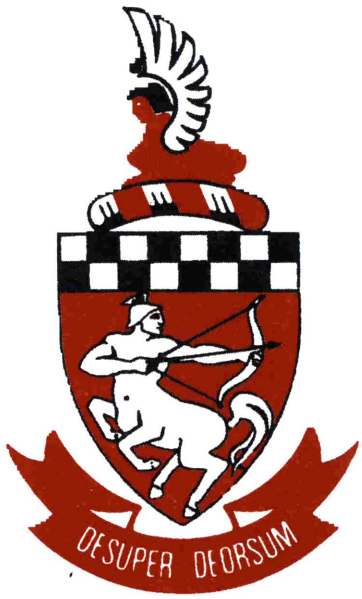
South African Computer Journal

Number 25
August 2000

Suid-Afrikaanse Rekenaar-Tydskrif

Nommer 25
Augustus 2000



The South African Computer Journal
An official publication of the Computer Society of South Africa and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarvereniging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

World-Wide Web: `http://www.cs.up.ac.za/sacj/`

Subscriptions

	Annual	Single copy
Southern Africa	R80.00	R40.00
Elsewhere	US$40.00	US$20.00

An additional US$15 per year is charged for airmail outside Southern Africa

to be sent to:

Computer Society of South Africa
Box 1714, Halfway House, 1685
Phone: +27 (11) 315-1319 Fax: +27 (11) 315-2276

In Memoriam:
Stef Postma

South Africa has lost one of her most colourful and eminent computer scientists. Professor Stef Postma passed away peacefully in his sleep on May 5, 2000 after a short illness. He will be remembered for his forthright views and total integrity. Never a man to shy from controversy, he always debated his position with vigour, displaying his extensive vocabulary at every opportunity.

Those who knew him mourn the loss of a very good friend.

Stef was born on August 10, 1938 in Graaff-Reinet and matriculated from Höerskool Linden in Johannesburg. He majored in geology and mathematics at the University of the Witwatersrand and graduated with honours in mathematics from that university. Stef devoted much of his life to promoting computer science as a science and to this end spent a lot of energy and time defining syllabi for undergraduate and honours courses at our universities. He was the prime mover in creating the South African Institute of Computer Scientists and Information Technologists (SAICSIT) in 1982, providing a professional body to represent the interests of local computer scientists. He was also instrumental in establishing Quaestiones Informaticae (now the South African Computer Journal) which afforded South African computer scientists the opportunity to publish papers locally in a refereed journal.

-Doug Laing

Links2Go "Computer Science Journals" Award

The SACJ production editing team received the following unsolicited e-mail:

The page at <http://www.cs.up.ac.za/sacj/>, was selected as a Links2Go "Key Resource" in the Computer Science Journals topic, at <http://www.links2go.com/topic/Computer_Science_Journals>.

How your page was selected:

Each quarter, Links2Go samples millions of web pages to determine which pages are most heavily cited by web pages authors, such as yourself. The most popular pages are downloaded and automatically categorized by topic. At most 50 of the pages related to a topic are selected as "Key Resources." Out of 50 pages selected as Key Resources for the Computer Science Journals topic, your page ranked 19th. For topics like Music, where there are a large number of interested authors and related pages, it is harder to achieve selection as a Key Resource than for a special-interest topic, such as Quantum Physics.

The Links2Go Key Resource award differs from other awards in two important ways. First, it is objective. Most awards rely on hand selection by one or more "experts," many of whom have only looked at tens or hundreds of thousands of pages in bestowing their awards. Selection for these awards means no more than that one person, somewhere, noticed your page and liked it enough to select it. The Key Resource award, on the other hand, is based on an analysis of millions of web pages. Any group or organization who conducts a similar analysis will arrive at similar conclusions. When Links2Go says your page is a Key Resource, we mean that your page is one of the most relevant pages related to a particular topic on the web today, using an objective statistical measure applied to an extremely large data set.

Second, the Key Resource award is exclusive. We get literally hundreds of people requesting that their page be added to one or more topics per week. All of these requests are denied. The only way to get listed as a Key Resource is to achieve enough popularity for our analysis to select your pages automatically. We do not accept fees, offers of link exchanges, free advertising, or bartered livestock as inducements to add new sites to our lists. Fewer than one page in one thousand will ever be selected as a Key Resource.

Once again, congratulations on your award! Links2Go Awards awards@links2go.com

Research Article

A Declarative Framework for Temporal Discrete Simulation

Habib Abdulrab[a] Macaire Ngomo[b] Abdenbi. Drissi-Talbi[a]

[a]Advanced Computer Science Laboratory PSI-LIRINSA, INSA Rouen
P.O. Box 08 76131 Mont-Saint-Aignan Cedex, France
{abdulrab,drissi}@insa-rouen.fr

[b]A6-MédiaGuide Group
42, rue Paul Claudel - 91000 Evry France
ngomo@caramail.com

Abstract

The aim of this paper is to introduce temporal processes and linear constraints on naturals for solving temporal discrete simulations problems. We first describe a new approach for solving such constraints, based on the generation of parametric solutions. The algorithm of the solver of linear constraints on naturals, its proof, as well as a general presentation of its implementation are presented. Then we present the main concepts that we introduce (temporal process, process reduction, process scheduler based on solving linear constraints on naturals...) for temporal discrete simulation. Temporal calculus is solved here via the resolution of linear constraints on naturals. Such constraints are especially generated by the predefined predicates associated with temporal processes, and by the processes scheduler. Terms used in this temporal calculus can use duration and dates as logical variables, and the resolution determines all their values in a parametric form. The resulting programming system allows the events and activities associated with an application to be simulated, and produces all their possible resolution dates.

