FIFO - NETS

Gérard Roucairol
Bull S.A.
68, route de Versailles, 78430 Louveciennes, France

ABSTRACT: This paper presents a survey of applications and theoretical properties of FIFO-nets, i.e. Place-Transition nets in which places behave as FIFO-queues rather than counters. In the first part, the adequacy of FIFO-nets for solving generic synchronization problems is shown and their impact on the fairness property of a concurrent system is discussed. The second part is devoted to the study of the computational power of this model as well as the characterization of some sub-classes for which some classical properties become decidable (liveness, boundedness). Finally, the notion of T-invariant for FIFO-nets is introduced.

Keywords: FIFO-nets, computational power, fairness, liveness, boundedness, invariants, process synchronization, parallelization of programs, serializability, resource allocation.

CONTENTS OF THE PAPER

1. BASIC DEFINITIONS AND NOTATIONS

2. APPLICATIONS OF FIFO-NETS
   2.1. The critical-section problem
   2.2. Parallelization of sequential flowcharts
   2.3. Serializability of iterated transactions

3. THEORETICAL PROPERTIES OF FIFO-NETS
   3.1. Computational power of FIFO-nets
   3.2. Free-choice FIFO-nets
   3.3. Monogeneous FIFO-nets
   3.4. About invariants of transitions

4. BIBLIOGRAPHY
INTRODUCTION

FIFO-nets are place-transition (Petri) nets in which places behave as FIFO-queues rather than counters. The main paradigm which illustrates the use of FIFO-nets consists of modeling the following specification of behaviour of a concurrent system: whenever an event precedes another one, then some action must take place before another action. Such a requirement which, as a matter of fact, concerns conflict resolution between concurrent transition firings, is very frequently encountered in many realistic systems. In chapter 2 of this paper several examples are discussed and the impact of the use of FIFO-nets on the fairness property of a system is pointed out. These examples include the control of maximal parallelism extracted from sequential programs, the optimal control of the serialization property among transactions accessing a common data base as well as resource allocation problems.

The FIFO principle extends the computational power of place-transition nets to that of a Turing machine, even if the number of different types of items which can occur in a queue (the queue alphabet) is restricted to 2. This increase of descriptive and computational power implies of course a lack of general algorithmic methods to check some properties of a net like boundedness or liveness. However two sub-classes of FIFO-nets are exhibited for which these properties are decidable.

The first sub-class generalizes the notion of free-choice in such a way the usual structural conditions for liveness of free-choice nets are still valuable. The second sub-class implies some regularity (rational) constraints over the possible configurations of the queues. These contraints allow the use of the classical procedure for checking boundedness. Of course these FIFO-nets increase the computational power of ordinary place-transition nets.

1. BASIC DEFINITIONS AND NOTATIONS

Before introducing the notion of FIFO-net, we define some notations we shall use along this paper.

Definition 1.1
Let $X$ be an alphabet; $X^*$ denotes the free monoid generated by $X$; $e$ is the empty word; $X^\omega$ is the set of infinite words over $X$. Let $u$ be in $X^* \setminus \{e\}$, $u^\omega$ is the infinite word obtained by catenating $u$ infinitely often with itself.
Let $x$ belonging to $X^* U X^W$. $Y$ being a non empty subset of $X$:

- $\text{proj}_Y(x)$ is the erasing homomorphism which suppresses from $x$ the symbols not in $Y$
- $u$ being an item of $X^*$, $u \prec x$ means $u$ is a prefix of $x$.

**Definition 1.2 (FIFO-net)**

A FIFO-net $N = (P, T, B, F, Q)$ is defined by:
- A finite set $P$ of places, also called queues,
- A finite set $T$ of transitions, disjoint from $P$,
- A finite queue alphabet $Q$ and two mappings $F : P \times T \rightarrow Q^*$ and $B : P \times T \rightarrow Q^*$
- Called respectively forward and backward incidence mappings.

Up to now, the only difference with a place-transition net relies on the fact that the incidence mappings take value in $Q^*$ instead of $\mathbb{N}$. The main change appears in the definition of a transition firing.

