Kahn's fixed-point characterization for linear dynamic networks

Shan-Hwei Nienhuys-Cheng Arie de Bruin
cheng@cs.few.eur.nl arie@cs.few.eur.nl

Dept of Comp. Science, Erasmus University Rotterdam
P.O. Box 1738, 3000DR, Rotterdam, the Netherlands

Abstract. We consider dynamic Kahn-like dataflow networks defined by a simple language **L** containing the fork-statement. The first part of the Kahn principle states that such networks are deterministic on the I/O level : for each network, different executions provided with the same input deliver the same output. The second part of the principle states that the function from input streams to output streams (which is now defined because of the first part) can be obtained as a fixed point of a suitable operator derived from the network specification. The first part has been proven by us in [BN96, BN97]. To prove the second part, we will use the metric framework. We introduce a nondeterministic transition system **NT** from which we derive an operational semantics O_n. We also define a deterministic transition system **DT** and prove that the operational semantics O_d derived from **DT** is the same as O_n. Finally, we define a denotational semantics D and prove $D = O_d$. This implies $O_n = D$. Thus the second part of the Kahn principle is established.

Keywords the Kahn principle, operational semantics, denotational semantics.

1 Introduction

A dataflow network consists of a number of parallel processes which are interconnected by directed channels. Processes communicate with each other only through these channels, they share no variables. The channels act as possibly infinite FIFO queues. The Kahn principle concerns networks with deterministic nodes: each process selects the channel it uses next in a deterministic way. However, which process gets the chance to execute a step at a certain moment depends on the underlying transition system. Such a system can be deterministic or nondeterministic. In a nondeterministic system, which process will execute next is not pre-determined. Thus different executions (computations) are possible.

In his seminal paper [K74], Kahn characterizes the processes as functions, transforming input histories into output histories, where a history models a stream of values which has appeared on a channel during a computation. He then states a conjecture which has since then become known as the Kahn principle : a network consisting of deterministic nodes computes a history function, and for each set of input histories on the input channels, the output histories can be obtained as the smallest solution of a set of equations derived from the network.

Thus the Kahn principle contains two parts. Part 1: The nondeterminism caused by the asynchronicity of the computing processes does not lead to global nondeterminism in the history level I/O-behaviour of the network. This means that given an initial state of the variables in the processes and a set of input streams, the set of output streams is uniquely determined, independent of the computations. This function can be used to define the operational semantics for the program which has induced these processes. Part 2: A meaning can be given to the language (denotational semantics) by defining it as a special solution of a set of equations and furthermore, the function defined in part 1 is in fact this solution. In other words, the operational semantics is equal to the denotational semantics.

The Kahn principle has been proved for certain types of networks [C72, A81, LS89], i.e., for static networks, but not for dynamic ones where it is possible that a single process expands into a new network of processes. Here we investigate a simple dynamic language **L** in which it is possible to dynamically create linear arrays of processes, not unlike a unix pipeline which can be built up using the unix primitive 'pipe' and 'fork'.

In [BN96, BN97], we have defined a nondeterministic transition system **NT** for **L** describing all possible interleavings and proved that all computations either give the same maximal output (when the computation is fair), or a prefix of it. Thus we have established the first part of the Kahn principle for linear dynamic networks.

In this article, we establish the second part of the Kahn principle. A denotational semantics is defined as the unique fixed point of a contraction in a complete metric space. To prove the equivalence between the operational semantics based on this asynchronous interleaved transition system **NT** and the denotational semantics, i.e. $O_n = D$, we introduce a synchronous deterministic transition system **DT** as a bridge. The idea is to prove the equivalence of operational semantics corresponding with **DT** and **NT** ($O_n = O_d$) first. Then, by establishing the equivalence between the denotational semantics and the operational semantics based on **DT** ($O_d = D$), we have also established the equivalence $O_n = D$. The proof of $O_d = D$ is similar to [BBB93] and section 9.2 in [BV96]. Thus our contributions are mainly the introduction of **DT** and the introduction of concepts such as big steps, etc. to establish $O_n = O_d$.

2 Operational Semantics in NT

We first introduce some statements in **L** informally. For a given input stream and an initial state, the execution of a program in **L** produces an output stream. Initially the program is executed by one process, using precisely one input and one output channel. Execution of the statement $read(x)$ will fetch the next value from the input channel and assign it to the variable x. Execution of the statement $write(e)$ will evaluate the expression e and write the resulting value to the output channel. A process can split up into two new, nearly identical subprocesses, a mother process and a daughter process. This effect is achieved by a statement of the form $fork(w)$. The original input channel becomes the mother's input channel and likewise the original output channel becomes the daughter's output channel. Between mother and daughter a new channel is created, which is the mother's output and the daughter's input channel. This channel will originally be empty. Both processes proceed by executing the statement following $fork(w)$. We distinguish between mother and daughter by giving the variable w in the mother process the value 0, in the daughter the value 1.

2.1 Preliminaries

- Let $(v \in) \mathbf{Var}$, $(\alpha, \beta, \gamma \in) \mathbf{Val}$, $(e \in) \mathbf{Exp}$ and $(b \in) \mathbf{Bexp}$ be given sets. They are called the set of *variables*, the set of *values*, the set of *expressions* and the set of *boolean expressions*, respectively. A *state* is a function $\sigma : \mathbf{Var} \to \mathbf{Val}$. Let **State** be the set of all states. The notation $\sigma\{\beta/v\}$ is used to denote a state equal to σ, except that now $\sigma(v) = \beta$. Two functions $V : \mathbf{Exp} \to (\mathbf{State} \to \mathbf{Val})$ and $B : \mathbf{Bexp} \to (\mathbf{State} \to \{true, false\})$ are assumed to be available.
- Let **L** be the language which contains the following statements:
  $s ::= v := e \mid skip \mid write(e) \mid read(v) \mid fork(v) \mid s;\ s \mid$
  $\mathbf{if}\ b\ \mathbf{then}\ s\ \mathbf{else}\ s\ \mathbf{fi} \mid \mathbf{while}\ b\ \mathbf{do}\ s\ \mathbf{od}$
- Consider a value $\tau \notin \mathbf{Val}$. Let $\mathbf{V} = \mathbf{Val} \cup \{\tau\}$. Let $(\eta, \zeta, \xi \in) \mathbf{V}^\infty$ be the set of finite or infinite sequences of elements from $\mathbf{V}$. Such a sequence is called a *stream*. Let ϵ be the *empty stream*. For two streams ξ and ζ, let $\xi \cdot \zeta (= \xi\zeta)$ be their concatenation. If $\zeta = \alpha\eta, \alpha \in \mathbf{V}$, then $first(\zeta) = \alpha$ and $rest(\zeta) = \eta$.
- A process is a function $P : \mathbf{State} \to (\mathbf{V}^\infty \to \mathbf{V}^\infty)$. The set of all processes is **Proc**
- Let E denote termination. A resumption is recursively defined by $r ::= E \mid s : r$, where s in **L**. Let **Res** be the set of all resumptions.

