NOTES SUR LA VÉRIFICATION

PREPRINT

Dominique Méry *
LORIA
Université de Lorraine
dominique.mery@loria.fr
https://members.loria.fr/Mery
https://mery54.github.io/mery/

January 8, 2025

ABSTRACT

This document describes the Event-B modelling language and illustrates the various elements implemented as part of the Rodin platform. Jean-Raymond Abrial's book is the reference and this document is a kind of user's guide providing elements of understanding for students. The Event-B modelling language is also supported by the Atelier-B platform, a different environment that integrates both the B language and the B System language corresponding to Event-B. We provide elements on compatibility and proof obligations.

Keywords Formal method · modelling · proving · refinement · Event-B · B System · safety · reactive systems · invariant · event · abstract machine · context · Formal IDE

*Supported by the ANR Project EBRP-EventB-Rodin-Plus (ANR-19-CE25-0010) and by the ANR Project DISCONT (ANR-17-CE25-0005)

Contents

1 Modélisation des Systèmes Réactifs **3**

2 Modélisation des ensembles et des constantes dans un contexte D de Event-B **11**

3 Bibliography **14**

1 Modélisation des Systèmes Réactifs

Nous rappelons quelques éléments de base pour modéliser ce qu'on appelera un système réactif dans ce chapitre. La modélisation d'un système est réalisée en utilisant des recettes bien connues que nous allons préciser dans un premier temps.

Definition 1 (transition system)

Un système de transition $\mathcal{ST}$ est la donnée d'un ensemble d'états Σ, d'un ensemble d'états initiaux $Init$ et d'une relation binaire $\mathcal{R}$ sur Σ.

L'ensemble d'états terminaux $Term$ détermine des états spécifiques identifiant des états particuliers associés à un état de terminaison; cet ensemble peut être vide et, dans ce cas, le système de transition 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 utilisons l'expression *système* plutôt que programme, dans la mesure où nous pouvons décrire des entités plus générales que des programmes au sens informatique mais aussi que ce formalisme est aussi utilisable pour les applications interactives, concurrentes, réparties ou encore hybrides[2]. Notre définition est générale mais nous allons l'appliquer aux systèmes discrets dans un premier temps. L'idée est d'exprimer les transformations observées sur les états du système concret. Avant de modéliser un système concret par un système de transition, nous devons observer ce qui constitue l'état du système concret et induire les transformations qui opèrent sur cet état. Une transformation peut être due à un évènement qui récupère une température via un capteur, ou bien un calculateur mettant à jour une variable informatique ou encore un actionneur envoyant un signal à une entité sous contrôle:

- un état $s \in \Sigma$ permet d'observer des éléments et rend compte de ces éléments comme le nombre de personnes `np` dans la salle de réunion ou la capacité `cap` de la salle: $s(np)$ et $s(cap)$ sont deux entiers positifs.
- une relation entre deux états s et s' observe une transformation de l'état s en un état s' et on notera $s \xrightarrow{R} s'$ qui exprime l'observation d'une relation R: $R = s(np) \in 0..s(cap)-1 \wedge s'(np) = s(np)+1 \wedge s'(cap) = s(cap)$ est une expression de R observant qu'une personne de plus est entrée dans la salle.
- une trace $s_0 \underset{R_0}{\longrightarrow} s_1 \underset{R_1}{\longrightarrow} s_2 \underset{R_2}{\longrightarrow} s_3 \underset{R}{\longrightarrow} \ldots \underset{R_{i-1}}{\longrightarrow} s_i \underset{R_i}{\longrightarrow} \ldots$ est une trace engendrée par les différentes observations $R_0, \ldots R_p, \ldots$

Nous insistons sur le fait que nous observons des changements d'états correspondant soit à des phénomènes physiques ou biologiques soit à des structures artéfactuelles comme un programme, un service ou une plate-forme. 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. The properties of a system S which interest us are the state properties expressing that *nothing bad can happen*. In other words, we wish to express state properties such as *the number of people in the meeting room is always smaller than the maximum allowed by law* or *the computer variable storing the number of wheel revolutions is sufficient and no overflow will happen*. A. van Gasteren et G. Tel (van Gasteren and Tel 1990) apporte un commentaire très important dans la définition de ce qui est *toujours vrai* et de ce qui est *invariant* et nous choisissons de désigner les propriétés d'états toujours vraies comme des propriétés de sûreté. Les propriétés de sûreté sont, par exemple, la correction partielle d'un algorithme A par rapport à ses spécifications, l'absence d'erreurs à l'exécution ...Les propriétés sont exprimées dans le langage $\mathcal{L}$ dont les éléments sont combinés par des connecteurs logiques ou par des instanciations de valeurs de variables au sens informatique appelées flexibles(Lamport 1994). Nous supposons qu'un système S est modélisé par un ensemble d'états Σ, et que $\Sigma \stackrel{def}{=} \texttt{Var} \longrightarrow$ D où `Var` est la variable ou la liste des variables du système S et D est le domaine des valeurs possibles des variables. Le langage d'assertions $\mathcal{L}$ permet de définir des formules du calcul des prédicats du premier ordre avec

[2]Nous verrons que les systèmes dits hybrides nécessiteront un traitement particulier en lien avec des propriétés mathématiques mettant à contribution des domaines discrets et des domaines continus notamment les espaces de Hilbert. Ces éléments feront l'objet de deux chapitres dédiés dans le second ouvrage de cette série(Méry and Singh ???) qui complétera la présentation de la modélisation Event-B .

des opérations et des opérateurs de la théorie des ensembles. L'interprétation d'une formule P en un état $s \in \Sigma$ est notée $[\![P]\!](s)$ ou parfois $s \in \hat{P}$. Cette hypothèse permet de passer d'une assertion à l'ensemble des états validant cette assertion. La définition de la validité d'une assertion de $\mathcal{L}$ peut être donnée sous une forme inductive de $[\![P]\!](s)$. L'objectif n'est pas de donner une définition complète mais de donner une idée sur l'interprétation des formules afin d'exposer les éléments propres au langage Event-B . On distingue les symboles de variables flexibles x et les symboles de variables logiques v et on utilise des symboles de constantes c.

