Box Calculus with High-Level Buffers

Cécile Bui Thanh, Hanna Klaudel, Franck Pommereau

Box Calculus with High-Level Buffers

Cécile Bui Thanh
LACL, Université Paris 12
61 avenue du général de Gaulle, 94010 Créteil, France
bui@univ-paris12.fr

Hanna Klaudel
LaMI, Université Evry-Val d'Essonne
523 Place des Terrasses, 91000 Evry, France
klaudel@lami.univ-evry.fr

Franck Pommereau
LACL, Université Paris 12
61 avenue du général de Gaulle, 94010 Créteil, France
pommereau@univ-paris12.fr

Keywords: Process algebra, handshake/buffered communication, structured operational semantics, coloured Petri nets.

Abstract

In this paper, we propose a high-level process algebra allowing to express the exchange of data values using both handshake and buffered communication. This allows a simple and compositional expression of interprocess communication; in particular the buffered one makes easy the representation of program variables, allowing a compact representation of large systems. The process terms are provided with an operational semantics as well as with a denotational one in terms of high-level Petri nets. The main result is that both semantics are consistent in the sense that a term and the corresponding Petri net generate isomorphic transition systems.

INTRODUCTION

The modelling of concurrent systems involves a representation of interprocess communication (which may be handshake or buffered), and very often that of data like program variables. This representation should be compact and readable in order to avoid design errors. A first step for improving the readability is the structured modelling which is a main characteristic of process algebras. Concerning the communication, process algebras are historically based on the paradigm of handshake communication and there are two classical ways to model the buffered one. The first way consists in considering each communication as buffered, as for instance in the asynchronous π-calculus [5]: the output prefix $\overline{a}x$ in the term $\overline{a}x|a(x).P$ is interpreted as a message x already present on the channel a, and available for reception. Doing so, handshake communication becomes not available, since the channels are implicitly buffered. The second way consists in considering a buffered communication as two different actions (a sending and a receiving) managed by a medium. This medium can be explicitely added to the modelisation by its parallel composition with the system, like in Milner's π-calculus [17], or implicitly included in the formalism, by adding a syntactic feature representing a message already sent but not yet received, like in the algebraic version of Linda[1] [9] or in Broy and Olderog's formalism used in [6]. Note also that multiway communication is usually not supported by process algebras, and if it is the case, only handshake is available, like in SCCS [16] or PBC [1–3]. Most efforts given in the domain of process algebras to add high-level features like, for instance, in M-expressions [14] or value-passing CCS [16], do not consider both buffered and handshake multiway communication. The *Asynchronous Box Calculus with Multiway Communication* [7] is a low-level formalism suitable for modelling distributed systems in a compositional way. In the following, we will call it *the low-level model*. It is composed of an algebra of process terms with a fully compositional translation into labelled Petri nets (called *boxes*) and it was shown that the semantics of a process term obtained by the rules of the structural operational semantics (SOS) was equivalent to the step sequence semantics of the associated Petri net. Thus, this model takes both advantages from process algebras for the design, and from Petri nets for the efficiency of system verification [11]. Moreover, it allows for both handshake and buffered multiway communication.

The aim of this paper is to introduce high-level features in this low-level model in order to simplify the modelling of complex systems manipulating data. Using the low-level model for such systems requires to use, for instance, one buffer to represent each possible value of a modelled variable, which is not readable for large data types and may become humanly intractable in complex cases. In the extension proposed here, the changes concern the syntactical level (terms), the associated SOS rules, as well as the Petri net level (boxes). The main extension consists in introducing high-level buffers, which are able to carry values from an associated set. (Thus, these buffers become useful for representing variables in a compact way.) This leads to introduce *parameters* in both actions and links, and also to add a *guard* (*i.e.*, a boolean expression) to atomic terms. As in the low-level model, an encapsulation mechanism for buffered communication is proposed; this does not exist in other process algebras and in particular in [9, 6].

The main result is that the obtained high-level framework, called *Box Calculus with Data* (BCD), is consistent

[1] Meanwhile, Linda does not allow for handshake communication.

in the sense that a term and the corresponding net generate isomorphic transition systems. This is obtained by showing the consistency with the low-level model w.r.t. a translation called *unfolding*. Moreover, it turns out that the used class of Petri nets was shown in [12, 18] sufficient to model the semantics of a high-level parallel specification language B(PN)^2 [4] and was used in [10] to express a compositional semantics of the π-calculus [15].

The paper is structured as follows: we first introduce the BCD algebra of terms. Next, we give the structural operational semantics and the denotational semantics through a translation to boxes. Finally, we give the main results of this work and conclude. The present paper is an extended abstract of the technical report [8] containing the complete formalisation and proofs.

AN ALGEBRA OF TERMS

We introduce first some notions and notations needed to formalise the high-level features of our extension.

