有限なロケーションを持つ並行プロセス

カサブランカ・ファビオ 坂部俊樹 稲垣康善

名古屋大学工学部情報工学科

〒464－01名古屋市千種区不老町

概要

並行性へのインターリビング・アップローチでは、計算の空間構造が考えれられていなかった。
その問題を扱うために最近提案されたモデルでは、動作を行なうことのできるサイトの数が無限であること、あるいは、プロセスが実行される前に、プロセスの空間構造が知られていることが仮定されている。
本稿では、有限な空間資源（ロケーション）の集合を管理するマネジャーと並行プロセスを拡張することを提案する。そのマネジャーはプロセスにロケーションを割り当て、プロセスから解放されたロケーションを集める。ρという特別な動作が明示的にロケーションの解放をモデリングする。
さらに、マネジャーの間に、ロケーションの相互提供が可能であること示唆する。そのために、ロケーションの遅延双方向転送の戦略を紹介する。

キーワード： プロセス代数、プロセス空間構造、ロケーション、動作意味論、ロケーション転送戦略

Processes with Bounded Locations

Fabio CASABLANCA, Toshiki SAKABE, Yasuyoshi INAGAKI

Nagoya University - Information Engineering Department

Furo-cho Chikusa-ku Nagoya-shi 464-01

Abstract

One of the aspects neglected by a interleaving approach to concurrency is the spatial distribution of the computation.
Approaches which have dealt so far with this problem assume the availability of an infinite number of sites where to perform actions or suppose that the spatial structure of a process is known before its execution.
In this paper we present a model where, coherently with the reality of concurrency, processes are extended with managers of finite sets of spatial resources (locations). The managers assign locations to agents, if any is available, and collect locations which are eventually released. Release of locations is modelled explicitly by a special action ρ (*release*). An operational semantics is proposed for this extension.
Also, we suggest that managers can ask for and receive locations from other managers. For this purpose we introduce and discuss a *lazy bidirectional* location transfer policy.

Keywords: Process Algebra, Process Space, Locations, Operational Semantics, Location Transfer Policy

1 Introduction

Process Algebra, as formalized, between the others, by R.Milner in CCS [Mi89] and by C.A.R.Hoare in CSP [Ho85], is a very influential model in the field of the formal studies on the structure and on the properties of concurrency.

A key property of Process Algebra, besides resorting to the algebraic formalism, is the assumption of the Interleaving Hypothesis, which assimilates concurrency to nondeterminism: actions performed concurrently by different automata are considered equivalent to nondeterministic choices between sequences of actions performed by a single automaton. Interleaving models lose information on the *causality relation* between the subprocesses of a process; also, they lose information on the *spatiality relation* between the automata which execute the subprocesses, in terms of amount of needed automata and their reciprocal connectivity relation.

Boudol et al. [BCHK92, BCHK91] have studied the possibility to extend CCS to a language which explicitly deals with spatial resources, and Kiehn, between the proponents of that model, has studied the difference between the spatiality relation and the causality relation [Ki91],.

In that approach, elementary spatial resources (*locations*), are added to CCS agents. Initially no location is assigned to processes; when it is needed to perform an action, let us say a, a location l is dynamically allocated; after that, the process p which follows a is said to reside at l.

In symbols, the representation of transitions is enriched as follows:

$$p \xrightarrow[l]{a} l :: p',$$

where $p \overset{def}{=} a.p'$, for processes p and p' , action a and location l.

The actions of p' will be performed at "sub-locations" of l; if $p' \overset{def}{=} b.p''$, we have:

$$l :: p' \xrightarrow[lm]{b} lm :: p''.$$

Through a very simple example it is easy to understand the difference between classical CCS processes and processes enriched with locations (from now on, we will use the abbreviation CCS-L).

In fact, in the classical approach we have:

$$a.b.\textbf{nil} + b.a.\textbf{nil} \approx a.\textbf{nil}|b.\textbf{nil}$$

but in the approach with locations we have:

$$a.b.\textbf{nil} + b.a.\textbf{nil} \not\approx a.\textbf{nil}|b.\textbf{nil}$$

because in the left-hand side process the two actions are executed in physically separated locations, while in the right-hand side process, in each of the two possible alternative branches the actions are executed on contiguous locations.

