THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Vincent Filou**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : **INFORMATIQUE**

Une étude formelle de la théorie des calculs locaux à l'aide de l'assistant de preuve Coq

---

Directeurs de thèse : **Mohamed MOSBAH** & **Pierre CASTÉRAN**

Numéro d'ordre : 4708
Soutenue le 21/12/2012
Devant la Commission d'Examen

JURY

Mme.	Sandrine Blazy	rapporteur
M.	Dominique Méry	rapporteur
M.	Yamine Ait Ameur	examinateur
M.	Yves Métivier	examinateur
M.	Pierre Castéran	Directeur de thèse
M.	Mohamed Mosbah	Directeur de thèse

Table des matières

Remerciements

Merci tout d'abord aux deux rapporteurs, Sandrine Blazy et Dominique Méry, pour leur remarques et leurs corrections. Merci aux membres de l'ANR Rimel, dont les travaux et les conseils sur le développement par raffinement m'ont servis aussi bien dans mes enseignements que dans mes travaux de recherche. Merci à Yves Bertot pour ses explications sur la proof irrelevance. Merci aux spécialistes des calculs locaux : Yves Métivier pour ses explications du lemme de relèvement et des revètements, et Jérémie Chalopin et Émmanuel Godard : grâce à nos discutions j'ai pu patcher la définition des calculs locaux. Merci à Allyx Fontaine et Akka Zemmari pour leur relecture, leur participation à mes répèts de soutenance et nos discussions du mercredi/vendredi après-midi. Merci a Nicolas Hannusse pour ses encouragements. Merci à mes directeurs de thèse, Pierre Castéran et Mohamed Mosbah, pour leur soutient et leur patience.

Merci à mes deux collègues de bureau : Loïc Martin et Mohamed Tounsi. Loïc a toujours été présent pour me donner un coup de main avec ViSiDIA, et est à l'origine de mon interet pour la composition. Les discussions avec Mohamed m'ont forcé à sortir la tête du guidon (Coq) et à envisager les choses sous d'autres angles. J'espère que nous aurons l'occasion de poursuivre ensemble nos travaux sur la composition. Merci aux CVTistes et à Zvebor pour m'avoir écouté ronchonner au déjeuner pendant quatres ans sur *Coq* et mes erreurs de formalisation.

Merci aux membres de ma famille pour leurs encouragements et leurs corrections. Enfin merci à Rémi et Alex, à Damien et Marion, à Clément et à David. Je vous dois beaucoup.

Chapitre 1

Introduction

La tendance actuelle des systèmes d'information est à la décentralisation : décentralisation des calculs et décentralisation du stockage des données [9, 65, 90, 87]. Cette tendance est soutenue théoriquement par le champ de l'algorithmique distribuée, dont l'objectif est de modéliser et d'analyser rigoureusement les algorithmes sur lesquels se fondent les systèmes modernes de décentralisation. L'étude de l'expressivité des modèles utilisés est aussi un des objectifs de ce champ d'étude. Les problèmes couramment rencontrés dans ce contexte peuvent selon nous se réduire aux trois questions suivantes : un algorithme distribué réalise-t-il la tâche pour laquelle il a été conçu ? Une tâche est-elle réalisable dans un certain modèle de calcul ? Une tâche est elle réalisée efficacement par un algorithme ? Dans ce document, nous nous intéresseront aux deux premiers problèmes.

Pour tenter de répondre le plus simplement possible à la première question et vérifier la correction d'algorithmes distribués, de nombreux modèles ont été proposés : parmi ceux-ci, on peut notamment citer la famille des *algèbres de processus* [71, 53, 88, 50], les *automates d'entrée-sortie* [60], le langage de modélisation *Unity* [27] et la logique *TLA* (Temporal Logic of Actions) [56]. Le modèle des calculs locaux, proposé par I. Litovsky, Y. Métivier et E. Sopena [59] semble le plus adéquat pour répondre aux deux questions posées ci-dessus : il permet de prouver la correction d'un algorithme à un niveau d'abstraction élevé, sans s'encombrer des détails de communication, tout en autorisant d'élégants méta-raisonnements permettant d'étudier l'expressivité de classes d'algorithmes. Dans ce modèle, descendant des travaux de Mazurkiewicz [67], un réseau est représenté par un graphe étiqueté : chaque processeur est représenté par un sommet, chaque lien de communication directe entre processeurs correspond à une arête. L'état de chaque processeur et lien de communication est représenté par une étiquette. Un *système de calculs locaux* est une relation de réétiquetage de graphe dont l'application est centrée sur un sommet. Contrairement aux algèbres de processus et aux automates d'entrée-sortie ou à *Unity*, ce modèle distingue clairement la topologie d'un réseau et le calcul réalisé. Ceci permet d'exprimer naturellement les questions de la forme : "peut-on élire un leader dans n'importe quel graphe ?", et d'y répondre à l'aide de techniques de preuves — héritées des travaux de D. Angluin [10] — se basant sur les morphismes de graphes.