We assume that there is a finite set $\mathbb{A}$ of *action symbols* used to model handshake communication. Each action symbol $a \in \mathbb{A}$ has the *arity* $\mathsf{ar}(a)$ corresponding to the number of parameters used during a handshake communication. We also assume that, for all $a \in \mathbb{A}$, there exists a conjugated $\widehat{a}$ in $\mathbb{A}$ such that $\widehat{\widehat{a}} = a$ and $\mathsf{ar}(a) = \mathsf{ar}(\widehat{a})$. An *action* is composed of an action symbol and parameters which may be *values* from the finite set $\mathbb{V}$ or *variables* from the set $\mathbb{X}$. For instance, if $\mathsf{ar}(a) = 2$, then $a(x, 1)$ and $\widehat{a}(2, y)$ are examples of actions. A multiset of such actions is called a *multiaction*. We consider also a finite set $\mathbb{B}$ of *buffer symbols* (or *buffers*) intended for buffered communication. Each buffer $b \in \mathbb{B}$ is assigned a non-empty set $\mathsf{type}(b) \subseteq \mathbb{V}$, representing all the values it can carry. Communication through a buffer b is represented with *links* of the form $b^+(e)$ for sending a value (represented by the expression e) to the buffer b, or $b^-(e)$ for receiving. For instance $b^+(x+1)$ or $b^-(2)$ denote links. Moreover, in process terms, we will use the notations $.b(v)$, with $v \in \mathsf{type}(b)$, to denote the presence of a value v in the buffer b.

We consider an algebra of process terms over the following signature:

$$\mathcal{C} \cup \{\overline{\cdot}, \underline{\cdot}\} \cup \{\|, \fatsemi, \Box, \circledast\} \cup \{\mathsf{sc}\, a \mid a \in \mathbb{A}\} \cup \{.b(v), \mathsf{tie}\, b \mid b \in \mathbb{B}, v \in \mathsf{type}(b)\}$$

where $\mathcal{C} \stackrel{\text{df}}{=} \{\langle\alpha|\beta|\gamma\rangle \mid \alpha$ is a multiaction, β is a multiset of links and γ is a guard$\}$ are the constants; the binary operators $\|$, $\fatsemi$, $\Box$ and $\circledast$ will be used in the infix mode; the unary operators $\mathsf{sc}\, a$, $\mathsf{tie}\, b$ and $.b(v)$ will be used in the postfix mode; and $\overline{\cdot}$ and $\underline{\cdot}$ are two positional unary operators (the position of the argument being given by the dot). The operators of BCD will have the following intuitive meaning: *iteration* $E_1 \circledast E_2$ (E_1 can be executed an arbitrary number of times, and then is followed by E_2); *buffer restriction* $E\,\mathsf{tie}\, b$ (the buffer b and the related links become private to E); *scoping* $E\,\mathsf{sc}\, a$ (setup handshake communication involving the actions a or $\widehat{a}$); and *buffer stuffing* $.b(v)$ (add a value v in the buffer b). *Sequence* $E_1 \fatsemi E_2$, *choice* $E_1 \Box E_2$ and *parallel composition* $E_1 \| E_2$ have their classical meanings.

There are two classes of process terms: the *static* and *dynamic* ones, denoted respectively by E and D. Collectively, we will refer to them as the *terms*, F. Their syntax is given by:

$$\begin{aligned} E ::= {} & \langle\alpha|\beta|\gamma\rangle \mid E\,\mathsf{sc}\, a \mid E\,\mathsf{tie}\, b \mid E.b(v) \mid E\|E \\ & \mid E \Box E \mid E \fatsemi E \mid E \circledast E \\ D ::= {} & \overline{E} \mid \underline{E} \mid D\,\mathsf{sc}\, a \mid D\,\mathsf{tie}\, b \mid D.b(v) \\ & \mid D\|D \mid E\,\mathsf{bin}\, D \mid D\,\mathsf{bin}\, E \end{aligned}$$

where $\langle\alpha|\beta|\gamma\rangle \in \mathcal{C}$, $a \in \mathbb{A}$, $b \in \mathbb{B}$, $v \in \mathsf{type}(b)$ and $\mathsf{bin} \in \{\Box, \|, \fatsemi, \circledast\}$. Notice that, since the order of $b_i(v_i)$'s will be irrelevant in terms of the form $F.b_1(v_1) \cdots .b_k(v_k)$, we will sometimes use the shorthand $F.B$, where B is the multiset containing all the $b_i(v_i)$'s.