2.2 Configurations

We will use 'configurations' to describe snapshots of a network at work. Later we will define a nondeterministic transition system **NT** which models the basic steps a configuration can take in an execution.

$$a : \xleftarrow{\epsilon} \boxed{read(y) : E, \{v = 1\}} \xleftarrow{\epsilon} \boxed{write(x) : E, \{x = 3, v = 0\}} \xleftarrow{4 \cdot 5}$$
$$b : \xleftarrow{\epsilon} \boxed{read(y) : E, \{v = 1\}} \xleftarrow{3} \boxed{E, \{x = 3, v = 0\}} \xleftarrow{4 \cdot 5}$$
$$b' : \xleftarrow{\epsilon} \boxed{read(y) : E, \{v = 1\}} \xleftarrow{\epsilon} \boxed{write(x) : E, \{x = 3, v = 0\}} \xleftarrow{4} \boxed{0^{th}\text{process}} \xleftarrow{5}$$

For example, consider the situation shown in item a above. A box represents a subprocess and it contains information about the state and the program still to be executed. They are connected by arrows representing I/O channels. We allow 3 to be written without bothering whether it is needed by the next subprocess (item b above). To describe such situations by configurations, we need a buffer stream containing what is written and not yet used in the input channel of a subprocess. We extend this model slightly in the following way. For consistency we also introduce a buffer stream for the first process from the right. Initially this stream equals ϵ. We can view this situation as if there is still another subprocess (the 0-th subprocess) which contains hidden *read*+ *write* statements. All it does, is to fetch the next element from the global input stream and to write it into the buffer of the first subprocess (see item b' above). The configuration we use to describe item a is given below in $\underline{a}$. (Notice that the buffer stream of the first subprocess is given as ϵ.) We can consider a transition from $\underline{a}$ to $\underline{b}$ caused by the 1st subprocess and a transition to $\underline{b'}$ caused by the 0-th subprocess.

$\underline{a} : \langle read(y) : E, \{v = 1\}, \epsilon, \langle write(x) : E, \quad \{x = 3, v = 0\}, \epsilon, 4 \cdot 5\rangle\rangle$
$\underline{b} : \langle read(y) : E, \{v = 1\}, 3, \langle E, \{x = 3, v = 0\}, \epsilon, 4 \cdot 5\rangle\rangle$
$\underline{b'} : \langle read(y) : E, \{v = 1\}, \epsilon, \langle write(x) : E, \quad \{x = 3, v = 0\}, 4, 5\rangle\rangle$

Definition 1. A configuration ρ is recursively defined as $\rho ::= \zeta | \langle r, \sigma, \eta, \rho' \rangle$, where r is a resumption, $\zeta \in \mathbf{Val}^\infty, \eta \in \mathbf{V}^\infty, \sigma \in \mathbf{State}$ and ρ' is a configuration. Let **Config** be the set of all configurations. A configuration of nesting number $n > 0$ has the form $\langle r_n, \sigma_n, \eta_n, \langle r_{n-1}, \sigma_{n-1}, \eta_{n-1}, \langle \ldots \langle r_1, \sigma_1, \eta_1, \zeta \rangle \ldots \rangle$. A stream ζ has nesting number 0.

2.3 A nondeterministic transition system NT for L

A transition system is a relation $\mapsto \subseteq \mathbf{Config} \times \mathbf{Config}$. If $(\rho, \rho') \in \mapsto$, then it is called a *transition* from ρ to ρ', denoted by $\rho \longrightarrow \rho'$. A configuration ρ is called a *terminal* if there is no ρ' such that $(\rho, \rho') \in \mapsto$. This is encoded by $\rho \longrightarrow \otimes$. For convenience we call it an *empty transition.* A transition can be accompanied by a value $\alpha \in \mathbf{V}$, in which case we write $\rho \stackrel{\alpha}{\longrightarrow} \rho'$. We say α is the output of the transition. We define this relation by induction on the nesting number of configurations. The following transition system is based on the principle that every subprocess is a candidate to take the next step. Thus different executions are possible.

1. $\epsilon \longrightarrow \otimes$
   If $\rho \longrightarrow \otimes$, then $\langle E, \sigma, \eta, \rho \rangle \longrightarrow \otimes$
   If $\rho \longrightarrow \otimes$, then $\langle read(v) : r, \sigma, \epsilon, \rho \rangle \longrightarrow \otimes$
2. $\beta\zeta \stackrel{\beta}{\longrightarrow} \zeta$
3. If $\rho \stackrel{\alpha}{\longrightarrow} \rho'$, then $\langle r, \sigma, \eta, \rho \rangle \longrightarrow \langle r, \sigma, \eta\alpha, \rho' \rangle$
   If $\rho \longrightarrow \rho'$, then $\langle r, \sigma, \eta, \rho \rangle \longrightarrow \langle r, \sigma, \eta, \rho' \rangle$
4. $\langle (v := e) : r, \sigma, \eta, \rho \rangle \stackrel{\tau}{\longrightarrow} \langle r, \sigma\{\beta/v\}, \eta, \rho \rangle$, where $\beta = V(e)(\sigma)$
5. $\langle skip : r, \sigma, \eta, \rho \rangle \stackrel{\tau}{\longrightarrow} \langle r, \sigma, \eta, \rho \rangle$
6. $\langle write(e) : r, \sigma, \eta, \rho \rangle \stackrel{\beta}{\longrightarrow} \langle r, \sigma, \eta, \rho \rangle$, where $\beta = V(e)(\sigma)$
7. For $\eta = \beta\eta', \beta \neq \tau$, $\langle read(v) : r, \sigma, \eta, \rho \rangle \stackrel{\tau}{\longrightarrow} \langle r, \sigma\{\beta/v\}, \eta', \rho \rangle$
   For $\eta = \tau\eta'$, $\langle read(v) : r, \sigma, \eta, \rho \rangle \stackrel{\tau}{\longrightarrow} \langle read(v) : r, \sigma, \eta', \rho \rangle$
8. $\langle fork(v) : r, \sigma, \eta, \rho \rangle \stackrel{\tau}{\longrightarrow} \langle r, \sigma\{1/v\}, \epsilon, \langle r, \sigma\{0/v\}, \eta, \rho \rangle\rangle$
9. $\langle (s_1;\ s_2) : r, \sigma, \eta, \rho \rangle \stackrel{\tau}{\longrightarrow} \langle s_1 : (s_2 : r), \sigma, \eta, \rho \rangle$

