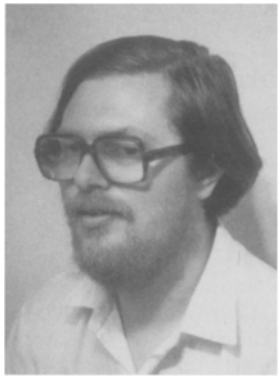
Distributed Computing (1989) 3:130–145

DISTRIBUTED COMPUTING

Step semantics for "true" concurrency with recursion

J.-J.C. Meyer and E.P. de Vink

Department of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1081, NL-1081 HV Amsterdam, The Netherlands



John-Jules Meyer received his Master's degree in Mathematics in 1979 from the University of Leiden, and his Ph.D. degree in 1985 from the Free University Amsterdam. He is currently a Professor of Theoretical Computer Science, both at the Free University Amsterdam and at the University of Nijmegen. His current research interests include semantics of programming languages and logics for computer science, in particular artifical intelligence.

Erik de Vink received the M.S. degree in Mathematics from the University of Amsterdam. He is currently a Junior Researcher at the Department of Mathematics and Computer Science of the Free University Amsterdam. At the moment his main research concerns the semantics of concurrent and logic programming languages.

Abstract. We present a variety of denotational linear time semantics for a language with recursion and "true" concurrency in a form of synchronous co-operation, which in the literature is known as step semantics. We show that this can be done by a generalization of known results for interleaving semantics. A general method is presented to define semantical operators and denotational semantics in the Smyth powerdomain of streams. With this method, first a naive and then more sophisticated semantics for synchronous co-operation are developed, which include such features as interleaving and synchronization. Then we refine the semantics to deal with a bounded number of processors, subatomic actions, maximal parallelism and a real-time operator. Finally, it is indicated how to apply these ideas to branching-time models, where it becomes possible to analyze deadlock behaviour as well as a form of "true" concurrency.

Key words: Denotational semantics – True concurrency – Smyth powerdomain of streams

Offprint requests to: J.-J.C. Meyer

1 Introduction

Much work has been done in the field of denotational semantics of languages with parallel operators and recursion where this parallelism is modeled by means of interleaving. (See de Bakker et al. (1986) for an overview.) This idea of using the interleaving model is motivated by two reasons: (i) Interleaving can be seen as scheduling the "parallel" components on *one* processing unit. (ii) Theoretically, interleaving reduces parallelism to nondeterminism, a notion well-understood in semantic theory (e.g., de Nicola and Hennessy 1987).

On the other hand, however, workers in other (theoretical) frameworks such as Petri nets, have always stressed the importance of being able to distinguish between non-determinism and what is called "true" concurrency (cf., Reisig 1985).

In this paper we present a denotational semantics in the style of e.g., de Bakker et al. (1986) for a language with recursion and a form of "true"

concurrency which is very close to what is called *step semantics* in Taubner and Vogler (1987). In fact, our proposal is a direct generalization of what is known for the interleaving model and can be viewed as a "non-standard" model of Bergstra and Klop's process algebras for interleaving semantics, (see Bergstra and Klop 1984). This generalized model can be thought of as containing a number of processors, all executing in parallel some parts of a program. In the program it is explicitly given what has to be executed in parallel. At any moment the number of active processors is assumed to be finite. (It is not very realistic to consider this otherwise.) However, in principle we assume that a finite but unbounded number of processors is available. (In Sect. 4 where we treat refinements of our semantics we consider how to deal with a bounded number of available processors.)

The semantics of a statement gives the set of possible schedulings for the parallel (concurrent) execution of this statement. In our model we assume the following: (i) There is a global clock. (ii) All processors co-operate synchronously, like in a systolic system. (iii) All atomic actions take *one* tick of the global clock to be executed. (We shall show how to generalize this in Sect. 4 again.)

Of course, these assumptions do not say anything about how to schedule the atomic actions of a program on the processors; we still have freedom to make choices concerning this and we shall investigate some of the alternatives that are possible here.

The main point of this paper is to show that we can still use the same techniques and concepts as for interleaving – including the Smyth powerdomain and techniques based on the Smyth ordering – once it is decided upon how to model the "truly" concurrent execution of two actions. In fact, we discuss several ways of modeling this behaviour, both using the idea of collecting concurrent actions in a *multi-set* or *bag* just as nondeterministic actions are collected in sets. (Naturally, we have to discriminate strictly between the multisets modeling concurrency and the sets representing nondeterminism.)

As in the linear time semantics for interleaving we use *streams* and sets of streams (cf., Broy 1986; Meyer 1985). In this setting, however, we use streams of *multi-sets* of elementary (or atomic) actions rather than streams of just elementary actions. The idea behind this is the following: streams record sequences (histories) of the actions that are (or have been) executed. If we allow truly concurrent actions we have to record sequences of "action packages" that are thought of as being executed together (simultaneously). This being understood we have to decide how the simultaneous performance of complex (i.e., nonelementary) actions are denoted in terms of these streams of action packages. For this we have two obvious choices: (i) a naive one, where it is assumed that two concurrent actions start at the same time; (ii) a more sophisticated one, where it is allowed that two concurrent actions may arbitrarily start in parallel: perhaps one of them starts before the other, and possibly suspends at some moment the execution of the next action for a while.

At first sight the second choice seems slightly awkward to work out, while the first choice is rather easy. However, the second choice of modeling can be elaborated using general techniques that were developed in the framework of *interleaving* semantics (cf., Meyer and de Vink 1987). Furthermore, in the second model of synchronous co-operation it is possible to include special *synchronization* primitives in order to obtain a denotational semantics for a much more realistic language, in which one can control the way co-operation is synchronized. This appears to work just as in the case of interleaving.

Next we investigate somewhat more refined issues, such as a multi-processor semantics for a *bounded* number of processors, a semantic model that deals with atomic actions that may take more than one tick of the clock to be executed, we discuss how *maximal parallelism* can be incorporated in our models and comment on *real-time* aspects involving a delay operator.

Finally, we indicate how to generalize the ideas expounded in the Sects. 3 and 4 to branching time semantics in the style of de Bakker and Zucker (1982), where it becomes possible to analyze *deadlock* as well as a form of "true" concurrency. We view this as an attempt to synthesize the metric approach of de Bakker and Zucker (1982) with approaches such as Taubner and Vogler (1987) and Aceto et al. (1987).

We appreciate that the present proposal is only a first attempt to obtain the full sophistication of true concurrency. Our model of concurrency is an extension of the models of de Bakker et al. for interleaving semantics, stretched as far as possible towards "true" concurrency. It is in between the interleaving model and "partial order" semantics for concurrency as it appears in the literature. In fact, since our model has an implicit notion of time (timing), it may be viewed as a manner of shaping the general partial order model into this built-in timing mechanism, thus rendering a more concrete model. Concrete in the sense that when two state-

ments s_1 and s_2 are to be executed in parallel, their mutual independence is reflected in our model by means of a set of specific schedulings on a multiprocessor of the atomic actions that constitute the statements s_1 and s_2. This seems to be in agreement with the results in Aceto et al. (1987), where the relationship between these three models of true concurrency is investigated in detail.

As we said before, our model has a close relationship with the step (failure) semantics of Taubner and Vogler (1987). Although we do not deal with failure sets we believe that this is not essential for our approach. We expect that our framework can be easily generalized to a failure (or ready set) model. We have chosen not to do this in order not to obscure the main issue of this paper, which is a rigorous treatment of *recursion* in the context of step semantics by means of fixed-point theory. Moreover, the emphasis in this paper is on a uniform method to prove the continuity of variations of parallel operators in this framework.

It would, of course, be interesting to investigate more precise relationships between our approach and other approaches to true concurrency, such as Petri nets (cf. e.g., Reisig 1985), Mazurkiewicz trace theory (Mazurkiewicz 1978), Winskel's (labeled) event structures (Winskel 1980) and subset and multiset languages (cf. e.g., Rozenberg and Verraedt 1983; Janicki 1987). We would like to mention van Glabbeek and Vaandrager (1987) in this context, where an interesting attempt is made to treat Petri nets by means of process algebra, and Degano et al. (1987), where a partially successful translation is given from CCS to a certain class of Petri nets. Also the recent work of Boudol and Castellani (Boudol and Castellani 1987a; Boudol and Castellani 1987b) must be mentioned here: in these papers an operational semantics based on a Plotkin-style transition system, is proposed such that sequentiality, non-determinism and concurrency are properly distinguished, using a synthesis of Milner's CCS and Winskel's event structures. These attempts of synthesizing superficially entirely different frameworks are of paramount importance in order to understand the full complexity of "true" concurrency.

2 Mathematical preliminaries

In this section we present some syntactial and semantical preliminaries that we shall need in the sequel of our paper. We start with the definition of our two languages. Then we introduce the domain of streams $\mathscr{B}^{st}$ and the powerdomain of compact stream sets $\mathscr{P}^*(\mathscr{B}^{st})$. We proceed with a general method to construct continuous functions on compact stream sets. We conclude this section with a scheme for denotational semantics for uniform concurrency with recursion, where the interpretation of the concurrency operator can still be varied.

2.1. Definition. Fix an alphabet $\mathscr{A}$ of atomic actions, an alphabet $\mathscr{C}$ of synchronization actions and a set Ξ of program variables, with typical elements a, c and ξ, respectively.

(i) The language $\mathscr{L}_0$ is given by BNF:
$s ::= a \mid s_1 ; s_2 \mid s_1 \cup s_2 \mid s_1 \| s_2 \mid \xi \mid \mu\xi[s]$.

(ii) The language $\mathscr{L}_1$ is given by BNF:
$s ::= a \mid c \mid s_1 ; s_2 \mid s_1 \cup s_2 \mid s_1 \| s_2 \mid \xi \mid \mu\xi[s]$.

This syntax for $\mathscr{L}_0$ and $\mathscr{L}_1$ is widely used in papers such as de Bakker et al. (1986) and de Bakker and Meyer (1987) as a core language for (uniform) concurrency with recursion. In the present paper the sets $\mathscr{A}$ and $\mathscr{C}$ are not assumed to be finite. The language $\mathscr{L}_0$ contains atomic actions in $\mathscr{A}$ (which remain uninterpreted), sequential composition, nondeterministic choice also known as local nondeterminism (Francez et al. 1979), a parallel composition, variables and the recursive μ-construct. The language $\mathscr{L}_1$ is an extension of $\mathscr{L}_0$ with synchronization actions in $\mathscr{C}$. We refer with the term *elementary action* to either an atomic action or a synchronization action. We distinguish $\tau \in \mathscr{A}$. Further, we stipulate a bijection $\bar{}: \mathscr{C} \longrightarrow \mathscr{C}$ which for every $c \in \mathscr{C}$ yields a matching synchronization action $\bar{c} \in \mathscr{C}$, and such that $\bar{\bar{c}} = c$ for all $c \in \mathscr{C}$. (Cf., Milner 1980.)

In the papers mentioned above the parallel composition operator is interpreted in the interleaving model. The intention of this paper is to vary the interpretation of this parallel operator in order to deal with "true" concurrency.

Next we present an overview of the basic definitions and facts regarding the domain of stream sets and the Smyth powerdomain that we shall employ in this paper. On the latter we shall base our semantic definitions.

Let $\mathscr{B}_0 = \{B: \mathscr{A} \longrightarrow \mathbb{N} \mid B(a) \neq 0$ for a finite and positive number of $a \in \mathscr{A}\}$ stand for the collection of all finite non-empty multi-sets or bags over $\mathscr{A}$. Let $\mathscr{B}_1 = \{B: \mathscr{A} \cup \mathscr{C} \longrightarrow \mathbb{N} \mid B(e) \neq 0$ for a finite and positive number of $e \in \mathscr{A} \cup \mathscr{C}\}$ stand for the collection of all finite non-empty multi-sets over $\mathscr{A} \cup \mathscr{C}$. One can think of elements of $\mathscr{B}_0$ and $\mathscr{B}_1$ as non-empty "buckets" or "packages" of atomic actions and elementary actions, respectively. In such packages a particular action may occur more than once; hence the use of *multi-sets* rather than (ordinary)

sets. We consider action buckets as a collection of (non-necessarily different) actions that can be executed simultaneously by a bunch of processors. (We return to this point later.) Concrete multi-sets are represented in the format $[e_1, \ldots, e_n]$.