Les formalismes cités ci-dessus, pourvus d'une sémantique clairement définie, permettent de *simuler* des exécutions d'algorithmes, et de *mécaniser* voire *d'automatiser* les preuves de ces derniers. L'algèbre de processus *CCS* [71] dispose par exemple d'un environnement : l'Edinburgh Concurrency Workbench [74], permettant de simuler les processus, d'étudier les relations d'équivalence entre ces derniers et d'explorer de manière exhaustive leurs exécutions (par model checking). M. Charpentier a proposé, dans le cadre de sa thèse de doctorat [28], un prototype d'assistant de preuve pour le formalisme *Unity* permettant de vérifier dynamiquement ou automatiquement la cohérence d'un programme avec sa spécification. Les automates d'entrée-sortie disposent également d'un environnement de preuve, *TAME* [11, 12, 72], basé sur l'assistant de preuve *PVS* [43]. Les calculs locaux bénéficient quant à eux de l'environnement de visualisation et de simulation *ViSiDiA* [94]. Plus récemment, les travaux de M. Tounsi [92, 73] ont permis d'adapter la méthode *B-événementielle* [5, 6] pour produire des algorithmes distribués certifiés et exécutables au sein de l'environnement *ViSiDiA*.

À notre connaissance, ces outils permettent uniquement les preuves de correction d'algorithmes. Nous proposons ici un prototype d'environnement qui ajoute aux preuves de correction de calculs locaux des raisonnements sur l'expressivité de ces derniers. Des passerelles sont établies entre ces deux niveaux de raisonnement. Appliquer les métathéorèmes concernant la théorie des calculs locaux à des algorithmes ou réseaux nécessite de considérer tous ces objets comme des "citoyens" de premier ordre, sur lesquels une quantification est possible. Nous nous basons donc sur un assistant de preuve dont la logique est suffisamment expressive pour nous permettre de définir les algorithmes comme des objets de premier ordre : l'assistant de preuve *Coq* [91, 16].

Coq, comme d'autres assistants de preuves (NuPRL [83], Lego [82]) est fondé sur la correspondance de *Curry-Howard* : une preuve est une séquence finie d'applications de règles logiques, et peut donc être représentée par un programme. Une preuve est représentée en *Coq* par un terme du *Calcul des Constructions Inductives* (ou *CIC*) [36, 81], un *lambda calcul* dont le système de types permet d'exprimer une logique d'ordre supérieur. L'utilisateur construira en général un terme de preuve à l'aide d'une succession d'applications de *tactiques*. L'implantation proposée étant un prototype, nous ne détaillerons pas dans ce mémoire les tactiques et autres outils spécifiques à *Coq* utilisés au cours de notre étude. Nous exposerons plutôt les *techniques de preuves* employées.

1.1 Formalisation et Vérification du Distribué : un bref État de l'Art

Nous effectuons ici un rapide tour d'horizon des modèles de l'algorithmique distribuée classiquement utilisés dans la litérature. Pour chacun nous décrivons les environnements mis en place pour mécaniser les raisonnements. Nous les comparons à notre propre proposition.

1.1.1 Automates d'entrée-sortie

N. Lynch *et al.* ont proposé, pour étudier la correction des algorithmes distribués asynchrones, le modèle des automates d'entrée-sortie [64]. Les automates d'entrée-sortie sont conçus principalement pour représenter les systèmes réactifs ou *événementiels*, i.e. , des systèmes dont l'objectif est de traiter des signaux issus de l'environnement.

Un réseau, dans le modèle de N. Lynch, est le résultat de la composition d'automates communiquant par signaux. Un automate d'entrée-sortie peut être représenté comme un système de transitions. Ces transitions peuvent être de trois types : les *transitions internes*, correspondant aux calculs effectués par un processeur, les *transitions de sortie*, ou le processeur émet un signal, et les *transitions d'entrée*, où le processeur reçoit un signal. Contrairement au modèle des calculs locaux, la distinction entre topologie du réseau et algorithme n'est pas explicite. L'alternance entre émission/réception de signaux et traitements internes oblige de plus la personne raisonnant sur le système à considérer une classe d'exécutions spécifiques où les traitements et la communication alternent de manière équitable (fairness).

Parmi les efforts de mécanisation des preuves sur les automates d'entrée-sortie, les travaux d'Archer [13, 11, 12, 72] ainsi que ceux de O. Muller [77] sont les plus proches de nos propres efforts. Archer se base sur l'assistant de preuve *PVS* [43] pour mécaniser les raisonnements sur les automates d'entrée-sortie (et les modèles dérivés). Dans [11, 12], Archer décrit la façon dont l'interface à *PVS TAME* (Timed Automata Modelling Environnement) et les *stratégies de preuves* qui y sont associées permettent d'écrire des spécifications et des preuves d'automate entrée-sortie dans un style "naturel", proche d'une preuve manuelle. Dans cet environnement, un automate entrée-sortie est défini comme l'instance d'un patron générique. Des stratégies de preuves *PVS* (l'équivalent des *tactiques Coq*), représentant les formes courantes de preuves d'invariants (induction, application des pré-conditions d'actions, preuves d'invariants automatiques), sont fournies. L'utilisation des stratégies plus basiques de *PVS* est toutefois nécessaire lorsque le type des états des automates manipulés est complexe. Dans[72], *l'instantiation de théorie* (concept similaire aux *foncteurs Coq*) est utilisée pour représenter les diverses types de relations d'abstraction (simulation et raffinement) entre automates.