**Definition 1.3**

A marking $M$ is a mapping $M : P \rightarrow Q^*$

A transition $t$ is fireable in $M$, written $M (t) > 0$, iff for every place $p$ $B(p, t) \prec M(p)$.

For a marking $M$, we define the firing of a transition $t$, written $M (t) > M'$, iff $M(t) > 0$ and for every place $p$ the following equation between words holds:

$$B(p, t) M'(p) = M(p) F(p, t)$$

In other words the firing of a transition $t$ removes $B(p, t)$ from the head of $M(p)$ and appends $F(p, t)$ to the end of the resulting word.

**Definition 1.4**

A FIFO-net $N$ together with an initial marking $M_0$ is also called a FIFO-net and is denoted $(N, M_0)$.

As usual the firing of a transition can be extended to the firing of a sequence of transitions and we note $FS(N, M_0)$ the set of firing sequences of this net. The firing of a sequence $u$ of transitions from a marking $M$ to a marking $M'$ is written $M (u) > M'$.

The set of markings reachable from $M_0$ will be denoted by $\text{Acc}(N, M_0)$.

The graphic representation of a FIFO-net follows the one of a regular P/T-net, except for the marking of the places and the labels of the edges which become words over the queue alphabet $Q$. 
2. APPLICATIONS OF FIFO-NETS

Three applications are successively described.
We start with a very classical problem allowing the reader to be more familiar with the use of FIFO-nets and which points out the interest of these nets in order to obtain a fair solution to synchronization problems.
(In ROUCAIROL [15], several classical synchronization problems are solved using a programming view of FIFO-nets).

2.1. The critical-section problem

Let us consider two processes $P_a$ and $P_b$ mutually exclusive for the access to their own, so-called critical section.

Using ordinary Petri-nets, the classical solution is:

Using a FIFO-net a possible solution is:
In that solution, firing transition \( r_a \) (resp. \( r_b \)) can be interpreted as the deposit of a request to enter critical section \( CS_a \) (resp. \( CS_b \)).

It is remarkable to see that just looking at the current marking at the place \( M \), all information can be given about the state of the system of processes, i.e. the current marking of the remaining places.

Let us compare these two solutions from the point of view of their infinite behaviours.

In the net of figure 2.1 \( x = (t_1t_2)^\omega \) is a possible infinite firing sequence. We can remark that along this sequence \( t_3 \) becomes alternatively fireable and not fireable. It is intrinsic to the solution.

Let us consider now the sequence \( y = (r_at_1t_2)^\omega \) of the net in figure 2.2. Along this sequence \( r_b \) remains always fireable and there is no reason a priori for not firing it.

If we think in terms of implementation of these processes on a single processor machine, the minimal assumption we can make about a scheduler of these two processes is that a process does not remain always in a ready state without being executed. This assumption is also automatically satisfied if each process is supported by its own physical processor.

The assumption we have described corresponds to the notion of finite delay property in [11]. This is a kind of minimal property to impose to infinite sequences in order to represent realistic infinite concurrent behaviours.
A similar notion appears also in the definition of "productive occurrence rule" introduced by R. VALK in [17].

Definition 2.1 (KARP and MILLER [11])
Let $x$ be an infinite sequence of a net $(N, M_0)$ i.e. a sequence whose every prefix is in $FS(N, M_0)$:
$x$ satisfies the finite delay property iff $\forall t \in T, \forall u \preceq x, \exists v \in T^*, s.t. uv \preceq x$ and $(uvt \preceq x$ or $uvt \notin FS(N, M_0))$.

Coming back to the previous examples we remark that $(t_1 t_2)^u$ satisfies the finite delay property in the first net but not $(r a_1 t_2)^u$ in the second. We can also verify that every infinite sequence of the second net which satisfies the finite delay property contains an infinite number of transitions of both $P_a$ and $P_b$. This means that this net represents a fair solution of the critical section problem with respect to the two processes and corresponds to the intuitive role a FIFO-queue can play in conflict arbitration.