For notational convenience below we use $\mathscr{L}$, $\mathscr{B}$, $\mathscr{E}$ to range over $\mathscr{L}_0$, $\mathscr{B}_0$, $\mathscr{A}$ and $\mathscr{L}_1$, $\mathscr{B}_1$, $\mathscr{A} \cup \mathscr{C}$, respectively.

2.2. Definition (Broy 1986). Distinguish a special symbol $\perp$ called "bottom" not in $\mathscr{A} \cup \mathscr{C}$. We define the set $\mathscr{B}^{st}$ of streams over $\mathscr{B}$ by $\mathscr{B}^{st} = \mathscr{B}^* \cup \mathscr{B}^*.\perp \cup \mathscr{B}^\omega$.

Here $\mathscr{B}^*$ is the collection of finite strings over $\mathscr{B}$; elements of $\mathscr{B}^*$ are called *finished* streams. Elements of $\mathscr{B}^*.\perp$, finite strings over $\mathscr{B}$ followed by.$\perp$, are called *unfinished* streams. $\mathscr{B}^\omega$ contains the infinite strings or ω-strings over $\mathscr{B}$. Elements of $\mathscr{B}^\omega$ are called *infinite* streams. We define the collection of *finite* streams $\mathscr{B}^f$ by $\mathscr{B}^f = \mathscr{B}^* \cup \mathscr{B}^*.\perp$. We shall use ε for the empty stream (i.e., the empty word in $\mathscr{B}^*$), and $\preceq$ for the prefix relation on streams. Furthermore, we use A, B, ... to range over $\mathscr{B}$, x, y, ... to range over $\mathscr{B}^{st}$, and X, Y, ... to range over subsets of $\mathscr{B}^{st}$, which we shall call stream sets. When dealing with tuples of streams we may write $\vec{x}$ instead of $\langle x_1, \ldots, x_k \rangle$.

We equip the sets of streams $\mathscr{B}_0^{st}$ and $\mathscr{B}_1^{st}$ with an ordering relation. In fact, this stream ordering turns $\mathscr{B}_0^{st}$ and $\mathscr{B}_1^{st}$ into cpo's.

2.3. Definition. We define the stream ordering $\leqq_{st}$ on $\mathscr{B}^{st}$ as follows:

(i) For all x, $y \in \mathscr{B}^{st}$: $x <_{st} y$ iff $\exists x' \in \mathscr{B}^* \exists y' \in \mathscr{B}^{st} \setminus \{\perp\}$ such that $x = x'\perp$ and $y = x'y'$.

(ii) We define $\leqq_{st}$ as the reflective closure of $<_{st}$.

Intuitively, a stream x is (stream-)less than a stream y exactly when x is unfinished, so ending in $\perp$ and x can be extended to y by expanding the trailing bottom.

2.4. Theorem (Back 1983). *$\mathscr{B}_0^{st}$ and $\mathscr{B}_1^{st}$ are complete partial orderings.*

The domains on $\mathscr{B}_0^{st}$ and $\mathscr{B}_1^{st}$ are not sufficiently rich to cater for all constructs in our language, in particular the non-deterministic choice. Therefore we change to powerdomains (cf., Plotkin 1976). So instead of streams we shall use sets of streams. To define these powerdomains it is convenient to formulate a technical notion, viz. truncation of streams. Moreover, truncations can be used for the extension of monotonic functions on finite streams to continuous functions on arbitrary streams.

2.5. Definition

(i) For $n \in \mathbb{N}$ and $x \in \mathscr{B}^{st}$ we define

$$x^{[n]} = \begin{cases} x & \text{if } length(x) < n \\ x'\perp & \text{if } length(x) \geqq n, \\ & \text{where } x' \in \mathscr{B}^n \text{ such that } x' \preceq x. \end{cases}$$

(ii) For $n \in \mathbb{N}$ and $X \subseteqq \mathscr{B}^{st}$ we define $X^{[n]} = \{x^{[n]} \mid x \in X\}$.

Next we present a canonical way to construct continuous functions on streams from monotonic functions on finite streams. This method relies on the monotonicity of truncations.

2.6. Lemma. *Let $\mathscr{D}$ be some cpo ordered by $\leqq_{\mathscr{D}}$.*

(i) *For all $n \in \mathbb{N}$ the truncation $\lambda x.x^{[n]} \in \mathscr{B}^{st} \longrightarrow \mathscr{B}^{st}$ is monotonic.*

(ii) (Extension lemma) *Let $f: (\mathscr{B}^f)^k \longrightarrow \mathscr{D}$ be monotonic. Define $F: (\mathscr{B}^{st})^k \longrightarrow \mathscr{D}$ by $F(\vec{x}) = lub_n f(\vec{x}^{[n]})$. Then F is well-defined and continuous.*

Proof

(i) Directly from the definition of $x^{[n]}$.

(ii) Note that F is well-defined by monotonicity of $\lambda n.x^{[n]}$, for fixed x and monotonicity of f.

(Monotonicity) Suppose $\vec{x} \leqq_{st} \vec{y}$ in $(\mathscr{B}^{st})^k$. By monotonicity of $\lambda x.x^{[n]}$ and of f, we have $F(\vec{x}) = lub_n f(\vec{x}^{[n]}) \leqq_{\mathscr{D}} lub_n f(\vec{y}^{[n]}) = F(\vec{y})$.

(Continuity property) We use the following fact: If $x = lub_i x_i$ in $\mathscr{B}^{st}$ then $\forall n \exists i_0 \forall i \geqq i_0 : x^{[n]} = x_i^{[n]}$. Suppose $\vec{x} = lub_i \vec{x}_i$ in $(\mathscr{B}^{st})^k$. Fix $n \in \mathbb{N}$. We have by the above fact $f(\vec{x}^{[n]}) = f(\vec{x}_{i_n}^{[n]})$ for suitable i_n. Hence $f(\vec{x}^{[n]}) \leqq_{\mathscr{D}} lub_m f(\vec{x}_{i_n}^{[m]}) = F(\vec{x}_{i_n})$ and $f(\vec{x}^{[n]}) \leqq_{\mathscr{D}} lub_i F(\vec{x}_i)$. So $F(\vec{x}) = lub_n f(\vec{x}^{[n]}) \leqq_{\mathscr{D}} lub_i F(\vec{x}_i)$. □

Note that the "converse" of the extension lemma trivially holds: suppose $F: (\mathscr{B}^{st})^k \longrightarrow \mathscr{B}^{st}$ is continuous. Define $f: (\mathscr{B}^f)^k \longrightarrow \mathscr{B}^{st}$ by $f = F|(\mathscr{B}^f)^k$, i.e., the restriction of F to $(\mathscr{B}^f)^k$. Then f is monotonic and $F(\vec{x}) = lub_n f(\vec{x}^{[n]})$, for all $\vec{x} \in (\mathscr{B}^{st})^k$.

On finite streams we have an obvious induction principle, that we shall call "stream induction". We define the norm $\|x\|$ of a finite stream x by $\|x\| = n$ iff $x \in \mathscr{B}^n \cup \mathscr{B}^n.\perp$. Let $X \subseteqq \mathscr{B}^f$ denote a set of finite streams with a certain property. Suppose that (i) $\perp$, $\varepsilon \in X$ and (ii) that from $\forall x \in \mathscr{B}^f : \|x\| < n \Rightarrow x \in X$ we derive $\forall x \in \mathscr{B}^f : \|x\| \leqq n \Rightarrow x \in X$. Then we have that $X = \mathscr{B}^f$, i.e., all finite streams satisfy this particular property.

We may use this induction principle to check the monotonicity of functions on finite streams. Suppose we are to prove that $f: \mathscr{B}^f \longrightarrow \mathscr{D}$ is monotonic. We let x_1, $x_2 \in \mathscr{B}^f$ such that $x_1 \leqq_{st} x_2$ and we proceed as follows with induction on (the norm

of) x_1. To prove $f(x_1) \leqq_{\mathscr{D}} f(x_2)$. If $x_1 = \perp$, then we are done in case of a strict function. If $x_1 = \varepsilon$, then $x_2 = \varepsilon$, and there is nothing to prove. If $x_1 = A x_1'$ for some $A \in \mathscr{B}$ and $x_1' \in \mathscr{B}^f$, then $x_2 = A x_2'$ for some $x_2' \in \mathscr{B}^f$ such that $x_1' \leqq_{st} x_2'$. We have $f(x_1') \leqq_{\mathscr{D}} f(x_2')$ by the induction hypothesis and may use this to arrive at $f(x_1) \leqq_{\mathscr{D}} f(x_2)$.

Next we define the powerdomains that we shall use in the sequel of our paper. We do not include all the stream sets in our domains, but only those that are *flat*, *closed* and *locally finite* for the following reasons: flatness is needed because of the anti-symmetry required for partial orders; closedness and local finiteness are needed because of a lifting lemma, that we intend to use to go from functions on streams to functions on stream sets.

2.7. Definition

(i) $X \subseteqq \mathscr{B}^{st}$ is flat iff $\forall x, y \in X: x \leqq_{st} y \Rightarrow x = y$.

(ii) $X \subseteqq \mathscr{B}^{st}$ is closed iff $\forall x \in \mathscr{B}^{st}$ is closed iff $\forall x \in \mathscr{B}^{st}: (\forall n \in \mathbb{N}: x^{[n]} \in X^{[n]}) \Rightarrow x \in X$.

(iii) $X \subseteqq \mathscr{B}^{st}$ is locally finite iff $\forall n \in \mathbb{N}: X^{[n]}$ is a finite set, i.e., all *truncations* of X are finite.

(iv) The collection $\mathscr{P}^*(\mathscr{B}^{st})$ of compact stream sets over $\mathscr{B}^{st}$ is defined by $\mathscr{P}^*(\mathscr{B}^{st}) = \{X \subseteqq \mathscr{B}^{st} \mid X$ flat, closed and locally finite$\}$.

The set $\mathscr{P}^*(\mathscr{B}^{st})$ of compact, i.e., flat, closed and locally finite stream sets is turned into a cpo by the Smyth ordering.

2.8. Definition (Smyth 1978). We define the Smyth ordering $\leqq_S$ on $\mathscr{P}^*(\mathscr{B}^{st})$ as follows: for all X, $Y \in \mathscr{P}^*(\mathscr{B}^{st})$: $X \leqq_S Y$ iff $\forall y \in Y\ \exists x \in X: x \leqq_{st} y$.

In Meyer and de Vink (1987) an extensive study is made of the domain of compact stream sets (over an arbitrary alphabet).[1] Especially the completeness of the partial ordering $\leqq_S$ and a lifting lemma were proved.

2.9. Theorem. *$\mathscr{P}^*(\mathscr{B}_0^{st})$ and $\mathscr{P}^*(\mathscr{B}_1^{st})$ are complete partial orderings with respect to the Smyth ordering.*

In these cpo's $\{\perp\}$ is the least element and for a chain $\langle X_i \rangle_i$ in $\mathscr{P}^*(\mathscr{B}^{st})$ the compact set $\{lub_i\, x_i \mid \langle x_i \rangle_i$ chain, $\forall i \in \mathbb{N}: x_i \in X_i\}$ acts as the least upperbound.

