Modèles et algorithmes

Event B

1er mars 2021

Dominique Méry

Université de Lorraine (Telecom Nancy)
LORIA, BP 239,Campus Scientifique
54506 Vandœuvre-lès-Nancy Cedex, FRANCE
dominique.mery@loria.fr
http://members.loria.fr/mery
version 1er mars 2021, commentaires à dominique.mery@loria.fr
Année universitaire 2020-2021

Ce document est édité le 1er mars 2021 pendant le cours MALG et sera, sans doute, modifié et enrichi en fonction des cours et des séances d'exercices. Deux mots dans le titre caractérisent les objectifs :

— *modèles* : nous allons donner des éléments sur les techniques de modélisation formelle avec une vue sur les applications en génie logiciel, notamment en vérification et validation de logiciels, de programmes, de systèmes.
— *algorithmes* : nous allons considérer les questions de construction des algorithmes corrects et nous aborderons les questions de calculabilité, de décidabilité et d'indécidabilité.

Deux mots caractérisent les concepts que nous allons introduire pour les modèles et les algorithmes :

— *abstraction* : une abstraction est une notion très importante quand on veut modéliser un système ; une abstraction désigne à la fois une action et l'objet de cette action. Abstraire un système, c'est en donner une vue détachée de la réalité mais conforme à la réalité du système. Cette notion d'abstraction permet de donner des vues simples mais fidèles des systèmes.
— *raffinement* : un raffinement est une action inverse de l'abstraction et consiste à préciser le modèle en cours d'examen, afin de lui donner des artifices identifiés sur le système en cours de modélisation.

Ces deux notions peuvent avoir d'autres sens mais sont essentielles pour modéliser des systèmes. Nous allons donc envisager des techniques mettant en œuvre ces deux notions et nous allons les utiliser dans ce cours mais aussi dans les cours où la sûreté et la sécurité sont requises dans la mesure où elles sont critiques et peuvent poser des problèmes critiques par rapport aux personnes et aux biens.

Cette édition ajoute la pratique des environnements PAT [67] (permettant d'analyser les systèmes informatiques dans le cadre de langages de programmation comme C# ou encore des formalismes spécifiquement développés pour permettre la construction d'outils de vérification) et Frama-C [34] (permettant l'analyse et la preuve *mécanisée* de programmes C réduit). Elle aborde toujours le langage TLA$^+$ [43, 45, 68] et les outils, ainsi que le langage Event-B [20, 3] et les environnements Rodin [4] et Atelier-B [27].

Les outils et les environnements associés comme Frama-C, PAT, Rodin, Atelier-B, TLA$^+$ Toolbox ... qui sont des environnements formels (F-IDE). Dans la mesure où ces environnements contiennent plus ou moins de fonctionnalités avancées comme une procédure de décision ou un assistant de preuve, nous avons ajouté des éléments de logique dans le chapitre *Modélisation et vérification des programmes et des systèmes*.

TABLE DES MATIÈRES

CHAPITRE 1

EVENT B

Sommaire

La méthode Event B [3, 20] s'appuie sur un langage de modélisation permettant de décrire des modèles à états et les propriétés de sûreté de ces modèles à états. L'objectif principal est de permettre la modélisation incrémentale et prouvée de systèmes réactifs. Elle s'appuie sur un langage intégrant des notations ensemblistes et un calcul des prédicats du premier ordre et offrant la possibilité de définir des modèles de systèmes réactifs, modèles appelés machines ; elle y inclut le concept de raffinement qui exprime la simulation d'une machine par une autre machine. Une machine Event B permet de décrire des systèmes réactifs c'est-à-dire des systèmes réagissant à leur environnement et à ses stimuli. Une propriété importante de telles machines est qu'elles maintiennent un invariant qui décrit l'ensemble des états possibles. Le développement de cette méthode a été réalisé à partir de la méthode classique B [1] et propose un cadre général pour développer des systèmes réactifs en utilisant une démarche progressive de conception des modèles par raffinement. Le raffinement [9, 32, 8, 10] est une relation qui lie deux modèles en exprimant un enrichissement d'un modèle par un autre ; le raffinement est aussi appelé simulation et le raffinement d'un modèle abstrait par un modèle concret signifie que le modèle concret simule le modèle abstrait et que toutes les propriétés de sûreté du modèle abstrait, en particulier l'invariant abstrait, sont préservées. Event B s'attache à exprimer des modèles de système caractérisés par leur invariant et par des propriétés de sûreté. On peut néanmoins considérer les propriétés de fatalité comme UNITY [22] ou TLA$^+$ [44, 43] mais dans un cadre limité.

1.1 Introduction

Le chapitre est organisé en huit sections. La section 2 présente des résultats concernant la modélisation et la vérification de systèmes à l'aide de systèmes de transition. Il s'agit de dopnner les idées sur lesquelles sont fondées l'approche Event B. En particulier, on explique comment sont définies les propriétés d'invariance et les propriétés de sûreté dans le cadre d'un système de transition modélisant un système qui peut être un programme, un algorithme ou un système réparti. Dans la section 3, nous détaillons le langage Event B et les concepts le définissant, comme les événements, les contextes, les machines et le raffinement. Nous donnons aussi une explication des obligations de preuves engendrées pour vérifier la validité de la structure Event B. Puis, dans les sections 4 et 5, nous développons trois études de cas, afin d'illustrer la modélisation incrémentale et prouvée avec la méthode Event B. En particulier, nous mettons en avant la notion de patron de raffinement appliquée à la méthode Event B. La section 6 décrit les outils attachés à cette méthode et nous terminons notre exposé par des éléments sur les travaux en cours et les perspectives de cette méthode.

1.2 Modélisation et vérification d'un système

1.2.1 Modélisation

Un modèle abstrait relationnel $\mathcal{AM}$ ($\mathcal{AM}_{\mathrm{P}}$ d'un programme P ou $\mathcal{AM}_{\mathrm{P}}$ d'un système P) est la donnée d'un espace d'états Σ, d'un ensemble d'états initiaux $Init_{\mathrm{P}}$, d'un ensemble d'états terminaux $Term_{\mathrm{P}}$ et d'une relation binaire $\mathcal{R}$ sur Σ. L'ensemble des états terminaux peut être vide et, dans ce cas, le programme ne termine pas ; cet aspect peut être utilisé pour modéliser les programmes des systèmes d'exploitation qui ne terminent pas et qui ne doivent pas terminer. Nous allons utiliser l'expression système plutôt que

programme, dans la mesure où nous pouvons décrire des objets plus généraux que des programmes au sens informatiques mais aussi que ce formalisme est aussi utilisable pour les applications répartis.
Un système est caractérisé par ses traces d'exécution construites à l'aide du modèle abstrait comme suit :
$s_0 \underset{R}{\longrightarrow} s_1 \underset{R}{\longrightarrow} s_2 \underset{R}{\longrightarrow} s_3 \underset{R}{\longrightarrow} \ldots \underset{R}{\longrightarrow} s_i \underset{R}{\longrightarrow} \ldots$ est une trace engendrée par le modèle abstrait.
L'observation d'un système peut donc être réalisée par l'analyse des traces du système ; Θ_S est l'ensemble de toutes les traces de S. Pour exprimer des propriétés, un langage d'assertions ou un langage de formules est important ; nous notons un langage d'assertions $\mathcal{L}$. Pour simplifier, nous pouvons prendre comme langage d'assertions $\mathcal{P}(\Sigma)$ (l'ensemble des parties de Σ) et $\varphi(s)$ (ou $s \in \varphi$) signifie que φ est vraie en s. Le langage d'assertions permet d'exprimer des propriétés mais il se peut que le langage considéré ne soit pas suffisamment expressif. Dans le cadre de la correction des programmes, nous supposerons que les langages d'assertions sont suffisamment complets (au sens de Cook) et cela veut dire que les propriétés requises pour la complétude sont exprimables dans le langage considéré.
Les propriétés d'un système S sont, en particulier, les propriétés de sûreté et les propriétés de fatalité. Les propriétés de sûreté sont, par exemple, la correction partielle d'un système S par rapport à ses spécifications, l'absence d'erreurs à l'exécution ; les propriétés de fatalité sont, par exemple, la terminaison d'un programme P par rapport à ses spécifications ou la correction totale de P par rapport à ses spécifications. Nous pourrions nous intéresser à d'autres propriétés de programmes comme ses performances mais cela impliquerait des modèles pour exprimer des propriétés dites non fonctionnelles. Les propriétés sont exprimées dans un langage $\mathcal{L}$ dont les éléments sont combinés par des connecteurs logiques ou par des instanciation de variables ; la relation d'implication modulo l'équivalence définit une relation d'ordre partiel.
Nous supposons qu'un système S est modélisé par un ensemble d'états Σ_S, noté Σ, et que $\Sigma \stackrel{def}{=}$ VARIABLES $\longrightarrow$ VALEURS. L'écriture $s \in A$ se traduit en une expression de la forme $s[\![\varphi(x)]\!]$ où x est une liste dont les éléments sont toutes les variables de VARIABLES et exclusivement ces variables ; cela signifie que $s \in A$ signifie de manière équivalente que $\varphi(x)$ est vraie en s. La définition de la validité d'une formule ou d'un prédicat peut être donnée sous une forme inductive de $s[\![\varphi(x)]\!]$.

Exemple 1.1

1. $s[\![x]\!]$ est la valeur de s en x c'est-à-dire $s(x)$ ou encore la valeur de x en s..
2. $s[\![\varphi(x) \wedge \psi(x)]\!] \stackrel{def}{=} s[\![\varphi(x)]\!]$ et $s[\![\psi(x)]\!]$.
3. $s[\![x = 6 \wedge y = x{+}8]\!] \stackrel{def}{=} s[\![x]\!] = 6$ et $s[\![y]\!] = s[\![x]\!]{+}8$.

Nous utilisons des notations simplifiant la référence aux états ; ainsi, $s[\![x]\!]$ est la valeur de x en s et le nom de la variable x et sa valeur ne seront pas distinguées. $s'[\![x]\!]$ est la valeur de x en s' et sera notée x'. Ainsi, $s[\![x = 6]\!] \wedge s'[\![y = x{+}8]\!]$ se simplifiera en $x = 6 \wedge y' = x'{+}8$. La conséquence est que l'on pourra écrire la relation de transition comme une relation liant l'état des variables en s et l'état des variables en s'.

Soient s, s' deux états de l'ensemble VARIABLES $\longrightarrow$ VALS.
$s \underset{R}{\longrightarrow} s'$ s'écrira comme une relation $\mathrm{R}(x, x')$ où x et x' désignent des valeurs de x.
Nous avons introduit la notion de variable primée de la logique temporelle des actions de Lamport [43] et nous pourrions utiliser aussi le système de règles d'inférence et d'axiomes pour modéliser les systèmes concurrents. x' est la valeur après la transition considérée et x est la valeur avant la transition considérée. Ainsi, la condition $\exists y.\mathrm{R}(x, y)$ définit la condition de transition ou la garde ; nous nous intéressons aux expressions particulières de la forme suivante $cond(x) \wedge x' = f(x)$ où $cond$ est une condition sur x et f est une fonction. On peut exprimer les principes d'induction en utilisant les relations entre les variables non-primées et les variables primées. Les conditions initiales sont définies par un prédicat caractérisant les valeurs des variables initialement. Nous proposons donc de définir plus généralement ce qu'est un modèle relationnel d'un système dont on a observé les variables flexibles. Nous avons noté que l'ensemble des états était Σ pour un système donné et nous identifions cet ensemble à l'ensemble des valeurs possibles des variables flexibles x. Nous utiliserons donc la même notation mais VALS sera en fait l'ensemble des valeurs possibles de x.

☼**Définition 1.1** Modèle relationnel d'un système
Un modèle relationnel $\mathcal{MS}$ pour un système $\mathcal{S}$ est une structure

$$(Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$$

où

- $Th(s, c)$ est une théorie définissant les ensembles, les constantes et les propriétés statiques de ces éléments.
- x est une liste de variables flexibles.
- VALS est un ensemble de valeurs possibles pour x.
- INIT(x) définit l'ensemble des valeurs initiales de x.
- $\{r_0, \ldots, r_n\}$ est un ensemble fini de relations reliant les valeurs avant x et les valeurs après x'.