As a matter of fact, the first net models a Dijkstra's semaphore. But nothing is represented in that net about the management of the process queue of the semaphore, so there is no reason a priori for this solution to be fair.

Finally let us remark that fairness in the second solution can be proved using the fact that the corresponding net is persistent i.e. every choice between transition firings is reversible.

Definition 2.2.
A net $(N, M_0)$ is persistent iff $\forall M \in Acc(N, M_0)$
$\forall t, t' \in T \quad M(t) and M(t') \implies M(tt')$ (and $M(t't)$).

Generally, Petri nets are not persistent (except for marked graphs). Using FIFO-nets conflicts can be modelled with persistent nets (or sub-nets). In many cases that property implies that a live net is also fair with respect to every transition. It is still an open problem to characterize these cases.

2.2. Parallelization of flowchart programs

In [12] R. KELLER has pointed out the use of a notion of queue-automaton in order to control the maximal concurrency extracted from sequential flowchart programs. This notion of queue-automaton can be very elegantly expressed in terms of FIFO-nets. We describe the parallelization procedure by an example.
Let us consider the following program which computes the quotient of the integer division of A by B.

\[ Q := 0 \text{ (a)} ; \]
\[ \text{while } A \geq B \text{ (p) do } Q := Q + 1 \text{ (b)} ; A := A - B \text{ (c) od ;} \]
\[ \text{write } (Q) \text{ (d)} \]

(we shall use the letters between brackets in order to refer to the operations in the program).

This program is modeled by a sequential schema which expresses on one hand the sequential control flow of operations and which retains on the other hand from the operations themselves a conflicting use of shared variables.

The sequential control flow is described by a finite automaton:

(P₁ and P₂ represent the two alternatives of the test operation p).

To this control flow is associated a symmetric "conflict" relation \( R \) deduced from the fact that two operations have in common either an output variable or a variable which is both an input variable for one operation and an output variable for the other operation.

For the previous example \( R \) is the symmetric closure of the relation

\[ R = \{ (a, b), (a, d), (p, c), (b, d) \} \]

The parallelization procedure now proceeds in three steps in order to allow concurrent execution of non conflicting operations while retaining from the sequential schema the basic control issued from test operations.

**Step 1** (control flow of test operations)
The sequential control flow is reduced to the control flow of test-operation only; this leads to obtain a kind of state-machine representing the reduced control flow.
Step 2 (conflict arbitration)
To each remaining operation in the schema is associated a corresponding transition. These transitions as well as test operations have input places defined as follows:

- for every pair \((a, b)\) in \(\bar{R}\), \(p_{ab}\) is an input place of only both transitions \(a\) and \(b\) such that

\[ B(p_{ab}, a) = 'a' \quad \text{and} \quad B(p_{ab}, b) = 'b' \]

if \(b\) is a test-operation with two alternatives \(b_1\) and \(b_2\) then we have

\[ B(p_{ab}, b_1) = B(p_{ab}, b_2) = 'b' \quad \text{and} \quad B(p_{ab}, a) = 'a' \]

- if an operation \(c\) does not appear in any pair of \(\bar{R}\), then the corresponding transition has an unique input place \(p_{cc}\) s.t. \(B(p_{cc}, c) = 'c'\).

From the previous example, we obtain:

![Diagram](image-url)
Step 3 (control induced by test operations)
In this step it is explained how the places are filled by the transitions as well as the construction of the initial marking.

Let us consider the sequential control flow of operations. This control flow is a composition of three linear sequences of operations called segments:

\[ s_0 = ap, \quad s_1 = bcp, \quad s_2 = d \]

\( s_0 \) is called the initial segment, and we can remark that the execution of operations in \( s_1 \) (resp. \( s_2 \)) is governed by \( p_1 \) (resp. \( p_2 \)).

It will be the responsibility of the transitions corresponding to the alternatives of a test-operation, to fill the input places of the other transitions.