10. If $B(b)(\sigma)$, then $\langle \mathbf{if}\ b\ \mathbf{then}\ \ s_1\ \mathbf{else}\ s_2\ \mathbf{fi} : r, \sigma, \eta, \rho\rangle \xrightarrow{\tau} \langle s_1 : r, \sigma, \eta, \rho\rangle$
If $\neg B(b)(\sigma)$, then $\langle \mathbf{if}\ b\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \mathbf{fi} : r, \sigma, \eta, \rho\rangle \xrightarrow{\tau} \langle s_2 : r, \sigma, \eta, \rho\rangle$
11. $\langle \mathbf{while}\ b\ \mathbf{do}\ s\ \mathbf{od} : r, \sigma, \eta, \rho\rangle \xrightarrow{\tau}$
$\langle \mathbf{if}\ b\ \mathbf{then}\ s;\ \mathbf{while}\ b\ \mathbf{do}\ s\ \mathbf{od}\ \mathbf{else}\ \mathit{skip}\ \mathbf{fi} : r, \sigma, \eta, \rho\rangle$

Example 1. The following is a possible transition sequence in **NT**.
$\rho = \rho_0 = \langle \mathit{read}(y) : (\mathit{fork}(v)) : E), \{\}, \alpha\beta, \gamma\rangle \longrightarrow$
$\rho_1 = \langle \mathit{read}(y) : (\mathit{fork}(v) : E), \{\}, \alpha\beta\gamma, \epsilon\rangle \xrightarrow{\tau}$
$\rho_2 = \langle \mathit{fork}(v) : E, \{y = \alpha\}, \beta\gamma, \epsilon\rangle \xrightarrow{\tau}$
$\rho_3 = \langle E, \{y = \alpha, v = 1\}, \epsilon, \langle E, \{y = \alpha, v = 0\}, \beta\gamma, \epsilon\rangle\rangle \ (\longrightarrow \otimes)$

2.4 Enabledness and computations

Let $\rho = \langle r_m, \sigma_m, \eta_m, \ldots, \langle r_i, \sigma_i, \eta_i, \langle \ldots \langle r_1, \sigma_1, \eta_1, \zeta\rangle \ldots\rangle$. The *input stream* ζ is defined as the *0-th subconfiguration.* For $1 \le i \le m$, the *i-th subconfiguration* is $\langle r_i, \sigma_i, \eta_i, \langle \ldots \langle r_1, \sigma_1, \eta_1, \zeta\rangle \ldots\rangle$. Informally we say that the (r_i, σ_i) pair is the *i-th subprocess of* ρ. Here r_i, σ_i and η_i are called the *resumption, state* and *buffer of* the i-th subprocess, respectively. The transition $\langle \ldots \eta_{i+1}, \langle r_i, \sigma_i, \eta_i, \langle \ldots\rangle \ldots\rangle \longrightarrow \langle \ldots \eta'_{i+1}, \langle r'_i, \sigma'_i, \eta'_i, \langle \ldots\rangle \ldots\rangle$ is said to be *caused by* the i-th subprocess. The i-th subprocess can cause a transition only if it is enabled. We define that the 0-th subprocess is *enabled* if the input stream is not empty. The i-th subprocess is *enabled* if one of the rules from 4 to 11 can be applied to $\langle r_i, \sigma_i, \eta_i, \epsilon\rangle$. If a subprocess is not enabled, then it is called *disabled.* For an initial ρ, a maximal transition sequence is called a *computation* of ρ (it ends either with $\otimes$ or it is infinite):
$c(\rho) : \rho = \rho_0 \longrightarrow \rho_1 \longrightarrow \ldots \rho_{n-1} \longrightarrow \rho_n \longrightarrow \ldots$
The transition from ρ_{n-1} to ρ_n is called the n-th *step* and ρ_n is called the n-th *stage* of the computation. The *output stream* of $c(\rho)$ is the sequence of values produced by $c(\rho)$. In Example 1 above, we have a computation with τ^2 as output.

2.5 Operational semantics

To define operational semantics we should first know something about a *canonical computation* $C_0(\rho)$. The computation $C_0(\rho)$ begins with the transition caused by the 0-th subprocess if it is enabled. In the next steps, the transitions are successively caused by the subprocesses from right to left (if they are enabled). After the transition caused by the leftmost subprocess we start anew with the 0-th subprocess. This procedure is repeated as often as possible. Example 1 described such a computation. The canonical computation $C_0(\rho)$ corresponds with the computation defined by a very fair scheduler, which allows each subprocess, in a right-to-left fashion, to take one step.

For a given input stream ζ, a statement s and a state σ, we can consider different computations starting from $\rho = \langle s : E, \sigma, \epsilon, \zeta\rangle$. In [BN96,BN97] while proving the first part of the Kahn principle, we showed that $C_0(\rho)$ delivers the maximal output stream, i.e., the output stream of any other computation of ρ

is a prefix of the output stream of $C_0(\rho)$. This maximal output stream can be defined as the operational semantics of **NT**.

Definition 2. The operational semantics based on **NT** is defined as the function O_n:$\mathbf{L}\rightarrow$ **Proc** such that for any $s \in \mathbf{L}, \sigma \in \mathbf{State}$ and $\zeta \in \mathbf{Val}^\infty, O_n(s)(\sigma)(\zeta) =$ the output stream of $C_0(\langle s : E, \sigma, \epsilon, \zeta\rangle)$.

3 A deterministic Synchronous Transition System DT and its Relation to NT

We know from the last section that any computation in **NT** can be used to define the operational semantics O_n provided it delivers the maximal output stream. In this section, we will introduce a computation $C(\rho)$ for this purpose. We will first define a deterministic transition system **DT**. In general, a transition in **DT** lets all subprocesses take a step simultaneously. The operational semantics O_d is determined by the output streams of the maximal transition sequences in **DT**. By comparing the maximal transition sequence starting from ρ in **DT** with $C(\rho)$ in **NT** we can establish the equivalence between O_d and O_n.