2.10. Lemma (Lifting lemma). *If f:*

$(\mathscr{B}^{st})^k \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$

is continuous and F:

$(\mathscr{P}^*(\mathscr{B}^{st}))^k \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$

is defined by $F(\vec{X}) = min(\bigcup \{f(\vec{x}) \mid \vec{x} \in \vec{X}\})$, then F is well defined and continuous.

In Lemma 2.10 the operator *min* takes the minimal elements of a stream set, i.e., for $X \subseteqq \mathscr{B}^{st}$ we have $min(X) = \{x \in X \mid \neg \exists x' \in X: x' <_{st} x\}$.

The extension lemma and the lifting lemma together give us a general method to construct semantical operators on the powerdomains $\mathscr{P}^*(\mathscr{B}_0^{st})$ and $\mathscr{P}^*(\mathscr{B}_1^{st})$. First we define a function from the collection of finite streams into the collection of streams or into the collection of compact stream sets and check that this function is monotonic. (To this end we may use stream induction.) Next we apply the extension lemma to obtain a continuous function on arbitrary, i.e., finite or infinite streams. Finally we obtain a continuous function on compact stream sets with the help of the lifting lemma.

We illustrate this method with the construction of sequential composition of compact stream sets over $\mathscr{B}_0$ and $\mathscr{B}_1$.

2.11. Definition

(i) We define $\cdot : \mathscr{B}^f \times \mathscr{B}^f \longrightarrow \mathscr{B}^{st}$ by the following: $\varepsilon \cdot y = y$, $\perp \cdot y = \perp$, $A x' \cdot y = A(x' \cdot y)$.

(ii) We define the sequential composition of streams $\cdot : \mathscr{B}^{st} \times \mathscr{B}^{st} \longrightarrow \mathscr{B}^{st}$ by

$$x \cdot y = lub_n\, x^{[n]} \cdot y^{[n]}$$

(where in the right-side expression $\cdot$ denotes sequential composition of finite streams).

(iii) We define the sequential composition of compact stream sets $\cdot : \mathscr{P}^*(\mathscr{B}^{st}) \times \mathscr{P}^*(\mathscr{B}^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ by $X \cdot Y = min(\{x \cdot y \mid x \in X,\ y \in Y\})$, (where on the right-hand side $\cdot$ denotes sequential composition of streams).

2.12. Theorem. *The sequential composition $\cdot$ on $\mathscr{P}^*(\mathscr{B}^{st})$ is continuous.*

Proof

(i) The sequential composition $\cdot$ on $\mathscr{B}^f$ is monotonic: this is easily verified by means of stream induction on (the norm of) x_1: Let $x_1, x_2, y_1, y_2 \in \mathscr{B}^f$ such that $x_1 \leqq_{st} x_2$ and $y_1 \leqq_{st} y_2$. We distinguish three cases: If $x_1 = \perp$, then $x_1 \cdot y_1 = \perp \leqq_{st} x_2 \cdot y_2$. If $x_1 = \varepsilon$, then $x_2 = \varepsilon$ and $x_1 \cdot y_1 = y_1 \leqq_{st} y_2 = x_2 \cdot y_2$. If $x_1 = A x_1'$ for some $A \in \mathscr{B}$ and $x_1' \in \mathscr{B}^f$, then $x_2 = A x_2'$ for some $x_2' \in \mathscr{B}^f$ such that $x_1' \leqq_{st} x_2'$. So $x_1 \cdot y_1 = A(x_1' \cdot y_1) \leqq_{st} A(x_2' \cdot y_2) = x_2 \cdot y_2$ by the induction hypothesis.

[1] In Meyer and de Vink (1987) we used the term "boundedness" instead of "local finiteness". In the present paper we do not use this term to prevent confusion with the notion of "bounded" in Sect. 4

(ii) The sequential composition $\cdot$ on $\mathscr{B}^{st}$ is continuous: since $\cdot : \mathscr{B}^f \times \mathscr{B}^f \longrightarrow \mathscr{B}^{st}$ is monotonic, we have by the extension lemma that $\cdot : \mathscr{B}^{st} \times \mathscr{B}^{st} \longrightarrow \mathscr{B}^{st}$ is continuous.

(iii) The sequential composition $\cdot$ on $\mathscr{P}^*(\mathscr{B}^{st})$ is continuous: from (ii) we derive immediately the continuity of $\lambda x y.\{x \cdot y\} \in \mathscr{B}^{st} \times \mathscr{B}^{st} \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$. Hence we have the continuity of $\cdot : \mathscr{P}^*(\mathscr{B}^{st}) \times \mathscr{P}^*(\mathscr{B}^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ by the lifting lemma. □

Another example of a function of which the continuity is already determined on finite streams is the so-called flat union $+$. We might define the flat union of compact streams sets as a lifted extended version of $\lambda x y.min(\{x, y\}) \in \mathscr{B}^{st} \times \mathscr{B}^{st} \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$. We shall not do this, since in this case we feel comfortable to define $+$ directly.

2.13. Definition. We define the flat union of compact stream sets $+ : \mathscr{P}^*(\mathscr{B}^{st}) \times \mathscr{P}^*(\mathscr{B}^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ by $X + Y = min(X \cup Y)$.

We invite the reader to check that $+$ as given in the above definition is indeed continuous and to compare this proof with the effort of a definition and a proof along the lines of Def. 2.11 and Lemma 2.12.

Note that $\{\perp\}$ acts as an absorbing element with respect to $+$, i.e., $X + \{\perp\} = \{\perp\} + X = \{\perp\}$.

We conclude this section with some comments on the semantics for the languages $\mathscr{L}_0$ and $\mathscr{L}_1$. We already have interpretations for the syntactical construct of sequential composition and nondeterministic choice, viz. $\cdot$ and $+$. We shall work within the context of *uniform* concurrency: the elementary actions a and c remain uninterpreted in this framework. We shall give meaning to variables with the help of environments and to the recursive μ-construct with the help of fixed point techniques.

If we have a semantical interpretation, say $\|_{sem}$ for the parallel composition, we are able to use the following scheme for the semantics of the languages $\mathscr{L}$, under the condition that $\|_{sem}$ is a continuous operator on $\mathscr{P}^*(\mathscr{B}^{st})$.

2.14. Definition. We let $Env = \Xi \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ with typical element η be the collection of environments. We define the semantics $Sem: \mathscr{L} \longrightarrow Env \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ with respect to the concurrency operator $\|_{sem}$ by the clauses:

$Sem[\![e]\!](\eta) = \{[e]\}$ for $e \in \mathscr{E}$
$Sem[\![s_1 ; s_2]\!](\eta) = Sem[\![s_1]\!](\eta) \cdot Sem[\![s_2]\!](\eta)$
$Sem[\![s_1 \cup s_2]\!](\eta) = Sem[\![s_1]\!](\eta) + Sem[\![s_2]\!](\eta)$
$Sem[\![s_1 \| s_2]\!](\eta) = Sem[\![s_1]\!](\eta) \|_{sem} Sem[\![s_2]\!](\eta)$
$Sem[\![\xi]\!](\eta) = \eta(\xi)$
$Sem[\![\mu\xi[s]]\!](\eta) = lfp(\Phi_{s,\eta})$
where $\Phi_{s,\eta} = \lambda X.Sem[\![s]\!](\eta\{X/\xi\})$.

In the above scheme we assign to the recursive μ-construct the least fixed point of the operator $\lambda X.Sem[\![s]\!](\eta\{X/\xi\})$, which evaluates for given stream set X the statement s in the environment η where the variable ξ is set by X. This fixed point construction is justified by the following lemma.

2.15. Lemma

(i) *If* $\|_{sem} : \mathscr{P}^*(\mathscr{B}^{st}) \times \mathscr{P}^*(\mathscr{B}^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ *is continuous, then we have that*

$\lambda X_1, \ldots, X_k.Sem[\![s]\!](\eta\{X_1/\xi_1, \ldots, X_k/\xi_k\})$

is continuous, for all $k \in \mathbb{N}$, $s \in \mathscr{L}$, $\eta \in Env$ *and* $\xi_1, \ldots, \xi_k \in \Xi$.

(ii) *Under the condition of* (i) *it holds that for all* $s \in \mathscr{L}$, $\eta \in Env$ *and* $\xi \in \Xi$ $\lambda X.Sem[\![s]\!](\eta\{X/\xi\})$ *has a least fixed point and that for all* $s \in \mathscr{L}$ *and* $\eta \in Env$ $Sem[\![s]\!](\eta)$ *is well-defined.*

Proof

(i) (Structural induction on s) By the continuity of $\cdot$, $+$ and $\|_{sem}$.

(ii) Follow from (i) (cf., de Bakker 1980). □

In the next sections we focus on the interpretation of the parallel composition. We shall give several semantical counterparts for the concurrency operator according to different models of parallelism, but in all cases the continuity of the (interpretation of) $\|$ will be established with the help of the Extension lemma and the Lifting lemma.

3 The basic semantic models: synchronous start and multi-processor concurrency

In this section we present a number of basic semantic models for "true" concurrency. Subsequently we discuss (i) a simple "synchronous start" semantics for $\mathscr{L}_0$, in which the components of a parallel statement start their execution simultaneously, (ii) a multi-processor semantics for $\mathscr{L}_0$, which provides a more general way of scheduling parallel statements on a multi-processor, and (iii) a multi-processor semantics for the language $\mathscr{L}_1$ with synchronization, which is an extension of the one mentioned under (ii). We also discuss a similar generalization to synchronization of the synchronous start semantics. Unfortunately, this semantics does not meet the intuition. In Sect. 4 we shall re-

medy this by the incorporation of the notion of maximal parallelism.

3.1 The synchronous start model for $\mathscr{L}_0$

The first semantics for $\mathscr{L}_0$ is based on what we shall call synchronous start concurrency: In this model the components of a parallel construct have to start and act synchronously; at each tick of the clock the components perform their actions simultaneously. (Cf., Salwicki and Müldner 1981.) So here both parallel operands in a parallel statement progress at the same pace. (If one of the components terminates, the other one proceeds on its own.) This is a simple form of 'truly' concurrent execution which will appear to be a special case of the more sophisticated one to be given in the next subsection.

Thus, for example, the semantics of the parallel statement $(a; b) \| a'$, where a, a', b are in $\mathscr{A}$, is given by the singleton stream set $\{[a, a'][b]\}$ in this model, expressing that the execution of the atomic actions a and a' is started synchronously, after which the atomic action b is executed.

The semantics of the statement

$$(\mu\,\xi[a;\xi] \| \mu\eta[b;\eta]) \| \mu\zeta[c;\zeta]$$

in the synchronous start model turns out to be $\{[a, b, c]^\omega\}$, i.e., infinitely many simultaneous executions of three actions, viz. a, b and c: The μ-constructs $\mu\xi[a;\xi]$, $\mu\eta[b;\eta]$ and $\mu\zeta[c;\zeta]$ yield $\{[a]^\omega\}$, $\{[b]^\omega\}$ and $\{[c]^\omega\}$, respectively. Now $\{[a]^\omega\} \|_{ss} \{[b]^\omega\} = \{[a, b]^\omega\}$ according to our notion of synchronous co-operation of $[a]^\omega$ and $[b]^\omega$, and analogously $\{[a, b]^\omega\} \|_{ss} \{[c]^\omega\} = \{[a, b, c]^\omega\}$. ($\|_{ss}$ is the parallel operator in the synchronous start model.)

Since we deal with streams of multi-sets, we formalize synchronous co-operation by the *multi-set union*. For finite multi-sets A, B over $\mathscr{A}$ (i.e., for two functions $A, B: \mathscr{A} \longrightarrow \mathbb{N}$ with $A(a), B(a) = 0$ for almost all but not all $a \in \mathscr{A}$) we write $A \cup_m B$ for their multi-set union (i.e., $A \cup_m B: \mathscr{A} \longrightarrow \mathbb{N}$ is such that $(A \cup_m B)(a) = A(a) + B(a)$ for all $a \in \mathscr{A}$). Analogously for multi-sets over $\mathscr{A} \cup \mathscr{C}$.

