TOWARDS MULTIPROCESSOR IMPLEMENTATION OF REAL-TIME, DATA-FLOW PROGRAMS.

C. FIGUEIRA, T. GAUTIER, B. LE GOFF, P. LE GUERNIC

IRISA-INRIA RENNES
Campus de Beaulieu, 35042 RENNES CEDEX FRANCE

ABSTRACT.

We introduce the synchronous approach of the data-flow language SIGNAL, to specify and implement real-time applications. Based on a model of the timing in the finite field \( \mathbb{Z}/3\mathbb{Z} \), a clock calculus guarantees the determinism of programs. These clock informations, together with the data dependences, are condensed in a graph representing completely the application. We propose an algorithm of collapsing into granules of this graph targeting the mapping onto multiprocessor machines.

1- INTRODUCTION.

SIGNAL is a programming language designed in the framework of a workstation for real-time signal processing\(^1\). In a real-time context parallelism is essential. Unfortunately, it has been shown that most existing parallel languages (/1/, /2/) are fundamentally non deterministic. However, in the main areas of real-time programming - control of industrial processes, signal and image processing in telecommunication, speech processing, biomedical applications, radar and sonar systems, ... - non determinism must be considered as harmful. LUSTRE (/4/), ESTEREL (/5/) and SIGNAL (/6/, /7/) are parallel languages that prevents from undesired non determinism thanks to the concept of synchrony, i.e. the complete knowledge of the timing of events. The former two are programming languages of pure reactive systems (/5/); they are exclusively deterministic. SIGNAL accepts the description of non deterministic intermediate sub-processes; its compiler is able to find the contexts in which a process would be deterministic. By using data-flow and synchrony properties, SIGNAL analyses, on the one hand, the coherence and the determinism of programs and exploits, on the other hand, the parallelism of the programs to implement them on multiprocessor machines.

A SIGNAL program is a set of definitions of infinite sequences of values, where the index may be considered as logical time index. These definitions are written through operators of two kinds, synchronous ones and asynchronous ones. In the second section we describe these fundamental operators, their temporal properties and their representation in the finite field \( \mathbb{Z}/3\mathbb{Z} \). The third section deals with the timing and dependences correctness of programs; this analysis is achieved using both the model in \( \mathbb{Z}/3\mathbb{Z} \) and the Conditional Dependences Graph. It is this graph, containing all the information about the data dependences and synchronization, the object to be implemented. Finally, we show in section four an algorithm that reduces the graph, in order to simplify the mapping task.

\(^1\)This work is supported by CNET, FRANCE.
2 - REAL-TIME PROGRAMS.

2.1 - Signals.

In order to define a signal, we define the sequence of its values, and its timing with respect to other signals.

2.1.1 - Extension to sequences.

As LUCID /3/, SIGNAL handles sequences: for example \( x := a + b \) will have to be read as \( \forall t : x_t = a_t + b_t \) where \( t \) may be considered as the index of a logical time, and this for all arithmetic and boolean functions, so called synchronous and immediate functions.

However, SIGNAL refuses the access to future values, only past values can be explicitly used via the delay operator: \( x := y \; \$ \; init \; y_0 \) for \( \forall t > 0 : x_t = y_{t-1} \) and \( x_0 = y_0 \).

2.1.2 - Temporal constraints.

The presence and the absence of a signal are defined relatively to other ones. For example, a signal is always present relatively to itself. An additional example is the presence of a signal relatively to any other appearing in the same expression of a synchronous and immediate function: they are both simultaneously present. It is the same for the delay.

We need a notion allowing to express the temporal constraints between signals. In a given context, i.e. a set of signals, we can define the clock of a signal as being its state - present or absent - every time another signal is present.

2.2 - Equations of signal definitions.

In order to obtain a finer analysis of temporal constraints we consider these constraints but also the boolean expressions of the program. So as to do this analysis we encode clocks and boolean signals in the finite commutative field \( \mathbb{Z}/3\mathbb{Z} \) - \((0,-1,1),+,\times)\) - : 0 denotes the state absent of a signal, 1 denotes both the state present of a signal and, when it is boolean, the fact that it has the value TRUE, -1 denotes, for a boolean signal, the fact that it has the value FALSE.