Keywords: *integer programming, process, model of temporal computation, simulation, constraints on natural numbers, unification.*

1 Introduction, definitions and notations

Simulation is a tool which allows the behaviour of a given system to be produced artificially [16, 17]. Its main goal is first to model the system and then to determine its temporal behaviour by evaluating a *schedule* that specifies the dates of the realisation of the *events* which describe this behaviour. In general, the modelling activity disposes of some variables (state variables) whose values represent the states of the simulated system. In the case of discrete events simulation, the state alterations arise at specific dates; each state alteration represents an *event*, and it is supposed that between two events the state of the system remains constant [18]. The instant at which an event is produced is called the *occurrence date* of the event. Action which has a duration is called *activity*. The realisation of an activity consists in executing its starting event and in delaying the execution of its ending event until that the simulation time reaches its occurrence date.

Intuitively, we suppose in this paper that we have n sequences of tasks. These will be represented by *temporal processes*. Tasks can be given with variable (or constant) duration and variable (or constant) starting or ending dates. We suppose that some entities (*resources*) of the system to simulate can be used by a limited number of tasks. The aim of our temporal simulation is to process these tasks, and to compute all the temporal solutions corresponding to the values of temporal variables.

In this paper we present (in the framework of discrete simulation) a simulation model, designed and implemented over an object oriented and logic programming language LOP [1, 3] (Section 2). The resulting system offers some helpful tools for the modelling phase (thanks to object and logical paradigms), and for the computation of the schedule (thanks to the resolution of temporal constraints via a new method [2] solving linear constraints on naturals (Section 3). The main concepts that we introduce (temporal process, process reduction, process scheduler based on solving linear constraints on naturals...) are presented in Section 4. Their applications in discrete simulation is presented in Section 5.

We denote by ϕ the set of all linear polynomials of the form: $\zeta_1 z_1+\zeta_2 z_2+ \ldots+\zeta_n z_n+\zeta_{n+1}$, where the z_i are natural variables, and the ζ_i are natural numbers.

A system Σ of linear diophantine equations and inequations (or a system of *linear constraints on naturals*) is given by:
L_i #i M_i, where i=1, ..., p, L_i, $M_i \in \phi$, and #i $\in \{=, <, >, \neq, \geq, \leq\}$.

A *transformation* ω of the variables z_1, z_2, ..., z_n is defined by the following application: $z_i \rightarrow N_i$, where i=1, ..., n, and $N_i \in \phi$.

A transformation ω is called a *parametric solution* of the system Σ, if for each i=1, ..., p, we have $\omega(L_i)$ #i $\omega(M_i)$, where $\omega(L_i)$ and $\omega(M_i)$ are the linear polynomials obtained after the substitution of each variable z_j of Σ by N_j. If in a parametric solution each

polynomial N_i is a natural number, we call it a *numerical solution* of the system Σ, or simply a *solution*. If we assign any natural values to the parameters $z_1, z_2, ..., z_n$ of a parametric solution, this becomes a numerical solution. Thus, a parametric solution describes a certain class of numerical solutions. The set of all the parametric solutions of Σ is called the *general solution* of Σ.

The *index* of the system Σ is a pair of natural numbers (p, r), where p is the number of the equations and inequations in the system Σ, r is the number of the positive coefficients in L_1 and M_1. We say that $(p_1, r_1)<(p_2, r_2)$, iff $(p_1<p_2)$ or $(p_1=p_2 \ \& \ r_1<r_2)$. Dates are represented, relative to an origin date, by the set Z of integers. The difference between two successive dates represents the chosen temporal unit (the second, the hundredth of a second,...).
The common usual representations of dates (second/ minute/hour/day/month/year,...) can be easily coded in this representation. The occurrence dates are represented by positive dates, and the constraints dedicated to compute them are linear constraints on naturals.

2 Features of object oriented and logic programming in LOP

LOP is an object oriented and logic programming system supporting some tools that can help the conception and the implementation of programs. LOP is entirely designed according to object methodology, under the form of some hierarchical systems. Thus, it is open to other extensions. In this section, our presentation is restricted to a few of the programming tools of LOP, and then we present in section 3 the new extensions that allow our simulation model to be developed.

The realisation of a LOP program consists in defining a base of classes and objects to be used, and in solving requests using this base. The classes that describe the structures and the behaviour of the objects are defined in this base. A class is defined by giving its slots and its logical methods. In addition, each class can indicate its superclasses (its inherited classes). The predicate defclass is used for the definition of a class as follows: defclass(cl (supers) (slots)); where cl is the name of the class, supers are the direct superclasses of cl and slots are the slots of the class. The objects (instances of the class) are defined by the predicate make-instance as follows: make-instance(obj cl $slot_1$ val_1 ... $slot_n$ val_n); where obj and cl are the names of the created object and its class respectively, $slot_i$ is the name of the slot of the class and val_i is the value (associated with obj) of the i-th slot ($slot_i$).

A logical method over a class can be defined by some particular clauses, called object clauses. An object clause has the form of a Horn clause (it has a head, and a body composed of literals), but the literals of the body may be methods call (i.e. message sending). The usual literals are formed from a predicate followed by their arguments: pred(p_1...p_n), and the calls of methods have the form: Rec<-pred(p_1...p_n); where Rec is the object that receives the method as a message and pred is the predicate associated with the method. The definition of a method m (over a class cl, by the clauses $clause_1$...$clause_n$) is realized by the predicate defmethod as follows: defmethod(cl $clause_1$); ..., defmethod(cl $clause_n$); where each $clause_i$ has the form: O<-m(...):-body; O is in general a variable to be unified with the object receiving the message. The literals of the body of the clause can be usual literals or calls of methods.

Requests to solve are composed of a sequence of terms which can be usual literals or calls of methods. The resolution scheme of LOP is an adaptation of that of Prolog in the framework of objects. The reduction of the usual literals is identical to that of Prolog. On the other hand, the reduction of a term of the form obj<-m(...) is done by using all the clauses that define m over the class of obj (these clauses can be locally defined over this class, or inherited from its superclasses).

We must mention here the two predicates (among the predefined LOP's predicates) get and set, used respectively to consult and to modify the values of the slots of the objects. The reduction of a term of the form: obj<-get(slot V) allows V to be unified with the value of the attribute slot associated with the object obj. This allows the state of the object obj to be consulted. And the reduction of a term of the form: obj<-set(slot NV) allows the current value of the slot slot of the object obj to be bound temporally with the new value NV. The last value will be restored after backtracking, and the new one will be burned.

