87*. *Springer-Verlag Lecture Notes in Computer Science* **247** 336–347.

van Glabbeek, R. J. (1997) Axiomatizing flat iteration. In: Mazurkiewicz, A. and Winkowski, J. (eds.) Proceedings CONCUR'97. *Springer-Verlag Lecture Notes in Computer Science* **1243** 228–242.

van Glabbeek, R. J. and Weijland, W. P. (1996) Branching time and abstraction in bisimulation semantics. *Journal of the ACM* **43** (3) 555–600.

Hansson, H. (1991) *A Calculus for Communicating Systems with Time and Probabilities*, Ph.D. thesis, University of Uppsala.

Hennessy, M. (1981) A term model for synchronous processes. *Information and Control* **51** 58–75.

Hennessy, M. (1984) Axiomatising finite delay operators. *Acta Informaticae* **21** 61–88.

Hennessy, M. and Regan, T. (1995) A process algebra for timed systems. *Information and Computation* **117** 221–239.

Hoare, C. A. R. (1985) *Communicating Sequential Processes*, Prentice Hall.

Jeffrey, A. S. (1991) Abstract timed observation and process algebra. In: Baeten, J. C. M. and Groote, J. F. (eds.) Proceedings CONCUR'91. *Springer-Verlag Lecture Notes in Computer Science* **527** 332–345.

Jeffrey, A. S. (1992) A linear time process algebra. In: Larsen, K. G. and Skou, A. (eds.) Proceedings CAV'91. *Springer-Verlag Lecture Notes in Computer Science* **575** 433–442.

Klusener, A. S. (1993) *Models and Axioms for a Fragment of Real Time Process Algebra*, Ph.D. thesis, Technische Universiteit Eindhoven.

Leduc, G. (1992) An upward compatible timed extension to LOTOS. In: Parker, K. R. and Rose, G. A. (eds.) *Proceedings FORTE'91*, North-Holland 217–232.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press

Leduc, G. and Leonard, L. (1993) A timed LOTOS supporting a dense time domain and including new timed operators. In: Diaz, M. and Groz, R. (eds.) *Proceedings FORTE'92*, North-Holland 87–102.

Miguel, C., Fernandez, A. and Vidaller, L. (1993) Extending LOTOS towards performance evaluation. In: Diaz, M. and Groz, R. (eds.) *Proceedings FORTE'92*, North-Holland 103–117.

Milner, R. (1983) Calculi for synchrony and asynchrony. *Theoretical Computer Science* **25** 267–310.

Milner, R. (1989) *Communication and Concurrency*, Prentice Hall.

Moller, F. and Tofts, C. (1990) A temporal calculus of communicating systems. In: Baeten, J. C. M. and Klop, J. W. (eds.) Proceedings CONCUR'90. *Springer-Verlag Lecture Notes in Computer Science* **458** 401–415.

Moller, F. and Tofts, C. (1991) Relating processes with respect to speed. In: Baeten, J. C. M. and Groote, J. F. (eds.) Proceedings CONCUR'91. *Springer-Verlag Lecture Notes in Computer Science* **527** 424–438.

Murphy, D. J. V. (1989) *Time, Causality, and Concurrency*, Ph.D. thesis, University of Surrey.

Nakata, A., Higashino, T. and Taniguchi, K. (1994) LOTOS enhancement to specify time constraint among non-adjacent actions using first order logic. In: Tenney, R. L., Amer, P. D. and Uyar, M. U. (eds.) *Proceedings FORTE'93*, North-Holland 451–466.

Nicollin, X., Richier, J.-L., Sifakis, J. and Voiron, J. (1990) ATP: an algebra for timed processes. In: *Proceedings IFIP TC2 Conference on Programming Concepts and Methods*, Sea of Gallilee, Israel.

Nicollin, X. and Sifakis, J. (1992) An overview and synthesis on timed process algebras. In: Larsen, K. G. and Skou, A. (eds.) Proceedings CAV'91. *Springer-Verlag Lecture Notes in Computer Science* **575** 376–398.

Nicollin, X. and Sifakis, J. (1994) The algebra of timed processes ATP: Theory and application. *Information and Computation* **114** 131–178.

Park, D. M. R. (1981) Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) Proceedings 5th GI Conference. *Springer-Verlag Lecture Notes in Computer Science* **104** 167–183.

Plotkin, G. D. (1983) An operational semantics for CSP. In: Bjørner, D. (ed.) *Proceedings Conference on Formal Description of Programming Concepts II, Amsterdam*, North-Holland 199–225.

Quemada, J., de Frutos, D. and Azcorra, A. (1993) TIC: A timed calculus. *Formal Aspects of Computing* **5** (3) 224–252.

Reed, G. M. and Roscoe, A. W. (1986) A timed model for communicating sequential processes. In: Proceedings 13th ICALP. *Springer-Verlag Lecture Notes in Computer Science* **226**.

Reed, G. M. and Roscoe, A. W. (1988) A timed model for communicating sequential processes. *Theoretical Computer Science* **58** 249–261.

Schneider, S. A. (2000) *Concurrent and Real-time Systems (the CSP Approach)*, Worldwide Series in Computer Science, Wiley.

Sewell, P. (1997) Nonaxiomatisability of equivalences over finite state processes. *Annals of Pure and Applied Logic* **90** 163–191.

Vereijken, J. J. (1997) *Discrete-Time Process Algebra*, Ph.D. thesis, Technische Universiteit Eindhoven.

Vrancken, J. L. M. (1997) The algebra of communicating processes with empty process. *Theoretical Computer Science* **177** (2) 287–328.

Wang, Y. (1990) Real-time behaviour of asynchronous agents. In: Baeten, J. C. M. and Klop, J. W. (eds.) Proceedings CONCUR'90. *Springer-Verlag Lecture Notes in Computer Science* **458** 502–520.

https://doi.org/10.1017/S0960129503004006 Published online by Cambridge University Press