Exemple 1

1. $[\![\mathtt{x}]\!](s) = s(\mathtt{x}) = x$: x est la valeur de la variable x en s.
2. $[\![\mathtt{x}]\!](s') = s'(\mathtt{x}) = x'$: x' est la valeur de la variable x en s'.
3. $[\![c]\!](s)$ est la valeur de c en s. c'est-à-dire la valeur de la constante c en s.
4. $[\![\varphi(x) \wedge \psi(x)]\!](s)$ = $[\![\varphi(x)]\!](s)$ et $[\![\psi(x)]\!](s)$ où et est l'interprétation classique du symbole $\wedge$ selon sa table de vérité.
5. $[\![\mathtt{x} = 6 \wedge y = \mathtt{x} + 8]\!](s) \stackrel{def}{=} [\![\mathtt{x}]\!](s) = [\![6]\!](s)$**et**$[\![y]\!](s) = [\![x]\!](s) + [\![8]\!](s) = (x = 6$**et**$y = x + 8$ où y est une variable logique différente de x et où $[\![\mathtt{x}]\!](s) = s(\mathtt{x}) = x$.

Nous utilisons des notations simplifiant la référence aux états; ainsi, $[\![\mathtt{x}]\!](s)$ est la valeur de x en s et le nom de la variable x et sa valeur seront distinguées par la fonte utilisée: x est la fonte tt de LaTeX et x est la fonte math de LaTeX.

De cette façon, on pourra utiliser le nom;de la variable x comme sa valeur courante c'est-à-dire x et $[\![\mathtt{x}]\!](s')$ est la valeur de x en s' et sera notée x'. Ainsi, $[\![x = 6]\!](s) \wedge [\![y = x + 8]\!](s')$ 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' en utilisant la notation prime comme Lamport l'a défini pour TLA (Lamport 1994). Nous distinguons plusieurs types de variables selon que l'on parle de la variable informatique, de sa valeur ou encore si on définit des constantes comme np le nombre de processus ou encore pi qui désigne la constante π. Dans l'approche Event-B , une observation courante fait référence à un état courant à la fois pour les données endurantes et les données perdurantes au sens de l'approche de Dines Bjorner (Bjørner 2021).

gindefinition(variables)

Une variable non-logique x est un nom associé à une adresse et à une information variable selon l'état de cette variable:

- x est la valeur courante de x c'est-à-dire la valeur au moment de l'observation.
- x' est la valeur suivante de x c'est-à-dire la valeur au moment de l'observation suivante.
- x_0 est la valeur initiale de de x c'est-à-dire la valeur au moment de l'observation initiale de x.

Pour un système donné S, on notera $\mathcal{V}(S)$ (resp. $\mathcal{V}ar(S)$) l'ensemble des variables logiques (non logiques) du système S. Les variables non-logiques sont des noms utilisés dans l'écriture des modèles associés à S et cette ensemble est utilisé pour distinguer ces variables des autres variables. Lors d'une observation d'un système S, on désire exprimer des relations entre les variables non-logiques de ce système et nous noterons une telle expression sous la forme d'une assertion de la forme $P(x)$ où x est une variable non-logique ou une liste de variables non-logiques. L'ensemble $\mathcal{V}ar(S)$ est mis sous la forme *x* c'est-à-dire que $x \in 1..n \rightarrow \mathcal{V}ar(S)$ où n est le nombre de variables non-logiques du système S. On pourra écrire $x = x_1 \ldots x_n$. Nous avons défini les variables non-logiques qui permettent de lier les valeurs du domaine D du système que nous sommes en train d'observer. L'observation d'un système S revient à déterminer les valeurs observées du domaine D.

Definition 2 (propriété d'état d'un système)

Soit un système S dont les variables *x* sont les éléments de $\mathcal{V}ar(S)$. Une propriété $P(x)$ de S est une expression logique mettant en jeu ,librement la variable non)-logique x et dont l'interprétation est l'ensemble des valeurs du domaine de x: $P(x)$ est vraie en x, si la valeur *x* satisfait $P(x)$.eor

Pour chaque propriété $P(x)$ on peut associer un sous-ensemble de D noté $\hat{P}$ et dans ce cas $P(x)$ est vrai en x équivaut à $x \in \hat{P}$.

Quelques exemples de propriété peuvent être donnés:

- $P_1(x) \stackrel{def}{=} x \in 18..22$: x est une valeur comprise entre 18 et 22 et $\hat{P_1} = \{18, 19, 20, 21, 22\}$.
- $P_2(p) \stackrel{def}{=} p \subset PEOPLE \wedge card(p) \leq n$: p est un ensemble de personnes et cet ensemble a au plus n éléments et $\hat{P_2} = \{p_1 \dots p_n\}$. Dans cet exemple, on utilise une variable logique n et un nom de constante $PEOPLE$.

Dans notre dernier exemple, nous avons utilisé $PEOPLE$ qui représente un ensemble de personnes et donc nous utiliserons pour écrire nos expressions. On notera une liste de symboles $s_1, s_2, \dots, s_p$ correspondant à des symboles d'ensembles qui constituent le domain D.

Definition 3 (ensemble de base du système S)

La liste de symboles $s_1, s_2, \dots, s_p$ correspond à la liste des symboles d'ensembles de base du domaine D de S et $s_1 \cup \dots \cup s_p\} \subseteq$ D.

Enfin, pour modéliser un système S, il est nécessaire de disposer de symboles de constantes et des symboles de fonctions.

Definition 4 (constante de base du système S)

La liste de symboles $c_1, c_2, \dots, c_q$ correspond à la liste des symboles de constantes de base de S.

Definition 5 (fonction du système S)

La liste de symboles $f_1, f_2, \dots, f_r$ correspond à la liste des symboles de fonctions de S.

En fait, nous reprenons la partition classique des langages logiques qui séparent les constantes d'un côté des symboles de fonctions de l'autre. Nous donnons quelques exemples qui permettront de fixer les idées.

- $fred$ est une constante et on la lie à l'ensemble $PEOPLE$ en utilisant l'expression $fred \in PEOPLE$ qui signifie que $fred$ est une personne de $PEOPE$.
- aut est une constante qui permet d'exprimer la table ds autorisations associée à l'usage de véhicules; nous utilisons l'expression $aut \subseteq PEOPLE \times CARS$ où $CARS$ déigne un ensemble de voitures.

