Livrable 1
Développement d'algorithmes répartis

29 février 2008 :6:21pm

Projet RIMEL

ANR-06-SETI-015

Equipe MOSEL, LORIA, Université Henri Poincaré Nancy 1
ClearSy
LABRI, Université de Bordeaux & CNRS
http://rimel.loria.fr
Années 2007-2008-2009

Avertissement
Ce document a été rédigé à partir des travaux de Dominique Cansell sur le développement de l'algorithme d'énumération et des travaux de Nazim Benaissa sur le développement du Mondex. Le document a été rédigé par Dominique Méry. Les résultats exposés dans ce document ont été obtenus grâce aux commentaires et aux connaissances sur les algorithmes répartis du partenaire LABRI.

Table des matières

Chapitre 1

Introduction

Sommaire

1.1 Motivations

Ce document constitue un premier bilan des activités collaboratives entre les partenaires du projet RIMEL. Afin de bien comprendre les objectifs motivant ce livrable, nous rappelons ce qui avait été écrit dans la proposition de projet :

> Deliverable 1 *Development of distributed algorithms* which will contain the incremental proof-based development of several distributed algorithms ; the resulting developments will be the input of the study of specific proof-based design patterns for distributed modelling.

Le projet vise à systématiser des développements guidés par la relation de raffinement qui assure le maintien de propriétés entre un modèle abstrait et un modèle concret. Le constat du développement fondé sur le raffinement est qu'il exige des qualités en modélisation et en démonstration mathématique qui sont difficiles à acquérir rapidement et qui demandent une période initiatique relativement longue et nécessairement intégrant des études de cas. Ainsi, le projet vise à la fois *à développer des études de cas liées aux systèmes répartis* mais aussi *à mettre en évidence des techniques permettant de faciliter l'utilisation du développement de systèmes répartis corrects par construction*. Ce document présente un travail réalisé par le partenaire LORIA et le partenaire LABRI à propos des algorithmes répartis. L'idée initiale repose sur l'observation que les collègues du LABRI développent des algorithmes répartis selon des hypothèses constituant des niveaux d'abstraction, qui peuvent être pris en compte par des modèles Event B. Le travail de formalisation et de modélisation a eu lieu pendant des visites ou des échanges et les algorithmes développés ont été produits par le processus de raffinement de B. Chaque algorithme ou système a été développé en veillant à démontrer chaque fait ou pas de raffinement à l'aide de l'outil RODIN. D'un certain point de vue, ce travail est aussi une validation de l'outil RODIN, puisqu'il a été utilisé de manière intensive et complète sur des exemples nécessitant des preuves non-triviales. La démarche est donc assez simple à expliquer et induit une certaine méthodologie dans le développement d'algorithmes répartis ; le partenaire LABRI apporte des connaissances mathématiques sur les graphes et sur les problèmes à résoudre et le partenaire LORIA traduit ces contraintes dans le langage Event B, en veillant à faire apparaître l'abstraction pertinente associée au problème.

Dans ce document, nous allons présenter quelques développements effectués dans le cadre de ce projet ; les travaux de développement en Event B ont été réalisés par le partenaire LORIA et des études de cas ont été proposées par le partenaire LABRI. Le partenaire LABRI a fourni des études de cas expliquées dans un langage algorithmique précis. Le travail du partenaire LORIA a permis de préciser des éléments propres à ces problèmes et à proposer un dévelopement complet des algorithmes répartis. De plus, les algorithmes produits sont corrects par construction et nous avons aussi montré comment développer de nouveaux algorithmes répartis à partir de développements d'autres algorithmes existants. De plus, nous avons utilisé une étude de cas d'un système de paiement électronique par carte ; cette étude de cas appartient à une classe de systèmes que nous appelons des systèmes à base de transactions. L'étude de cas dite du Mondex conduit à des schémas de conception orientés transaction. Ces études de cas visent à échanger les connaissances des différents groupes mais aussi de fournir une base de cas pour élaborer des schémas de développement réutilisables appelés patrons.

1.2 Développement et vérification d'algorithmes répartis

Les algorithmes répartis sont des algorithmes très complexes à vérifier et à concevoir ; l'idée fondamentale dans le développement prouvé incrémental est de découvrir le fonctionnement d'un algorithme réparti donné, en opérant à partir d'une vue très générale de ce qu'il est supposé réaliser puis d'introduire progressivement sous le contrôle d'un assistant de preuve les éléments constituant les modèles qui le définissent. Cette approche s'oppose en fait à l'approche *a posteriori* consistant à analyser le système ou l'algorithme produit après un processus fondé plus souvent sur l'expériience et l'expertise de la personne réalisant la conception. L'intérêt de ces algorithmes est de faire partie du paysage des enseignants-chercheurs et donc d'être connus. Ces algorithmes répartis n'en demeurent pas moins complexes et constituent une classe bien définie de systèmes répartis. Le partenaire LABRI a une très bonne connaissance de ce domaine et maîtrise bien les propriétés mathématiques liées à ces algorithmes. Nous allons considérer les techniques utilisées pour vérifier les

algorithmes répartis. Une première approche consiste donc à vérifier que l'algorithme réparti satisfait des propriétés ; ces propriétés sont principalement des propriétés de sûreté (correction partielle, absence de blocage) et sont validées par l'utilisation d'outils comme ceux du model checking [22] ou les assistants de preuve généralistes comme COQ [28], PVS [37], STEP [33], CROCOS [35], ISABELLE [36], ... après un codage des techniques de vérification dans ces outils si nécesssaire. Par exemple, la vérification du protocole d'élection du leader IEEE 1394 a été réalisée, en utilisant l'assistant de preuve PVS [23] et le document comportait un certain nombre de propriétés appelées invariants avec des démonstrations utilisant l'assistant de preuve mais cette vérification ne donne pas réellement d'explication sur la façon dont l'algorithme fonctionne. Dans un document publié plus tard, la méthode incrémentale [10] a été appliquée en vue de rédevelopper l'algorithme d'élection du leader et d'y apporter une explication quant à son fonctionnement ; la démarche reposait sur une modélisation des graphes acycliques et sur la preuve de propriétés de ces structures avec l'outil supportant la méthode B. La consultation des ouvrages pédagogiques sur les algorithmes répartis [41, 32] témoignent d'une grande palette de modèles pour décrire les algorithmes répartis et ces ouvrages rassemblent de nombreuses versions d'algorithmes. Même si les notations sont quelque peu différentes, elles restent fondées sur les systèmes de transition discrets et la question de la production de modèles en accord avec les exigences se résout par des outils comme les model checkers ou par la preuve notamment dans le cas de TLA$^+$ [31] un format de preuve est proposé pour structurer celle-ci mais la découverte de l'invariant participe d'un processus difficile et complexe.

1.3 Sommaire du rapport

Dans la suite de ce rapport, nous allons présenter de manière plus détaillée la méthode Event B mise en œuvre dans le cadre de la plate-forme RODIN [39]. Cette méthode sera ensuite utilisée pour développer quelques algorithmes répartis se rapportant à des problèmes bien spécifiques. Les algorithmes répartis ont été choisis comme études de cas car ils constituaient des objets que nous utilisions dans nos enseignements : le partenaire LABRI enseigne au niveau du master et d'une école d'ingénieur. L'équipe du LABRI a fourni des problèmes considérés comme très difficiles et la méthode de développement incrémentale prouvée a été appliquée en vue de dégager ultérieurement des techniques systématiques de développement comparables aux *design patterns* [25].

1.3.1 Les algorithmes d'élection du leader de la famille IEEE 1394

Nous avons développé le protocole d'élection du leader utilisé dans la norme IEEE 1394 et ce développement nous a permis d'illustrer comment développer un tel algorithme. Cet algorithme est fondé sur une stratégie de soumission : un site demande à son voisin d'être adopté comme son enfant. Au cours du développement, une erreur a conduit à mettre en évidence un autre algorithme puisque le développement peut être partiellement réutilisé et modifié. Ce nouvel algorithme résolvait la contention en considérant le site de numéro le plus important ; la contention apparaît éventuellement lorsque deux sites sont en compétition pour élire un leader parmi eux deux et la résolution dans le cas de l'algorithme classique est fondé sur un choix probabliliste. Enfin, une observation du développement a conduit à montrer que la confirmation pouvait être supprimée dans les deux algorithmes. Nous avons ainsi obtenu quatre algorithmes à partir du même développement transformé par des choix différents. Nous n'avons pas défini ce qu'est un développement mais cet aspect sera défini dans la section suivante et on peut simplement énoncer qu'un développement est une suite de modèles liés par une relation de raffinement et assurant des propriétés de sûreté et des invariants.

Le développement de l'algorithme du IEEE 1394 comprend les étapes suivantes :

- Un modèle initial exprimant la spécification de l'élection par un unique événement opérant sur une structure qui est un graphe acyclique non-orienté et qui garantit qu'il est toujours possible d'élire un leader.
- Un second modèle exprime que l'élection a lieu en accord avec une induction qui exprime qu'une forêt de nœuds converge vers un unique arbre recouvrant le graphe initial et dont la racine est le leader. Ce modèle simule le modèle précédent selon la relation de raffinement.

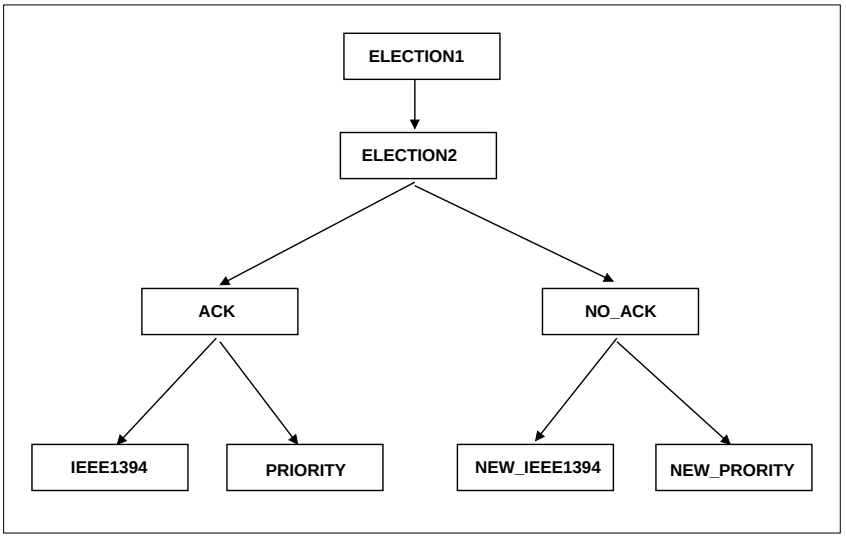


FIG. 1.1 – Développement de l'élection du leader

- Une troisième étape introduit des communications par messages entre les nœuds ; des messages de requêtes, des accusés de réception et de confirmation sont utilisés. A ce niveau, les preuves nous apportent des éléments pour simplifier l'algorithme en utilisant ou non des confirmations.
- Une quatrième étape résout la contention en utilisant la terminaison du processus sous contrainte probabiliste ou bien le numéro des sites, quand cela est possible.

Indications des obligations de preuve réalisées durant ce développement :

Modèles	Nombre de PO	PO Interactives
ELECTION_1	2	1
ELECTION_2	10	2
ELECTION_IEEE	41	0
ELECTION_PRIORITY	31	2
ELECTION_NEW_IEEE	33	0
ELECTION_NEW_PRIORITY	13	0
TOTAL	120	5

1.3.2 Algorithme d'énumération

L'algorithme d'énumération des nœuds d'un graphe de Mazurkiewicz [34] est un algorithme distribué qui attribue, à chaque nœud d'un graphe non orienté, un numéro entre 1 et m où m est le nombre de nœuds du graphe. Un nœud se synchronise avec l'ensemble de ses voisins et un nœud peut modifier son état, ainsi que celui de ses voisins. Il est donc distribué sur les boules du graphe. Une boule de centre x est composée du nœud x et de l'ensemble de ses voisins. L'algorithme construit une numérotation qui converge vers un revêtement du graphe. Cet algorithme termine sur des graphes non orientés non-réflexifs connexes et mininaux pour les revêtements. Un revêtement est un homomorphisme surjectif des nœuds du graphe vers ceux d'un autre qui conserve la structure des arêtes (donc des boules). Un revêtement est localement bijectif (sur une boule). Pour un graphe minimal, au sens des revêtements, les seuls revêtements surjectifs sont des injections (donc des bijections). Pour montrer la correction et la terminaison de l'algorithme d'énumération de Mazurkiewicz, il faut montrer de nombreuses propriétés qui sont trop compliquées à prouver en une seule fois avec des assistants de preuve. Le but de cette étude est d'exhiber une suite de modèles de plus en plus concrets, tels que chaque modèle capture une des propriétés énoncées.

L'algorithme d'énumération de Mazurkiewicz est défini à l'aide de règles qui expliquent comment un nœud du graphe change son numéro courant (Règle de renommage) et comment l'information

sur les nœuds se propage (Règle de diffusion). La variable N est la numérotation courante. La variable n est une fonction des nœuds vers l'ensemble des numéros (différents de 0) de ses voisins. La variable M (pour mémoire ou boite à lettres) associe à chaque nœud un ensemble de couples (k, Nk) où k est un numéro attribué et Nk un ensemble de numéros différents de 0. Lors de l'exécution de l'algorithme, un nœud x avait k comme numéro et Vk comme $N(x)$.

L'algorithme d'énumération de Mazurkiewicz est défini à l'aide de deux règles qui expliquent comment un nœud et ses voisins changent leurs variables. On supose que les autres nœuds ne changent pas leurs variables durant l'application de ces règles.

- Règle de renommage : Lorsqu'un nœud x voit que tous ses voisins ont la même mémoire que lui et qu'il a son numéro à 0 ou alors il y a dans sa mémoire un couple $(n(x), V)$ tel que V est plus grand que $N(x)$ alors il peut changer de numéro.

  Precondition

  $\forall v \in B(x), M(v) = M(x)$

  $(n(x) = 0) \ \vee \ (n(x) > 0 \wedge \exists (n(x), V) \in M(x) \mid (N(x) \prec V)$

  Postcondition

  $n'(x) = 1 + \max\{n \mid (n, V) \in M(x)\}$

  $\forall y \in X - \{x\}, \ n'(y) = n(y)$

  $\forall v \in B(x) - \{x\}, \ N'(v) = (N(v) - \{n(x)\}) \cup \{n'(x)\}$

  $\forall v \in \{x\} \cup (X - B(x)), \ N'(v) = N(v)$

  $\forall v \in B(x), \ M'(v) = M(v) \cup \{(n'(w), N'(w)) \mid w \in B(x)\}$

  $\forall v \in X - B(x), \ M'(v) = M(v)$

- Règle de diffusion : Si un nœud x n'a pas la même mémoire qu'un de ses voisins, alors l'ensemble de nœuds de $B(x)$ (le nœud x et ses voisins) se synchronisent en prenant l'union de toutes les mémoires de x et de ses voisins.

  Precondition

  $\exists v \in B(x)$ such that $M(v) \neq M(x)$

  Postcondition

  $\forall v \in B(x), \ M'(v) = \bigcup_{w \in B(x)} M(w)$

  $\forall v \in X - B(x), \ M'(v) = M(v)$

Lorsqu'aucune des rêgles n'est applicable, alors tous les nœuds ont la même mémoire et la numérotation est un revêtement. Lors d'une réunion RIMEL (avril 2007), Yves Métivier a expliqué cet algorithme au tableau. Il a exposé les deux règles et, pour nous convaincre de la correction de l'algorithme, il a donné des arguments et des propriétés de cet algorithme. Pour ce développement, nous avons voulu capturer ces propriétés fondamentales de l'algorithme pour ne pas faire toutes les preuves sur le modèle final.

- Le modèle "one shot" sur les nœuds seulement (global)
- Introduction de l'essence de l'algo. Comment les nœuds modifient la numérotation courante (encore global) avec une abstraction de la règle de renomage. Ce modèle définit le codomaine et la surjectivité du futur homomorphisme.
- Introduction des arêtes ; un nœud et ses voisins auront des numéros différents (voisins des voisins + global). Sur ce modèle nous avons une injection locale du futur homomorphisme.
- Introduction de la variable N qui donne les numéros ($\neq 0$) des voisins (raffinement de donnée + superposition).
- Introduction de la variable M de l'algorithme de Mazurkiewicz (local sur les boules) et de la règle de diffusion.
- Introduction de la connexité et de la minimalité du graphe pour la correction.

