De la transformation de prédicats à la transformation de programmes

Dominique Méry
Institut Universitaire de France
Université Henri Poincaré Nancy 1
CRIN URA 262 du CNRS
BP 239
54506 Vandœuvre-lès-Nancy
(France)
email: mery@loria.fr

1er février 1996

Résumé

Les transformateurs de prédicats ont été introduits par Dijkstra et ont permis de fonder la théorie des preuves de programmes et la théorie de l'affinement des systèmes d'actions. Ce texte analyse le lien existant entre les transformateurs de prédicats et les systèmes d'actions affinés. Les hypothèses d'équité sont prises en compte et une uniformisation de la présentation permet de démontrer le lien fort entre le raffinement d'actions ou de programmes et le raffinement logique.

Table des matières

1 Introduction

La transformation de programmes recouvre des aspects multiples mais il s'agit de donner une version s'appuyant sur des notions sémantiques claires et rigoureuses. Un programme est une entité formelle ayant des propriétés qui la distinguent des autres textes formels que sont les formules d'un langage logique. Ce document traite de l'analyse de textes formels qui sont soit des programmes, soit des formules, soit un mélange des deux. La question fondamentale est de donner un cadre rigoureux à la notion de transformation de programmes. Rappelons que de telles transformations existent déjà notamment dans le cas de techniques de compilations par exemple. Un compilateur transforme un texte formel en un autre texte formel: la relation entre les deux textes garantit la conservation des propriétés. Partant de ce constat et de cette propriété minimale, nous fondons les transformations de textes sur l'idée de préservation de propriétés. La question est alors de définir un langage de propriétés et de lier les propriétés aux programmes.

La transformation de programmes est souvent appelée *raffinement de programmes*, ou encore, *affinement de programmes*. Le mélange de textes de différente nature comme les programmes et les spécifications a conduit à différentes approches. Le calcul du raffinement de Back et all [3, 4, 5, 28] consiste à définir une relation de raffinement sur les programmes ou actions et à intégrer les spécifications pré-post comme des actions particulières. L'approche suivie par Lamport [13, 14] avec TLA consiste à considérer tout programme comme une formule temporelle particulière: en fait, tout texte est une formule de logique temporelle et le raffinement est la relation de déduction sur les formules de la logique. Enfin, l'approche UNITY [11] est hybride aux deux précédentes car il y a deux phases successives: une première phase consiste à construire un premier ensemble d'actions à partir du raffinement d'une spécification temporelle et une seconde phase consiste à raffiner les programmes à l'aide de transformations appelées *superposition*

Les propriétés des programmes sont de différente nature et reposent sur des sémantiques différentes et ne décrivant pas toujours la même idée sous-jacente. Deux classes de propriétés sont principalement visées dans notre travail. D'une part, les propriétés d'invariance ou *safety* et, d'autre part, les propriétés de *fatalité*. Une propriété d'invariance exprime le fait que *rien de mal ne peut arriver* mais il est clair que,

si rien n'arrive dans un système, alors les propriétés d'invariance sont satisfaites. La propriété de fatalité exprime qu'une propriété finit par être satisfaite au cours du calcul et ce type de propriété est très difficile à démontrer, puisqu'elle repose généralement sur la découverte de relation dite bien fondée. Les logiques temporelles [15, **?**] permettent d'exprimer ces différentes propriétés et conduisent à la mise au point à la fois de méthodes de spécification comme TLA$^+$ [14], à des méthodes de vérification et à des méthodes rigoureuses de construction [11]. L'émergence de ces logiques et systèmes formels a conduit naturellement des expérimentations sur les outils permettant d'aider au mieux l'utilisateur dans ses recherches de preuves. Un certain nombre de travaux ont conduit à une mise en œuvre d'outils interactifs d'aide à la preuve pour des logiques assez puissantes et pour des besoins plus ou moins industriellement applicables. La méthode B [1], par exemple, fournit à l'utilisateur un cadre rigoureux de développement de programmes en se fondant sur la notation AMN et sur la démarche du développement pas-à-pas par preuve d'obligations afin de maintenir un invariant ou de le renforcer: l'Atelier B [2] aide l'utilisateur à développer ses machines abstraites, en lui résolvant la preuve de certaines preuves par le biais d'un prouveur astucieux et modeste [9, 8, 7]. Cette méthode B a l'avantage d'avoir été utilisée par un vrai milieu industriel soucieux de fournir un sécurité à dans leurs produits. Nous aurions pu citer aussi la méthode Z ou la méthode VDM qui sont des méthodes de spécifications fondées sur les mêmes idées que B mais B est une méthode plus large au niveau du logiciel. La limitation de B concerne le type de propriétés concernées car seuls les invariants sont considérés et une preuve de terminaison est nécessaire pour le raffinement d'une opération à l'aide d'une itération.