Essentially, a term encodes the structure of the control, together with the current state of the execution (using overbars and underbars) and the state of the buffers (using the $.b(v)$'s). Thus, a term $\overline{E}$ represents E in its initial state, while $\underline{E}$ represents E in its final state. The execution of a term depends on the evaluation of the expressions it contains which may be done for a chosen binding $\sigma : \mathbb{X} \to \mathbb{V}$ which assigns values to variables. This way, an atomic BCD term can have as many executions as there are distinct enabling bindings for it.[2] A binding σ is *enabling* for a term $\langle\alpha|\beta|\gamma\rangle$ if the guard evaluates to true, $\mathsf{eval}(\sigma(\gamma)) = \top$, and if for each link of the form $b^-(e)$ or $b^+(e)$ in β, the concerned expression evaluates to a value from the type of the link, $\mathsf{eval}(\sigma(e)) \in \mathsf{type}(b)$. For instance, if $\mathsf{type}(b) = \mathbb{V}$, the atomic term $\langle a(x) \mid b^+(x+1) \mid x \in \{1,2\}\rangle$ has two enabling bindings $\sigma_1 = \{x \mapsto 1\}$ and $\sigma_2 = \{x \mapsto 2\}$ each of them leading to a distinct potential execution. Such an execution is possible if the term is in its initial state; for instance, $\overline{\langle a(x) \mid b^+(x+1) \mid x \in \{1,2\}\rangle}$ may be executed with the binding σ_2, producing a value 3 in the buffer b and a visible bound action $(\{a(x)\}, \sigma_2)$ (which intuitively corresponds to the action $a(2)$), and leads to the term $\underline{\langle a(x) \mid b^+(x+1) \mid x \in \{1,2\}\rangle.b(3)}$.

It is worth remarking that the buffer stuffing $.b(v)$ is needed for static as well as for dynamic terms because the dormant part of a dynamic term may have $.b(v)$'s which may be later needed if it becomes active.

The scoping is defined through partial functions $\varphi_{\mathsf{sc}\, a}$ (for each $a \in \mathbb{A}$) which allow to enforce the handshake communication w.r.t. a and to forbid the independent execution of events still involving a or $\widehat{a}$. More precisely, $\varphi_{\mathsf{sc}\, a}$ may be applied to a pair (m_α, δ), where m_α is a multiset of multiactions whose elements (multiactions) can synchronise together under the substitution δ; in this case, $\varphi_{\mathsf{sc}\, a}(m_\alpha, \delta)$ yields the resulting multiaction. For instance, the fact that $\varphi_{\mathsf{sc}\, a}(\{\{a(x), a(3)\}, \{\widehat{a}(5)\}, \{\widehat{a}(y), a_2\}\}, \{x \mapsto 5, y \mapsto 3\}) = \{a_2\}$ means that, the involved multiactions can perform a three-way synchronisation ($a(x)$ is synchronised

[2] We identify the bindings which coincide for all the variables of a term but may differ on other variables not involved in the term.

CON1	$\dfrac{F \equiv F'}{F\,\mathsf{una} \equiv F'\,\mathsf{una}}$	CON2	$\dfrac{F_1 \equiv F_1', F_2 \equiv F_2'}{F_1\,\mathsf{bin}\,F_2 \equiv F_1'\,\mathsf{bin}\,F_2'}$
ENT	$\dfrac{E \equiv E'}{\overline{E} \equiv \overline{E'}}$	EX	$\dfrac{E \equiv E'}{\underline{E} \equiv \underline{E'}}$
OPL	$(F.b(v))\,\mathsf{bin}\,F' \equiv (F\,\mathsf{bin}\,F').b(v)$	OPR	$F\,\mathsf{bin}(F'.b(v)) \equiv (F\,\mathsf{bin}\,F').b(v)$
E1	$\overline{E}\,\mathsf{una} \equiv \overline{E\,\mathsf{una}}$	X1	$\underline{E}\,\mathsf{una} \equiv \underline{E\,\mathsf{una}}$
B1	$(F.b(v))\,\mathsf{una} \equiv (F\,\mathsf{una}).b(v)$ if $\mathsf{una} \neq \mathsf{tie}\,b$		
IS1	$\overline{E_1 \fatsemi E_2} \equiv \overline{E_1} \fatsemi E_2$	IS2	$\underline{E_1} \fatsemi E_2 \equiv E_1 \fatsemi \overline{E_2}$
IS3	$E_1 \fatsemi \underline{E_2} \equiv \underline{E_1 \fatsemi E_2}$	REN	$F \equiv \rho(F)$
IPAR1	$\overline{E_1 \| E_2} \equiv \overline{E_1} \| \overline{E_2}$	IPAR2	$\underline{E_1} \| \underline{E_2} \equiv \underline{E_1 \| E_2}$
IC1L	$\overline{E_1 \Box E_2} \equiv \overline{E_1} \Box E_2$	IC1R	$\overline{E_1 \Box E_2} \equiv E_1 \Box \overline{E_2}$
IC2L	$\underline{E_1} \Box E_2 \equiv \underline{E_1 \Box E_2}$	IC2R	$E_1 \Box \underline{E_2} \equiv \underline{E_1 \Box E_2}$
IIT1	$\overline{E_1 \circledast E_2} \equiv \overline{E_1} \circledast E_2$	IIT2	$\underline{E_1} \circledast E_2 \equiv \overline{E_1} \circledast E_2$
IIT3	$\underline{E_1} \circledast E_2 \equiv E_1 \circledast \overline{E_2}$	IIT4	$\overline{E_1} \circledast E_2 \equiv E_1 \circledast \overline{E_2}$
IIT5	$E_1 \circledast \underline{E_2} \equiv \underline{E_1 \circledast E_2}$		