3.1. Definition. We define the parallel composition $\|_{ss}$ with respect to the synchronous start model as follows:

(i) $\|_{ss}: \mathscr{B}_0^f \times \mathscr{B}_0^f \longrightarrow \mathscr{B}_0^{st}$ is defined by
$\varepsilon \|_{ss} y = y$ and $x \|_{ss} \varepsilon = x$
$\perp \|_{ss} y = x \|_{ss} \perp = \perp$
$A x' \|_{ss} B y' = (A \cup_m B)(x' \|_{ss} y')$.

(ii) $\|_{ss}: \mathscr{B}_0^{st} \times \mathscr{B}_0^{st} \longrightarrow \mathscr{B}_0^{st}$ is defined by
$x \|_{ss} y = lub_n\, x^{[n]} \|_{ss} y^{[n]}$.

(iii) $\|_{ss}: \mathscr{P}^*(\mathscr{B}_0^{st}) \times \mathscr{P}^*(\mathscr{B}_0^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}_0^{st})$ is defined by
$X \|_{ss} Y = min\{x \|_{ss} y \mid x \in X,\ y \in Y\}$.

We check with the tools developed in Sect. 2, that $\|_{ss}$ is a well-defined and continuous operator on compact stream sets.

3.2. Theorem. *$\|_{ss}$ is continuous on $\mathscr{P}^*(\mathscr{B}_0^{st})$.*

Proof

(i) $\|_{ss}$ monotonic on $\mathscr{B}_0^f$ (stream induction to x_1 and y_1): Let $x_1, x_2, y_1, y_2 \in \mathscr{B}_0^f$ such that $x_1 \leqq_{st} x_2$, $y_1 \leqq_{st} y_2$. If $x_1 = \perp$ or $y_1 = \perp$ then $x_1 \|_{ss} y_1 = \perp \leqq_{st} x_2 \|_{ss} y_2$. If $x_1 = \varepsilon$, then $x_2 = \varepsilon$ and so $x_1 \|_{ss} y_1 = y_1 \leqq_{st} y_2 = x_2 \|_{ss} y_2$. If $y_1 = \varepsilon$, then $y_2 = \varepsilon$. Hence $x_1 \|_{ss} y_1 = x_1 \leqq_{st} x_2 = x_2 \|_{ss} y_2$. If $x_1 = A x_1'$ and $y_1 = B y_1'$ for some $A, B \in \mathscr{B}_0$ and $x_1', y_1' \in \mathscr{B}_0^f$, then $x_2 = A x_2'$ and $y_2 = B y_2'$ with $x_2', y_2' \in \mathscr{B}_0^f$ such that $x_1' \leqq_{st} x_2'$ and $y_1' \leqq_{st} y_2'$. We have

$$x_1 \|_{ss} y_1 = (A \cup_m B)(x_1' \|_{ss} y_1') \leqq_{st} (A \cup_m B)(x_2' \|_{ss} y_2') = x_2 \|_{ss} y_2$$

by the induction hypothesis.

(ii) $\|_{ss}$ is continuous on $\mathscr{B}_0^{st}$: By (i) and the Extension lemma 2.6(ii).

(iii) $\|_{ss}$ is continuous on $\mathscr{P}^*(\mathscr{B}_0^{st})$: By (ii) is $\lambda x y.\{x \|_{ss} y\}$ continuous. Hence is $\|_{ss}$ on $\mathscr{P}^*(\mathscr{B}_0^{st})$ by the Lifting lemma 2.10. □

We are now in the position to give the first semantics for $\mathscr{L}_0$. It uses the scheme given at the end of Sect. 2. Note that by Theorem 3.2 the condition of Lemma 2.15 viz. continuity of the concurrency operator, is fulfilled, so that the least fixed point in the clause for the recursive μ-construct exists.

3.3. Definition. (Synchronous start semantics for $\mathscr{L}_0$) Let $Env_0 = \Xi \longrightarrow \mathscr{P}^*(\mathscr{B}_0^{st})$. We define the synchronous start semantics SS: $\mathscr{L}_0 \longrightarrow Env_0 \longrightarrow \mathscr{P}^*(\mathscr{B}_0^{st})$ for $\mathscr{L}_0$ by the following clauses:

$SS[\![a]\!](\eta) = \{[a]\}$ for $a \in \mathscr{A}$
$SS[\![s_1; s_2]\!](\eta) = SS[\![s_1]\!](\eta) \cdot SS[\![s_2]\!](\eta)$
$SS[\![s_1 \cup s_2]\!](\eta) = SS[\![s_1]\!](\eta) + SS[\![s_2]\!](\eta)$
$SS[\![s_1 \| s_2]\!](\eta) = SS[\![s_1]\!](\eta) \|_{ss} SS[\![s_2]\!](\eta)$
$SS[\![\xi]\!](\eta) = \eta(\xi)$
$SS[\![\mu\xi[s]]\!](\eta) = lfp(\Phi_{s,\eta})$
where $\Phi_{s,\eta} = \lambda X.SS[\![s]\!](\eta\{X/\xi\})$.

3.2 The multi-processor model for $\mathscr{L}_0$

Next we show how we can refine the synchronous start semantics to a so-called multi-processor semantics, where the execution of two parallel components is less restricted. In this multi-processor

scheduling statements that are put in parallel can start "on their own". (Cf., Taubner and Vogler 1987.) We imagine our programs to run on a multiprocessor, where sometimes actions can be done simultaneously but sometimes have to be serialized. In fact, the semantics to be given in this subsection will resemble the (uniprocessor oriented) interleaving semantics of de Bakker et al. (1986) with the main difference that now more processors can be employed at one time.

The synchronous start semantics requires parallel statements to be started synchronously. We relax this requirement by allowing one (but only one) of the two parallel components to be suspended for some (units of) time. So apart from synchronous co-operation we also obtain the pure interleavings and combinations of synchronous co-operation and interleaving "schedulings" of a parallel construct.

In the example $(a; b) \| c$ this amounts to the following: The two parallel components can synchronously yielding $[a, c][b]$ as we had before, but now we also have the pure interleavings $[a][b][c]$, $[a][c][b]$, $[c][a][b]$ and the outcome $[a][b, c]$, due to suspension. In $[a][b][c]$ the action c is postponed until ab has finished; in $[a][c][b]$ the action c has waited one tick of the clock and is executed after a but before b; in $[a][b, c]$ the action c has again waited one unit of time but now co-operates with the action b; in $[c][a][b]$ the execution of c precedes the execution of ab.

The second example

$$(\mu\xi[a; \xi] \| \mu\eta[b; \eta]) \| \mu\zeta(c; \zeta]$$

has in the multi-processor model $[a, b, c]^\omega$ and the "unfair" $[a]^\omega$ among its outcomes. The latter is obtained by suspension of $\mu\eta[b; \eta]$ and $\mu\zeta[c; \zeta]$ ad infinitum. In fact, all the infinite streams $A_1 A_2 A_3 \ldots$ such that $A_i \subseteq [a, b, c]$ are possible behaviours of the statement

$$(\mu\xi[a; \xi] \| \mu\eta[b; \eta]) \| \mu\zeta[c; \zeta].$$

3.4. Definition. We define the parallel composition $\|_{mp}$ with respect to the multi-processor model as follows:

(i) $\|_{mp}, \lfloor\!\lfloor_{mp}, |_{mp} : \mathscr{B}_0^f \times \mathscr{B}_0^f \longrightarrow \mathscr{P}^*(\mathscr{B}_0^{st})$ are defined by

$x \|_{mp} y = x \lfloor\!\lfloor_{mp} y + y \lfloor\!\lfloor_{mp} x + x |_{mp} y$

$\varepsilon \lfloor\!\lfloor_{mp} y = \{y\}$

$\perp \lfloor\!\lfloor_{mp} y = \{\perp\}$

$A x' \lfloor\!\lfloor_{mp} y = A(x' \|_{mp} y)$

$\varepsilon |_{mp} y = x |_{mp} \varepsilon = \varnothing$

$\perp |_{mp} y = x |_{mp} \perp = \varnothing$

$A x' |_{mp} B y' = (A \cup_m B)(x' \|_{mp} y')$.

(ii) $\|_{mp} : \mathscr{B}_0^{st} \times \mathscr{B}_0^{st} \longrightarrow \mathscr{P}^*(\mathscr{B}_0^{st})$ is defined by $x \|_{mp} y = lub_n\, x^{[n]} \|_{mp} y^{[n]}$.

(iii) $\|_{mp} : \mathscr{P}^*(\mathscr{B}_0^{st}) \times \mathscr{P}^*(\mathscr{B}_0^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}_0^{st})$ is defined by $X \|_{mp} Y = min(\cup\{x \|_{mp} y \mid x \in X, y \in Y\})$.

In the above definition we use the auxiliary operator $\lfloor\!\lfloor_{mp}$ in the context of process algebra known as "left-merge", (see Bergstra and Klop 1984).

We have that $\|_{mp} : \mathscr{P}^*(\mathscr{B}_0^{st}) \times \mathscr{P}^*(\mathscr{B}_0^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}_0^{st})$ is continuous, as is formulated in the lemma below. The proof again uses the tools of Sect. 2.

3.5. Theorem. *$\|_{mp}$ is continuous on $\mathscr{P}^*(\mathscr{B}_0^{st})$.*

Proof

(i) $\|_{mp}$ is monotonic on $\mathscr{B}_0^f$: let $x_1, x_2, y_1, y_2 \in \mathscr{B}_0^f$ such that $x_1 \leqq_{st} x_2$, $y_1 \leqq_{st} y_2$. We prove by stream induction on x_1 and y_1: (#) $x_1 \otimes y_1 \leqq_S x_2 \otimes y_2$ for $\otimes \in \{\|_{mp}, \lfloor\!\lfloor_{mp}\}$.

If $x_1 = \perp$ then $x_1 \lfloor\!\lfloor_{mp} y_1 = x_1 \|_{mp} y_1 = \{\perp\}$ and (#) is obviously satisfied.

If $x_1 = \varepsilon$ and $y_1 = \perp$, then we have $x_1 \lfloor\!\lfloor_{mp} y_1 = \{\perp\}$, hence $x_1 \lfloor\!\lfloor_{mp} y_1 \leqq_S x_2 \lfloor\!\lfloor_{mp} y_2$ and $x_1 \|_{mp} y_1 = \{\perp\}$. So $x_1 \|_{mp} y_1 \leqq_S x_2 \|_{mp} y_2$.

If $x_1 = \varepsilon$ and $y_1 = \varepsilon$, then $x_2 = \varepsilon$ and $y_2 = \varepsilon$ so (#) is trivially fulfilled.

If $x_1 = \varepsilon$ and $y_1 = B y_1'$ for some $B \in \mathscr{B}_0$ and $y_1' \in \mathscr{B}_0^f$, then $x_2 = \varepsilon$ and $y_2 = B y_2'$ with $y_2' \in \mathscr{B}_0^f$ such that $y_1' \leqq_{st} y_2'$. We have

$$x_1 \lfloor\!\lfloor_{mp} y_1 = \{y_1\} \leqq_S \{y_2\} = x_2 \lfloor\!\lfloor_{mp} y_2$$

and

$$\begin{aligned} x_1 \|_{mp} y_1 &= x_1 \lfloor\!\lfloor_{mp} y_1 + y_1 \lfloor\!\lfloor_{mp} x_1 + x_1 |_{mp} y_1 \\ &= \{y_1\} + B(y_1' \|_{mp} x_1) + \varnothing \\ &\leqq_S \{y_2\} + B(y_2' \|_{mp} x_2) + \varnothing = x_2 \|_{mp} y_2 \end{aligned}$$

by monotonicity of $+$ and by the induction hypothesis.

