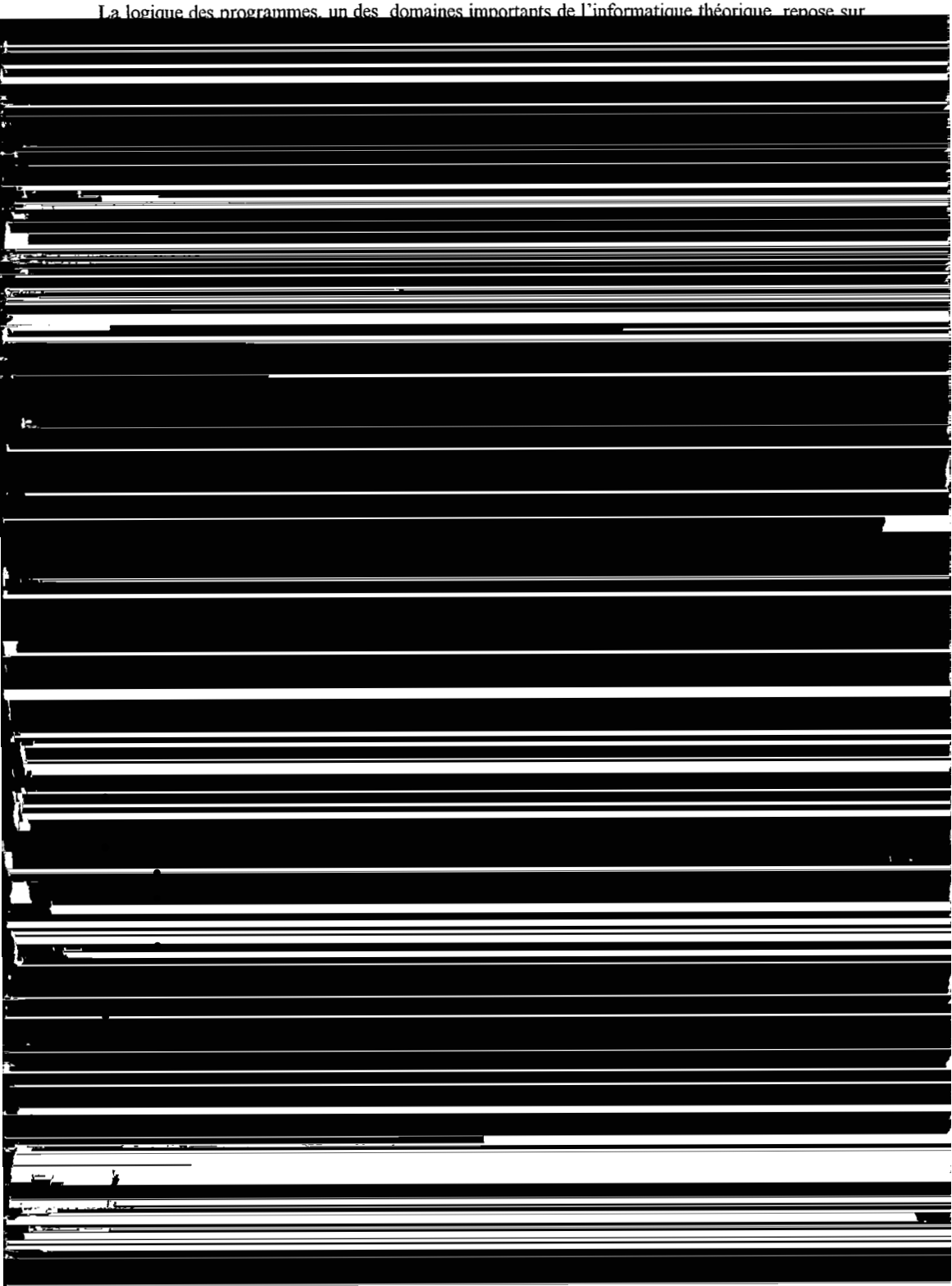
LOGIQUE TEMPORELLE POUR LA SPECIFICATION ET LA VERIFICATION DES SYSTEMES REACTIFS

M[r] GUERROUMI FAWZI
Centre de Développement des Technologies Avancées
128, chemin Mohamed Gacem B. P. 245 El-Madania Alger - ALGERIE

[redacted]

hension des phénomènes parallèles nécessite de prendre en compte un contrôle démultiplié et distribué.