Les travaux d'Archer se concentrent sur l'aspect *preuves de systèmes*. Nous nous intéressons davantage aux propriétés génériques des systèmes distribués. Les contributions citées précédemment ne contiennent, par exemple, aucun plan pour permettre les preuves d'impossibilité avec *TAME*.

1.1.2 Algèbres de processus

Dans la lignée des travaux de Church [33] sur la calculabilité, une algèbre de processus est un langage "à la lambda calcul" modélisant les programmes parallèles et distribués. Les

deux principaux langages de cette famille sont les langages *Calculus of Communicating Systems* (*CCS*) [71] de R. Milner et le *Communicating Sequential Process* (*CSP*) [53] de C.A.R. Hoare. Plus proche du modèle des calculs locaux, le langage $D\pi$ [84, 85] basé sur les travaux de Milner et d'Henessy [51, 34, 52] ajoute la notion de "location" aux constructions des algèbres de processus. Une passerelle entre les algèbres de processus et la réécriture de graphe est établie par J.R.W. Glauert [44].

Cette famille de langages représente des processus, communiquant par le biais *d'actions*, composés à l'aide *d'opérateurs* : composition séquentielle, parallèle, choix non déterministe, etc. Les raisonnements sur les systèmes ainsi définis peuvent prendre plusieurs formes. Une relation de simulation entre un système simple et abstrait et une implantation plus complexe peut être définie, prouvant ainsi la correction de l'implantation [70]. Une autre voie est l'utilisation de systèmes de types, assurant que les processus bien typés respectent certaines propriétés : par exemple la préservation de données "secrètes" [66, 3].

Il existe aujourd'hui une variété d'implantation et d'outils de vérification pour ces modèles. Parmi ceux-ci nous retiendrons l'*Edinburgh Concurrency Workbench* [74], regroupant plusieurs fonctionnalités aidant à la vérification de systèmes décrits par *CCS* : simulation, vérification d'équivalence entre processus, model-checking, etc. On notera aussi l'outil *PAM* (Process Algebra Manipulator) [58] permettant à un utilisateur de définir une algèbre de processus et de raisonner sur les programmes y étant exprimés. *PAM* peut être décrit comme un assistant de preuve spécialisé dans les algèbres de processus, les preuves y étant construites de manière interactive. Il existe des études en *Coq* (voir les contributions de *Coq* [1]) de *CCS* et du π-*calcul*. La première, par S. Coupet-Grimal, est une étude de trois notions d'équivalence entre processus. La seconde, par I. Scagnetto est une vérification de la méta théorie de la bisimilarité forte.

1.1.3 Langage Unity

Unity [27] est un langage permettant de définir et de raisonner sur les programmes parallèles. Un programme est défini en *Unity* comme un ensemble de variables, une valuation initiale de ces variables et un ensemble d'affectations. Le parallélisme est représenté en supposant que chaque affectation d'un programme est exécutée sur un processeur. Pour raisonner sur un programme, des prédicats "à la Hoare" sont utilisés. Une fois un premier programme abstrait (non déterministe, pas de spécificité concernant l'environnement d'exécution) correct défini, un programme concret peut être dérivé de celui-ci par raffinement. La proposition originelle de K. Mani Chandy et J. Misra n'étant pas conçue pour représenter des systèmes distribués, M. Charpentier propose dans [29] une adaptation du formalisme dans ce but. Cette adaptation consiste en deux changements : l'ajout d'une relation *d'observation* aux programmes *Unity*, représentant le fait que la valeur d'une variable reflète, de manière décalée dans le temps, la valeur d'une variable stockée sur un processeur distant. Un opérateur dit *produit* de processus est défini, permettant de représenter la répartition d'un programme sur différents processeurs. Toutefois cette

adaptation n'atteint toujours pas selon nous le niveau de séparation entre topologie et programme offert par les systèmes de calculs locaux.

Parmi les tentatives de mécanisation de la théorie d'*Unity*, nous retiendrons la thèse de M. Charpentier [28] et les travaux de (entre autres) F. Andersen [7, 8]. Le premier, en plus d'étendre le langage avec des constructions facilitant la représentation de systèmes distribués, propose un assistant de preuve (DADA) des programmes *Unity*. Toutefois cet environnement a uniquement pour objectif de mécaniser les preuves de programmes. Les travaux de F. Andersen se basent sur une implantation de la théorie de *Unity* pour l'assistant de preuve *HOL*. Là encore, les théorèmes formalisés ne concernent pas l'expressivité du langage mais uniquement la preuve de programmes.