If $x_1 = A x_1'$ and $y_1 = \perp$ for some $A \in \mathscr{B}_0$ and $x_1' \in \mathscr{B}_0^f$, then $x_2 = A x_2'$ with $x_2' \in \mathscr{B}_0^f$ such that $x_1' \leqq_{st} x_2'$. We have $x_1 \lfloor\!\lfloor_{mp} y_1 = A(x_1' \|_{mp} y_1) \leqq_S A(x_2' \|_{mp} y_2) = x_2 \lfloor\!\lfloor_{mp} y_2$ by the induction hypothesis and $x_1 \|_{mp} y_1 = \{\perp\} \leqq_S x_2 \|_{mp} y_2$.

If $x_1 = A x_1'$ and $y_1 = \varepsilon$ for some $A \in \mathscr{B}_0$ and $x_1' \in \mathscr{B}_0^f$, then $x_2 = A x_2'$ and $y_2 = \varepsilon$ with $x_2' \in \mathscr{B}_0^f$ such that $x_1' \leqq_{st} x_2'$. We have $x_1 \lfloor\!\lfloor_{mp} y_1 = A(x_1' \|_{mp} y_1) \leqq_S A(x_2' \|_{mp} y_2) = x_2 \lfloor\!\lfloor_{mp} y_2$ by the induction hypothesis and

$$\begin{aligned} x_1 \|_{mp} y_1 &= A(x_1' \|_{mp} y_1) + \{x_1\} + \varnothing \\ &\leqq_S A(x_2' \|_{mp} y_2) + \{x_2\} + \varnothing = x_2 \|_{mp} y_2 \end{aligned}$$

by monotonicity of $+$ and the induction hypotheses.

If $x_1 = A x_1'$ and $y_1 = B y_1'$ for some $A, B \in \mathscr{B}_0$ and $x_1', y_1' \in \mathscr{B}_0^f$, then $x_2 = A x_2'$ and $y_2 = B y_2'$ with $x_2', y_2' \in \mathscr{B}_0^f$ such that $x_1' \leqq_{st} x_2'$ and $y_1' \leqq_{st} y_2'$. We have

$$x_1 \|_{mp} y_1 = A(x_1' \|_{mp} y_1) \leqq_S A(x_2' \|_{mp} y_2) = x_2 \|_{mp} y_2$$

by the induction hypothesis and

$$\begin{aligned} x_1 \|_{mp} y_1 &= A(x_1' \|_{mp} y_1) + B(y_1' \|_{mp} x_1) + (A \cup_m B)(x_1' \|_{mp} y_1') \\ &\leqq_S A(x_2' \|_{mp} y_2) + B(y_2' \|_{mp} x_2) + (A \cup_m B)(x_2' \|_{mp} y_2') = x_2 \|_{mp} y_2 \end{aligned}$$

again by the induction hypothesis.

(ii) $\|_{mp}$ is continuous on $\mathscr{B}_0^{st}$: by (i) and the Extension lemma.

(iii) $\|_{mp}$ is continuous on $\mathscr{P}^*(\mathscr{B}_0^{st})$: by (ii) and the Lifting lemma. □

The interpretation of $\|$ in the multi-processor model by $\|_{mp}$ as given in Def. 3.4 induces the second semantics for $\mathscr{L}_0$.

3.6. Definition (Multi-processor semantics for $\mathscr{L}_0$). *Let* $Env_0 = \Xi \longrightarrow \mathscr{P}^*(\mathscr{B}_0^{st})$ as before. We define MP_0: $\mathscr{L}_0 \longrightarrow Env_0 \longrightarrow \mathscr{P}^*(\mathscr{B}_0^{st})$ by the following clauses:

$MP_0[\![a]\!](\eta) = \{[a]\}$ for $a \in \mathscr{A}$
$MP_0[\![s_1 ; s_2]\!](\eta) = MP_0[\![s_1]\!](\eta) \cdot MP_0[\![s_2]\!](\eta)$
$MP_0[\![s_1 \cup s_2]\!](\eta) = MP_0[\![s_1]\!](\eta) + MP_0[\![s_2]\!](\eta)$
$MP_0[\![s_1 \| s_2]\!](\eta) = MP_0[\![s_1]\!](\eta) \|_{mp} MP_0[\![s_2]\!](\eta)$
$MP_0[\![\xi]\!](\eta) = \eta(\xi)$
$MP_0[\![\mu\xi[s]]\!](\eta) = lfp(\Phi_{s,\eta})$.
where $\Phi_{s,\eta} = \lambda X.MP_0[\![s]\!](\eta\{X/\xi\})$.

As before the definition is justified by the continuity of $\|_{mp}$ and Lemma 2.15.

3.3 *The multi-processor model for* $\mathscr{L}_1$

In this subsection we focus on the language $\mathscr{L}_1$, i.e., $\mathscr{L}_0$ augmented with synchronization actions. CCS-like synchronization primitives enable us to force the execution of parts of a parallel program to take place at some particular time. We extend the semantics of the previous subsection to incorporate this new feature.

In the synchronous start model and in the multi-processor model (without synchronization) synchronous co-operation between action-packages was formalized by the multi-set union. Now we shall use a "synchronization union" (an extension of the multi-set union) for the co-operation of multi-sets.

In the synchronization union matching pairs of synchronizations are replaced by two dummy actions τ, indicating that two processors are involved in some synchronization. Remaining synchronization primitives are preserved for possible synchronizations with other packages in the context. Afterwards, i.e., after the determination of the meaning of an entire statement, we may remove all streams in which still synchronization actions are present, i.e., not replaced by the τ-action. This will be done by means of an abstraction operator. The intuition behind this is that we may regard these streams as synchronized unsuccessfully.

We next define the synchronization union for multi-sets. Let $A, B \in \mathscr{B}_1$ be two finite bags of elementary actions. We construct the synchronization union $A \cup_s B$ of A and B as follows: we first take the multi-set union $A \cup_m B$. Next we replace an arbitrary number of subbags $[c, \bar{c}]$of matching synchronization actions by subbags $[\tau, \tau]$. We may repeat this until no matching synchronization pairs are left. It is obvious that the resulting bag does not depend on the order in which the subbags $[c, \bar{c}]$ are chosen. Moreover, this rewriting procedure always terminates since $A \cup_m B$ is finite, being the union of two finite multi-sets, so it contains only finitely many synchronizations and with each replacement the number of synchronization actions decreases. Hence there are finitely many multi-sets M obtained from $A \cup_m B$ by repeatedly replacing matching pairs. We formalize this by introducing a Noetherian relation $\longrightarrow_\tau$ (i.e., for all $A \in \mathscr{B}_1$ there is no infinite sequence $\langle B_n \rangle_n$ in $\mathscr{B}_1$ such that $B_0 = A$ and $\forall n \in \mathbb{N}$: $B_n \longrightarrow_\tau B_{n+1}$).

3.7. Definition

(i) We define the relation $\longrightarrow_\tau$ on $\mathscr{B}_1$ by $A \longrightarrow_\tau B$ iff $\exists c \in \mathscr{C}$: $A \cup_m [\tau, \tau] = B \cup_m [c, \bar{c}]$, for all $A, B \in \mathscr{B}_1$.

(ii) We define the synchronization union $\cup_s$: $\mathscr{B}_1.br \times \mathscr{B}_1 \longrightarrow \mathscr{P}(\mathscr{B}_1)$ by $A \cup_s B = \{M \mid A \cup_m B \to_\tau^* M\}.\to_\tau$.

For example, $[a, b, c] \cup_s [a, \bar{c}] = \{[a, a, b, \tau, \tau], [a, a, b, c, \bar{c}]\}$ and $[a, b, c] \cup_s [a, b] = \{[a, b, c] \cup_m [a, b]\} = \{[a, a, b, b, c]\}$.
Note that the restriction of the synchronization union to $\mathscr{B}_0$ coincides with the multi-set union. So $\cup_s$ is indeed an extension of $\cup_m$.

We use the synchronization union in the next definition to formalize synchronous co-operation. Note the simularity between Def. 3.4 for $\mathscr{L}_0$ and Def. 3.8 for $\mathscr{L}_1$.

3.8. Definition. We define the parallel composition $\|_{mp}$ with respect to the multi-processor model with synchronization as follows:

(i) $\|_{mp}, \lfloor\!\lfloor_{mp}, |_{mp} : \mathscr{B}_1^f \times \mathscr{B}_1^f \longrightarrow \mathscr{P}^*(\mathscr{B}_1^{st})$ are defined by
$x \|_{mp} y = x \lfloor\!\lfloor_{mp} y + y \lfloor\!\lfloor_{mp} x + x|_{mp} y$
$\varepsilon \lfloor\!\lfloor_{mp} y = \{y\}$
$\perp \lfloor\!\lfloor_{mp} y = \{\perp\}$
$A x' \lfloor\!\lfloor_{mp} y = A(x' \|_{mp} y)$
$\varepsilon|_{mp} y = x|_{mp} \varepsilon = \varnothing$
$\perp|_{mp} y = x|_{mp} \perp = \varnothing$
$A x'|_{mp} B y' = (A \cup_s B)(x' \|_{mp} y')$.

(ii) $\|_{mp} : \mathscr{B}_1^{st} \times \mathscr{B}_1^{st} \longrightarrow \mathscr{P}^*(\mathscr{B}_1^{st})$ is defined by
$x \|_{mp} y = lub_n\, x^{[n]} \|_{mp} y^{[n]}$.

(iii) $\|_{mp} : \mathscr{P}^*(\mathscr{B}_1^{st}) \times \mathscr{P}^*(\mathscr{B}_1^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}_1^{st})$ is defined by
$X \|_{mp} Y = min(\cup\{x \|_{mp} y \mid x \in X,\ y \in Y\})$.

We have again the monotonicity anc continuity for $\|_{mp}$ with respect to $\mathscr{B}_1$ as stated in Theorem 3.9. The proof of 3.9 is literally the same as the proof of 3.5 when we substitute $\cup_s$ for $\cup_m$; so it is omitted here.

3.9. Theorem. *$\|_{mp}$ is continuous on $\mathscr{P}^*(\mathscr{B}_1^{st})$.*

The first semantics MP_1 for $\mathscr{L}_1$ follows the route of SS and MP_0. In the semantic definition scheme we use the semantic $\|_{mp}$ with respect to $\mathscr{B}_1$ to serve as the counterpart of the syntactic $\|$.

3.10. Definition (Multi-processor semantics for $\mathscr{L}_1$). Let $Env_1 = \Xi \longrightarrow \mathscr{P}^*(\mathscr{B}_1^{st})$. We define $MP_1 : \mathscr{L}_1 \longrightarrow Env_1 \longrightarrow \mathscr{P}^*(\mathscr{B}_1^{st})$ by the following clauses:

$MP_1[\![e]\!](\eta) = \{[e]\}$ for $e \in \mathscr{E}$
$MP_1[\![s_1 ; s_2]\!](\eta) = MP_1[\![s_1]\!](\eta) \cdot MP_1[\![s_2]\!](\eta)$
$MP_1[\![s_1 \cup s_2]\!](\eta) = MP_1[\![s_1]\!](\eta) + MP_1[\![s_2]\!](\eta)$
$MP_1[\![s_1 \| s_2]\!](\eta) = MP_1[\![s_2]\!](\eta) \|_{mp} MP_1[\![s_2]\!](\eta)$
$MP_1[\![\xi]\!](\eta) = \eta(\xi)$
$MP_1[\![\mu\xi[s]]\!](\eta) = lfp(\Phi_{s,\eta})$
where $\Phi_{s,\eta} = \lambda X.MP_1[\![s]\!](\eta\{X/\xi\})$.