Notre démarche consiste à mettre en avant la thèse suivante: *Un texte (programme ou spécification) est affiné en un texte Q (programme, spécification), si toute propriété de P est une propriété de Q*. Cette thèse repose sur la définition de ce qu'est une propriété et sur le lien entre une propriété et un texte. Notre document montrera que ce lien repose sur des systèmes formels et sur des transformateurs de prédicats et nous envisagerons les limites de notre approche.

Le document est organisé selon les sections suivantes. Une section introduit la terminologie relative à la sémantique des langages, leur syntaxe et les propriétés de programmes et décrit les transformateurs

de prédicats. La section suivante montre comment ces transformateurs aident à la construction de systèmes de preuves sémantiquement complets et corrects. Une section montre les applications au cas de l'affinement de programmes fondé sur les transformateurs de prédicats.

2 Transformateurs de prédicats

Cette section vise à rappeler des résultats que nous avons obtenus dans le passé et à montrer comment se situe la thèse que nous défendons au travers de ce document mais aussi des autres documents présentés. Nous rappelons les notions d'action et de programme et nous fixons ces notions syntaxiquement et sémantiquement. L'affinement de spécifications et de programmes est lié à cet aspect.

Définition 1 *Syntaxe et sémantique des actions*

Soit $\mathcal{L}$ un langage de formules (langage des prédicats du premier ordre avec sortes). Soit $\mathcal{A}$ le langage des actions.

- *Pour toute action α de $\mathcal{A}$ et pour toute formule φ de $\mathcal{L}$, $\alpha \bullet \varphi \in \mathcal{L}$ et $\alpha \bullet \varphi$ est la plus faible précondition de φ par rapport à α (incluant à la fois la terminaison et la correction partielle).*

- *$\mathcal{A}$ est inductivement défini pour sa syntaxe et sa sémantique selon les cas suivants:*

  1. $(X := E) \bullet \varphi \stackrel{def}{=} \varphi[e/x]$
  2. $(\alpha \sim g) \bullet \varphi \stackrel{def}{=} [g \wedge (\alpha \bullet \varphi)] \vee [\neg g \wedge \varphi]$
  3. $(g \longrightarrow_{\wedge} \alpha) \bullet \varphi \stackrel{def}{=} [g \wedge (\alpha \bullet \varphi)]$
  4. $(g \longrightarrow_{\Rightarrow} \alpha) \bullet \varphi \stackrel{def}{=} [g \Rightarrow (\alpha \bullet \varphi)]$
  5. $\left(\alpha_1 \,\blacksquare_{\vee} \ldots \blacksquare_{\vee} \alpha_n\right) \bullet \varphi \stackrel{def}{=} [\alpha_1 \bullet \varphi] \vee \ldots \vee [\alpha_n \bullet \varphi]$
  6. $\left(\alpha_1 \,\blacksquare_{\wedge} \ldots \blacksquare_{\wedge} \alpha_n\right) \bullet \varphi \stackrel{def}{=} [\alpha_1 \bullet \varphi] \wedge \ldots \wedge [\alpha_n \bullet \varphi]$
  7. $(\texttt{l(op)}(\alpha_1, \ldots, \alpha_n) \bullet \varphi \stackrel{def}{=} op\,(\alpha_1 \bullet \varphi, \ldots, \alpha_n \bullet \varphi)$