1.1.4 TLA/TLA+

TLA (Temporal Logic of Action) [55, 56] est une logique de spécification des systèmes parallèles et distribués définie par L. Lamport. Dérivées des logiques temporelles (*LTL*, *CTL*), une spécification *TLA* est en général composée de la conjonction d'un prédicat d'initialisation et d'une suite de prédicats "avant-après", représentant l'état du système avant et après l'exécution d'une action. La boîte à outil de *TLA* dispose, pour la vérification de spécifications, d'un "model checker" (TLC) et d'un prouveur (TLAPS) [30]. Il existe des implantations de *TLA* en Isabelle [54, 69].

1.2 Contributions et organisation du document

Nous introduisons tout d'abord le modèle des calculs locaux, ainsi que les définitions de théorie des graphes dont nous ferons usage dans le reste de ce document. Nous présentons notamment les définitions des notions de *tâche*, *mode de synchronisation* et *mode de détection de la terminaison*.

Le chapitre 3 présente notre première contribution : la définition en *Coq* d'une sémantique relationnelle des calculs locaux. Nous y proposons une sémantique pour chaque mode de synchronisation et montrons comment celles-ci peuvent être utilisées pour produire des preuves de systèmes de calculs locaux. Nous montrons en outre que certains systèmes de réétiquetage, correspondant à la définition intuitive d'un système localement engendré, échappent à la définition semi-formelle couramment trouvée dans la littérature, et proposons une nouvelle définition formelle. Nous expliquons la technique mise en œuvre pour prouver formellement que chaque mode de synchronisation étudié est capturé par cette nouvelle définition.

Dans le chapitre 4, nous raisonnons sur la sémantique des modes de synchronisations et montrons comment l'utiliser pour définir des *invariants de classes*. Nous prouvons ainsi formellement deux nouvelles propriétés des *calculs locaux sur les arêtes* (ou **LC0**) [23]

dont le principe est similaire au *lemme de pompage* de la théorie des automates finis. Nous utilisons ces résultats pour fournir de nouvelles preuves d'irréalisabilité par les systèmes **LC0** de quelques problèmes courants de l'algorithmique distribuée : le calcul des degrés de chaque sommet d'un graphe avec détection locale de la terminaison locale, l'élection et le calcul d'un arbre recouvrant avec détection de la terminaison globale. Nous suggérons une généralisation informelle de ces résultats, en proposant une nouvelle classe de graphe pour laquelle l'élection est irréalisable par un système **LC0**. Nous décrivons enfin une formalisation de la preuve d'impossibilité issue des travaux d'Angluin [10] : il n'existe pas de système de calculs locaux réalisant l'élection dans un graphe quelconque.

Dans le chapitre 5, nous proposons une adaptation de la définition de la *forward simulation* [61] aux systèmes de calculs locaux, et montrons comment celle-ci peut être utilisée pour simplifier les preuves de systèmes de calculs locaux. Nous utilisons la notion de *forward simulation* et un ensemble d'opérateurs sur les règles **LC0** pour démontrer la correction d'une transformation d'un système de calcul **LC0** avec *détection locale de la terminaison globale* en un système avec *détection observée de la terminaison*.

Dans le dernier chapitre, nous étudions, en collaboration avec M. Tounsi, la possibilité de composer les systèmes de calculs locaux. Nous proposons notamment une technique de preuve s'appuyant sur la commutativité des systèmes de réétiquetage.

1.3 Conventions lexicales

Un récapitulatif des systèmes dont la correction a été prouvée à l'aide de nos bibliothèques *Coq* est fourni en annexe. Le developpement *Coq* et l'implantation ViSiDIA des systèmes donnés en Annexe A sont disponibles à l'adresse : `http://www.labri.fr/perso/filou/These/`.

Dans les chapitres suivants le chapitre 2, chaque définition, lemme et théorème formalisé en *Coq* est accompagné d'une référence à l'annexe où est donnée un pointeur vers le fichiers de l'implantation contenant la version formelle. Un résultat non prouvé en *Coq* est explicitement noté par le sigle *(Informel)*.

Chapitre 2

Introduction à la théorie des Calculs Locaux

Dans ce chapitre nous présentons la théorie des calculs locaux ainsi que les notions nécessaires à sa compréhension. Nous rappelons les principaux travaux et résultats concernant ce modèle de calcul.

Sommaire

2.1 Introduction

Le modèle des calculs locaux, introduit par Litovsky, Métivier et Sopena [59], représente un réseau comme un graphe non orienté, simple, et généralement connexe. Un processeur participant au réseau est représenté par un sommet, et un lien de communication est représenté par une arête. À chaque sommet (resp. à chaque arête) est associée une étiquette, représentant l'état du processeur (resp. lien de communication). Un algorithme distribué est représenté par une *relation de réétiquetage*, décrivant le changement d'état des processeurs en fonction de leur voisinage. Nous considérons par la suite des topologies statiques, où seuls les états des processeurs et liens de communication changent. Toutefois, les techniques de preuves décrites sont adaptables au modèle de A. Casteigts *et al.* [20, 21, 19] pour étudier les réseaux dynamiques.