To any signal \( x \), we can associate a variable, at which we give the same name, in \( \mathbb{Z}/3\mathbb{Z} \), representing the coding of the signal when it is boolean, and whose square value \( -x^2 \) represents the coding of its clock in any case. Then polynomials in \( \mathbb{Z}/3\mathbb{Z} \) will express temporal constraints and boolean expressions. So, to any equation of signal definition we associate an equation in \( \mathbb{Z}/3\mathbb{Z} \).
2.2.1 - Synchronous equations.

Consider a synchronous immediate and non boolean function \( x := f(\ldots, x_i, \ldots) \). As we said above, the signals \( x, \ldots, x_i, \ldots \) are simultaneously present, which is expressed by the following equations in \( \mathbb{Z}/3\mathbb{Z} \) :

\[
x^2 = \ldots = x_i^2 = \ldots
\]

For the synchronous immediate and boolean functions, the associated equations will have to express both synchronization and value constraints :

\[
y := \textit{not} \ c \quad \text{is encoded by} \quad y = - c
\]
\[
y := \textit{a or b} \quad \text{is encoded by} \quad y = ab(1 - (a + b + ab))
\]
\[
y := \textit{a and b} \quad \text{is encoded by} \quad y = ab(ab - (a + b + 1))
\]

In this paper, we describe a static representation of the temporal constraints, i.e. we do not consider the dynamic transfer of values in delay, then in both cases, boolean and not boolean, the delay

\[
y := x$\textit{ init} x_0 \quad \text{is encoded by} \quad y^2 = x^2
\]

For a given signal \( x \), it is interesting to permit an explicit handling of its clock. The operator \( \textit{event} - \text{syntax} : b := \textit{event} x \) - provides the always TRUE boolean signal \( b \); it is present only when \( x \) is present.

\[
b := \textit{event} x \quad \text{is encoded by} \quad b = x^2
\]

2.2.2 - Asynchronous equations.

The previous operators allow the handling of synchronous signals and their clocks. In SIGNAL, we are able to generate a signal from asynchronous ones by \textit{extraction} or \textit{merge}. Depending on whether these operators apply to non boolean or boolean signals, only the temporal constraints, or both temporal and value constraints will be encoded.

\textbf{Extraction} : transmission of a signal according to the value of a boolean one.

For \( a, y \) non boolean and \( c \) boolean,

\[
y := \textit{a when} \ c \quad \text{is encoded by} \quad y^2 = a^2(-c - c^2)
\]

The output \( y \) takes the value of \( a \) when the boolean signal \( c \) is present and TRUE.

For \( a, y, c \) boolean,

\[
y := \textit{a when} \ c \quad \text{is encoded by} \quad y = a(-c - c^2)
\]
Exclusive merge: construction of a signal by merge of two signals and elimination of the values simultaneously present.

For $a, b, y$ non boolean,

$y := a \text{ exmerge } b$ is encoded by $y^2 = a^2 + b^2 + a^2b^2$

The output $y$ takes the value of $a$ when the boolean signal $c$ is present and TRUE.

For $a, b, y$ boolean,

$y := a \text{ exmerge } b$ is encoded by $y = a(1-b^2)+b(1-a^2)$

2.3 - Systems of equations.

2.3.1 - Composition.

Any such definition of a signal may be seen as the specification of a basic process. A new process is defined by the composition of smaller ones with the operator denoted $\mid$, in a block-diagram building style. Let us see through an example the use of this operator.

We want to program the simple DSP-filter $y_t = u_t + a \cdot y_{t-1}$. It is made up of three basic processes: an addition, a product and a delay (see table A).

<table>
<thead>
<tr>
<th>operation</th>
<th>signal program</th>
<th>block diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td>$y := u + x$</td>
<td><img src="image1" alt="Diagram" /></td>
</tr>
<tr>
<td>product</td>
<td>$x := a \cdot y$</td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td>delay</td>
<td>$z y := y $</td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Table A

The semantics of the operator $\mid$ is: one name for one signal, i.e. a name cannot represent two different signals, in the block-diagram, ports having the same name are connected (see table B).

<table>
<thead>
<tr>
<th>SIGNAL program</th>
<th>block diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mid { y := u + x \mid x := a \cdot y \mid z y := y $ }$</td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Table B

In this way we can define processes as systems of equations. Then to any process will be associated a system of equations in $\mathbb{Z} / 3\mathbb{Z}$ ($y^2 = u^2 = x^2 = a^2 = zy^2$ for the simple DSP-filter, where all the signals are synchronous).
2.3.2 - Two higher level operators.