3 LOP's solver of linear constraints on naturals

3.1 Different representations of the set of all the solutions

LOP's solver of linear constraints on naturals is based on a new approach [2] that computes the general solution of linear constraints on naturals.

We must recall first the classical approach for solving this problem. Given a system of linear diophantine equations on naturals (namely: $Ax = b$, $x \in |N^n$, where A is a matrix of m lines and n columns, and all the

coefficients of A and b are integers), the classical approach aims to compute (via different methods, such as integer programming for example) two bases B_1 and B_0, where B_1 is the set of the *minimal* solutions (for the component-wise order) of $Ax = b$ and B_0 is the set of the *minimal* (for the component-wise order) non-zero solutions of $Ax = 0$:

$$B_1 = \{ x \in |N^n / Ax = b \text{ and } \nexists \; y \in |N^n / y < x \text{ and } Ay = b\}$$

$$B_0 = \{ x \in |N^n / x \neq 0 \text{ and } Ax = 0 \text{ and } \nexists \; y \in |N^n - \{0\} / y < x \text{ and } Ay = 0\}$$

Note that B_0 and B_1 are finite sets, and the general solution of $Ax = b$ can be represented by $B_1 + B_0^*$, where $B_0^* = \{ \sum_{i=1}^{k} k_i p_i \; / p_i \in B_0 \text{ and } k_i \in |N\}$.

For example, the general solution of the equation $2x_1 = x_2 + x_3$ is represented by $B_0 = \{(1,0,2), (1,2,0), (1,1,1)\}$, and $B_1 = \{(0,0,0)\}$. The resolution of inequations, in this framework, consists in introducing slack variables.

A variety of works shows how to compute a representation of the general solution based on this approach. A non exhaustive list of such works can be found in [5,6,7,8,9,10,11,12, 13,14].

The new approach [2] is based on a constructive proof of the following theorem.

Theorem: There exists an algorithm that computes the general solution of any system Σ of linear constraints on naturals, by means of a *finite* set S of parametric solutions. In addition, each solution of Σ can be deduced from a *unique* parametric solution of S.

This allows the computation of a new *parametric* representation of the set of the solutions of linear constraints on naturals (without adding slack variables in the case of inequations). Unlike the classical representation of the general solution, the parametric representation gives direct functions representing the general solution, without any linear combination. For example, the general solution of the equation $2x_1 = x_2 + x_3$ already seen is given by the two parametric solutions:

$$\begin{cases} x_1 \to x_1 + x_2 \\ x_2 \to 2x_1 \\ x_3 \to 2x_2 \end{cases}$$

Corresponding to the case where x_2, x_3 are even

and

$$\begin{cases} x_1 \to x_1 + x_2 + 1 \\ x_2 \to 2x_1 + 1 \\ x_3 \to 2x_2 + 1 \end{cases}$$

Corresponding to the case where x_2, x_3 are odd .

Note that x_1 and x_2 of the right hand-side of the transformations can be seen as two new and *free* variables. The general solution is obtained here by giving to the free variables all the nonnegative integer values.

Unlike the classical representation, this parametric one is *nonambiguous*, in the sense that each solution can be deduced from only one of the finite set of parametric solutions. As a counter-example, the solution (3,3,3) can be deduced from $B_0 = \{(1,0,2), (1,2,0), (1,1,1)\}$, and $B_1 = \{(0,0,0)\}$ in an ambiguous manner:

$(3,3,3) = (0,0,0) + 3\,(1,1,1) = (0,0,0) + (1,0,2) + (1,2,0) + (1,1,1)$.

Here is an example of a system of three inequations: $3z_1 + 2z_2 > 8$, $2z_1 < 4z_2 + 7$, and $z_1 > 2$. It has the two following parametric solutions:

$$\begin{cases} z_1 \varnothing 3 + 2z_1 \\ z_2 \varnothing z_1 + z_2 \end{cases}, \quad \begin{cases} z_1 \varnothing 4 + 2z_1 \\ z_2 \varnothing z_1 + z_2 + 1 \end{cases}$$

This new representation of the general solution is particularly suitable for constraint logic programming, because it gives a direct (without linear combinations) and incremental description of the general solution.

3.2 Algorithm

LOP's solver of linear constraints on naturals is based on an algorithm, that we extract here from the proof of the last theorem, called *Solve(Σ, α)*, where Σ is the system to be solved, and α is a transformation. The initial value of α is the identity Id: $z_i \to z_i$, i=1, ..., n. Σ' denotes here the equations and the inequations: $L_i \; \#i \; M_i$; i=2, ..., p. [k] denotes the greatest integer less than or equal to k. The algorithm considers here only the equality = and the inequality <; the constraints using the other predicates $\{>, \geq, \leq, \neq\}$ can be transformed to these two predicates, because: a) if an inequation of the system has the form $L_i > M_i$, we can exchange L_i and M_i and obtain the inequation $L_i < M_i$; b) if an inequation has the form $L_i \neq M_i$, we can replace the system by two systems, by substituting the i-th inequation by $L_i < M_i$ in the first one and by $L_i > M_i$ in the second one. The solution of the system Σ is obtained by the union of the solutions of these systems; c) if an inequation has the form $L_i \neq M_i$, we can replace the system by two systems, by substituting the i-th inequation by $L_i < M_i$ in the first one and by $L_i = M_i$ in the second one. The solution of the system Σ is the union of the solutions of these systems; d) the case $L_i \geq M_i$ is analogous to c).

The output of this algorithm is a finite set of parametric solutions representing the general solution.

Solve(Σ, α):

1) If Σ is empty, Then Return α.

2) Else If $L_1\#1M_1$ has the form: $\zeta_{n+1}\{<, =\}\eta_{n+1}$, Then If such an expression is false, Then Return (), Else Return Solve(Σ', α).