Contrairement à la plupart des autres modèles, les calculs locaux nous permettent de séparer deux aspects d'un système distribué : la topologie du réseau sur lequel l'algorithme est exécuté et le comportement de l'algorithme lui même. Le réseau est ainsi traité comme une *donnée initiale*. Sans s'y limiter, cette modélisation est particulièrement adaptée aux réseaux pair à pair, où un unique algorithme est exécuté sur l'ensemble des noeuds d'un réseau quelconque. Elle nous permet en plus de raisonner sur un problème en fonction de la forme du réseau sur lequel il doit être résolu [45] : peut-on, par exemple, élire dans un arbre ? Dans un graphe quelconque ? Quelles connaissances initiales sont nécessaires en fonction de la topologie considérée ?

Y. Métivier *et al.* proposent, pour définir formellement un problème sur un réseau, la notion de *tâche* [24, 47]. Une tâche spécifie un problème en définissant un ensemble d'*états initiaux* (ou *pré-condition*) et une relation entre états initiaux et finaux. Dans le cadre de l'algorithmique distribuée, où les connaissances ne peuvent être immédiatement partagées par deux sommets distants, définir la forme d'un résultat n'est qu'une spécification partielle : dans ce contexte, le problème de la détection de la terminaison devient fondamental. Comment signaler aux noeuds d'un réseau qu'un leader a été élu ? Qu'une table de routage peut être utilisée ? Y. Métivier *et al.* ont défini dans [47] un ensemble de réponses standards à ces questions, ou *modes de détection de la terminaison*. Ces modes décrivent quels acteurs du réseau possèdent la connaissance de la terminaison : par exemple, dans le mode de *détection locale de la terminaison locale*, chaque processeur sait si le résultat de ses calculs est susceptible ou non de changer à l'avenir. Dans le mode de *détection locale de la terminaison globale*, un processeur est averti de la terminaison de l'exécution de l'algorithme sur l'ensemble du réseau. Nous verrons par la suite (voir chapitre 6) que ces modes ont une importance particulière lorsque l'on considère la *composition* d'algorithmes.

À l'aide des notions décrites plus haut, J. Chalopin [23] a, entre autres, étudié la réalisabilité de diverses tâches. Ses travaux établissent que les réponses varient en fonction des *primitives de synchronisation* employées et de la topologie des réseaux. Il a ainsi défini une *hiérarchie* des modes de synchronisation. Au sommet de cette hiérarchie, nous retrouvons l'ensemble des calculs locaux, où calculs **LC2**, modifiant l'étiquetage d'un *graphe en étoile*. Au bas de cette hiérarchie se trouvent les *calculs locaux sur les arêtes*, ou **LC0**, dont les calculs affectent un couple de sommets adjacents. Entre ces deux extrêmes existe une variété de classes intermédiaires, dont nous retiendrons les calculs **LC1**.

Le reste de ce chapitre est organisé comme suit : nous commençons par définir les notions fondamentales de la théorie des graphes, nécéssaires à la compréhension des calculs locaux. Nous continuons par définir les relations, systèmes et règles de réétiquetage de graphes, ainsi que les concepts de tâche et de détection de terminaison présentés plus haut. Nous finissons par définir les trois principaux modes de synchronisation de la hiérarchie des calculs locaux.

2.2 Graphes et graphes étiquetés

Nous considérerons ici des graphes *simples* et non orientés. Soit G un graphe, on notera $V(G)$ (resp. $E(G)$) l'ensemble de ses sommets (resp. de ses arêtes). Les graphes seront représentés par la suite par une des lettres F, G, H, et leurs sommets par une des lettres u, v, w. On notera une arête comme un couple de sommets (u, v), u et v étant les *extrémités* de l'arête (u, v). Deux sommets u et v sont dits *adjacents* dans G si l'arête (u, v) appartient à l'ensemble $E(G)$. Nous considérons ici des graphes non orientés, i.e. des graphes dont la relation d'adjacence est symétrique. Une *boucle* est une arête dont les extrémités sont égales. Un graphe *simple* est un graphe ne contenant pas de boucle.

Le *voisinage* d'un sommet v dans un graphe G est noté $N_G(v)$ et est défini comme l'ensemble des sommets adjacents à v dans G. Le *degré* d'un sommet v dans un graphe G est le nombre de sommets adjacents à v i.e. $|N_G(v)|$. Un graphe H est dit un *sous-graphe* de G si tous les sommets (resp. arêtes) de H sont des sommets (resp. arêtes) de G. Le *sous-graphe engendré* par un ensemble de sommets X d'un graphe G est le sous-graphe de G contenant X et les arêtes de G dont les deux extrémités appartiennent à X.

