31
From LOTOS to Petri Nets through Iexpansion

D. Larrabeiti, J. Quemada, S. Pavón
Technical University of Madrid
Department of Telematic Engineering, ETSIT
Ciudad Universitaria s/n, 28040 Madrid, Spain
email: {dlarra,jquemada,spavon}@dit.upm.es

Abstract

The *interleaved expansion* presented in [QLP93] is a method to compute and represent the global events of a system preserving parallelism. As a side effect, this method provides important advantages in the technology of translation from LOTOS to Petri nets over static (direct) transformations and opens a simple process algebra form of representation for an important class of Petri nets. This work presents an algorithm that maps ITcalulus elements into labeled Petri nets and gives a comprehensive overview of different aspects of the translation from LOTOS to Petri nets using this intermediate form of representation.

Keywords

FDT-based system and protocol engineering, LOTOS, Petri nets, Extensions of FDTs

1 INTRODUCTION

Many approaches exist searching for efficient execution models for LOTOS [ISO89] both for automatic derivation of prototypes and for symbolic validation. The most straightforward is the EFSM obtained by expansion tools [QPF89]. However, although the analysis and execution of an EFSM is far less complex than any equivalent representation on concurrent models, in practice EFSMs of most real systems have a size too big to be used in verification or as run-time engine models for prototypes. This problem has opened several paradigms around EFSM exploration - mainly classical testing and on-the-fly methods of verification - and about EFSM representation - namely reduction theory, compositional minimization, compressed representation of EFSMs such as BDDs, etc -.

At the other end of the rope of execution models, there is LOTOS itself. In between, there is a variety of forms of representation product of transformational approaches which attempt to reduce the complexity of LOTOS at compilation-time. Examples of these forms of representation and execution, to cite but a few contrasted with known tools, are: execution kernels based on native LOTOS [MS92], communicating state machines [Kar88], locally expanded processes [Kre94], etc.

R. Gotzhein et al. (eds.), *Formal Description Techniques IX*

One of these alternative models can be Petri nets due to its well-known properties and the nice trade-off between compactness and complexity they provide. In fact, the corresponding Petri net representation of a LOTOS specification usually has a reasonable size, quite close to the original specification, and it includes all global events of the system. Furthermore, it is well known the feasibility of prototype execution on Petri nets e.g. [LDdlP+93].

From the protocol design viewpoint, the automatic translation from LOTOS to Petri nets is specially interesting in multi-FDT design environments where both description techniques must cooperate so as to take advantage of the specific good feature of each technique most suitable at each stage of the design process. This is particularly important in a top-down design methodology with Petri nets based prototypes as target [LDdlP+93]. By this way, one can take advantage of the high level design features of LOTOS -compositionality, abstraction, correctness preserving transformations, ..- in a first phase, and then benefit from the lower level properties of Petri nets, mainly executability, making use of a wide range of available PN tools, and even apply true concurrency semantics

However, the technology of execution of LOTOS specifications on net-based models seems to go behind LOTOS-based prototype execution, due to considerable constraints in the domain of the transformation (mainly, static control constraints) and the static nature of existing translations [ML89] [GS90] [SV95].

In [QLP93] it was pointed out the strong relationship between ITcalculus and Petri nets. Furthermore, ITcalculus can be considered as a process algebra representation of a large family of Petri nets. In the following sections, the most complex aspects of the translation from LOTOS to Petri nets are presented and how this operation can be simplified by using the interleaved expansion as a previous step. The added-value of using the interleaved expansion for LOTOS to Petri net translation is the dynamic nature of this transformation which presents several advantages over existing works, as discussed in section 5.

2 LOTOS AND PETRI NETS

No strict equivalence between Petri nets and LOTOS models exist. It is only possible to translate a subset of LOTOS expressions into a certain class of Petri nets, although most practical specifications with a bounded number of processes and events fall in this category. The type of semantics adopted is not relevant as long as all the parallelism of the original specification is thoroughly preserved. Anyway, a vast work exists on Petri nets using interleaving semantics e.g. [Rei85] [Rei87] [Roz87], especially on reachability analysis.

It is known that the description of concurrency in LOTOS presents a certain resemblance with the one of the Petri nets refered above. *AND*-nodes in Petri nets (transition with several input arcs) have their counterparts in LOTOS synchronizations, LOTOS choice can be expressed as *OR*-nodes (places with several output arcs), etc. The main difference is that Petri nets display the global events of the system, not explicit in LOTOS. In this sense, LOTOS provides a higher level of abstraction -may be closer to the way the designer structures and implements his concurrent systems- and allows to design separately each concurrent thread and use multiway rendez-vous as interprocess commu-

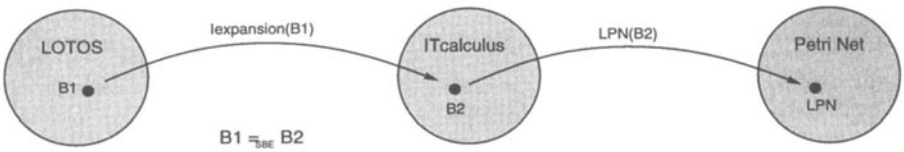


Figure 1 LOTOS, ITcalculus, Iexpansion, Petri Nets

nication mechanism. Furthermore, LOTOS potentially allows a richer variety of types of process interactions. On the other hand, the classic procedure to obtain the global events of a LOTOS specification is by means of the application of the Expansion Theorem, which most times is an unfeasible path in practice.

There are a few factors which determine the quality of a translation from LOTOS to Petri nets and the scope of LOTOS specifications which can be converted. These are: 1.Preservation of Parallelism, 2.Gate relabeling, 3.Process instantiation and recursion, 4.Data types and 5.Disabling. We shall review these items in detail in section 5 in order to compare results with previous works.