[BCHK91, BCHK92] propose an operational semantics, location bisimulation equivalence and its axiomatisation for CCS-L.

CCS-L describes the ideal spatiality of the process, independently from any constraint on the amount and configuration of the spatial resources held by the process. It is not possible to model situations as deadlock as a consequence of lack of locations; also, it is not possible to describe transfers of spatial resources through the process, and their influence on the performance.

In this work, we extend processes-with-locations to processes with a finite number of available locations, i.e., *processes with bounded locations*. Through this extension, it is possible to model a computational environment which is closer to reality, where spatial deadlock and the need for efficient management of spatial resources is often observed.

In our approach, all agents are associated to a finite, possibly empty, set of locations, which is managed by a *Spatial Resources Manager (SR-manager)*. A SR-manager allocates a location to a process which performs an action and at the same time decreases by one the size of the location set. When a process has only a finite number of locations, it becomes extremely important to be able to "reuse" locations. By means of a special action ρ (*release*) we allow the SR-manager to deallocate locations from a process and increase by one the size of the location set.

In this new computational environment, it is possible to observe a *spatial deadlock*, a behaviour which at our knowledge has not been described yet in the Process Algebra framework.

<u>*Example. 1*</u> Let us consider the transition $1@a.b \xrightarrow[l]{a} 0@l :: b$, where a, b are actions, l is a location. $n@$ denotes an SR-manager by means of the size

of its location set.
The intuitive meaning is that action a is been performed on location a. We need now another location for action b, but, as the SR-manager has no available locations, the computation suspends.
Notice that it is possible to modify the initial agent, using the release operator, such that the process can successfully terminate:

$$1@a.\rho.b \xrightarrow[l]{a} 0@l :: \rho.b \xrightarrow[\epsilon]{\tau} 1@l :: b \xrightarrow[m]{b} 0@m :: \mathbf{nil}$$

□

Another interesting aspect is the possibility to describe *the structure of the spatiality* of the process by means of the configuration of the SR-managers.

For example, the two processes:

$$P_1 \stackrel{def}{=} A@(B@p|C@q|D@r)$$
$$P_2 \stackrel{def}{=} P@(Q@p|(K@(M@q|N@r))$$

independently by the size of the SR-managers, differ in their configurations.

Generally, the same agent will be in the scope of more than one SR-manager: q is in the scope of $A@$ and $C@$ in process P_1 and in the scope of $K@, L@$ and $M@$ in P_2.

It is natural to think of the SR-manager "closer" to an agent as its "local" SR-manager; at first, the agent will use locations from this SR-manager and later locations from farer, more "global" SR-managers. This suggests the possibility of location transfers between SR-managers and the need to specify a *location transfer policy* to avoid misbehaviours, as potentially infinite useless exchanges of locations between SR-managers.

In the following, we detail the model here sketched, which we call BL-CCS (CCS with Bounded Locations). We will define an operational semantics, based on a "lazy bidirectional" location transfer protocol, which avoids meaningless transfers (Section 2). In Section 3 we give an example to illustrate the model's behaviour and in Section 4 we give indications for future work.

2 Syntax and Operational Semantics

CCS-L extends CCS with locations and operators like location prefixing and action-with-location prefixing. CCS operational semantics is modified to keep in account the effects of the existence of locations on the behaviour of a process.

In BL-CCS, processes with locations are augmented with SR-managers and the release special action. The operational semantics is extended not only with rules that deal with these new objects, but should also describe, according to the observations developed in the introduction, a location transfer policy.

In BL-CCS there are three syntactic objects: agents, locations and SR-managers.