Higher level macros are provided to the users of SIGNAL. For example, a merge with
priority - so called default, syntax : \( y := a \text{ default } b \) - which does not eliminate the values of \( a \) and
\( b \) simultaneously present but takes those of \( a \) and throws away those of \( b \). This operator is not
commutative. It may be programmed as follows :

\[
\begin{align*}
(l & \ d := a \ \text{merge} \ b & d^2 = a^2 + b^2 + a^2b^2 \\
(l & \ c := a \ \text{when event} \ b & c^2 = a^2b^2 \\
(l & \ y := d \ \text{merge} \ c & y^2 = d^2 + c^2 + d^2c^2 \\
) & \)
\]

As it will be described in the next section we calculate the expression of the clock of \( y \)
from this system : \( y^2 = a^2 + b^2 - a^2b^2 \).

Another example, very useful when programming, is the memorisation of the value of a
signal - denoted cell, syntax : \( x := a \ \text{cell} \ c \) where \( c \) is a boolean signal. The signal \( x \) is present
carrying the last value of the signal \( a \) when either \( a \) is present or \( c \) has the value TRUE. By using
the operator default and another easily derived operator, the synchronisation, - denoted synchro,
syntax : \( synchro \ u,v \) - which makes nothing but constraints \( u^2 = v^2 \), we may program cell as
follows :

\[
\begin{align*}
(l & \ x := a \ \text{default} \ zx & x^2 = a^2 + zx^2 - a^2z^2 \\
(l & \ zx := x \ \$ \text{init} \ x0 & zx^2 = x^2 \\
(l & \ synchro \ x,u & x^2 = u^2 \\
(l & \ t := c \ \text{when} \ c & t = - c - c^2 \\
(l & \ u := (\text{event} \ a) \ \text{default} \ t & u^2 = a^2 + t^2 - a^2t^2 \\
) & \)
\]

The expression of the clock of \( x \) is : \( x^2 = a^2 + (1 - a^2)(-c - c^2) \).

3 - TIME CORRECTNESS AND DEPENDENCES ANALYSIS.

3.1 - The clock calculus.

For a given process \( P \), two kinds of non boolean signals must be distinguished :
1 - those which are not defined in \( P \) (referred to as inputs)
2 - those which are defined in \( P \) (referred to as outputs)
and three kinds of boolean signals must be distinguished too:

1 - those which are not defined in P (referred to as inputs)
2 - those which are defined in P by a boolean function - function of boolean variables -
( referred to as outputs )
3 - those which are defined in P by a non boolean function - for example \( x := a > b \)
( referred to as free outputs for they are considered like that in the clock calculus ).

The clock calculus considers the equations of the system in \( \mathbb{Z}/3\mathbb{Z} \), as relations. The compiler solves this system so as to define all the involved clocks and boolean signals. It must associate to every clock an expression, function of one clock - so called the root clock - and of input and free output boolean signals. The root clock is the more frequent one; it is synthethised by the clock calculus so it may be an internal - i.e. not that of an input - clock.

During the resolution, if the calculus

i) exhibits some constraints on free output boolean values, without any reasoning on
non boolean values, the process P must be rejected.

ii) exhibits some constraints on the input boolean signals, P will be runnable only in
some contexts.

iii) has to introduce parameters to solve implicit functions, P is non deterministic,
which means that some clocks cannot be exactly defined, as in ii) P will be runnable only in some contexts.

Comment: In order to detect deadlocks an analysis of the data dependences graph must be
associated to the clock calculus.

3.2 - The Conditional Dependences Graph.

The graph is the second object - the first is the system of equations in \( \mathbb{Z}/3\mathbb{Z} \) - required to control the correctness of the process. A process is "runable" if for every signal and at any time it is possible to decide if the signal is present or not and when it is present it is possible to produce it. On the other hand, we must verify that we can evaluate the clock and signal expressions every time this is required. It is a matter of detecting effective circuits in a directed graph whose arcs express the data dependences between non boolean signals, the calculus dependences between clocks and boolean signals, and the pre-eminence of any clock on all the signals of this clock.