Les constantes de S recouvrent les constantes de base et les fonctions de S. Chaque constante c de S doit être définie par une liste d'expressions appelés axiomes.

Definition 6 (axiome du système S)

Un axiome ax(s,c) de S est une expression décrivant une constante ou des constantes de S et peut être définie comme une expression dépendant des symboles de constantes exprimant une expression logico-ensembliste mettant en œuvre les symboles d'ensembles et les symboles de constantes déjà définies.

Nous donnons quelques exemples d'axiomes qui peuvent être définis:

- $ax1(fred \in PEOPLE)$: *fred est une personne de l'ensemble PEOPLE*
- $ax2(suc \in \mathbb{N} \to \mathbb{N} \wedge (!i.i \in \mathbb{N} \Rightarrow suc(i) = i + 1))$: *La fonction suc est la fonction totale qui associe à tout naturel i son successeur.*
- $ax3(\forall A.A \subseteq \mathbb{N} \wedge 0 \in \mathbb{N} \wedge suc[A] \subseteq A \Rightarrow \mathbb{N} \to \subseteq A)$: Cet axiome ébonce la propriété d'induction sur les naturels.

- $ax4(\forall x.x = 2 \Rightarrow x + 2 = 1)$: Cet axiome pose un problème manifeste de consistance et il faudra prendre soin de ne pas utiliser ce genre d'axiome.

Nous avons numéroté les axiomes et nous utiliserons cette numérotation pour définir les axiomes du système S. Une hypothèse est que les axiomes sont consistants. Pour tout système, nous utiliserons une liste d'axiomes permettant de décrire les constantes de S.

Definition 7 (axiomatique pour S)

La liste des axiomes de S est appelée axiomatique de S et est notée $AX(S, s, c)$ où s désigne les ensembles de base et c désigne les constantes de S.

Advice (Consistance de l'axiomatique pour S)

La vérification de la consistance de $AX(S, s, c)$ est un élément important. à ne pas négliger dans la modélisation d'un système. Il est assez facile d'introduire une inconsistance et les outils comme Rodin fournissent la technique ProB fondée sur la découverte d'un modèle au sens logique. Cependant cette technique a ses limites et il convient d'être très prudent.

Nous avons défini un système axiomatique $AX(S, s, c)$ pour le système S et nous allons maintenant dériver ds propriétés à partir de ce système. Ces propriétés seront prouvées à partir de ce système axiomatique et seroe xemplnt les théorèmes pour S.

Definition 8 (théorème de S)

Une propriété $P(s, c)$ est un théorème pour S si $AX(S, s, c) \vdash P(s, c)$ est un séquent valide.

On notera $TH(S, s, c)$ les théorèmes pour S.

Nous avons montré comment on organisait les définitions des ensembles de bases, des constantes, des axiomes et des théorèmes. Les variables non-logiques bénéficient d'une qualité essentielle, puisqu'elles permettent de rendre compte de l'état du système en cours d'observation.

Soient σ, σ' deux états de S ($\sigma, \sigma' \text{VAR} \longrightarrow$ D). $s \underset{R}{\longrightarrow} s'$ s'écrira comme une relation R(x, x') où x et x' désignent des valeurs de x avant et après l'observation de R. Nous définissons ce qu'est un événement observé sur les *variables non-logiques* du système observé S.

Definition 9 (événement)

Soit $\mathcal{V}ar(S)$ l'ensemble des variables non-logiques de S noté sous la forme x. Soient s les ensembles de base et c les constantes de S. Un événement e pour S est une expression relationnelle de la forme $R(s, c, x, x')$ notée $BA(e)(s, c, x, x')$.

Cette définition est empruntée directement à TLA (Lamport 1994) et simplifie l'exposé de nos préliminaires. En revanche, nous n'emprunterons pas le langage de la théorie des ensembles de TLA$^+$ (Lamport 2002) . Nous donnons quelques exemples d'événements:

- $BA(e_1)(x, x') \stackrel{def}{=} x' = x + 1$: e_1 observe l'accroissement de x d'une unité.
- $BA(e_2)(x, y, x', y') \stackrel{def}{=} x' + y' = x + y$: e_2 observe que les valeurs de x et de y évoluent de telle façon à ce que la somme des deux variables soit constante.

Convention (méta-langage des preuves)

Nous choisissons d'utiliser des expressions logiques de la forme $P \Rightarrow Q$ ou $\forall x.P(x)$ ou $\forall x \in E.P(x)$ dans le méta-langage des preuves et des expression formelles. Cela permettra de présenter des résultats fondamentaux sur les principes d'induction ou sur les conditions de vérification. Puis, nous exprimerons les conditions de vérification sous unes forme plus proche des outils logiques qui sont utilisés .

A la suite d'une observation d'un système S, un ensemble d'événements E est identifié et on obtient un modèle événementiel du système S.

Definition 10 Modèle événementiel d'un système

Soit $\mathcal{V}ar(S)$ l'ensemble des variables non-logiques de S noté sous la forme $\mathtt{x}$. Soit s la liste des ensembles de base du système S. Soit c la list des constantes du systèmes S. Soit D un doçmaine contenant les ensembles s. Un modèle événementiel $\mathcal{M}$ pour un système S est défini par

$$(AX(s,c), \mathtt{x}, \mathrm{D}, Init(x), \{e_0, \ldots, e_n\})$$

où

- $AX(s,c)$ est une théorie axiomatique définissant les ensembles, les constantes et les propriétés statiques de ces éléments.
- $Init(x)$ définit les valeurs initiales possibles de $\mathtt{x}$.
- $\{e_0, \ldots, e_n\}$ est un ensemble fini d'événements de S et e_0 est un événement particulier présent dans chaque modèle événementiel défini par $BA(e_0)(x, x') = (x' = x)$.

Ce modèle événementiel est noté $EM(s, c, \mathtt{x}, \mathrm{D}, Init(x)\{e_0, \ldots, e_n\}) = (AX(s,c), \mathtt{x}, \mathrm{D}, Init(x), \{e_0, \ldots, e_n\})$.