L'effort de preuve correspond à une activité de deux semaines à temps plein (71 POs + 45 interactives avec Click'n'Prove).

1.3.3 Systèmes répartis à transactions

Dans cet exemple, il s'agit de développer des systèmes distribués communiquant par des canaux non fiables, en particulier des protocoles de communication utilisés dans les porte-monnaie électroniques. L'exemple utilisé est celui connu sous le nom MONDEX et a été proposé dans le cadre

du Grand Challenge [26, 29]. Ce travail met en avant une explication des mécanismes sous-jacents qui permettent à ce système de satisfaire les exigences initiales résumées sous la forme suivante : il n'y a pas de perte d'argent ni de création d'argent lors des transferts. Le développement permet de préserver cette propriété de sûreté attendue du système final et identifie des relations de communication spécifiques pour assurer la mise en œuvre du protocole. Ainsi, un premier modèle énonce la spécification globale de sûreté ; un second modèle raffiné décrit le comportement détaillé des différents protagonistes ; un troisième modèle localise les données ; un quatrième modèle élimine des variables utiles pour faciliter les preuves. Cette étude nous permet de vérifier la robustesse du protocole à la perte de messages. Mondex [40, 1] est un système de paiement électronique qui fut introduit pour la première fois par la *National Westminster Bank* en 1990 avant d'être repris par *MasterCard*. Le système est basé sur les cartes à puce servant de porte-monnaie électroniques contenant chacun un solde d'argent. Une des spécificités du système est que les transactions sont *hors-ligne*, ceci signifie que les transactions se passent entre deux porte-monnaie sans l'intervention d'une autorité centralisée. Au cours d'une transaction, chaque porte-monnaie doit donc être capable de garantir lui-même les mesures de sécurité inhérente à ce genre de système, sans l'intervention d'une quelconque autorité de régulation.

Chaque porte-monnaie est initialement crédité d'un solde. La transaction consiste en un transfert d'argent entre un porte-monnaie source et un porte-monnaie destination. La figure 4.1 montre le schéma global d'une transaction.

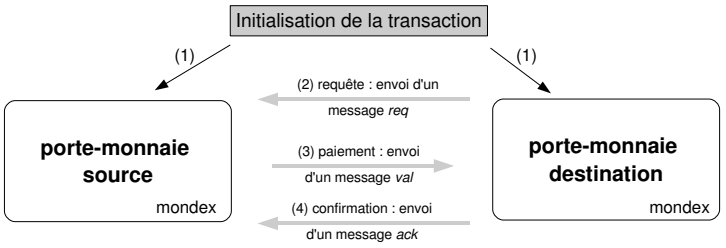


FIG. 1.2 – Les différentes étapes d'une transaction

Le schéma de la figure 4.1 montre le cas d'une transaction où tout se passe bien. Mais dans la réalité plusieurs événements peuvent perturber le déroulement d'une transaction. Il peut s'agir par exemple d'une défaillance du canal de communication ou encore d'un retrait prématuré de la carte à puce du lecteur de carte, avant que la transaction n'arrive à son terme. Pour prévenir ce genre d'erreurs, un certain nombre de dispositifs ont été prévus. En plus de son identifiant et de son solde, un porte-monnaie contient un journal d'erreurs dans lequel les transactions qui ont échoué sont transcrites. Il contient également le numéro de séquence de la prochaine transaction à effectuer. Un porte-monnaie a aussi un état qui varie au fur et à mesure que la transaction avance. Les différents états possibles sont *eaFrom, eaTo, epr, epv, epa* (voir figure 4.2).

Lorsqu'un porte-monnaie source abandonne la transaction prématurément alors qu'il a déjà débité son solde (*état epa*), il inscrit la transaction en cours sur son propre journal d'erreurs. De même pour un porte-monnaie cible qui a envoyé un message de type *Req* et qui n'a pas encore crédité son solde (*état epv*). Grâce à ce procédé, le croisement des informations contenues dans les journaux d'erreurs des différents porte-monnaie permettra de récupérer l'argent apparemment perdu localement.

Nous avons dans cette étude modélisé et prouvé le protocole Mondex dans la perspective de développer des patrons pour les protocoles de communications à travers des canaux non fiables. Cette étude de cas nous a permis de voir qu'elles étaient les difficultés liées à ce genre de système et la manière avec laquelle les gérer :

– Les propriétés de sûreté du système qui doivent être préservées sont exprimées dans le modèle

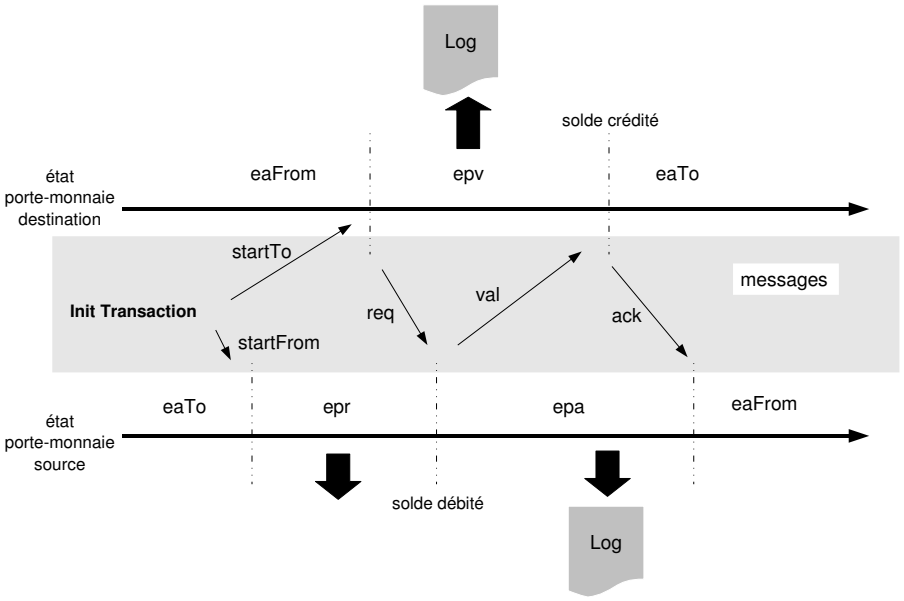


FIG. 1.3 – Le protocole Mondex

abstrait dans lequel la transaction est atomique.
- Modéliser d'abord des transactions abstraites et raffiner ensuite le système pour que les transactions ne soient plus atomiques jusqu'à arriver au protocole sous sa forme concrète.
- Les exceptions (transaction interrompues prématurément) sont gérées avec une variable abstraite *abAuthLost* qui est ensuite concrétisée dans les raffinements suivants (avec les journaux d'erreurs dans le cas Mondex).
- La nécessité de distinguer les messages de même type afin de prévenir les risques qu'un même message soit utilisé plusieurs fois. Dans le cas Mondex un numéro séquentiel associé à la transaction courante figurant sur les messages et les porte-monnaie est utlisé.
- Lorsqu'il y a perte des messages, la transaction doit pouvoir être interrompue par la partie qui attend ce message sans que cela ne remette en cause les propriétés de sûreté du système. Dans le cas Mondex, ceci est garanti par le fait qu'à chaque fois qu'un porte-monnaie interrompt une transaction il la rajoute dans son journal d'erreurs.

L'ensemble du processus de modélisation a produit 557 obligations de preuves réparties sur les différents raffinements comme suit :

Modèle	Nombre Total de PO	Automatique	Interactives
Modèle abstrait	27	27 (100%)	0 (0%)
Premier raffinement	45	45 (100%)	0 (0%)
Deuxième raffinement	17	17 (100%)	0 (0%)
Troisième raffinement	263	191 (72%)	72 (28%)
Quatrième raffinement	114	114 (100%)	0 (0%)
5^e,6^e et 7^e raffinement	104	104 (100%)	0 (0%)
Total	570	498 (87%)	72 (13%)

1.4 Présentation des chapitres

Les trois familles de développement ne constituent pas les uniques développements réalisés mais apportent des éléments pour fertiliser la phase de mise en évidence des patrons. Nous présentons brièvement la suite du document :

- Le chapitre 2 présente la méthode Evenet B.

- Le chapitre 3 décrit le développement correspondnat à l'élection du leader.
- Le chapitre 4 présente le développement de l'algorithme d'énumération.
- Le chapitre 5 présente le développement d'un système réparti fondé sur des transactions.
- Le chapitre 6 donne des conclusions générales.

Nous avons réalisé d'autres développements avec le LABRI et ils seront finalisés ultérieurement :

- Développement incrémental et preuve de l'algorithme de Szymanski, Shi et Prywes
- Développement incrémental et preuve de l'algorithme d'élection du leader dans un 2-arbre

Chapitre 2

La méthode incrémentale Event B

Sommaire

2.1 Modélisation de systèmes

La modélisation des systèmes à événements discrets peut être réalisée à l'aide de formalismes variés comme les *action systems* [12], le langage SDL [21], les réseaux de Petri [38], le langage TLA$^+$ [30, 31] et le langage UML [15]. Nous utilisons le langage B événementiel appelé aussi Event B, qui intègre une notation ensembliste pour les données, un langage d'événements pour exprimer les changements d'états et un langage de modèles, qui peuvent être liés entre eux par une relation de raffinement. Nous parlerons aussi de systèmes réactifs qui réagissent à des stimuli de l'environnement et notre démarche s'appuie sur le développement progressif et prouvé de modèles s'approchant de plus en plus d'une version concrète d'un modèle du système à modéliser.

Ainsi, plusieurs questions sont posées. D'une part, l'objet construit dans une notation donnée est un modèle qu'il faut vérifier et valider ; il faut s'assurer de l'adéquation de ce modèle aux exigences du cahier des charges et, en quelque sorte, conserver une traçabilité permettant de lier les éléments formels et les éléments plus appréhendables par le client. L'approche (ou la méthode) « B événementielle » se situe dans le cadre d'une réelle exploitation des bilans de preuve formelle dans le cadre du développement de modèles formels précis d'un système donné ; cette démarche permet de construire graduellement un modèle concret du système pas-à-pas par la preuve objective des raffinements. Cette objectivité est obtenue par l'emploi de l'assistant de preuve [4], dont le rôle est de produire automatiquement une partie des preuves et de vérifier les autres au cours de la session interactive de construction de la preuve.

Cependant, plutôt que de construire un modèle formel complexe, mal structuré et difficile à comprendre et à valider, nous pensons qu'une démarche progressive fondée sur le raffinement de modèles formels avec l'assistance d'un prouveur est une démarche originale et effective et surtout qu'elle permet d'intégrer des éléments théoriques de manière naturelle. Les études de cas comme Mondex sont clairement fondamentales pour illustrer notre point de vue. Les modèles abstraits sont constitués d'une liste d'actions activées selon les gardes activables à chaque état.

La méthode « B événementielle » [3, 9, 19] est une extension de la méthode B [9] dans son utilisation et dans sa philosophie. Elle est de plus en plus utilisée pour des études de cas allant des systèmes répartis [5, 6] aux protocoles cryptographiques et même aux circuits électroniques [2]. Elle permet une étude « système » complète d'un système où cohabitent et communiquent des parties matérielles et logicielles.

Un modèle abstrait de système est défini par un nom, une liste de variables, un invariant et un ensemble d'événements dont un détermine l'initialisation du système. Un événement est défini à l'aide d'une garde et d'une substitution. Cette notion d'événement est proche des actions de Back [12] ou des commandes gardées de Dijkstra [24]. Un événement ne peut se déclencher que si sa garde est vraie ; quand un événement est observé, sa garde était vraie juste avant son observation. Le raffinement « B événementiel » est différent de celui du B classique car on peut ajouter de nouveaux événements qui, en général, ne modifient pas les variables abstraites (skip joue le rôle du bégaiement).

2.2 Modèles de systèmes avec Event B

Le modèle d'un système est simplement défini par une structure appelée machine ayant les clauses suivantes :

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$A(x)$
INITIALISATION
$Init(x, s, c)$
EVENTS
$e_1, e_2, \ldots e_n$
END

CONTEXT
c
SETS
s
CONSTANTS
k
AXIOMS
$P(s, k)$
THEOREMS
$A(s, k)$
END

Le CONTEXT c d'une machine m définit la théorie logico-mathématique du problème traité par la machine m. Une notation précise l'ensemble des variables du système, les propriétés invariantes de ces variables, la liste des événements pris en compte par le modèle ainsi que les théories englobantes au modèle considéré. Nous ne précisons pas les aspects liées aux modalités temporelles proposées par J.-R. Abrial et L. Mussat [8] ; nous évoquerons ces aspects dans le cadre des diagrammes de prédicats.

On retiendra trois uniques formes possibles pour les événements et cela suffira pour modéliser les systèmes au sens général. Les substitutions généralisées ont différentes formes possibles et nous évoquons chaque cas possible. 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 $BA(x, x')$.

Event : e	Prédicat Before-After $BA(e)(x, x')$
BEGIN $x : \lvert P(x, x')$ END	$P(x, x')$
WHEN $G(x)$ THEN $x : \lvert P(x, x')$ END	$G(x) \ \wedge \ P(x, x')$
ANY t WHERE $G(t, x)$ THEN $x : \lvert P(x, x', t)$ END	$\exists t \cdot (G(t, x) \ \wedge \ P(x, x', t))$

Un événement est caractérisé par sa garde qui est déterminée au moment de la modélisation et il ne peut être déclenché que si cette garde est vraie. D'une certain mesure, cela signifie qu'une condition apparaît et que la partie transformation associée est exécutée.

Event : e	Guard : grd(E)
BEGIN S END	$TRUE$
WHEN $G(x)$ THEN T END	$G(x)$
ANY t WHERE $G(t,x)$ THEN T END	$\exists t \cdot G(t,x)$

La garde d'un événement définit la condition d'activation de cette événement et il est important de veiller à ce que le système ne soit pas bloqué définitivement. Pour vérifier cela, on doit montrer qu'au moins un événement est activable à tout instant du système :

$$I(x) \Rightarrow (\mathsf{grd}(e_1) \vee \ldots \mathsf{grd}(e_n)) \ (\mathsf{DEAD}) \tag{2.1}$$

L'invariant d'un modèle est une propriété invariante pour tous les événements du système modélisé y compris l'événement initial. Si $I(x)$ est l'invariant du modèle et si e est un événement du modèle, alors la condition de préservation de cet invariant par cet événement est le suivant :

$$I(x) \ \wedge \ BA(e)(x,x') \ \Rightarrow \ I(x') \quad (INV) \tag{2.2}$$

La condition sur les conditions initiales est la suivante :

$$Init(x,s,c) \ \Rightarrow \ I(x) \quad (INIT) \tag{2.3}$$

Quand un événement définit le prédicat before-after $P(x,x')$, la faisabilité de cet événement signifie que sous l'hypothèse définie par l'invariant $I(x)$ et par la garde $\mathsf{grd}\,(E)$, de l'événement, il existe toujours x' tel que $P(x,x')$; 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 :

$$I(x) \ \wedge \ \mathsf{grd}\,(e) \ \Rightarrow \ \exists x' \cdot P(x,x') \qquad (FIS) \tag{2.4}$$

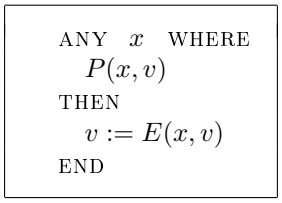
Les propriétés de sûreté sont déduites par la preuve que l'invariant du système implique la propriétét de sûreté et nous ajoutons en plus, le contexte de cette preuve. Le contexte de cette preuve est donné par les propriétés $P(s,c)$ des ensembles s et des constantes c définies dans le modèle :

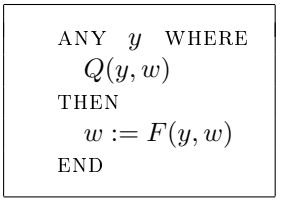
$$P(s,c) \ \wedge \ I(x) \ \Rightarrow \ A(x) \qquad (THM) \tag{2.5}$$

Les événements ont des formes particulières et sont souvent des affectations gardées par une condition. Les variables doivent respecter l'invariant et la vérification de cet aspect est réalisée par la preuve de conditions appelées obligations de preuve. Par exemple, nous donnons l'expression formelle de cette obligation de preuve pour un événement de la forme donnée.