We build a directed and labelled graph \( G=(X,\Gamma,CLOCK) \). The set \( X \) of its vertices is the union of the set of signals of the process SIGNAL and of the set of clock classes calculated by the clock calculus - for example, from the SIGNAL expression \( (x := \text{event } y \mid y := a + b) \) the clock calculcs builds the class \( \{ x, y^2, a^2, b^2 \} \) for \( x = y^2 = a^2 = b^2 \), then \( x \) will be considered as a clock, not as a signal.
The set $\Gamma$ of its arcs is defined as follows.

1- DATA DEPENDENCES:

Let $f$ be a synchronous immediate and non boolean function - i.e. function of non boolean variables but $x$ can be boolean -

\[ x := f(x_1, \ldots, x_n) \quad \text{gives} \quad \forall i \in [1,n] \quad (x_i, x) \in \Gamma \]

Let $a, b, x$ be non boolean signals

\[ x := a \, \text{exmerge} \, b \quad \text{gives} \quad (a, x) \in \Gamma \quad \text{and} \quad (b, x) \in \Gamma \]

Let $a, x$ be non boolean signals and $c$ be a boolean signal

\[ x := a \, \text{when} \, c \quad \text{gives} \quad (a, x) \in \Gamma \]

2- CALCULUS DEPENDENCES:

Let $f$ be a definition of clock produced by the compiler as a result of the clock calculus

\[ x = f(x_1, \ldots, x_n) \quad \text{gives} \quad \forall i \in [1,n] \quad (x_i, x) \in \Gamma \]

3- CLOCK PRE-EMINENCE:

Let $x$ be a signal, we note $x^2$ its clock

\[ (x^2, x) \in \Gamma \]

At last we define the arc labelling $\text{CLOCK. CLOCK}$ associates the expression $x^2, y^2$ to every arc $(x, y)$ of $G$, where $x^2$ and $y^2$ are the clocks of $x$ and $y$. The product $x^2, y^2$ is nothing but the clock at which $x$ makes a real contribution to $y$. In other words the arc $(x, y)$ is only valid when $x^2, y^2 = 1$. In some sense we obtain a superposing of instantaneous graphs.

We call the graph $G$ the Conditional Dependences Graph.

DEADLOCK CRITERIUM.

Deadlocks will appear during runnings of a process $P$ if there are instantaneous circuits in the associated Conditional Dependences Graph $G$.

Let $(x_0, \ldots, x_n)$ be a circuit of $G$, it is a matter of proving the equality

\[ \Pi_{i=1}^n x_{i-1}^2 \cdot x_i^2 = 0 \]

Such circuits are not effective, i.e. they represent no instantaneous circuit. So they do not prohibit the running of $P$. 
4 - TOWARDS MULTIPROCESSOR IMPLEMENTATION.

3.1 - Granulation.

We want to define tasks groups, such that the whole group could be assigned to the same processor. Searching for groups is equivalent to searching for groups whose tasks can be executed in a sequential order without interruption. From now on, these groups will be seen as black boxes for we know that no deadlock will be caused by the construction of the groups. The main idea is that the partition must add no constraint: any path in the parts graph has an associated path in the graph. Hence, if there is a circuit in the paths graph, it has an associated circuit in the graph. Moreover this property is preserved by natural composition of partitioned graphs. This is very important for separated compilation.

4.1.1 - Granules definition.

Given a directed graph $G$, and a partition $P$ of $G$, we consider $G/P$ the following graph: the $G/P$ vertices are the classes of $P$. Let $A$ and $B$ be two classes of $P$, $(A,B)$ is an arc in $G/P$ iff we can find an arc $(a,b)$ of $G$ such us $(a,b)$ is in $A \times B$. For short, $G/P$ is called collapsed graph. $(9)$ - $G/P$ is a 1-graph, i.e. let $(a,b)$ and $(a',b')$ in $A \times B$ be two arcs of $G$, one and only one arc will be from $A$ to $B$.

Definition 1: Let $G$ be a directed graph, let $P$ be a partition of $G$, $P$ is said transitive ly consistent iff $(\forall A,B \in P)(\exists$ a path from $A$ to $B) \Rightarrow (\exists a \in A, \exists b \in B: \exists$ a path from $a$ to $b$ in $G$).