From the previous discussion, it can be drawn that it should not be difficult to find a translation from LOTOS to Petri nets but for the fact that a lot of information about the global events of the specification is "hidden" by the LOTOS semantics. Figure 1 describes the approach proposed in this work, in which a LOTOS behaviour B_1 is transformed by the *interleaved expansion* into an equivalent iexpanded form B_2 which can be mapped onto a Petri net LPN more easily since B_2 contains all the dynamic semantics information required for this purpose. Section 3 give a description of the domain of the latter transformation before defining function **LPN**(B) itself in section 4.

3 INTERLEAVED EXPANSION

3.1 Concept

The *Interleaved Expansion* [QLP93] - abbreviated *Iexpansion* - is a transformation that takes as input a LOTOS behaviour and yields an equivalent (strong bisimulation) behaviour expression in a new calculus called *ITcalculus* (centre of figure 1). We shall call this resultant expression the *iexpanded form* of the original specification.

The most interesting property of the iexpansion is its ability to factor out and isolate the interleaving zones of a LOTOS behaviour, avoiding the application of those rules of the Expansion Theorem of LOTOS [ISO89] which provoke state or transition explosion: namely those that derive interleaving actions in parallel operators or actions from the first operand of disabling operators.

The purpose of the interleaved expansion is the computation of all the global events of a system and represent them once, jointly with the inter-event relation (precedence, conflict, disabling,..). Thus, it is not a classical expansion in the sense that it does not construct an EFSM in which states are explicit and one system event is represented by as many transitions as states can trigger it. However, it has many common features with

expansions due to its dynamic nature: all synchronizations are computed and the data value exchanged in them is replaced in the subsequent behaviour to compute next synchronizations, predicates get evaluated, hiding and gate relabeling are removed, it accepts parameterization, etc. Only parallelism is preserved, like in Petri nets or event structures, but following an approach based on extending LOTOS, therefore taking advantage of the efficiency of these true concurrency forms of representation from the LOTOS standpoint.

The representation of the output of the iexpansion in the process algebra domain is possible thanks to ITcalculus. ITcalculus is a subset of LOTOS enriched with three new elements that jointly allow to preserve independence of processes, to do without non-interleaving parallel operators and to express a multievent precedence relation.

These elements are **IT** operator, continuations and terminations. The intuition behind these elements is the following. A LOTOS expression can be expressed as an ITcalculus expression headed by an **IT** operator $\mathbf{IT}(B, Cs)$. Its first operand, B, is the active behaviour which may derive either normal actions or *terminations*, that are special events filtered out by the **IT** operator and whose only utility is enabling real actions from the continuations. The second operand,$Cs = \cup_{c \in C} < c > B_c$, the *continuation set*, is a set of behaviours labeled with the names of the terminations $< c >$ which must be offered in the active behaviour in order to enable its attached behaviour B_c. This idea is illustrated by the following simple example.

Example 31 Let B = a; d; stop|[d]|b; d; stop|[d]|c; d; stop . The execution of events a , b and c is independent, but all of them must precede d. This can be expressed in ITcalculus as

$$\mathbf{Iexpansion}(B) = B_1 = \mathbf{IT}(a; 1|||b; 2|||c; 3, \{<1,2,3> d; stop\})$$

where 1, 2, 3 are terminations used to label the local states whose composition enables a synchronized action. This can also be read as *our system has an active behaviour in which three independent events can be executed concurrently; if the system evolved to a state in which there were three processes in a local state* $<1,2,3>$ *then* d *would be enabled, and the involved behaviours would become* **stop**. Any order of execution leads to the state in which d is enabled.e.g. $B_1 \xrightarrow{b;a;c} \mathbf{IT}(1|||2|||3, \{<1,2,3> d; stop\}) \xrightarrow{d} \mathbf{stop}$

The number of terminations labeling the continuation denotes the cardinality of the synchronization. □

3.2 Notation

The symbols used to represent universal sets and variables over their sets and their elements are the following:

- $\mathcal{N}$ is the set of natural numbers.
  $I, J, K, M, N \subset \mathcal{N}$ arbitrary index sets.
  $i, j, k, l \in \mathcal{N}$ indexes.
- $\mathcal{B}$ superset of behaviour expressions.
  $\mathcal{B}_{\text{LOTOS}}$ and $\mathcal{B}_{\text{IT}}$ respectively denote the universe of LOTOS and ITcalculus expressions when it is necessary to make such distinction. The syntax and semantics of these expressions are defined in section 3.3.
  $B, B_n, B', B_n', B'' \in \mathcal{B}$ are variables over behaviour expressions

- $\mathcal{T} \subset$ N is the alphabet of terminations.
  $c, c', c_i \in \mathcal{P}(\mathcal{T})$ are finite termination sets.
  $C, C', C_n, ... \subset \mathcal{P}(\mathcal{T})$ sets of termination sets.
  $n, m, n', n_i, m_i, .. \in \mathcal{T}$ are variables ranging over termination labels. In ITcalculus, the terms termination and termination label can be used indistinctly.
- $\mathcal{L}$ is the vocabulary of non-termination event names.
  $A, A', A_n \subset \mathcal{L}$ will represent finite gate sets.
  $g, g', g_n \in \mathcal{L}$, gate names, visible event labels. They stand for alphanumeric strings of finite length.
- $\mathcal{W} = \mathcal{L} \cup \mathcal{T}$ alphabet of event labels (terminations and non-terminations)
  W, W', W_n stand for event labels sets.
  $w, w', .., w_n \in \mathcal{W}$ are event labels.
- $Cs, Cs_1, Cs_2 \subset \mathcal{P}(\mathcal{T}) \times \mathcal{B}$ continuation sets.
  Each continuation is characterised by its first component: a termination set, that labels the continuation. i.e. $\forall Cs \in \mathcal{B}, \forall (c_1, B_1), (c_2, B_2) \in Cs : c_1 = c_2 \Rightarrow B_1 = B_2$.
  The sintax chosen for the pairs (c, B) in ITcalculus is $< c > B_c$. Therefore, the continuation sets are represented : $Cs = \cup_{c \in C} < c > B_c$.
- $\sum_{i \in I} B_i$ is used as a generalised choice operator. $\sum_{i \in \{1..n\}} B_i = B_1 \; [] \; \ldots \; [] \; B_n$.
  Note that $\sum_{i \in \{\}} B_i =$ stop.
- $\mathbf{L}(B)$ is the set of visible event names of a behaviour B. $\mathbf{L} : \mathcal{B} \rightarrow \mathcal{L}$
  $\mathbf{T}(B)$ is the set of terminations of B. $\mathbf{T} : \mathcal{B} \rightarrow \mathcal{T}$
- $\mathcal{P}$ superset of Petri net places.
  P, P', P_n stand for event place sets.
  $p, p', p_n \in \mathcal{P}$ represent places and t, t', t_n Petri net transitions.
  $\bullet t$ represents the set of input places to a transition t. $t\bullet$ is the output place set of t.