ANY x WHERE
$P(x,v)$
THEN
$v := E(x,v)$
END

L'événement préserve l'invariant I, quand l'obligation de preuve est vraie : $I(v) \wedge P(x,v) \Rightarrow I(E(x,v))$. v est la valeur de la variable v, quand l'événement est déclenché ; $E(x,v)$ est la valeur de la variable v juste après l'activation de l'événement. Chaque événement du modèle abstrait (le modèle à raffiner) est raffiné au sens des événements en un événement associé du modèle concret (le modèle raffiné). Un événement concret raffine son abstraction, quand la garde de l'événement concret est plus forte que la garde de l'événement abstrait et quand l'invariant de collage est préservé par l'action conjointe des deux événements. Nous décrivons plus précisèment cette notion ci-après :





Si la formule suivante appelée *obligation de preuve* est prouvée dans le contexte du système en développement :

$I(v) \wedge J(v,w) \wedge Q(y,w) \Rightarrow \exists x.(P(x,v) \wedge J(E(x,v),F(y,w)))$

où I est l'invariant abstrait et J est l'invariant de collage.

Cela signifie que pour tout choix possible des variables locales de l'événement concret (y), il y a un choix pour les variables locales de l'événement abstrait (x) qui rend l'invariant de collage vrai. La garde abstraite est renforcée par la garde concrète. Les obligations de preuve sont engendrées par *Click'n'Prove* [4] automatiquement et prouvées par le prouveur intégré avec d'éventuelles interactions. Depuis quelques mois, le partenaire LORIA utilise l'outil RODIN qui est l'outil dédié à la méthode Event B et nous avons donc réalisé les preuves dans ce nouvel outil. Enfin, nous donnons quelques notations utiles pour comprendre les formules dans les chapitres suivants.

Nom	Syntaxe	Définition
Relation Binaire	$s \leftrightarrow t$	$\mathcal{P}(s \times t)$
Composition de relations	$r_1; r_2$	$\{x,y \mid x \in a \quad \wedge \quad y \in b \quad \wedge \quad \exists z.(z \in c \quad \wedge \quad x,z \in r_1 \quad \wedge \quad z,y \in r_2)\}$
Relation inverse	r^{-1}	$\{x,y \mid x \in \mathcal{P}(a) \quad \wedge \quad y \in \mathcal{P}(b) \quad \wedge \quad y,x \in r\}$
Domaine	$\mathsf{dom}(r)$	$\{a \mid a \in s \quad \wedge \quad \exists b.(b \in t \quad \wedge \quad a \mapsto b \in r)\}$
Image	$\mathsf{ran}(r)$	$\mathsf{dom}(r^{-1})$
Identité	$\mathsf{id}(s)$	$\{x,y \mid x \in s \quad \wedge \quad y \in s \quad \wedge \quad x = y\}$
Restriction	$s \lhd r$	$\mathsf{id}(s); r$
Co-restriction	$r \rhd s$	$r; \mathsf{id}(s)$
Anti-restriction	$s \lhd\!\!\!- r$	$(\mathsf{dom}(r) - s) \lhd r$
Anti-co-restriction	$r \rhd\!\!\!- s$	$r \rhd (\mathsf{ran}(r) - s)$
Image	$r[w]$	$\mathsf{ran}(w \lhd r)$
Surcharge	$q \lhd\!\!\!+ r$	$(\mathsf{dom}(r) \lhd\!\!\!- q) \cup r$
Fonction Partielle	$s \nrightarrow t$	$\{r \mid r \in s \leftrightarrow t \quad \wedge \quad (r^{-1}; r) \subseteq \mathsf{id}(t)\}$

Ce tableau est sans doute partiel mais la bibliographie apporte quelques pointeurs pour identifier certains points syntaxiques.

2.3 Bibliographie

La méthode Event B a été présentée dans un chapitre écrit par D. Cansell et D. Méry [19] ; ce chapitre comporte des exemples développés complètement. Il y a bien sûr les documents que l'on peut obtenir à partir du site RODIN (www.event-b.org) [39] et les publications de J.-R. Abrial disponibles sur le site RODIN ou via DBLP.

Chapitre 3

Les algorithmes d'élection du leader de la famille IEEE 1394

Sommaire

3.1 Introduction

This chapter shows how we re-use a previous incremental proof-based development of a distributed algorithm [7] to discover two other simpler correct-by-construction distributed algorithms : the resulting algorithm does not exist. Abstract models of the previous development express a context that leads us to the emergence of a simple idea using refinement and a tool which yields a new correct solution by construction. Our methodology uses the event B method and the Click'n'Prove tool [4], which together become a true laboratory for replaying developments.

The case study is the IEEE 1394 leader election protocol [7] in an acyclic graph. We start the new modelling from the two first models which explain the goal of the algorithm : election of a leader by submission. The real algorithm uses several messages between two nodes of the graph : a *submission* message is sent by a node x to a node y ; when y has received the submission message, an *acknowledgement* message is sent by y to x, which sends a *confirmation* message to y. Our new algorithms use only the *submission* messages.

The IEEE 1394 case study enabled us to state a methodology of development for distributed algorithms :

- The starting phase is an abstract model - called the one-shot model - and it expresses the specification of the problem to solve. It has only one event, which is modelling the execution of the whole computation process ; the event defines a pre/post specification in the event B modelling language. The event is a computation relation and does not express any execution model.
- A second phase of modelling gradually introduces the essence of the algorithm, namely the submission ; the first refinement produces a refinement model based on an induction principle over a tree structure. This refinement model includes two events ; one event corresponds to a refined version of the election, the second event models the induction step helping the tree-like structure to converge to a unique tree-like structure with a unique root node. The induction step is based on a submission principle - a node x sends a submission message to a node y and accepts the principle of submission (slave/master). We note that this is the most difficult proof to carry out in an interactive way and we claim that this proof would have been very difficult in a more concrete model.
- A third phase adds the management of messages between nodes ; a node may send message to its neighbours and may receive messages from its neighbours. The network is still abstract and the information is still global. New events are defined to model the submission messages, the acknowledgement messages and the confirmation messages.
- A fourth phase localises global data and global variables ; the knowledge per node is effectively local. The decision of localisation is taken late, since the technical proofs are easier in the lower abstract levels. If the localisation is not possible, it may be possible that the algorithmic principles introduced in the previous refinement steps do not allow a distribution of data and actions. A solution would be to improve the previous refinement models.

The first phase is a very crucial step ; the underlying mathematical theory should be precisely defined. In the IEEE 1394 case study, the definitions of trees, graphs, induction over trees, etc should have been stated in the set-theoretical language ; the problem has been stated in the mathematical theory and theorems, such as the existence of a spanning tree for any acyclic graph. Many properties for data structures are generally proved in algorithmic books and should be integrated in a safe way ; the proof tool helps us to discharge the proofs of these properties. The effort involved in these proofs, is important ; it will be done only one time and will be integrated into a library of proved theories.

When the algorithm exists, the last model should be close to the algorithm ; it helps us in our refinement steps ; it allows us to understand how the algorithm is working and to extract at an abstract view of the algorithm. The main advantage of an incremental development is not to construct the final algorithm, but the incremental development collects much information about the algorithm that solves the problem and about the mathematical properties related to that problem.

It is possible to modify proved models or to start a new development from any model ; for instance, the IEEE 1394 protocol leader election solves contention by a mechanism based on the choice of a timeout delay. In our first development, we decided to choose the higher node in the two competing nodes. This choice of refinement led to a new algorithm which is possible and

correct with respect to the requirements. Another consideration is the complexity of the proof ; is it possible to write a simpler proof of the property that is difficult to prove? How can we simply prove that the last step of the algorithm determines the leader among two possible candidates rather that allow two pairs of nodes in the network compete for the leadership? Can we find a more efficient algorithm? For instance, the efficiency of a distributed algorithm is based on the number of messages which are required for a given task. The analysis of the proved models shows us that a new invariant simplifies the interactive proofs. We present the first two models of the development [7].

3.2 Analysis of the leader election problem

3.2.1 Formalizing the leader election problem in Event B

This first step is crucial in the event B development. It aims to state the leader election problem in a event B model, which expresses the properties of data structures and the existence of a solution. The data structures used are undirected acyclic graph. The solution exists if there is a directed spanning tree for each node considered as a root. The initial model is called ELECTION_1 and simply gathers definitions :

DEFINITIONS

$$\mathsf{tree}\,(r,N,t) = \left(\begin{array}{l} r \in N \\ t \in N - \{r\} \rightarrow N \\ \forall q \cdot \left(\begin{array}{l} q \subseteq N \ \wedge \\ r \in q \ \wedge \\ t^{-1}\,[q] \subseteq q \\ \Longrightarrow \\ N \subseteq q \end{array}\right) \end{array}\right)$$

$$\mathsf{spanning}\,(r,t,g,N) = \mathsf{tree}\,(r,N,t) \wedge t \subseteq g$$

– N stands for the set of nodes in the graph.
– t is a relation conforms to g ; it defines a tree over g.
– r is the root of the tree defined by t.
and properties as follows :
N is the (finite) set of nodes and g is a symmetric and irreflexive graph such that each node x is the root of a unique spanning tree $f(x)$ of the graph.

PROPERTIES
$g \subseteq N \times N$
$g = g^{-1}$
$\mathsf{id}\,(N) \ \cap \ g \ = \ \varnothing$
$f \in N \rightarrow (N \nrightarrow N)$
$\forall (r,t) \cdot (r \in N \ \wedge \ t \in N \nrightarrow N \implies t = f(r) \Leftrightarrow \mathsf{spanning}\,(r,t,g,N))$

f states that a unique spanning tree for g can be assigned to each node; f is a total function over N defining the tree assigned to a given node. The first abstract model only has the event elect_1 which expresses the election of l and builds the spanning tree s in one shot :

$\mathsf{elect}_1 \ \widehat{=}$ ANY x WHERE $x \in N$ THEN $l, s := x, f(x)$ END

3.2.2 First algorithmic model of the leader election problem

The second model introduces the submission algorithm. A node x who has been asked to be the leader by each neighbour, is in fact the leader of the global network. The new refined event elect_2

has a guard checking the condition of the election. The event $\mathsf{progress}_2$ can be triggered, when each neighbour of a given node x has asked him to be the leader, but if one neighbour y of x has not asked ; the node x becomes the child of y. The progress is ensured by the convergence of the current forest towards a spanning tree, which exists by properties of the mathematical structure.

The leader election problem is characterized by definitions of acyclic undirected graphs and the event elect_1 which expresses the proved [7] existence of a directed spanning tree for each node. The next goal is to refine the current model ELECTION_1 to obtain a model ELECTION_2 which is more algorithmic than ELECTION_1 .

This refinement introduces the essence of the algorithm : the process of election is based on submission. The submission is atomic : two nodes x and y, where x has accepted the submission of all its neighbours except y, gives its submission to y and y accepts the submission. The model ELECTION_2 introduces a new event which simulates the progressive computation of the directed spanning tree ; the process models an inductive step and provides a simple expression of the convergence of a forest toward a tree. The model ELECTION_2 has an event $\mathsf{progress}_2$ which helps the process to reach the tree. The guard is true for two nodes x and y, when x is the root of the forest and y is a neighbour of x under the relation defining the underlying graph structure. Its effect is to add the link which makes the node x the child of y in t. When x has accepted the submission of all its neighbours, the event elect_2 can be triggered.

$\mathsf{progress}_2 \ \widehat{=}$
ANY x, y WHERE
$x \mapsto y \in g \ \wedge$
$x \notin \mathsf{dom}\,(t) \ \wedge$
$y \notin \mathsf{dom}\,(t) \ \wedge$
$g[\{x\}] = t^{-1}[\{x\}] \cup \{y\}$
THEN
$t := t \cup \{x \mapsto y\}$
END

$\mathsf{elect}_2 \ \widehat{=}$
ANY x WHERE
$x \in N \ \wedge$
$g[\{x\}] = t^{-1}[\{x\}]$
THEN
$l, s := x, t$
END

The difficulty is in proving the refinement of the event elect_1 by the event elect_2. The problem is to prove that, when each neighbour of a node x is its child, then the resulting structure t is a spanning tree for g rooted by x : $f(x)$ returns the tree t. The invariant expresses the fact that the set of spanning trees converge towards a spanning tree.

$$t \in N \nrightarrow N$$
$$t \cap t^{-1} = \varnothing$$
$$\mathsf{dom}\,(t) \lhd (t \cup t^{-1}) \ = \ \mathsf{dom}\,(t) \lhd g$$

The invariant states that t is a partial function over N (t is intended to model the functional relation called *be the child of*) ; t is asymmetric. The third sentence expresses the progression of the process and the fact that only one edge is chosen for the tree t (either x to y or y to x) ; moreover, t is g up-to the reverse. The property $x \notin \mathsf{dom}(t)$ is required and is proven automatically by the automatic prover (predicate prover). The next difficult step is the proof of $N \subseteq \{n | n \in N - \{x\} \ \wedge \ n \mapsto f(x)(n) \in t\} \cup \{x\}$ which is discharged using the induction principle over the spanning tree rooted by x ; $f(x)$ and t are in fact equal on $N - \{x\}$ and x is not in their domain ; hence, $f(x)$ and t are equal on N.

The proof summary of the first model is 12 proof obligations generated by the tool and 5 proved by interaction.

The next proof is to show that there are at most two nodes x and y such that x and y are neighbours and are not yet in the tree t under construction : if y (resp. x) requests to x to be its child ($y \mapsto x \in g$ and $x \notin \mathsf{dom}(t)$), then $y \mapsto x$ is appended to the set t, which is $f(x)$ and symmetric for x and y. One concludes that $t \subseteq f(x)$ and $t \subseteq f(y)$. Both properties $x \notin \mathsf{dom}(t)$ and $y \notin \mathsf{dom}(t)$ are discharged. The idea is to propose an invariant expressing this fact :

$$\forall x \cdot (x \in N - \mathsf{dom}\,(t) \implies t \subseteq f(x))$$

In fact, the additional invariant tells us simply that a spanning tree always exists and that each node not in the domain of t is a candidate for the leadership and that node may become the leader. The leadership can be constrained by a variable attached to each node. A boolean variable, called *root*, forbids (freezes) the submission to the current node. Only one node should have a local variable *root* set to *true*, if one wants to avoid the deadlock. Consequently, if a node *wants* to be the leader, he/she can decide to wait requests from his/her neighbours ; however, if two nodes are using the same strategy, the global system will deadlock. The refinement proof is based on the property $\mathsf{dom}(t) = N - \{x\}$, whose proof is divided into the following steps :

1. $x \notin \mathsf{dom}(t)$: automatically discharged by the predicate prover.
2. $N \subseteq \mathsf{dom}(t) \cup \{x\}$: the proof is based on the induction principle over the spanning tree rooted by x $f(x)$; the universally quantified variable is instantiated by $\mathsf{dom}(t) \cup \{x\}$. After this instantiation, the predicate prover automatically discharges the waiting goal.