3.1 A deterministic synchronous transition systems DT on L

In **DT** we use $\otimes$ and $\Longrightarrow$ the same way as we use $\otimes$ and $\longrightarrow$ in **NT**. Every transition in **DT** from a nonterminal delivers a value to the leftmost output channel. The transition from $\langle r, \sigma, \eta, \rho\rangle$ is defined by the transition from ρ. In most of the cases below, the right hand side of a transition will feature a buffer stream $\underline{\eta}$ and a configuration $\underline{\rho}$. These $\underline{\eta}, \underline{\rho}$ depend on the transition from ρ. We define $\underline{\rho} = \rho'$, $\underline{\eta} = \eta\alpha$ if $\rho \overset{\alpha}{\Longrightarrow} \rho'$ and $\underline{\rho} = \rho$, $\underline{\eta} = \eta$ if $\rho \Longrightarrow \otimes$.

1. $\epsilon \Longrightarrow \otimes$
   For all $\rho, \langle E, \sigma, \eta, \rho\rangle \Longrightarrow \otimes$
   $\langle read(v) : r, \sigma, \epsilon, \rho\rangle \Longrightarrow \otimes$ if $\rho \Longrightarrow \otimes$
2. $\beta\zeta \overset{\beta}{\Longrightarrow} \zeta$
3. $\langle (v := e) : r, \sigma, \eta, \rho\rangle \overset{\tau}{\Longrightarrow} \langle r, \sigma\{\beta/v\}, \underline{\eta}, \underline{\rho}\rangle$, where $\beta = V(e)(\sigma)$.
4. $\langle skip : r, \sigma, \eta, \rho\rangle \overset{\tau}{\Longrightarrow} \langle r, \sigma, \underline{\eta}, \underline{\rho}\rangle$
5. $\langle write(e) : r, \sigma, \eta, \rho\rangle \overset{\beta}{\Longrightarrow} \langle r, \sigma, \underline{\eta}, \underline{\rho}\rangle$, where $\beta = V(e)(\sigma)$
6. $\langle read(v) : r, \sigma, \eta, \rho\rangle \overset{\tau}{\Longrightarrow} \langle r, \sigma\{\beta/v\}, rest(\underline{\eta}), \underline{\rho}\rangle$, for $\underline{\eta} \neq \epsilon$ and $\beta = first(\underline{\eta}) \neq \tau$
   $\langle read(v) : r, \sigma, \eta, \rho\rangle \overset{\tau}{\Longrightarrow} \langle read(v) : r, \sigma, rest(\underline{\eta}), \underline{\rho}\rangle$, for $\underline{\eta} \neq \epsilon$ and $\tau = first(\underline{\eta})$
7. $\langle fork(v) : r, \sigma, \eta, \rho\rangle \overset{\tau}{\Longrightarrow} \langle r, \sigma\{1/v\}, \epsilon, \langle r, \sigma\{0/v\}, \underline{\eta}, \underline{\rho}\rangle\rangle$
8. $\langle (s_1;\ s_2) : r, \sigma, \eta, \rho\rangle \overset{\tau}{\Longrightarrow} \langle s_1 : (s_2 : r), \sigma, \underline{\eta}, \underline{\rho}\rangle$
9. If $B(b)(\sigma)$, then $\langle \mathbf{if}\ b\ \mathbf{then}\ \ s_1\ \mathbf{else}\ s_2\ \mathbf{fi} : r, \sigma, \eta, \rho\rangle \overset{\tau}{\Longrightarrow} \langle s_1 : r, \sigma, \underline{\eta}, \underline{\rho}\rangle$
   If $\neg B(b)(\sigma)$, then $\langle \mathbf{if}\ b\ \mathbf{then}\ \ s_1\ \mathbf{else}\ s_2\ \mathbf{fi} : r, \sigma, \eta, \rho\rangle \overset{\tau}{\Longrightarrow} \langle s_2 : r, \sigma, \underline{\eta}, \underline{\rho}\rangle$
10. $\langle \mathbf{while}\ b\ \mathbf{do}\ \ s\ \ \mathbf{od} : r, \sigma, \eta, \rho\rangle \overset{\tau}{\Longrightarrow}$
    $\langle \mathbf{if}\ b\ \mathbf{then}\ s;\ \mathbf{while}\ b\ \mathbf{do}\ s\ \mathbf{od}\ \mathbf{else}\ skip\ \mathbf{fi} : r, \sigma, \underline{\eta}, \underline{\rho}\rangle$

Example 2. A transition sequence in **DT** from the same ρ as in Example 1:
$\rho = \langle read(y) : (fork(v) : E), \{\}, \alpha\beta, \gamma\rangle$
$\overset{\tau}{\Longrightarrow} \langle fork(v) : E, \{y = \alpha\}, \beta\gamma, \epsilon\rangle$
{the first two steps in example 1}
$\overset{\tau}{\Longrightarrow} \langle E, \{y = \alpha, v = 1\}, \epsilon, \langle E, \{y = \alpha, v = 0\}, \beta\gamma, \epsilon\rangle\rangle$
{the last step in example 1} ($\Longrightarrow \otimes$)

Example 3. The following are two transition sequences in **DT**.

(*a*) $\langle read(x) : E, \sigma, \epsilon, \alpha\beta\rangle \overset{\tau}{\Longrightarrow} \langle E, \sigma\{\alpha/x\}, \epsilon, \beta\rangle \Longrightarrow \otimes$.

(*b*) $\langle read(w) : E, \sigma_1, \epsilon, \langle read(w) : E, \sigma_0, \alpha, \epsilon\rangle\rangle$.
$\overset{\tau}{\Longrightarrow} \langle read(w) : E, \sigma_1, \epsilon, \langle E, \sigma_0\{\alpha/w\}, \epsilon, \epsilon\rangle\rangle \Longrightarrow \otimes$

Example 4. Notice that a terminal in **NT** is also a terminal in **DT**. The terminals in **DT** are not always terminals in **NT**. E.g. $\langle E, \sigma, \eta, \langle (v := e) : E, \sigma', \epsilon, \alpha\rangle\rangle$ and $\langle read(x) : E, \sigma, \epsilon, \langle read(y), \sigma, \epsilon, \langle E, \sigma', \epsilon, \alpha\rangle\rangle\rangle$ are terminals in **DT** but not in **NT**.

3.2 The operational semantics derived from DT

We define the operational semantics O_d, based on **DT**, as the unique fixed point o of a contraction Φ on the complete ultrametric space $\mathbf{SemO} = \mathbf{Config} \to \mathbf{V}^\infty$. This is rather standard, for more background, definitions and proofs, cf. [BV96], in particular section 9.2.