l(op) est un combinateur associé à une combinaison logique. La signification de $\alpha \bullet \varphi$ inclut l'absence d'erreurs à l'exécution et nous pouvons définir une condition d'activation pour chaque action $\mathcal{A}$ par $cond(\alpha) \stackrel{def}{=} \alpha \bullet \mathsf{true}$. La signification d'une action α est donnée relativement à un langage de formules, $\mathcal{L}$, mais l'interprétation du langage $\mathcal{L}$ est supposée clairement établie. Afin d'aider l'utilisateur dans sa tâche de développement d'une preuve, nous avons développé un environnement de preuves de propriétés de programmes dans le cadre d'un projet en collaboration avec le CNET[24, 25, 22]. La codification du système avait été réalisée à l'aide du prouveur générique Isabelle de Paulson [30]. Une autre expérimentation nous a conduits à développer un assistant à la preuve pour UNITY mais dans le prouveur B [9, 8, 7]. En relation avec la notion d'action, Lamport [12] considère une action comme une relation entre des variables et la version primées de ces variables; il interdit qu'une action soit l'identité puisque ce cas est réservé au bégaiement. On peut effectiveement lier cette approche à la nôtre mais nous avons remarqué que la profusion des systèmes d'actions méritait une unification des écritures. Ainsi, une action ou une opération au sens de la méthode B est exécutée si la précondition est vraie sinon il ne se passe rien. La garde des systèmes d'actions de Back conduit à un miracle possible et, en fait, il s'agit d'une implication au sens de la garde. Ces différences sont trè s importantes lorsqu'on traite un exemple et que les systèmes peuvent se bloquer. La correspondance entre la sémantique opérationnelle et la sémantique par wp des actions est donnée par la propriété suivante de M. Bonsangue et J.N. Kok [6].

Propriété 1 *Let* $\alpha \in \mathcal{A}$ *and* $\varphi \in \mathcal{L}$. $(\alpha \bullet \varphi) \equiv \left(\begin{array}{l} \forall x' \in S : (\{\![x]\!\}\ \alpha\ \{\![x]\!\}) \Rightarrow \varphi(x') \\ \wedge \\ \exists x' \in S : \{\![x]\!\}\ \alpha\ \{\![x]\!\} \end{array} \right)$

Ce résultat nous conduit à montrer les lien entre la définition des actions par leur sémantique par wp et l'approche relationnelle de Lamport dans TLA$^+$. Mais, nous devons d'abord construire la classe des programmes. Un programme est un ensemble d'actions exécutées selon une politique d'ordonnancement choisie comme une hypothèse d'équité portant sur les traces et pas seulement sur les états seuls.

Définition 2 *Syntaxe des programmes*

Soit $\mathcal{A}$ la classe des actions. Un programme sur $\mathcal{A}$ est défini par une structure $(Dec, Init, Act, Pol)$:

1. *Dec est une liste de déclarations de la forme $x : type_x$.*
2. *$Init$ spécifie les conditions initiales des variables.*
3. *Act est un ensemble fini d'actions de la forme $\{\alpha_1 \ldots \alpha_n\}$.*
4. *Pol est une indication de la politique d'ordonnancement choisie (équité globale, justice, équité)*

La classe des programmes est $\mathcal{P}$ et un programme individuel est noté par la métavariable π, π_1, ..., π_n, π'. π est canoniquement représenté par $(Dec(\pi), Act(\pi), Pol(\pi))$. La classe des programmes UNITY est incluse dans cette classe de programmes, en précisant que les actions sont choisies selon un critère de choix juste. Les calculs sont définis comme des transformations d'actions sur les variables e l'ensemble des traces satisfaisant $Pol(\pi)$.

Définition 3 *Sémantique opérationnelle des programmes*
La sémantique opérationnelle de π dans $\mathcal{P}$ est une structure $O(\pi)$ définie par
$(\Sigma_\pi, \{\{\![]\!\} \ \alpha \ \{\![]\!\} : \alpha \in Act(\pi)\}, Init_\pi, Trace(\Sigma_\pi, Act(\pi), Pol(\pi)))$ *where*