We suppose that the locations $\texttt{Locs} = \{u, v, ...\}$ are built by concatenation from a set of elementary locations $\texttt{ElLocs} = \{l, m, ...\}$ plus an infinite set of location variables $\texttt{Lvar} = \{x, y, ...\}$. $\epsilon \in \texttt{ElLocs}$ denotes the *empty* location, which we also use as a dummy location. SR-Managers $\texttt{SrMan} = \{S, T, ...\}$ are completely specified by a finite natural number or by a symbol, which belongs to a set of reserved symbols Σ_{LTP}. Agents are defined using a set of actions $\texttt{ElAg} = \{\mu, \nu, ...\} = \texttt{Act} \cup \bar{\texttt{Act}} \cup \{\tau, \rho\} \cup \Lambda_{LTP}$, a set of agent variables $\texttt{Avar} = \{X, Y, ...\}$ plus Locs and SrMan. $\texttt{Act} = \{a, b, ...\}$ is a set of elementary actions, $\bar{\texttt{Act}} = \{\bar{a}, \bar{b}, ...\}$ is a set of actions dual to Act, Λ_{LTP} is a set of reserved actions, τ is the silent action and ρ is the release action. Σ_{LTP} and Λ_{LTP} are introduced for the sake of the location transfer policy. As alternative policies might be defined, these sets are specified together with the policy.

Then the agents'grammar is the following:

$$\begin{aligned} p ::= \;& \mathbf{nil} \mid \mu.p \mid p + p \mid p \mid p|p \mid p[\Phi] \\ & \mid p\backslash\alpha \mid X \mid (\mathbf{rec}X.p) \mid \\ & \mid u :: p \mid \; < \mu \;\mathbf{at}\; ux > .p \mid \\ & \mid S@p \end{aligned}$$

Here α denotes a set of elementary actions and Φ a renaming function defined on the elementary actions. The operators in the first two lines are the classical CCS operators: nil,prefixing, choice, labelling, communication, restriction,and recursion. In the second line we have written the CCS-with-locations operators: *location prefixing*,(the agent p resides at location u) and *action-location prefixing*, which bounds location variables as the recursive operator binds process variables. Finally, in the third line is the new BL-CCS operator: @ binds an agent to a SR-manager. The priority between operators (from the highest to the lowest) is specified as follows: . , < **at** > , :: , \ , [] , **rec** , @ , | , + .

We introduce the following definitions.

Def. 1 *A BL-CCS agent p is* **well-formed** *iff any action which appears in the agent is in the scope of a SR-manager operator.*

Given two SR-managers S@ and T@, such that T@ is in the scope of S@, S@ (T@) is said to be more **global (local)** *than T@ (S@). The* **local SR-manager** *of an agent p is (if it exists) the most local SR-manager S@ such that p is in the scope of S@.*

An agent p is **initial** *if no location symbol occurs in it.*

The operational semantics is given in the familiar SOS (Structured Operational Semantics) style, typical of Process Algebra.

The transition relation is of type `Ag` $\times$ `Act` $\times$ Loc $\mapsto$ `Ag` and the generic rule will be written as:

$$p \xrightarrow[u]{\mu} q.$$

According to the role played by the locations, rules may be divided broadly in three groups:

- rules which allocate and release locations (for example, action prefixing);
- rules which are only indirectly influenced by locations (the modified rules inherited by CCS and CCS-with-locations);
- rules which describe the location transfer protocol.

The first two groups of rules are summarized in Table.1. Particular attention is paid to the last group of rules, which will be discussed in detail in a separate section.

The rules for action and action-location prefixing simply modify the CCS-L rules, requiring that the local SR-managers have available locations. In **LPR** and **LAC** the SR-manager loses a location, which is used to execute an action and is allocated to an agent. So the application of the rule is possible only if locations are available. When the actions are performed on the dummy location, according to the intuition, no location is consumed and no location is prefixed to the agent. Notice that **LOC** is applicable also when $u = \epsilon$; in this case, $v\epsilon = v$. In **REL1**, only a location is released per-time. The rule is applicable only in the restricted case when the operator is immediately preceded by a location and a SR-manager. **REL2** applies when there is no location to be released.

The second group of rules are identical to CCS-L rules, with the exception of the rules which describe the behaviour of SR-managers(**MAN**) and the synchronization between two processes(**PA3**). **MAN** states the possibility of performing actions using SR-managers local to subagents.