This proof summary of the first refinement is 10 proof obligations generated by the tool, two of which require interaction. The proof of the two last nodes is based on :

$$\forall (x, y) \cdot \left(\begin{array}{l} x \mapsto y \in g \wedge \\ x \notin \mathsf{dom}\,(t) \wedge \\ y \notin \mathsf{dom}\,(t) \wedge \\ g[\{x\}] = t^{-1}[\{x\}] \cup \{y\} \wedge \\ g[\{y\}] = t^{-1}[\{y\}] \cup \{x\} \\ \Longrightarrow \\ \mathsf{dom}\,(t) = N - \{x, y\} \end{array} \right)$$

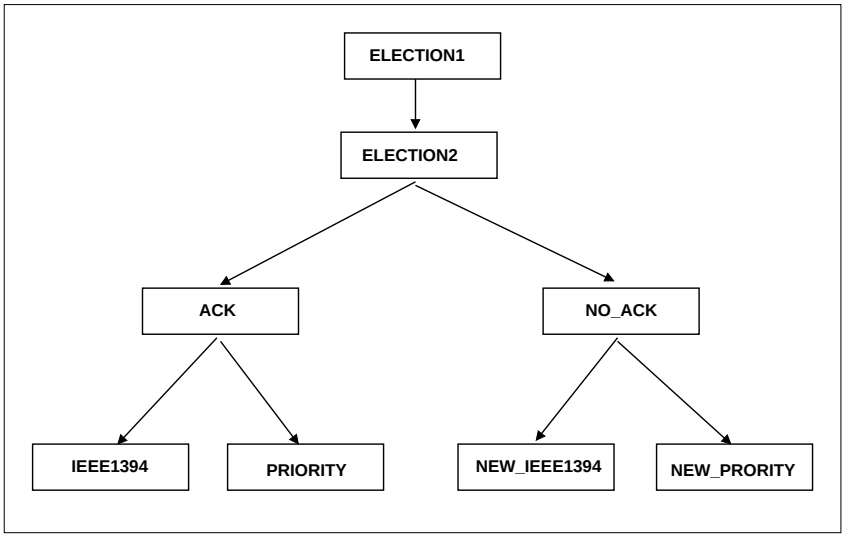


FIG. 3.1 – Proof-based development for the leader election problem

The property is proved using the same induction principle over spanning trees $f(x)$ and $f(y)$; the universally quantified variable is instantiated by $\mathsf{dom}(t) \cup \{x, y\}$. The current state of the development refers to a first model expressing the one-shot election ELECTION_1 and a second model ELECTION_2 refining ELECTION_1 (see the figure 1). Deadlock-freeness is proved by adding the condition that at least one event is triggered ; the graph is connected and only two nodes can be in contention. In fact, the global process is a set of trees converging to a tree.

The next step is crucial, since it prepares the distribution of models by introducing messages. The localisation of variables should be delayed as far as possible, while refining models : proofs are generally less complex on abstract models. Before we give the next refinements, we should recall that the first proof-based development of the IEEE 1394 leader election protocol [7] was based on a

description using a C-like programming language, where we had misunderstood how the contention problem was solved. The contention problem is either solved, or avoided :

- avoidance of contention is ensured by a mechanism of priority between nodes ; since at most two nodes may be in contention, we elect the node with the highest priority to be the leader ; the solution requires a total ordering over nodes and the identification of each plugged component : refinement model ELECTION_PRIORITY (see [19] for details of the development).
- resolution of contention is solved by a probabilistic mechanism ; when two nodes are in contention, both go back to the state before sending a message to the other node, as long as the contention state is happening ; there may exist an infinite sequence of contention states but under probabilistic assumption, the contention is solved : ELECTION_IEEE [7].

Both solutions use acknowledgements and we have located both solutions under the node **ACK**. We now show that acknowldgement messages can be remoted and we can partially replay both designs of algorithm and derive two new solutions without **ACK**.

3.3 The leader election protocol without acknowledgement

The development of the two new distributed algorithms shares events, which are independant on the contention problem. In order to improve the presentation, the first subsection describes events not related to the contention management ; the second subsection addresses the contention resolution and the use of unnamed nodes ; the last subsection derives a solution without acknowledgement and is based on the priority counters for avoiding the contention.

3.3.1 Contention-free Development Part

In our initial work [7], we introduced events for managing messages during the second refinement. Since messages were followed by one acknowledgement (t), we should also model confirmations. We discovered that there were confirmations, when we localised the graph and the variables. The call for the workshop dedicated to the protocol IEEE 1394 asked whether confirmations were necessary. At that time, we answered yes. The current work answers no, because we can assert that confirmations are not required. On the other hand, we can not remove the sending of messages. The event progress is introduced and the node x sends a message to y ; the reader will notice that the message is sent only, if x did not yet accept the submission of y ($y \mapsto x \notin t$) and that x did not send yet its message ($x \notin \mathsf{dom}\,(m)$) :

$\mathsf{send\_msg}_3 \;\widehat{=}$
ANY x, y WHERE
$x \mapsto y \in g \;\wedge$
$g[\{x\}] = t^{-1}[\{x\}] \cup \{y\} \;\wedge$
$y \mapsto x \notin t \;\wedge$
$x \notin \mathsf{dom}\,(m)$
THEN
$m := m \cup \{x \mapsto y\}$
END

$\mathsf{progress}_3 \;\widehat{=}$
ANY x, y WHERE
$x \mapsto y \in m - t \;\wedge$
$y \notin \mathsf{dom}\,(m)$
THEN
$t := t \cup \{x \mapsto y\}$
END

The abstract event progress expresses that a node y, whose neighbour x has asked to be its child, has got the message and has accepted the parenthood of x. In the previous version, the node y receives the message of x ($x \mapsto y \in m$)) and y did not yet accept it ($x \mapsto y \notin t$) ; y can also be in the situation of sending a request itself and it did not send a message ($y \notin \mathsf{dom}\,(m)$). Invariants based on invariants of our prevous work, but are much simpler.

The first three lines express typing information and inclusion properties : messages follow the route of the graph. The fourth line expresses that, if x has sent a message to y ($x \mapsto y \in m$), then x is in a submission status and x accepts that y is its parent ($g[\{x\}] = t^{-1}[\{x\}] \cup \{y\}$).

$$
\begin{array}{l}
m \in N \nrightarrow N \\
t \subseteq m \\
m \subseteq g \\
\forall (x,y) \cdot \big(x \mapsto y \in m \implies g[\{x\}] = t^{-1}[\{x\}] \cup \{y\}\big)
\end{array}
$$

3.3.2 Contention resolution

$$
\begin{array}{l}
x \mapsto y \in m - t \\
y \in \mathsf{dom}\,(m)
\end{array}
$$

The guard of $\mathsf{progress}_3$ is not validated, when x has sent a message to y and when y has sent a message.

The current events are not enough for reacting to this state and the current model is completed by a new event $\mathsf{contention}_3$, which discovers the contention status and which reacts to the previous guard. We have now two new events for managing contention : $\mathsf{contention}_3$ discovers the contention and $\mathsf{solve\_contention}_3$ solves the contention. A contention channel called c is used to control the contention status.

$\mathsf{contention}_3 \ \widehat{=}$
ANY x, y WHERE
$x \mapsto y \in m - (t \cup c) \ \wedge$
$y \in \mathsf{dom}\,(m)$
THEN
$c := c \cup \{x \mapsto y\}$
END

$\mathsf{solve\_contention}_3 \ \widehat{=}$
ANY x, y WHERE
$x \in N \ \wedge \ y \in N \ \wedge$
$c = \{x \mapsto y, y \mapsto x\}$
THEN
$m := m - c \quad \|$
$c := \varnothing$
END

In the new invariant, if x has sent a message to y and if y has sent a message $y \in \mathsf{dom}\,(m)$, then y has sent its message to x (no other choice) and this message was not yet confirmed ($y \mapsto x \in m - t$). It includes properties satisfied by the variable c.

$$
\forall (x,y) \cdot \left(\begin{array}{l} x \mapsto y \in m - (t \cup c) \\ y \in \mathsf{dom}\,(m) \\ \implies \\ y \mapsto x \in m - t \end{array} \right)
$$

$$
\begin{array}{l}
c \subseteq m \\
t \cap c = \varnothing \\
\forall (x,y) \cdot (x \mapsto y \in c \implies y \mapsto x \in m - t)
\end{array}
$$

The proof summary of the new refinement is : 33 proof obligations are generated by the tool and no interaction is necessary. At this point, we know that confirmations are not necessary and confirmations can not be justified by lossy channels. The number of messages is three times less than in the first development. As long as a node which has sent a message, does not get a message from another node, there is no contention; if there is a problem on the protocol, the timeout mechanism allows us to reinitialise the protocol for a new election. The current model is called ELECTION_NEW_IEEE$_3$ and it can be refined to localise variables.

3.3.3 Contention prevention

Another way to deal with the contention problem is to use a priority technique, which will avoid the contention. The solution was discovered after a misunderstanding of the IEEE 1394 initial solution and we did not know it. In fact, the solution was mentionned by N. Lynch [32] and was first proposed by D. Anluin [11].

When two nodes are in contention (and at most two nodes can be in contention, proved mechanically and formally), each node can not send an acknowledgement to the other node; one of them should not be able to send this ack and the other one must do it. The main idea is to introduce a *unique counter* called *ctr* and it means that each node is uniquely identified and must be

identifiable. The assumption is that there is a mechanism for naming. ctr is a total injection from Nodes into natural numbers. In a real network, one can assume that equipment might be uniquely identified by an unique address, for instance, but it is not the general rule.

The new event is called avoid_solve_cnt$_3$. Like for send_ack$_3$, it adds the pair $x \mapsto y$ to ack.

avoid_solve_cnt $=$
ANY x, y WHERE
$x, y \in m - t \quad \wedge \; y \in \mathsf{dom}\,(m) \; \wedge \; ctr(x) < ctr(y)$
THEN
$t := t \; \cup \; \{x \mapsto y\}$
END

The two differences with the guard of event progress$_3$ concern the condition $y \in \mathsf{dom}\,(m)$, which is true in avoid_solve_cnt and false in progress$_3$ and the guard $ctr(x) < ctr(y)$ is added to the event avoid_solve_cnt. Since ctr is an injection, both nodes x and y can not be triggered concurrently. The proof of the invariant requires the following additional invariant :

$$\forall\,(x,y)\cdot\left(\left(\begin{array}{c} x,y \in t \;\wedge \\ y \mapsto x \in m \end{array}\right) \implies ctr(x) < ctr(y)\right)$$

The proof summary of the current refinement is : 13 proof obligations are generated by the tool and no interaction is necessary. The current model is called ELECTION_NEW_PRIORITY$_3$ and it can be refined to localise variables.

3.4 Localisation of events and data

The final step is to localise information on the events and the data. For instance, we have localised the graph g by the constants n (for neighbour) and the variable t with the variable b like in our previous work [7]. The gluing invariant is required to localise information. For instance, the event elect is a refined version of the initial event of the model ELECTION_1 and the guard is expressed using $n(x)$ and $b(x)$ which are related to the defintion of g and t by the gluing invariant.

$n \in N \to \mathbb{P}(N)$

$\forall x \cdot (\, x \in N \implies n(x) = g[\{x\}]\,)$

$b \in N \to \mathbb{P}(N)$

$\forall x \cdot (\, x \in N \implies b(x) = t^{-1}[\{x\}]\,)$

elect $\widehat{=}$
ANY x WHERE
$x \in N \; \wedge$
$n(x) = b(x)$
THEN
$l := x$
END

The final refinement of ELECTION_NEW_IEEE$_3$ leads to the modelling of wires. For each node, the variable D specifies the state ($high$ or low) of the wire between neighbours. If $y \in \mathsf{dom}(D(x))$, then there exists a wire between x and y. $D(x)(y) = low$ if x does not accepted the submisson from y. Variable B is a function with gives the state of a node. This state allows us to determine if the node has sent a message. We have substituted variables d (resp. bm) by D (resp. by B). We have introduced this model in two refinement steps.

$d(x) = n(x) - t^{-1}[/x/]$
$r(x) = card(d(x))$
$D \in N \to (N \nrightarrow \{low, high\})$
$\forall x \cdot (\, x \in N \implies \mathsf{dom}\,(D(x)) = n(x))$
$\forall x \cdot (\, x \in N \implies d(x) = D(x)^{-1}[\{low\}]\,)$

$B \in N \to \{low, high\})$
$bm = B^{-1}[\{high\}]$

$\mathsf{send\_msg}_9 \;\widehat{=}$
ANY x, y WHERE
$x \in N \;\wedge$
$B(x) = low \;\wedge$
$y \in D(x)^{-1}[\{low\}] \;\wedge$
$r(x) = 1$
THEN
$M := M \cup \{x \mapsto y\} \quad \|$
$B(x) := high$
END

$\mathsf{progress}_9 \;\widehat{=}$
ANY x, y WHERE
$x \mapsto y \in M \;\wedge$
$B(y) = low$
THEN
$D(y)(x) := high \quad \|$
$r(y) := r(y) - 1 \quad \|$
$M := M - \{x \mapsto y\}$
END

The proof summary of the fourth refinement is : 20 (9+11) proof obligations are generated by the tool and 7 (3+4) among them require interaction. We do not give the complete set of events of the last models but note that the localisation is systematic, as long as information of each node contain localisable information.

3.5 Conclusion and future work

The current work is a general presentation of work on the leader election problem in a distributed environment. We have used the development of the classical leader election protocol of the IEEE 1394 and we propose an improvement of the classical solution. Two new algorithms are developed incrementally by replaying previous developments and we show that acknowledgements are not needed in the leader election protocol. We have shown that a proof-based development can be easily reused to improve and/or discover new distributed algorithms. The new algorithm is still working like a submission algorithm but the traffic of messages is simpler. We replay from our previous development starting from the model which introduces messages. The call for the workshop dedicated to the protocol IEEE 1394 asked whether confirmations were necessary. At that time, we answered yes. The current work answers no, because we can assert that neither acknowledgements, nor confirmations are required. On the other hand, we can not remove the sending of messages.

Proof summary for the four leaader election algorithms is given in the array as follows :

Models	Number of proof obligations	Number of interactive steps
ELECTION_1	2	1
ELECTION_2	10	2
ELECTION_IEEE	41	0
ELECTION_PRIORITY	31	2
ELECTION_NEW_IEEE	33	0
ELECTION_NEW_PRIORITY	13	0
TOTAL	120	5

The localisation refinement requires the same proof obligations as those in the initial development in [7] and in [19] for ELECTION_PRIORITY.

Chapitre 4

Systèmes répartis à transactions

Sommaire

Notre travail se situe dans le cadre du développement incrémental prouvé de systèmes distribués communiquant par des canaux non fiables, en particulier des protocoles de communication utilisés dans les porte-monnaie électroniques. L'approche B événementielle est mise en application sur une étude de cas réputée du Grand Challenge et appelée Mondex. Ce travail met en avant une explication des mécanismes sous-jacents qui permettent à ce système de satisfaire les exigences initiales résumées sous la forme suivante : il n'y a pas de perte d'argent ni de création d'argent lors des transferts. Le développement permet de préserver cette propriété de sûreté attendue du système final et identifie des relations de communication spécifiques pour assurer la mise en œuvre du protocole. Ainsi, un premier modèle énonce la spécification globale de sûreté ; un second modèle raffiné décrit le comportement détaillé des différents protagonistes ; un troisième modèle localise les données ; un quatrième modèle élimine des variables utiles pour faciliter les preuves. Cette étude nous permet de vérifier la robustesse du protocole à la perte de messages.

4.1 Introduction

Notre démarche vise à faire des études de cas de systèmes distribués communiquant par des canaux non fiables et de voir quelles sont les difficultés inhérentes à ce genre de systèmes afin de pouvoir en déduire plus tard des patrons de développement de systèmes distribués à l'aide de la méthode B événementielle. Pour y parvenir on doit s'appuyer sur des études de cas suffisamment complexes afin de témoigner de l'utilité des techniques de preuve et de raffinement. Aussi, notre choix de développer Mondex [40, 1] s'appuie sur la complexité inhérente du cas d'étude et sur l'intérêt de cette étude dans le cadre du Grand Challenge [27, 42]. Michael Butler [16] a développé ce système en B événementiel et nous apporterons quelques éléments de comparaison dans la dernière partie. On peut donc considérer notre travail comme un exercice de modélisation et une première étape pour la mise en œuvre de patrons de développement pour ce genre de système.

Mondex [40, 1] est un système de paiement électronique qui fut introduit pour la première fois par la *National Westminster Bank* en 1990 avant d'être repris par *MasterCard*. Le système est basé sur les cartes à puce servant de porte-monnaie électroniques contenant chacun un solde d'argent. Une des spécificités du système est que les transactions sont *hors-ligne*, ceci signifie que les transactions se passent entre deux porte-monnaie sans l'intervention d'une autorité centralisée. Au cours d'une transaction, chaque porte-monnaie doit donc être capable de garantir lui-même les mesures de sécurité inhérente à ce genre de système, sans l'intervention d'une quelconque autorité de régulation.

Le protocole *Mondex*

Chaque porte-monnaie est initialement crédité d'un solde. La transaction consiste en un transfert d'argent entre un porte-monnaie source et un porte-monnaie destination. La figure 4.1 montre le schéma global d'une transaction.