Un modèle relationnel $\mathcal{MS} = (Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ pour un système $\mathcal{S}$ est une structure permettant de d'étudier un système au travers d'un modèle. Nous supposons que la relation r_0 est la relation $Id[\text{VALS}]$, identité sur VALS.

☼**Définition 1.2**
Soit $(Th(s, c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel d'un système $\mathcal{S}$. La relation NEXT associée à ce modèle est définie par la disjonction des relations r_i : NEXT $\stackrel{def}{=} r_0 \vee \ldots \vee r_n$.

La modélisation d'un système comprend la donnée des variables x, du prédicat caractériant les valeurs initiales des variables et une relation NEXT modélisant la relation entre les valeurs avant et les valeurs après. Les principes d'induction sont transcrits dans le cadre des modèles relationnels et nous introduisons la définition de la sûreté dans un modèle relationnel.

Soit $(Th(s,c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel d'un système $\mathcal{S}$. La théorie $Th(s,c)$ est définie dans un langage d'assertions qui permet de décrire un certain nombre de propriétés et de définir des ensembles. Un exemple est la théorie des ensembles du langage B. Nous apporterons des éléments plus précis dans les parties suivantes de ce cours, quand nous introduirons les notations de B. Quand nous parlerons d'une propriété φ, il s'agira de ce langage implicite dans notre exposé. Pour être précis et ne pas confondre les différentes notations, il est important de bien définir les valeurs du système étudié ; ainsi, pour une variable x, nous définissons les valeurs suivantes :

— x est la valeur courante de la variable x.
— x' est la valeur suivante de la variable x.

1.2.2 Propriétés de sûreté

Les propriétés de sûreté expriment que *rien de mauvais ne peut arriver [42]*. Par exemple, la valeur de la variable x est toujours comprise entre 0 et 567 ; la somme des valeurs courantes des variables x et y est égale à la valeur courante de la valeur z. On se donne un langage d'assertions $(\mathcal{P}(\Sigma), \subseteq)$ et une interprétation ensembliste de la relation de satisfaction.

☼**Définition 1.3**
Une propriété φ est une propriété de sûreté pour le système S, si

$$\forall s, s' \in \Sigma . s \in Init_{\text{S}} \wedge s \xrightarrow[R]{\star} s' \Rightarrow s' \in \varphi.$$

L'expression $\xrightarrow[R]{\star}$ désigne la fermeture réflexive transitive de la relation $\xrightarrow[R]{}$. La propriété de sûreté utilise une expression universelle pour quantifier les états. Pour démontrer une telle propriété, on peut soit vérifier la propriété pour chaque état possible de Σ_{P}, à condition que cet ensemble soit fini, ou bien trouver un principe d'induction. Pour ce qui est de la vérification pour chaque état, on utilise un algorithme de calcul des états accessibles à partir d'un état initial. Cette technique de calcul des états accessibles est souvent utilisée et est à la base des techniques de vérification automatique comme le *model checking* [46, 40, 23] Nous donnons maintenant un principe d'induction, pour prouver les propriétés de sûreté ; l'objectif est de montrer comment les techniques de preuves de propriétés de programmes sont construites et sur quels principes elles reposent.

☺ **Propriété 1.1** (Principe d'induction)
Une propriété φ est une propriété de sûreté pour le programme P si, et seulement si, il existe une propriété INV telle que

$$\begin{cases} (1)\ Init_{\text{P}} \subseteq INV \\ (2)\ INV \subseteq \varphi \\ (3)\ \forall s, s' \in \Sigma_{\text{P}} . s \in INV \wedge s \xrightarrow[R]{} s' \Rightarrow s' \in INV \end{cases}$$

La propriété INV est appelée un invariant du programme et est une propriété de sûreté particulière plus forte que les autres propriétés de sûreté. La justification de ce principe d'induction est assez simple.

PREUVE:

$\langle 1 \rangle 1$. SUPPOSONS QUE: Il existe une propriété INV telle que

$$\begin{cases} (1)\ Init_{\mathrm{P}} \subseteq INV \\ (2)\ INV \subseteq \varphi \\ (3)\ \forall s, s' \in \Sigma_{\mathrm{P}}.s \in INV \wedge s \underset{R}{\longrightarrow} s' \Rightarrow s' \in INV \end{cases}$$

PROUVONS QUE: φ est une propriété de sûreté pour le programme P

PREUVE:Soient deux états s, s' tels $s \in Init_{\mathrm{P}} \wedge s \xrightarrow[R]{\star} s'$. On peut construire une suite d'états $s = s_0 \underset{R}{\longrightarrow} s_1 \underset{R}{\longrightarrow} s_2 \underset{R}{\longrightarrow} s_3 \underset{R}{\longrightarrow} \ldots \underset{R}{\longrightarrow} s_i = s'$. Par l'hypothèse (1), on en déduit que INV est vraie en s. En utilisant (3) pour tous les couples de la trace, on en déduit que INV est vraie en $s_1, s_2, \ldots s_i$. Puis, nous appliquons (2) pour l'état s' et nous en déduisons que s' satisfait φ.□

$\langle 1 \rangle 2$. SUPPOSONS QUE: $\forall s, s' \in \Sigma.s \in Init_{\mathrm{P}} \wedge s \xrightarrow[R]{\star} s' \Rightarrow s' \in \varphi$

PROUVONS QUE: Il existe une propriété INV telle que

$$\begin{cases} (1)\ Init_{\mathrm{P}} \subseteq INV \\ (2)\ INV \subseteq \varphi \\ (3)\ \forall s, s' \in \Sigma_{\mathrm{P}}.s \in INV \wedge s \underset{R}{\longrightarrow} s' \Rightarrow s' \in INV \end{cases}$$

PROUVONS QUE: φ est une propriété de sûreté pour le programme P

PREUVE:On considère la propriété suivante $INV \stackrel{def}{=} \exists s \in \Sigma.s \in Init_{\mathrm{P}} \wedge s \xrightarrow[R]{\star} s'$. INV exprime que l'état s' est un état accessible à partir d'un état initial de P. $\mathcal{R}^\star$ est la fermeture réflexive transitive de $\mathcal{R}$. Les trois propriétés sont simples à vérifier pour INV. INV est appelé le plus fort invariant du programme P. □

$\langle 1 \rangle 3$. Q.E.D.

PREUVE:On en déduit l'équivalence par les pas $\langle 1 \rangle 1$ et $\langle 1 \rangle 2$.□

□

Cette propriété justifie la méthode de preuve par induction plus connue comme la méthode de Floyd/Hoare [33, 39], imaginée par Turing en 1949 [69]. La propriété énoncée donne une forme de l'induction qu'il faut ramener à des formes plus connues. P. et R. Cousot [29, 30, 31, 28] donne une synthèse complète sur les principes d'induction équivalents à ce principe d'induction. Nous appliquons ces résultats au cas des modèles relationnels de système et nous obtenons une expression de la définition d'une propriété de sûreté dans le cas d'un modèle relationnel de système.

☼**Définition 1.4**

Soit $(Th(s,c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel d'un système $\mathcal{S}$. Une propriété φ est une propriété de sûreté pour le système $\mathcal{S}$, si $\forall y, x \in \Sigma.Init(y) \wedge \text{NEXT}^\star(y, x) \Rightarrow \varphi(x)$.

A partir des principes d'induction de la section précédente, on peut dériver la propriété suivante.

☺ **Propriété 1.2** (Principe d'induction pour un modèle relationnel)

Soit $(Th(s,c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel d'un système $\mathcal{S}$. Une propriété $\varphi(x)$ est une propriété de sûreté pour $\mathcal{S}$ si, et seulement si, il existe une

propriété $i(x)$ telle que

$$\begin{cases} (1)\ \forall x \in \text{VALS}.Init(x) \Rightarrow i(x) \\ (2)\ \forall x \in \text{VALS}.i(x) \Rightarrow \varphi(x) \\ (3)\ \forall x, x' \in \text{VALS}.i(x) \wedge \text{NEXT}(x, x') \Rightarrow i(x') \end{cases}$$

PREUVE:La preuve est déduite de la preuve de la propriété 2.□

Si on transforme les propriétés, on obtient une forme plus proche de ce que nous utiliserons dans la suite et plus proche des notions de systèmes abstraits.

☺ **Propriété 1.3**

Les deux énoncés suivants sont équivalents :

(I) Il existe une propriété d'état $\mathrm{I}(x)$ telle que :

$$\forall x, x' \in \text{VALS} : \begin{cases} (1)\ \text{INIT}(x) \Rightarrow \mathrm{I}(x) \\ (2)\ \mathrm{I}(x) \Rightarrow \mathrm{P}(x) \\ (3)\ \mathrm{I}(x) \wedge \text{NEXT}(x, x') \Rightarrow \mathrm{I}(x') \end{cases}$$

(II) Il existe une propriété d'état $\mathrm{I}(x)$ telle que :

$$\forall x, x' \in \text{VALS} : \begin{cases} (1)\ \text{INIT}(x) \Rightarrow \mathrm{I}(x) \\ (2)\ \mathrm{I}(x) \Rightarrow \mathrm{P}(x) \\ (3)\ \forall i \in \{0, \ldots, n\} : \mathrm{I}(x) \wedge x\ r_i\ x' \Rightarrow \mathrm{I}(x') \end{cases}$$

PREUVE:La preuve est immédiate en appliquant la règle suivante : $\forall i \in \{0, \ldots, n\} : A \wedge x\ r_i\ x' \Rightarrow B \equiv (A \wedge (\exists i \in \{0, \ldots, n\} : x\ r_i\ x')) \Rightarrow B$ et la définition de $\text{NEXT}(x, x')$.□

Nous avons ainsi donné une explication de la règle d'induction qui est utilisée dans la méthode de Floyd [33, 69, 39] ; cette règle permet de mettre d'un côté les propriétés dites d'invariance c'est-à-dire celles qui nécessitent un pas d'induction et les propriétés plus générales de sûreté qui nécessitent une propriété d'invariance c'est-à-dire inductives pour pouvoir être prouvées. La méthode Event B met en œuvre ces deux types de propriétés d'une part par la clause INVARIANTS pour les invariants et d'autre part par la clause THEOREMS pour les propriétés de sûreté. Nous précisons que toute propriété d'invariance est une propriété de sûreté. Nous allons décrire le langage Event B et la méthode de développement incrémental de modèles événementiels.

1.3 Event B : un langage de modélisation

Event B désigne à la fois le langage et la méthode utilisant ce langage. Les concepts de ce langage sont limités et permettent à l'utilisateur de gérer une palette réduite : axiome, théorème, théorie, événement, machine, raffinement. Nous allons présenter ces notions de manière plus détaillée dans ce qui suit mais il est assez clair que les exemples constituent des moyens opérationnels pour comprendre par le jeu avec les outils mettant en œuvre ce langage.

La construction d'un modèle Event B repose sur des constructions syntaxiques comme les ensembles, les constantes, les axiomes, les théorèmes, les variables, les invariants, les événements ; ces constructions syntaxiques sont organisées dans des structures de deux types :

— les contextes rassemblent les informations statiques du domaine d'étude : les ensembles, les constantes, les axiomes, les théorèmes ; ces contextes sont extensibles et réutilisables au travers d'une clause spécifique permettant de les étendre ; ils sont utilisés pour définir la théorie mathématqiue du problème et apportent aux outils de preuve les informations nécessaires pour assister l'utilisateur dans la preuve des obligations de preuve.
— les machines organisent la définition des transitions du système en cours de modélisation et des changements des variables (d'état) du système ; elles utilisent les contextes pour faire référence à des informations statiques requises. Ces transitions d'état sont définies par des événements qui *réagissent* selon l'état courant des variables. Les variables d'une machine sont caractérisées par une liste de propriétés appelées *invariants* et peuvent aussi satisfaire des propriétés appelées *théorèmes*. Enfin, comme tout contexte peut être étendus par un autre contexte, une machine peut être raffinée par une autre machine.
— Que cela soit un contexte ou une machine Event B, leur cohérence doit être démontrée par la preuve des *obligations de preuve* engendrées par les outils et conformes aux résulatats exposés dans la section précédente. Si ces obligations de preuve sont démontrées, alors la structure est valide au moins du moins de vue du typage. En effet, le point délicat est l'explicitation et la preuve des propriétés d'invariance d'une machine donnée mais le raffinement est un outil très important pour assurer une progression dans la définition d'un modèle d'un système.