Let \( P_{ab} \) be a place, \( p_i \) be an alternative of a test-operation and \( s_i \) the segment governed by \( p_i \), then we have:

\[ F (P_{ab}, p_i) = \text{proj} (s_i) \]

\( \{a, b\} \)

The initial marking of \( P_{ab} \) being \( \text{proj} (s_0) \)

\( \{a, b\} \)

We obtain finally from the previous example the following concurrent schema:

![Diagram](fig.2.4)
Remark that for this example the number of reachable markings is infinite, even though the number of states of the initial schema was finite: the sequence of operations \( pc \) can take an unbounded advance over the operation \( b \).

Let us examine now the equivalence which exists between the initial schema and the one we have built. Due to the construction, the equivalence is such that for one behaviour of one schema there exists a behaviour of the other schema such that the relative ordering of conflicting operations is the same in both behaviours.

Let us state this more formally:

- a behaviour of the sequential schema is either a finite word accepted by terminal state \( q_2 \) in the example) or an infinite word whose every prefix is accepted by the control flow automaton;

- a behaviour of the concurrent schema is either a complete firing sequence (i.e. it cannot be extended into another firing sequence) or an infinite firing sequence (i.e. a sequence whose every prefix is a firing sequence) satisfying the finite delay property (definition 2.1).

Then the equivalence between behaviours we consider is:

**Definition 2.3**

Let \( x \) and \( y \) be two behaviours. \( x \) is said equivalent to \( y \) and we write \( x \equiv y \) iff

- for every symbol \( a \):
  
  \[
  \text{proj}(x) = \text{proj}(y) \quad \text{(identical occurrences of symbols)}
  \]

  \[
  \{ a \} \quad \{ a \}
  \]

- for every pair \( (a, b) \) in the relation \( R \):
  
  \[
  \text{proj}(x) = \text{proj}(y) \quad \text{(identical ordering of conflicting symbols)}
  \]

  \[
  \{ a, b \} \quad \{ a, b \}
  \]

A semantical interpretation of this equivalence has been given by Keller [12]: if two behaviours are equivalent, then the sequence of values taken by each variable of the program is the same in both behaviours.

It can also be proved that the concurrent schema we have described is maximally concurrent in the following sense: any word equivalent to a behaviour of the sequential schema is a behaviour of the concurrent schema.
Moreover, as a corollary of a result of KELLER, it can be shown that no place-transition net or even no counting automaton can control the amount of concurrency we have exhibited especially when there are imbedded loops in the sequential schema.

Finally, let us indicate to the interested reader that ROUCAIROL in [16] has considered parallelization of flowcharts under a weaker definition of equivalence for which a slight extension of the notion of FIFO-queue is necessary.

2.3. Serializability of iterated transactions

The serializability problem is a synchronization problem which has been mainly studied in the framework of concurrent accesses to a Data Base [2]. Being given a so-called consistency predicate over the content of a Data Base and a set of transactions — i.e. a finite sequence of operations — each one preserving individually the consistency predicate, the serializability problem consists in synchronizing the transactions in order to allow concurrent behaviours which are equivalent to some serial composition of the transactions. Hence, these behaviours preserve also the consistency predicate and are called correct behaviours.

Let us remark that this problem represents an instance of a fundamental phenomenon in the control of concurrent systems which turns to be the achievement of the global correctness of a system of concurrent processes being supposed the individual correctness of each of its component.

In [8], [9], [10] FLE and ROUCAIROL have characterized, in terms of language theory, a generalized version of the serializability problem for transactions which can be infinitely often repeated on more generally for iterative programs as might behave for instance preexisting service processes in an operating system. Their results show that serializability can be controlled by a finite automaton and that resource allocation problems can be modelled as a serializability problem. However, infinite behaviours allowed by this automaton are not necessarily fair i.e. not every transaction or process is repeated infinitely often.

In [7], FLE and ROUCAIROL have pointed out a synchronization algorithm valuable for iterated transactions which guaranties fairness of infinite behaviours. This algorithm is based upon the use of FIFO-nets.