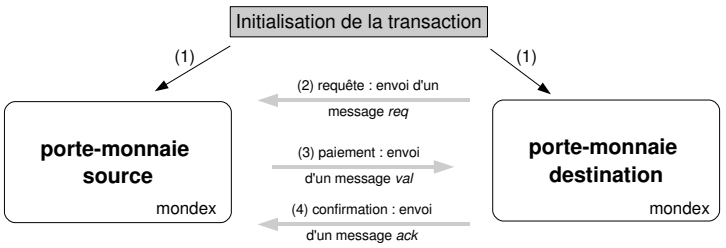


FIG. 4.1 – Les différentes étapes d'une transaction

Le schéma de la figure 4.1 montre le cas d'une transaction où tout se passe bien. Mais dans la réalité plusieurs événements peuvent perturber le déroulement d'une transaction. Il peut s'agir

par exemple d'une défaillance du canal de communication ou encore d'un retrait prématuré de la carte à puce du lecteur de carte, avant que la transaction n'arrive à son terme. Pour prévenir ce genre d'erreurs, un certain nombre de dispositifs ont été prévus. En plus de son identifiant et de son solde, un porte-monnaie contient un journal d'erreurs dans lequel les transactions qui ont échoué sont transcrites. Il contient également le numéro de séquence de la prochaine transaction à effectuer. Un porte-monnaie a aussi un état qui varie au fur et à mesure que la transaction avance. Les différents états possibles sont *eaFrom, eaTo, epr, epv, epa* (voir figure 4.2).

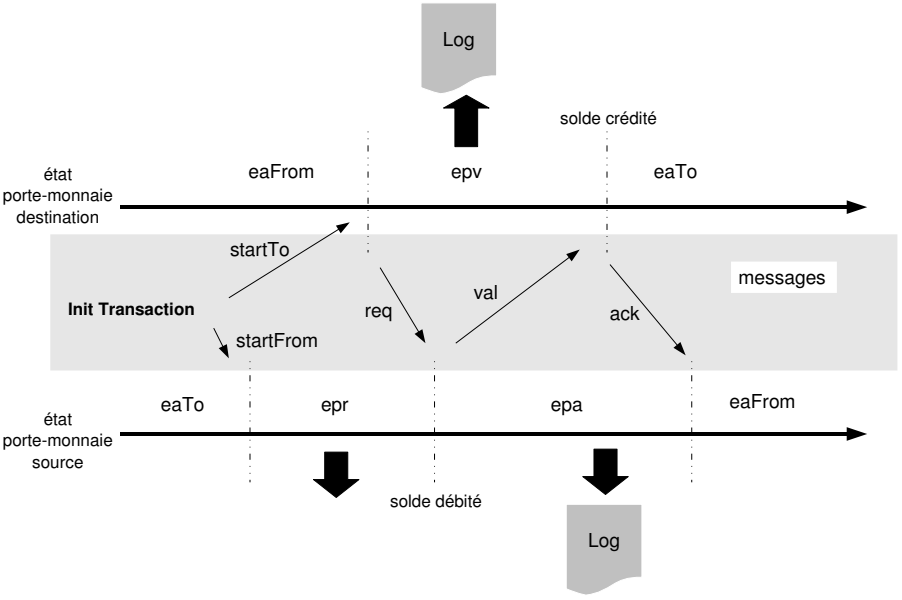


FIG. 4.2 – Le protocole Mondex

Lorsqu'un porte-monnaie source abandonne la transaction prématurément alors qu'il a déjà débité son solde (*état epa*), il inscrit la transaction en cours sur son propre journal d'erreurs. De même pour un porte-monnaie cible qui a envoyé un message de type *Req* et qui n'a pas encore crédité son solde (*état epv*). Grâce à ce procédé, le croisement des informations contenues dans les journaux d'erreurs des différents porte-monnaie permettra de récupérer l'argent apparemment perdu localement.

4.2 Développement du Mondex

Le développement du Mondex se fera en plusieurs étapes successives : un premier modèle abstrait suivi de dix raffinements qui rendront le modèle de plus en plus concret jusqu'à ce que tous les aspects du protocole soient pris en considération.
Dans le premier modèle abstrait, la transaction entre deux porte-monnaie est atomique, le transfert d'argent se fait en un coup. Le solde du porte-monnaie source est débité au même temps que le solde du porte-monnaie cible est crédité (figure 4.3). Dans ce premier modèle sont exprimées les

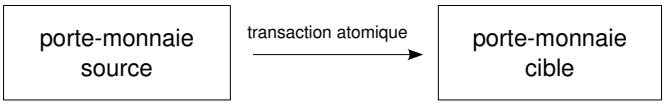


FIG. 4.3 – Une transaction dans le modèle abstrait

principales propriétés de sûreté que le système doit garantir, à savoir :
- Pas d'argent créé dans le système.
- Pas d'argent perdu dans le système.

4.2.1 Modèle abstrait : Transfert atomique

Dans ce premier modèle, les porte-monnaie ont un identifiant propre à chacun. Tous les identifiants possibles sont contenus dans un ensemble *NAME*. Chaque porte-monnaie contient en plus de son nom deux paramètres :
- Solde courant du porte-monnaie.
- Argent perdu localement : quand une transaction échoue, l'argent qui n'est pas arrivé au porte-monnaie cible est stocké comme argent perdu localement sur le porte-monnaie source.

Un certain nombre de types sont introduits pour les besoins de la modélisation :

SETS

$NAME$ /* ensemble des identifiants des porte-monnaie authentiques */

CONSTANTS

$VALUE$

AXIOMS

$axm1 : VALUE = \mathbb{N}$

END

L'état du système à un moment donné est reflété par des variables qui contiennent le solde et l'argent perdu localement par chaque porte-monnaie :

VARIABLES

$abAuthB$ /* solde de chaque porte-monnaie authentique */
$abAuthB'$ /* sauvegarde de l'ancienne valeur de la variable abAuthB */
$abAuthL$ /*argent perdu localement par chaque porte-monnaie authentique*/
$abAuthL'$ /* sauvegarde de l'ancienne valeur de la variable abAuthL */

INVARIANTS

$inv1 : abAuthB \in NAME \rightarrow VALUE$
$inv2 : abAuthB1 \in NAME \rightarrow VALUE$
$inv3 : abAuthL \in NAME \rightarrow VALUE$
$inv4 : abAuthL1 \in NAME \rightarrow VALUE$

L'invariant quand à lui exprime les deux propriétés de sûreté énoncées précédemment, pas d'argent créé frauduleusement :

$inv5 : SUM(abAuthB) \leq SUM(abAuthB')$

Par ailleurs, l'argent ne peut être perdu globalement, ceci est exprimé par le fait que la somme des soldes et de l'argent perdu localement par tous les porte-monnaie demeurent inchangés :

$inv6 : SUM(abAuthB) + SUM(abAuthL) = SUM(abAuthB') + SUM(abAuthL')$

Les transactions étant atomique dans ce premier modèle, il y a donc que deux événements $TransfertOk$ et $TransfertLost$. En un coup la transaction est soit réussie ou non et les variables sont modifiées en conséquence :

EVENT TransfertOk
ANY
$from$
to
v
WHERE
$grd1 : from \in NAME \wedge to \in NAME \wedge v \in VALUE$
$grd2 : abAuthB(from) \geq v$
$grd3 : from \neq to$
THEN
$act1 : abAuthB' := abAuthB$
$act2 : abAuthL' := abAuthL$
$act3 : abAuthB := abAuthB \Leftarrow \{from \mapsto (abAuthB(from) - v), to \mapsto (abAuthB(to) + v)\}$
END

Contrairement à l'événement $TransfertOk$, dans l'événement $TransfertLost$ l'argent n'est pas transféré vers le solde du porte-monnaie destination mais il est stocké comme argent perdu localement par le porte-monnaie source dans la variable $abAuthL$.

EVENT TransfertLost
ANY
$from$
to
v
WHERE
$grd1 : from \in NAME \wedge to \in NAME \wedge v \in VALUE$
$grd2 : abAuthB(from) \geq v$
$grd3 : from \neq to$
THEN
$act1 : abAuthB' := abAuthB$
$act2 : abAuthL' := abAuthL$
$act3 : abAuthB(from) := (abAuthB(from) - v)$
$act4 : abAuthL(from) := (abAuthL(from) + v)$
END

4.2.2 Premier raffinement

Pour implémenter le protocole Mondex (voir figure 4.2), chaque porte-monnaie électronique a une structure complexe comme le montre la figure 4.4.

Il est donc nécessaire d'introduire les détails du protocole étape par étape pour réduire l'effort de preuve. Nous utiliserons pour cela la notion de transactions qui sera très abstraite dans le premier raffinement avant d'arriver aux transactions sous leur forme concrète de Mondex au bout du dixième raffinement. Une transaction concrète telle que montrée dans la figure 4.2 se déroule au niveau abstrait comme le montre la figure 4.5

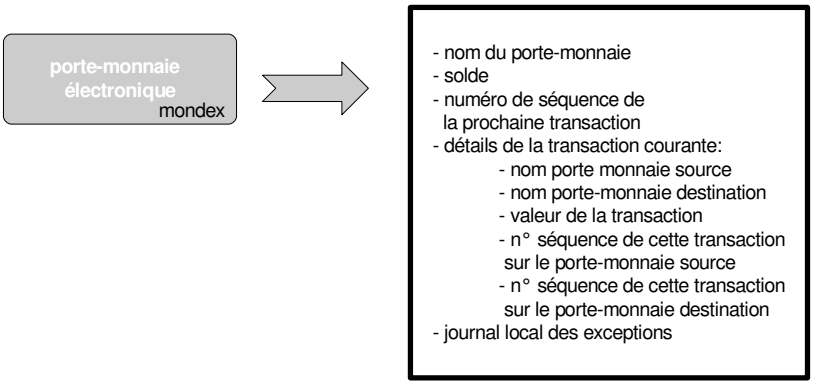


FIG. 4.4 – La structure d'un porte-monnaie électronique

Deux nouveaux ensembles porteurs sont introduits, $transactions$ qui contient l'ensemble des toutes les transactions possibles. L'ensemble $STATUS$ contient quand à lui les états possibles d'une transaction. Chaque transaction a un porte-monnaie source, un porte-monnaie destination et une valeur.

SETS

$transactions$
$STATUS = \{ready, progress, success, fail\}$

CONSTANTS

t_src
t_dst
t_value

AXIOMS

$axm1 : t\_src \in transactions \rightarrow NAME$
$axm2 : t\_dst \in transactions \rightarrow NAME$
$axm3 : t\_value \in transactions \rightarrow VALUE$

END

L'état d'une transaction est donné par une variable $status$ dont le domaine est un ensemble $trans$ qui contient les transaction déjà terminées ou qui sont encore en cours :
$inv1 : trans \subseteq transactions$
$inv2 : status \in trans \rightarrow STATUS$

Un nouvel événement $initTrans$ initialisant les transactions est introduit :

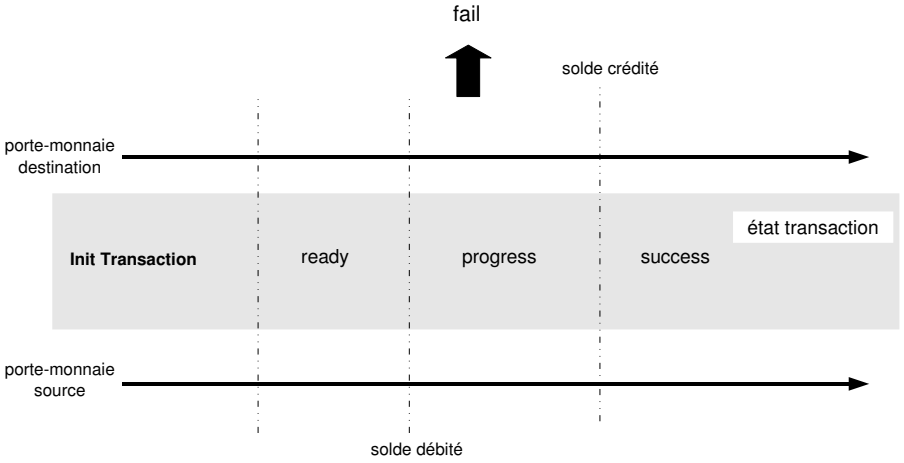


FIG. 4.5 – Une transaction abstraite

EVENT InitTrans
ANY
t
WHERE
$grd1 : t \in transactions$
$grd2 : t\_src(t) \neq t\_dst(t)$
$grd3 : t \notin trans$
THEN
$act1 : trans := trans \cup \{t\}$
$act2 : status := status \lhd\!\!+ \{t \mapsto ready\}$
END

Dans ce premier raffinement, les transactions ne sont plus atomiques. Entre le moment où l'argent est débité du porte-monnaie source et le moment où il est crédité dans le porte-monnaie destination, il s'écoule un temps où l'issue de la transaction est incertaine. Nous utilisons une variable *maybelost* qui contiendra pour un porte-monnaie donné les transactions dont l'issue est incertaine :

$inv3 : maybelost \in NAME \rightarrow \mathbb{P}(transactions)$

Le solde concret des porte-monnaie est donné par une variable *conBalance* :

$inv4 : conBalance \in NAME \rightarrow VALUE$

Comme le montre la figure 4.6, la relation entre le solde abstrait *abAuthB* de la spécification et les deux variable *maybelost* et *conBalance* est exprimée dans l'invariant comme suit :

. $inv5 : \forall from \cdot (from \in NAME \Rightarrow$
$abAuthB(from) = conBalance(from) + SUMT(maybelost(from)))$

$SUMT$ étant une constante servant à calculer la somme d'argent contenu dans un ensemble de transactions donné.

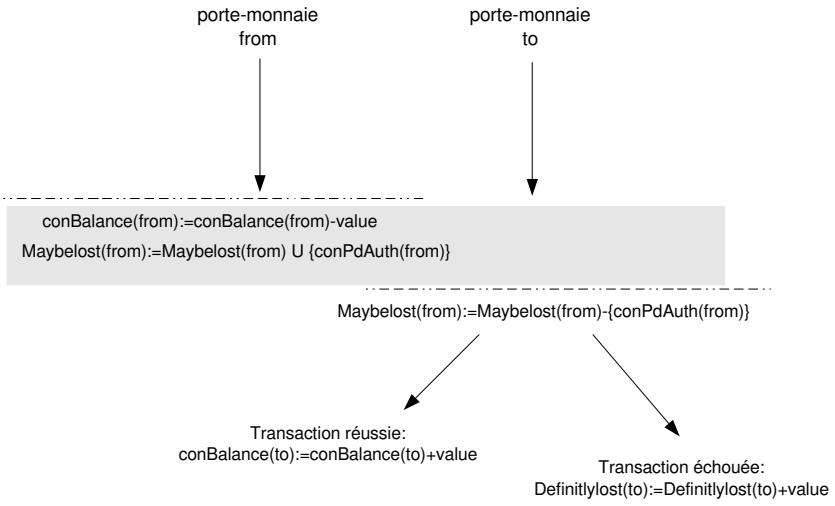


FIG. 4.6 – La variable *maybeLost*

Un événement $transfert$ correspondant au débit du porte-monnaie source apparaît dans ce raffinement :

EVENT transfert
ANY
t
WHERE
$grd1 : t \in transactions$
$grd2 : t\_src(t) \neq t\_dst(t)$
$grd3 : t \in trans$
$grd4 : t \notin maybelost(t\_src(t))$
$grd5 : status(t) = ready$
$grd6 : conBalance(t\_src(t)) \geq t\_value(t)$
THEN
$act1 : maybelost(t\_src(t)) := maybelost(t\_src(t)) \cup \{t\}$
$act2 : status(t) := progress$
$act3 : conBalance(t\_src(t)) := conBalance(t\_src(t)) - t\_value(t)$
END

Les deux événements$transfertOk$ et $transfertLost$ sont quand à eux raffinés comme suit :

EVENT TransfertOk
REFINES TransfertOk
ANY
t
WHERE
$grd1 : t \in transactions$
$grd2 : t\_src(t) \neq t\_dst(t)$
$grd3 : t \in trans$
$grd4 : t \in maybelost(t\_src(t))$
$grd5 : status(t) = progress$
WITHNESSES
$from : from = t\_src(t)$
$to : to = t\_dst(t)$
$v : v = t\_value(t)$
THEN
$act1 : maybelost(t\_src(t)) := maybelost(t\_src(t)) \setminus \{t\}$
$act2 : status(t) := success$
$act3 : conBalance(t\_dst(t)) := conBalance(t\_dst(t)) + t\_value(t)$
END

EVENT TransfertLost
REFINES TransfertLost
ANY
t
WHERE
$grd1 : t \in transactions$
$grd2 : t\_src(t) \neq t\_dst(t)$
$grd3 : t \in trans$
$grd4 : t \in maybelost(t\_src(t))$
$grd5 : status(t) = progress$
WITHNESSES
$from : from = t\_src(t)$
$to : to = t\_dst(t)$
$v : v = t\_value(t)$
THEN
$act1 : maybelost(t\_src(t)) := maybelost(t\_src(t)) \setminus \{t\}$
$act2 : status(t) := fail$
$act3 : abAuthL(from) := (abAuthL(from) + v)$
END

4.2.3 Deuxième raffinement

Dans ce deuxième raffinement les variables *maybelost* et *abAuthL* sont supprimées et sont exprimée avec les autres variables à l'aide de l'invariant de collage :

$inv1 : \forall from \cdot (from \in NAME \Rightarrow$
$maybelost(from) = (dom(status \rhd \{progress\}) \cap dom(t\_src \rhd \{from\})))$
$inv2 : \forall from \cdot (from \in NAME \Rightarrow$
$abAuthL(from) = (dom(status \rhd \{fail\}) \cap dom(t\_src \rhd \{from\})))$

4.2.4 Troisième raffinement

Le but de ce raffinement est de remplacer l'état d'une transaction contenu dans la variable *s*tatus par l'état de chacun des porte-monnaie impliqués dans cette transaction. L'état des porte-monnaie est contenu dans une nouvelle variable $c_$status. Dans Mondex, un porte-monnaie est dans l'un des états suivant :

- $eaFrom$, $eaTo$: le porte-monnaie n'est pas actif.
- epr : le porte-monnaie est source dans la transaction et il attend que la destination lui envoie la somme à verser.
- epa : le porte-monnaie est source de la transactions, il a versé l'argent à la destination et il attend l'acquittement.
- epv : le porte-monnaie est destination, il a fait la demande à la source et il attend le versement de la somme requise.