3.3 ITcalculus

For the sake of clarity in the presentation, we shall consider only the ITcalculus image of LOTOS without data types and disabling. The inclusion of these elements would simply add extra verbosity and a little more complexity in the calculus, without any new concepts in exchange. These elements are reviewed in section 4.

Definition 1 Syntax of ITcalculus . Table 1 summarizes the *syntax of ITcalculus.* The first part of this table is practically common to the language accepted as input for the iexpansion. The second part contains the three new elements: *terminations*, *continuation set* and **IT** operator.

□

Note that only interleaving rules of LOTOS parallel operators are present (**A4.1**, **A4.2**) since the iexpansion computes all the synchronizations of the specification. ITcalculus expressions resulting from the iexpansion have the additional syntactical constraint that `i)` terminations only appear in behaviour expressions that are arguments of an **IT** operator and `ii)` continuation sets can only be the second operand of an **IT** operator. The interleaved expansion produces a single **IT** operator which heads the iexpanded form. That is, **Iexpansion**$(B) =$ **IT**(B_1, Cs).

Operator	Syntax	L(B)	T(B)
Inaction	**stop**	{}	{}
Action Prefix	g;B	$\mathbf{L}(B) \cup \{g\}$	$\mathbf{T}(B)$
Choice	$B_1[]B_2$	$\mathbf{L}(B_1) \cup \mathbf{L}(B_2)$	$\mathbf{T}(B_1) \cup \mathbf{T}(B_2)$
Parallel	$B_1 \vert\vert\vert B_2$	$\mathbf{L}(B_1) \cup \mathbf{L}(B_2)$	$\mathbf{T}(B_1) \cup \mathbf{T}(B_2)$
Process Instantiation	*proc_name*[*gl*] where **process** *proc_name*[*gl*] : *func* := B **endproc**	$\mathbf{L}(B)$	
Termination	n	{}	{n}
Continuation Set	$\{<c_1>B_{c_1},..,<c_n>B_{c_n}\}$	$\cup_{c \in \{c_1..c_n\}} \mathbf{L}(B_c)$	$\cup_{c \in \{c_1..c_n\}} \mathbf{T}(B_c)$
IT	IT(B, Cs)	$\mathbf{L}(B) \cup \mathbf{L}(Cs)$	{}

Table 1 Syntax of ITcalculus

Definition 2 ITcalculus semantics is defined as a ***LTS*** (Labeled Transition System) which is obtained by the application of the inference rules listed in table 2 to an ITcalculus behaviour expression. □

Rule **A5.2** formalizes the concepts introduced in section 3.1. Terminations labeled by natural numbers $n_1, .., n_i$ identify local states which enable real transitions present in a set of behaviours labeled $<n_1, .., n_i>$. Since terminations only appear within an **IT**, the transition system will never have terminations as transitions, because **A5.2** filters them out. After a transition has been triggered out of a continuation, the remainder of the active behaviour still interleaves.

Example 32 Let B = `a;y;d;stop |[y]| b;y;z;e;stop |[z]| c;z;f;stop`
its interleaved expansion yields
B′ = `IT( a;1|||b;2|||c;3, { <1,2>y;(d;stop|||4), <3,4>z;(e;stop|||f;stop) } )`
The behaviour following an action of a continuation contains the resumption of all interrupted processes. The escape through a continuation does not affect the processes that do not take part in the synchronization. Thus e.g. B′ $\xrightarrow{a;b;y}$ `IT(d;stop|||4|||c;3,{..})`
□

3.4 Interleaved Expansion

This work only defines ITcalculus. We shall assume that there exists a transformation which generates the iexpanded form of any LOTOS behaviour expression and hence is capable of removing all operators not defined in ITcalculus from the original specification (hiding,relabeling,let,synchronizations,etc.). The reader is referred to [QLP93] for a detailed definition of the interleaved expansion for a subset of basic LOTOS and to [Lar96] for a definition for full LOTOS.