Nous allons donc décrire chaque élément mentionné ci-dessus sans entrer dans des détails de justification mais la section précédente apporte une explication des structures de modélisation en Event B. L'important est aussi d'avoir des illustrations pratiques de la modélisation en Event B. Nous résumons dans le diagramme ci-contre les relations entre les structures de Event B et la forme d'un développement d'une modélisation d'un système. Cette modélisation est très rarement à un seul niveau et doit bénéficier au maximum du raffinement.

1.3.1 Eléments de base d'un modèle Event B

Nous allons commencer par définir les événements qui sont au cœur de cette méthode et qui réagissent à une condition appelée une garde. Un événement est donc caractérisé par une condition et par une action. On retiendra trois uniques formes possibles pour les événements et cela suffira pour modéliser les systèmes au sens général. La première forme constitue une forme normale dans le sens où on peut réduire les deux suivantes sous cette forme. Le second cas correspond à un événement gardé et le troisième cas correspond à un événement quantifié qardé. Intuitivement, l'observation d'un événement est faite dans le cas où la garde est vraie mais le fait que la garde soit vraie ne permet pas d'en déduire que l'événement est ou sera observé. Chaque événement peut être défini par une relation before-after notée $BA(x, x')$.
Un événement est caractérisé par sa garde qui est déterminée au moment de la modélisa-

tion et il ne peut être déclenché que si cette garde est vraie. D'une certaine mesure, cela signifie qu'une condition apparaît et que la partie transformation associée est exécutée.
Nous allons détailler les obligations de preuve engendrées pour un événement donné e et expliquer la signification de ces obligations de preuve. Dans notre exposé, nous avons souligné le rôle du raffinement qui est défini sur les événements. La forme générale d'un événement est la suivante :

EVENT e
ANY t
WHERE
$G(c, s, t, x)$
THEN
$x :\mid (P(c, s, t, x, x'))$
END

— c et s désignent les constantes et les ensembles visibles par l'événement e et sont définis dans un contexte.
— x est la variable d'état ou une liste de variables.
— $G(c, s, t, x)$ est la condition d'activation de e.
— $P(c, s, t, x, x')$ est le prédicat établissant la relation entre la valeur avant de x, notée x, et la valeur après de x, notée x'.
— $BA(e)(c, s, x, x')$ est la relation *before-after* associée à e et définie par $\exists t.G(c, s, t, x) \wedge P(c, s, t, x, x')$.

Pour chaque événement e, les obligations de preuve sont désignées selon le format suivant : $e/inv/ < type >$ où $< type >$ est soit INV, soit FIS, GRD, SIM, THM, $WFIS$, WD, ...et correspondent à des obligations de preuves engendrées pour garantir l'invariance, le renforcement de la garde, la simulation, une propriété de sûreté, une définition d'une valeur, ...Nous ne recherchons pas l'exhaustivité et renvoyons le lecteur au livre de J.-R. Abrial [3] pour un exposé complet, ainsi qu'à la plateforme Rodin. Nous allons donner quelques éléments pour comprendre comment sont engendrées ces obligations de preuves.

1.3.2 Propriétés d'invariance en Event B

L'invariant $I(x)$ d'un modèle est une propriété invariante pour tous les événements du système modélisé y compris l'événement initial. Si e est un événement du modèle, alors la condition de préservation de cet invariant par cet événement est la suivante : $I(x) \wedge BA(e)(c, s, x, x') \Rightarrow I(x')$ (INV). $I(x)$ est écrit sous la forme d'une liste de prédicats étiquettés $inv_1, \ldots inv_n$ et est interprétée comme une conjonction. La condition sur les conditions initiales est la suivante : $Init(x, s, c) \Rightarrow I(x)$ $(INIT)$.
Quand un événement$e$ définit le prédicat before-after $BA(e)(c, s, x, x')$, la faisabilité de cet événement signifie que sous l'hypothèse définie par l'invariant $I(x)$ et par la garde grd (e), de l'événement, il existe toujours x' tel que $BA(e)(c, s, x, x')$; en d'autres termes cela veut dire que cet événement, quand il est observé, n'induit pas des comportements non souhaités et nous donnons une condition pour chaque événement : $I(x) \wedge$ grd $(e) \Rightarrow \exists x' \cdot BA(e)(c, s, x, x')$ (FIS).
Les propriétés de sûreté sont déduites par la preuve que l'invariant du système implique la propriété de sûreté $A(x)$ et nous ajoutons en plus, le contexte $C(s, c)$ de cette preuve. Le contexte de cette preuve est donné par les propriétés $C(s, c)$ des ensembles s et des constantes c définies dans le modèle : $C(s, c) \wedge I(x) \Rightarrow A(s, c, x)$ (THM).
Pour conclure ce point des obligations de preuve, elles ont été dérivées du théorème 2 et

nous pouvons donc en conclure la propriété suivante.

☺ **Propriété 1.4**

Soit $Th(s, c)$ une théorie définie par les ensembles s, les constantes c et les propriétés $C(s, c)$ et soit une liste finie E d'événements modifiant x et définis dans le contexte de la théorie $Th(s, c)$. On considère les points suivants :

- VALSest un ensemble de valeurs possibles pour x.
- $\{r_0, \ldots, r_n\}$ est l'ensemble des relations $BA(e)(s, c, x, x')$ définies pour les événements e de E et l'un des événements correspond à l'événement $skip$.
- INIT(x) est le prédicat définissant les conditions initiales de x.

Si les obligations de preuves (INIT) et (INV) sont satisfaites, alors le modèle relationnel $(Th(s, c), x,$ VALS, INIT$(x), \{r_0, \ldots, r_n\})$ satisfait l'invariant $I(x)$ et les propriétés de sûreté $A(s, c, x)$.

Nous allons maintenant donner les différentes obligations de preuves engendrées à partir de la forme générale donnée plus haut. On suppose que le contexte de la théorie est $C(s, c)$ et nous utiliserons la notation $C(s, c) \vdash P$ pour exprimer l'obligation de preuve dans le contexte $C(s, c)$. Nous avons donc les reformulations suivantes :

- *INIT/I/INV* : $C(s, c), INIT(c, s, x) \vdash I(c, s, x)$
- *e/I/INV* : $C(s, c), I(c, s, x), G(c, s, t, x), P(c, s, t, x, x') \vdash I(c, s, x')$
- *e/act/FIS* : $C(s, c), I(c, s, x), G(c, s, t, x) \vdash \exists x'.P(c, s, t, x, x')$

Nous avons donc instancié les principes d'induction pour assurer l'invariance de I. Le générateur d'obligations de preuve effectue aussi des simplifications importantes dans les énoncés produits.

1.3.3 Raffinement des événements

EVENT e
ANY t
WHERE
$G(c, s, t, x)$
THEN
$x :\mid (P(c, s, t, x, x'))$
END

EVENT f
REFINES e
ANY u
WHERE
$H(c, s, u, y)$
WITNESSES
$t : W1(c, s, t, u, y)$
$x' : W2(c, s, t, x', y)$
THEN
$y :\mid (Q(c, s, t, y, y'))$
END

Dans le schéma ci-dessus, $t : W1(c, s, t, u, y)$ est un témoin de preuve pour relier le paramètre courant u et le paramètre t de l'événement e ; $x' : W2(c, s, t, x', y)$ est un témoin de preuve pour relier la variable y et la valeur suivante de x. L'événement f raffine l'événement e, quand l'observation de f au niveau concret implique que l'événement e au niveau abstrait apparaît aussi. Plus formellement, le raffinement de e par f est défini par la formule : $I(c, s, x) \wedge J(c, s, x, y) \wedge BA(f)(c, s, y, y') \Rightarrow \exists x'.(BA(e)(c, s, x, x') \wedge J(c, s, x', y'))$ où $J(c, s, x, y)$ est l'invariant au niveau concret assurant la relation entre

les variables concrètes et abstraites. . Nous schématisons le raffinement comme suit :

On notera que le rôle des prédicats $W1$ et $W2$ est de fournir des valeurs pour prouver les propriétés existencielles induites par les paramètres mais aussi par la référence au niveau abstrait. On peut s'en passer mais il faut alors que l'utilisateur fournissent une valeur définie pour instancier un paramètre existenciel. La forme générale de l'obligation de preuve pour le raffinement de f par e prend en compte aussi le cas où f est un nouvel événement au niveau concret et dans ce cas f raffine skip qui ne modifie par la variable x. Nous allons comme précédemment donner les obligations de preuves engendrées à partir de la formulation ci-dessus.

— *e/act/SIM* : $\left(\begin{array}{l} C(s,c), \\ I(c,s,x), J(c,s,x,y), H(c,s,t,y), \\ Q(c,s,t,y,y'), \\ W1(c,s,t,u,y), W2(c,s,t,x',y) \end{array}\right) \vdash P(c,s,t,x,x')$

— *e/grd/FIS* : $\left(\begin{array}{l} C(s,c), \\ I(c,s,x), J(c,s,x,y), \\ H(c,s,t,y), W1(c,s,t,u,y) \end{array}\right) \vdash G(c,s,t,x)$

Nous avons donc donné des définitions précises des obligations de preuves engendrées pour vérifier le raffinement d'un événement par un autre. Il reste à définir les structures de machines et de contextes.

1.3.4 Structures des machines et des contextes en Event B

Un contexte rassemble les définitions des objets statiques du modèle du système à développer. Les ensembles de base sont définis dans la clause SETS et son *a priori* quelconques ; on peut déclarer qu'ils sont finis par le prédicat $finite(S)$.

CONTEXT $\mathcal{D}$
EXTENDS $\mathcal{AD}$
SETS
$S_1, \ldots S_n$
CONSTANTS
$C_1, \ldots, C_m$
AXIOMS
$ax_1 : P_1(S_1, \ldots S_n, C_1, \ldots, C_m)$
$\ldots$
$ax_p : P_p(S_1, \ldots S_n, C_1, \ldots, C_m)$
THEOREMS
$th_1 : Q_1(S_1, \ldots S_n, C_1, \ldots, C_m)$
$\ldots$
$th_q : Q_q(S_1, \ldots S_n, C_1, \ldots, C_m)$

— Les constantes sont déclarées dans la clause CONSTANTS.
— Les axiomes sont énumérés dans la clause AXIOMS et définissent les propriétés des constantes.
— Les théorèmes sont des propriétés déclarées dans la clauseTHEOREMS et doivent être démontrées valides en fonction des axiomes.
— Le contexte définit une théorie logico-mathématique qui doit être consistante.
— La clause EXTENDS étend le contexte mentionné et étend donc la théorie définie par le contexte de cette clause.

Un environnement de preuve $\Gamma(\mathcal{D})$ permet de formaliser le contexte dans un cadre logique et les propriétés suivantes doivent être prouvées logiquement :

$$\text{for any } j \text{ in } \{1..q\}, \Gamma(\mathcal{D}) \vdash th_j : Q_j(S_1, \ldots S_n, C_1, \ldots, C_m).$$

MACHINE $\mathcal{M}$
REFINES $\mathcal{AM}$
SEES $\mathcal{D}$
VARIABLES x
INVARIANTS
$inv_1 : I_1(x, S_1, \ldots S_n, C_1, \ldots, C_m)$
$\ldots$
$inv_r : I_r(x, S_1, \ldots S_n, C_1, \ldots, C_m)$
THEOREMS
$th_1 : SAFE_1(x, S_1, \ldots S_n, C_1, \ldots, C_m)$
$\ldots$
$th_s : SAFE_s(x, S_1, \ldots S_n, C_1, \ldots, C_m)$
EVENTS
EVENT initialisation
BEGIN
$x :|(P(x'))$
END
$\ldots$
EVENT e
ANY t
WHERE
$G(x, t)$
THEN
$x :|(P(x, x', t))$
END
$\ldots$
END

— Une machine est un modèle décrivant un ensemble d'événements modifiant des variables déclarées dans la clause VARIABLES.
— Un événement particulier définit l'initialisation des variables : EVENT Initialisation
— Une clause INVARIANTS décrit l'invariant que cette machine est supposée respecter à condition que les conditions de vérification associées soient démontrées valides dans la théorie induite par le contexte mentionné par la clause SEES.
— Enfin, la clause THEOREMS introduit la liste des propriétés de sûreté dérivées dans la théorie induite par le contexte et l'invariant ; ces propriétés portent sur les variables et doivent être démontrées valides.