Definition 3. Let $\mathbf{SemO} = \mathbf{Config} \to \mathbf{V}^\infty$. Let $\Phi : \mathbf{SemO} \to \mathbf{SemO}$ be defined by

$\Phi(f)(\rho) = \epsilon$ if $\rho \Longrightarrow \otimes$ in **DT**.

$\Phi(f)(\rho) = \beta f(\rho')$ if $\rho \overset{\beta}{\Longrightarrow} \rho'$ in **DT**.

Theorem 4. $\Phi : \mathbf{SemO} \to \mathbf{SemO}$ is a 1/2-contraction and hence Φ has a unique fixed point o.

By Definition 3, $o(\rho) = \epsilon$ if $\rho \Longrightarrow \otimes$ and $o(\rho) = \beta o(\rho')$ if $\rho \overset{\beta}{\Longrightarrow} \rho'$. Thus $o(\rho)$ is the sequence of values produced by the unique maximal transition sequence starting from ρ. For instance, in Example 2, we have $o(\rho) = \tau^2$. The operational semantics $O_d : \mathbf{L} \to \mathbf{Proc}$ is defined by $O_d(s) = \lambda\sigma.\lambda\zeta.o(\langle s : E, \sigma, \epsilon, \zeta\rangle)$.

3.3 The relations between transitions in NT and DT

The operational semantics in **NT** is defined by the canonical computation $C_0(\rho)$ or other computations which delivers the same maximal output stream. On the other hand, the operational semantics in **DT** is based on o which delivers the output stream of the maximal transition sequence of synchronous steps from a configuration. In this section we want to show that a step in **DT** corresponds with a finite sequence of steps in **NT**. Such a sequence is called a *big step*. A combination of such big steps will be a computation $C(\rho)$ in **NT**. $C(\rho)$ delivers also the maximal output stream.

An example of $C(\rho)$ as combination of big steps. Given a configuration ρ, we can define a computation $C_j(\rho)$ which leaves all k-th ($k < j$) subconfigurations of ρ unchanged. $C_j(\rho)$ is a computation defined by the following procedure. It begins with the transition caused by the j-th subprocess if it is enabled. In the next steps, the subprocesses from right to left cause successively transitions and begin again with the j-th subprocess. This procedure is repeated as often as possible. The canonical computation $C_0(\rho)$ is a special case for $j = 0$. A *big step* of $C_j(\rho)$ is a sequence of transition steps in $C_j(\rho)$ where the transitions start from the j-th subprocess and end at the j-th subprocess again. For more precise definitions of $C_j(\rho)$ and big steps, see the appendix. We first give an example to show how $C(\rho)$ is defined.

Example 5. Suppose $\underline{\rho} = \langle write(1) : (write(2) : E), \sigma, \epsilon, \langle write(3) : E, \sigma', \epsilon, 4.5\rangle\rangle$. We first use the following big step induced by $C_0(\underline{\rho})$ to define the first steps in $C(\underline{\rho})$ in **NT**:
$\underline{\rho} \longrightarrow \langle write(1) : (write(2) : E), \sigma, \epsilon, \langle write(3) : E,\ \sigma', 4, 5\rangle\rangle$
$\longrightarrow \langle write(1) : (write(2) : E),\ \sigma, 3,\ \langle E, \sigma', 4, 5\rangle\rangle$
$\xrightarrow{1} \langle write(2) : E, \sigma, 3, \langle E, \sigma', 4, 5\rangle\rangle = \underline{\rho}_1$.
Notice that $\langle E, \sigma', 4, 5\rangle$ is a terminal in **DT** but not in **NT**. This part will stay the same in a transition sequence from ρ_1 in **DT**. Hence we use a big step in $C_2(\underline{\rho}_1)$ for defining $C(\underline{\rho})$ in **NT** further:

$\underline{\rho}_1 \xrightarrow{2} \langle E, \sigma, 3, \langle E, \sigma', 4, 5\rangle\rangle = \underline{\rho}_2 \longrightarrow \otimes$

The combinations of transitions in **NT** from $\underline{\rho}$ to $\underline{\rho}_1$ to $\underline{\rho}_2$ to $\otimes$ is defined as $C(\underline{\rho})$. We denote the big steps by $\underline{\rho} \xrightarrow{*1} \underline{\rho}_1 \xrightarrow{*2} \underline{\rho}_2 \longrightarrow \otimes$. Furthermore, $\underline{\rho} \xRightarrow{1} \underline{\rho}_1 \xRightarrow{2} \underline{\rho}_2 \Longrightarrow \otimes$ is the unique maximal transition sequence from ρ in **DT**.

We want to build a relation between a *big step* in **NT** and a synchronous transition step in **DT**. For a given ρ, usually a big step of $C_0(\rho)$ in **NT** corresponds with a synchronous transition step in **DT**. However, if a subconfiguration of ρ is $\langle E, \sigma, \eta, \rho'\rangle$, then this subconfiguration is a terminal in **DT** but may not be a terminal in **NT**. Fortunately, the transitions of ρ' in **NT** will not influence the global output stream because of the obstruction of E. If two configurations differ only in subconfigurations which are terminals in **DT**, they would produce the same maximal output streams in **NT**. To understand this, we first give a characterization of terminals in **DT** (see also Example 4). Then we will define $C(\rho)$ more formally as a sequence of different types of big steps. The lemma given afterwards will imply that $C(\rho)$ delivers the same maximal stream as $C_0(\rho)$.

Terminals in the transition system DT. The transition system **DT** is complete in the sense that $\rho \xRightarrow{\alpha} \rho'$ or $\rho \Longrightarrow \otimes$ for all $\rho \in$ **Config**. Let $T_0 = \{\epsilon\} \cup \{\rho |\, \rho = \langle E, \sigma, \eta, \rho'\rangle\}$ and $T_i = \{\rho |\, \rho = \langle read(v) : r, \sigma, \epsilon, \rho'\rangle, \rho' \in T_{i-1}\}$. Then $T = \bigcup T_i$ is precisely the set of all terminals in this transition system **DT**. The terminals in **DT** which are not terminals in **NT**, are either of the form

type 1 : $\langle E, \sigma_m, \eta_m, \ldots, \langle r_i, \sigma_i, \eta_i, \langle \ldots, \langle r_1, \sigma_1, \eta_1, \zeta \rangle \ldots \rangle$,
where some i-th subprocess is enabled in **NT**, or of the form
type 2 : $\langle r_m, \sigma_m, \epsilon, \langle r_{m-1}, \sigma_{m-1}, \epsilon, \langle \ldots, \langle r_1, \sigma_1, \epsilon, \rho' \rangle \ldots \rangle$,
where all $r_m, \ldots, r_1$ begin with a *read*-statement and ρ' is of type 1.