Fig. 1. Structural similarity relation, where $b \in \mathbb{B}$, $v \in \mathsf{type}(b)$, una stands for any unary operator, bin stands for any binary operator and ρ is a substitution on $\mathbb{X}$.

with $\widehat{a}(5)$ and $a(3)$ is synchronised with $\widehat{a}(y)$), where the substitution $\{x \mapsto 5, y \mapsto 3\}$ has to be applied, and this synchonisations leads to the multiaction $\{a_2\}$. These functions $\varphi_{\mathsf{sc}\,a}$ are used to enforce CCS-like synchronisations, but with no limitation on the number of simultaneous participants. They existed already in the low-level model and are extended in order to take the bindings of variables into account (see [7, sec. 2.2] for all the details).

SEMANTICS

Structural operational semantics

As usual, the operational semantics of terms is given by SOS rules; however, instead of expressing the evolutions by modifying the structure of the terms, like $a.E \xrightarrow{a} E$ in CCS, the idea here is to represent the current state of the evolution using overbars and underbars, corresponding respectively to the initial and final states of (sub)terms. These rules are entirely detailed in [8].

The first kind of rules defines an equivalence relation $\equiv$, called the *structural similarity relation*, which allows to identify different terms which actually denote the very same state. For instance, in a sequential composition, having the first component in final state is equivalent to having the second one in initial state: $\underline{F_1} \fatsemi F_2 \equiv F_1 \fatsemi \overline{F_2}$ for any BCD term F_1 and F_2. The relation $\equiv$ as the least equivalence relation on terms such that all the equations in the figure 1 are satisfied. Most of these rules are those of the low-level version of the process algebra, and adapted to fit the syntax which now specifies which value v is stored in a buffer b, by using $.b(v)$ instead of just $.b$. A new rule specifies also that variables may be renamed. It is worth noting that an important part of the operational semantics relies in this equivalence relation.

The second kind of rules specifies when an expression may evolve producing a state change. This is expressed through rewriting rules of the form $F \xrightarrow{\Gamma} F'$, where F and F' are terms, and Γ is a step, *i.e.*, is a multiset of pairs (α, σ), where α is a multiaction and σ a binding. For instance, we will have:

$$\overline{\langle a_2 \mid b^-(y) \mid \top \rangle.b(3)} \xrightarrow{\{(\{a_2\},\{y \mapsto 3\})\}} \underline{\langle a_2 \mid b^-(y) \mid \top \rangle}\ .$$

These rules are given in the figure 2. Each α in a step corresponds to the execution of one atomic term and the associated σ is the binding used during this execution. In the rule LA, a term $\overline{\langle \alpha | \beta | \gamma \rangle.B}$ may produce a move (α, σ) if the buffers hold sufficiently many values (represented by the multiset B of values in the buffers) in order to cover all the receiving links from β. Such a move consumes B and produces the multiset B' of new values in the buffers, according to the sending links in β.

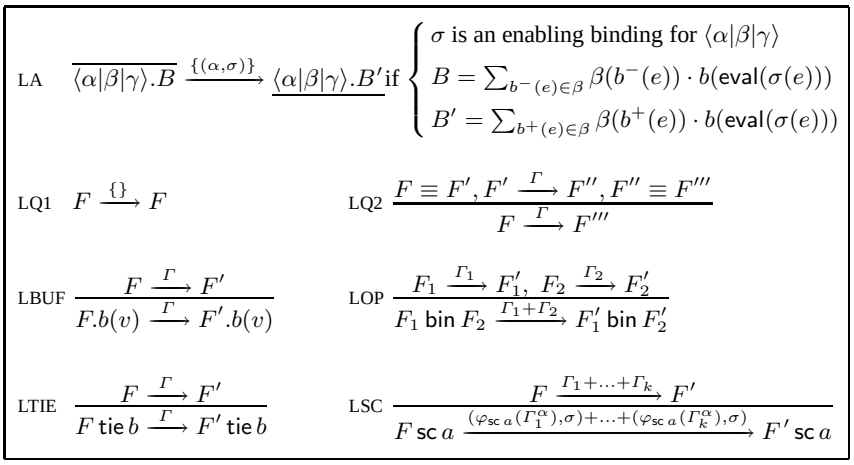

Fig. 2. The operational semantics for BCD, where $a \in \mathbb{A}$, $b \in \mathbb{B}$, $v \in type(b)$, bin stands for any binary BCD operator, and in the rule LSC σ is a binding such that for $1 \leq i \leq k$ and for all $(\alpha, \sigma') \in \Gamma_i$, we have $\sigma' = \sigma$ and $\Gamma_i^\alpha \stackrel{\text{df}}{=} \left(\sum_{(\alpha,\sigma')\in\Gamma_i} \Gamma_i(\alpha,\sigma)\cdot\sigma(\alpha)\right) \in \mathsf{dom}(\varphi_{\mathsf{sc}\,a})$.