We are going to describe this algorithm with an example.
Let us consider the two following transactions, each one preserving individually the predicate "A = B".

\[ T_1: A = A \times 2 \ (a); \ B = B \times 2 \ (b) \quad (T_1 = ab) \]

\[ T_2: A = A + 10 \ (c); \ B = B + 10 \ (d) \quad (T_2 = cd) \]

The iterated behaviour of each transaction is modelled by the following net:

\[ \text{(The numbers 1 and 2 represent the characters '1' and '2' and not the corresponding values).} \]

Let us call a behaviour \( x \) of the previous transaction system, an infinite firing sequence such that for \( i \in \{1, 2\} \) \( \text{proj}(x) \in T_i^{\infty} \cup \{T_i^\omega\} \) (any occurrence of transaction in \( x \) is completed in \( x \)) where \( A_i \) is the alphabet of the operations occurring in the transaction \( T_i \). If we consider a prefix \( y \) of a behaviour such that each occurrence of transaction is completed in that prefix, the sequence of operations designated in \( y \) does not lead necessarily to values in variables \( A \) and \( B \) such that \( A = B \).

Consider for instance the prefix \( y = acdb \).

As we have already said we are going to consider correct behaviours, i.e. behaviours equivalent to serial behaviours, a serial behaviour being in our case an item of the set \( \{T_1, T_2\}^\omega \)

It is remarkable to observe that the equivalence which is generally used in the literature on serializability is exactly the one we have introduced in the previous section.

For the example we consider the conflict relation is then the symmetric closure of the relation:

\[ \overline{R} = \{(a, c), (b, d)\} \]
One can verify that the prefix \( y = acbd \) of a behaviour is equivalent to the prefix \( abcd \) of a serial behaviour; the operations performed in \( y \) lead to a situation where "\( A = B \)".

Now we are going to build a control of the two previous nets allowing only correct and fair behaviours. The procedure is somewhat similar to the one of the previous section.

For every pair in the \( \mathcal{R} \) relation we build a place which is an input place of the transitions whose names appear in the pair (the edge leading to a transition being labelled by the index of the transaction to which the transition belongs).

![Diagram of a control net]

Then to each transaction is attached a controller:

A single loop which appends simultaneously to every place of the transaction, previously created, the index of the transaction.

![Diagram of a controller]

Let us call \( A \) the alphabet of the operations performed by the transactions \( A = A_1 \cup A_2 \).
The following completeness and soundness results have been shown:

- $\text{proj}_A(\text{FS}(N,H_0))$ is exactly the set of prefixes of correct behaviours.
- Let us call FDB the set of infinite sequences satisfying the finite delay property.
  - FS $(N, H_0)$ is exactly the set of prefixes of FDB
  - For any item $x$ of FDB, for every transaction $T_i$
    \[ \text{Proj}(x) = T_i^\omega : \text{fairness} \]
    \[ A_i \]

One can remark that the synchronizing places between the two transactions are unbounded. As a matter of fact it has been shown that in order to achieve completeness of the previous construction, the controllers must be independent of the evolution of the transactions. However some particular cases have been identified in [7] for which the evolution of the controller can be synchronized with the evolution of the transactions in order to obtain only bounded places. (Notice that these cases include the case where only two transactions are considered).

We have said that serializability problem can model resource allocation problem. As an exercise the reader is invited to give a fair solution to the very classical "dining philosophers" problem.

The transactions to be considered are:
for each philosopher $i$:

$$(\text{think})_i ; (\text{takefork}_i)_i ; (\text{takefork}_{i+1})_i ; (\text{eat})_i ; (\text{releasefork}_i)_i ; (\text{releasefork}_{i+1})_i$$

where operation $(\text{takefork}_{i+1})_i$ conflicts with operation $(\text{releasefork}_{i+1})_i$ and operation $(\text{takefork})_i$ conflicts with operation $(\text{releasefork}_{i-1})_i$ (additions and subtractions are supposed to be mod $n$).

3. THEORETICAL PROPERTIES OF FIFO-NETS

In the first section of this chapter we state that FIFO-nets have the computational power of Turing machines. Of course the price to pay to this degree of generality is the undecidability of classical properties for Petri-nets, like liveness or boundedness. However we point out two classes of FIFO-nets for which these properties can be decided. Further work remain to do to characterize other classes of interest from the point of view of decidability results
and possibilities of applications. A good candidate seems to be the class of nets obtained by the parallelization procedure we have explained in the previous sections.