Note again that the least fixed point in Def. 3.10 is justified by an appeal to Lemma 2.15 and Theorem 3.9.

We derive the second (non-compositional) semantics MP_1' for $\mathscr{L}_1$ by (non-compositional) applying an abstraction operator to the semantics MP_1. This operator, called *failure removal*, deletes all streams that still contain synchronization actions from the semantics of a statement. (Cf., de Bakker et al. 1989.)

3.11. Definition

(i) The failure removal operator fr: $\mathscr{P}(\mathscr{B}_1^{st}) \longrightarrow \mathscr{P}(\mathscr{B}_0^{st})$ is defined by $fr(X) = X \cap \mathscr{B}_0^{st}$, for all $X \in \mathscr{P}^*(\mathscr{B}_1^{st})$.

(ii) (Multi-processor semantics with failure removal for $\mathscr{L}_1$) The semantics $MP_1' : \mathscr{L}_1 \longrightarrow Env_1 \longrightarrow \mathscr{P}^*(\mathscr{B}_0^{st})$ is defined by $MP_1' = fr \circ MP_1$.

Remark. In the semantics MP_1' the failure of one of the processors is interpreted as a *global* failure. One might object that this is too crude an approach in the context of true concurrency. Perhaps one would like to model in this case that just one processor fails while other ones may continue. This could be modelled in a slightly modified framework where we take tuples instead of multisets as basis "buckets" of actions; the elements of these tuples then correspond exactly to the available processors in a certain fixed order.

3.4 The synchronous start model for $\mathscr{L}_1$

After we have generalized successfully our multi-processor semantics from $\mathscr{L}_0$ to $\mathscr{L}_1$, it seems an obvious attempt to generalize the synchronous start semantics from $\mathscr{L}_0$ to $\mathscr{L}_1$. However, when we try to do this by replacing in Def. 3.1 $A \cup_m B$ by $A \cup_s B$, and using our usual scheme for the definition of the semantics, we end up with a semantics which is not correct intuitively. This can be easily seen from the following example: the semantics of the statement $(a \| c) \| (b; \bar{c})$ yields

$$(\{[a]\} \|_{ss} \{[c]\}) \|_{ss} \{[b][\bar{c}]\} = \{[a, c]\} \|_{ss} \{[b][\bar{c}]\} = \{[a, b, c][\bar{c}]\}$$

which gives $\varnothing$ after applying failure removal). So no synchronization has taken place; in some sense the synchronous start operator $\|_{ss}$ optimizes parallel execution in a way which is too local!.

On the other hand in the multi-processor model of Sect. 3.3 the semantics MP_1' gives too many outcomes, due to unnecessary interleavings – although unsuccessfully synchronized streams are deleted by the abstraction operator fr. We remedy this shortcoming by the introduction of yet another abstraction operator ($maxpar$) that will model a notion of maximal parallelism. This will be done in the next section.

4 Further refinements of the basic semantic models

In this section we shall discuss a number of refinements of the basic models in Sect. 3 that one may

wish to consider. First we consider a model in which there are a *bounded* (fixed) number of processors which can be used for scheduling our program statements. We shall indicate how to modify our semantic definitions in order to model this case. Next we turn to a way to model atomic actions that take longer to be executed than one tick of the global clock. This issue has, of course, to be taken into consideration in order to make our approach useful to more practical examples. We shall see that it is fairly simple to accommodate for these actions in our framework of true concurrency by varying the definition of the concurrent operator(s), in particular the auxiliary left-merge operator. In the third subsection we shall pay attention to the issue of maximal parallelism. Once again truncation of streams turns out to be a useful tool in the domain of streams. At the end of this section we apply the findings of the semantics for subatomic actions to model some aspects of *real-time* programming, viz. a treatment of a *delay* operator.

4.1 Bounded number of processors

Up to now we have assumed that we have a finite, but arbitrarily great number of processors that can be put to work at any moment. One may to drop this assumption, since this implies that no matter how many processors we have working, we can always add one processor to do the next job in parallel. In practice, this may lead to employing more and more processors, ad infinitum. In this subsection we restrict ourselves to the situation that we are given a fixed amount of processors, say N, and with these we must do the job. We shall refer to this situation as having a *bounded* number of processors.

We shall modify our definitions in order to accommodate for this situation of a bounded number of processors. As it will be expected, now the interleaving part of our definitions will play a more visible role in the outcomes of the scheduling of a fixed number of processors: now it may happen that not all parallel statements described in the program can be truly in parallel at once (because all processors are occupied already), so that some of these must be interleaved. In our definitions it is easy to give a straight forward implementation by adaptation of the synchronization union:

4.1 Definition

(i) We define the N-union $\cup_N : \mathscr{B} \times \mathscr{B} \longrightarrow \mathscr{B}$ by

$$A \cup_N B = \begin{cases} A \cup_s B & \text{if } \#(A \cup_s B) \leqq N \\ \varnothing & \text{otherwise.} \end{cases}$$

(ii) The parallel composition $\|_N : \mathscr{P}^*(\mathscr{B}^{st}) \times \mathscr{P}^*(\mathscr{B}^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ with respect to the bounded number of processors model is defined as the lifted extended version of $\|_N$, $\lfloor\!\lfloor_N$, $|_N : \mathscr{B}^f \times \mathscr{B}^f \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ such that

$x \|_N y = x \lfloor\!\lfloor_N y + y \lfloor\!\lfloor_N x + x |_N y$
$\varepsilon \lfloor\!\lfloor_N y = \{y\}$
$\perp \lfloor\!\lfloor_N y = \{\perp\}$
$A x' \lfloor\!\lfloor_N y = A(x' \|_N y)$
$\varepsilon |_N y = x |_N \varepsilon = \varnothing$
$\perp |_N y = x |_N \perp = \varnothing$
$A x' |_N B y' = (A \cup_N B)(x' \|_N y')$.

Note that Def. 4.1 applies to the $\mathscr{B}_0$-*case* (without synchronization) as well as the $\mathscr{B}_1$-*case* (with synchronization), cf., the note following Def. 3.7.

On the basis of these operators we can give a proper semantics for our language(s) with respect to this model of true concurrency with a bounded of processors. We leave the definition of the semantic function to the reader. (Note that alternatively, this semantics can be obtained from MP, by application of a suitable abstraction operator.)

It is easy to verify that in case $N = 1$ the semantics based on $\cup_N$ specializes to a purely interleaving semantics such as in Meyer (1985). So this model of true concurrency is a true generalization of the interleaving model.

For example, if $N = 3$ the meaning of

$([a, b] \| [a]) \| [b]$

becomes the stream set

$(\{[a, b]\} \|_N \{[a]\}) \|_N \{[b]\}$
$= \{[a, b][a], [a][a, b], [a, a, b]\} \|_N \{[b]\}$
$= \{[a, b, b][a], [a, b][a, b], [a][a, b, b],$
$[b][a, b][a], [b][a][a,b],$
$[b][a, a, b], [a, b][b][a],$
$[a][b][a, b], [a, a, b][b], [a, b][a][b],$
$[a][a, b][b]\},$

so we do not have $[a, a, b, b]$. Analogously

$([a, b] \| [c]) \| [\bar{c}]$

denotes (after failure removal)

$\{[a, b][\tau, \tau], [\tau, \tau][a, b]\}$

for $N = 3$, but denotes

$\{[a, b][\tau, \tau], [\tau, \tau][a, b], [\tau, \tau, a, b]\}$

for $N = 4$.

It is clear that in the context of a bounded number (say N) of processors, we can restrict ourselves in

our semantic domain to multi-sets of cardinality $\leqq N$. If we also have finiteness of the alphabet $\mathscr{E}$ then each stream set X is trivially locally finite, since $X^{[n]} \subseteq \mathscr{B}^n \cup \mathscr{B}^n.\perp$ and the set on the right-hand side is finite under these assumptions. In this special case of a finite number of elementary actions on a bounded number of processes we have immediately that the semantical operators $\cdot$, $+$, and $\|$ preserve compactness (and are therefore well-defined), provided that they preserve closedness.

4.2 Atomic actions that take more than one unit of time

As we admitted already, it is not very realistic to assume that atomic actions take only one tick of the global clock. We shall now see how to generalize our semantics in order to cater for atomic actions that may take longer to be executed.

We do this by splitting up atomic actions a into sequences of "subatomic" actions that take one unit (tick) of time: the atomic action $a = a^{(1)}a^{(2)} \ldots a^{(n)}$ takes n units of time to be executed. Of course this is still not sufficient to deal with these actions properly: although $a = a^{(1)}a^{(2)} \ldots a^{(n)}$ is now written in terms of subatomic part(icle)s $a^{(i)}$, and can thus be interpreted in the model by the stream $[a^{(1)}][a^{(2)}] \ldots [a^{(n)}]$, we have to be careful that these atomic actions may not be interrupted by interleaving with other actions. So, for example, if we consider $a \| b$ for $a = a^{(1)}a^{(2)}$ and $b = b^{(1)}$, we obtain in the model with subatomic action as resulting set $\{[a^{(1)}, b^{(1)}][a^{(2)}], [a^{(1)}][a^{(2)}, b^{(1)}], [a^{(1)}][a^{(2)}][b^{(1)}], [b^{(1)}][a^{(1)}][a^{(2)}]\}$. This set does *not* include the possibility $[a^{(1)}][b^{(1)}][a^{(2)}]$, which expresses that the atomic action a is interrupted for the execution of b!

In the multi-processor model unfortunately this stream is included in the semantics of $a \| b = a^{(1)}a^{(2)} \| b^{(1)}$. Therefore, we have to modify our parallel execution operator. However, we can still deal with concurrency with subatomic actions within the frame of Sect. 2.

4.2. Definition. Let $\mathscr{A}^{>1}$ be the collection of actions of the format $a^{(i)}$ with $i > 1$. The parallel composition $\|_{sa} : \mathscr{P}^*(\mathscr{B}^{st}) \times \mathscr{P}^*(\mathscr{B}^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ with respect to subatomic actions is defined as the lifted extended version of $\|_{sa}$, $\|_{sa}$, $|_{sa}$: $\mathscr{B}^f \times \mathscr{B}^f \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ such that

$x \|_{sa} y = x \|_{sa} y + y \|_{sa} x + x|_{sa} y$
$\varepsilon \|_{sa} y = \{y\}$
$\perp \|_{sa} y = \{\perp\}$
$Ax' \|_{sa} y = A(x' \|_{sa} y)$
if $y = By'$ and $B \cap \mathscr{A}^{>1} = \varnothing$
$Ax' \|_{sa} y = \varnothing$ otherwise
$\varepsilon|_{sa} y = x|_{sa} \varepsilon = \varnothing$
$\perp|_{sa} y = x|_{sa} \perp = \varnothing$
$Ax'|_{sa} By' = (A \cup_s B)(x' \|_{sa} y')$.

We calculate as illustration the stream set of the above example:

$$
\begin{aligned}
&[a^{(1)}][a^{(2)}] \|_{sa} [b^{(1)}] \\
&= [a^{(1)}][a^{(2)}] \|_{sa} [b^{(1)}] + [b^{(1)}] \|_{sa} [a^{(1)}][a^{(2)}] \\
&\quad + [a^{(1)}][a^{(2)}]|_{sa} [b^{(1)}] \\
&= [a^{(1)}]([a^{(2)}] \|_{sa} [b^{(1)}]) + [b^{(1)}](\varepsilon \|_{sa} [a^{(1)}][a^{(2)}]) \\
&\quad + [a^{(1)}, b^{(1)}]([a^{(2)}] \|_{sa} \varepsilon) \\
&= [a^{(1)}]([a^{(2)}] \|_{sa} [b^{(1)}]) + [a^{(1)}]([b^{(1)}] \|_{sa} [a^{(2)}]) \\
&\quad + [a^{(1)}]([a^{(2)}]|_{sa} [b^{(1)}]) + [b^{(1)}](\varepsilon \|_{sa} [a^{(1)}][a^{(2)}]) \\
&\quad + [a^{(1)}, b^{(1)}][a^{(2)}] = \ldots = [a^{(1)}][a^{(2)}][b^{(1)}] + \varnothing \\
&\quad + [a^{(1)}][a^{(2)}, b^{(1)}] + [b^{(1)}][a^{(1)}][a^{(2)}]) \\
&\quad + [a^{(1)}, b^{(1)}][a^{(2)}].
\end{aligned}
$$

Note that indeed the resulting set does not include the stream $[a^{(1)}][b^{(1)}][a^{(2)}]$!