La logique des programmes, un des domaines importants de l'informatique théorique, repose sur



• I est une fonction d'interprétation qui associe à la séquence *cond* des conditions sur les données une valeur de vérité.
Un marquage M d'un modèle PIPN est un ensemble de prédicats de base.

II.2 REGLE DE FRANCHISSEMENT

a) Transition sensibilisée
Si E est un ensemble de prédicats et $\theta = \{x_1=e_1, x_2=e_2, \ldots, x_n=e_n\}$ une substitution, E/θ repré-

III.2 INTERPRETATION PAR CLAUSES DE HORN

Chaque élément syntaxique est un fait ou une clause, décrit dans la syntaxe PROLOG. Un modèle PIPN constitue un programme PROLOG qui est directement simulé via une procédure de tir PROLOG. La forme externe du langage coïncide avec sa représentation interne dans l'outil.

A) INTERPRETATION DES TRANSITIONS

Convention: dans ce qui suit, un identificateur préfixé par le caractère « * » dénote un symbole de variable dans la syntaxe des clauses de Horn.

Une transition tr=(t, pré, post, cond) est interprétée par une clause de Horn sans condition de la forme:

tr(*t, pre(*pre), post(*post), cond(*cond)).

où les ensembles de préconditions (resp. postconditions) ainsi que les conditions sur les données sont interprétés par des listes de préconditions, de postconditions et de conditions sur les données respectivement.

Exemple: nous interpréterons la transition définie par la syntaxe suivante

```
trans   send(*site_i)
pre:    idle(*site_i)
post:   request(*site_i, *site_j)
        wait(*site_i)
cond:   dest(*site_i, *site_j)
```

par la clause de Horn sans condition:

```
tr(send(*site_i),
   pre(idle(*site_i).nil),
   post(request(*site_i,*site_j).wait(*site_i).nil),
   cond(dest(*site_i,*site_j).nil)).
```

Le marquage initial est interprété par une liste de prédicats

Le franchissement de la transition *transition donne le marquage *m' défini par l'équation:
*m' = *m - *pre/θ ∪ *post/θ
Le franchissement d'une transition défini par l'équation ci-dessus est interprété par la règle d'inférence:

```
tir(*m,*t,*m')- tr(*t,pre(*pre),post(*post),cond(*cond))
                - interprete(*cond)
                - enlever(*pre,*m,*m'')
                - inserer (*post,*m'',*m').
```

Les prédicats utilitaires **enlever** et **inserer** sont définis comme suit:

1. **enlever(*l1, *l2, *l)** est vrai si en enlevant les éléments de la liste *l1 à la liste *l2 on obtient la liste *l.

La logique du temps linéaire correspond à la nature des programmes déterministes, tandis que la logique du temps arborescent est le formalisme adéquat pour l'étude des programmes non-déterministes **[Lam 80]**.
Comme la plupart des systèmes réactifs sont non-déterministes, nous ne considérons dans cet article que la logique temporelle arborescente et plus particulièrement la logique temporelle arborescente CTL (Conditonal Temporal Logic). Cette dernière spécifie des contraintes temporelles telles que:

« *il est toujours vrai que ...* »,
« *il existe un instant dans le futur où ...* »,
« *il a été vrai, au moins une fois dans le passé, que ...* »,
« *P doit être toujours vrai, jusqu'à ce que ...* »,
« *P était toujours vrai, depuis que ...* »,
. . .

IV.2 SEMANTIQUE DE LA LOGIQUE TEMPORELLE ARBORESCENTE CTL

La suite des états d'un programme est interprétée comme un modèle de la logique CTL constitué des éléments suivants:

a) S: un ensemble fini d'*états*. Chaque état représente une configuration de l'ensemble du système
b) Init: le *prédicat initial*. Il caractérise l'état initial du système
c) T: une relation finie de transitions

V. INTERPRETATION DES OPERATEURS DE LA LOGIQUE CTL

Nous utilisons la programmation logique dans la définition des algorithmes de *model-*

```
ALL(*p,*q,*l,*s):- in(*s,*l) - //.
ALL(*p,*q,*l,*s):- *q(*s) - NOT(*p(*s)) -//.
ALL(*p,*q,*l,*s):- *q(*s)
                  - PRETILDA(ALL(*p,*q),*s.*l,*s).
```

in(*s,*l) est vrai si la variable *s est un élément de la liste *l.