Action prefix	**A1**	$g;B \xrightarrow{g} B$
Termination	**A2**	$n \xrightarrow{n} stop$
Choice	**A3.1**	$\dfrac{B_1 \xrightarrow{w} B_1'}{B_1 \Box B_2 \xrightarrow{w} B_1'}$ **A3.2** $\dfrac{B_2 \xrightarrow{w} B_2'}{B_1 \Box B_2 \xrightarrow{w} B_2'}$
Parallel	**A4.1**	$\dfrac{B_1 \xrightarrow{w} B_1'}{B_1 \|\|\| B_2 \xrightarrow{w} B_1' \|\|\| B_2}$ **A4.2** $\dfrac{B_2 \xrightarrow{w} B_2'}{B_1 \|\|\| B_2 \xrightarrow{w} B_1 \|\|\| B_2'}$
IT	**A5.1**	$\dfrac{B_1 \xrightarrow{g} B_1'}{IT(B_1, \cup_{c \in C} <c> B_c) \xrightarrow{g} IT(B_1', \cup_{c \in C} <c> B_c)}$
	A5.2	$\dfrac{B_0 \xrightarrow{n_1} B_1, B_1 \xrightarrow{n_2} B_2, .., B_{i-1} \xrightarrow{n_i} B_i, B_{\{n_1,..,n_i\}} \xrightarrow{g} B', \{n_1, .., n_i\} \in C}{IT(B_0, \cup_{c \in C} <c> B_c) \xrightarrow{g} IT(B_i \|\|\| B', \cup_{c \in C} <c> B_c)}$

Table 2 LTS($\mathcal{B}$). Transition System of ITcalculus.

As shown intuitively in the examples, what the interleaved expansion does is pruning the original behaviour and setting on it terminations to tag local states enabling synchronized events. These references allow to index and build the continuations that contains the events resultant from each synchronization. This procedure is repeated in every new continuation until all synchronizations are contained in the global continuation set.

3.5 Duplicate behaviour detection

According to all previous works it can be claimed that the most difficult issue to deal with - and thereby important- in the translation from LOTOS to Petri nets is the conversion of recursion. Therefore, it is worth explaining how the interleaved expansion deals with this problem.

The compositionality of LOTOS makes it possible to speak about processes and processes made up by the composition of processes. One way to formalise and refer to each possible composition of processes in a LOTOS specification is by means of the concept of *synchronization context.* From a generic context we can obtain information such as: synchronization set, gate relabeling function, hiding set, values of variables, etc. In particular, for the definition of interleaved expansion for the subset of LOTOS chosen is

enough to know the structure of parallels of the context. We shall call this information *synchronization context* and it will allow us to identify and refer to different zones in the structures of processes in the specification. In order to formalise this in a simple way we shall define the following concepts:

Definition 3 Parameterized behaviour expressions . Let $\mathcal{C}$ be the superset of behaviour expressions resultant of extending $\mathcal{B}_{\text{LOTOS}}$ with free variables over $\mathcal{B}_{\text{LOTOS}}$. Expressions in $\mathcal{C}$ will be called *parameterized behaviour expressions.* $C[p_1, .., p_n] \in \mathcal{C}$ stands for a behaviour expression parameterized by variables $p_1, .., p_n$. □

Definition 4 Context. Two types of parameterized behaviour expressions are of particular interest: the ones with only one parameter, also known as *LOTOS contexts* [ISO89], and the subset of these which presents only one occurrence of its parameter, which will be denoted by $\mathcal{C}_{11}$. We shall use the term *context* to refer to expressions $C[p] \in \mathcal{C}_{11}$. □

Definition 5 Void Context. Let p be a variable $p \in \mathcal{C}_{11}$; we say that $C[p] = p$ is the *void context* or *identity* context. □

As mentioned before, we shall use contexts to identify different zones in the dynamic structure of parallel operators in the specification. Thus, C[p] = p, the void context, identifies the global process defined by the specification; C[p] = stop|[A]|p represents the synchronization context that characterizes the right operand of parallel |[A]|, etc.

Each synchronization context defines a local state subspace in which it is possible to perform a duplicate behaviour detection, etc. Note that no difference is made between the contexts of B_2 and B_4 in $B_1|[A_1]|((B_2|[A_2]|B_3)[](B_4|[A_2]|B_5))$ (or likewise between B_3 and B_5)

The procedure to deal with recursive LOTOS behaviours is analogous to the one used in the normal expansion. In outline, the expansion stores *global states* visited in depth-first order after having been compared -syntactical comparison of behaviours except for variable renaming- with the ones already visited [Pav90]. When a duplicate state is found, its expanded behaviour -whose expansion may not be complete- is reused using the basic mechanism provided by LOTOS for reutilization: process instantiation and definition.

In the interleaved expansion, that method is essentially valid, but *within synchronization contexts instead of in a single global state space.* This means that the detection of duplicate behaviours is performed in disjoint sets of *compound states* (ranging from compound states made up by one process (local states), a subset of processes, to all of them (global states)).

As the space necessary to perform a duplicate behaviour detection is proportional to the number of behaviours stored, the memory consumption depends upon the degree of interleaving in the specification. If the specification is a strongly interleaved one, the memory used should match the sum of sizes of each process local state space, much less than the size of the global state space, proportional to the product.

A natural question arisen is, what happens with identical compound states but in different synchronization contexts ? The answer is that omitting the detection of this kind of recursion does not influence whether the iexpansion is terminating or not. Indeed, the behavioural causes for non-expandibility are: unbounded unguarded recursions and unbounded dynamic composition of processes. The former hinder the computation of a

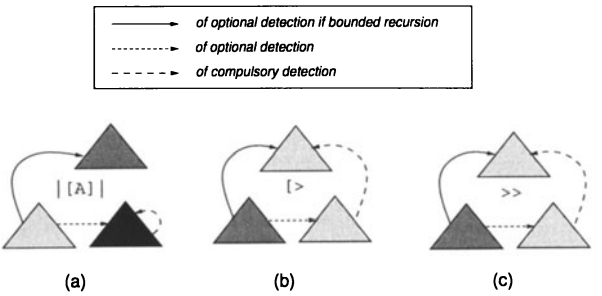


Figure 2 Types of duplicate behaviours detection according to their contexts

state in any expansion; the latter obstruct terminateness of state exploration algorithms and are always due to recursions through different synchronization contexts.

Figure 2 shows symbolically the different types of recursion identified according with its incidence in the terminateness of the expansion and iexpansion w.r.t. the parallel operator **(a)**, disabling **(b)** and enabling **(c)**. Shaded zones represent different synchronization contexts. The current definition and implementation of the interleaved expansion presented here does not detect potentially divergent recursions (depicted in continuous trace). The synchronization constraints and data values will determine dynamically whether the recursion actually causes infiniteness or not.