A partir de ce modèle, on peut définir une relation $Next(x, x')$ comme suit $Next(s, c, x, x') \stackrel{def}{=} BA(e_0)(s, c, x, x') \vee \ldots \vee BA(e_n)(s, c, x, x')$. La modélisation d'un système comprend la donnée de la variable $\mathtt{x}$, du prédicat $Init(x)$ caractériant les valeurs initiales des variables et une relation $Next(s, c, x, x')$ modélisant la relation entre les valeurs avant et les valeurs après. Les propriétés de sûreté expriment que *rien de mauvais ne peut arriver (Lamport 1980)*. Par exemple, la valeur de la variable $\mathtt{x}$ est toujours comprise entre 0 et 567; la somme des valeurs courantes des variables $\mathtt{x}$ et y est égale à la valeur courante de la valeur z. Pour poursuivre nos descriptions, nous devons introduire la clôture réflexive transitive de la relation $Next(s, c, x, x')$,

$$Next^{\star}(s, c, x_0, x) \stackrel{def}{=} \begin{cases} \vee\ x_0 = x \\ \vee\ Next(s, c, x_0, x) \\ \vee\ \exists\ xi \in \mathrm{D}.Next^{\star}(s, c, x_0, xi) \wedge Next(s, c, xi, x) \end{cases}$$

Definition 11 (propriété de sûreté)

Une propriété $P(x)$ est une propriété de sûreté pour le système S, si

$$\forall x_0, x \in \mathrm{D}.Init(x_0) \wedge Next^{\star}(s, c, x_0, x) \Rightarrow P(x).$$

La propriété de sûreté utilise une expression universelle pour quantifier les valeurs possibles de la variable $\mathtt{x}$. Pour démontrer une telle propriété, on peut soit vérifier la propriété pour chaque valeur possible de $\mathtt{x}$ dans le domaine D,à condition que cet ensemble soit fini, ou bien utiliser une abstraction du domaine Dou bien utiliser un principe d'induction. Pour ce qui est de la vérification pour chaque valeur possible, on utilise un algorithme de calcul des valeurs accessibles à partir d'un état initial. Cette technique de calcul des valeurs accessibles est souvent utilisée et est à la base

des techniques de vérification automatique comme le *model checking* (McMillan 1993 ; Holzmann 1997 ; Clarke et al. 2000) Nous envisageons une technique inductive, pour prouver les propriétés de sûreté. Nous observons la propriété logique suivante entre les deux équations:

$$\forall x_0, x \in \mathrm{D}.Init(x_0) \wedge Next^{\star}(s, c, x_0, x) \Rightarrow P(x) \tag{1}$$

$$\forall x \in \mathrm{D}.(\exists x_0 \in \mathrm{D}.Init(x_0) \wedge Next^{\star}(s, c, x_0, x)) \Rightarrow P(x) \tag{2}$$

Ainsi, la seconde équation exprime que les valeurs accessibles sont des valeurs sûres par rapport à $P(x)$ et nous donne la clé du principe d'induction à mettre en œuvre. En fait, la propriété $(\exists x_0 \in \mathrm{D}.Init(x_0) \wedge Next^{\star}(s, c, x_0, x))$ définit un plus petit point-fixe qui est une propriété inductive.

Property 1 (Principe d'induction)

Une propriété $P(x)$ est une propriété de sûreté pour S, si, et seulement si, il existe une propriété $I(x)$ telle que

$$\begin{cases} (1)\ Init(x) \Rightarrow I(x) \\ (2)\ I(x) \Rightarrow P(x) \\ (3)\ \forall x, x' \in \mathrm{D}.I(x) \wedge Next(s, c, x, x') \Rightarrow I(x') \end{cases}$$

La propriété $I(x)$ est appelée un invariant 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.

Cette propriété justifie la méthode de preuve par induction plus connue comme la méthode de Floyd/Hoare (Floyd 1967 ; Hoare 1969), imaginée par Turing en 1949 (Turing 1949). La propriété énoncée donne une forme de l'induction qu'il faut ramener à des formes plus connues. P. et R. Cousot (Cousot 2000 ; Cousot and Cousot 1979, 1992 ; Cousot 1978) donne une synthèse complète sur les principes d'induction équivalents à ce principe d'induction. Nous appliquons ces résultats au cas des modèles événementiels de système et nous obtenons une expression de la définition d'une propriété de sûreté dans le cas d'un modèle relationnel de système. 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 modèles événementiels.

Property 2

Les deux énoncés suivants sont équivalents:

(I) Il existe une propriété d'état $I(x)$ telle que:

$$\begin{cases} (1)\ Init(x) \Rightarrow I(x) \\ (2)\ I(x) \Rightarrow P(x) \\ (3)\ \forall x, x' \in \mathrm{D}.I(x) \wedge Next(s, c, x, x') \Rightarrow I(x') \end{cases}$$

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

$$\begin{cases} (1)\ Init(x) \Rightarrow I(x) \\ (2)\ I(x) \Rightarrow P(x) \\ (3)\ \forall i \in \{0, \ldots, n\} : \forall x, x' \in \mathrm{D}.I(x) \wedge BA(e_i)(s, c, x, x') \Rightarrow I(x') \end{cases}$$

Nous avons ainsi donné une explication de la règle d'induction qui est utilisée dans la méthode de Floyd (Floyd 1967 ; Turing 1949 ; Hoare 1969); 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. Les conditions de vérification suivantes sont dérivées à partir des propriétés ci-dessus.

Definition 12 (Conditions de vérification pour un système S et ses propriétés d'invariance et de sûreté)

Soit $(AX(s,c), \mathtt{x}, \mathrm{D}, Init(x), \{e_0, \ldots, e_n\})$ un modèle événementiel pour le système S. Les conditions de vérification pour un système S et ses propriétés d'invariance I(x) et de sûreté A(x) sont les suivantes:

- $AX(s,c) \vdash \forall x \in \mathrm{D} : Init(x) \Rightarrow I(x)$
- Pour tout événement e de S, $AX(s,c) \vdash \forall x, x' \in \mathrm{D} : I(x) \wedge BA(e)(x,x') \Rightarrow I(x')$
- $AX(s,c) \vdash \forall x \in \mathrm{D} : I(x) \Rightarrow A(x)$

Avant de poursuivre cette présentation dédiée à Event-B , nous voulons rappeler qu'Event-B est un langage support d'une approche de développement et de modélisation de systèmes corrects par construction. Par conséquent, le modèle événementiel du système S $\mathcal{M} = (AX(s,c), \mathtt{x}, \mathrm{D}, Init(x), \{e_0, \ldots, e_n\})$ (définition 10) est caractérisé par l'existence de points-fixes sur le treillis complet $(\mathbb{P}(D), \subseteq)$.