Un *chemin* d'un graphe G est une succession de sommets de G : $v_0, v_1 \ldots, v_n$, telle que pour tout $i \geq 0$, v_i est adjacent à v_{i+1} dans G. Un graphe G est dit *connexe* s'il existe, pour chaque couple de sommets distincts de G, un chemin les reliant. Un *cycle* est un chemin dont les deux extrémités sont égales. Un *arbre* est un graphe connexe ne contenant pas de cycle. Une *feuille* d'un arbre est un sommet de degré inférieur ou égal à un. Un *arbre recouvrant* T d'un graphe G est un sous-graphe de G contenant tous les sommets de G et étant un arbre.

Définition 2.1. *Arbre recouvrant*
Soit G un graphe et T un sous-graphe de G. T est un arbre recouvrant de G ssi :

$$V(T) = V(G) \text{ et est un arbre.}$$

Soient G un graphe et c un sommet de celui-ci. La *boule* de rayon 1 de G centrée en c, notée $B_G(c)$, est le sous-graphe de G contenant c et son voisinage. Plus formellement :

Définition 2.2. *Boule*
Soit G un graphe et c un sommet de G. La boule $B_G(c)$ *est le sous-graphe de G contenant c, son voisinage $N_G(c)$, et l'ensemble des arêtes incidentes à c :*

$$\begin{aligned} V(B_G(c)) &= \{x | x \in N_G(c)\} \cup \{c\} \\ E(B_G(c)) &= \{(c, x) | x \in N_G(c)\} \end{aligned}$$

2.2.1 Morphismes de graphes

La notion de calcul local se base sur les *homomorphismes de graphes*. Un *homomorphisme de graphe* – ou plus communément, *morphisme de graphe* – est une fonction totale φ de l'ensemble des sommets d'un graphe vers l'ensemble des sommets d'un autre graphe telle que la relation d'adjacence est conservée. On notera en général $\varphi(e)$ l'image d'une arête e par la fonction associant à chacune des extrémités de e son image par φ. Un isomorphisme (resp. monomorphisme/épimorphisme) est un morphisme bijectif (resp. injectif/surjectif). Dans le cadre de la théorie des graphes, un *revêtement* est un homomorphisme surjectif, localement bijectif. La figure 2.1 page ci-contre donne un exemple simple de revêtement.

Définition 2.3. *Homomorphisme de graphes*
Soient G et H deux graphes, soit φ une fonction totale de $V(G)$ vers $V(H)$, φ est un homomorphisme ssi :

$$\forall (v, w) \in E(G), (\varphi(v), \varphi(w)) \in E(H)$$

Définition 2.4. *Isomorphisme de graphes*
Soient G et H deux graphes, soit φ un morphisme de G dans H, φ est un isomorphisme ssi :

$$\begin{gathered} \forall w \in V(H), \exists v \in V(G), \varphi(v) = w \wedge \\ \forall v \ \in V(G), \forall w \ \in V(G), \varphi(v) = \varphi(w) \Rightarrow v = w \end{gathered}$$

Définition 2.5. *Revêtement*
Soient G et H deux graphes. Soit φ un morphisme de G dans H, φ est un revêtement *ssi :*

$$\begin{gathered} \forall v \in V(H), \exists w \in V(G), \varphi(w) = vtextet \\ \forall c \in V(G), \forall v \in B_G(c), \forall w \in B_G(c), \varphi(v) = \varphi(w) \Rightarrow v = w \end{gathered}$$

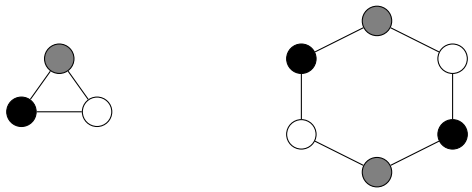


FIGURE 2.1 – Un exemple simple de revêtement : G est revêtement de H par la fonction associant à chaque sommet de H les sommets de même couleur dans G.

2.2.2 Graphes étiquetés

L'état d'un réseau est, dans le cadre des calculs locaux, défini par un *étiquetage*. On considère par la suite des étiquettes pour lesquelles l'égalité est décidable, ce qui nous permet par la suite d'utiliser des tests d'égalité d'étiquettes dans la définition de relation de réétiquetage décidable. Soit L un alphabet sur lequel l'égalité est décidable. Un *étiquetage* de G est une fonction $\sigma : V(G) \cup E(G) \to L$. On dénote par Σ_L l'ensemble des étiquetages sur l'alphabet L, Σ_L^G l'ensemble des étiquetages de G sur l'alphabet L et $\mathcal{G}_L$ l'ensemble des graphes étiquetés sur L. On note en général i, s ou t un étiquetage. Un *graphe étiqueté* sur L est noté G_s. On appelle G le graphe *sous-jacent* de G_s. L'étiquette d'un sommet ou d'une arête x est notée $s(x)$. Nous ne faisons pas ici la distinction entre étiquetage des sommets et étiquetage des arêtes pour des raisons de lisibilité. Pour être rigoureux, les arêtes et sommets étant des objets de types différents, nous devrions distinguer deux fonctions d'étiquetage : une sur les sommets et l'autre sur les arêtes. La formalisation présentée au chapitre suivant fait cette distinction.