The application of the classical invariant technique is faced to the problem of non commutativity of the operations performed on the FIFO-queues. However significant invariant properties can be found by considering the content of a place as a bag of letters instead of a word. See the lecture of MEMMI and VAUTHERIN in this course. Nevertheless, concerning the invariant of transitions (T-invariant) we point out a structural characterization of cyclic firing sequences.

Finally let us notice that a software tool exists in order to analyse the properties of a system of concurrent processes communicating by FIFO places (BEHM [1]).

3.1. Computational power of FIFO-nets

FIFO-nets extend strictly the computational power of Petri nets and reach the computational power of Turing machine. As a matter of fact it is enough that the queue alphabet contains two letters and edges are labelled by at most one letter, to reach the computational power of a Turing machine.

Let us call alphabetical, a FIFO-net whose edges are labelled by one letter at most, then we have :

Theorem 3.1
Alphabetical FIFO-nets with a queue alphabet Q, such that $|Q| \geq 1$, have the computational power of Turing machines.

A constructive proof of this result has been given by MEMMI [13], another proof based upon language theory has been found by FINKEL [6]. These authors have also shown that FIFO-nets can be simulated by alphabetical ones up to an homomorphism.

Definition 3.1
A labelled FIFO-net is a pair $(N, Mo, h)$ such that $(N, Mo)$ is a FIFO-net and and $h$ a labelling function of the transitions $h : T \rightarrow X'U\{e\}$ where $X$ is a finite alphabet. $h$ is naturally extended to words in order to define the language of a labelled net :

$L ((N, Mo), h) = \{ h(x)/x \in FS (N, Mo) \}$. 
Theorem 3.2
For every labelled FIFO-net \(((N, M_0), h)\) there exists an alphabetical labelled FIFO-net \(((N', M', h')\) where

- \(Q \subseteq Q', \ P \subseteq P', \ T \subseteq T'\)
- \(\forall t \in T \quad h(t) = h'(t)\)
- \(\forall t \notin T \quad h'(t) = e\)

and such that \(L((N, M_0), h) = L ((N', M'), h')\)

3.2. Free choice FIFO-nets

The following definition extends the initial definition of free-choice Petri-nets.

Définition 3.2
A FIFO-net is free choice iff for every place \(p\) the following holds:
- for every transition \(t \mid B \ (p, t) \mid \leq 1\) every output edge of \(p\) is labelled by at most one letter.
- \(\{a \in Q/ \exists \ t \in T, \ \text{proj} \ (B(p,t)) \neq e\} = \{a \in Q/ \exists \ t \in T, \ \text{proj} \ (F(p,t)) \neq e\} \backslash \{a\}\)

the set of letters which can be added to \(p\) is the set of letters which can be extracted from \(p\).
- for any pair of transitions sharing \(p\) as an input place, \(p\) is the only input place.

Like free-choice Petri-nets, free-choice FIFO-nets have nice structural properties from the point of view of liveness. As a matter of fact the FIFO principle does not play a role for the liveness property.

Let us call the associated coloured net of a FIFO-net, a net with an identical structure but with the marking of a place considered as a bag of letters instead of a word; the firing of a transition requiring only the presence of a necessary number of occurrences of letters in a place and not some specific ordered list of these occurrences.
Then we have (FINKEL [6]) :

**Theorem 3.3**
A free-choice FIFO-net is alive iff its associated coloured net is alive.

It is interesting to remark that the associated coloured net of a free-choice FIFO-net can be unfolded into an ordinary free-choice Petri-net (a place being split into as many places as there are different letters which may enter in the initial place). It can be shown that this new net is also equivalent to the coloured net from the point of view of liveness. FINKEL [6]. Hence the usual Commoner's condition concerning traps and deadlocks can be applied in order to decide about the liveness of a free-choice FIFO-net.

From the reasoning above, we could think that free-choice FIFO-nets have nothing more than the computational power of usual free-choice nets. This is not the case.