La figure 4.2 montre les différents états des porte-monnaie. La variable $c_$status est donc définie comme suit :

$axm1 : c\_STATES = \{eaFrom, epr, epa, epv\}$
$inv1 : c\_status \in NAME \rightarrow c\_STATES$

Comme le montre la figure 4.4, chaque porte-monnaie connaît à un moment donné les détails de la transaction dans laquelle il est partie prenante. Nous avons donc introduit une variable p_trans liant chaque porte-monnaie à la transaction dans laquelle il est impliqué :

$inv3 : p\_trans \in NAME \rightarrow transactions$

Des invariants ont été rajoutés pour prouver ce raffinement. Ces invariants sont nécessaires pour décrire l'état d'une transaction avec l'état des porte-monnaie. Un exemple d'invariant est donné en ce qui suit :

$$\begin{aligned} inv6 : \forall t \cdot (t \in trans \wedge\ & p\_trans(t\_src(t)) = t \wedge \\ & p\_trans(t\_dst(t)) = t \wedge \\ & t\_src(t) \neq t\_dst(t) \wedge \\ & c\_status(t\_src(t)) = epa \wedge \\ & c\_status(t\_dst(t)) = epv \Rightarrow \\ & status(t) = progress) \end{aligned}$$

4.2.5 Quatrième raffinement

Ce quatrième raffinement vise à enlever la variable *t*rans. Cette variable contient les transactions en cours ou les transactions déjà terminées. Dans le protocole Mondex chaque porte-monnaie a un numéro séquentiel qui sera associé à la prochaine transaction dans laquelle il sera partie prenante. Une variable $p_$seq est utilisée :

$inv1 : p\_seq \in NAME \rightarrow \mathbb{N}$

A chaque transaction sont associés le numéro séquentiel du porte-monnaie source et celui du porte-monnaie destination au moment ou la transaction a commencé :

$axm1 : f\_seq \in transactions \rightarrow \mathbb{N}$
$axm2 : t\_seq \in transactions \rightarrow \mathbb{N}$

Pour enlever la variable *t*rans, les invariants nécessaires sont les suivants :

$inv2 : \forall t \cdot (t \in transactions \wedge t \in trans \Rightarrow f\_seq(t) < p\_seq(t\_src(t)))$
$inv3 : \forall t \cdot (t \in transactions \wedge t \in trans \Rightarrow t\_seq(t) < p\_seq(t\_dst(t)))$

4.2.6 Cinquième, sixième et septième raffinements

Ces raffinements sont essentiellement des raffinements de données. A l'issue du septième raffinement, les porte-monnaie sont modélisés sous leur forme la plus concrète comme le montre la figure 4.4. Les variables suivantes sont introduites :

- identifiant du porte-monnaie source de la transaction en cours :
  $inv1 : p\_from \in NAME \rightarrow NAME$
- identifiant du porte-monnaie destination de la transaction en cours :
  $inv2 : p\_to \in NAME \rightarrow NAME$
- numéro séquentiel du porte-monnaie source :
  $inv3 : p\_f\_seq \in NAME \rightarrow \mathbb{N}$
- numéro séquentiel du porte-monnaie destination :
  $inv4 : p\_t\_seq \in NAME \rightarrow \mathbb{N}$
- valeur monnaitaire de la transaction en cours :
  $inv1 : p\_value \in NAME \rightarrow \mathbb{N}$
- journal d'erreurs local :
  $inv1 : p\_log \in NAME \rightarrow \mathbb{P}(NAME \times NAME \times \mathbb{N} \times \mathbb{N} \times \mathbb{N})$

4.2.7 Huitième raffinement

Un des principes essentiels des systèmes distribués est qu'une entité du système ne peut connaître l'état des autres entités. Une manière de synchroniser les différentes entités du système est l'utilisation des messages. Dans ce huitième raffinement divers types de messages sont introduits avec des variables contenant chacune un type de message (voir figure 4.2).

Rejeu et perte de messages

Afin de tester la robustesse du système, des événements simulant la perte des messages aléatoirement ont été introduits. Les preuves de raffinement ont pu être faites ce qui prouve qu'en cas de disparition d'un message, la transaction ne peut aller à son terme mais que l'argent ne peut être perdu car en abandonnant la transaction le porte-monnaie rajoute la transaction en cours dans son journal d'erreurs.

Pour le rejeu, il faut noter que dans les différents événements les messages ne sont pas effacés après leurs lecture d'où le risque de lire plusieurs fois le même message. Ceci peut, par exemple, générer de l'argent illégalement. Dans notre cas, grâce à la présence du numéro de la transaction courante sur le porte-monnaie et sur le message, on arrive à prouver que message d'une transaction précédente ne peut être confondu avec un message de la transaction courante.

4.3 Conclusion

Nous avons dans cette étude modélisé et prouvé le protocole Mondex dans la perspective de développer des patrons pour les protocoles de communications à travers des canaux non fiables. Cette étude de cas nous a permis de voir qu'elles étaient les difficultés liées à ce genre de système et la manière avec laquelle les gérer :

- Les propriétés de sûreté du système qui doivent être préservées sont exprimées dans le modèle abstrait dans lequel la transaction est atomique.
- Modéliser d'abord des transactions abstraites et raffiner ensuite le système pour que les transactions ne soient plus atomiques jusqu'à arriver au protocole sous sa forme concrète.
- Les exceptions (transaction interrompues prématurément) sont gérées avec une variable abstraite *abAuthLost* qui est ensuite concrétisée dans les raffinements suivants (avec les journaux d'erreurs dans le cas Mondex).
- La nécessité de distinguer les messages de même type afin de prévenir les risques qu'un même message soit utilisé plusieurs fois. Dans le cas Mondex un numéro séquentiel associé à la transaction courante figurant sur les messages et les porte-monnaie est utilisé.
- Lorsqu'il y a perte des messages, la transaction doit pouvoir être interrompue par la partie qui attend ce message sans que cela ne remette en cause les propriétés de sûreté du système. Dans le cas Mondex, ceci est garanti par le fait qu'à chaque fois qu'un porte-monnaie interrompt une transaction il la rajoute dans son journal d'erreurs.

L'ensemble du processus de modélisation a produit 557 obligations de preuves réparties sur les différents raffinements comme suit :

Modèle	Nombre Total de PO	Automatique	Interactives
Modèle abstrait	27	27 (100%)	0 (0%)
Premier raffinement	45	45 (100%)	0 (0%)
Deuxième raffinement	17	17 (100%)	0 (0%)
Troisième raffinement	263	191 (72%)	72 (28%)
Quatrième raffinement	114	114 (100%)	0 (0%)
5^e,6^e et 7^e raffinement	104	104 (100%)	0 (86%)
Total	570	498 (87%)	72 (13%)

Notre modélisation est fondée sur le modèle Z [40] et comprend des variables intermédiaires permettant d'exprimer que la somme des soldes ne peut pas augmenter ce qui n'est pas le cas de l'article de Butler [16] nous montre que cet élément n'est pas intégré à ses modèles. Enfin, Butler ajoute un événement qui transfère l'argent se trouvant perdu vers le solde, alors que cette opération est à réaliser par la banque dans le cadre de la mise à jour de la carte.

Chapitre 5

Algorithme d'énumération

Sommaire

5.1 Introduction

L'algorithme d'énumération des nœuds d'un graphe de Mazurkiewicz [34] est un algorithme distribué qui attribue à chaque nœud d'un graphe non orienté un numéro entre 1 et m où m est le nombre de nœuds du graphe. Un nœud se synchronise avec l'ensemble de ses voisins et un nœud peut modifier son état ainsi que celui de ses voisins. Il est donc distribué sur les boules du graphe. Une boule de centre x est composée du nœud x et de l'ensemble de ses voisins. L'algorithme construit une numérotation qui converge vers un revêtement du graphe. Un revêtement est un homomorphisme surjectif des nœuds du graphe vers ceux d'un autre qui conserve la structure des arêtes (donc des boules). Un revêtement est localement bijectif (sur une boule). Pour un graphe minimal au sens des revêtements les seuls revêtements surjectifs sont des injections (donc des bijections). Cet algorithme termine sur des graphes non orientés non-réflexifs connexes et mininaux pour les revêtements.

Pour montrer la correction et la terminaison de l'algorithme d'énumération de Mazurkiewicz, il faut montrer de nombreuses propriétés qui, de notre point de vue, sont trop compliquées à prouver *directement* avec des assistants de preuve. Le but de cette étude est d'exhiber une suite de modèles de plus en plus concrets tels que chaque modèle capture une des propriétés énoncées. Cette technique s'appelle le raffinement et nous prouvons la relation de raffinement qui lie chaque modèle avec son abstraction.

5.2 L'algorithme d'énumération de Mazurkiewicz

L'algorithme d'énumération de Mazurkiewicz est défini à l'aide de règles qui expliquent comment un nœud du graphe change son numéro courant (Règle de renommage) et comment l'information sur les nœuds se propage (Règle de diffusion). La variable n est la numérotation courante. La variable N est une fonction des nœuds vers l'ensemble des numéros (différents de 0) de ses voisins. La variable M (pour mémoire ou boite à lettres) associe à chaque nœud un ensemble de couples (k, Nk) où k est un numéro attribué et Nk un ensemble de numéros différents de 0. Lors de l'exécution de l'algorithme, un nœud x a k comme numéro et Nk comme $N(x)$.

L'algorithme d'énumération de Mazurkiewicz est défini à l'aide de deux règles qui expliquent comment un nœud et ses voisins changent leurs variables. On suppose que les autres nœuds ne changent pas leurs variables durant l'application de ces règles.

- Règle de renommage : Lorsqu'un nœud x voit que tous ses voisins ont la même mémoire que lui et qu'il a son numéro à 0 ou s'il y a dans sa mémoire un couple $(n(x), V)$ tel que V est plus grand que $N(x)$, alors il peut changer de numéro.

  Precondition

  $\forall v \in B(x), M(v) = M(x)$

  $(n(x) = 0) \ \vee \ (n(x) > 0 \wedge \exists (n(x), V) \in M(x) \mid (N(x) \prec V)$

  Postcondition

  $n'(x) = 1 + \max\{n \mid (n, V) \in M(x)\}$

  $\forall y \in X - \{x\}, \ n'(y) = n(y)$

  $\forall v \in B(x) - \{x\}, \ N'(v) = (N(v) - \{n(x)\}) \cup \{n'(x)\}$

  $\forall v \in \{x\} \cup (X - B(x)), \ N'(v) = N(v)$

  $\forall v \in B(x), \ M'(v) = M(v) \cup \{(n'(w), N'(w)) \mid w \in B(x)\}$

  $\forall v \in X - B(x), \ M'(v) = M(v)$

- Règle de diffusion : Si un nœud x n'a pas la même mémoire qu'un de ses voisins, alors l'ensemble des nœuds de $B(x)$ (le nœud x et ses voisins) se synchronisent en prenant l'union de toutes les mémoires de x et de ses voisins.

  Precondition

  $\exists v \in B(x)$ such that $M(v) \neq M(x)$

  Postcondition

  $\forall v \in B(x), \ M'(v) = \bigcup_{w \in B(x)} M(w)$

  $\forall v \in X - B(x), \ M'(v) = M(v)$

Lorsqu'aucune des rêgles n'est applicable, alors tous les nœuds ont la même mémoire et la numérotation est un revêtement.

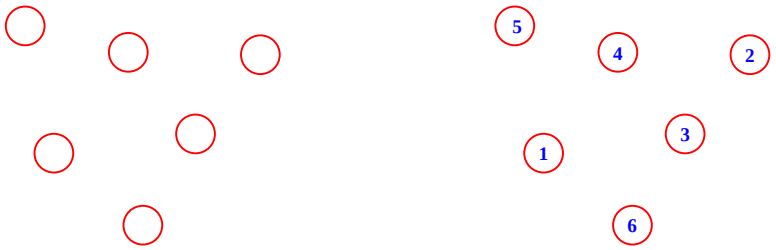


FIG. 5.1 – Premier modèle : un seul coup

5.3 Plan du développement

Lors d'une réunion RIMEL (avril 2007), Yves Métivier nous a expliqué cet algorithme. Il nous a exposé les deux règles et, pour nous convaincre de la correction de l'algorithme, il nous a donné des arguments et des propriétés de cet algorithme. Pour notre développement, nous avons voulu capturer ces propriétés fondamentales de l'algorithme pour ne pas faire toutes les preuves sur le modèle final.

- Le modèle "one shot" sur les nœuds seulement (global).
- Introduction de l'essence de l'algorithlme. Comment les nœuds modifient la numérotation courante (encore global) avec une abstraction de la règle de renommage. Ce modèle définit le codomaine et la surjectivité du futur homomorphisme.
- Introduction des arêtes ; un nœud et ses voisins auront des numéros différents (voisins des voisins + global). Sur ce modèle nous avons une injection locale du futur homomorphisme.
- Introduction de la variable N qui donne les numéros ($\neq 0$) des voisins (raffinement de donnée + superposition).
- Introduction de la variable M de l'algorithme de Mazurkiewicz (local sur les boules) et de la règle de diffusion.
- Introduction de la connexité et de la minimalité du graphe pour la correction.

5.4 Premier modèle

Nous avons les déclarations et les événements suivants :

SETS
X
CONSTANTS
m
AXIOMS
$finite(X)$
$m \in \mathbb{N}$
$m = card(X)$

VARIABLES
nb
INVARIANT
$nb \in X \rightarrow 1..m$
INITIALISATION
$nb :\in X \rightarrow 1..m$
EVENTS
enum $\widehat{=}$ $nb :\in X \rightarrowtail\!\!\!\!\!\rightarrow 1..m$

X est un ensemble fini de m éléments ($card(X) = m$). L'événement enum fait en sorte que la valeur suivante de nb soit choisie, de manière non déterministe, dans l'ensemble des bijections de X vers $1..m$.

La figure 5.1 montre deux états d'un graphe de six nœuds. Avant et après l'événement enum. Ensuite, les nœuds du graphe se sont correctement numérotés en choisissant tous un numéro différent dans 1..6.

5.5 Second modèle : essence de l'algorithme

Ce modèle introduit l'essence de l'algorithme. La propagation de la numérotation jusqu'à une énumération. Seule la règle de renommage est introduite dans ce modèle.