3) Else If $L_1\#1M_1$ has the form: $\zeta_{i1}z_{i1}+\zeta_{i2}z_{i2}+ ...+\zeta_{ir}z_{ir}+\zeta_{n+1}\{<, =\}\eta_{n+1}$, where $r>0$, $\zeta_{i1}>0$, ..., $\zeta_{ir}>0$, $\zeta_{n+1}\geq 0$, $\eta_{n+1}\geq 0$; Then, let β_1, ..., β_u be the numerical solutions of this equation or inequation.

Return $\cup\{i=1,..., u\}$ Solve($\Sigma'\beta_i$, $\alpha\beta_i$).

4) Else if $L_1\#1M_1$ has the form: ζ_{n+1} $\{<, =\}$ $\eta_{i1}z_{i1}+\eta_{i2}z_{i2}+ ...+\eta_{ir}z_{ir}+\eta_{n+1}$, where $r>0$, $\eta_{i1}>0$, ..., $\eta_{ir}>0$, $\zeta_{n+1}\geq 0$, $\eta_{n+1}\geq 0$.

Then, If #1 is =, Then exchange the two hand-sides and consider step 3), Else,

4.1) If $\zeta_{n+1} < \eta_{n+1}$, Then Return Solve(Σ', α).

4.2) If $\zeta_{n+1} \geq \eta_{n+1}$,

let $c_r=[(\zeta_{n+1}-\eta_{n+1})/\eta_{ir} +1]$, and $\beta_{cr} = z_{ir}\rightarrow z_{ir}+c_r$,

If r = 1, Then Return Solve($\Sigma'\beta_{cr}$, $\alpha\beta_{cr}$),

Else, let $\beta_k = z_{ir}\rightarrow k$; $k = 0,..., c_r -1$.

Return $\cup\{k=0,...,cr-1\}$ Solve($\Sigma\beta_k$, $\alpha\beta_k$) $\cup$ Solve($\Sigma'\beta_{cr}$, $\alpha\beta_{cr}$).

5) Else if $L_1\#1M_1$ has the form: $L^1 +\zeta_u z_u+L^1$ $\{<, =\}$ $M^1 +\eta_u z_u +M;^1$, where $\zeta_u>0$, $\eta_u>0$. Then, If $\zeta_u<\eta_u$, replace this equation or inequation by $L^1 +L_1$ $\{<, =\}$ $M^1 +(\eta_u-\zeta_u)z_u+M;^1$, respectively. Else, replace it by $L^1 +(\zeta_u-\eta_u)z_u+L^1$ $\{<, =\}M^1 +M;^1$, respectively.

Return Solve(Σ, α).

6) Else $L_1\#1M_1$ has the form: $\zeta_i z_i+L^1$ $\#1$ $\eta_j z_j+M^1$, where $\zeta_i>0$, $\eta_j>0$ and $i \neq j$.

Let β_1, ..., β_u ; $u = \eta_j\zeta_i/k^2$, be the u transformations :

$z_i\rightarrow(\eta_j/k)z_i+\beta$,

$z_j\rightarrow(\zeta_i/k)z_j+\gamma$;

where $k = gcd(\eta_j, \zeta_i)$, β and γ are any integers that satisfy $0 \leq \beta < \eta_j/k$, $0 \leq \gamma < \zeta_i/k$. Let γ_1 and γ_2 be the two transformations: $z_i\rightarrow z_j+z_i$ and $z_j\rightarrow z_j+z_i+1$, respectively.

Return $\cup\{s=1,...,$ $u\}$ (Solve($\Sigma\beta_s\gamma_1$, $\alpha\beta_s\gamma_1$) $\cup$ Solve($\Sigma\beta_s\gamma_2$, $\alpha\beta_s\gamma_2$)).

3.3 Proof of the algorithm

We prove by induction on the index of the system Σ, that Solve(Σ, α) ends with a finite set of parametric solutions describing the general solution of Σ (see also [2]).

If the index equals (0, 0), then the system Σ is empty. Its general solution is $z_i\rightarrow z_i$ (i=1, ..., n).

Suppose that this fact is true for each system Σ whose index is smaller than (p, r), and consider any system whose index is (p, r).

The main idea of the proof, in this case, is the following: Solve(Σ, α) returns Solve($\Sigma\delta_1$, $\alpha\delta_1$) $\cup$ ... $\cup$ Solve($\Sigma\delta_t$, $\alpha\delta_t$); $t \geq 0$, where the index of each $\Sigma\delta_i$ is clearly smaller than (p, r), and if $\theta_{i,1}$, ..., $\theta_{i,s}$ is the general solution of $\Sigma\delta_i$, then the general solution of Σ is $S = \{\alpha\delta_1\theta_{1,1}, ..., \alpha\delta_1\theta_{1,s}, ... \cup \alpha\delta_i\theta_{i,1}, ..., \alpha\delta_i\theta_{i,s}, \cup ..., \alpha\delta_t\theta_{t,1}, ..., \alpha\delta_1\theta_{t,s}\}$. Thus, by induction, Solve(Σ, α) returns a finite set of parametric solutions representing the general solution of Σ. In addition, one can prove that each solution of Σ can be deduced from only one element of S.

3.4 Implementation

The implementation of this algorithm uses several improvements that exceed the scope of this extended abstract. The main features of our implementation are the followings:

1) All the equations of the form: $\zeta_{i1}z_{i1}+\zeta_{i2}z_{i2}+ ...+\zeta_{ir}z_{ir}+\zeta_{n+1}\{<, =\}\eta_{n+1}$, where $r>0$, $\zeta_{i1}>0$, ..., $\zeta_{ir}>0$, $\zeta_{n+1}\geq 0$, $\eta_{n+1}\geq 0$; (cf. the third step of the algorithm) are solved as a system, and not by solving the first equation and substituting its solutions to the rest of the system, and solving new systems. The resolution of the whole system of such equations is directly done by using some powerful heuristics introduced for the numeration of the solutions of such systems.

2) A technique of intervals is introduced to denote a sequence of equations of the form $L= n_1$, $L= n_1+1$..., $L= n_2$, ($L \in \phi$), by $L=[n_1, n_2]$. The resolution of systems where the second numbers are intervals $[n_1, ..., n_2]$ (which appear frequently) is particularly optimised.