- $\Sigma_\pi = Dec(\pi) \longrightarrow M(\pi)$, *($M(\pi)$ désigne les données de π)*
- *pour toute* $\alpha \in Act(\pi)$, $\{\![]\!\} \ \alpha \ \{\![]\!\} = \{(\sigma, \sigma') \in (\Sigma_\pi)^2 : (\alpha \bullet \{\sigma'\})(\sigma)\}$
- $(\sigma_i)_{i\in\omega} \in Trace(\Sigma_\pi, Act(\pi), Pol(\pi))$:
  1. $\sigma_0 \in Init_\pi$
  2. $\forall i \in \omega : \exists \alpha_i \in Act(\pi) : \{\![\sigma_i]\!\} \ \alpha_i \ \{\![\sigma_i]\!\}$
  3. $(\sigma_i)_{i\in\omega}$ *satisfait* $Pol(\pi)$

Puisque nous pouvons lier la sémantique d'une action avec une précondition et la sémantique relationnelle d'une action, nous pouvons définir une sémantique fondée sur la notion de précondition pour un

programme π. La motivation principale de ce choix réside dans la définition de la notion d'affinement selon Back [3, 4], puisque un transformateur de prédicat associe à un programme une assertion et le raffinement est exprimé sous la forme d'une déduction. Une sémantique par trasnformateur de prédicats pour une programme π de $\mathcal{P}$ est constituée des éléments ci-dessous.

Définition 4 *Sémantique par wp des programmes*

Une sémantique par transformateur de prédicats pour un programme π de $\mathcal{P}$ est une collection de transformateurs de prédicat, $\{\alpha_i\bullet : \alpha_i \in Act(\pi)\}\cup\{\pi\bullet\}$, une collection d'hypothèses pour π, $\{Init_\pi, Term_\pi, Block_\pi\}$, et une théorie Th for π:

1. *$Init_\pi$ spécifie les conditions initiales de π.*
2. *$Term_\pi$ spécifie les conditions terminales de π.*
3. *$Block_\pi$ spécifie les conditions de blocage pour π.*

La relation entre la sémantique opérationnelle des programmes et la sémantique par transformateurs de prédicats n'est pas toujours facile à démontrer. L'existence de transformateurs de prédicats repose sur leur construction effective à partir de la sémantique opérationnelle [29, 16, 17, 18, 10]. Nous utiliserons la notation suivante pour désigner la sémantique par wp du programme π : $PT_\pi = (WP_\pi, AS_\pi, Th_\pi)$.

3 Systèmes de preuves de programmes

L'expression *preuves de programmes* est utilisée pour exprimer le fait qu'on démontre des propriétés sur les programmes à l'aide de systèmes de preuves. Notre cadre se limitera aux deux types de propriétés que sont d'une part les propriétés d'invariance et, d'autre part, les propriétés de fatalité. Une propriété d'invariance exprime que rien de mal ne peut arriver et sa définition sémantique ne repose que sur les transitions. Une propriété de fatalité exprime que quelque chose de souhaitable arrivera et sa sémantique nécessite les traces d'exécution. Deux qualités sont souhaitables pour ces deux types de propriétés: la correction de la méthode utilisée et la complétéde sémantique. Nos résultats

antérieurs [16, 17, 18, 22] se sont fondés sur la définition et la construction de transformateurs de prédicats ad hoc, pour prendre en compte la notion d'équité (faible ou forte). Dans le cas des systèmes temporels de preuves de propriétés de programmes, nous soulignerons le cas de la méthode de preuve d'UNITY dont l'axiome de substitution introduit une inconsistance dans le raisonnement. En fait, le système axiomatique doit s'appuyer sur les transformateurs de prédicats pour être sémantiquement complet et correct. D'une certaine mesure, le transformateur de prédicat décrit le système d'une manière synthétique. L'utilisation du système peut alors conduire à lui ajouter d'autres règles non nécessaires pour la complétude mais terriblement utiles pour les preuves.

Définition 5 *Invariants pour* π
$Invariant(\pi)$ *est l'ensemble des invariants de* π*, i.e.:*
$I \in Invariant(\pi)$ *si, et seulementsi,*

1. $Th_\pi \models Init_\pi \Rightarrow I$
2. *Pour tout* α *de* $Act(\pi)$*:* $Th_\pi \models (I \wedge cond(\alpha)) \Rightarrow \alpha \bullet I$

Le plus fort invariant joue un rôle important dans la preuve de complétude sémantique de la méthode.