Les conditions de vérification sont les suivantes :

— For any j in $\{1..r\}$,

$$\Gamma(\mathcal{D}, M) \vdash INITIALISATION(x') \Rightarrow I_j(x', S_1, \ldots S_n, C_1, \ldots, C_m)$$

— For any j in $\{1..r\}$, for any event e of M,

$$\Gamma(\mathcal{D}, M) \vdash \left(\begin{array}{c} \left(\bigwedge_{j \in \{1..r\}} I_j(x, S_1, \ldots S_n, C_1, \ldots, C_m) \right) \ \wedge \ BA(e)(x, x') \\ \Rightarrow \\ I_j(x', S_1, \ldots S_n, C_1, \ldots, C_m) \end{array} \right)$$

— For any k in $\{1..s\}$,

$$\Gamma(\mathcal{D}, M) \vdash \begin{array}{c} \left(\bigwedge_{j \in \{1..r\}} I_j(x, S_1, \ldots S_n, C_1, \ldots, C_m) \right) \\ \Rightarrow \\ SAFE_k(x, S_1, \ldots S_n, C_1, \ldots, C_m) \end{array}$$

— For any j in $\{1..r\}$,
$\mathcal{D}, M \longrightarrow \Box I_j(x, S_1, \ldots S_n, C_1, \ldots, C_m)$.

— For any k in $\{1..s\}$,
$\mathcal{D}, M \longrightarrow \Box SAFE_k(x, S_1, \ldots S_n, C_1, \ldots, C_m)$.

$$\mathcal{D}, M \longrightarrow \Box \left(\begin{array}{l} \left(\bigwedge_{j \in \{1..r\}} I_j(x, S_1, \ldots S_n, C_1, \ldots, C_m) \right) \\ \left(\bigwedge_{k \in \{1..s\}} SAFE_k(x, S_1, \ldots S_n, C_1, \ldots, C_m) \right) \end{array} \right)$$

Ces conditions contiennent des notations logiques et mathématiques de la déduction qui seront explicitées dans le chapitre de la logique. Il faut simplement retenir que l'expression $\Gamma(\mathcal{D}, M) \vdash \varphi$ signifie que la formule φ est valide dans la théorie définie par $\mathcal{D}$ et M.

1.4 Développement d'un algorithme séquentiel

Dans cette section, nous abordons deux études de cas très simples qui visent à montrer comment on peut développer des algorithmes séquentiels en utilisant la méthode Event B selon deux techniques. Parmi les travaux déjà réalisés, nous citerons d'une part les études de cas développées par Jean-Raymond Abrial [2] et les règles de transformations utilisées à partir des modèles Event B ; d'autre part, nous avons proposé aussi une méthode [58, 57] proposant un cadre pour guider les pas de raffinement, en s'appuyant sur une observation simple de ce qui est un événement. Nous allons illustrer ces deux techniques dans les exemples très simples de la somme d'une suite de valeurs entières $v_1, \ldots, v_n$ et de la recherche d'un élément dans une table. Afin de comparer les deux technqiues mais aussi de les illustrer, nous développerons le calcul de la somme avec la technique de J.-R. Abrial et la recherche dans une table avec notre technique, puisque J.-R. Abrial a développé une recherche dans une table.

1.4.1 Construction d'un algorithme de calcul de la somme par raffinement et transformation de modèles en algorithme

Description du domaine du problème à résoudre

Dans un premier temps, il faut exprimer la somme s de la suite v dans le langage Event B ; cette formulation est immédiate sur le plan mathématique : $s = \sum_{k=1}^{k=n} v(k)$. Comme la notation de sommation d'une suite finie de valeurs n'est pas proposée dans les élémentrs de base du langage, nous devons *définir* cette notion dans un contexte *somme0* qui va contenir les données du problèmes et les notations définies spécifiquement pour ce cas.
Dans un premier temps, les *données* n et v sont définies comme étant respectivement un entier naturel non nul (axiomes axm1, axm2) et une fonction v de domaine $1..n$ et de codomaine $\mathbb{N}$ (axiome axm3). Il s'agit de définir la théorie dans laquelle nous allons décrire nos données.
Dans un second temps, nous introduisons une suite u de valeurs correspondant aux sommes partielles $\sum_{k=1}^{k=i} v(k)$. Pour cela, l'idée est donc de définir les sommes partielles en utilisant une définition inductive qui va nécessiter sur le plan technique de s'assurer de la *bonne définition* de cette suite u. La suite u est donc définie comme suit :

— u est une fonction totale de $\mathbb{N}$ dans $\mathbb{N}$ (axiome axm4).
— Initialement, la sommation commence par 0 et $u(0) = 0$ (axiome axm5).
— Pour des valeurs de i inférieures à n, la valeur de $u(i)$ est définie à partir de celle

de $u(i-1)$ et de $v(i)$ (axiome axm6).
— Pour toutes les valeurs supérieures à n, la valeur de $u(i)$ est égale à celle de $u(n)$ (axiome axm7).

Les axiomes sont donnés dans le contexte *somme0* et constituent une théorie qui sera utile pour démontrer les propriétés des modèles que nous développerons plus tard.

CONTEXT $somme0$
CONSTANTS
n, v, u
AXIOMS
$axm1 : n \in \mathbb{N}$
$axm2 : n \neq 0$
$axm3 : v \in 1 \mathinner{..} n \rightarrow \mathbb{N}$
$axm4 : u \in \mathbb{N} \rightarrow \mathbb{N}$
$axm5 : u(0) = 0$
$axm6 : \forall i \cdot i \in \mathbb{N} \wedge i > 0 \wedge i \leq n \Rightarrow u(i) = u(i-1)+v(i)$
$axm7 : \forall i \cdot i \in \mathbb{N} \wedge i > n \Rightarrow u(i) = u(n)$
THEOREMS
$thm1 : \forall i \cdot i \in \mathbb{N} \Rightarrow u(i) \geq 0$
END

Dans le contexte ci-dessus, on notera que la clause THEOREMS est utilisée et son utilisation permet de dériver des propriétés mathématiques des données définies par leurs axiomes. Dans l'outil Rodin actuel, les auteurs ont pris le parti de mêler les axiomes et les théorèmes, en les différenciant pâr un nom en italique pour les théorèmes. Nous avons utilisé une notation qui permet de mieux exprimer ces théories. Enfin, chaque axiome est validé par un ensemble d'obligations de preuve engrendrées pour assurer la cohérence des définitions. Il en va de même des théorèmes qui doivent être prouvés à partir de l'environnement défini par les axiomes avec les règles de l'assistant de preuve. Nous avons donc défini le cadre mathématique du problème et nous allons maintenant définir le problème de la sommation de la suite v.

Spécification du problème à résoudre

Le problème est donc de calculer la valeur de la somme des éléments de la suite v. Nous définissons une machine *somme1* qui est un modèle exprimant au travers de l'événement EVENT sommation l'expression de la *postcondition* $somme = u(n)$. En fait, la nouvelle valeur de la variable $somme$ sera $u(n)$, quand l'événement EVENT sommation aura été observé. La valeur initiale de $somme$ est quelconque à l'initialisation. Enfin, la variable $omme$ doit satisfaire l'invariant très simple $inv1 : somme \in \mathbb{N}$; cette information constitue un typage de la variable $somme$. L'événement EVENT sommation est donc simplement une affectation de la valeur $u(n)$ à $somme$.

MACHINE $somme1$
SEES $somme0$
VARIABLES
$somme$
INVARIANTS
$inv1 : somme \in \mathbb{N}$
EVENT INITIALISATION
BEGIN
$act1 : somme :\in \mathbb{N}$
END
EVENT sommation1
BEGIN
$act1 : somme := u(n)$
END
END

On peut donner une expression sous la forme d'un triplet de HOARE : $\{n > 0 \wedge v \in 1 \,..\, n \rightarrow \mathbb{N}\} SOMMATION \{somme = u(n)\}$ où $SOMMATION$ est un algorithme solution. Il est à noter que les données sont *visibles* depuis le contexte *somme0*. Le problème est donc de trouver un algorithme $SOMMATION$ calculant la valeut $u(n)$ et la rangeant dans $somme$. Carrol Morgan [54] utilise la même méthode et nous ne faisons que *simuler* son calcul de raffinement, en ayant comme objectif de construire une solution algorithmique pour le problème spécifié comme une pré et post spécification.

Nous avons donc décrit le domaine du problème à résoudre et nous avons formulé ce que nous voulons calculer. La prochaine étape est celle de l'invention d'une *méthode de calcul* et elle requiert une *idée de solution* et l'utilisation du raffinement.

Raffiner pour calculer

Nous avons défini la spécification du problème du calcul de la somme des éléments d'une suite v et nous devons maintenant trouver un moyen de *calculer* la valeur de la suite u au terme n. L'affectation $somme := u(n)$ est une expression mêlant une variable $somme$ et une constante $u(n)$. On connaît bien une solution triviale et peu efficace : stocker les valeurs de la suite u dans un tableau t et traduire l'affectation sous la forme $somme := t(n)$ où t vérifie la propriété suivante $\forall k.k \in dom(t) \Rightarrow t(k) = u(k)$ et cette propriété constitue un élément de l'invariant inv8. L'idée est donc d'utiliser la variable t ($t \in 0..n \rightarrow\!\!\!\!\!\!+ \; \mathbb{N}$) pour contrôler le calcul et sa progression. La progression est assurée par l'événement pas2 qui permet de faire décroître la quantité $n-i$ et qui assure donc de la convergence du processus de progression.

MACHINE $somme2$
REFINES $somme1$
SEES $somme0$
VARIABLES
$somme, t, i$
INVARIANTS
$inv1 : i \in \mathbb{N}$
$inv2 : i \geq 0$
$inv3 : i \leq n$
$inv4 : t \in 0 \,..\, n \pfun \mathbb{N}$
$inv5 : dom(t) = 0 \,..\, i$
$inv6 : n \notin dom(t) \Rightarrow i < n$
$inv7 : dom(t) \subseteq dom(u)$
$inv8 : \forall k \cdot \left(\begin{array}{l} k \in dom(t) \\ \Rightarrow \\ t(k) = u(k) \end{array} \right)$
$inv9 : dom(u) = \mathbb{N}$

EVENT INITIALISATION
BEGIN
$act1 : somme :\in \mathbb{N}$
$act2 : t := \{0 \mapsto 0\}$
$act3 : i := 0$
END
EVENT sommation2
REFINES sommation1
WHEN
$grd1 : n \in dom(t)$
THEN
$act1 : somme := t(n)$
END
EVENT pas2
WHEN
$grd11 : n \notin dom(t)$
THEN
$act11 : t(i{+}1) := t(i){+}v(i{+}1)$
$act12 : i := i{+}1$
END
END

Le modèle $somme2$ dcrit donc un processus qui progressivement remplit le tableau t et conserve donc tous les résultats intermédaires. Les obligations de preuve sont assez faciles à prouver dans la mesure où nous avons *préparé* le travail de l'assistant de preuve. Nous donnerons les détails des statistiques dans une table en fin de développement. Il est assez calir que la variable t est en fait un témoin ou une trace des valeurs intermédiaires et que cette variable peut donc être cachée dans ce modèle qui devra être raffiné. Avant de cacher cette variable, nous allons mettre de côté la valeur que nous devons conserver $t(i)$.