Property 3

Soit D le domaine de valeurs du modèle événementiel $\mathcal{M} = (AX(s,c), \mathtt{x}, \mathrm{D}, Init(x), \{e_0, \ldots, e_n\})$.

Soit $Next(s,c,x,x') \stackrel{def}{=} BA(e_0)(s,c,x,x') \vee \ldots \vee BA(e_n)(s,c,x,x')$.

Soit $INIT = \{d | d \in \mathrm{D} \wedge Init(d)\}$.

- $(\mathbb{P}(\mathrm{D}), \subseteq)$ est un treillis complet.
- La fonction $F = \lambda X \in \mathbb{P}(D).(INIT \cup Next[X]$ est monotone croissante et l'équation $X = F(X)$ admet un ensemble non-vide de solutions formant un treillis complet.
- Toute solution I de l'équation$X = F(X)$ vérifie la propriété suivante: $I = F(I)$, $INIT \subseteq I$, $Next[I] \subseteq I$, $lfp(F) \subseteq I$.
- Pour toute propriété $P(x)$ de sûreté pour le système S, $lfp(F) \subseteq \{d | d \in \mathrm{D} \wedge P(d)\}$

Une conséquence de cette propriété est qu'on peut associer un invariant canonique à tout modèle événementiel et cet invariant est défini opérationnellement par le plus petit point-fixe de la fonction F. Cette approche est généralement celle de la vérification d'un système. Dans le cas du langage Event-B , nous sommes intéressés par une approche de correction par construction. Par conséquent, le problème est de définir un modèle équipé d'un invariant qui sera vérifié et cette méthode est menée de manière incrémentale avec l'aide du raffinement. Les solutions de l'équation $X = F(X)$ correspondent à ds invariants inductifs et sont utiles pour montrer qu'une propriétét $P(x)$ est une propriété de sûreté. Le triptyque $(\mathcal{M}, I, P)$ met en avant un modèle événementiel ($\mathcal{M}$ dont l'invariant (inductif) permet de vérifier la propriété de sûreté $P(x)$ (en vérifiant les conditions de vérification nécessaires pour la vérification de l'invariance de I et la sûreté de P).

Avant de passer à la présentation des structures pour modéliser les systèmes réactifs, nous nous intéressons à la notion d'événement et à des conditions dites de faisabilité associées à la vérification des modèles. J.-R. Abrial (Abrial 1996) fonde la vérification des machines abstraites sur le calcul wp et utilise la notation $[S]P$ (substitution généralisée), pour exprimer que *S établit P*.Cette expression signifie aussi que S termine en *P* et le prédicat de terminaison est noté trm(e)(s,c,x). Dans notre cas, S est un événement paramétré et nous rappelons les définitions associées à S pour

Event e
  any u where
    $G(u,s,c,x)$
  then
    $x :| BAP(u,s,c,x,x')$
end

.

Property 4

Soit une propriété d'états $P(x)$.

- $[\mathsf{e}]P(x) = \forall u.(G(u,s,c,x) \Rightarrow [x : |BAP(u,s,c,x,x')]P(x))$
- $\mathsf{trm}(\mathsf{e})(x,c,x) = \forall u.(G(u,s,c,x) \Rightarrow \exists x' : BAP(u,s,c,x,x')$
- les deux propriétés suivantes sont équivalentes:
  1. $I(s,c,x) \wedge \mathsf{trm}(e)(s,c,x) \Rightarrow [e]I(s,c,x)$
  2. $\left\{ \begin{array}{l} I(s,c,x) \wedge G(u,s,c,x) \Rightarrow \exists x'.BAP(u,s,c,x,x') \\ I(s,c,x) \wedge G(u,s,c,x) \wedge BAP(u,s,c,x,x') \Rightarrow I(s,c,x) \end{array} \right.$

Les deux premières propriétés sont une simple application des résultats de J.-R. Abrial (Abrial 1996). La troisième propriété est une équivalence entre deux expressions de la préservation d'une propriété d'états par un événement e.

Explanation (Explication de la propriété)

Explication

(1) $\left\{ \begin{array}{l} I(s,c,x) \wedge G(u,s,c,x) \Rightarrow \exists x'.BAP(u,s,c,x,x') \\ I(s,c,x) \wedge G(u,s,c,x) \wedge BAP(u,s,c,x,x') \Rightarrow I(s,c,x) \end{array} \right.$

est équivalent par transformation des connecteurs logiques.

$\left\{ \begin{array}{l} I(s,c,x) \wedge G(u,s,c,x) \Rightarrow \exists x'.BAP(u,s,c,x,x') \\ I(s,c,x) \wedge G(u,s,c,x) \Rightarrow \forall x'.(BAP(u,s,c,x,x') \Rightarrow I(s,c,x')) \end{array} \right.$

est équivalente par conjonction des buts uy $I(s,c,x) \wedge G(u,s,c,x) \Rightarrow \left\{ \begin{array}{l} a\exists x'.BAP(u,s,c,x,x') \\ d\forall x'.(BAP(u,s,c,x,x') \Rightarrow I(s,c,x')) \end{array} \right.$

est équivalente par transformation ds connecteurs logiques

u $I(s,c,x) \wedge G(u,s,c,x) \Rightarrow \left\{ \begin{array}{l} \exists x'.BAP(u,s,c,x,x') \\ ivalente\forall x'.(BAP(u,s,c,x,x') \Rightarrow I(s,c,x')) \end{array} \right.$

est équivalente la propriété du calcul wp

$I(s,c,x) \wedge G(u,s,c,x) \Rightarrow [x : |BAP(u,s,c,x,x')]I(s,c,x)$

est équivalente par transformation des connecteurs logiques

$I(s,c,x) \Rightarrow (G(u,s,c,x) \Rightarrow [x : |BAP(u,s,c,x,x')])I(s,c,x)$

est équivalente par internalisation de la quabtification sur u.