If a partition is not transitively consistent (see figure 1) further harmful constraints will appear in the collapsed graph which may bring on parasitic deadlocks. For example, if the graph of the figure 1 is composed with another one such as a path from the vertex 3 to the vertex 4 appears in the composed graph, an inconsistent circuit including $B$ and $C$ will be observed in the collapsed composed graph.
Definition 2: let $G$ be a directed graph, let $A$ be a sub-graph of $G$, $A$ is a granule of $G$ iff there is a path from any incoming arc of $A$ to any outgoing arc of $A$.

Proposition: let $G$ be a directed graph, let $P$ be a partition of $G$, $(\forall p \in P)(p \text{ is a granule of } G) \Rightarrow (P \text{ is transitively consistent}).$

4.1.2 - An algorithm of granulation.

A first algorithm, formally founded /8/, called glutton, is implemented to compute partitions whose any part is a granule. Let us describe it. We are building a transitively consistent partition of a directed graph. Let $s$ and $s'$ be two already built granules, if $s'$ is the only successor of $s$ then we say $s$ is a root, if $s$ is the only predecessor of $s'$ then we say that $s'$ is a bough. Now we have the two following rules: a bough can grow on whatever it follows (see figure 2) and a root can grow on whatever follows it (see figure 3), and they remain granules.

![Figure 2](image2.png)

![Figure 3](image3.png)

On the contrary, the other cases give a slip and are not allowed (see figure 4).
The complexity of the algorithm \textit{glutton} is linear for no backtracking is useful for building, but it does not give the biggest granules. It makes up a new granule from two neighbours already existing, whereas sometimes three granules, such that no pair of them can give one, are necessary to build a new one (see figure 5).

Actually, new more powerful algorithms are being developed.

The Conditional Dependences Graph is a labelled graph, whose labels are clocks indicating the validity instants. Obviously, a granule must remain a granule at any instant. If \textsc{CLOCK} is not constant on the whole granule there are instants where some arcs are not valid while some others are. In this case the granule property may be lost. If the partition in granules follows a partition by clock, i.e. we apply \textit{glutton} to the "mono-clock" parts, this problem is solved.

\section*{4.2 - The hierarchy of the Conditional Dependences Graph.}

Thus we initiate a \textit{hierarchy} of the Conditionnal Dependences Graph.

\textit{Definition 3} : a hierarchy of graph is a pair \((G,PC)\) where

(i) \(G\) is a directed graph

(ii) \(PC\) is a totally ordered class of partitions of the vertices of \(G\)
The order of partitions stems from inclusion over the set of parts of $X$, the set of the vertices of $G$: given $P$ and $Q$ two partitions of $X$, $P \leq Q$ iff $(\forall p \in P)(\exists q \in Q, p \subseteq q)$.

We have two levels of partitions, the granulation and the "mono-clock," partition. They constitute the first stages of the hierarchy. Other stages have to be added to obtain a hierarchy on the control. For this two interesting relations can be considered on the set of clocks.

Clock lattice.

Easily, we can build a partial order on the clocks from their equations of definition. It is provided by the following implication

$$\forall x,y,z \in \mathbb{Z}/3\mathbb{Z} \quad x^2 = y \quad z^2 \Rightarrow x^2 \leq z^2$$

This order is the intuitive order on signals: $x^2 \leq z^2$ means that the corresponding signal $x$ is less frequent than the corresponding signal $z$.

Exclusion relation.

Another useful relation is the exclusion:

$$\forall x,y \in \mathbb{Z}/3\mathbb{Z} \quad x,y \text{ excludes themselves} \iff x, y = 0$$

It is clear that the affectation problems are quite difficult and no automatic method is perfect. So our users have access to interactive handling tools of the hierarchy: permanently a SIGNAL syntactic tree is managed, this tree can be lexically and graphically displayed.

5 - CONCLUSION.

The programming language SIGNAL is made up of high level operators carefully choosen for their appropriateness to real-time applications. Improvements of our algorithms are under consideration: geometrics approach of the clock calculus which regards equations as manifolds, thorough study of intrinsic properties of graphs in order to find not very complex algorithms giving bigger granules, indeed, even the biggest ones, new interpretation of the clock equations providing a control automaton almost immediately, development of valuation tools of graphs to improve the mapping. About this last point, path algebras, dioids, ordered spaces, probability spaces are good theories for evaluating. So it is a matter of elaborating morphisms from graphs to these structures.
REFERENCES.