Lemma 5. Let $\underline{\rho} = \langle r_m, \sigma_m, \eta_m, \ldots, \langle r_1, \sigma_1, \eta_1, \rho \rangle \rangle$ be a non-terminal in **NT**. Suppose ρ has nesting numer k and is the leftmost **DT**-terminal of type 1 or 2 in $\underline{\rho}$. If $k < m$, then a big step $\underline{\rho} \xrightarrow{*\alpha} \underline{\rho}_1$ of $C_{k+1}(\rho)$ in **NT** has the same effect as a single step $\underline{\rho} \overset{\alpha}{\Longrightarrow} \underline{\rho}_1$ in **DT**. If $k = m$, then $\underline{\rho} \xrightarrow{*} \underline{\rho}_1$ has empty global output in **NT** and thus it corresponds with the empty transition $\underline{\rho} \Longrightarrow \otimes$ in **DT**.

Remark. If $\underline{\rho}$ does not contain such a **DT**-terminal, then let $k = -1$. In this situation, $\underline{\rho} \xrightarrow{*\alpha} \underline{\rho}_1$ means a big step of $C_0(\rho)$.

Definition 6. Given $\underline{\rho}$, we define the computation $C(\underline{\rho})$ in **NT** as follows:

$$C(\underline{\rho}) : \underline{\rho} = \underline{\rho}_0 \xrightarrow{*\alpha_0} \underline{\rho}_1 \xrightarrow{*\alpha_1} \ldots \xrightarrow{*\alpha_{i-1}} \underline{\rho}_i \xrightarrow{*\alpha_i} \underline{\rho}_{i+1} \cdots,$$

If $\underline{\rho}_i$ is not a terminal in **NT** and the k-th subconfiguration is the leftmost **DT**-terminal of type 1 or 2 in $\underline{\rho}_i$, then $\underline{\rho}_i \xrightarrow{*\alpha_i} \underline{\rho}_{i+1}$ is the sequence in a big step in **NT** induced by $C_{k+1}(\underline{\rho}_i)$.

Proposition 7. Given $\underline{\rho}$, the unique maximal transition sequence in **DT** from $\underline{\rho}$ delivers the same global output as $C(\underline{\rho})$ in **NT**.

Now we want to compare $C(\underline{\rho})$ and $C_0(\underline{\rho})$. If we know that they also deliver the same output stream, then we have proved that $O_d = O_n$ by Proposition 7.

Lemma 8. Let $\underline{\rho} = \langle r_m, \sigma_m, \eta_m, \ldots, \langle r_1, \sigma_1, \eta_1, \rho \rangle \rangle$, where ρ is a terminal in **DT** with nesting number $k \geq -1$ and $\underline{\rho}' = \langle r_m, \sigma_m, \eta_m, \ldots, \langle r_1, \sigma_1, \eta_1, \rho' \rangle \rangle$, where ρ' is a terminal in **DT** with nesting number $i \geq -1$. If for some $j \leq k+1$, the stage after a big step of $C_j(\underline{\rho})$ from ρ is $\underline{\rho}_1 = \langle r'_n, \sigma'_n, \eta'_n, \ldots, \langle r'_1, \sigma'_1, \eta'_1, \rho_1 \rangle \rangle$, then for every $s \leq i+1$, the stage after a big step of $C_s(\underline{\rho}')$ from $\underline{\rho}'$ has the form $\underline{\rho}'_1 = \langle r'_n, \sigma'_n, \eta'_n, \ldots, \langle r'_1, \sigma'_1, \eta'_1, \rho'_1 \rangle \rangle$. Moreover, both big steps deliver the same global output (if any) and both ρ_1 and ρ'_1 are terminals in **DT**.

Starting from a configuration, we want to compare the transition sequence defined by C and C_0. We apply Lemma 8 in the following way. Suppose using computation C we come to a stage $\underline{\rho}$ as in Lemma 8. Suppose also the leftmost **DT**-terminal in $\underline{\rho}$ has nesting number k. Then the computation C uses $C_{k+1}(\underline{\rho})$ to induce the next big step. The new stage is $\underline{\rho}_1$. Now suppose also using C_0 we come to a stage $\underline{\rho}'$ as in Lemma 8. We use C_0 again to induce the next big step from $\underline{\rho}'$ to some $\underline{\rho}'_1$. The value of i is irrelevant, because $0 \leq i+1$ for all i. The lemma guarantees that we can compare $\underline{\rho}_1$ and $\underline{\rho}'_1$ in the same way as $\underline{\rho}$ and $\underline{\rho}'$. Therefore:

Lemma 9. The global output of $C_0(\underline{\rho})$ and $C(\underline{\rho})$ are the same.

Theorem 10. $O_d = O_n$.

4 Denotational Semantics and Equivalence of Semantics

In this section we will define a denotational semantics by using the unique fixed point m of some contraction Ψ. Let $\mathbf{V}^\infty \to_{\mathbf{n}} \mathbf{V}^\infty$ be the set of all nonexpansive functions from $\mathbf{V}^\infty$ to $\mathbf{V}^\infty$. We further define the set of all *continuations*, denoted by **Cont**, as the set of all functions $\theta : \mathbf{State} \to (\mathbf{V}^\infty \to_{\mathbf{n}} \mathbf{V}^\infty)$. The set of all nonexpansive functions from **Cont** to **Cont**, is denoted by $\mathbf{Cont} \to_{\mathbf{n}} \mathbf{Cont}$. The set **SemD** denotes the set of all functions $\psi : \mathbf{L} \to (\mathbf{Cont} \to_{\mathbf{n}} \mathbf{Cont})$. Also in this section the approach is standard, we again refer to [BV96].

4.1 Using a contraction Ψ to define the denotational semantics

Definition 11. Let Ψ be defined as follows.