- La numérotation n se fait progressivement
- Au départ tous les nœuds sont numérotés à 0
- mx est le maximum des numéros alloués
- un numéro alloué le reste (sauf 0)
- l'objectif est que tous les numéros entre 0 ou 1 et mx soient alloués.
- 0 ou 1 est le minimum des numéros alloués.

5.5.1 Variables et invariant pour le second modèle

Les variables sont variables nb, n and mx :

$$
\begin{array}{|l}
\hline
mx \in 0..m \\
\\
n \in X \to 0..mx \\
\\
n \in X \twoheadrightarrow min(\mathsf{ran}(n))..mx \\
\\
min(\mathsf{ran}(n)) = 0 \ \lor \ min(\mathsf{ran}(n)) = 1 \\
\hline
\end{array}
$$

Nous avons prouvé les théorèmes suivants :

$$
\begin{array}{|l}
\hline
\forall(n, s, f, a, b) \cdot \left(\begin{array}{l} s \subseteq X \\ n \in \mathbb{N} \\ n = card(s) \\ f \in s \twoheadrightarrow a..b \\ a \leq b \\ \implies \\ card(s) \geq b - a + 1 \end{array} \right) \\
\\
\exists(x, y) \cdot \left(\begin{array}{l} x \in X \\ y \in X \\ x \neq y \\ n(x) = n(y) \end{array} \right) \implies mx < m \\
\hline
\end{array}
$$

5.5.2 Les événements du second modèle

On va introduire deux nouveaux événements renommage et renommage0 qui sont des abstractions de la règle associée. Le nouveau numéro est toujours choisi entre le numéro courant (strictement) et $mx + 1$. Comme notre invariant dit que mx doit toujours être plus petit que m il faudra absolument que mx soit strictement plus petit que m pour que $mx + 1$ soit plus petit ou égal à m. Si un nœud x à le numéro 0 on est sûr que mx est strictement plus petit que m car il n'existe pas (il faut le montrer) de surjection de X vers $0..m$ car le cardinal de X est égal à m. renommage0 explique comment un nœud x dont le numéro est 0 peut changer. En revanche, si un nœud à un numéro différent de 0 alors il pourra changer de numéro seulement s'il *sait* qu'il existe un autre nœud qui a le même numéro que le sien.

renaming0 $\widehat{=}$
ANY x, k WHERE
$x \in X$
$n(x) = 0$
$k \in n(x) + 1..mx + 1$
THEN
$n(x) := k$
$mx := max(mx, k)$
END

renaming $\widehat{=}$
ANY x, y, k WHERE
$x \in X$
$y \in X$
$x \neq y$
$n(x) = n(y)$
$k \in n(x) + 1..mx + 1$
THEN
$n(x) := k$
$mx := max(mx, k)$
END

5.6 Troisième modèle : introduction du graphe

L'objectif de cette étape de raffinement est l'introduction des arêtes du graphe. Nous avons les éléments suivants :

- Un nœud aura la connaissance de ses voisins.
- Si on ne considère pas le numéro 0, alors la numérotation n restreinte à une boule (un nœud et ses voisins) est une injection.

5.6.1 Modéliser un graphe non orienté non-réflexif à l'aide de propriétés

$$\mathsf{graph}\,(g) \;\widehat{=}\; \left(\begin{array}{lll} g \in X \leftrightarrow X & \wedge \\ g = g^{-1} & \wedge \\ g \cap \mathsf{id}(X) = \varnothing & \wedge \end{array}\right)$$

5.6.2 Les événements

Ce modèle n'introduit pas de nouveaux événements. Il rend les événements renommage et renommage0 plus déterministes : on renforce les gardes.

renaming $\widehat{=}$
ANY x, y, k WHERE
$x \in X$
$y \in X$
$x \neq y$
$n(x) = n(y)$
$k \in n(x) + 1..mx + 1$
$k \notin n[g[\{x\}]]$
$k \notin n[g[g[\{x\}]]]$
THEN
$n(x) := k$
$mx := max(mx, k)$
END

renaming0 $\widehat{=}$
ANY x, k WHERE
$x \in X$
$n(x) = 0$
$k \in n(x) + 1..mx + 1$
$k \notin n[g[\{x\}]]$
$k \notin n[g[g[\{x\}]]]$
THEN
$n(x) := k$
$mx := max(mx, k)$
END

5.6.3 Invariants et théorèmes dérivés

Les deux invariants suivants précisent que deux noeuds différents d'une boule dont les numéros sont identiques ne peuvent avoir que le numéro 0.

$$\forall(x,y)\cdot\left(\begin{array}{l} x\mapsto y\in g \\ n(x)=n(y) \\ \Longrightarrow \\ n(x)=0 \end{array}\right)$$

$$\forall(x,y,z)\cdot\left(\begin{array}{l} x\in g[\{z\}] \\ y\in g[\{z\}] \\ n(x)=n(y) \\ x\neq y \\ \Longrightarrow \\ n(x)=0 \end{array}\right)$$

Le thèorème suivant, déduit des invariants précédents, dit que si on ne considère pas le numéro 0, alors la numérotation n restreinte à une boule $B(x)$ (un nœud x et ses voisins) est une injection.

$$\forall x\cdot(x\in X \implies B(x)\lhd n\rhd\{0\}\in X\rightarrowtail 1..mx)$$

5.7 Quatrième modèle : introduction de la variable N

Ce modèle introduit la variable N de l'algorithme de Mazurkiewicz avec les éléments suivants :

- $N(x)$ est l'ensemble des numéros ($\neq 0$) des voisins de x
- C'est le début de la localisation pour la distribution de l'algorithme
- Un nœud regardera ses voisins et les voisins des voisins pour décider du nouveau numéro (semi locale, semi globale)
- L'initialisation $N := X\times\{\varnothing\}$

renaming $\widehat{=}$
ANY x,y,k,Nx WHERE
$x\in X$
$y\in X$
$x\neq y$
$n(x)=n(y)$
$k\in n(x)+1..mx+1$
$k\notin N(x)$
$k\notin union(N[g[\{x\}]])$
$Nx\in g[\{x\}]\rightarrow\mathbb{P}(1..mx)$
$\forall z\cdot(z\in g[\{x\}] \implies Nx(z)=N(z)-\{n(x)\}\cup\{k\})$
THEN
$n(x):=k$
$mx:=max(mx,k)$
$N:=N\lhd\!\!+Nx$
END

renaming0 $\widehat{=}$
ANY x, k, Nx WHERE
$x \in X$
$n(x) = 0$
$k \in n(x) + 1..mx + 1$
$k \notin N(x)$
$k \notin union(N[g[\{x\}]])$
$Nx \in g[\{x\}] \rightarrow \mathbb{P}(1..mx)$
$\forall z \cdot (z \in g[\{x\}] \implies Nx(z) = N(z) - \{n(x)\} \cup \{k\})$
THEN
$n(x) := k$
$mx := max(mx, k)$
$N := N \lhd\!\!+ Nx$
END

La nouvelle variable locale Nx aux deux événements définit la nouvelle valeur de la variable N sur la boule de centre x.

L'invariant $\forall x \cdot (x \in X \implies N(x) = n[g[\{x\}] - \{0\})$ nous permet de remplacer les gardes $k \notin n[g[\{x\}]] \wedge k \notin n[g[g[\{x\}]]]$ par $k \notin N(x) \wedge k \notin union(N[g[\{x\}]])$ car k n'est pas égal à 0.

5.8 Cinquième modèle : introduction de la variable M de l'algorithme

Ce modèle introduit la variable M de l'algorithme de Mazurkiewicz avec les éléments suivants :

- M est un ensemble qui contient des éléments de la forme $\mathbb{N} \times \mathbb{P}(\mathbb{N}_1)$ qui représente des numéros d'un nœud ainsi que les numéros ($\neq 0$) de ses voisins obtenus au cours d'une exécution.
- Ces éléments sont diffusés aux voisins (et ainsi de suite ...). On introduit donc dans ce modèle la règle de diffusion.
- D'autres éléments sont ajoutés aux mémoires dès qu'un nœud change son numéro.
- L'événement renaming s'en sert pour savoir si deux nœuds différents ont le même numéro.
- Les événements renommage et renommage0 s'en servent pour calculer le nouveau numéro de x.
- Ce modèle "implémente" exactement les deux règles de l'algorithme. Chaque événement (sauf enum) prend ses décisions sur une boule.

5.8.1 Les variables et invariants

Les variables nb, n, N restent dans ce raffinement. La variable mx disparait ; elle contenait une valeur globale ; M est une nouvelle variable de ce modèle. L'invariant de typage est $M \in X \rightarrow (0..mx \leftrightarrow \mathbb{P}(1..mx))$; l'initialiation de cette nouvelle variable est $M := X \times \{\varnothing\}$.

La clé de l'algorithme est de montrer que si un nœud a dans sa mémoire un couple $n(x) \mapsto Nz$ tel que Nz est plus grand que $N(x)$, alors il existe un autre nœud que x qui a le même numéro que x. Cette propriété ne peut pas être définie comme cela ; en revanche, nous avons pu définir et prouver l'invariant capital suivant :

$$\forall(x, Nz, k) \cdot \left(\begin{array}{l} k \geq 1 \\ k \mapsto Nz \in M(x) \\ \implies \\ \exists y \cdot \left(\begin{array}{l} y \in X \\ n(y) = k \\ \left(\begin{array}{l} Nz \neq N(y) \\ \implies \\ \mathsf{maxdif}(Nz, N(y)) \in N(y) \end{array} \right) \end{array} \right) \end{array} \right)$$

et l'invariant suivant :

$$\forall x \cdot (x \in X \implies N(x) \subseteq dom(M(x)))$$

Pour faire la preuve de l'invariant capital nous avons prouvé le théorème suivant :

$$\forall (a,b,c) \cdot \left(\begin{array}{l} a \subseteq 0..mx+1 \\ b \subseteq 0..mx+1 \\ c \subseteq 0..mx+1 \\ a \neq b \\ b \neq c \\ \mathsf{maxdif}(a,b) \in b \\ \mathsf{maxdif}(b,c) \in c \\ \implies \\ \mathsf{maxdif}(a,c) \in c \end{array} \right)$$

qui montre que $maxdif$ est transitive avec la définition suivante :

$$\mathsf{maxdif}(a,b) \ \widehat{=} \ max(a-b \cup b-a)$$

renaming $\widehat{=}$
ANY x, k, Nz, Nx WHERE
$x \in X$
$n(x) \neq 0$
$\forall z \cdot (z \in B(x) \implies M(x) = M(z))$
$n(x) \mapsto Nz \in M(x)$
$\mathsf{maxdif}(Nz, N(x)) \in Nz$
$k - 1 = max(\mathsf{dom}(M(x)))$
$Nx \in g[\{x\}] \rightarrow \mathbb{P}(1..mx)$
$\forall z \cdot (z \in g[\{x\}] \implies Nx(z) = N(z) - \{n(x)\} \cup \{k\})$
THEN
$n(x) := k$
$N := N \lhd\!+ Nx$
$M := M \lhd\!+ B(x) \times \{M(x) \cup \{k \mapsto N(x)\} \cup NewS(x)\}$
END

où $NewS(x) \ \widehat{=} \ \bigcup(z).(z \in g[\{x\}] | \{n(z) \mapsto Nx(z)\})$

L'événement **renaming0** est le suivant. Le lecteur remarquera que la définition de k est un peu différente : $k - 1 = max(\mathsf{dom}(M(x)) \cup \{0\})$. En effet $M(x)$ peut être vide est donc $max(\mathsf{dom}(M(x)))$ peut ne pas être défini[1]

renaming0 $\widehat{=}$
ANY x, k, Nx WHERE
$x \in X$
$n(x) = 0$
$\forall z \cdot (z \in B(x) \implies M(x) = M(z))$
$k - 1 = max(\mathsf{dom}(M(x)) \cup \{0\})$
$Nx \in g[\{x\}] \rightarrow \mathbb{P}(1..mx)$
$\forall z \cdot (z \in g[\{x\}] \implies Nx(z) = N(z) - \{n(x)\} \cup \{k\})$
THEN
$n(x) := k$
$N := N \lhd\!+ Nx$
$M := M \lhd\!+ B(x) \times \{M(x) \cup \{k \mapsto N(x)\} \cup NewS(x)\}$
END

où $NewS(x) \ \widehat{=} \ \bigcup(z).(z \in g[\{x\}] | \{n(z) \mapsto Nx(z)\})$

[1] Merci à Rodin qui vérifie que chaque expression est bien définie. Pour $max(s)$ Rodin génère des obligations de preuves qui vont assurer que s est non vide, inclus dans les entiers et qu'un majorant de s existe.

5.8.2 L'événement diffusion

Lorsque les mémoires de deux nœuds d'une même boule sont différentes, alors toutes les mémoires de cette boule se synchronisent en prenant l'union des mémoires des nœuds de cette boule.

diffusion $\widehat{=}$
ANY $x, x1, x2$ WHERE
$x \in X$
$x1 \in B(x)$
$x1 \in B(x)$
$M(x1) \neq M(x2)$
THEN
$M := M \lhd\!\!\!\!- B(x) \times \{\bigcup(z).(z \in B(x) | M(z))\}$
END

5.9 Sixième modèle

- L'algorithme de Mazurkiewicz termine correctement lorsque le graphe est minimal et connexe
- La minimalité est définie par rapport à la notion de revêtement
- Un revêtement est un homomorphisme surjectif qui conserve la structure d'arêtes.
- Un revêtement est surjectif
- Un revêtement est localement bijectif (sur une boule)
- Pour un graphe minimal les seuls revêtements sont des bijections (graphe équivalent)

La propriété suivante définit un graphe non orienté connexe.

$$\text{conected}\,(g) \;\widehat{=}\; \left(\begin{array}{l} graph(g) \;\wedge \\ \forall S \cdot (\\ \quad S \subseteq X \;\; \wedge \\ \quad S \neq \varnothing \;\; \wedge \\ \quad g\,[S] \subseteq S \\ \quad \Longrightarrow \\ \quad X \subseteq S\,) \end{array}\right)$$

La propriété suivante définit un graphe (X, g) minimal pour les revêtements :

$$\forall (h, s, \phi) \cdot \left(\begin{array}{l} s \subseteq \mathbb{N} \\ h \in s \leftrightarrow s \\ \phi \in X \twoheadrightarrow s \\ \forall (x, y) \cdot \left(\begin{array}{l} x \mapsto y \in g \\ \Longrightarrow \\ \phi(x) \mapsto \phi(y) \in h \end{array}\right) \\ \forall x \cdot \left(\begin{array}{l} x \in X \\ \Longrightarrow \\ (B(x) \lhd \phi) \in (B(x) \rightarrowtail\!\!\!\!\twoheadrightarrow B_h(\phi(x))) \end{array}\right) \\ \Longrightarrow \\ \phi \in X \rightarrowtail\!\!\!\!\twoheadrightarrow s \end{array}\right)$$

On peut aussi éviter d'avoir la surjection en hypothèse et donc ne déduire que l'injectivité en conclusion.

$$\forall(h,s,\phi)\cdot\left(\begin{array}{l} s\subseteq\mathbb{N}\\ h\in \mathsf{ran}(phi)\leftrightarrow\mathsf{ran}(phi)\\ \phi\in X\rightarrow s\\ \forall(x,y)\cdot\left(\begin{array}{l} x\mapsto y\in g\\ \Longrightarrow\\ \phi(x)\mapsto\phi(y)\in h\end{array}\right)\\ \forall x\cdot\left(\begin{array}{l} x\in X\\ \Longrightarrow\\ B(x)\lhd\phi\in B(x)\rightarrowtail\!\!\!\!\!\rightarrow B_h(\phi(x))\end{array}\right)\\ \Longrightarrow\\ \phi\in X\rightarrowtail s\end{array}\right)$$

Les deux définitions ont été prouvées équivalentes.