4.3 Maximal parallelism

At the end of Sect. 3 it was pointed out that the synchronous start semantics does not satisfactorily deal with synchronization primitives. In some sense the concurrency operator $\|_{ss}$ optimizes schedulings in too local a manner. The multi-processor semantics for $\mathscr{L}_1$ does not suffer from this; both synchronous co-operation *and interleaving* are modeled with the concurrency operator $\|_{mp}$. This however, may be regarded as undesirable:

Consider again the statement $(a \| c) \| (b; \bar{c})$. In the multi-processor semantics this statement yields (after failure removal) the stream set

$\{[a][b][\tau, \tau], [a, b][\tau, \tau], [b][a][\tau, \tau],$
$[b][a, \tau, \tau], [b][\tau, \tau][a]\}$.

One might object that the semantics MP_1' gives *too* many outcomes, due to unnecessary interleavings. We shall remedy this shortcoming by the introduction of the abstraction operator $maxpar$. $maxpar$ will select the fastest (successfully synchronized) scheduling. For the above example we shall obtain in the maximal parallelism model the stream set $\{[a, b][\tau, \tau], [b][a, \tau, \tau]\}$.

The operator $maxpar$ is introduced as follows. First we define an ordering $\leqq_{maxpar}$ on streams. $x \leqq_{maxpar} y$ if x can be obtained by taking together (multi-set union) consecutive actions in y. So there is a many-one correspondence between actions in

y and actions in x. For example

$[a, b][\tau, \tau] \leqq_{maxpar} [a][b][\tau, \tau]$

but *not*

$[a, b][\tau, \tau] \leqq_{maxpar} [b][a, \tau, \tau]$.

The relation $\leqq_{maxpar}$ will be defined (as usual) in two stages: first on finite streams, then on arbitrary streams with the help of truncations. On the basis of $\leqq_{maxpar}$ we define the operator $maxpar$ that takes the minimal streams from a stream set with respect to the maximal parallelism ordering.

4.3. Definition

(i) The maximal parallelism ordering $\leqq_{maxpar}$ on $\mathscr{B}^f$ is defined by the following clauses:
$\varepsilon \leqq_{maxpar} \varepsilon$
$\perp \leqq_{maxpar} \perp$
$A \leqq_{maxpar} A_1 \ldots A_k$
if $A = A_1 \cup_m \ldots \cup_m A_k$, $(k \geqq 1)$
$A\,x' \leqq_{maxpar} y$ if $\exists\, y_A, y_{x'} \in \mathscr{B}^f$:
$A \leqq_{maxpar} y_A \wedge x' \leqq_{maxpar} y_{x'} \wedge y_A\, y_{x'} = y$.

(ii) The maximal parallelism ordering $\leqq_{maxpar}$ on $\mathscr{B}^{st}$ is defined by $x \leqq_{maxpar} y$ iff $\forall m\, \exists n \geqq m$: $x[m] \leqq_{maxpar} y[n]$.

(iii) The abstraction operator $maxpar$:
$\mathscr{P}(\mathscr{B}^{st}) \longrightarrow \mathscr{P}(\mathscr{B}^{st})$
is defined by
$maxpar(X) = \{x \in X \mid \neg\, \exists\, x' \in X : x' <_{maxpar} x\}$.

Note the restriction $n \geqq m$ in 4.3(ii). This condition excludes pathological situations as

$[a]\perp \leqq_{maxpar} [a][b]$.

($[a][b]$ is *not* a serialization of $[a]\perp$.)

We use the maximal parallelism operator to derive a new multi-processor semantics, as we did before with the failure removal operator fr.
(Note however that unfortunately this abstraction operator is too weak, in that it excludes comparison of, e.g., $[a][a, b][a, b]$ ... and $[a, b][a, b]$ ... in $MP_1'[\![\mu\xi[a; b\xi] \,\|\, \mu\zeta[b; \zeta]]\!]$. This touches upon fairness issues that fall outside the scope of this paper.)

4.4. Definition (Multi-processor semantic with failure removal and maximal parallelism for $\mathscr{L}_1$). The semantics $MP_1'' : \mathscr{L}_1 \longrightarrow Env_1 \longrightarrow \mathscr{P}(\mathscr{B}_0^{st})$ is defined by $MP_1'' = maxpar \circ fr \circ MP_1$.

From Koymans et al. (1985) we adopt the example $s = ((c_1 \,\|\, c_2) \,\|\, (\bar{c}_2 ;\ c_1)) \,\|\, (\bar{c}_1 ;\ \bar{c}_1)$. We calculate the multi-processor semantics $MP_1[\![s]\!]$ (already anticipating to failure removal):

$(\{[c_1][c_2], [c_2][c_1], [c_1, c_2]\} \|_{mp} \{[\bar{c}_2][c_1]\}) \|_{mp} \{[\bar{c}_1][\bar{c}_1]\}$
$= \{[c_1][\tau, \tau][c_1], [\tau, \tau][c_1, c_1], [\tau, \tau][c_1][c_1], [\tau, \tau, c_1][c_1], \ldots\} \|_{mp} \{[\bar{c}_1][\bar{c}_1]\}$
$= \{[\tau, \tau][\tau, \tau][\tau, \tau], [\tau, \tau, \tau, \tau][\tau, \tau], \ldots\}$.

Application of fr yields

$\{[\tau, \tau][\tau, \tau][\tau, \tau], [\tau, \tau, \tau, \tau][\tau, \tau]\}$.

Finally we arrive at $\{[\tau, \tau, \tau, \tau][\tau, \tau]\}$, since

$[\tau, \tau, \tau, \tau][\tau, \tau] \leqq_{maxpar} [\tau, \tau][\tau, \tau][\tau, \tau]$.

4.4 The delay action and real-time semantics

The considerations of Subsect. 4.2 enable us to give a proper treatment of an operator that is associated with *real-time* programming, viz. the *delay* operator (cf., Koymans et al. 1985). So now, in this subsection, we include the special *atomic* action $d(n)$ in the syntax of our language(s), denoting a delay of n units of time. It is important to stress that this action $d(n)$ is an atomic one: it is not intended to be interrupted by some interleaving of another action. This would clearly be contradictionary with our intuitions concerning a delay of n units of (real) time.

In our set-up we can express the delay action $d(n)$ as an atomic action that takes n units of time to execute: $d^{(1)}, \ldots, d^{(n)}$. (The $d^{(i)}$ may be interpreted as dummy or skip (sub)actions expressing waiting one unit of time, but this is not within the scope of our uniform semantics.) To treat delays properly in this manner it is essential to use a model with true concurrency as expounded in Subsect. 4.2. Moreover, when considering e.g., the parallel statement $a \,\|\, (d^{(1)};\ b)$, it is not intended to include the scheduling possibility $[a][d^{(1)}][b]$, because this would again not correspond to our intuition about a delay (of one unit of time, in this case). We have to require that delays start execution as soon (fast) as possible. Even under the assumption of maximal parallelism as has been discussed in the previous subsection this requirement is not fulfilled automatically, as can be observed from the following example:

$maxpar(MP_1[\![a; b \,\|\, d^{(1)}]\!])$
$= maxpar(\{[a][d^{(1)}][b], [a][b][d^{(1)}], [a][b, d^{(1)}][a, d^{(1)}][b], [d^{(1)}][a][b]\})$
$= \{[a, d^{(1)}][b], [a][b, d^{(1)}]\}$.

Clearly, oinly the first outcome in the resulting set is a correct outcome. However, it is easy to remedy

this by imposing an extra property of $\|_{sa}$ regarding delays:

4.5. Definition. Let $\mathscr{A}^{>1}$ be the collection of actions of the format $a^{(i)}$ with $i>1$ and $\mathscr{D}$ be the collection of subatomic parts of delay actions, i.e., actions of the format $d^{(i)}$. The parallel composition $\|_{rt}: \mathscr{P}^*(\mathscr{B}^{st}) \times \mathscr{P}^*(\mathscr{B}^{st}) \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ in the real-time model is defined as the lifted extendeed version of $\|_{rt}, \|_{rt}, |_{rt}: \mathscr{B}^f \times \mathscr{B}^f \longrightarrow \mathscr{P}^*(\mathscr{B}^{st})$ such that

$x\|_{rt}y = x\|_{rt}y + y\|_{rt}x + x|_{rt}y$
$\varepsilon\|_{rt}y = \{y\}$
$\perp\|_{rt}y = \{\perp\}$
$Ax'\|_{rt}y = A(x'\|_{rt}y)$ if $y = By', B \cap \mathscr{A}^{>1} = \varnothing$ and $B \cap \mathscr{D} = \varnothing$
$Ax'\|_{rt}y = \varnothing$ otherwise
$\varepsilon|_{rt}y = x|_{rt}\varepsilon = \varnothing$
$\perp|_{rt}y = x|_{rt}\perp = \varnothing$
$Ax'|_{rt}By' = (A \cup_N B)(x'\|_{rt}y')$.

The operator $\|_{rt}$ is continuous as can be checked with the usual tools. On the basis of this real-time concurrent operator one may define a semantical function along the lines of the previous sections.

5 A branching-time step semantics for true concurrency

In the previous sections we have occupied ourselves with so-called "linear-time" semantics for our language(s). It is well-known (cf., Milner 1980) that – in the context of synchronization – if one is interested in an analysis of *deadlock* behaviour, a linear-time semantics of the kind we have presented is not adequate. One has to refine the semantic model by means of such notions as *ready* or *failure sets* (cf., Hoare 1985; Olderog and Hoare 1986; Taubner and Vogler 1987) or extend the semantics to a *"branching-time"* one, in which one can suitably model the structure of nondeterministic choices (cf., de Bakker et al. 1989).

In this section we indicate how to work out the latter possibility, although we expect no serious problems when pursuing the former alternative. Here we shall employ de Bakker-Zucker branching-time processes. In effect, it appears to be fairly straightforward to extend the domains appearing in de Bakker and Zucker (1982) and de Bakker and Meyer (1987), such that we can model one of our versions of true concurrency.

In this section we restrict ourselves to statements of $\mathscr{L}_1$ that are *guarded*. Informally, this means that every call of the variable ξ in a construct $\mu\xi[s]$ must be (semantically) preceded by (the execution of) some elementary action. So, for example, $\mu\xi[a;\xi]$ and $\mu[(a;\xi)\|b]$ are guarded, but $\mu\xi[\xi]$, $\mu\xi[\xi;a]$ and $\mu\xi[a\|\xi]$ are not.

We use a semantic reflexive domain P given by the following domain equation

$P \cong \mathscr{P}_c(\{\perp\} \cup \mathscr{B} \cup (\mathscr{B} \times P))$

where $\mathscr{P}_c(\cdot)$ denotes the powerset of closed subsets. Here $\perp$ acts as a nil-process. P is a complete metric space.

The definitions of the operators $\cdot$ and $+$ are analogous to those in de Bakker and Zucker (1982) and de Bakker et al. (1986). We now proceed with the definition of $\|_{bt}$.