Denotational semantics

The denotational semantics of terms is given by a mapping box which associates to each BCD term a labelled high-level Petri net called a *box*. A box is a Petri net in which we distinguish through a special labelling the *entry*, *internal* and *exit* places which are used to represent the control flow of the modelled system and can only carry ordinary tokens $\bullet$. In particular, a term E in its initial state $\overline{E}$ will be translated to a box with one token in each entry place (initial marking), represented by $\overline{\mathsf{box}(E)}$; similarly the final state $\underline{E}$ corresponds to have one token in each exit place of $\mathsf{box}(E)$ (final marking). We also distinguish the *buffer places* which are used to implement the buffered communication: each $b \in \mathbb{B}$ corresponds to a buffer place s_b labelled by b which may hold tokens in $\mathsf{type}(b)$. The transitions of a box are labelled by pairs $\alpha\gamma$ where α is a multiaction and γ is a guard; each transition actually corresponds to an atomic term. The links specified by the β part of atomic terms give rise to the arcs between transitions and buffer places.

The class of boxes is provided with a set of operations (based on the labelling of places and transitions) which exactly correspond to those defined on BCD terms: for each operator on terms there exists a similar one operating on boxes. For instance, the sequential composition of two boxes N_1 and N_2 is defined and denoted by $N_1 \fatsemi N_2$; it glues the exit places of N_1 with the exit places of N_2 and merges the buffer places with the same label. The complete definitions of all these operations are given in [8]. The strong similarity between the two domains allows us to give a simple inductive definition of the semantical mapping box.

The basic building blocks, from which composite boxes are constructed, are the boxes $N_{\langle\alpha|\beta|\gamma\rangle}$, each one being defined as follows. Its set of places is composed of one entry place s_e, one exit place s_x and one buffer place s_b labelled by b for each $b \in \mathbb{B}$. It has only one transition t labelled by $\alpha\gamma$, which has one incoming arc labelled by $\{\bullet\}$ from s_e and one outgoing arc to s_x with the same label. The other arcs correspond to the links in β and we have for each $b \in \mathbb{B}$ as many arcs from s_b to t labelled by e as there are links $b^-(e)$ in β, and analogously for the arcs from t to s_b. Given this base case, the function box is defined inductively by:

$$\mathsf{box}(\langle\alpha|\beta|\gamma\rangle) \stackrel{\text{df}}{=} N_{\langle\alpha|\beta|\gamma\rangle} \qquad \mathsf{box}(F\,\mathsf{una}) \stackrel{\text{df}}{=} \mathsf{box}(F)\,\mathsf{una}$$

$$\mathsf{box}(\overline{E}) \stackrel{\text{df}}{=} \overline{\mathsf{box}(E)} \qquad \mathsf{box}(\underline{E}) \stackrel{\text{df}}{=} \underline{\mathsf{box}(E)}$$

$$\mathsf{box}(F_1\,\mathsf{bin}\,F_2) \stackrel{\text{df}}{=} \mathsf{box}(F_1)\,\mathsf{bin}\,\mathsf{box}(F_2)$$

where $\langle\alpha|\beta|\gamma\rangle \in \mathcal{C}$, una stands for any unary operator ($\mathsf{sc}\,a$, $\mathsf{tie}\,b$ or $.b(v)$, with $a \in \mathbb{A}$, $b \in \mathbb{B}$, $v \in \mathbb{V}$), and bin for any binary operator ($\|$, $\Box$, $\fatsemi$ or $\circledast$).

An example of box of BCD terms in a producer-consumer system is given in the figure 3:

$$\textsc{Prod} \stackrel{\text{df}}{=} \big(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(0) \circledast \langle f_p \mid b_p^-(n) \mid \top\rangle\big)\,\mathsf{tie}\,b_p\ ,$$

$$\textsc{Cons} \stackrel{\text{df}}{=} \big(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x), b^-(x) \mid \top\rangle.b_c(0) \circledast \langle f_c \mid b_c^-(k) \mid k \geq 4\rangle\big)\,\mathsf{tie}\,b_c$$

$$\textsc{Sys} \stackrel{\text{df}}{=} \textsc{Prod} \,||\, \textsc{Cons}$$

The producer repetitively generates random resources on a buffer b, updates the private counter of produced resources b_p and emits a multiaction $\{a_p\}$. It may terminate at any time, by emitting a multiaction $\{f_p\}$. The consumer repetitively consumes a pair of the same value, updates the private counter of consumed resources b_c and emits a multiaction $\{a_c\}$. It may terminate at any time, if it has consumed at least 4 resources.