$I(s,c,x) \Rightarrow \forall u.(G(u,s,c,x) \Rightarrow [x : |BAP(u,s,c,x,x')])I(s,c,x)$

est équivalente la propriété du calcul wp

$I(s,c,x) \Rightarrow \forall u.[G(u,s,c,x) ==> x : |BAP(u,s,c,x,x')])I(s,c,x)$

est équivalente la propriété du calcul wp

$I(s,c,x) \Rightarrow [@u.G(u,s,c,x) ==> x : |BAP(u,s,c,x,x')])I(s,c,x)$

est équivalente la propriété du calcul wp

$I(s,c,x) \Rightarrow [\mathsf{e}](s,c,x)$ est équivalente la propriété du calcul wp

$I(s,c,x) \wedge \mathsf{trm}(\mathsf{e}) \Rightarrow [\mathsf{e}](s,c,x)$

Une conséquence de ce résultat est de permettre une définition de la préservation selon deux modes de mise en œuvre (Atelier-B (ClearSy) et Rodin (Abrial et al. 2010)). Nous allons décliner la définition de la préservation selon la forme séparant la vérification en un pas d'induction et en une preuve de faisabilité. En particulier, il définit les conditions de vérification (PO(e)) à l'aide de ces éléments de la façon suivante.

Definition 13 (Condition de vérification init)

init est un événement d'initialisation et on suppose qu'il est défini comme suit: init $\stackrel{def}{=}$ begin x : $|(initBAP(s,c,x')$ end. La condition de vérification pour init (initialisation) notée PO(init) est définie par
$\begin{cases} (FIS)\ AX(s,c) \vdash \exists x'.initBAP(s,c,x') \\ (INV)\ AX(s,c), I(s,c,x), initBAP(s,c,x) \vdash I(s,c,x) \end{cases}$
,

Definition 14 (Condition de vérification e)

La condition de vérification pour e (événement e) notée PO(e) est définie par $AX(s,c) \vdash I(x) \wedge \mathsf{trm}(e) \Rightarrow [e]I(s,c,x)$.

$\begin{cases} (FIS)\ AX(s,c), I(s,c,x), G(u,s,c,x) \vdash \exists x'.BAP(u,s,c,x,x') \\ (INV)\ AX(s,c), I(s,c,x), G(u,s,c,x), BAP(u,s,c,x,x') \vdash I(s,c,x') \end{cases}$

Cette définition met en avant deux conditions INV et FIS à vérifier et permettant d'assurer la préservation de $I(s,c,x)$. Nous avons précisé les éléments que nous allons utiliser pour présenter la mise en œuvre dans le cadre du langage Event-B .

Convention (étiquette des textes formels)

Dans la définition du langage Event-B , la mise en œuvre repose sur un étiqettage de chaque texte formel (axiome, théorème, garde, action ...) et cet étiquettage est noté $\ell(text)$. Cela signifie que l'expression $\ell(text)$: $text$ figure dans le texte de modélisation. Cette étiquettage permet de se régérer à des éléments du modèle.

2 Modélisation des ensembles et des constantes dans un contexte D de Event-B

Un contexte Event-B rassemble les définitions des entités endurantes du système S à développer et donc à modéliser. Noys nous référons au document (Métayer and Voisin 2009) présentant le langage mathématique Event-B constituant le noyau du langage Event-B .

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

- Les constantes c 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.
- $AX(s,c)$ désigne la liste des axiomes correspondant aux ensembles s et aux constantes c.

Expression e	WD(e)
$P \wedge Q, P \Rightarrow Q$	$WD(P) \wedge (P \Rightarrow WD(Q))$
$P \vee Q$	$WD(P) \wedge (WD(P) \wedge (P \vee WD(Q))$
$P \Leftrightarrow Q$	$WD(P) \wedge WD(Q)$
$\neg P$	$WD(P)$
$\forall L.P$, $\exists L.P$	$\forall L.WD(P)$
$\top$, $\bot$	$\top$
$\text{finite}(E)$	$WD(E)$
$partition(E_1, E_2, \ldots, E_n)$	$WD(E_1) \wedge WD(E_2) \wedge \ldots \wedge WD(E_n)$
E op F with op $\in \{=, \neq, \in, \notin, \subset, \not\subset, \subseteq, \not\subseteq\}$	$WD(E) \wedge WD(F)$

Table 1: WD for predicates

Expression e	WD(e)
$F(E)$	$\begin{pmatrix} WD(E) \wedge WD(F) \\ E \in dom(F) \wedge F \in S \pfun T \\ F \subseteq S \times T \end{pmatrix}$
$E[F], E \mapsto F, E \leftrightarrow F, E \nrightarrow F, E \nleftrightarrow F$ $E \nrightarrow F, E \to F, E \pfun F, E \rightarrowtail F, E \rightarrowtail F$ $E \twoheadrightarrow F, E \pinj F, E \rightarrowtail F, E \cup F, E \cap F$ $E \backslash F, E \times F, E \otimes F, E \parallel F, E ; F, E \circ F$ $E \rhd F, E \ndres F, E \rhd F, E \nrhd F, E \nrhd F$ $E..F, E + F, E - F, E * F$	$WD(E) \wedge WD(F)$
E / F, E mod F	$WD(E) \wedge WD(F) \wedge F \neq 0$
E $\widehat{}$ F	$WD(E) \wedge 0 \leq E \wedge WD(F) \wedge 0 \leq F$
$-E, E^{-1}, \mathbb{P}(E), \mathbb{P}_1(E)$ $\text{dom}(E), \text{ran}(E), \text{union}(E)$	$WD(E)$
$\text{card}(E)$	$WD(E) \wedge \text{finite}(E)$
$\text{inter}(E)$	$WD(E) \wedge E \neq \varnothing$
$\min(E)$	$WD(E) \wedge E \neq \varnothing \wedge (\exists v. \forall u. u \in E \Rightarrow v \leq u)$
$\max(E)$	$WD(E) \wedge E \neq \varnothing \wedge (\exists v. \forall u. u \in E \Rightarrow v \geq u)$

Table 2: WD for unary and binary expressions

Chaque expression Event-B doit être bien définie et une condition de vérification est systématiquement produite à partir du texte de la propriété à prouver *th/WD*; la condition de vérification d'établissement que *th* est un théorème est notée *th/TH*. Intuitivement, une expression *e* est bien définie, quand elle obéit à certaines règles d'utilisation, et elle est notée *WD(e)*. Nous avons reproduit les tables définissant le prédicat *WD(e)* selon la syntaxe de *e* et selon les classes des expressions qui sont des prédicates, des relations ...