On notera $s|G$ la restriction de l'étiquetage s au graphe G. On dira que H_t est un *sous-graphe étiqueté* de G_s ou encore que G_s *étend* H_t si H est un sous-graphe de G et si $s|G = t|G$. On note $s \uplus t$ la *mise à jour* de t avec s i.e. les éléments communs aux domaines des deux fonctions prennent leur étiquette dans s. Une fonction φ est un homomorphisme du graphe étiqueté G_s vers H_n si φ est un homomorphisme de G vers H préservant l'étiquetage i.e. pour tout sommet x de G, $s(x) = n(\varphi(x))$. On étendra de la même manière la définition des revêtements aux graphes étiquetés.

Définition 2.6. *Restriction*
Soient G un graphe, L un alphabet et s un étiquetage, on note $s|G$ la restriction de s à G telle que :

$$\forall x, x \in dom(s|G) \Leftrightarrow x \in V(G) \cup E(G)$$

Définition 2.7. *Sous-graphes étiquetés*
Soient G_s et H_t deux graphes étiquetés. H_t est un sous-graphe étiqueté *de G_s, ou encore*

G_s étend H_t *ssi :*

$$H \text{ est un sous-graphe de } G \wedge$$
$$\forall x \in V(H) \cup E(H), t(x) = s(x)$$

Définition 2.8. *Mise à jour*
Soient s et t deux étiquetages. On définit $s \uplus t$ comme :

$$s \uplus t(x) = \begin{cases} s(x) & si\ x \in dom(s) \\ t(x) & sinon \end{cases}$$

Nous appliquerons souvent par la suite une fonction f à l'ensemble des étiquettes d'un étiquetage s. Nous utilisons pour ce faire la notation $\widehat{f}(s)$ ou $\widehat{f}(G_s)$. Lorsque nous ferons la distinction entre étiquettes de sommets et d'arêtes, nous supposerons que f est un couple de fonction $f = (f_v, f_e)$, f_v étant appliquée aux étiquettes de sommet et f_e aux étiquettes d'arêtes.

Définition 2.9. *Image d'un étiquetage par une fonction*
Soient L et L' deux alphabets, s un étiquetages appartenant à Σ_L et f une fonction $f : L \to L'$. On note $\widehat{f}(s) \in \Sigma_{L'}$ l'état résultant de l'application de f à toute étiquette dans s :

$$\widehat{f}(s)(x) = f(s(x))$$

Lorsque nous traitons de graphes étiquetés, par exemple lorsque nous souhaitons définir un arbre recouvrant, nous considérons souvent un sous-graphe défini en fonction d'un étiquetage. Pour ce faire, nous proposons la notion de *filtre* de graphe étiqueté, à la manière des filtres de liste dans les langages fonctionnels. Soient P et P' deux prédicats définis respectivement sur L (l'étiquette d'un sommet) et L^3 (les étiquettes de deux sommets adjacents ainsi que de l'arête les reliant). Le *filtre* d'un graphe étiqueté G_s associé à P et P' est la fonction $F_{(P,P')}(G_s)$ dont le résultat contient les sommets (resp. arêtes) pour lesquels la propriété P (resp. P') est vérifiée.

Définition 2.10. *Filtre de graphe étiqueté*
Soient $G_s \in \mathcal{G}_L$ un graphe étiqueté, $P \subseteq L$ et $P' \subseteq L \times L \times L$ deux prédicats. $F_{(P,P')}(G_s)$ est défini comme :

$$V(F_{(P,P')}(G_s)) = \{v | v \in V(G) \wedge s(v) \in P\}$$
$$E(F_{(P,P')}(G_s)) = \{(v,w) | (v,w) \in E(G) \wedge (s(v), s((v,w)), s(w)) \in P'\}$$

Considérons par exemple un graphe où chaque sommet et arête est étiqueté par *true* (noir) ou $false$ (blanc). On définit le sous-graphe engendré par les sommets et arêtes étiqueté par *true* comme le filtre engendré par les prédicats $P(x) =_{def} x = true$ et $P'(x,y,z) =_{def} y = true$ (figure 2.2 page suivante).

(a) (b)

FIGURE 2.2 – Le graphe de la figure 2.2b est le résultat de l'application du filtre sélectionnant les sommets (gris) et arêtes (noires) marqués par *true* au graphe de la figure 2.2a

2.3 Systèmes de réétiquetage de graphes et calculs locaux

Le modèle des calculs locaux, tel qu'introduit par Litovsky, Métivier et Sopena [59] considère une classe d'algorithmes distribués pouvant être représentés par le biais de *relations de réétiquetage de graphes*. Une relation de réétiquetage de graphes est une *relation de réécriture* ayant pour domaine les graphes étiquetés et ne modifiant que l'étiquetage, en laissant le graphe sous-jacent inchangé. La figure 2.3 présente une relation de réétiquetage calculant un arbre recouvrant en un unique pas de calcul.