In the case of synchronization, we notice that it is not important "where" the paired actions $\{a, \bar{a}\}$ in the premise are performed, because the net result of the synchronization is the silent action on the empty location. In CCS-L, it is possible to perform a and $\bar{a}$ on on "real" locations because we have an infinite number of locations and we do not need to look for locations on the SR-managers. If we would adopt the same policy in BL-CCS, we would modify unnecessarily SR-managers and agents and we could also prevent synchronizations, which instead should be always possible. For this reason we perform the actions on the empty location.

2.1 Communication Protocols for SR-Managers

When a pattern of connectivity is specified between SR-managers, we might add flexibility to the language allowing, under some circumstances, the migration of locations from a SR-manager to another. We adopt a scheme where an SR-manager is connected to its direct sub-agents. An exception to this "vertical" connectivity is made for parallel operators, which can exchange locations "horizontally". The connectivity scheme is expressed by the rules in Table.2.a. (**SC** rules). Notice rules **SCPA2** and **SCPA3** which implement the "horizontal" location transfer between parallel operators.

Location transfers might be performed between adjacent SR-managers. Several alternative policies can be implemented, according to the directionality and modality of the transfer.

We call *unidirectional* a policy where locations only go from more global SR-managers to more local SR-managers. However, unidirectional policy is not enough powerful to avoid deadlock, caused by lack of locations (*spatial deadlock*), for a class of agents, which would benefit by a more general, *bidirectional*, policy. In a bidirectional policy, we allow also transfers from a more local SR-manager to a more global one.

The *modality* of a transfer specifies when it can be performed. First of all, it is clear that transfers should be allowed only when a SR-manager is empty, because otherwise a pair of SR-managers could always engage in an useless, infinite, exchange of locations. This risk remains if we allow SR-managers to be *eager*, i.e., allow them to ask locations whenever they are empty, even if they do not need them. To avoid it, we introduce a different, *lazy*, modality.

LPR $S@a.p \xrightarrow[l]{a} (S-1)@l :: p$ if $S \geq 1$; **LPR**ϵ $a.p \xrightarrow[\epsilon]{a} l :: p$

LAC $S@ < a \textbf{ at } ux > .p \xrightarrow[ul]{a} (S-1)@p[l/x]$ if $S \geq 1$; **LAC**ϵ $< a \textbf{ at } ux > .p \xrightarrow[u\epsilon]{a} p[\epsilon/x]$

LOC $\dfrac{p \xrightarrow[u]{\mu} p'}{v :: p \xrightarrow[vu]{\mu} v :: p'}$

REL1 $S@vl :: \rho.p \xrightarrow[\epsilon]{\rho} (S+1)@v :: p$ if $S \notin \Sigma_{LTP}$;

REL2 $S@\rho.p \xrightarrow[\epsilon]{\rho} S@p$

a. *Rules which allocate or release locations*

MAN $\dfrac{p \xrightarrow[u]{\mu} p'}{S@p \xrightarrow[u]{\mu} S@p'}$

b. *The new rule for SR-managers*

CH1 $\dfrac{p \xrightarrow[u]{\mu} p'}{(p+q) \xrightarrow[u]{\mu} p'}$ **CH2** $\dfrac{p \xrightarrow[u]{\mu} p'}{(q+p) \xrightarrow[u]{\mu} p'}$

PA1 $\dfrac{p \xrightarrow[u]{\mu} p'}{(p|q) \xrightarrow[u]{\mu} (p'|q)}$ **PA2** $\dfrac{p \xrightarrow[u]{\mu} p'}{(q|p) \xrightarrow[u]{\mu} (q|p')}$

PA3 $\dfrac{p \xrightarrow[\epsilon]{a} p' \text{ , } q \xrightarrow[\epsilon]{\bar{a}} q}{(p|q) \xrightarrow[\epsilon]{\tau} (p'|q')}$

LAB $\dfrac{p \xrightarrow[u]{\mu} p'}{p[\Phi] \xrightarrow[u]{\Phi(\mu)} p'[\Phi]}$

RES $\dfrac{p \xrightarrow[u]{\mu} p'}{p\backslash\alpha \xrightarrow[u]{\mu} p'\backslash\alpha}$ if $\mu \notin \alpha \cup \bar{\alpha}$;