Définition 6 *Expression des propriétés*
Soit π *un programme,* $\mathcal{L}$ *un lanage de formules,* φ*,* ψ *des formules de* $\mathcal{L}$*,* $\alpha \in Act(\pi)$*.*

1. $\{[\varphi]\}\ \alpha : \pi\ \{[\varphi]\} \stackrel{def}{=} [Th_\pi \models (\varphi \Rightarrow \alpha \bullet \psi)]$
   La propriété $\{[\varphi]\}\ \alpha : \pi\ \{[\varphi]\}$ *est vraie pour* Th_π*, si* α *est une action de* π*. Elle est appelée une propriété de transition ou d'action.*
2. $Inv(\varphi, \pi)(I) \stackrel{def}{=} \begin{cases} (1)\ Th_\pi \models \varphi \Rightarrow I \\ (2)\ \texttt{Pour tout}\ \alpha\ \texttt{in}\ Act(\pi) : Th_\pi \models (I \wedge cond(\alpha)) \Rightarrow \alpha \bullet I \end{cases}$
   I est vraie à partir des états satisfaisant φ *et aussi longtemps qu'une action est activable. C'est une propriété d'invariance conditionnelle.*

3. $\varphi \stackrel{\pi}{\longmapsto} \psi \stackrel{def}{=}$

$$\exists I \in \mathcal{L} : \begin{cases} (1)\ Inv(Init_\pi, \pi)(I) \\ (2)\ Th_\pi \models (\varphi \wedge I) \Rightarrow \pi \bullet \psi \end{cases}$$

Si φ est vrai au cours du calcul de π, alors ψ sera fatalement vraie. Elle est appelée une propriété de fatalité.

L'étude des propriétés de fatalité nous a conduits à mettre en évidence les transformateurs de prédicats convenant à la notion de fatalité dans le cas de l'équité faible ou forte. Nous rappelon les cas de l'équité faible ou justice, pour illustrer notre propos. Il va de soi que notre fatalité de base est en fait appelée propriété de type *ensures* dans le cas d'UNITY et que cette notion est la clé pour obtenir la complétude sémantique et la correction. Le travail fait sur les transformateurs de prédicats est payant.

Définition 7 *Fatalité basique*

1. $OP(G)(\varphi, \psi) \stackrel{def}{=}$ `Pour tout` $\alpha\ in\ Act(\pi) : Th_\pi \models (\varphi \wedge \neg\psi \wedge cond(\alpha)) \Rightarrow \alpha \bullet \psi$ `et il existe` $\alpha\ in\ Act(\pi) : Th_\pi \models (\varphi \wedge \neg\psi) \Rightarrow cond(\alpha)$

   $OP(G)(\varphi, \psi)$ exprime le fait que toute action exécutée conduira à un état satisfaisant ψ et que les états ne sont pas bloqués.

2. $OP(J)(\varphi, \psi) \stackrel{def}{=}$

$$\begin{cases} (1)\ \texttt{Pour tout}\ \alpha\ in\ Act(\pi) : Th_\pi \models (\varphi \wedge \neg\psi \wedge cond(\alpha)) \Rightarrow \alpha \bullet (\varphi \vee \psi) \\ (2)\ \texttt{Il existe}\ \alpha\ in\ Act(\pi) : Th_\pi \models (\varphi \wedge \neg\psi) \Rightarrow \alpha \bullet \psi \end{cases}$$

   Au moins une action de π est activable à partir de $\varphi \wedge \neg\psi$, et toute activation à partir de $\varphi \wedge \neg\psi$ conduit à un état satisfaisant $\varphi \vee \psi$.

L'opérateur $OP(J)$ permet de montrer que la notion d'action critique est très importante pour assurer la convergence au niveau des propriétés de fatalité et cet opérateur ressemble à l'opérateur *ensures* de la logique d'UNITY. La méthode B [1] permet de démontrer des propriétés d'invariance de manière incrémentale à l'aide d'un outil de preuve. Un système est caractérisé par un ensemble d'actions mais il n'y a pas d'hypothèse d'exécution : toute action doit terminer et cet aspect est très difficile à démontrer au cours du raffinement.