Focaliser sur la valeur à conserver

Le raffinement suivant $somme3$ va conduire à introduire une nouvelle variable $psomme$ qui conservera la valeur $t(i)$. On opère donc une *superposition* [22] sur le modèle. L'idée est donc que ce modèle raffine ou simule le modèle précédent $somme2$; cela signifie aussi que les propriétés des modèles raffinés restent vérifiées par le nouveau modèle $somme3$ dans la mesure où les obligations de preuve sont toutes vérifiées.

MACHINE $somme3$
REFINES $somme2$
SEES $somme0$
VARIABLES
$somme, i, t, psomme$
INVARIANTS
$inv1 : psomme \in \mathbb{N}$
$inv2 : psomme = u(i)$
EVENT INITIALISATION
BEGIN
$act1 : somme :\in \mathbb{N}$
$act2 : i := 0$
$act3 : t := \{0 \mapsto 0\}$
$act4 : psomme := 0$
END

EVENT somme3
REFINES somme2
WHEN
$grd1 : n \in dom(t)$
$grd2 : i = n$
THEN
$act1 : somme := psomme$
END
EVENT pas3 REFINES pas2
WHEN
$grd1 : n \notin dom(t)$
$grd2 : i < n$
THEN
$act1 : t(i+1) := t(i)+v(i+1)$
$act2 : i := i+1$
$act3 : psomme := psomme+v(i+1)$
END
END

Ce modèle est très expressif et apporte de nombreuses informations sur les informations requises pour s'assurer de l'adquation du modèle au problème exprimé dans le modèle $somme1$ qui est raffiné par ce modèle $somme3$. Il est encore plus clair que ce modèle $somme3$ est coûteux en variables et le raffinement nous permet de ne laisser que les variables utiles pour le calcul. Dans la suite, nous allons rendre le modèle plus algorithmioque et ne conserver dans le modèle concret les variables suffisantes pour le calcul.

Obtenir un modèle algorithmique

Dans cette dernière étape, nous raffinons le modèle $somme3$ en un modèle $somme4$ et nous cachons dans le modèle abstrait $somme3$ la variable t. Ainsi, le modèle $somme4$ comprend les variables $somme$, $psomme$ et i et on notera aussi qu'il satisfait des propriétés de sûreté appelées théorèmes dans le modèle $somme4$. Ces propriétés sont prouvées à partir des propriétés des modèles raffinés précédents. On a donc obtenu un modèle comprenant une initialisation et deux événements :

— L'événement somme4 est observé, quand la valeur de i vaut n et, dans ce cas, la variable $psomme$contient la valeur $u(n)$. L'invariant nous garantit que la valeur de $psomme$ est $u(n)$.
— L'événement pas4 est observé, quand la valeur de i est inférieure à n. Cela signifie aussi que, tant que cette valeur est inférieure à n, l'événement peut être observé et les traces engendrées à partir de ces événements correspondent donc à une structure algorithmique d'itération.

MACHINE $somme4$
REFINES $somme3$
SEES $somme0$
VARIABLES
$somme, i, psomme$
THEOREMS
$inv1 : psomme = u(i)$
$inv2 : i \leq n$
EVENT INITIALISATION
BEGIN
$act1 : somme :\in \mathbb{N}$
$act2 : i := 0$
$act3 : psomme := 0$
END

EVENT somme4 REFINES somme3
WHEN
$grd1 : i = n$
THEN
$act1 : somme := psomme$
END
EVENT pas3 REFINES pas3
WHEN
$grd1 : i < n$
THEN
$act1 : i := i{+}1$
$act2 : psomme := psomme{+}v(i{+}1)$
END
END

Jean-Raymond Abrial [3] suggère des règles de transformation progressive à appliquer sur les événements de modèle comme $somme4$. Ces règles sont assez simples à comprendre et nous allons les énoncer puis les appliquer sur notre exemple.

Fusion de deux événements pour construire une itération

Soient les deux événements suivants que nous souhaitons fusionner pour en obtenir une expersion algorithmique :

— Si P est invariant pour S, alors les deux événements sont fusionnés en l'événement suivant :

WHEN
P
Q
THEN
S
END

WHEN
P
$\neg Q$
THEN
T
END

WHEN
P
THEN
WHILE Q **DO**
S
OD;
T;
END

— Si P ne figure pas dans les événements, alors ilm n'y a pas de garde.

Fusion de deux événements pour construire une conditionnelle

Soient les deux événements suivants que nous souhaitons fusionner pour en obtenir une expersion algorithmique :

WHEN
P
Q
THEN
S
END

WHEN
P
$\neg Q$
THEN
T
END

— La condition sur P est moins forte et nous introduisons une structure conditionnelle :

WHEN
P
THEN
IF Q **THEN**
S
ELSE;
T;
FI;
END

— Si P ne figure pas dans les événements, alors ilm n'y a pas de garde.

Ces deux transformations sont correctes dans la mesure où elles conservent les traces possibles engendrées. Dans notre cas, nous pouvons appliquer la première transformation sur le modèle $somme4$. Soient les trois événements de notre modèle $somme4$:

EVENT somme4
REFINES somme3
WHEN
$grd1 : i = n$
THEN
$act1 : somme := psomme$
END

EVENT pas3
REFINES pas3
WHEN
$grd1 : i < n$
THEN
$act1 : i := i{+}1$
$act2 : psomme := psomme{+}v(i{+}1)$
END

EVENT INITIALISATION
BEGIN
$act1 : somme :\in \mathbb{N}$
$act2 : i := 0$
$act3 : psomme := 0$
END

BEGIN
$act1 : somme :\in \mathbb{N}$;
$act2 : i := 0$;
$act3 : psomme := 0$;
WHILE $grd1 : i < n$ **DO**
$act1 : i := i{+}1$
$act2 : psomme := psomme{+}v(i{+}1)$
OD;
$act1 : somme := psomme$
END

L'algorithme est obtenu par fusion des deux événements somme4 et pas4 en une itértaion et par une composition séquentielle de l'événement d'initialisation avec le produit de la fusion. L'algorithme construit satisfait par construction le triplet de HOARE :

Modèle	Total	Auto	Manual	% Auto	% Manual
somme0	5	0	5	0 %	100 %
somme1	4	3	1	75 %	25 %
somme2	23	21	2	87 %	13 %
somme3	7	5	2	71 %	29 %
somme4	7	7	0	100 %	0 %
total	46	36	10	78,2 %	21,7 %

TABLE 1.1 – Tableau des statistiques du développement de la sommation

$$\left\{ \left(\begin{array}{l} n > 0 \\ \wedge v \in 1 \,..\, n \to \mathbb{N} \end{array} \right) \right\}$$

BEGIN
$somme :\in \mathbb{N}$;
$i := 0$;
$psomme := 0$;
WHILE $i < n$ **DO**
  $i := i{+}1$
  $psomme := psomme{+}v(i{+}1)$
OD;
$somme := psomme$
END

$$\{somme = u(n)\}$$

Un certain nombre d'exemples ont été traités avec cette méthode et peuvent être consultés sur le site dédié au projet Rodin. Dans ses publications [3, 2], Jean-Raymond Abrial traite à la fois cette technique et des exemples plus ou moins compliqués. Avant de terminer cette étude, il est important de donner des statistiques sur le nombre d'obligations de preuve et sur les difficultés des preuves réalisées manuellement avec l'assistant de preuve. La table 1.1 précise que 78,2 % des obligations de preuve sont automatiques mais les 21,7 % réalisés par interaction avec l'assistant de preuve ne sont pas très complqiuées dans la mesure où le raffinement est progressif.

1.4.2 Développement d'une algorithme séquentiel avec le patron de développement *appel-événement*

Si nous reprenons le problème que nous voulons résoudre, nous rappelons que nous recherchons une procédure `PROCEDURE` satisfaisant une spécification de la forme générale suivante :

PROCEDURE `PROCEDURE`$(x;$ **VAR** $y)$
PRECONDITION $P(x)$
POSTCONDITION $Q(x, y)$

Pour le second développement d'un algorithme séquentiel, nous allons utiliser un patron de développement décrit par le schéma suivant :

□

Le schéma est interprété comme suit :

- `CALL` est l'appel d'une procédure `PROCEDURE`
- PREPOST est la machine contenant les événememnts *simulant* les différents appels de la procédure à construire ; cette machine décrit sous la forme d'événements la pré-post specification.
- La transformation *call-as-event* produit un modèle PREPOST et un contexte PB à partir de `CALL` et d'infoirmations sur le problème et son domaine.
- La transformation *mapping* nous permet de dériver la procédure algorithmique à partir du modèle M obtneu par raffinement du modèle PREPOST.
- `PROCEDURE` correspond à la procédure ddérivée à partir del M. `CALL` est une instanciation de `PROCEDURE` avec les paramètres x et y.

Nous considérons l'exemple de la recherche d'un élément x dans une table t de n éléments appartenant à un ensemble déclaré A. Nous considérons un spécification de la forme suivante :

PROCEDURE `search`$(x, t, n;$ **VAR** $i, ok)$
PRECONDITION $x \in A \land n > 0 \land t \in 1 \,..\, n \to A$
POSTCONDITION $\left(\begin{array}{l} (\forall k \cdot k \in 1 \,..\, n \Rightarrow t(k) \neq x) \Rightarrow ok = non \\ (\exists k \cdot k \in 1 \,..\, n \land t(k) = x) \Rightarrow ok = oui \end{array} \right)$

Nous recherchons donc les éléments suivants dans le patron instancié sur notre problème :

□

CONTEXT $search0$
SETS
  A
  $REPONSES$
CONSTANTS
  x, t, n, oui, non
AXIOMS
  $axm1 : n \in \mathbb{N}_1$
  $axm2 : x \in A$
  $axm3 : t \in 1 \,..\, n \to A$
  $axm4 : REPONSES = \{oui, non\}$
  $axm5 : oui \neq non$
END

La description du contexte du problème PB est donc donnée par le contexte $search0$ qui définit très simplement la structure de recherche t. Nous devons aussi définir un ensemble des réponses possibles de la recherche. Ce contexte est en fait utilisé pour décrire correctement la précondition de la procédure de recherche $search0$. Nous pouvons maintenant définir la spécification elle-même en écrivant la machine $specsearch$ qui comprendra les événements simulant l'appel de `search`.

La machine $specsearch$ comprend donc un événement d'initialisation et deux événements : find qui modélise la procédure `search` quand elle trouve un élément x dans le tableau t. et unfind qui modélise la procédure `search` quand elle ne trouve pas d'élément x dans le tableau t. En fait, les deux événements ne donnent qu'une définition du *quoi* mais pas du *comment* ; ces événements constituent une façon simple de décrire l'effet attendu. Pour les définir de manière plus opérationnelle, nous allons les *raffiner*. Ainsi, ces deux événements ne sont que deux instances de l'appel de cette procédure.

MACHINE $specsearch$
SEES $search0$
VARIABLES
i, ok
INVARIANTS
$inv1 : ok \in REPONSES$
$inv2 : i \in 1 \,..\, n$

EVENT INITIALISATION
BEGIN
$act1 : i :\in 1 \,..\, n$
$act2 : ok := non$
END

EVENT find
ANY
j
WHERE
$grd1 : j \in 1 \,..\, n$
$grd2 : t(j) = x$
THEN
$act1 : ok := oui$
$act2 : i := j$
END
EVENT unfind
WHEN
$grd1 : \forall k \cdot k \in 1 \,..\, n \Rightarrow t(k) \neq x$
THEN
$skip$
END
END

Pour résoudre notre problème d'un point de vue opérationnel, nous allons devoir analyser le problème en considérant plusieurs cas et nous introduisons une nouvelle variable c qui modélise le contrôle du processus de recherche. Nous avons utilisé un nouveau contexte $csearch0$ qui étend le contexte $search0$ en définissant les points de contrôle possibles :

CONTEXT $csearch0$
EXTENDS $search0$
SETS
$LOCS$
CONSTANTS
$start, end, call1$
AXIOMS
$axm1 : partition(LOCS, \{start\}, \{end\}, \{call1\})$
END

MACHINE $simsearch$
REFINES $specsearch$
SEES $csearch0$
VARIABLES
i, ok, c
INVARIANTS
$inv1 : c \in LOCS$
$inv2 : c = call1 \Rightarrow n \neq 1 \wedge ok = non$
$inv3 : c = call1 \Rightarrow t(n) \neq x$
$inv4 : c = end \wedge ok = oui \Rightarrow t(i) = x$
$inv5 : c = end \wedge ok = non \Rightarrow (\forall g \cdot g \in 1 \,..\, n \Rightarrow t(g) \neq x)$
$inv6 : c = start \Rightarrow ok = non$
...