REC $\dfrac{p[\textbf{rec}P.p/P] \xrightarrow[u]{\mu} p'}{\textbf{rec}P.p \xrightarrow[u]{\mu} p'}$

c. *Rules "inherited" by CCS and CCS-L*

Table 1 *BL-CCS Rules - (First part)*

Together with bidirectionality, the lazy modality defines a location transfer protocol, based on a set of reserved actions $\Lambda_{LTP} = \{\lambda?, \lambda!\}$, and a set of reserved symbols $\Sigma_{LTP} = \{?, w, !\}$ which are allowed to appear on the SR-managers.

$$\mathbf{SC_{LOC}} \quad \frac{S@p \xrightarrow[u]{\mu} S'@p'}{S@v :: p \xrightarrow[vu]{\mu} S'@v :: p'}$$

$$\mathbf{SC_{CH1}} \quad \frac{S@p \xrightarrow[u]{\mu} S'@p'}{S@(p+q) \xrightarrow[u]{\mu} S'@p'} \qquad \mathbf{SC_{CH2}} \quad \frac{S@p \xrightarrow[u]{\mu} S'@p'}{S@(q+p) \xrightarrow[u]{\mu} S'@p'}$$

$$\mathbf{SC_{PA1}} \quad \frac{S@p \xrightarrow[u]{\mu} S'@p'}{S@(p|q) \xrightarrow[u]{\mu} S'@(p'|q)} \qquad \mathbf{SC_{PA2}} \quad \frac{S@p \xrightarrow[u]{\mu} S'@p'}{S@(q|p) \xrightarrow[u]{\mu} S'@(q|p')}$$

$$\mathbf{SC_{PA3}} \quad \frac{S@q \xrightarrow[u]{\mu} S'@q'}{(S@p)|q \xrightarrow[u]{\mu} (S'@p)|q'} \qquad \mathbf{SC_{PA4}} \quad \frac{S@q \xrightarrow[u]{\mu} S'@q'}{q|(S@p) \xrightarrow[u]{\mu} q'|(S'@p)}$$

$$\mathbf{SC_{LAB}} \quad \frac{S@p \xrightarrow[u]{\mu} S'@p'}{S@p[\Phi] \xrightarrow[u]{\Phi(\mu)} S'@p'[\Phi]}$$

$$\mathbf{SC_{RES}} \quad \frac{S@p \xrightarrow[u]{\mu} S'@p'}{S@p\backslash\alpha \xrightarrow[u]{\mu} S'@p'\backslash\alpha} \qquad \text{if } \mu \notin \{\alpha, \bar{\alpha}\};$$

$$\mathbf{SC_{REC}} \quad \frac{S@p[\mathbf{rec}P.p/P] \xrightarrow[u]{\mu} S'@p'}{S@\mathbf{rec}P.p \xrightarrow[u]{\mu} S'@p'}$$

a. *Rules based on the SR-Managers' connectivity*

$\mathbf{LT_{LPR}}\ 0@\mu.p \xrightarrow[\epsilon]{\lambda?} ?@\mu.p$ $\qquad$ $\mathbf{LT_{LAC}}\ 0@ < a\ \mathbf{at}\ ux > .p \xrightarrow[\epsilon]{\lambda?} ?@ < a\ \mathbf{at}\ ux > .p$

$\mathbf{LT1}\ 0@?@p \xrightarrow[\epsilon]{\lambda?} ?@\mathbf{w}@p$ $\qquad$ $\mathbf{LT2}\ ?@0@p \xrightarrow[\epsilon]{\lambda?} \mathbf{w}@?@p$

$\mathbf{LT3}\ ?@S@p \xrightarrow[\epsilon]{\lambda!} !@(S-1)@p$ $\qquad$ $\mathbf{LT4}\ S@?@p \xrightarrow[\epsilon]{\lambda!} (S-1)@!@p$

$\mathbf{LT5}\ \mathbf{w}@!@p \xrightarrow[\epsilon]{\lambda!} !@0@p$ $\qquad$ $\mathbf{LT6}\ !@\mathbf{w}@p \xrightarrow[\epsilon]{\lambda!} 0@!@p$