$s \models INEV(p,q) \Leftarrow s \models q$ (6.1)

$s \models INEV(p,q) \Leftarrow s \models p \wedge PRETILDA\ INEV(p,q)$ (6.2)

```
INEV(*p,*q,*l,*s):- out(*s,*l) -*q(*s) - //.
INEV(*p,*q,*l,*s):- out(*s,*l) - arc(*s,*t,*s') - // - *p(*s)
                  - PRETILDA(INEV(*p,*q),*s.*l,*s).
```

$s \models SOME(p,q) \Leftarrow s \models q \wedge \neg p$ (4.1)

$s \models SOME(p,q) \Leftarrow s \models q \wedge PRE\ SOME(p,q)$ (4.2)

```
SOME(*p,*q,*l,*s):- in(*s,*l) - //.
SOME(*p,*q,*l,*s):- *q(*s)
                  - NOT(arc(*s,*t,*s')) - //.
SOME(*p,*q,*l,*s):- *q(*s) - NOT(*p(*s)) - //.
```

[redacted]

V.3 SIMPLIFICATIONS

```
POT(*p,*q,*s):- POT(*p,*q,nil,*s).
```

- la variable a_i est modifiée uniquement par G; a_i vaut **vrai** si la ressource a été accordée à D_i, ce dernier ne l'ayant pas encore rendue au gestionnaire.

Nous pouvons alors *spécifier formellement* les comportements corrects du système en énonçant des propriétés souhaitables, exprimables par des *formules de logique temporelle*.

P_1 " la ressource est accordée à au plus un demandeur à la fois "
Soit init(*s) le prédicat qui dénote l'état initial. P_1 s'exprime alors par la formule:
init $\Rightarrow$ ALL $\neg (a_i \wedge a_j)$ $i \neq j$
Il s'agit d'une propriété d'exclusion mutuelle, exprimant l'invariance d'une certaine condition ($\neg (a_i \wedge a_j)$). De manière générale, les *propriétés d'invariance* s'exprimeront comme sur notre exemple à l'aide de l'opérateur ALL.

P_2 " Si le processus D_i demande la ressource, il l'obtiendra "
s'exprime par la formule:
init $\Rightarrow$ ALL$(d_i \Rightarrow$ INEV $a_i)$

P_3 " Si le processus D_i obtient la ressource, il la relâchera "
s'exprime par la formule:
init $\Rightarrow$ ALL$(a_i \Rightarrow$ INEV $\neg d_i)$

P_2 et P_3 sont des propriétés de *vivacité*. Ces propriétés s'expriment en général à l'aide d'un opérateur INEV.