L'invariant décrit ce qui se passe pendant le calcul :

— Si le point de contrôle est en fin et si la variable ok contient oui, alors $t(i) = x$.
— Si le point de contrôle est en fin et si la variable contient non, alors i contient n'importe quellel valeur et t ne contient aucune occurence de x.

Pour réaliser ce calcul, on introduit deux cas : le cas où n vaut 1 et le cas où n est différent de 1. Considérons le cas où n vaut 1. Dans ce cas, nous raffinons find par findone, pour expliquer comment on trouve la valeur x dans un tableau à une valeur, si elle est effectivement dans ce tableau, et nous raffinons unfind par notfoundone pour le cas où la valeur x n'est pas en $t(1)$. Nous avons traité les deux sous-cas de base et la traduction en notation algorithmique de ces deux événements est immédiate. Chaque événement (findone et notfindone) est traduit sous la forme d'une conditionnelle. On peut d'ailleurs utiliser l'outil EB2ALL [61] pour traduire le mdoèle complet et obtenir ainsi un programme C, C++, C# ou Java exécutable.

EVENT INITIALISATION
BEGIN
$act1 : i :\in 1 \,..\, n$
$act2 : ok := non$
$act3 : c := start$
END

EVENT findone
REFINES find
ANY
j
WHERE
$grd1 : j \in 1 \,..\, n$
$grd2 : t(j) = x$
$grd3 : c = start$
$grd4 : n = 1$
$grd5 : t(n) = x$
THEN
$act1 : ok := oui$
$act2 : i := 1$
$act4 : c := end$
END

EVENT nofindone
REFINES unfind
WHEN
$grd1 : \forall k \cdot k \in 1 \,..\, n \Rightarrow t(k) \neq x$
$grd2 : n = 1$
$grd3 : t(n) \neq x$
$grd4 : c = start$
THEN
$act1 : c := end$
END

Pour le second cas, nous avons donc l'hypothèse que n est différente de 1 et nous allons raffiner les deux événements find et unfind, en simulant une technique de recherche récursive. Nous avons donc des événements correspondant à des parties de l'analyse récursive et ces événements sont contrôlés à l'aide de c :

— foundlastone trouve la valeur x à la dernière place du tableau t et positionne ok à oui. Le contrôle c est changé et la recherche est terminée.
— notfoundlastone ne trouve pas la valeur x à la dernière place du tableau t et la recherche doit se poursuivre sur le reste du tableau t entre 1 et $n-1$. Le contrôle est donc transmis en $call1$.
— Les deux événements suivants sont donc activables en $c = call1$ selon qu'il existe ou pas une valeur. Chacun de ces événements simule en fait la procédure `search` mais entre 1 et $n-1$. Bien évidemment, on ne dit pâs comment on fait la recherche et on traduit ces deux événements par des appels récursifs. Ce point permet de simplifier les invariants et les preuves ; on pourra se reporter au document [58, 57] introduisant cette technique, pour y consulter la calcul du plus court chemin et ainsi constater que l'invariant est assez facile àn trouver mêm s'il est complexe dans sa forme finale.

EVENT foundlastone
REFINES find
WHEN
$grd1 : n \neq 1$
$grd2 : t(n) = x$
$grd3 : c = start$
WITNESSES
$j : j = n$
THEN
$act1 : c := end$
$act2 : i := n$
$act3 : ok := oui$
END
EVENT notfoundlastone
WHEN
$grd1 : c = start$
$grd2 : n \neq 1$
$grd3 : t(n) \neq x$
THEN
$act1 : c := call1$
END

EVENT foundrec
REFINES find
ANY
k
WHERE
$grd1 : k \in 1 \,..\, n{-}1$
$grd2 : c = call1$
$grd3 : t(k) = x$
WITNESSES
$j : j = k$
THEN
$act1 : c := end$
$act2 : i := k$
$act3 : ok := oui$
END
EVENT notfounrec
REFINES unfind
WHEN
$grd1 : \forall l \cdot l \in 1 \,..\, n{-}1 \Rightarrow t(l) \neq x$
$grd2 : c = call1$
THEN
$act1 : c := end$
END
END

Pour conclure notre patron, nous devons produire l'algorithme à partir des événements de ce dernier modèle.

`Procedure search(x, t, n; i, ok)`
BEGIN
$i :\in 1 \,..\, n; ok := non;$
IF $n = 1 \wedge t(n) = x$ **THEN**
$ok := oui; \quad i := 1$
ELSEIF $n = 1 \wedge t(n) \neq x$ **THEN**
skip
ELSEIF $n \neq 1 \wedge t(n) = x$ **THEN**
$ok := oui; \quad i := n;$
ELSE $search(x, t, n{-}1, i, ok);$
FI
END

La procédure est produite par transformation des événements en fragments de code et ces fragments sont organisés selon les valeurs de la variable c.

Comme nous l'avons déjà souligné, cette technique permet de simplifier la construction de l'invariant et de simplifier aussi sa preuve. Nous [58, 57] avons réalisé des exemples assez classiques comme le calcul des coefficients du binôme, le calcul du plus court chemin, les fonctions primitives récursives, l'algorithme CYK d'analyse syntaxique. L'outil EB2ALL [61] put être utilisé pour traduire ces modèles en C, C++, C# ou Java. Ces deux

Modèle	Total	Auto	Manual	% Auto	% Manual
search0	0	0	0	0 %	0 %
csearch0	0	0	0	0 %	0 %
specsearch	5	5	0	100 %	0 %
simsearch	52	49	3	94,2 %	5,8 %
total	57	54	3	94,7 %	5,3 %

TABLE 1.2 – Tableau des statistiques du développement de la recherche

techniques de développement simulent la méthode de Carrol Morgan [54] mais une différence réside dans la systématisation du raffinement le plus simple possible. Il ne faut pas aller trop vite mais introduire des machines ou modèles intermédiaires qui vont simplifier le travail de preuve. On notera enfin que ce modèle a utilisé la clause WITNESSES dans l'événement foundrec sous la forme syuivante : $j : j = k$; cette clause permet d'aider le prouveur a instancié un quantificateur existenciel qui exprime que dans l'événement abstrait raffiné par foundrec, il faut lui donner une valeur j permettant d'observer l'événement abstrait. Cet artifice permet de conserver des informations présentes dans la preuve du modèle abstrait. Au niveau des bilan des obligations de preuve, le tableau 1.2 détaille les preuves automatiques et interactives ; les trois preuves interactives nécessitent quelques interactions mais peu de temps.

1.5 Développement d'un algorithme réparti

1.5.1 Modélisation des algorithmes répartis

Nous allons illustrer une technique de développement d'algorithme réparti en utilisant la méthode Event B. Cette technique s'appuie sur un modèle de calcul réparti appelé Visidia [56] et l'objectif est de produire un algorithme Visidia à partir d'un problème posé. Nous considérons le problème de l'arbre de recouvrement d'un graphe et nous considérons un patron de développement ibtégrant le raffinement et permettant de développer des algorithmes Visidia qui peuvent être simulés sur la plateforme VISIDIA [56]. Le patron de développement est caractérisé par le schéma suivant :

□

- Le contexte *C* donne les propriétés nécessaires des graphes, puisque les algorithmes répartis utilisent bien souvent des propriétés des graphes.
- Ma machine *M0* décrit par des événements le problème à résoudre en en donnant une expression abstraite, comme par exmeple, l'élection d'un leader dans un réseau est exprimée par l'émergence d'un nœud sachant qu'il est leader et les autres nœuds sachant qu'ils ne sont pas leader mais qu'un leader est élu. L'existence d'une solution dépend évidemment des propriétés du graphe support du calcul réparti.
- Le raffinement de *M0* par *M1* exprime comment le modèle Visidia réalise un calcul ; un modèle Visidia est une liste finie de règles de récriture de graphes qui simule l'exécution d'un calcul réparti en localisant les calculs autour d'un nœud ou bien encore entre deux voisins. Le modèle de clcul Visidia est très simple et relativement abstrait mais est supporté par un outil de simulation.

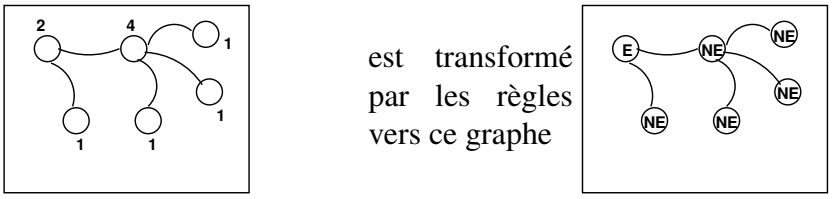


FIGURE 1.1 – Graphe d'élection du leader

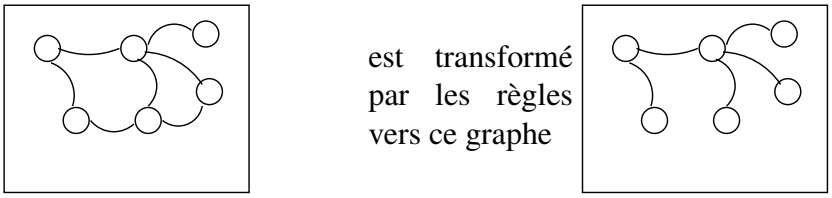


FIGURE 1.2 – Calcul de l'arbre de recouvrement d'un graphe avec le modèle Visidia

— Le raffinement suivant simplifie le modèle *M1* en un mdoèle où ne figurent plus que les régles de récriture de graphes Visidia ; il s'agit d'obtenir en quelque sorte un modèle Visidia en Event B qui est traduit facilement en Visidia pour être simulé.
— *V* est le modèle Visidia dérivé de *VM1* ; *mapping* assure la traduction de *VM1* en VISIDIA [56].

L'élection du leader est simplement défini par les régles suivantes appliquées sur deux nœuds voisins :

— $\overset{d}{\bullet}\!\!-\!\!-\!\!\overset{1}{\bullet} \longrightarrow \overset{d-1}{\bullet}\!\!-\!\!-\!\!\overset{NE}{\bullet}$
— $\overset{0}{\bullet} \longrightarrow \overset{E}{\bullet}$

Chaque nœud est étiquetté par le nombre de voisins et l'application des règles est non-déterministe. Le leader est l'unique nœud dont tous les vosins lui ont demandé d'être leader. Nous avons développé l'élection du leader pour le protocole IEEE 1394 [41] sur ce principe mais en allant jusqu'à un nioveau plus concret, sans néanmoins résoudre les question des probabilités inhérentes dans ce type d'algorithme. Nous donnons dans la figure 1.1 un exmeple d'exécution des régles sur un exemple de graphe étiquetté. Notons que ces règles calculent le leader dans un graphe sans cycle.

Nous allons maintenant présenter le développement du calcul de l'arbre de recouvrement d'un graphe avec ce modèle de calcul. Nous donnons un exemple de calcul d'un arbre à partir d'un graphe avec cycles dans la figure 1.2.

Nous allons décrire le patron suivant permettant de résoudre le problème du calcul de l'arbre de recouvrement d'un graphe connexe. Nous désignerons ce problème par le nom SPAN.

□

1.5.2 Eléments du patron de développement

La modélisation des graphes est le point de départ de ce développement. Le contexte $graph$ définit un graphe g comme un sous-ensemble de l'ensemble $N \times N$ et apporte des axiomes pour le caractériser notamment qu'il est symétrique (axm2) et connexe (axm3).