In outline, the consequences of not detecting duplicate behaviours in disjoint contexts are

- either *it does not affect the termination of the algorithm*(although in some cases can be optionally performed to improve performance), or the behaviour is divergent and hence the expansion is not possible anyway.
- *independence of code* among processes is achieved. Each concurrent thread gets its own subset of processes. This enables an homogeneous translation from ITcalculus to Petri nets, just by replacing process instantiations with the Petri net equivalent of its process definition, preventing different tokens from sharing places.

4 FROM ITCALCULUS TO PETRI NETS

Once defined ITcalculus and described the way the interleaved expansion deals with recursion, the translation of a LOTOS specification to a Petri net can be notably simplified if performed on its iexpanded form and, it can provide less redundant nets. All transitions in the iexpanded form are already computed without expansion and all of them are *actual* transitions, not *potential* transitions like in a static translation. There are less elements to be translated: hiding, relabeling, enabling, synchronization, local definition, gate sum-expression, par-expression and most guards, predicates and value choices are resolved in an iexpanded form. Finally, an important subject stated in section 3.5, recursions are computed over disjoint contexts, and thus resultant nets will be contact-free and one-safe with little effort.

4.1 Translation

Let us give a brief definition of a translation from ITcalculus (without disabling and data types) to Petri nets. The class of Petri nets chosen as target for the translation are contact-free one-safe Petri nets, where arc weights are binary and places have one token utmost. Each token will represent a concurrent thread. Petri net transitions will be labeled with the event names from an alphabet $\mathcal{L}$ in order to yield a labeled reachability graph using interleaving semantics, which should be equivalent to the LTS generated by the LOTOS expression. Places originated by terminations are labeled with the corresponding termination identifier (in fact, local states), but this labeling can be removed at the end.

Definition 6 Labeled Petri Net is a graph with two types of nodes alternated: *places* -which may have associated one token- and *transitions*, which may be labeled. It is represented by a tuple LPN $=< P, T, F, P_0, L_T, L_P >$ where

- $P \subset \mathcal{P}$ is a non-empty finite set of places.
- T is a finite set of transitions.
- $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs.
- $P_0 \subseteq P$ is the set of initially marked places.
- $L_T : T \rightarrow \mathcal{L}$ is a function to label transitions.
- $L_P : P \rightarrow \mathcal{N}$ is a function to label places.

Its semantics is defined by

$$\bullet t \subseteq P_0 \implies < P, T, F, P_0, L_T, L_P > \xrightarrow{L_T(t)} < P, T, F, (P_0 - \bullet t) \cup t\bullet, L_T, L_P > \qquad \square$$

We do accept the case of a net with an empty set of transitions to represent the LOTOS behaviour **stop**. Other authors [ML89] translate stop as a net transition, what hinders a natural interpretation of the equivalence B[]stop = stop (B would be disabled by an inexistent transition).

The translation ITcalculus to Petri nets is defined by function **LPN** table 3 case **A0**. The equivalent Petri net of a subexpression of an iexpanded form headed by any operator is constructed by composition of the subnets of its operands. **A1** through **A7** in table 3 define the translation of each operator.

Definition 7

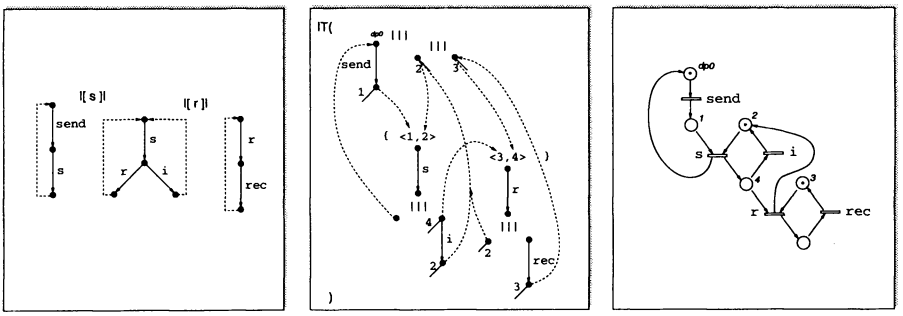
- Let **Place** be an arbitrary injective function that assigns a different place to each behaviour in a given synchronization context (definition 4):
  Place : $\mathcal{C}_{11} \times \mathcal{B}_{IT} \longrightarrow \mathcal{P}$
- Let **Trans** be an arbitrary injective function that assigns a different transition to each behaviour in a given synchronization context:
  Trans : $\mathcal{C}_{11} \times \mathcal{B}_{IT} \longrightarrow \mathcal{T}$ $\square$

On Recursive behaviours

An implementation of function **PN** should interpret case **A7** (table 3) as a rule that only applies if **PN**(C[x], *proc_name*[*gl*]) has not yet been computed. Otherwise, the algorithm would not finish on recursive behaviours. This would imply adding a new parameter