In the figure, the unrestricted buffers are represented by a place with double line, while the restricted buffers are represented by a place in bold. In the producer/consumer example, the restricted buffers correspond to the buffers b_p and b_c after the application of $\mathsf{tie}\,b_p$ and $\mathsf{tie}\,b_c$. Notice that

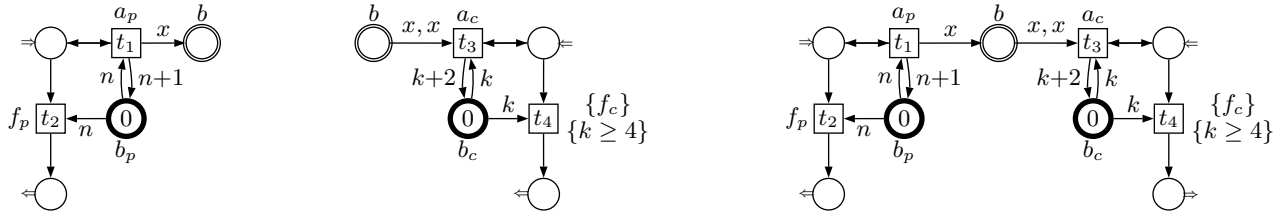

Fig. 3. Example of BCD boxes involved in a producer-consumer system. The buffer places may carry tokens in $\mathbb{N}$ while the other places may only hold $\bullet$.

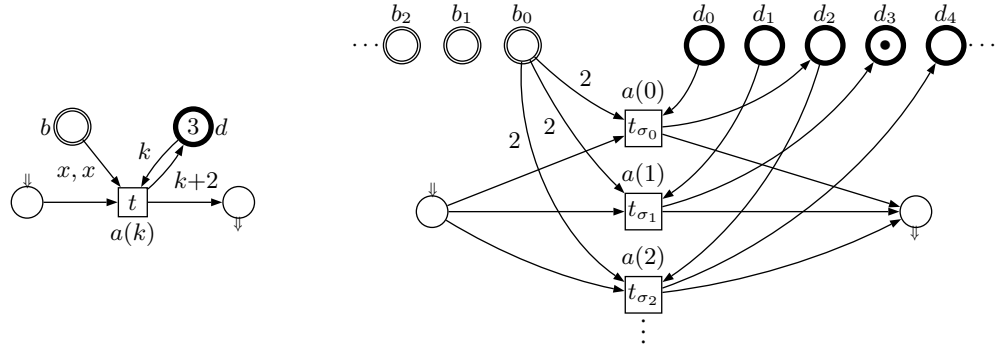

Fig. 4. The box of the BCD term $\langle a(k) \mid b^-(x), b^-(x), d^-(k), d^+(k+2) \mid \top \rangle . d(3) \,\mathsf{tie}\, d$ and a fragment of its unfolding, where $\sigma_0 = \{x \mapsto 0, k \mapsto 0\}$, $\sigma_1 = \{x \mapsto 0, k \mapsto 1\}$, and $\sigma_2 = \{x \mapsto 0, k \mapsto 2\}$.

some annotations are omitted in the figure, like the guards which are always true, or the annotation $\{\bullet\}$ in the arcs.

UNFOLDING AND CONSISTENCY RESULTS

The translations from the high-level domains of BCD terms, boxes and steps to the corresponding low-level domains are called *unfoldings* and are denoted in all cases by unf.

Low-level boxes sensibly differ from high-level ones: the places are not typed and may only carry tokens $\bullet$; the transitions are labelled by multisets of low-level actions (whose parameters are only values) and do not have guards; the arcs are defined through a weight function representing the number of tokens flowing on the arcs; the marking function returns the number of tokens held by each place instead of a multiset of values. The unfolding is defined in such a way that each transition t of a high-level box gives rise to a set of low-level transitions t_σ, where σ is an enabling binding for t; the label of t_σ is that of t evaluated through σ (without the guard which evaluates to true). Each high-level place s is unfolded into a set of low-level places s_v, for each value v in the set of tokens allowed for s. The low-level arcs and markings are obtained in a similar way. An example of the unfolding of a box is provided in the figure 4.