CONTEXT $graph$
SETS
N
CONSTANTS
g, r
AXIOMS
$axm0 : r \in N$
$axm1 : g \subseteq N \times N$
$axm2 : g = g^{-1}$
$axm3 : \forall s \cdot s \subseteq N \wedge r \in s \wedge g[s] \subseteq s \Rightarrow N \subseteq s$
END

Puis, nous donnons une expression de l'événement modéilsant le calcul en un coup d'un arbre de recouvrement. La machine $one-shot$ comporte un seul événement span qui affecte à la variable $span$ un arbre de recouvrement deu graphe g. L'invariant est simplment l'expression que $span$ est un sous-ensemble de g mais qu'il est un arbre de recouvrement comme l'indique l'expression de la valeur at. Le point important est de démontrer qu'une valeur at existe bien. Cette preuve est faite en démontrant que cet événement est faisable.

MACHINE $one-shot$
SEES $graph$
VARIABLES
$span$
INVARIANTS
$inv2 : span \subseteq g$
EVENT INITIALISATION
BEGIN
$act2 : span := \varnothing$
END
EVENT span
ANY
at
WHERE
$grd1 : at \subseteq g$
$grd2 : at \in N \setminus \{r\} \rightarrow N$
$grd3 : \forall s \cdot s \subseteq N \wedge r \in s \wedge at^{-1}[s] \subseteq s \Rightarrow N \subseteq s$
THEN
$act1 : span := at$
END
END

Puis, nous allons raffiner cette machine en une autre machine qui simule le calcul de cet arbre en utilisant deux variables a et tr. La variable a sert à contenir les nœuds déjà choisis durant le calcul pour être dans l'arbre et tr contient l'arbre de recouvrement en construction. L'invariant exprime que tr est une forêt c'est-à-dire que tr est un sous-ensemble de g sans cycle (inv7). L'invariant inv6 exprime que tr est une fonction totale de domaine a sans r et donc que r joue le rôle de racine de cet arbre en construction.

MACHINE $rules$ REFINES $one-shot$
SEES $graph$
VARIABLES
$span, tr, a$
INVARIANTS
$inv4 : a \subseteq N$
$inv2 : tr \subseteq g$
$inv5 : r \in a$
$inv6 : tr \in a \setminus \{r\} \rightarrow a$
$inv7 : \forall s \cdot \left(\begin{array}{l} s \subseteq a \\ \wedge r \in s \\ \wedge tr^{-1}[s] \subseteq s \end{array} \right) \Rightarrow a \subseteq s$

Deux événements span et rule1 modélisent les évolutions possibles de tr et de a. Au passage, on notera qu'il est important de chosir un nœud particuluier pour débuter le processus et a est initialisé au singleton contenat r un nœud choisi de façon arbitraire. L'événement span détecte la fin du processus en testant si a contient tous les éléments de N et positionne $span$ à la valeur de tr atteinte. Le rôle de rule1 est différent et chosi un nœud y qui n'est pas encore dans a mais qui est accessible à partir d'un nœud de a par le graphe g. Cette condition vise à éviter de créer un cycle.

EVENT INITIALISATION
BEGIN
$act4 : span := \varnothing$
$act2 : tr := \varnothing$
$act3 : a := \{r\}$
END
EVENT span
REFINES span
WHEN
$grd1 : a = N$
WITNESSES
$at : at = tr$
THEN
$act1 : span := tr$
END

EVENT rule1
ANY
x, y
WHERE
$grd1 : x \in N$
$grd2 : y \in N$
$grd3 : x \mapsto y \in g$
$grd4 : x \in a$
$grd5 : y \notin a$
THEN
$act2 : a := a \cup \{y\}$
$act1 : tr := tr \cup \{y \mapsto x\}$
END
END

La machine $visidia$ raffine la machine $rules$ en la rendant plus proche du modèle Visidia. En fait, il s'agit d'un raffinement *cosmétique* permettant de rendre l'arbre symétrique et de trasnformer les événements en des règles du modèle Visidia. Pour représenter l'appartenance à a, nous utilisons la coloration du nœud en noir. La nouvelle variable lb est utilisée pour localiser cette information pour chaque nœud de a et l'invariant inv2 exprime cette propriété de relation entre les nœuds de a et les nœuds colorés en noir. Les couleurs du marquaqge sont indiquées dans l'ensemble $MARKING$. A l'initialisation, tous les nœuds sont blancs sauf r.

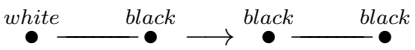
MACHINE $visidia$ REFINES $rules$
SEES $cvisidia$
VARIABLES
tr, lb
INVARIANTS
$inv1 : lb \in N \rightarrow MARKING$
$inv2 : \forall i \cdot i \in a \Leftrightarrow lb(i) = BLACK$

Les deux événements span et rule1 raffinent les événements de même nom dans le modèle $rules$ et expriment des conditions locales par la variable lb.

EVENT INITIALISATION
BEGIN
$act2 : tr := \varnothing$
$act4 : lb := lb0$
END
EVENT span
REFINES span
WHEN
$grd1 : \forall i \cdot i \in N \Rightarrow lb(i) = BLACK$
THEN
$skip$
END

EVENT rule1
REFINES rule1
ANY
x, y
WHERE
$grd3 : x \mapsto y \in g$
$grd4 : lb(x) = BLACK$
$grd5 : lb(y) = WHITE$
THEN
$act2 : lb(y) := BLACK$
$act1 : tr := tr \cup \{y \mapsto x\}$
END
END

La dernière étape est donc la production des règles dans le modèle de programmation répartie Visidia et à partir de l'événement rule1 on dérive une seule règle à condition qu'un des nœuds soit noir à l'initialisation :



Nous avons donc extrait la règle définissant le programme Visidia à partir des événements de la machine $visidia$ qui ne contient dans ses événements que des informations localisables. On peut donc en déduire que le programme réparti conçu construit un arbre de recouvrement. La question de la convergence de ce système est déduite d'une analyse de la décroissante de l'ensemble $N\ a$ par l'événement rule1.

1.6 Outils

Les outils mettant en œuvre cette méthode sont d'une part l'Atelier B [24] et d'autre part Rodin [4].

1.6.1 Atelier B

L'Atelier B [24] est diffusé gratuitement par la société ClearSy qui le propose pour les quatre plateformes Windows, Linux, MacOs, Solaris ; une diffusion sous licence est proposée et permet d'avoir accès à un certain nombre de documentation et d'études de cas. Cette plateforme propose dans un même cadre la méthode B Classique avec des outils de traduction mais aussi Event B avec une syntaxe légèrement différente. Les fonctionnalités offertes incluent la génération d'obligations de preuve, l'assistance à la preuve

interactive, le raffinement automatique avec l'outil Bart [26] et des outils de traductions vers C ou ADA. Cette même société poursuit la diffusion gratuiite d'une plateforme appelée B4Free[25] basée sur les travaux communs de J.-R. Abrial et de D. Cansell sur la balbulette [6]. L'idée de la balbulette était de fournir une interface avec les composants de l'Atelier B comme le générateur d'obligations de preuve, le prouvuer ou des traducteurs, afin de faciliter la tâche du développeur en l'assistant dans la démarche de preuve interactive et lma gestion des projets. Une des difficultés dans l'utilisation d'outil comme l'Atelier B réside dans l'utilisation interactive de l'assistant pour prouver des obligations de preuve qui n'ont pu être traitées par les procédures automatiques. B4Free offre donc des aides durant le processus de preuve et propose des règles à appliquer et ces règles sont ensuite sélectionnées par l'utilisateur. Cet outil a rencontré un grand succès auprès des partenaires académiques et ses fonctionnalités sont intégrées à la plateforme Rodin sous Eclipse.

1.6.2 La plateforme Rodin

La plateforme Rodin est la mise en œuvre de la méthode Event B dans l'environnement Eclipse ; elle fait suite aux travaux menés dans le cadre des outils comme Click'N'Prove [19]. Elle est dédiée uniquement à Event B mais propose des fonctionnalités sous forme de plugins (traduction vers des langages de programmation à partir de modèles Event B ou encore intégration de méthodologies comme UML). La plateforme Rodin a été utilisée pour développer les études de cas illustrant ce texte et nous [62, 65, 63, 64] avons utilisé des outils complémentaires comme ProB [36] qui offre les fonctionnalités à l'assistant de preuve, comme l'animation et le model-checking.

1.7 Conclusion et Perspectives

1.7.1 Applications à des cas d'études

Les applications de cette technique sont très nombreuses et le développement des outils a permis de faciliter ces études de cas. Dans notre exposé, nous avons principalement utilisé la plateforme Rodin mais la plateforme Atelier B peut être substituée. L'assistant de preuve est en partie fourni par cette plateforme et des prouveurs spécifiques ont été développés pour Rodin. Nous allons énoncer quelques applications développées sur cette plateforme, afin de montrer à la fois ce qui peut être modélisé et aussi la puissance des outils.

Les algorithmes répartis [41] constituent une classe intéressante de problèmes algorithmiques complexes ; le développement de l'algorithme d'élection du leader dans le cas d'un réseau non-orienté sans cycle a permis d'ouvrir des voies de recherche pour étudier les questions d'intégration du temps [21, 66] dans le développement et de gestion des probabilités [55, 7, 35] inhérentes et souvent implicites. Parmi les algorithmes répartis, les algorithmes cryptologiques consituent aussi une classe intéressante pour mesurer l'impact du raffinement dans leur dérivation mais aussi de mesurer le pouvoir d'expressivité du langage Event B face au modèle de l'attaquant de Dolev-Yao [11]. Cela a permis de développer des algorithmes d'authentification et de distribution de clés [16, 15, 14, 17, 12], en mettant en avant des mécanismes élémentaires constituant ces algorithmes. D'une certaine mesure les difficultés résident dans la compréhension de la propriété à modéliser. Ces études ont conduit aussi à la proposition de patrons de développement facilitant l'in-

troduction du temps [21, 66] et de patrons de développement dans le cadre du modèle de programmation répartie Visidia [59, 60]. Plus récemment, la question des réseaux dynamiques c'est-à-dire des réseaux sur des graphes qui évoluent avec le temps a été étudiée en Event B pour la découverte de topologie [38] ou le routage dynamique [47].
Pour ce qui concerne les algorithmes séquentiels, J.-R. Abrial [2] a proposé des régles de traduction des modèles Event B en une notation algorithmique. La démarche a été exposée dans ce chapitre et permet effectivement de (re)développer des algorithmes séquentiels. L'approche fondée sur l'association *appel-événement* [58, 57] permet de développer relativement simplement des algorithmes séquentiels en facilitant l'expression de l'invariant et en mettant en avant une analyse récursive du problème.
Plus classiquement, la méthode Event B est utilisée pour développer des systèmes intégrant des éléments logiciels et nécessitant des arguments objectifs pour certifier leur fonctionnement. J.-R. Abrial a développé un modèle d'une presse mécanique [3], en veillant à maintenir les propriétés de sécurité. Cette technique fondée sur le langage Event B constitue une véritable ingénierie système formelle et a produit des patrons de développement [3, 37] et des structures, les chartes de raffinement [50]. Parmi des études importantes, on pourra citer la modélisation du pacemaker [63, 49], du modèle électrique du cœur [51] et du protocole médical lui-même [53]. Des éléments pour traiter les questions de sécurité comme le contrôle d'accès [13, 12] ont montré aussi que le langage Event B était très flexible pour intégrer des modèles de contrôle d'accès comme RBAC ou OrBaC. Enfin, le développement de code à partir de modèles Event B [52, 48] s'appuie sur un ensemble d'outils intégrables à Rodin ; ces outils permettent de produire du code qui est ensuite assemblé.