4 Applications

Nous appliquons les points précédents à l'affinement de programmes et de spécifications et au développement de programmes. *L'affinement* d'un programme P en un programme Q est une transformation conduisant à préciser le calcul de P par un renforcement.

 : *P est affiné en Q, si toute propriété de P est une propriété de Q.*

Cette définition peut aussi être considérée comme un pricnipe général des relations de raffinements. La question à se poser est de donner une description de la classe des propriétés associées à chaque programme. Le calcul de raffinement de Back et all satisfait ce principe mais, pour cela, les transformateurs de prédicats demeurent les outils intermédiaires pour valider le principe.

Définition 8 *Calcul de raffinement*

$P \sqsubseteq_{cal-raf} Q \stackrel{def}{=} \forall \in \mathcal{L} : wp(P)(\varphi) \Rightarrow wp(Q)(\varphi)$

Propriété 2 $P \sqsubseteq_{cal-raf} Q$ *si, et seulement si, si P est totalement correct , alors Q est totalement correct.*

Une propriété se réduit à la correction totale d'un programme. Il se peut que le programme que l'on décrit ne termine pas et les propriétés sont alors plus générales : il s'agit des propriétés de fatalité. La logique temporelle est un cadre d'expression très naturel depuis les travaux de Pnueli [31].

Définition 9 $P \sqsubseteq_{raf-fat} Q$ *si, et seulementsi, toute propriété d e fatalité de P est une propriété de fatalité pour Q.*

Propriété 3 $P \sqsubseteq_{raf-fat} Q$ *si, et seulement si,* $\begin{cases} (1)\ invariant(P) \subseteq invariant(Q) \\ (2)\ WP(P) \Rightarrow WF(Q) \end{cases}$

Nous avons développé plusieurs études de cas montrant comment ce type de relations pouvait être utilisée en pratique. Le document de notre thèse d'état [22] présente des études cas assez classiques comme

l'exclusion mutuelle et les solutions centralisées ou distribuées de cette approche. Une autre étude de cas concernant le ramasse-miettes à la volée [23] a permis d'illustrer le modèle de développement plus général que nous avions présenté dans [22] et la découverte de nouvelles solutions par des choix différents de stratégies de développement nous a conduit à une documentation formelle des solutions.

5 Conclusion et perspectives

Nos travaux ont porté sur des aspects théoriques mais ils se sont appuyés sur des outils [19, 24, 21, 20, 25, 26, 10, 22, 27] et des expérimentations. Les outils ne sont pas des produits diffusables pour des raisons de manque de moyens en ingénieur mais ils permettent de montrer que notre thèse sur l'affinement est inhérente à nos systèmes formels.

Les perspectives viseront à développer cette thèse au travers d'autres systèmes et logiques comme TLA [13, 14], UNITY [11], MTL [27], B [1], ... Les travaux concerneront à la fois les outils mais aussi les domaines d'applications comme les services dans le cadre des réseaux intelligents.

Références

[1] J. R. Abrial. *The B book*. Cambridge University Press, 1995. to appear.

[2] GEC ALSTHOM TRANSPORT DIGILOG SNCF INRETS RATP. *Atelier B, Version 2.0, Manuel de Référence du Langage B*. DIGILOG, 1995.

[3] R. J. R. Back. On correct refinement of programs. *Journal of Computer and System Sciences*, 23(1):49–68, 1979.

[4] R. J. R. Back. Correctness preserving programs refinements: proof theory and applications. Mathematical Centre Tracts 131, Mathematical Centre, Amsterdam, 1980.

[5] R. J. R. Back. A method for refining atomicity in parallel algorithms. In E. Odijk, M. Rem, and J. C. Syre, editors, *Parallel Architectures and Languages Europe:PARLE*, pages 199–219. Springer-Verlag, june 1989. lncs 366.

[6] M. Bonsangue and J. N. Kok. Semantics, orderings and recursion in the weakest precondition calculus. Technical report, CWI, 1992.

[7] BP Innovation Centre and Edinburgh Portable Compilers Ltd. *B-Tool Version1. 1, Reference Manual*, 1991.

[8] BP Innovation Centre and Edinburgh Portable Compilers Ltd. *B-Tool Version1. 1, Tutorial*, 1991.