The low-level version of terms has the following syntax (with $\mathsf{bin} \in \{\|, \Box, \fatsemi, \circledast\}$ as above):

$$E ::= \langle \alpha_\ell | \beta_\ell \rangle \mid E \,\mathsf{sc}\, a_\ell \mid E \,\mathsf{tie}\, b_\ell \mid \quad E.b_\ell \quad \mid E \,\mathsf{bin}\, E$$

$$D ::= \quad \overline{E} \quad \mid \quad \underline{E} \quad \mid D \,\mathsf{sc}\, a_\ell \mid D \,\mathsf{tie}\, b_\ell \mid \quad D.b_\ell \quad \mid \quad D \| D \quad \mid D \,\mathsf{bin}\, E \mid E \,\mathsf{bin}\, D$$

where a_ℓ is a low-level action and α_ℓ is a multiset of such actions, b_ℓ is a low-level buffer and β_ℓ is a multiset of low-level links (with only values as parameters).

The unfolding of BCD terms is defined by induction on their syntax. As the base case, the unfolding of an atomic term is the choice between all the low-level terms it encodes:

$$\mathsf{unf}(\langle \alpha | \beta | \gamma \rangle) \stackrel{\text{df}}{=} \bigsqcap_{\sigma \in \mathfrak{B}_{\langle \alpha | \beta | \gamma \rangle}} \langle \alpha_\sigma | \beta_\sigma \rangle \,,$$

where $\mathfrak{B}_{\langle \alpha | \beta | \gamma \rangle}$ is the set of all the bindings enabling $\langle \alpha | \beta | \gamma \rangle$, α_σ is the multiset of low-level actions obtained by evaluating through σ all the parameters of actions from α, and β_σ is the multiset of low-level links obtained from β in a similar way. For instance, the unfolding of the BCD term $\langle a(x) \mid b^+(x+1) \mid x \in \{1,2\} \rangle$ is $\langle a(1) \mid b^+(2) \rangle \,\Box\, \langle a(2) \mid b^+(3) \rangle$.

Then, for terms F, F_1 and F_2, $a \in \mathbb{A}$, $b \in \mathbb{B}$, $d \in \mathsf{type}(b)$ and any binary operator bin, the unfolding is defined as follows:

$$\begin{aligned} \mathsf{unf}(F \,\mathsf{sc}\, a) &\stackrel{\text{df}}{=} \mathsf{unf}(F) \,\mathsf{sc}\, \mathsf{unf}(a) \\ \mathsf{unf}(F \,\mathsf{tie}\, b) &\stackrel{\text{df}}{=} \mathsf{unf}(F) \,\mathsf{tie}\, \mathsf{unf}(b) \\ \mathsf{unf}(F.b(d)) &\stackrel{\text{df}}{=} \mathsf{unf}(F).b_d \\ \mathsf{unf}(F_1 \,\mathsf{bin}\, F_2) &\stackrel{\text{df}}{=} \mathsf{unf}(F_1) \,\mathsf{bin}\, \mathsf{unf}(F_2) \\ \mathsf{unf}(\overline{F}) \stackrel{\text{df}}{=} \overline{\mathsf{unf}(F)} \qquad & \mathsf{unf}(\underline{F}) \stackrel{\text{df}}{=} \underline{\mathsf{unf}(F)} \end{aligned}$$

where $\mathsf{unf}(a)$ is the set of all low-level actions of the form $a(v, \ldots, v')$ in which $v, \ldots, v'$ are values from $\mathbb{V}$, and $\mathsf{unf}(b)$ is the set of all low-level buffers of the form b_v for all $v \in \mathsf{type}(b)$. Notice that, since $\mathsf{unf}(a)$ is a finite set of low-level actions, $\mathsf{sc}\, \mathsf{unf}(a)$ is in fact a shorthand for successive applications of scoping with respect to all the elements of $\mathsf{unf}(a)$ (which may be applied in any order thanks

to the commutativity of scoping). The situation is similar for $\mathsf{tie}\,\mathsf{unf}(b)$.

The unfolding of a step is obtained by applying its bindings to the corresponding multiactions.

Consistency results

The main result of the paper is the consistency of the model w.r.t. the unfolding which states that *for any term, the semantics of its unfolding coincides with the unfolding of its semantics*. This statement holds for each piece of the semantics: the box mapping, the step semantics of boxes, the relation $\equiv$ and the moves defined by $\longrightarrow$. The moves in the box domain are denoted by $[\,\rangle$. The proof and more results can be found in [8].

Theorem 1. *Let F and F' be two terms and Γ a step. Then:*

1. $\mathsf{unf}(\mathsf{box}(F)) = \mathsf{box}(\mathsf{unf}(F))$.
2. $\mathsf{box}(F)[\Gamma\rangle\,\mathsf{box}(F')$ $\iff$ $\mathsf{unf}(\mathsf{box}(F))[\mathsf{unf}(\Gamma)\rangle\,\mathsf{unf}(\mathsf{box}(F'))$.
3. $F \equiv F' \Longrightarrow \mathsf{unf}(F) \equiv \mathsf{unf}(F')$.
4. $F \xrightarrow{\Gamma} F' \iff \mathsf{unf}(F) \xrightarrow{\mathsf{unf}(\Gamma)} \mathsf{unf}(F')$.