A partir des tables , et , nous pouvons simplement établir la condition de vérification à prouver notée th/WD.

Proof Obligation th/WD

$AX(s, c) \vdash WF(th)$

Quelques exemples permettent d'illustrer la notation $WD(e)$ et de comprendre les conditions de vérification produites par l'outil, Rodin.

Exemple 2 $(WD(e))$

- $E1 = \forall k \cdot k \in \mathbb{N} \Rightarrow 2 * s(k) = k * k + k$

  Il suffit d'appliquer les transformations des tables , et :

Expression e	WD(e)
$\lambda P.Q\|E$	$\forall \mathcal{F}_Q.WD(P) \wedge (Q \Rightarrow WD(E))$
$\bigcup L.P\|E$ $\{L.P\|E\}$	$\forall L.WD(P) \wedge (P \Rightarrow WD(E))$
$\bigcup E\|P$ $\{E\|P\}$	$\forall l\mathcal{F}_E.WD(P) \wedge (P \Rightarrow WD(E))$
$\bigcap L.P\|E$	$\forall L.WD(P) \wedge (P \Rightarrow WD(E))$ $\wedge$ $\exists L.WD(P)$
$\bigcap E\|P$	$\forall l\mathcal{F}_E.WD(P) \wedge (P \Rightarrow WD(E))$ $\wedge$ $\exists L.WD(P)$
$\mathrm{bool}(P)$	$WD(P)$
$\{E_1, \ldots, E_n\}$	$WD(E_1) \wedge \ldots \wedge WD(E_n)$
$I, \mathbb{Z}, \mathbb{N}, \mathbb{N}_1, pred, succ, \mathrm{BOOL}$ $\mathrm{TRUE}, \mathrm{FALSE}, \varnothing, \mathrm{prj}_1, \mathrm{prj}_2, \mathrm{id}, n$	$\top$

Table 3: WD for other expressions

$WD(\forall k \cdot k \in \mathbb{N} \Rightarrow 2 * s(k) = k * k + k)$
$=$
$\forall k.WD(k \in \mathbb{N} \Rightarrow 2 * s(k) = k * k + k)$
$\forall k.WD(k \in \mathbb{N}) \wedge (k \in \mathbb{N} \Rightarrow WD(2 * s(k) = k * k + k))$
$\forall k.WD(k \in \mathbb{N}) \wedge (k \in \mathbb{N} \Rightarrow WD(2 * s(k)) \wedge WD(k * k + k))$
$\forall k.WD(k \in \mathbb{N}) \wedge (k \in \mathbb{N} \Rightarrow WD(2) \wedge WD(s(k)) \wedge WD(k * k) \wedge WD(k))$
$\forall k.WD(k) \wedge WDS(\mathbb{N}) \wedge (k \in \mathbb{N} \Rightarrow WD(2) \wedge WD(s(k)) \wedge WD(k * k) \wedge WD(k))$
$\forall k.\top \wedge \top \wedge (k \in \mathbb{N} \Rightarrow \top \wedge WD(s(k)) \wedge WD(k * k) \wedge WD(k))$
$\forall k.(k \in \mathbb{N} \Rightarrow WD(s(k)) \wedge WD(k * k) \wedge WD(k))$
$\forall k.(k \in \mathbb{N} \Rightarrow WD(s(k)) \wedge WD(k) \wedge WD(k) \wedge WD(k))$
$\forall k.(k \in \mathbb{N} \Rightarrow WD(s(k)) \wedge \top \wedge \top \wedge \top)$
$\forall k.(k \in \mathbb{N} \Rightarrow WD(s(k)))$
$\forall k.(k \in \mathbb{N} \Rightarrow WD(s) \wedge WD(k) \wedge k \in dom(s) \wedge s \in \mathbb{Z} \nrightarrow \mathbb{Z} \wedge s \subseteq \mathbb{Z} \times \mathbb{Z}))$
$\forall k.(k \in \mathbb{N} \Rightarrow \top \wedge \top \wedge k \in dom(s) \wedge s \in \mathbb{Z} \nrightarrow \mathbb{Z} \wedge s \subseteq \mathbb{Z} \times \mathbb{Z}))$
$\forall k.(k \in \mathbb{N} \Rightarrow k \in dom(s) \wedge s \in \mathbb{Z} \nrightarrow \mathbb{Z} \wedge s \subseteq \mathbb{Z} \times \mathbb{Z}))$

On en déduit la condition $WD(E1)$:
$WD(\forall k \cdot k \in \mathbb{N} \Rightarrow 2 * s(k) = k * k + k) = \forall k.(k \in \mathbb{N} \Rightarrow k \in dom(s) \wedge s \in \mathbb{Z} \nrightarrow \mathbb{Z} \wedge s \subseteq \mathbb{Z} \times \mathbb{Z}))$

- $E2 \stackrel{def}{=} s(0) = 0$
  $WD(s(0) = 0)$
  $=$
  $WD(s(0)) \wedge WD(0)$
  $WD(s(0)) \wedge \top$
  $WD(s(0))$
  $WD(s) \wedge WD(0) \wedge 0 \in dom(s) \wedge \wedge s \in \mathbb{Z} \nrightarrow \mathbb{Z} \wedge s \subseteq \mathbb{Z} \times \mathbb{Z})$
  $\top \wedge \top \wedge 0 \in dom(s) \wedge \wedge s \in \mathbb{Z} \nrightarrow \mathbb{Z} \wedge s \subseteq \mathbb{Z} \times \mathbb{Z})$
  $0 \in dom(s) \wedge \wedge s \in \mathbb{Z} \nrightarrow \mathbb{Z} \wedge s \subseteq \mathbb{Z} \times \mathbb{Z})$

  On en déduit la condition $WD(E2)$:
  $WD(s(0) = 0) = 0 \in dom(s) \wedge \wedge s \in \mathbb{Z} \nrightarrow \mathbb{Z} \wedge s \subseteq \mathbb{Z} \times \mathbb{Z})$

La condition de vérification th/TH est assez simple et exige parfois des exercices d'interaction avec l'assistant de preuve.