3) The solutions where $\#i \in \{>, \neq, \geq, \leq\}$ are computed directly, in the same manner as $\#i \in \{=, <\}$, without duplicating the equations and inequations of the initial system, which are given here to simplify the algorithm.

A solver based on this implementation is integrated in the resolution scheme of LOP. The new unification algorithm takes into consideration the terms of ϕ. For example, the unification of the two terms L and $M\in\phi$ is transformed into the resolution of the constraint L=M. Natural variables that appear in L and M are considered as logical variables to be bound with the parametric solutions produced by the constraints resolution. The extension of the unification algorithm by this constraints solver allows some common predicates related to naturals to be simply expressed. For example, one can define the clauses:

```
even(2X).
odd(2X+1).
modulo(QM+R,M,R) :- R<M.
```

where even(N) and odd(N) test the parity of N, and modulo(N,M,R) expresses that N is equal to R modulo M. Thus, one can solve the following requests:

```
?- even(3).
solution1: No.
?- even(4).
solution1: yes.
?- even(N).
solution1: N=2X.
?- modulo(N,3,R).
solution1: N=3Q', R=0.
solution2: N=3Q"+1,R=1.
solution3: N=3Q"'+2,R=2.
```

4 Concepts of temporal discrete simulation

The goal of this section is to describe the main concepts (temporal process, process reduction, process scheduler based on solving linear constraints on naturals...) that we introduce for modelling a discrete simulation of a temporal systems. These concepts allow the representation of the usual notions of *events* and *activities* that are necessary for simulation (cf. Section 5).

4.1 The resources

Typically, a resource is a particular object (machine, place,...) used by the simulation entities in order to carry out some particular tasks. The *capacity* of a resource is the number of entities which can use it together. By default, the capacity value is 1. The operations seize and release are defined to seize and release resources, respectively.

4.2 Temporal processes

A temporal process is defined by the following term: process(p,T,F,(tasks))

where p is the process name, $T \in \phi$ is the current time of the process, $F \in \phi$ is the final time of the process, and tasks is the process request. It is important to observe that T and F are any expressions of ϕ and not only natural numbers.

The current time of the process corresponds to its simulation time since the beginning of the simulation. This time progresses with the resolution of the process tasks. The final time corresponds to the simulation time that the process can reach when its tasks are completely solved. The process *tasks* is composed of a sequence of terms to be solved sequentially.

A request can be composed of some processes separated by the character &, as follows:

process(p1,T1,F1,(task1)) &
process(p2,T2,F2,(task2)) & ...

The resolution of such a request consists in the *reduction* of its processes. A request is successively solved if each one of its processes is successively solved.

4.3 Process reduction

The reduction of a process corresponds to the usual logical resolution of the terms of its *tasks*. This resolution allows its temporal evolution to be represented and computed by using the following predefined predicates:

1) delay(Δ): the reduction of this term increases the current time of the active process by Δ. Thus, the process process(p,T,F,(delay(Δ),...)) is reduced to process(p,T+Δ,F,(...)).

2) new_time(T'): the reduction of this term allows the current process time T of the active process to be *replaced* by the new current time T'. This succeeds if the constraint T'>=T is satiable.
The process process(p,T,F,(new_time(T'),...)) is then reduced to the process process(p,T',F,(...)). The old current time is restored after backtracking.

3) current_time(X): allows the current time of the active process to be extracted. Thus, process(p,T,F,(current_time(X),...)) is reduced to process(p,T,F,(...)) after the unification X=T.

4) last_time(X): allows the final time of the active process to be extracted. Thus, process (p,T,F, (last_time(X),...)) is reduced to process (p,T,F,(...)) after the unification X=F.

Observe that a process should always satisfy the constraint " current time <= final time". For instance, the reduction of the term process(p,T,F,(delay(Δ),...)) is only taken into account if T+Δ <= F succeeds, otherwise the reduction fails.

Another particular reduction is realised in the case of the definition of a son process. This corresponds to the reduction of the process: process (p,T,F,(process (p',T',F',(...)), ...)). This consists in separating the son process from its father process as follows: process (p,T',F,(...)) & process(p',T',F',(...)). The current time of the father process p becomes the current time of the son process p'. This reduction succeeds if the constraint T'>=T is satiable.

The remaining cases correspond to the reduction of usual literals (system's predefined or user's literals). This reduction has no effect on the temporal arguments of the process except the resource allocation operations *seize* and *release*. The effect of these operations is described in 4.4.

Finally, a success is obtained whenever a process becomes empty and the unification T=F succeeds.

4.4 Process scheduler

If $p_1,...,p_n$ are all the processes composing a request, and $T_1,...,T_n$ their respective current times, a process is called *activable* if and only if the system of constraints : $T_i<=T_1,...,T_i<=T_n$ admits a solution on naturals. This means that the hypothesis "T_i is the minimum of all the current times" is satiable. Note that more than one process can be *activable* at the same time, and at most one process can be *active* at each moment.

The *active* process is chosen among all the *activable* processes. In order to avoid favouring one particular process, LOP's scheduler considers all the possible choices of the active process among the *activable* processes. It starts by choosing an *activable* process p_i as active (after the resolution of the system $T_i<=T_1,...,T_i<=T_n$), and continues the resolution. When it comes back to this resolution point (by backtracking), the choice of p_i is cancelled, and another *activable* process is chosen to be active. Thus, all the choices corresponding to all the *activable* processes are taken in account. This is called here the *non deterministic activation* mechanism. The point of the resolution at which the scheduler is invited to determine the active process is called the *choice point of activity*.

In addition, a process can have one of the following states:

1) Active: it is in the reduction phase. In LOP, we consider for the moment that at most one process can be active at each moment. (But, this choice can be "Or-paralleled" to all the *activable* processes at the same time). 2) Waiting: the process satisfies all the activation conditions, but another process is already active. 3) Suspended: the process waits for the realisation of some conditions (for example, waiting for a resource). 4) Finished: the process has finished the resolution of its task.