1. $\Psi(\psi)(v := e)(\theta)(\sigma)(\zeta) = \tau\theta(\sigma\{\beta/v\})(\zeta)$, where $\beta = V(e)(\sigma)$
2. $\Psi(\psi)(skip)(\theta)(\sigma)(\zeta) = \tau\theta(\sigma)(\zeta)$
3. $\Psi(\psi)(write(e))(\theta)(\sigma)(\zeta) = \beta\theta(\sigma)(\zeta)$, where $\beta = V(e)(\sigma)$
4. $\Psi(\psi)(read(v))(\theta)(\sigma)(\epsilon) = \epsilon$
   If $\zeta \neq \epsilon$, $first(\zeta) = \tau$, then $\Psi(\psi)(read(v))(\theta)(\sigma)(\zeta) = \tau\psi(read(v))(\theta)(\sigma)(rest(\zeta))$
   If $\zeta \neq \epsilon$, $first(\zeta) = \alpha \neq \tau$, then $\Psi(\psi)(read(v))(\theta)(\sigma)(\zeta) = \tau\theta(\sigma\{\alpha/v\})(rest(\zeta))$
5. $\Psi(\psi)(fork(v))(\theta)(\sigma)(\zeta) = \tau\theta(\sigma\{1/v\})(\theta(\sigma\{0/v\})(\zeta))$
6. $\Psi(\psi)(s_1;\ s_2)(\theta)(\sigma)(\zeta) = \tau\Psi(\psi)(s_1)(\Psi(\psi)(s_2)(\theta))(\sigma)(\zeta)$
7. If $B(b)(\sigma)$, then $\Psi(\psi)(\mathbf{if}\ b\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \mathbf{fi})(\theta)(\sigma)(\zeta) = \tau\Psi(\psi)(s_1)(\theta)(\sigma)(\zeta)$
   If $\neg B(b)(\sigma)$, then $\Psi(\psi)(\mathbf{if}\ b\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \mathbf{fi})(\theta)(\sigma)(\zeta) = \tau\Psi(\psi)(s_2)(\theta)(\sigma)(\zeta)$
8. $\Psi(\psi)(\mathbf{while}\ b\ \mathbf{do}\ s\ \mathbf{od})(\theta)(\sigma)(\zeta)$
   $= \tau\psi(\mathbf{if}\ b\ \mathbf{then}\ s;\ \mathbf{while}\ b\ \mathbf{do}\ s\ \mathbf{od}\ \mathbf{else}\ skip\ \mathbf{fi})(\theta)(\sigma)(\zeta)$

Theorem 12. $\Psi : \mathbf{SemD} \to \mathbf{SemD}$ is a 1/2-contraction.

Definition 13. Let $\theta_0 \in \mathbf{State} \to (\mathbf{V}^\infty \to \mathbf{V}^\infty)$ be the trivial continuation defined by $\theta_0(\sigma)(\zeta) = \epsilon$. Let m be the unique fixed point of Ψ, i.e. $\Psi(m) = m$. The *denotational semantics* **D** of **L** is defined as $D(s)(\sigma)(\zeta) = m(s)(\theta_0)(\sigma)(\zeta)$ for all $\sigma \in \mathbf{State}$ and $\zeta \in \mathbf{V}^\infty$.

4.2 Equivalence of O_n, O_d and D

We will define a mapping I using the fixed point m of Ψ. We can prove that I is also a fixed point of Φ. From the uniqueness of the fixed point of Φ, we can conclude $I = o$ and then $D = O_d$. The proofs of Lemma 15 and 16 are by induction on the nesting number of $\underline{\rho}$. Lemma 17 follows immediately from the previous ones.

Definition 14. The mapping $H : \mathbf{Res} \to \mathbf{Cont}$ is defined by $H(E) = \theta_0$ and $H(s : r) = m(s)H(r)$. The mapping $I : \mathbf{Config} \to \mathbf{V}^\infty$ is defined by $I(\zeta) = \zeta$ and $I(\langle r, \sigma, \eta, \rho\rangle) = H(r)(\sigma)(\eta I(\rho))$.

Example 6. Consider Example 3(a). We have:
$o(\langle read(x) : E, \sigma, \epsilon, \alpha\beta\rangle) = \tau$,
$I(\langle read(x) : E, \sigma, \epsilon, \alpha\beta\rangle) = H(read(x) : E)(\sigma)(\epsilon.\alpha\beta) = m(read(x))(H(E))(\sigma)(\alpha\beta)$
$= \tau H(E)(\sigma\{\alpha/x\})(\beta) = \tau\theta_0(\sigma\{\alpha/x\})(\beta) = \tau$. Thus $I(\rho) = o(\rho)$.

Lemma 15. If $\underline{\rho} \Longrightarrow \otimes$ in **DT**, then $I(\underline{\rho}) = \epsilon$.

Lemma 16. For all $\underline{\rho} \overset{\beta}{\Longrightarrow} \underline{\rho}'$ in **DT**, we have $I(\underline{\rho}) = \beta I(\underline{\rho}')$.

Lemma 17. $\Phi(I) = I$.

Since Φ has only one fixed point, we have $I = o$. We can then prove the equivalence of all semantics defined in this article. This establishes the second part of the Kahn principle:

Theorem 18. $D(s) = O_d(s) = O_n(s)$ for all $s \in \mathbf{L}$.

5 Summary

As far as we aware, Kahn's principle has never been proved for dynamic networks. In this paper and the related ones [BN96,BN97], we consider a simple language **L** containing the fork-statement. In [BN96, BN97] we have defined a nondeterministic interleaving transition system without τ's. In these papers we also proved the first part of the Kahn principle, namely that every computation delivers the maximal output stream or a prefix of it. In this paper, we first defined a nondeterministic interleaving system **NT** with τ's. The proof in [BN96, BN97] can easily be adapted to prove the first part of Kahn's principle with respect to **NT**. Thus we can assume that all computations in **NT** either give a maximal stream or a prefix of it. This can be used to define the operational semantics O_n. We have concentrated on the second part of Kahn's principle in this paper. Our proof contains two parts.

1. $O_n = O_d$: We need first to prove that a synchronous step in **DT** is a type of big step in **NT**. Then we observe that a transition sequence from $\underline{\rho}$ in **DT** corresponds with a computation $C(\underline{\rho})$ in **NT** which is a composition of big steps. We also proved that $C(\underline{\rho})$ delivers the maximum output in **NT**, just as $C_0(\underline{\rho})$. That means that we can also use $C(\underline{\rho})$ to establish O_n. Thus we establish $O_d = O_n$.
2. $O_d = D$: The proofs (omitted) in section 4 established this result. These type of proofs are standard.

In short, our contribution for the second part of the Kahn principle for linear dynamic networks is that we do not prove $O_n = D$ directly. Instead we define **DT** and use it as a bridge to prove $O_n = D$. The new ideas for proving this are essentially summarized in item 1 above.

Several extensions of the results about part 1 ([BN96, BN97]) and part 2 come to mind. A natural idea is to study the general case introduced in [K74], i.e. to allow processes to expand into arbitrary networks, and to allow feedback loops, (sequences of) channels starting from and arriving at the same node. Apart from notational inconveniences, we do not see many problems. However, we can not go ahead to prove a more complicated situation if we do not know how the simpler situations works. This paper and [BN96, BN97] gives enough fundament for further research.