A0 LPN(B)	$= \mathbf{PN}(x, B) =< P, T, F, P_0, L_T, L_P >,\ x \in \mathcal{C}_{11}$
A1 PN(C[x], **stop**)	$= < \{p\}, \{\}, \{\}, \{p\}, \{\}, \{\} >$ where $p = \mathbf{Place}(C[x], \mathbf{stop})$
A2 PN(C[x], e; B′)	$= < \{p\} \cup P', \{t\} \cup T', F \cup F', \{p\}, (t, \mathbf{Label}(e)) \cup L'_T, L'_P >$ if $\mathbf{Label}(e) \neq \delta$ where $t = \mathbf{Trans}(C[x], e; B')$ $p = \mathbf{Place}(C[x], e; B')$ $\mathbf{PN}(C[x], B') =< P', T', F', P'_0, L'_T, L'_P >$ $F = \{(p, t) \cup \cup_{p_i \in P'_0} \{(t, p_i)\}\}$
A3 PN(C[x], $B_1 \vert\vert\vert B_2$)	$= < P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2, P_{01} \cup P_{02}, L_{T1} \cup L_{T2}, L_{P1} \cup L_{P2} >$ where $\mathbf{PN}(C[x] \vert\vert\vert \mathbf{stop}, B_1) =< P_1, T_1, F_1, P_{01}, L_{T1}, L_{P1} >$ $\mathbf{PN}(\mathbf{stop} \vert\vert\vert C[x], B_2) =< P_2, T_2, F_2, P_{02}, L_{T2}, L_{P2} >$
A4 PN(C[x], $B_1 [] B_2$)	$= < P, T, F, P_0, L_{T1} \cup L_{T2}, L_{P1} \cup L_{P2} \cup L_{P12} >$ where $\mathbf{PN}(C[x], B_1) =< P_1, T_1, F_1, P_{01}, L_{T1}, L_{P1} >$ $\mathbf{PN}(C[x], B_2) =< P_2, T_2, F_2, P_{02}, L_{T2}, L_{P2} >$ $P_0 = \{p_{ij} \in \mathcal{P} \mid (i,j) \in P_{01} \times P_{02} \ \wedge \ p_{ij} \notin P_1 \cup P_2\}$ $P = (P_1 - P_{01}) \cup (P_2 - P_{02}) \cup P_0$ $T = T_1 \cup T_2$ $F = (F_1 - F_{01}) \cup (F_2 - F_{02}) \cup F_{12}$ $F_{01} = \{(p, t) \in F_1 \mid p \in P_{01}\}$ $F_{02} = \{(p, t) \in F_2 \mid p \in P_{02}\}$ $F_{12} = \{(p_{ij}, t) \subseteq P_0 \times T \mid (i, t) \in F_{01} \ \vee \ (j, t) \in F_{02}\}$ $L_{P12} = \{(p_{ij}, n) \subseteq P_0 \times \mathcal{N} \mid (i,j) \in P_{01} \times P_{02} \ \wedge$ $((i, n) \in L_{P1} \ \vee \ (j, n) \in L_{P2})\}$
A5 PN(C[x], n)	$= < \{p\}, \{\}, \{\}, \{p\}, \{\}, (p, n) >$ where $p = \mathbf{Place}(C[x], n)$
A6 PN(C[x], **IT**($B_1, \cup_{c \in C} <c> B_c$))	$= < P, T_1 \cup \cup_{c \in C} T_c, F_1 \cup \cup_c F''_c, P_{01}, L_{T1} \cup \cup_{c \in C} L_{Tc}, L_P >$ where $P = P_1 \cup_c ((P_c - \{p_{0c}\}) \cup P''_{0c})$ $L_P = L_{P1} \cup \cup_{c \in C} L_{Pc}$ $\mathbf{PN}(C[x], B_1) =< P_1, T_1, F_1, P_{01}, L_{T1}, L_{P1} >$ and, for each $c = \{n_1, \ldots, n_m\}$ $\mathbf{PN}(C[x], B_c) =< P_c, T_c, F_c, \{p_{0c}\}, L_{Tc}, L_{Pc} >$ since $B_c = \sum_i g_i; B_i$ $P''_{0c} = \{p \in P_1 \cup_{c \in C} P_c \mid L_P(p) \in c\}$ $T_{0c} = p_{0c}{}^\bullet$ $F_{0c} = \{p_{0c}\} \times T_{0c} = \{(p, t) \in F_c \mid p = p_{0c}\}$ $F''_c = (F_c - F_{0c}) \cup (P''_{0c} \times T_{0c})$
A7 PN(C[x], *proc_name*[*gl*])	$= \mathbf{PN}(C[x], B)$ where **process** *proc_name* [gl]:*func*:= B **endproc**

Table 3 Equivalent Petri net of an ITcalculus expression B



(a) (b) (c)

Figure 3 Example of translation

keeping the set of visited process instantiations and their equivalent subnets to be returned instead of being recomputed.

Note that the way in which the detection of duplicate behaviours is performed during the iexpansion ensures that there are not instantiations of the same process in different synchronization contexts, and the ones within the same synchronization context are necessarily in choice. This, jointly with the definition of the function employed to generate places **Place**(), means that all processes will have disjoint place sets and hence, the nets are contact-free. Since there can only be one token per synchronization context and recursion can only occur within the same synchronization context, the whole net has to be one-safe.

Example 41 The following specification describes a system composed of two link layer protocol entities communicated by an unreliable physical medium. The communication is simplex and no error recovery protocol is implemented. Actions `send` and `rec` are abbreviations of link layer primitives `link_data_request` and `link_data_indication` respectively. The same holds for the physical layer pdus `s` and `r`.

```
specification system[send,s,r,rec]:noexit
 behaviour
  Entity[send,s] |[s]| Medium[s,r] |[r]| Entity[r,rec]
 where
  process Entity[a,b]:noexit:=a;b;Entity[a,b] endproc
  process Medium[s,r]:noexit:=s;(r;Medium[s,r]□i;Medium[s,r]) endproc
endspec
```

Figure 3.(**b**) shows the interleaved expanded form of the LOTOS specification (**a**).

```
IT(dp0[send]|||2|||3, {<1,2>s;(dp0[send]|||4□i;2) , <3,4>r;(2|||rec;3)})
where   process dp0[send]:noexit:=send;dp0[send] endproc
```

The dashed lines reveals that its topology is closely related to its equivalent Petri net (**c**), output from the application of function **LPN** to behaviour (**b**). This Petri net has 5 transitions and 6 places, whereas the equivalent EFSM has 16 transitions and 8 states. □

Another important issue to be outlined is that, as in [ML89], it is necessary that every set of entry places P_0 to the subnets being composed by binary LOTOS operators holds

$\forall p \in P_0$, $\bullet p = \{\}$, for which it is necessary a one-level unfolding of such operands to remove such recursion. This exception is caused by the method of composition of subnets.