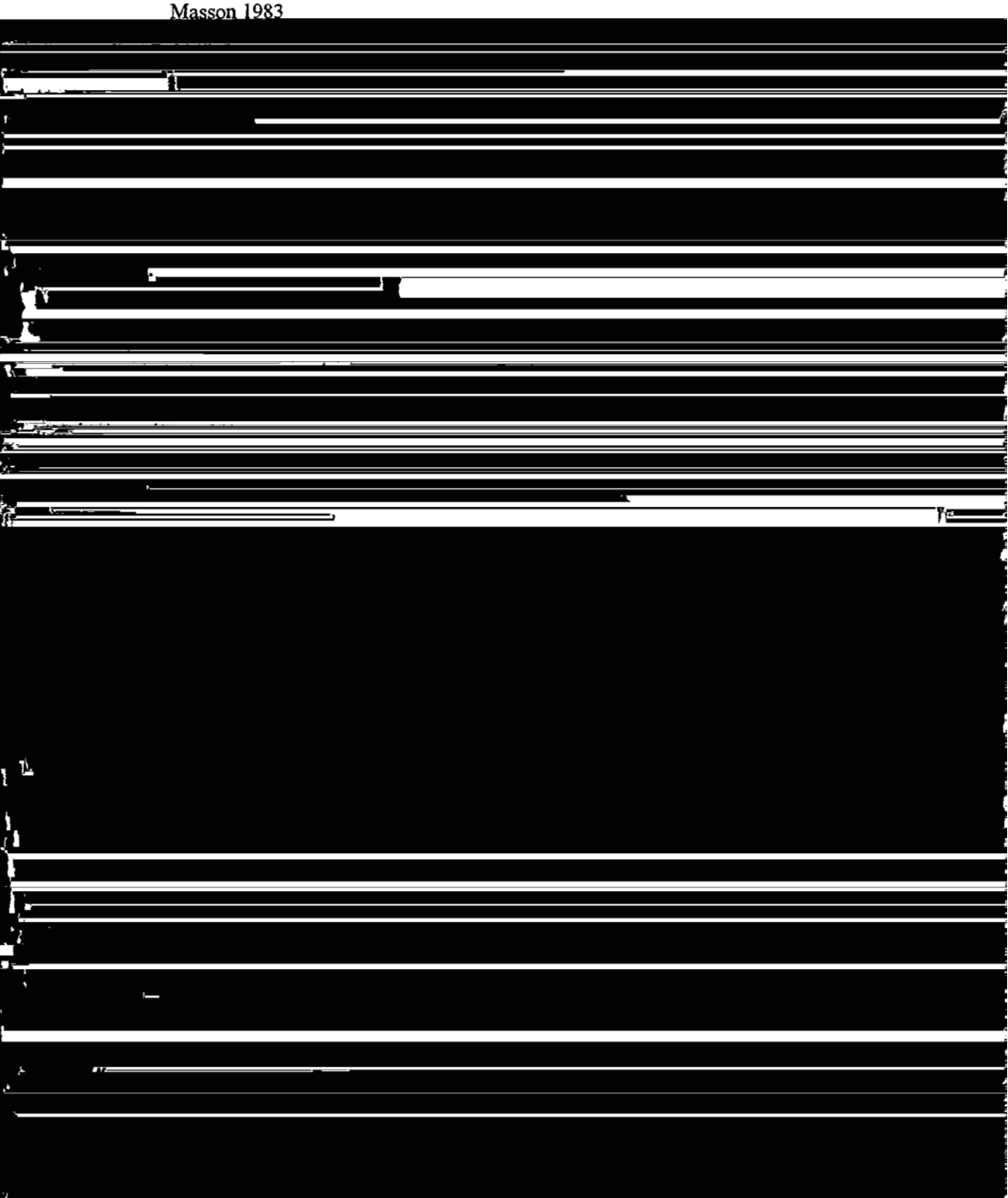
VII. CONCLUSION

Nous avons montré dans cet article comment la logique temporelle pouvait être un bon

Bak 83 J. W. Bakker, J. A. Bergstra, J. J. C. Klopp
Linear time and branching time semantics for recursion with merge.
Proc. ICALP 83, LNCS, #154, pages 39-51, Springer-Verlag, 1983

BENA 83 M. Ben-Ari, A. Pnuelli, Z. Manna
The temporal logic of branching time.
Acta Informatica, volume 20, 1983

BRAMS 83 G. W. BRAMS
Réseaux de Petri: théorie et pratique.
Masson 1983



Pnue 85 A. Pnuelli
Linear and branching structures in the semantics and logics of reactive systems.
12th ICALP, LNCS, pages 15-32, Springer-Verlag, 1985

Pnue 86a A. Pnuelli
Specification and development of reactive systems.
In Proc. IFIP'86, pp 845-858, North Holland, 1986

Pnue 86b A. Pnuelli
Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends.
LNCS #224, Current trends in concurrency, pages 510-584, Springer-Verlag, 1986

Pratt 76 Pratt V.R
Semantical considerations on Floyd-Hoare logic.
Proc. of the 17th Symp. on Found of Computational Science, pp 109-121, 1976

Queil 81 J. P. Queille, J. Sifakis
Specification and verification of concurrent systems in CESAR.
Proc. 5th International Symposium in Programming, 1981

Salwicki 70 Salwicki A.
Formalized algorithm language.
Bull. Ac. Pol. Sc., 18(5), pp 227-232, 1970

Sis 82 A. P. Sistla, E. M. Clarke
The complexity of propositionnal linear temporal logics.
10th Symposium on Principles of Programming Languages, Austin, Texas, pages 159-168, 1982