The call of the scheduler during the resolution is limited to some precise moments. It arises when a process is in one of these cases:

a) Its current time is changed. b) Its tasks are completely resolved. c) The reduction of one literal of its tasks fails. d) The reduction of the resource allocation operations *seize* and *release*. The effect of these operations is done as follows: 1) If c is a resource then the reduction of the term c<-seize() depends on the current capacity value n of the resource c. If n>0 then the reduction succeeds and the new current capacity value of c becomes n-1. Otherwise, the active process p is *suspended* until the resource c is released by another process. 2) The reduction of the term c<-release() by a given process p, increments by 1 the current capacity value of c. When a given resource c is released at a time t, the processes which have been suspended because of this resource become *activable*, with the new current time t.

In the cases a) and b) the scheduler determines the *activable* process at this resolution phase (it takes into account all the possible deblocked processes as well as the newly created processes) and a new choice point of activity is developed. In the case c) the active process backtracks in order to explore the other remaining choices.

5 Simulation in LOP

The resolution scheme of LOP adopts the same strategy as Prolog. Each resolution step of a request consists in reducing first the leftmost literal. The imposed order of this strategy contradicts the simulation spirit, where, in general, the activities are to be realised simultaneously [19]. The extension of LOP by the notion of temporal process allows this problem to be solved. Thus, requests can be composed of several parts to be solved alternatively. Each part can be associated with a process and will be considered as the task of the process.

LOP's simulation model is essentially based on the notion of temporal process. A process is seen as a sequence of activities (the tasks associated with the process) which will be carried out sequentially. The dates and the duration of the events and the activities can be determined via the predicates acting on the current times of the processes. Moreover, the activities related to different processes can be realised simultaneously. This realisation may depend on the availability of some *resources*.

5.1 The events

Recall that an event is characterised by its occurrence time and an action on the simulated system (almost a state alteration). In LOP, the action associated with an event can be defined by one or several clauses. The body of each clause contains the terms allowing all the necessary state alterations to be accomplished. An event occurrence time T' can be indicated in the body of a clause by the term new_time(T'). For example, consider the class person defined by: defclass(person () (place)); the slot place can take one of the values work, home or path according to the places that can be occupied by a person at different times of the day, as shown in Fig 1.

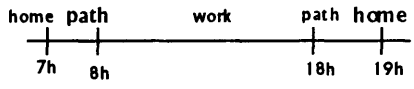


Figure 1: Places of a person in several times of a day.

The departures from home and from work are two events that can be defined over the class person by the following clauses:

O<-start(home):-current_time(T), T<=7, O<-get(place home),new_time(7),O<-set(place path).
O<-start(work):-current_time(T), T>=8, O<-get(place work), new_time(18),O<-set(place path).

The first clause means that a person who is at home, leaves it at 7h and takes a certain path to work. The second clause concerns the situation of a person at work. For a person created by: make_instance(per1 person (place home)), one can solve the following requests:

?- process(p,T,F,(per1<-start(X))).
solution 1: T<=7, F=7, X=home.
?- process(p,T,F,
(per1<-set(place work),per1<-start(X))).
solution 1: 8<=T<=18, F=18, X=work.

Fig2 shows the resolution steps of the first request. In the body of these clauses, the term new_time fixes the occurrence time of an event, and the term O<-set(place ...) represents the event actions. The term O<-get(place ...) is a precondition that selects the departure place of the person. Finally, the primitive current_time extracts the current time of the process p. Its use allows the introduction of some temporal constraints (T<=7 in the first clause, and T>=8 in the second) which play the role of preconditions on the realisation of events. In simulation terminology, events of this type are called *conditional*. Such preconditions on the current time events are generally related to the simulated system state at the moment of their realisations. In LOP, one can precondition the events by temporal constraints. This allows a better ordering of the actions to be undertaken. In addition, in LOP such temporal conditions can be any linear constraints on naturals using logical variables.

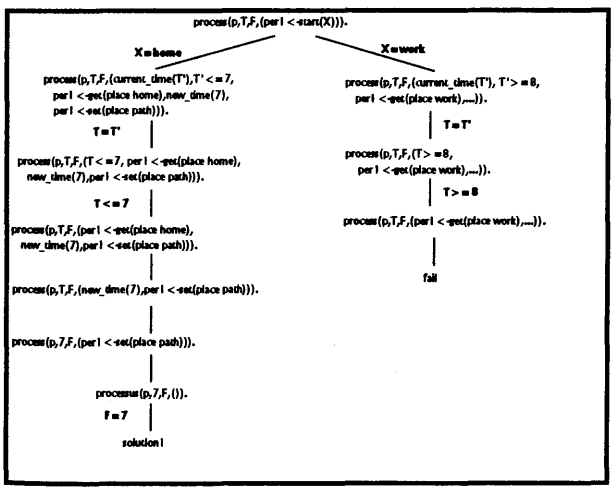


Figure 2: Starting point of the person per t

5.2 The activities

The activities are actions that continue between a starting event and an ending event. They are represented in LOP by logical clauses. The body of a clause that defines an activity is composed of terms corresponding to the starting and ending events, separated by other terms that represent the actions linked to the activities. The activity duration can be indicated in two ways. The first consists in giving the occurrence time D of starting event, and the occurrence time T of the ending event. Thus, the activity duration is F-D. The second way is to use the term delay(Δ) in order to specify the duration Δ of the activity. For instance, one can define the two activities "going to work" and "going home", of the example of Figure 1 by the two clauses:

O<-go_to(work):- O<-start(home), delay(1), O<-set(place work).
O<-go_to(home):- O<-start(work), delay(1), O<-set(place home).

Thus, it is possible to solve the following requests:

?- process(p,T,F,(per1<-go_to(X))).
solution 1: T<=7, F=8, X=work.
?- process(p,T,F,(per1<-set(place work),
per1<-go_to(X))).
solution 1: 8<= T<=18, F=19, X=home.