Let us call \( L(\mathcal{N}) \) the language of a class \( \mathcal{N} \) of nets - i.e. the set of sets of firing sequences of nets belonging to this class.

FINKEL [6] has shown.

**Theorem 3.4**

\( L(\text{free-choice Petri nets}) \not\subseteq L(\text{free-choice FIFO-nets}) \not\subseteq \text{recursively enumerable sets.} \)

(This is due to the fact that the anti-DYCK language (VAUQUELIN [18]) is a free-choice FIFO-net language and not a Petri-net language).

Another kind of nice features of free-choice FIFO-nets comes from the fact that properties are preserved when increasing the initial marking.

First of all let us define an order relation between words and markings.

**Définition 3.3**

Let \( u \) and \( v \) be two words of \( Q^* \). We say that \( u \) divides \( v \), \( u/v \), iff there exists a word \( w \) of \( Q^* \) such that:

\[
 v = w_1 u_1 w_2 \ldots w_n u_n w_n + 1 \quad \text{with} \quad w = w_1 w_2 \ldots w_n + 1 \quad \text{and} \quad u = u_1 u_2 \ldots u_n.
\]

Then we say that a marking \( M \) divides a marking \( M' \), \( M/M' \), iff for every place \( p \) \( M(p)/M'(p) \).
Theorem 3.5
Let \((N,M)\) be a free-choice FIFO net and \(M'\) a marking such that:
\(M/M'\) and the set of letters occurring in \(M'\) (p) is the same as the set of letters occurring in \(M(p)\) for every place p.

1) \((N,M)\) unbounded \(\implies\) \((N,M')\) unbounded
2) FS \((N,M)\) infinite \(\implies\) FS \((N,M')\) infinite
3) \((N,M)\) is alive \(\implies\) \((N,M')\) is alive

3.3. Monogeneous FIFO-nets

This category of nets has been characterized by FINKEL [4] in order to define a class of FIFO-nets containing at least a class isomorphic to Petri-nets and for which the boundedness problem is decidable using the classical KARP and MILLER procedure [11].

Definition 3.4
A language \(L \subseteq Q^*\) is monogeneous if there exist two words u and v of \(Q^*\) such that \(L\) is included in the set of prefixes of \(uv^*\).

A language \(L\) is said semi-monogeneous (shortly s-monogeneous) if it is a finite union of monogeneous languages.

Now we say that a FIFO-net is (s-) monogeneous if for every place the set of sequences of words which can be added to a place whenever firing sequences of transitions is a(s-) monogeneous language. In other words:

Definition 3.5
For each place p we consider the mapping \(I_p : T \rightarrow Q^*\), such that \(I_p(t) = F(p,t)\). This mapping is naturally extended to words and we say that a FIFO-net is (s-) monogeneous iff for every place p \(I_p(\text{FS}(N,M_0))\) is (s-) monogeneous. \(I_p(\text{FS}(N,M_0))\) is called the input language of p.

The following net models an example of communication protocol between two processes, given by VUONG and COWAN [19].

The processes exchange messages via the FIFO-queues f12 and f21.

This net is a monogeneous net: \(I_f12(\text{FS}(N,M_0))\) is the set of prefixes of \((12)^*\).
In order to decide if a FIFO-net is bounded, using the KARP and MILLER procedure, a "good" order relation between markings has to be defined.

**Definition 3.6**

Let $N = (P,T,Q,F,B)$ be a FIFO-net and $M$, $M'$ be two markings. We say that $M \ll M'$ iff for every place $p$, for every firing sequence $x$ of $FS(N,M)$

$$M(p) \leq_{p} M'(p) \leq_{p} M(p)$$

For $(s,-)$ monogeneous nets this relation can be proved to be an order relation such that every infinite sequence of markings contain a non decreasing infinite subsequence. Moreover we have the classical monotony property:

$$M \ll M' \implies Acc(N,M) \subseteq Acc(N,M')$$

Finally this order relation can be shown to be decidable.

Then we obtain:

**Theorem 3.6**

The boundedness problem is decidable for $(s,-)$ monogeneous FIFO-nets using the KARP and MILLER procedure.