Proof Obligation th/TH

$AX(s,c) \vdash th$

Pour terminer cette présentation, nous présentons un exemple de contexte dans le domaine de l'arithmétique. Cela nous permet de poser un problème qui sera utilisé pour illustrer les différentes notations et les différents concepts.

Exemple 3 (somme des entiers naturels pairs ou impairs)

Le problème est de calculer la somme $s(n)$ des entiers naturels entre 0 et un entier donné n et cette somme $s(n)$ est assez évidente à calculer avec la formule classiquement utilisée dans les livres de mathématiques élémentaires: $\forall n.n \in \mathbb{N} \Rightarrow s(n) = \sum_{k=0}^{k=n} = n * (n+1)/2$ ou encore $2 * s(n) = n * (n+1)$. Le problème est de proposer un algorithme qui calcule cette somme et qui respecte la propriété de correction énoncée explicitement par la relation $2 * s(n) = n * (n+2)$. De plus on calculera la somme des entiers pairs $os(n)$ et la somme des entiers impairs $es(n)$. Le problème est de calculer les fonctions s, es, os mais nous devons définir ces fonctions et établir un certain nombre de propriétés inductives.

La première étape est de définir les propriétés axiomatiques (axm1, ... axm7) des ensembles et des constantes nécessaires. L'axiome axm7 est une expression axiomatique de l'induction pour le domaine des naturels et il va faciliter nos preuves par induction que nous devrons établir dans la partie THEOREMS.

AXIOMS
- $axm1 : n \in \mathbb{N}$
- $axm2 : s \in \mathbb{N} \to \mathbb{N} \wedge os \in \mathbb{N} \to \mathbb{N} \wedge es \in \mathbb{N} \to \mathbb{N}$
- $axm3 : es(0) = 0 \wedge os(0) = 0 \wedge s(0) = 0$
- $axm4 : \forall i, l \cdot i \in \mathbb{N} \wedge l \in \mathbb{N} \wedge i = 2 * l$
  $\Rightarrow s(i+1) = s(i) + i + 1 \wedge es(i+1) = es(i) \wedge os(i+1) = os(i) + i + 1$
- $axm5 : \forall i, l \cdot i \in \mathbb{N} \wedge l \in \mathbb{N} \wedge i = 2 * l + 1$
  $\Rightarrow s(i+1) = s(i) + i + 1 \wedge es(i+1) = es(i) + i + 1 \wedge os(i+1) = os(i)$
- $axm6 : suc \in \mathbb{N} \to \mathbb{N} \wedge (\forall i \cdot i \in \mathbb{N} \Rightarrow suc(i) = i + 1)$
- $axm7 : \forall A \cdot A \subseteq \mathbb{N} \wedge 0 \in A \wedge suc[A] \subseteq A \Rightarrow \mathbb{N} \subseteq A$

THEOREMS
- $th1 : \forall i \cdot i \in \mathbb{N} \Rightarrow s(i+1) = s(i) + i + 1$
- $th2 : \forall u, v \cdot u \in \mathbb{N} \wedge v \in \mathbb{N} \wedge 2 * u = v \Rightarrow u = v/2$
- $th3 : \forall k \cdot k \in \mathbb{N} \Rightarrow 2 * s(k) = k * k + k$
- $th4 : \forall k \cdot k \in \mathbb{N} \Rightarrow s(k) = (k * k + k)/2$
- $th5 : \forall k \cdot k \in \mathbb{N} \Rightarrow es(2 * k) = 2 * s(k)$
- $th6 : \forall k \cdot k \in \mathbb{N} \Rightarrow es(2 * k + 1) = 2 * s(k)$
- $th7 : \forall k \cdot k \in \mathbb{N} \wedge k \neq 0 \Rightarrow os(2 * k) = k * k$
- $th8 : \forall k \cdot k \in \mathbb{N} \Rightarrow os(2 * k + 1) = (k+1) * (k+1)$

Les théorèmes reprennent des propriétés assez élémentaires, qui permettent de démontrer les théorèmes th1, th5, th6, th7, th8. Ces théorèmes ont été démontrés par l'outil Rodin avec l'aide de l'axiome axm7. Ils constituent un élément important pour lier la définition inductive des suites et la propriété attendue pour cette suite. Ainsi, la définition inductive de la suite s et la propriété attendue sont liées comme suit: $\forall i.i \in \mathbb{N} \Rightarrow s(i) = i * (i+1)/2$; le rôle de la définition inductive de s est de donner une méthode de calcul. Nous utiliserons cette méthode dans l'illustration du raffinement.

3 Bibliography

Dominique Méry and Neeraj Kumar Singh, editors. *Modelling Software-Based Systems*, volume Volume 2: Practical Applications and Extension. ISTE, ???

A. J. M. van Gasteren and G. Tel. Comments on "on the proof of a distributed algorithm": always true is not invariant. *Information Processing Letters*, 35:277–279, 1990.

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

Dines Bjørner. *Domain Science and Engineering - A Foundation for Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2021. ISBN 978-3-030-73483-1. doi:10.1007/978-3-030-73484-8.

URL https://doi.org/10.1007/978-3-030-73484-8.

Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002. ISBN 0-3211-4306-X. URL http://research.microsoft.com/users/lamport/tla/book.html.

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.

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

G. Holzmann. The spin model checker. *IEEE Trans. on software engineering*, 16(5):1512–1542, May 1997.

Edmund M. Clarke, Orna Grunberg, and Dov A. Peled. *Model Checking*. The MIT Press, 2000.

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.

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

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

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

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.

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

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.

Jean-Raymond Abrial. *The B book - Assigning Programs to Meanings*. Cambridge University Press, 1996.

ClearSy. *Atelier B*. ClearSy, Aix-en-Provence (F), 2002. http://www.atelierb.eu.

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.

Christophe Métayer and Laurent Voisin. The Event-B Mathematical Language. Technical report, The Rodin Project, March 26 2009.