As shown in [7], the box and the SOS semantics of a low-level term are equivalent in arguably the strongest sense (they generate isomorphic transition systems), which allows to show the following result.

Theorem 2. *The SOS and box semantics of a BCD term generate isomorphic transition systems. Thus:*

$$F \xrightarrow{\Gamma} F' \iff \mathsf{box}(F)[\Gamma\rangle\,\mathsf{box}(F')\ .$$

CONCLUSION

We presented how a compositional low-level framework may be extended in order to cope with systems manipulating large data types. The main change consisted in using buffers capable to carry coloured tokens. This feature was exploited in both buffered and handshake communication through high-level parameterised links and actions. This extension provided a real progress from a practical point of view since it allows to represent in a very compact way systems with large data types (as, for instance, the FIFO channel developed in the running example). Moreover, independent papers [12, 10] showed the relevance of the used class of composable Petri nets which was powerful enough to give an elegant compositional semantics to parallel specification languages like B(PN)^2 or formalisms like π-calculus. From the theoretical point of view, the obtained framework, called BCD, was shown to be a coherent high-level counter-part of the existing low-level model in the sense that a BCD term and its unfolding have the same behaviour and similarly for a BCD box and its unfolding. As a consequence, a BCD term and its corresponding box generate isomorphic transition systems.

Future works will be dedicated to introduce in this framework the modelling of preemption inspired by [13]. It will need first a representation of sub-terms of which several concurrent instances may be started. This should lead to a very complete process algebra capable to express, in particular, the semantics of parallel programming languages with exceptions.

References

1. E. Best, R. Devillers and J. Hall. *The Petri Box Calculus: a New Causal Algebra with Multilabel Communication.* Advances in Petri Nets 1992, LNCS 609, 1992.
2. E. Best, R. Devillers and M. Koutny. *A Unified Model for Nets and Process Algebras.* Handbook of Process Algebra, Elsevier, 2001.
3. E. Best, R. Devillers and M. Koutny. *Petri Net Algebra.* EATCS Monographs on TCS, Springer, 2001.
4. E. Best and R. P. Hopkins. *B(PN)^2 – a Basic Petri Net Programming Notation.* PARLE'93, LNCS 694, 1993.
5. G. Boudol. *Asynchrony and the π-calculus.* Research Report 1702, INRIA, Sophia Antipolis, 1992.
6. M. Broy and E.-R. Olderog. *Trace-Oriented Models of Concurrency.* J.A. Bergstra, A.Ponse, and S.A. Scott, editors, Handbook of Process Algebra, Elsevier Science, 2001.
7. C. Bui Thanh, H. Klaudel and F. Pommereau. *Asynchronous Box Calculus with Multi-way Communication.* LACL Tech. Report, Univ. Paris 12, 2002. Available on ⟨`http://www.univ-paris12.fr/lacl`⟩.
8. C. Bui Thanh, H. Klaudel and F. Pommereau. *Box Calculus with Coloured Buffers.* LACL Tech. Report, Univ. Paris 12, 2002. Available on ⟨`http://www.univ-paris12.fr/lacl`⟩.
9. N. Busi, R. Gorrieri and G. Zavattaro *A process algebraic view of Linda coordination primitives.* Theoretical Computer Science 192(2), 1998.
10. R. Devillers, H. Klaudel and M. Koutny. *Compositional High-Level Petri Net Semantics of π-calculus.* Manuscript.
11. J. Esparza. *Model checking using net unfoldings.* Science of Computer Programming, 23. Elsevier, 1994.
12. H. Klaudel *Parameterized M-expression semantics of parallel procedures.* DAPSYS'00, Kluwer Academic Publishers, 2000.
13. H. Klaudel et F. Pommereau. *A class of composable and preemptible high-level Petri nets with an application to multitasking systems.* Fundamenta Informaticae, 50(1), IOS Press, 2002.
14. H. Klaudel and R.-C. Riemann *High Level Expressions with their SOS Semantics.* Proceedings of CONCUR'97, LNCS 1243, Springer, 1997
15. R. Milner, J. Parrow and J. Walker. *A calculus of mobile processes, Parts I and II.* Information and Computation, 100(1), 1992.
16. R. Milner *Communication and Concurrency.* Prentice Hall, 1989.
17. J. Parrow. *An Introduction to the π-calculus.* J.A. Bergstra, A.Ponse, and S.A. Scott, editors, Handbook of Process Algebra, Elsevier Science, 2001.
18. F. Pommereau. *FIFO buffers in tie sauce.* DAPSYS'00, Kluwer Academic Publishers, 2000.