Let us consider the following example:
For this net $\text{Ip}_3(\text{FS}(N,M_0))$ is the set of prefixes of $(ab)^*$

We have $M_0 = (1,e,e) (t_1 t_2 > (1,e,ab) = M_1$

Hence $M_0 \ll M_1$ and $p_3$ is unbounded. We have also $M_0 = (1,e,e)(t_1 t_3 t_2)$

$(1,e,b) = M_2 (t_1 t_2) (1,e,bab) = M_3$

$M_2 \ll M_3$

While building the reachability tree for the KARP and MILLER procedure, $M_1$ will be replaced by $M'_1 = (1,e,(ab)^\omega)$ and $M_3$ by $M'_3 = (1,e,(ab)^\omega)$. Using this convention, the construction of this reachability tree is finite providing the fact $x^\omega \cdot u = x^\omega$ and $x^\omega = xx^\omega$ for any finite words $x$ and $u$.

From the previous example the beginning of the construction of the reachability tree is:

As an exercise the reader is asked to check whether places $f12$ and $f21$ in the protocol example are bounded or not.
Like for Petri-nets the boundedness problem is decidable for \((s-)\) monogeneous nets. More generally it can be shown that every problem decidable for Petri-nets is also decidable for \((s-)\) monogeneous nets. This can be proved by pointing out that the set of firing sequences of a \((s-)\) monogeneous set is the intersection of a regular language with the set of firing sequences of a usual Petri-net.

The regular language is obtained as follows:

the reachability tree is converted into a reachability graph by merging into only one node two identical markings on the same branch of the tree; then we obtain the graph of a finite automaton whose accepted prefix language is designated by \(RL(N,Mo)\).

The usual Petri-net is obtained as follows:

the graph of the net is the one of the FIFO-net, but edges are labelled by the number of letters which are added or removed from a place of the original FIFO-net; the initial marking of a place is just the number of letters existing in the initial marking of the same place of FIFO-net; let us call \((\bar{N},\bar{M}_0)\) such a net.

Then we obtain:

Theorem 3.7

For a \((s-)\) monogeneous net \((N,Mo)\):

\[FS(N,Mo) = FS(\bar{N},\bar{M}_0) \cap RL(N,Mo)\]

Let us remark also that \((s-)\) monogeneous nets are strictly more powerful than Petri-nets without transitions identically labelled. As a matter of fact it is clear that \(L(\text{Petri-nets}) \subseteq L(\text{monogeneous FIFO-nets})\).

But the set of prefixes of \((abba)^*\) is not a language of a Petri-net with an injective labelling of the transitions and it is a language of a monogeneous FIFO-net.

In order to check if a FIFO-net is \((s-)\) monogeneous it is necessary to express the input language of each place. There is no general procedure to do it. But in [6] the reader will find several necessary or sufficient conditions for a net to be \((s-)\) monogeneous.
3.4. About invariant of transitions

Invariants of transitions (T-invariants) characterize cyclic firing sequences i.e. sequences which reproduce their source marking.

In order to characterize such sequences for FIFO-nets we first extend the firing equation of definition 1.3 to sequences of transitions.

Let \( s = t_1 \ldots t_m \) be a sequence of transitions and \( M, M' \) be two markings such that \( M(s) > M' \).

Then it is easy to show that for every place \( p \) the following equation holds:

\[
B(p, t_1) \ldots B(p, t_m) M'(p) = M(p) F(p, t_1) \ldots F(p, t_m).
\]

For cyclic firing sequences we then obtain an equation of the form:

\[
xy = yz
\]

It is noteworthy to observe that this equation over the free-monoid \( Q^* \) has a solution which is: there exist two words \( w \) and \( w' \) such that

\[
x = ww', \quad z = w'w \quad \text{and} \quad y \in w(w'w)^*
\]

This solution links together the word which is input in a place and the word which is output from a place as well as the marking of this place in a very particular way. Exploitation of this result remains to be done.

BIBLIOGRAPHY

[1] P. BEHM