References

[AG78] Arvind and K.P. Gostelow, Some relationships between asynchronous interpreters of a data flow language, in: Formal description of programming concepts, W.J. Neuhold (ed.), pp. 95-119, North-Holland, 1978.

[A81] A. Arnold, Sémantique des processus communicants, RAIRO Theor.Inf. 15,2, pp.103-139, 1981.

[B86] A. de Bruin, Experiments with continuation semantics: jumps, backtracking, dynamic networks, Ph.D. Thesis, Free University Amsterdam, 1986.

[B88] M. Broy, Nondeterministic data flow programs: how to avoid the merge anomaly, Science of Computer Programming, 10, pp.65-85, 1988.

[BA81] J. Brock and W. Ackerman, Scenarios: a model of non-determinate computation, in: Formalization of programming concepts, LNCS 107, pp.252-259, Springer, 1981.

[BB85] A. P. W. Böhm and A. de Bruin, The denotational semantics of dynamic networks of processes, ACM Trans. Prog. Lang. Syst, 7, pp.656-679,1985.

[BBB93] J. W. de Bakker, F. van Breugel and A. de Bruin, Comparative semantics for linear arrays of communicating processes, a study of the UNIX fork and pipe commands, in: A.M. Borzyszkowski and S. Sokolowski (eds.) Proc. Math. Foundations of Computer Science, LNCS 711, pp.252-261, Springer, 1993.

[BV96] J. W. de Bakker, E. de Vink, *Control Flow Semantics*, MIT press, 1996.

[BN96] A. de Bruin, S. H. Nienhuys-Cheng, Linear dynamic Kahn networks are deterministic, in: W. Penczek, A. Szalas (Eds.), Proc. Math. Foundations of Computer Science, LNCS 1113, pp.242-254, Springer, 1996.

[BN97] A. de Bruin, S. H. Nienhuys-Cheng, Linear dynamic Kahn networks are deterministic, complete version of [BN96]. Accepted by *Theoretical Computer Science*, 1997, special issue for MFCS96. It is based on Technical Report cs-eur-94-06, Erasmus University Rotterdam. It can be obtained by anonymous ftp from ftp.cs.few.eur.nl from directory pub/doc/techreports/1994 as eur-cs-94-06.

[C72] J.M. Cadiou, Recursive definitions of partial functions and their computations, Ph.D. thesis, Stanford Univ., 1972.

[F82] A.A. Faustini, An operational semantics for pure data flow, in: Automata, languages and programming, 9th Coll., LNCS 140, M.Nielsen and E.M. Schmidt (eds.), pp.212-224, 1982.

[JK90] B. Jonsson and J.N. Kok, Towards a complete hierarchy of compositional dataflow networks, in: Proc. theoretical aspects of computer software, LNCS 526, pp.204-225, Springer, 1990.

[J85] B. Jonsson, A model and proof system for asynchronous networks, in: Proc. 4th ACM PoDC, pp.49-58, 1985.

[K74] G. Kahn, The semantics of a simple language for parallel programming, in: Proc. IFIP74, J.L. Rosenfeld (ed.), North-Holland, pp.471-475, 1974.

[K86] J. Kok, Denotational semantics of nets with nondeterminism, in: European Symp. Programming, Saarbrücken, LNCS 206, pp.237-249, Springer, 1986.
[K78] P. Kosinski, A straight-forward denotational semantics for nondeterminate data flow programs, in: Proc. 5th ACM PoPL, pp.214-219, 1978.
[KM77] G. Kahn and D.B. MacQueen, Coroutines and networks of parallel processes, in: Proc. IFIP77, B. Gilchrist (ed.), North-Holland, pp.993-998, 1977.
[LS89] N. Lynch and E. Stark, A proof of the Kahn principle for input/output automata, Information and Computation, 82, 1, pp.81-92, 1989.
[SN85] J. Staples and V. Nguyen, A fixpoint semantics for nondeterministic data flow, JACM, 32 (2), pp.411-444, 1985.

Appendix

Computation $C_j(\rho)$. $C_j(\rho)$ is a computation in **NT** defined by the following procedure. It begins with the transition caused by the j-th subprocess if it is enabled. In the next steps, the subprocesses from right to left cause successively transitions and begin again with the j-th subprocess. This procedure is repeated as often as possible. More formally,

1. Let $\rho_0 = \rho$. We say the j-th subprocess of ρ is ready to proceed.
2. Suppose the stage of computation $C_j(\rho)$ is now $\langle r_m, \sigma_m, \eta_m, \ldots, \langle r_i, \sigma_i, \eta_i, \langle \ldots \langle r_1, \sigma_1, \eta_1, \zeta \rangle \ldots \rangle$ and suppose the i-th subprocess of ρ_k is ready to proceed. If this subprocess is enabled, then perform the transition caused by it which yields the next stage ρ_{k+1}. If it is disabled, then do nothing. (In the next definition of big steps, we also use $\rho_{k+1}(= \rho_k)$ to denote this stage.) If $i = m$, then define that now the j-th subprocess of ρ_{k+1} is ready to proceed. If $i < m$ and ρ_i does not start with a fork, then we define that the $(i+1)$-th subprocess of ρ_{k+1} is ready to proceed. If $i < m$ and ρ_i starts with a fork-statement, then define that the $(i+2)$-th subprocess of ρ_{k+1} is ready to proceed.

Big step induced by $C_j(\rho)$. Big steps are used to define a computation $C(\rho)$. A big step induced by $C_j(\rho)$ is a sequence of transition steps in **NT** where the transitions start from the j-th subprocess and end at the j-th subprocess again. More formally, suppose $\rho_k = \langle r_m, \sigma_m, \eta_m, \ldots, \langle r_i, \sigma_i, \eta_i, \langle \ldots \langle r_1, \sigma_1, \eta_1, \zeta \rangle \ldots \rangle$, which is defined as in $C_j(\rho)$. Suppose also that the j-th subprocess of ρ_k is ready to proceed. Then after the steps given above: from ρ_k to $\rho_{k+1}, \ldots, \rho_{k+m+1-j}$, we will again reach the stage that the j-th subprocess is ready to proceed (this is some sort of cycle). The transition from ρ_k to $\rho_{k+m+1-j}$ is called a big step induced by $C_j(\rho)$.

This article was processed using the LaTeX macro package with LLNCS style