1.7.2 Conclusion et Perspectives

La méthode Event B repose sur un langage puissant fondé sur la théorie des ensembles et le calcul des prédicats du premier ordre et offre des structures simples de machines pour décrire des systèmes réactifs. D'une certaine mesure, on peut décrire dans d'autres langages les systèmes réactifs mais le raffinement est un concept qui permet de développer de manière incrémentale et assurée des modèles complexes de systèmes de taille relativement importante comme une presse mécanique ou encore un pacemaker. De plus, les outils ont atteint une maturité à la fois dans l'interface et dans la puissance des outils de preuve ; ils exigent une certaine pratique mais que cela soit l'assistant de preuve ou l'animateur ProB, chacun apporte des éclairages sur les modèles développés et contribue à la validation des modèles. Pour ce qui est des perspectives, nous considérons que le traitement du temps, les aspects probabilistes des systèmes, l'intégration de langages moins formels, la mise en évidence de patrons de développement et la rédaction d'études de cas sont les points à explorer, tout en maintenant un développement des outils [5].

BIBLIOGRAPHIE

[1] J.-R. Abrial. *The B book - Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] J.-R. Abrial. Event based sequential program development : Application to constructing a pointer program. In *FME 2003*, pages 51–74, 2003.

[3] J.-R. Abrial. *Modeling in Event-B : System and Software Engineering*. Cambridge University Press, 2010.

[4] Jean-Raymond Abrial, Michael J. Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin : an open toolset for modelling and reasoning in event-b. *STTT*, 12(6) :447–466, 2010.

[5] Jean-Raymond Abrial, Michael J. Butler, Stefan Hallerstede, and Laurent Voisin. A roadmap for the rodin toolset. In *ABZ*, page 347, 2008.

[6] Jean-Raymond Abrial and Dominique Cansell. Click'n prove : Interactive proofs within set theory. In *TPHOLs*, pages 1–24, 2003.

[7] Manamiary Bruno Andriamiarina and Dominique MÃ©ry. Stepwise development of distributed vertex coloring algorithms. Rapport technique, July 2011.

[8] R.-J. Back and J. von Wright. *Refinement Calculus A Systematic Introduction*. Graduate Texts in Computer Science. Springer-Verlag, 1998.

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

[10] Ralph-Johan Back and Reino Kurki-Suonio. Decentralization of process nets with centralized control. *Distributed Computing*, 3(2) :73–87, 1989.

[11] Nazim Benaissa. Modelling Attacker's Knowledge for Cascade Cryptographic Protocols. In Egon Börger, Michael Butler, Jonathan P. Bowen, and Paul Boca, editors, *First International Conference on Abstract State Machines, B and Z - ABZ 2008*, volume 5238 of *Lecture Notes in Computer Science*, pages 251–264, London, United Kingdom, 2008. Springer.

[12] Nazim Benaissa. *La composition des protocoles de sécurité avec la méthode B événementielle*. PhD thesis, Université Henri Poincaré - Nancy I, May 2010.

[13] Nazim Benaissa, Dominique Cansell, and Dominique Mery. Integration of Security Policy into System Modeling. In *The 7th International B Conference - B2007*, Besançon, France, January 2007.

[14] Nazim Benaissa and Dominique MÃ©ry. Cryptographic Protocols Analysis in Event B. In *Seventh International Andrei Ershov Memorial Conference «PERSPECTIVES OF SYSTEM INFORMATICS» - PSI 2009*, Lectures Notes in Computer Science, Novosibisrk, Russie, FÃ©dÃ©ration De, November 2009. Springer-Verlag.

[15] Nazim Benaissa and Dominique MÃ©ry. Cryptologic protocols analysis using proof-based patterns. In *Seventh International Andrei Ershov Memorial Conference "PERSPECTIVES OF SYSTEM INFORMATICS" - PSI 2009*, Lecture Notes in Computer Science, Novosibirsk, Russie, FÃ©dÃ©ration De, June 2009. Springer-Verlag.

[16] Nazim Benaissa and Dominique MÃ©ry. DÃ©veloppement combinÃ© et prouvÃ© de systÃ¨mes transactionnels cryptologiques. In *Approches Formelles dans l'Assistance au DÃ©veloppement de Logiciels - AFADL 2009*, Toulouse, France, January 2009.

[17] Nazim Benaissa and Dominique MÃ©ry. Proof-Based Design of Security Protocols. In Ernst W. Mayr, editor, *5th International Computer Science Symposium in Russia, CSR 2010*, volume 6072 of *Lecture Notes in Computer Science*, pages 25–36, KAZAN, Russie, FÃ©dÃ©ration De, June 2010. Farid Ablayev, Springer.

[18] Dines Bjørner and Martin C. Henson, editors. *Logics of Specification Languages*. EATCS Textbook in Computer Science. Springer, 2007.

[19] Dominique Cansell. Click'N'Prove. http ://plateforme-qsl.loria.fr/click

[20] Dominique Cansell and Dominique Méry. *The event-B Modelling Method : Concepts and Case Studies*, pages 33–140. Springer, 2007. See [18].

[21] Dominique Cansell, Dominique MÃ©ry, and Joris Rehm. Time Constraint Patterns for Event B Development. In Jacques Julliand and Olga Kouchnarenko, editors, *7th International Conference of B Users, January 17-19, 2007*, volume 4355 of *Lecture Notes in Computer Science*, pages 140–154, Besançon, France, 2007. Springer-Verlag. ISSN : 0302-9743 (Print) ; 1611-3349 (Online) ; ISBN : 978-3-540-68760-3.

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

[23] E. M. Clarke, O. Grunberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.

[24] ClearSy. Atelier B. `http://www.atelierb.eu/`.

[25] ClearSy, Aix-en-Provence (F). *B4FREE*, 2004. http ://www.b4free.com.

[26] ClearSy, Aix-en-Provence (F). *BART*, 2010. http ://www.atelierb.eu.

[27] ClearSy, www.atelierb.eu. *Atelier B, Version 4.4.2*, 2016.

[28] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes*. PhD thesis, Université Scientifique et Médicale de Grenoble Institut National Polytechnique de Grenoble, 21 mars 1978.

[29] P. Cousot. Interprétation abstraite. *Technique et science informatique*, 19(1-2-3) :155–164, January 2000.

[30] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings Records of Sixth Proceedings of the Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. The ACM Press.

[31] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4) :511–547, 1992.

[32] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[33] R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Proc. Symp. Appl. Math. 19, Mathematical Aspects of Computer Science*, pages 19 – 32. American Mathematical Society, 1967.

[34] Frama-C. http ://www.frama-c.com.

[35] Stefan Hallerstede and Thai Son Hoang. Qualitative probabilistic modelling in event-b. In *IFM*, pages 293–312, 2007.

[36] Heinrich-Heine-Universität DÃ$\frac{1}{4}$sseldorf. *The ProB Animator and Model Checker*. http ://www.stups.uni-duesseldorf.de/ProB.

[37] Thai Son Hoang, Andreas Furst, and Jean-Raymond Abrial. Event-b patterns and their tool support. In *SEFM*, pages 210–219, 2009.

[38] Thai Son Hoang, Hironobu Kuruma, David A. Basin, and Jean-Raymond Abrial. Developing topology discovery in event-b. *Sci. Comput. Program.*, 74(11-12) :879–899, 2009.

[39] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the Association for Computing Machinery*, 12 :576–580, 1969.

[40] G. Holzmann. The spin model checker. *IEEE Trans. on software engi*.

[41] IEEE. *IEEE Standard for a High Performance Serial Bus. Std 1394-1995*, August 1995.

[42] L. Lamport. Sometime is sometimes Not never : A tutorial on the temporal logic of programs. In *Proceedings of the Seventh Annual Symposium on Principles of Programming Languages*, pages 174–185. ACM SIGACT-SIGPLAN, ACM, 1980.

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

[44] L. Lamport. *Specifying Systems : The TLA$^+$+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[45] Leslie Lamport. *Specifying Systems : The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[46] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[47] Dominique Méry and Neeraj Kumar Singh. Analysis of dsr protocol in event-b. In *13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2011)*. Springer Verlag, LNCS, 2011.

[48] Dominique Méry and Neeraj Kumar Singh. Automatic code generation from event-b models. In *Proceedings of the 2011 Symposium on Information and Communication Technology (SoICT)*, Hanoi, Vietnam, 2011. ACM, ACM International Conference Proceeding Series.

[49] Dominique Méry and Neeraj Kumar Singh. Formal development and automatic code generation : Cardiac pacemaker. In *International Conference on Computers and Advanced Technology in Education (ICCATE 2011)*. Communications in Computer and Information Science (CCIS), Springer, 2011.

[50] Dominique Méry and Neeraj Kumar Singh. Formal specification of medical systems by proof-based refinement (in press). *ACM Transaction Embedded Computing Systems*, 2011.

[51] Dominique Méry and Neeraj Kumar Singh. Formalisation of the heart based on conduction of electrical impulses and cellular-automata. In *International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2011)*. LNCS, 2011.

[52] Dominique Méry and Neeraj Kumar Singh. A generic framework : from modeling to code. *Fourth IEEE International workshop UML and Formal Methods (UML&FM'2011), (To appear in special issue of ISSE NASA Journal, Innovations in Systems and Software Engineering) (in Press)*, 2011.

[53] Dominique Méry and Neeraj Kumar Singh. Medical protocol diagnosis using formal methods. In *International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2011)*. LNCS, 2011.

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

[55] Carroll Morgan, Thai Son Hoang, and Jean-Raymond Abrial. The challenge of probabilistic *vent b* - extended abstract. In *ZB*, pages 162–171, 2005.

[56] Mohammed Mosbah. VISIDIA. http ://visidia.labri.fr.

[57] Dominique MÃ©ry. A Simple Refinement-based Method for Constructing Algorithms. *ACM SIGCSE Bulletin*, 41(2) :51–59, June 2009.

[58] Dominique MÃ©ry. Refinement-Based Guidelines for Algorithmic Systems. *International Journal of Software and Informatics*, 3(2-3) :197–239, September 2009.

[59] Dominique MÃ©ry, Mohamed Mosbah, and Mohamed Tounsi. Refinement-based Verification of Local Synchronization Algorithms. In *17TH INTERNATIONAL SYMPOSIUM ON FORMAL METHODS*, Lecture Notes in Computer Science, page Ã paraÃ®tre, Limerick, Irlande, June 2011. Springer.

[60] Dominique MÃ©ry, Mohammed Mosbah, and Mohammed Tounsi. Proving Distributed Algorithms by Combining Refinement and Local Computations. In Jens Bendisposto, Michael Leuschel, and Markus Roggenbach, editors, *AVOCS 2010 10th International Workshop on Automated Verification of Critical Systems*, Dusseldorf, Allemagne, September 2010.

[61] Dominique MÃ©ry and Neeraj Kumar Singh. EB2C : A Tool for Event-B to C Conversion Support. http ://eb2all.loria.fr.

[62] Dominique MÃ©ry and Neeraj Kumar Singh. Pacemaker's Functional Behaviors in Event-B. Research report, 2009.

[63] Dominique MÃ©ry and Neeraj Kumar Singh. Functional Behavior of a Cardiac Pacing System. *International Journal of Discrete Event Control Systems (IJDECS)*, Digital Equipment Corporation 2010.

[64] Dominique MÃ©ry and Neeraj Kumar Singh. Technical Report on Formal Development of Two-Electrode Cardiac Pacing System. Research report, February 2010.

[65] Dominique MÃ©ry and Neeraj Kumar Singh. Trustable Formal Specification for Software Certification. In T. Margaria and B. Ste, editors, *4th International Symposium On Leveraging Applications of Formal Methods - ISOLA 2010*, volume 6416 of *Lecture Notes in Computer Science*, pages 312–326, Heraklion, Crete, Greece, October 2010. Springer.

[66] Joris Rehm. *Gestion du temps par le raffinement*. PhD thesis, Université Henri Poincaré - Nancy I, Digital Equipment Corporation 2009.

[67] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. Pat : Towards flexible verification under fairness. volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009.

[68] The TLA+ Proof System (TLAPS). https ://tla.msr-inria.inria.fr/tlaps/content/Home.html.

[69] A. Turing. On checking a large routine. In *Conference on High-Speed Automatic Calculating Machines*. University Mathematical Laboratory,Cambridge, 1949.