$\mathbf{LT_{REL1}}\ ?@vl :: \rho.p \xrightarrow[\epsilon]{\rho} !@v :: p$

b. *Rules for Location Transfers*

$\mathbf{LPR_!}\ !@a.p \xrightarrow[l]{a} 0@l :: p$ $\qquad$ $\mathbf{LAC_!}\ !@ < a\ \mathbf{at}\ ux > .p \xrightarrow[ul]{a} 0@p[l/x]$

c. *Rules for Action and Action-Location Prefixing with ! symbol*

Table 2. *BL-CCS Rules (Second Part)*

The presence of **?** on the SR-manager means a request of a location, and is generated by a failed attempt to perform an action, unsuccessful because the SR-manager was empty. Location transfers are allowed only to SR-managers marked by this symbol.

Symbol **w** is left on a SR-manager that has received a request of a location (i.e., has assumed the **?** state) and has transmitted the request to a connected SR-manager. The SR-manager remains *w*-aiting for the location.

If the current symbol on the SR-manager is **!**, the SR-manager holds a single location which should be mandatorily used to perform an action, and can not be transferred if not to SR-managers marked by **w**.

Reserved actions $\lambda?$ and $\lambda!$ are used when **?** and **!** are generated and communicated in the process.

This protocol prevents in the major part of the cases spatial deadlock or divergence because of the SR-manager's behaviour. Its rules are summarized in Table.2.b (rules **LT**). Between the others, rule $\mathbf{LT_{REL1}}$ complete the description of the behaviour of the release action ; implicitly a **!** symbol on the SR-manager temporarily suspends a release action.

Finally rules $\mathbf{LPR_!}$ and $\mathbf{LAC_!}$ make explicit the use of **!** as a "borrowed" location.

3 An Example

We would like now to illustrate, through an example, the behaviour of the transition system. The context in which a rule is applied should justify its definition.

Let us consider the agent:

$$1@(a.\bar{c}.b|(2@\mathbf{nil}+c.d)|1@(a.c+a.b.c.d))$$

Its connectivity pattern is illustrated in Figure.1. Here is a possible computation. The subagent on which the action is reduced is underlined.

$$\begin{array}{lll}
1@(a.\bar{c}.b|(2@\mathbf{nil}+c.d)|\underline{1@}(a.c+\underline{a.b.c.d})) & \xrightarrow[l_1]{a} & (1)\\
1@(a.\bar{c}.b|(2@\mathbf{nil}+c.d)|\underline{0@l_1 :: b.c.d})) & \xrightarrow[\epsilon]{\lambda?} & (2)\\
\underline{1@}(a.\bar{c}.b|(2@\mathbf{nil}+c.d)|\underline{?@l_1 :: b.c.d})) & \xrightarrow[\epsilon]{\lambda!} & (3)\\
\underline{0@}(\underline{a.\bar{c}.b}|(2@\mathbf{nil}+c.d)|!@l_1 :: b.c.d)) & \xrightarrow[\epsilon]{\lambda?} & (4)\\
\underline{?@}(a.\bar{c}.b|\underline{(2@\mathbf{nil}+c.d)}|!@l_1 :: b.c.d)) & \xrightarrow[\epsilon]{\lambda!} & (5)\\
\underline{!@}(\underline{a.\bar{c}.b}|1@\mathbf{nil}|!@l_1 :: b.c.d)) & \xrightarrow[m]{a} & (6)\\
0@(m :: \bar{c}.b|1@\mathbf{nil}|\underline{!@l_1 :: b.c.d})) & \xrightarrow[l_2]{b} & (7)\\
0@(\underline{m :: \bar{c}.b}|1@\mathbf{nil}|\underline{0@l_1 l_2 :: c.d})) & \xrightarrow[\epsilon]{\tau} & (8)\\
0@(m :: b|1@\mathbf{nil}|\underline{0@l_1 l_2 :: d})) & \xrightarrow[\epsilon]{\lambda?} & (9)\\
0@(m :: b|\underline{1@\mathbf{nil}|?@l_1 l_2 :: d})) & \xrightarrow[\epsilon]{\lambda!} & (10)\\
0@(m :: b|1@\mathbf{nil}|\underline{!@l_1 l_2 :: d})) & \xrightarrow[l_3]{d} & (11)\\
0@(m :: b|0@\mathbf{nil}|0@l_1 l_2 l_3 :: \mathbf{nil})) & &
\end{array}$$