When delay appears in the body of a clause defining an activity, we say that it is of *explicit duration.* The argument of delay can be any term of ϕ using logical variables. The resolution determines all the temporal values of these variables. Consider now the following request:

?- process(p,T,F,(per1<-go_to(X),
per1<-go_to(Y))).
solution 1: T<=7, F=18, X=work, Y=home.

This corresponds to the realization of two successive displacements. X can only take the value work because per1 is initially at home. Similarly, Y can only take the value home because of the reduction of per1<-go_to(X). If now we compose this request with two processes:

?- process(p1,T1,F1,(per1<-go_to(X))) &
process(p2,T2,F2, (per1<-go_to(Y))).
solution 1: T1<=7, F1=8, X=work, 8<=
T2<=18, F2=19, Y=home.
solution 2: 8<=T1<=18, F1=19, X=home,
T2<=7, F2=8, Y=work.

we obtain two symmetrical solutions. The non deterministic activation (cf. 3.2.2) allows the roles of the two processes p1 and p2 to be exchanged. Both processes share the object per1 (communication by shared object). When one of them changes the state of the object, the other one realises the new activity corresponding to the new state.

5.3 Use of resources

Let us consider again Fig1. We suppose now that "going to work" path can be allocated to only one person at the same time. In this case the path can be considered as a resource with a capacity value equal to

1. In LOP, we can define the class *path* which inherits from the class *resource*. The "going to work" path can be defined as an instance, called path1, of the *path* class. The activity cross can be defined over the class path, by the following clause:

O<-cross():- O<-seize(), delay(1),
O<-release().

Thus, we can solve the request:

?- process(p_1,T_1,F_1,(path1<-cross())) &
process(p_2,T_2,F_2,(path1<-cross())).
solution 1: $T_1+1<=T_2$, $F_1=T_1+1$, $F_2=T_2+1$.
solution 2: $T_1<=T_2<=T_1+1$, $F_1=T_1+1$,
$F_2=T_1+2$.
solution 3: $T_2<=T_1<=T_2+1$, $F_1=T_2+2$,
$F_2=T_2+1$.
solution 4: $T_2+1<=T_1$, $F_1=T_1+1$, $F_2=T_2+1$.

Fig3 shows the different behaviours of the processes p_1 and p_2 in order to seize the resource $path_1$.

The first solution expresses the case where p2 seizes path1 after being released by p1. In this case, p2 is not suspended. In the second solution, p2 is suspended because it attempts to allocate the resource path1 when p1 is seizing it. This implies a tardiness, of one temporal unit, of F_2 according to F_1. In the first two solutions, p_1 has priority. The two remaining solutions are symmetrical to the first ones, and are obtained by the mechanism of non deterministic activation.

The non deterministic activation mechanism allows all the possibilities of resources allocation to be generated. For example, when both processes want to seize the same resource at the same time, LOP's scheduler allows two possibilities. Fig4 shows the resolution steps of the request:

?- process(p_1,7,F_1,(path1<-cross())) &
process(p_2,7,F_2,(path1<-cross())).
solution 1: $F_1=8$, $F_2=9$.
solution 2: $F_1=9$, $F_2=8$.

In classical simulation languages, different approaches are used to solve the situation described below (a conflict among the processes which attempt to seize simultaneously the same resource); processes are often divided according to some predefined priorities already fixed, and only one choice is made without any backtracking. If we need to know all the possible solutions, one should perform many simulation tests (without any guarantee that all the solutions will be found). In LOP, the non deterministic activation allows the complete generation of all the solutions.

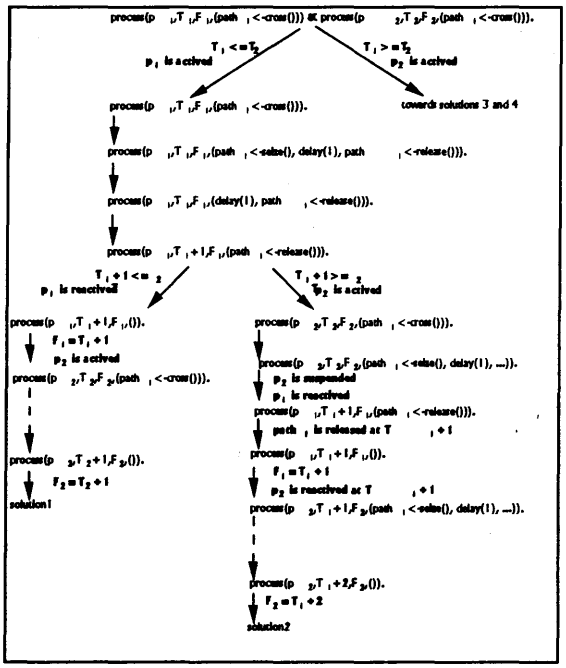


Figure 3: Two processes crossing the same path

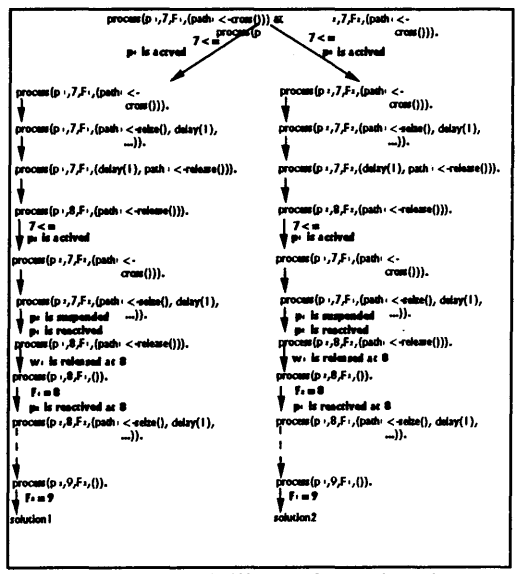


Figure 4: The two possibilities of crossing the same path by two processes having the same current time

6 Conclusions, comparison with related work, and perspectives

The originality of the simulation in LOP comes from the use of a solver of linear constraints on naturals for temporal calculus and temporal processes. This allows both the formulation of the requests with processes using logical variables as current times, and the use of variable duration. Linear constraints on these variables can be of a general form, without any restrictions on their values. The resolution determines all the temporal

values of these variables. This can not be done in other non deterministic simulation models (for example, T-PROLOG [19]), where duration and dates can not be formulated as logical variables. Moreover, a supplementary non deterministic mechanism is introduced to determine all the different temporal possibilities for the activation of the processes.