5.1. Definition.

(i) The operator $\|_{bt}$ on the subset of P with processes of finite depth is given by:
$p\|_{bt}q = (p\|_{bt}q) + (q\|_{bt}p) + (p|_{bt}q)$
$\perp\|_{bt}q = \perp$
$A\|_{bt}q = \langle A, q\rangle$
$\langle A, p'\rangle\|_{bt}q = \langle A, p'\|q\rangle$
$p\|_{bt}q = \cup\{x\|q \mid x \in p\}$
$\perp|_{bt}q = p|_{bt}\perp = \perp$
$A|_{bt}B$ as in the linear-time case
$\langle A, p\rangle|_{bt}B = \{\langle A|_{bt}B, p\rangle\}$
$A|_{bt}\langle B, q\rangle = \{\langle A|_{bt}B, q\rangle\}$
$\langle A, p\rangle|_{bt}\langle B, q\rangle = \{\langle A|_{bt}B, p\|_{bt}q\rangle\}$
$p|_{bt}q = \cup\{x|_{bt}y \mid x \in p, y \in q\}$.

(ii) The operator $\|_{bt}$ on the domain P with processes of arbitrary depth is given by:
$p\|_{bt}q = lim_n\, p^{(n)}\|_{bt}q^{(n)}$.

Now we may define our branching-time semantics for our (guarded) language. We use a similar scheme as we did for the linear-time semantics. In the clause for the recursive μ-construct we take a unique fixed point (of a contracting operator on a complete metric space) where we took a least fixed point (of a continuous operator on a cpo) before.

5.2. Definition (Branching-time semantics for the guarded sublanguage of $\mathscr{L}_1$). We take the reflexive domain P as before. Let $Env = \Xi \longrightarrow P$. We define the semantics BT: $\mathscr{L} \longrightarrow Env \longrightarrow P$ with respect to the branching time model by the clauses:

$BT[\![e]\!](\eta) = \{[e]\}$ for $e \in \mathscr{E}$
$BT[\![s_1; s_2]\!](\eta) = BT[\![s_1]\!](\eta) \cdot BT[\![s_2]\!](\eta)$
$BT[\![s_1 \cup s_2]\!](\eta) = BT[\![s_1]\!](\eta) + BT[\![s_2]\!](\eta)$
$BT[\![s_1 \| s_2]\!](\eta) = BT[\![s_1]\!](\eta)\|_{bt}BT[\![s_2]\!](\eta)$
$BT[\![\xi]\!](\eta) = \eta(\xi)$

$BT[\![\mu\xi[s]]\!](\eta) = ufp(\Psi_{s,\eta})$
where $\Psi_{s,\eta} = \lambda p.BT[\![s]]\!](\eta\{p/\xi\})$.

In the above definition we have assigned to the recursive μ-construct the unique fixed point of $\Psi_{s,\eta} = \lambda p.BT[\![s]]\!](\eta\{p/\xi\})$. The existence of this unique fixed point is guaranteed by the counterpart of Lemma 2.15: for guarded statements s the function $\Psi_{s,\eta}$ is contracting. We then use Banach's fixed point theorem for complete metric spaces, to derive that the unique fixed point of $\Psi_{s,\eta}$ exists. (Cf., de Bakker and Zucker 1982.)

Concluding this section we may remark that in effect we have indicated that de Bakker-Zucker process theory is suited for a treatment of true concurrency (as well as a correct analysis of deadlock behaviour).

6 Conclusion

In this paper we have attempted to show that much work done on denotational semantics in the context of the interleaving model can be extended to models of "true" concurrency. We have presented several more or less sophisticated semantic models that treat various aspects of "true" concurrency in a form which is known as step semantics. The semantical definitions are developed by means of a general method available for the Smyth powerdomain. We have chosen this powerdomain, since it is simpler than the Egli-Milner one, and still is adequate for infinite behaviour. (Our results are also valid if one would adopt the Egli-Milner powerdomain instead.)

One might object that the role of $\perp$ in the context of the Smyth powerdomain is not compatible with the intuitions regarding forms of "true" concurrency, since we have that $X \cup \{\perp\} \equiv_S \{\perp\}$ and nontermination in a distributed environment should not be that catastrophic as is suggested by this equivalence. Our answer to this objection is that $\perp$ must be viewed as an *atomic* divergence rather than an *internal* divergence: if one still wants $\perp$ to represent the meaning of a divergence by hiding (such as e.g., $\mu\xi[a; \xi]$ where a is hided), the consequence is that one should be careful to postpone the hiding of internal actions to a stadium where the context has been taken into consideration. For example, consider

$$X = SS[\![\mu\xi[a; \xi] \|_{ss} \mu\eta[b; \eta]]\!] = \{[a, b]^\omega\}.$$

Hiding a in X would yield $\{[b]^\omega\}$, whereas hiding a directly in $SS[\![\mu\xi[a; \xi]]\!]$ yields the uninforming result $\{\perp\}$!

Following the presentation of our basic semantic models we have investigated refinements of these models in order to deal with a bound on the number of available processors, actions that take more than one unit of time and a delay construct. Furthermore, we have encountered a notion of maximal parallelism similar to that appearing in work on real-time semantics (Koymans et al. 1985). We stress that the continuity proofs of the various parallel operators were facilitated considerably by a number of general considerations such as the lifting of continuous functions on streams to compact stream sets.

Although we have considered a very simple linear-time model, we believe that our results can be generalized in a straightforward manner to more refined models such as failure step semantics in the sense of Taubner and Vogler (1987). Moreover, we also expect that the technical metric results of papers of de Bakker et al. carry over to this framework of true concurrency. In particular, it seems also possible to give an operational semantics for our languages with true concurrency based upon transition systems in the style of Plotkin (1980). The transition systems needed are variants of those of de Bakker et al. (1989) and de Bakker and Meyer (1987), rendered suitable for streams of action packages rather than actions on their own. Moreover, a rule is needed to express the character of true concurrency, such as (in the notation of de Bakker and Meyer (1987), for the language $\mathcal{L}_0$ without synchronization):

$$\frac{s_1 \longrightarrow^A s' \quad s_2 \longrightarrow^B s''}{s_1 \| s_2 \longrightarrow^{A \cup_s B} s' \| s''}.$$

We expect no problems to relate this operational semantics to the denotational one along the lines of de Bakker et al. (1989), Kok and Rutten (1988) and de Bakker and Meyer (1987).

References

Aceto L, de Nicola R, Fantechi A (1987) Testing equivalences for event structures. In: Venturini Zilli M (ed) Proc Advanced School on Mathematical Models for the Semantics of Parallelism (Lect Notes Comput Sci 280) Springer, Berlin Heidelberg New York Tokyo, pp 1–20

Back RJ (1983) A continuous semantics for unbounded nondeterminism. TCS 23:187–210

de Bakker JW (1980) Mathematical theory of program correctness. Prentice Hall, London

Boudol G, Castellani I (1987) On the semantics of concurrency: partial orders and transition systems. In: Ehrig H et al. (eds) Proc TAPSOFT/CAAP '87 (Lect Notes Comput Sci 249) Springer, Berlin Heidelberg New York Tokyo, pp 123–137

Boudol G, Castellani I (1987) Concurrency and atomicity, rapports de recherche 748, INRIA Sophia Antipolis

Bergstra JA, Klop JW (1984) Process algebra for synchronous communication. Inf Control 60:109–137

de Bakker JW, Kok JN, Meyer J-JC, Olderog E-R, Zucker JI (1986) Contrasting themes in the semantics of imperative concurrency. In: de Bakker JW, de Roever WP, Rozenberg G (eds) (Lect Notes Comput Sci 224) Springer, Berlin Heidelberg New York Tokyo, pp 51–121

de Bakker JW, Meyer J-JC (1987) Metric semantics for concurrency, Tech Rep IR-139, Free University, Amsterdam

de Bakker JW, Meyer J-JC, Olderog E-R, Zucker JI (1989) Transition systems, metric spaces and ready sets in the semantics of uniform concurrency. JCSS 36:158–224

Broy M (1986) A theory for nondeterminism, parallelism, communication and concurrency. TCS 45:1–62

de Bakker JW, Zucker JI (1982) Processes and the denotational semantics of concurrency. Inf Control 54:70–120

de Nicola R, Hennessy M (1987) CCS without τ's, In: Ehrig H et al. (eds) Proc TAPSOFT/CAAP '87 (Lect Notes Comput Sci 249) Springer, Berlin Heidelberg New York Tokyo, pp 138–152

Degano P, de Nicola R, Montanari U (1987) CCS is an (augmented) contact free C/E system. In: Venturini Zilli M (ed) Proc Advanced School on Mathematical Models for the Semantics of Parallelism (Lect Notes Comput Sci 280) Springer, Berlin Heidelberg New York Tokyo, 144–165

Francez N, Hoare CAR, Lehmann DJ, de Roever WP (1979) Semantics of nondeterminism, concurrency and communication. JCSS 19:290–308

van Glabbeek RJ, Vaandrager FW (1987) Petri net models for algebraic theories of concurrency. In: de Bakker et al. (eds) Proc PARLE (Lect Notes Comput Sci 259) Springer, Berlin Heidelberg New York Tokyo, pp 224–242

Janicki R (1987) A formal semantics for concurrent systems with a priority relation. Acta Inf 24:33–55

Hoare CAR (1985) Communicating sequential processes. Prentice-Hall Englewood Cliffs, New Jersey

Kok JN, Rutten JJMM (1988) Contractions in comparing concurrency semantics. In: Lepistö T, Salomaa A (eds) Proc ICALP '88 (Lect Notes Comput Sci 317) Springer, Berlin Heidelberg New York Tokyo, pp 317–332

Koymans R, Shyamasundar RK, de Roever WP, Gerth R, Arun-Kumar S (1985) Compositional semantics for real-time distributed computing. In: Parikh R (ed) Proc Logics of Programs (Lect Notes Comput Sci 193) Springer, Berlin Heidelberg New York, pp 167–189

Mazurkiewicz A (1978) Concurrent program schemes and their interpretation, Rep DIAMI PB-78, Comput Sci Dept, Aarhus University, Aarhus, Denmark

Meyer J-JC (1985) Programming calculi based on fixed point transformations: semantics and applications. Dissertation, Free University, Amsterdam

Milner R (1980) A calculus for communicating systems (Lect Notes Comput Sci 92) Springer, Berlin Heidelberg New York Tokyo

Meyer J-JC, de Vink EP (1987) Applications of compactness in the Smyth powerdomain of streams. In: Ehrig H et al. (eds) Proc TAPSOFT/CAAP '87. (Lect Notes Comput Sci 249) Springer, Berlin Heidelberg New York Tokyo, 241–255

Olderog E-R, Hoare CAR (1986) Specification oriented semantics for communicating processes. Acta Inf 23:9–66

Plotkin GD (1976) A powerdomain construction. SIAM J Comput 5:452–487

Plotkin GD (1980) An operational semantics for CSP. In: Bjørner D (ed) Formal description of programming concepts II (Lect Notes Comput Sci 86) Springer, Berlin Heidelberg New York Tokyo, pp 527–553

Reisig W (1985) Petri nets. Springer, Berlin Heidelberg New York Tokyo

Rozenberg G, Verraedt R (1983) Subset languages of Petri nets I: the relationship to string languages and normal forms. TCS 26:301–326

Salwicki A, Müldner T (1981) On the algebraic properties of concurrent programs. In: Engeler E (ed) Proc Logic of Programs (Lect Notes Comput Sci 125) Springer, Berlin Heidelberg New York, pp 169–197

Smyth MB (1978) Power domains. JCSS 16:23–36

Taubner DA, Vogler W (1987) The step failure semantics. In: Brandenburg FJ, Vidal Naquet G (eds) Proc STACS '87 (Lect Notes Comput Sci 247) Springer, Berlin Heidelberg New York Tokyo, pp 348–359

Winskel G (1980) Events in computation. Ph.D. Thesis, Edinburgh University, Edinburgh