Here are some short comments. (1) is a standard action execution. In (3) there is a "vertical" location transfer between a "father" SR-manager and one of its "children" SR-managers. The request of location in (4) provokes the solution of a non-deterministic choice(5). At this point the connectivity pattern allows "horizontal" location transfers, as shown in Figure.2. Notice that it is not possible for the "father" to acquire back the location previously ceased to the child SR-manager. It is easy to see that it would be possible for the two SR-managers to engage in a potentially infinite exchange of locations.

(8) solves in favour of synchronization the alternative with a request of location. Without the dummy location, it would be quite involved to define the correct behaviour, which should leave untouched the other parts of the agents.

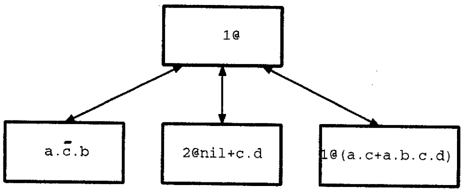

Figure.1. *Initial connections in the agent*

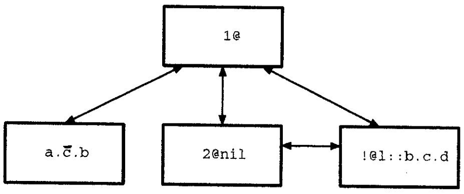

Figure.2. *Connections after step.5*

Finally in (10) we have an "horizontal" location transfer between two sibling agents. However, as the leftmost parallel operand can not be completed, because there is no location available, there is spatial deadlock.

Clearly the equivalent CCS agent, without SR-managers and locations, is not in deadlock; also the equivalent CCS-L agent, without SR-managers, can supply always locations and here also no deadlock would be observed.

4 Future Work

The research which has been introduced in this paper can continue along various thematic lines.

In concurrent languages, it is often observed the need to combine the functionality of a program with the management of its resource requirements. It would be interesting to test in a more practical setting the expressiveness of BL-CCS, where these two aspects are combined in the same framework.

We have proposed a definition of BL-CCS which is parametric to the location transfer policy,and we have consequently defined the lazy bidirectional policy. Other policies can be defined to capture alternative desired behaviours.

Process equivalences and axiomatizations are very important and thoroughly studied in process-algebraic models. The standard bisimulation technique, which make equivalent two programs when they have the same set of possibile sequential computations has been applied to CCS-L, together with suitable techniques to deal with locations. At the moment we are studying an analogous equivalence(and partial order) for BL-CCS. An alternative, finer, equivalence(and partial order) can be defined to compare the efficiency of the processes in terms of number and length of performed location trasfers.

As a closing observation, we would like to notice that BL-CCS completes the separation between spatial contiguity and causal dependency. Such separation exists already in CCS-L, but the two notions still remain connected for sequence of actions, which will always lie in adjacent locations. Inserting a release action between two actions, we allow them to be performed on non-contiguous locations in spite of being causal dependent.

References

[BCHK91] G.Boudol, I.Castellani, M. Hennessy, A.Kiehn. *Observing Localities (Extended Abstract).* Mathematical Foundations of Computer Science, 1991. LNCS 520

[BCHK92] G.Boudol, I.Castellani, M.Hennessy, A.Kiehn. *A Theory of Processes with Localities (Extended Abstract).* CONCUR'92. LNCS 630

[FM91] G.L.Ferrari, U.Montanari. *The Observation Algebra of Spatial Pomsets.* CONCUR'91,LNCS 527

[Ho85] C.A.R.Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, N.J., 1989

[Ki91] A.Kiehn. *Local and Global Causes.* Report 342/23/91, Technische Universität München.

[Mi89] R.Milner. *Communication and Concurrency.* Prentice-Hall, Englewood Cliffs, N.J., 1989