5 RELATED WORKS

An important result of this work is the reduction of premises to be fulfilled by a LOTOS specification in order to be translated into a Petri net. Let us review the list of problematic aspects and use it to contrast this approach with related works.

1. **Preservation of Parallelism.** [ML89] [GS90] and interleaved expansion do respect entirely the original parallelism. However the latter yields less parameterized structures and no impossible transitions since the iexpansion applies the necessary semantic rules to solve parameterization and compute *actual* events.
2. **Process instantiation. Recursion.** The translation of several cases of recursive process instantiations is the main limitation identified by [GS90] [ML89]. The problem is that recursion is translated despite the context in which the instantiation is done. Consequently, any case of recursion which may potentially cause divergence, although it actually creates a bounded number of processes, will be translated into: either an infinite net -like in a parameterized expansion-, or a token-unbounded net, which furthermore, can be incorrect (e.g. synchronization cases of unbounded cardinality).
   The interleaved expansion solves utterly this problem in finite cases due to the segregation of state subspaces for the detection of recursion as stated in section 3.5. Precedent approaches do not replace variables with values in the behaviours after each instantiation and synchronization. Consequently, previous translations cannot cope with dynamic bounded process composition (whether bounded by predicates or by synchronizations).

Example 51 The following specification models a FIFO queue of size 100 in a resource oriented style by composition of 100 processes.

```
specification buffer_n[ input, output ]:noexit
library Boolean, NaturalNumber, Data endlib
behaviour
 buffer[ input, output ]( 100 )
 where
  process buffer[ input, output ](n:nat):noexit:=
     [n equal 1]-> buffer1[input,output]
   []
     [not(n equal 1)]-> (hide m in buffer1[input,m] |[m]| buffer[m,output](n-1))
  where
   process buffer1[ input, output ]:noexit:=
    input  ?x:data; output !x; buffer1[ input, output ]
   endproc
  endproc
endspec
```

Precedent works are not capable of translating this behaviour because it does not keep static control constraints (unless we instantiate each process by hand). The interleaved

expansion would compose the 100 processes in parallel generating an expression without guards, relabeling and hiding easily convertible into a Petri net. □

3. **Gate relabeling.** [GS90] [ML89] do not support non-statically-solvable relabeling cases, quite a frequent case. For both transformations the relabeling function must be bijective, whereas it is just surjective in general.
   e.g. `P[a,a] where process P[a,b]:noexit:= a;stop||b;stop endproc` is not supported. [ML89] interprets the process instantiation with gate relabeling as the relabeling of the Petri net corresponding to the process definition, and halts the translation when a previously visited instantiation is detected. Therefore, it does not support properly recursion with relabeling
   The iexpanded form has no gate relabeling (it has been solved during the interleaved expansion), and hence there is not such a limitation.
4. **Data types.** Only [GS90] extends Petri nets with data types by means of a set of variables that makes up the context of the graph. Its treatment is static at translation-time: no computation of values exchanged at synchronizations is performed, nor substitution of exchanged data value in the subsequent behaviour. They are computed only when the Petri net is executed (at run-time). Hence, the precomputation of synchronization is limited to determine which events in the original specification can potentially synchronize, not the exchanged values nor actual events.
   LOTOS operators let, guard and value choice are translated into void transitions, which implies undoing and backtracking of such transitions. This can be avoided through the interleaved expansion since iexpanded forms with data types [Lar96] have all value choices and guards bound to actions in the format **choice** *vl*[] [*BE*]–> g[*BE'*]; B, what enables an scheme based on tokens with a value/variable environment attached plus conditioned transitions, or others [Rei85].
5. **Disabling.** There is not an element alike in Petri nets. Cited works do not extend Petri nets with elements able to model disabling composition, what obliges to develop every reachable state in the disabled behaviour. Since disabling is a mixed interleaving-choice operator, the consequences of this operation are (see example 52):
   - It is necessary to compute all the markings reachable by the subnet equivalent to the first argument, i.e. *computing states*, operation which we try to avoid by all means.
   - There is an important explosion in the number of transitions representing the same event (and of arcs), in the order of the number of states of the first operand multiplied by the number of first transitions of the second.
   - The Petri net loses legibility and conciseness.

   The only way to avoid these undesired effects is to extend Petri nets to model asymmetric relations. Nevertheless, we have not found a proper Petri net extension prepared to model disabling explicitly, although this extension is possible in events structures [Lan92]. Note that Petri nets with inhibitor arcs [SV95] do not handle the mentioned problems. ITcalculus includes disabling, what can be a reason to use this model. If the output of an ordinary net is mandatory, [ML89] is still the procedure to follow.
   Example 52 Let B = (`a; stop|||b; stop`)[> (`c; stop|||d; stop`). Converting both operands of the disabling into Petri nets figure 4.**(a)** is obtained.
   The resultant global net is depicted in figure 4.**(b)**. It has been necessary the computation of the four markings reachable by subnet `a; stop|||b; stop`. □