[9] BP Innovation Centre and Edinburgh Portable Compilers Ltd. *B-Tool Version1. 1, User Manual*, 1991.

[10] N. Brown and D. Méry. A proof environment for concurrent programs. In J. C. P. Woodcock, editor, *FMÉ93: Industrial-Strength Formal Methods*, pages 196–215. IFAD, Springer-Verlag, 1993. LNCS 670.

[11] K. M. Chandy and J. Misra. *Parallel Program Design A Foundation*. Addison-Wesley Publishing Company, 1988. ISBN 0-201-05866-9.

[12] L. Lamport. Introducing TLA$^+$. Technical report, Digital Equipment Corporation, 1 september 1991. Preliminary Draft.

[13] L. Lamport. A temporal logic of actions. *Transactions On Programming Languages and Systems*, 16(3):872–923, May 1994.

[14] L. Lamport. TLA$^+$. Technical report, Digital Equipment Corporation, 5th july 1995.

[15] Z. Manna and A. Pnueli. *The temporal logics of reactive and concurrent systems*. Springer-Verlag, 1992.

[16] D. Méry. *Une Méthode Axiomatique de Preuves de Propriétés de Fatalité de Programmes Paralleles avec Hypothèses d'Execution équitable*. Thèse de troisième cycle, Institut National Polytechnique de Lorraine, 1983.

[17] D. Méry. A proof system to derive eventuality properties under justice hypothesis. In *LNCS*, number 233 in Lecture Notes in Computer Science. Mathematical Foundations of Computer Science, 1986. Bratislava, Tchecoslovaquie.

[18] D. Méry. Méthode axiomatique pour les propriétés de fatalité des programmes parallèles. *RAIRO Informatique Théorique et Application*, 21(3):287–322, 1987.

[19] D. Méry. The nu system as a development system of concurrent programs: δ-nu. In C. Mauduit, editor, *VIIIième Journées du PRC Mathématiques - Infomatique*. Springer Verlag, 1990.

[20] D. Méry. The $\backslash\sqcap$ system as a development system for concurrent programs: $\delta\backslash\sqcap$. *Theoretical Computer Science*, 94(2):311 – 334, march 1992.

[21] D. Méry. Proving and developing concurrent programs: a small system. In C. M. I. Rattray and R. G. Clark, editors, *Proceedings of the IMA conference on the Unified Computation Laboratory*. The IMA, OXFORD UNIVERSITY PRESS, 1992.

[22] D. Méry. *Une Méthode de Raffinement et de Développement pour la Programmation Parallèle*. Doctorat ès sciences mathématiques, Université de Nancy 1,UFR STMIA, DFD Informatique, Février 1993.

[23] D. Méry. Refining solutions of the on the fly garbage collection from formal specifications. Technical report, CRIN, 1995.

[24] D. Méry and A. Mokkedem. A proof environment for a subset of sdl. In O. Faergemand and R. Reed, editors, *Fifth SDL Forum Evolving methods*. North-Holland, 1991.

[25] D. Méry and A. Mokkedem. Crocos: An integrated environment for interactive verification of sdl specifications. In G. Bochmann, editor, *Computer-Aided Verification Proceedings*. Springer Verlag, 1992.

[26] D. Méry, A. Mokkedem, and D. Roegel. CROCOS : un environnement de preuve interactive de spécifications SDL . Technical

Report numéro de convention 89-58 00 790 92 45/PAA, C entre de R echerche en I nformatique de N ancy, 1992.

[27] A. Mokkedem and D. Méry. On using temporal logic for refinement and compositional verification of concurrent systems. *Theoretical Computer Science*, 140, April 1995.

[28] C. Morgan. *Programming from Specifications*. Prentice Hall International Series in Computer Science. Prentice Hall, 1990.

[29] D. Park. A predicate transformer for weak fair iteration. In IBM, editor, *Proceedings of the IBM conference*, pages 259–275. IBM, IBM, 1981.

[30] L. C. Paulson. *The Isabelle Reference Manual*. Computer Laboratory, University of Cambridge, first edition, 1994.

[31] A. Pnueli. The temporal logics of programs. In *Proceedings of 18th IEEE Symposium on Foundations of Computer Science*, pages 46 – 57. IEEE, 1977.