Solving linear constraints on naturals is realised in CHIP [4] in the framework of finite domains. The resolution in our approach gives a finite representation of the set of all the solutions (generally infinite) over the infinite domain of natural numbers.

Another interesting finite description of the set of the solutions of linear constraints on natural, using parametric solutions, is given more recently in [15]. It uses other types of transformations. These are less suitable from an implementational point of view, because of efficiency reasons, and because they do not allow to obtain a nonambiguous set of parametric solutions. At present, we are studying the improvements of the efficiency of the solver of linear constraints on naturals via the introduction of cutting planes methods in some steps of the solver's algorithm.

Bibliography

[1] Abdulrab H, *Logic, Objects and Parallelism,* In Proc. IWWERT'91, Lect. notes in Computer Science, n° 677, ed. Springer-Verlag, pp 133-149, 1991.

[2] Abdulrab H and Maksimenko M, *General Solution of Systems of Linear Diophantine Equations and Inequations,* In Proc. RTA'95, LNCS, n° 914, pp. 339-351, Kaiserslautern 1995.

[3] Drissi Talbi A and Abdulrab H, *LOP2: Extension du langage LOP1 par le paradigme objet,* In Proc.RJCIA'94, pp. 155-165, Marseille 1994.

[4] Van Hentenryck Pascal, *Constraint Satisfaction in Logic Programming*, MIT Press, 1989.

[5] Koscielski A and Pacholski L, *Complexity of Unification in Free Groups and Free Semi-groups*, Technical report, Institute of Computer Science, University of Wroclaw and Institute of Mathematics, Polish Academy of Sciences, 1989.

[6] Clausen M and Fortenbacher A, *Efficient Solution of Linear Diophantine Equations*, Special issue on Unification , JSC, 8:201-216, 1989.

[7] Contejean E, *On efficient Algorithm for Solving Systems of Linear Diophantine Equations*, Information and Computation, vol. 113, n. 1, pp. 143-172, 1994.

[8] Lambert, *Une Borne pour les Générateurs des Solutions Entières Positives d'une Equation Diophantienne Linéaire*. J-L. Compte rendu de L'académie des Sciences de Paris, 305:39-40, 1978.

[9] Pottier L, *Minimal Solutions of Linear Diophantine Systems: Bounds and Algorithms*, Proceedings of the fourth International Conference on Rewritting Technics and Applications, Como, Italy, pp. 162-173, 1991.

[10] Romeuf J. F., *Solutions of Linear Diophantine Systems*. Rapport LITP 88-76, Paris, 1990.

[11] Romeuf J.F., *A polynomial Algorithm for Solving Systems of Two Linear Diophantine Equations*. Technical report, LIR, Rouen, 1990.

[12] Domenjoud E, *Solving Systems of Linear Diophantine Equations: An Algebraic Approach*, Proceedings of 16th Mathematical Foundation of Computer Science, Warsaw, LNCS 520, Springer Verlag, 1991.

[13] Lamber, *Le problème de l'accessibilité dans les réseaux de Petri*, J-L, Nouvelle thèse, Université de Paris-sud, centre d'Orsay, 1987.

[14] Moulinet-Ossola C, *Algorithmique des réseaux et des Systèmes Diophantiens Linéaires*, Thèse, Université de Nice Sophia-Antipolis, 1995.

[15] Domenjod E and Tomas A.P., *From Elliott-Mac-Mahon to an algorithm for general linear constraints on naturals*. Proceedings of CP'95 (Constraints Programming), 1995.

[16] Narayanan S, Bodner D. A., Mitchell C. M., McGinnis L. F., Govindaraj T, Platzman L. K, *Object-Oriented Simulation Support Modeling and Control of Automated Manufacturing Systems.*", pp 59-63.

[17] Carrie A, *Simulation of Manufacturing Systems.*" John Wiley & Sons Ltd., 1988.

[18] Horwood E, Evans J.B, *Structure of Discret Event Simulation,* Chichester, 1988.

[19] Horwood E, *Artificial Intelligence in Simulation.*", I.Futo, T.Gergely Chichester 1990.

South African Computer Journal

Number 25, August 2000
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 25, Augustus 2000
ISSN 1015-7999

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research notes. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications of Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript
Manuscripts for *review* should be prepared according to the following guidelines:

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
- the title (as brief as possible)
- the author's initials and surname
- the author's affiliation and address
- an abstract of less than 200 words
- an appropriate keyword list
- a list of relevant Computing Review Categories
- Tables and figures should be numbered and titled.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text according to the Harvard. References should also be according to the Harvard method.

Manuscripts accepted for publication should comply with guidelines as set out on the SACJ web page,

`http://www.cs.up.ac.za/sacj`

which gives a number of examples.

SACJ is produced using the LaTeX document preparation system, in particular LaTeX 2_ε. Previous versions were produced using a style file for a much older version of LaTeX, which is no longer supported. Please see the web site for further information on how to produce manuscripts which have been accepted for publication.

Authors of accepted publications will be required to sign a copyright transfer form.

Charges
Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect typesetting, reproduction and other costs. Currently, the minimum rate is R30.00 per final page for contributions which require no further attention. The maximum is R120.00, prices inclusive of VAT.

These charges may be waived upon request of the author and the discretion of the editor.

Proofs
Proofs of accepted papers may be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Letters and Communications
Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words. Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book Reviews
Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement
Placement of advertisements at R1000.00 per full page per issue and R500.00 per half page per issue will be considered. These charges exclude specialised production costs, which will be borne by the advertiser. Enquiries should be directed to the editor.

South African Computer Journal

Number 25, August 2000
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 25, August 2000
ISSN 1015-7999

Contents