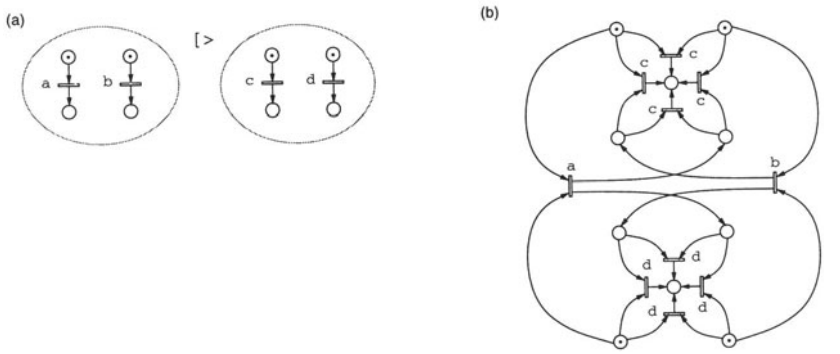


Figure 4 Petri nets composition with disabling

6 CONCLUSIONS AND FUTURE WORK

The transformation from LOTOS to Petri nets has practical interest as a low level base formalism for efficient reachability analysis, as well as to exploit the extensive theory developed for Petri nets and its tools. We have defined a mapping between an iexpanded form of a LOTOS behaviour expression and its equivalent Petri net representation. The work has also shown the advantages provided by the application of the interleaved expansion as an intermediate step in the translation from LOTOS to Petri nets over existing transformations. Namely, reduction of redundant elements, translation of specifications with bounded dynamic composition of processes and all cases of relabeling, and reduction of complexity.

Conversely, the close relationship between ITcalculus and petri nets can be applied to represent some Petri nets as expressions of a process algebra by extension of LOTOS. Furthermore, one of the advantages of this approach is that it enables the application of a wide range of state exploration techniques to LOTOS based on analysis of independence of events, techniques only suitable for Petri nets [Val88] and similar models [God95] which need as input explicit global events and the bundles of local states that enable them.

Further work should be devoted to device an extension of ITcalculus to represent specifications with infinite processes but bounded cardinality in synchronizations, which would be mapped to non binary Petri nets.

REFERENCES

[God95] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems . An Approach to the State Explosion Problem.* PhD thesis, Faculté de Sciences Appliquées, University of Liege, 1995.

[GS90] Hubert Garavel and Joseph Sifakis. Compilation and Verification of LOTOS Specifications. In Luigi Logrippo, Robert L. Probert, and Hasan Ural, editors, *Protocol Specification, Testing, and Verification X*, pages 359–376, Ottawa, Ontario (CA), June

1990. IFIP, Elsevier Science B.V. (North-Holland).

[ISO89] ISO. *Information Processing Systems – Open Systems Interconnection – LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour.* IS-8807. International Standards Organization, 1989. [published 15/feb/1989].

[Kar88] Günter Karjoth. Implementing Process Algebra Specifications by State Machines. In Sudhir Aggarwal and Krishan Sabnani, editors, *Protocol Specification, Testing, and Verification VIII*, pages 47–60, Atlantic City, New Jersey, USA, June 1988. IFIP, Elsevier Science B.V. (North-Holland).

[Kre94] H. Kremer. Derivation of efficient implementations from formal descriptions - issues, methods and conformance. In D. Hogrefe and S. Leue, editors, *Formal Description Techniques, VII*, pages 421–436, Berna (Switzerland), 1994. IFIP, Elsevier Science B.V. (North-Holland). Proceedings FORTE'94, 4–7 October, 1994.

[Lan92] R. Langerak. *Transformations and Semantics for LOTOS.* PhD thesis, Univ. of Twente, 1992.

[Lar96] David Larrabeiti. *Contribución al Análisis del Espacio de Estados de Especificaciones LOTOS.* PhD thesis, Technical University of Madrid, Spain, 1996.

[LDdlP+93] G. León, J. C. Dueñas, J. A. de la Puente, N. Zakhama, and A. Alonso. The IPTES Environment: Support for Incremental Heterogeneous and Distributed Prototyping. In *Real-Time Systems.* Kluwer Academic Press, may 1993.

[ML89] Saturnino Marchena and Gonzalo Leon. Transformation from LOTOS specs to Galileo Nets. In Ken J. Turner, editor, *Formal Description Techniques, I*, pages 217–230, Stirling, Scotland, UK, 1989. IFIP, North-Holland. Proc. FORTE'88, 1988.

[MS92] José A. Mañas and Joaquín Salvachúa. $\Lambda\beta$: A Virtual LOTOS Machine. In Gordon Rose and Ken Parker, editors, *Formal Description Techniques, IV*, Sydney (AU), 1992. IFIP, Elsevier Science B.V. (North-Holland). Proc.t FORTE'91, 19–22 November, 1991.

[Pav90] Santiago Pavón. *Contribución al Análisis y Transformación de Especificaciones LOTOS.* PhD thesis, Technical Univ. of Madrid, 1990.

[QLP93] J. Quemada, D. Larrabeiti, and S. Pavón. Compressed State Space Representation of LOTOS Specifications. In Ken J. Turner, editor, *Formal Description Techniques, VI*, pages 19 – 34, Boston, Massachussetts, EEUU, 1993. IFIP, North-Holland. Proceedings FORTE'93, 26–29 October, 1993.

[QPF89] Juan Quemada, Santiago Pavón, and Angel Fernández. State Exploration by Transformation with LOLA. In *Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, June 1989.

[Rei85] W. Reisig. *Petri Nets, an Introduction.* Springer, Berlin (DE), 1985.

[Rei87] W. Reisig. Place/Transition Systems. In *Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986.*, volume 254 of *LNCS*, pages 117–141. Springer-Verlag, Berlin (DE), 1987.

[Roz87] G. Rozenberg. Behaviour of Elementary Petri Nets. volume 254 of *LNCS*, pages 60–94. Springer-Verlag, Berlin (DE), 1987.

[SV95] Riccardo Sisto and Adriano Valenzano. Mapping Petri nets with Inhibitor Arcs onto Basic LOTOS Behaviour Expressions. *IEEE Transactions on Computers.*, 44(12):1361–1370, December 1995.

[Val88] Antti Valmari. *State Space Generation: Efficiency and Practicality.* PhD thesis, Tampere University of Technology, Tampere, Finland, 1988.