5.9.1 L'événement enum

enum $\widehat{=}$
WHEN
$\forall(x,z)\cdot(x\in X\ \wedge\ z\in B(x)\ \Longrightarrow\ M(x)=M(z))$
$\forall x\cdot(x\in X\ \Longrightarrow\ n(x)\neq 0)$
$$\forall(x,Nz)\cdot\left(\begin{array}{l} x\in X\\ n(x)\mapsto Nz\in M(x)\\ Nz\neq N(x)\\ \Longrightarrow\\ \mathsf{maxdif}(Nz,N(x))\in N(x)\end{array}\right)$$
THEN
$nb:=n$
END

L'événement enum abstrait

enum $\widehat{=}$
WHEN
$m\in\mathsf{ran}(n)$
THEN
$nb:=n$
END

La preuve de raffinement est donc

$$\begin{array}{l}\forall(x,z)\cdot\left(\begin{array}{l} x\in X\ \wedge\ z\in B(x)\\ \Longrightarrow\\ M(x)=M(z)\end{array}\right)\\ \forall x\cdot(x\in X\ \Longrightarrow\ n(x)\neq 0)\\ \forall(x,Nz)\cdot\left(\begin{array}{l} x\in X\\ n(x)\mapsto Nz\in M(x)\\ Nz\neq N(x)\\ \Longrightarrow\\ \mathsf{maxdif}(Nz,N(x))\in N(x)\end{array}\right)\\ \Longrightarrow\\ m\in\mathsf{ran}(n)\end{array}$$

Une idée pour la preuve de raffinement (gardes) est la suivante. On instancie le quantificateur universel qui définit la minimalité d'un graphe avec $\bigcup(x).(x\in X|\{n(x)\}\times N(x))$, $1..mx$ et n. On en déduira la bijection de n sur $1..mx$ et on doit prouver (par récurrence) que si l'on a une bijection de X vers $1..mx$ où m est le cardinal de X alors on a $mx=m$ et donc m est dans le codomaine de n. La partie délicate de la preuve est de montrer la bijection locale de n sur une boule. Pour cela, on montre que $\bigcup(x).(x\in X|\{n(x)\}\times N(x))[\{n(x)\}]=N(x)$ car, comme le graphe est connexe, on peut en déduire que tous les nœuds ont le même M car chaque nœud a le même M que ses voisins.

Nous savons aussi que chaque $M(x)$ contient le couple $n(x) \mapsto N(x)$ (dernier invariant) et $N(x)$ est le plus grand Nz tel que $n(x) \mapsto Nz \in M(x)$ (garde de l'événement enum).

$$\left(\begin{array}{l} 1..mx \subseteq \mathbb{N} \\ \bigcup(x).(x \in X | \{n(x)\} \times N(x)) \in 1..mx \leftrightarrow 1..mx \\ n \in X \twoheadrightarrow 1..mx \\ \forall(x,y) \cdot \left(\begin{array}{l} x \mapsto y \in g \\ \implies \\ n(x) \mapsto n(y) \in \bigcup(x).(x \in X | \{n(x)\} \times N(x)) \end{array}\right) \\ \forall x \cdot \left(\begin{array}{l} x \in X \\ \implies \\ B(x) \lhd n \in B(x) \rightarrowtail B_{\bigcup(x).(x \in X | \{n(x)\} \times N(x))}(n(x)) \end{array}\right) \\ \implies \\ n \in X \rightarrowtail 1..mx \end{array}\right)$$

Un dernier invariant :

$$\forall x \cdot (x \in X \ \wedge \ N(x) \neq \varnothing \ \implies \ n(x) \mapsto N(x) \subseteq M(x))$$

5.10 Preuve de non blocage

Pour l'ensemble des raffinements présentés, nous n'avons pas prouvé que le système concret ne se bloque pas plus que son abstraction. Ces preuves sont "dérivables" mais elles ne sont pas "héritées" par les raffinements ultérieurs. On peut donc repousser ces preuves jusqu'au dernier modèle. En effet le dernier modèle ne se bloque pas car la garde de l'événement enum est, par construction, la négation de la disjonction des gardes des autres événements.

5.11 Une optimisation de l'algorithme

Nous avons modifié le cinquième modèle (introduction de la variable M) pour ne garder dans M que l'ensemble de numéros le plus grand (au sens de maxdif). M est une fonction qui contient des éléments de la forme $\mathbb{N}_1 \times \mathbb{P}(\mathbb{N}_1)$ qui représente des numéros d'un nœud ainsi que les numéros ($\neq 0$) de ses voisins obtenu au cours d'une exécution. L'invariant qui définit M est le suivant :

$M \in X \rightarrow (1..mx \nrightarrow \mathbb{P}(1..mx))$

Les événements renaming et renaming0 doivent choisir pour chaque voisin z de x un ensemble, pour l'associer à $mb(z)$. Il n'y a, au pire que deux candidats, qui sont $M(x)(mb(z))$ et $Nx(z)$. Si $M(x)$ est vide, il n'y a que $Nx(z)$. Comme $Nx(z)$ contient k, le plus grand des deux est $Nx(z)$.

renaming $\widehat{=}$
ANY $x, k, NzNx$ WHERE
$x \in X$
$n(x) \neq 0$
$\forall z \cdot (z \in B(x) \implies M(x) = M(z))$
$n(x) \mapsto Nz \in M(x)$
$N(x) \neq Nz$
$k - 1 = max(\mathsf{dom}(M(x)))$
$Nx \in g[\{x\}] \rightarrow \mathbb{P}(1..mx)$
$\forall z \cdot (z \in g[\{x\}] \implies Nx(z) = N(z) - \{n(x)\} \cup \{k\})$
THEN
$n(x) := k$
$N := N \lhd\!\!+ Nx$
$M := M \lhd\!\!+ B(x) \times \{M(x) \lhd\!\!+ NS(x) \cup \{k \mapsto N(x)\}\}$
END

où $NS(x) \mathrel{\widehat{=}} \bigcup(z).(z \in g[\{x\}] \land mb(z) \neq 0 | \{mb(z) \mapsto Nx(z)\})$
L'invariant suivant permet de montrer le raffinement de la garde :

$$\forall(x, Nz) \cdot \left(\begin{array}{l} n(x) \mapsto Nz \in M(x) \\ Nz \neq N(y) \\ \implies \\ \mathsf{maxdif}(Nz, N(y)) \in Nz \end{array} \right)$$

renaming0 $\mathrel{\widehat{=}}$
ANY x, k, Nx WHERE
$x \in X$
$n(x) = 0$
$\forall z \cdot (z \in B(x) \implies M(x) = M(z))$
$k - 1 = max(\mathsf{dom}(M(x)) \cup \{0\})$
$Nx \in g[\{x\}] \to \mathbb{P}(1..mx)$
$\forall z \cdot (z \in g[\{x\}] \implies Nx(z) = N(z) - \{n(x)\} \cup \{k\})$
THEN
$n(x) := k$
$N := N \lhd Nx$
$M := M \lhd B(x) \times \{M(x) \lhd NS(x) \cup \{k \mapsto N(x)\}\}$
END

Pour l'événement **diffusion**, il faut prendre le maximum pour tous les candidats :

diffusion $\mathrel{\widehat{=}}$
ANY $x, x1, x2$ WHERE
$x \in X$
$x1 \in B(x)$
$x1 \in B(x)$
$M(x1) \neq M(x2)$
THEN
$M := M \lhd B(x) \times \{NB(x)\}$
END

$NB(x) \mathrel{\widehat{=}} \bigcup(k).(k \in \mathsf{dom}(\mathsf{union}(M[B(x)]) | \{k \mapsto MD(\mathsf{union}(M[B(x)])[\{k\}])\})\})$
où MD est défini de la manière suivante :

$$MD \in \mathbb{P}_1(\mathbb{P}(1..m)) \to \mathbb{P}(1..m)$$

$$\forall P \cdot (P \in \mathbb{P}_1(\mathbb{P}(1..m)) \implies MD(P) \in P$$

$$\forall(P, Q) \cdot \left(\begin{array}{l} P \in \mathbb{P}_1(\mathbb{P}(1..m)) \\ Q \in P \\ Q \neq MP(P) \\ \implies \\ \mathsf{maxdif}(MD(P), Q) \in MD(P) \end{array} \right)$$

Un autre invariant a été prouvé. Le domaine de $M(x)$ est un intervalle de borne inférieure égale à 1 (si $M(x)$ est non vide).

$$\forall(x, j, k) \cdot \left(\begin{array}{l} x \in X \\ j \in \mathsf{dom}(M(x)) \\ k \in 1..j \\ \implies \\ k \in \mathsf{dom}(M(x)) \end{array} \right)$$

5.12 Conclusion

Cette étude de cas a été proposée par le LABRI et constituait un challenge pour la méthode incrémentale Event B. Nous avons conclu les points suivants :

- Une preuve validée par les outils Click'n'Prove et RODIN de l'algorithme de Mazurkiewicz a été produite ; malgré la grande difficulté inhérente à cet algorithme et aux structures associées, nous avons pu mener à terme une démonstration complètement formelle et mécanisée ; cela est possible grâce au mécanisme de raffinement qui permet d'expliquer l'algorithme et sa preuve.
- Le premier raffinement exprime un comportement abstrait de la règle de renommage et la surjectivité de n
- Le deuxième raffinement apporte l'injection locale (important pour la bijection locale).
- Un effort de deux semaines à temps plein (71 POs + 45 interactives avec Click'n'Prove) a été nécessaire pour la partie technique de preuve.
- Nous avons obtenu une optimisation de la variable M (avec Rodin)

Cet exemple est très encourageant et nous avons poursuivi le développement d'autres algorithmes proposés par le LABRI :

- l'algorithme de détection de terminaison de Szymanski, Shi et Prywes,
- les algorithmes d'élection dans un k-arbre.

Chapitre 6

Conclusion

Les études de cas ont été réalisées pour plusieurs raisons. D'une part, elles ont permis de montrer comment de telles études de cas pouvaient être développées en Event B ; elles ont donc été formatives pour l'ensemble du projet. D'autres algorithmes répartis sont en cours de développement et font l'objet de rapports techniques intermédiaires ; un autre aspect de ce travail est la constitution d'archives RODIN de ces développements. En effet, le développement d'une étude de cas est réalisé dans l'outil RODIN qui permet de créer des archives conservant les sources des modèles mais aussi les preuves qiuand celles-ci ont été réalisées.

D'autre part, les études de cas sont importantes pour l'identification de patrons prouvés, qui sont les objets à mettre en évidence dans notre projet. On peut ainsi retenir que l'étude de cas du Mondex est certainement intéressante car elle introduit des systèmes à base de transactions et les travaux se poursuivent, afin de rejouer de telles études notamment, en intégrant les aspects cryptographiques. Le développement du IEEE avec une analyse des propriétés témoigne de l'intérêt d'une telle méthodologie qui permet la réutilisation et la découverte de nouvelles solutions. Enfin, ces études de cas ont permis d'acquérir un vocabulaire commun au niveau des systèmes répartis et de jeter les bases d'outils qui sont en cours de développement et qui vont faciliter la validation des algorithmes répartis.

Nous n'avons pa couvert les questions liées au temps et aux probabilités que l'on trouve généralement dans les algorithmes classiques notamment le IEEE 1394. Ces questions feront l'objet d'études complémentaires. Nous n'avons pas rapporté notre travail réalisé dans le cadre du e-voting [17, 18] préparatoire à un travail de la tâche sur les patrons de développement. Enfin, un travail sur les algorithmes séquentiels [20] servira aussi pour proposer des patrons de développement prouvé.

Bibliographie

[1] The mondex electronic purse system. http ://www.mondex.com.

[2] J.-R. Abrial. Formal construction of proved circuits. Internal Note Version 4(Août 2001), Consultant, août 2001.

[3] J.-R. Abrial. B# : Toward a synthesis between z and b. In D. Bert and M. Walden, editors, *3nd International Conference of B and Z Users - ZB 2003, Turku, Finland*, Lectures Notes in Computer Science. Springer, June 2003.

[4] J.-R. Abrial and D. Cansell. Click'n prove : Interactive proofs within set theory. In *TPHOLs [13]*, pages 1–24, 2003.

[5] J.-R. Abrial, D. Cansell, and D. Méry. A mechanically proved and incremental development of the IEEE 1394 Tree Identify Protocol . Technical report, LORIA, March 2001.

[6] J.-R. Abrial, D. Cansell, and D. Méry. The Complete B Project for the Development of the IEEE 1394 Tree Identify Protocol . http ://www.loria.fr/ mery/ieee1394, March 2001.

[7] J.-R. Abrial, D. Cansell, and D. Méry. A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. *Formal Aspects of Computing*, 14(3) :215–227, 2003.

[8] J.-R. Abrial and L. Mussat. Introducing dynamic constraints in B. In D. Bert, editor, *B'98 :Recent Advances in the Development and Use of the B Method*, volume 1393 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[9] Jean-Raymond Abrial. *The B book - Assigning Programs to Meanings*. 1996.

[10] Jean-Raymond Abrial, Dominique Cansell, and Dominique Méry. A mechanically proved and incremental development of ieee 1394 tree identify protocol. *Formal Asp. Comput.*, 14(3) :215–227, 2003.

[11] D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th Symposium on theory of computing*, pages 82–93, 1980.

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

[13] David A. Basin and Burkhart Wolff, editors. *Theorem Proving in Higher Order Logics, 16th International Conference, TPHOLs 2003, Rom, Italy, September 8-12, 2003, Proceedings*, volume 2758 of *Lecture Notes in Computer Science*. Springer, 2003.

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

[15] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. ADDISON-WESLEY, 1999.

[16] M. Butler. Mondex in Event B. Technical report, ECS, 2007.

[17] Dominique Cansell, Paul Gibson, and Dominique Méry. Formal verification of *tamper-evident* storage for e-voting. In *SEFM 2007 5th IEEE International Conference on Software Engineering and Formal Methods*, 2007.

[18] Dominique Cansell, Paul Gibson, and Dominique Méry. Refinement : A constructive approach to formal software design for a secure e-voting interface. *Electr. Notes Theor. Comput. Sci.*, 183, 2007.

[19] Dominique Cansell and Dominique Méry. *The event-B Modelling Method : Concepts and Case Studies*, pages 33–140. Springer, 2007. See [14].

[20] Dominique Cansell and Dominique Méry. Proved-patterns-based development for structured programs. In *CSR*, pages 104–114, 2007.

[21] CCITT. Recommendation z. 100 specification and description language sdl. Note, 1988.

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

[23] M. Devillers, D. Griffioen, J. Romin, and F. Vaandrager. Verification of a Leader Election Protocol : Formal Methods Applied to IEEE 1394. *Formal Methods in System Design*, 16 :307–320, 2000. Kluwer Academic Publishers.

[24] E.W. Dijkstra. *A discipline of programming*. Prentice-Hall International, 1976.

[25] E. Gamma, R. Helm, R. Johnson, R. Vlissides, and P. Gamma. *Design Patterns : Elements of Reusable Object-Oriented Software design Patterns*. Addison-Wesley Professional Computing, 1997.

[26] Tony Hoare. The verifying compiler : A grand challenge for computing research. *J. ACM*, 50(1) :63–69, 2003.

[27] Tony Hoare. The verifying compiler : A grand challenge for computing research. *J. ACM*, 50(1) :63–69, 2003.

[28] INRIA. *COQ Manual*.

[29] Cliff Jones, Peter O'Hearn, and Jim Woodcock. Verified software : A grand challenge. *Computer*, April 2006.

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

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

[32] N. Lynch. *An Introduction to Distributed Systems*. ??,??

[33] Zohar Manna, Nikolaj Bjørner, Anca Browne, Edward Y. Chang, Michael Colón, Luca de Alfaro, Harish Devarajan, Arjun Kapur, Jaejin Lee, Henny Sipma, and Tomás E. Uribe. Step : The stanford temporal prover. In Peter D. Mosses, Mogens Nielsen, and Michael I. Schwartzbach, editors, *TAPSOFT*, volume 915 of *Lecture Notes in Computer Science*, pages 793–794. Springer, 1995.

[34] Antoni W. Mazurkiewicz. Distributed enumeration. *Inf. Process. Lett.*, 61(5) :233–239, 1997.

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

[36] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[37] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[38] W. Reisig. *Petri Nets An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, 1985. ISNB 3-540-13723-8.

[39] Project RODIN. The rodin project : Rigorous open development environment for complex systems. http ://rodin-b-sharp.sourceforge.net/.

[40] Susan Stepney, David Cooper, and Jim Woodcock. An electronic purse : Specification, refinement, and proof. Technical monograph PRG-126, Oxford University Computing Laboratory, July 2000.

[41] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[42] J. Woodcock. Grand challenge 6 : Dependable systems evolution. http ://www.fmnet.info/gc6/, 2004.