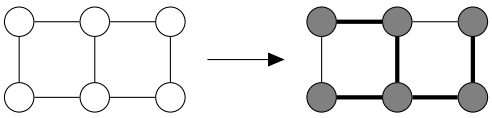


FIGURE 2.3 – Relation de réétiquetage marquant les sommets et arêtes de manière à ce que ceux-ci forment un arbre recouvrant.

Définition 2.11. *Relation de réétiquetage de graphe*
Soit L un alphabet. Une relation de réétiquetage R est un sous-ensemble de $\{(G_s, G_{s'})|(G_s, G_{s'}) \in \mathcal{G}_L{}^2\}$.

On appellera *R-réétiquetage* un couple de graphe appartenant à R. On notera un R-réétiquetage $G_s \xrightarrow{R} G_{s'}$. Un graphe étiqueté $G_{s'}$ est dit *accessible* depuis un graphe étiqueté G_s par R (noté $G_s \xrightarrow{R^*} G_{s'}$) si $(G_s, G_{s'})$ appartient à la clôture transitive et réflexive de R, que l'on notera R^*. Par la suite, nous utiliserons courament les notions de *formes normales*, *commutation* de relation de réétiquetage et *préservation des formes normales*.

Définition 2.12. Forme normale
Soit R une relation de réétiquetage de graphe. On dit d'un graphe étiqueté G_s qu'il est une forme normale *pour R, ou encore qu'il est* irréductible *pour R, que l'on note $G_s \in nf(R)$ ssi :*

$$\forall G_{s'}, (G_s, G_{s'}) \notin R$$

Définition 2.13. Commutation
Soient R et R' deux relations de réétiquetage de graphe. On dit que R' commute avec R ssi :

$$\forall G_s\ G_{s'}\ G_{s''}, G_s \xrightarrow{R'} G_{s'} \xrightarrow{R} G_{s''} \Rightarrow \exists G_t, G_s \xrightarrow{R} G_t \xrightarrow{R'} G_{s''}$$

Définition 2.14. Préservation des formes normales
Soient R et R' deux relations de réétiquetage de graphe. On dit que R' préserve les formes normales de R *ssi :*

$$\forall G_s\ G_{s'}, G_s \xrightarrow{R'} G_{s'} \wedge G_{s'} \in nf(R) \Rightarrow G_s \in nf(R)$$

On définit un *système de réétiquetage de graphe* comme un alphabet L, un ensemble de graphes étiquetés initiaux I, inclus dans l'ensemble des graphes étiquetés $\mathcal{G}_L$ et une relation de réétiquetage. On appellera *état* du système tout graphe étiqueté accessible depuis un graphe étiqueté initial. On remarque que dans l'exemple de la figure 2.3 page précédente, la relation de réétiquetage a une portée globale (modification de l'étiquetage de l'ensemble du graphe). On considérera par la suite des relations de réétiquetage dont la portée est au plus une boule.

Définition 2.15. *Système de réétiquetage de graphes*

Un système de réétiquetage est défini comme un tuple (L, I, R) tel que L est un alphabet, $I \subseteq \mathcal{G}_L$ est l'ensemble d'états initiaux *du système et $R \subseteq \{(G_s, G_{s'}) | (G_s, G_{s'}) \in {\mathcal{G}_L}^2\}$ est la* relation de réétiquetage *du système.*

L'exemple 1 page ci-contre est un système de réétiquetage dont l'objectif est de calculer un arbre recouvrant de tout graphe simple et connexe. L'arbre en question est engendré par les arêtes marquées *true*. Afin de définir complètement le système de réétiquetage réalisant ce calcul, il nous faut définir un ensemble d'états initiaux pour lesquels le calcul est valide. En l'occurrence, pour construire un arbre grâce à R, il est nécessaire de se donner *a priori* une racine. Les états initiaux de ce système de calcul seront donc les graphes étiquetés où un unique sommet est étiqueté par A. Nous verrons par la suite que nous pouvons donner au terme "a priori" une définition formelle, et montrerons que la connaissance à priori d'une racine est une condition nécessaire au bon fonctionnement du calcul décrit par R.

On nomme *exécution* d'un système de réétiquetage $S = (L, I, R)$ toute séquence de R-réétiquetages commençant par un graphe étiqueté appartenant à I. La figure 2.4 page suivante montre un exemple d'exécution du système SP. On remarque que le dernier état de la séquence est une *forme normale* pour R i.e. aucun R-réétiquetage n'existe depuis celui-ci. Par la suite, nous considérerons en général des exécutions *finies*. L'étude d'exécutions infinies peut toutefois s'avérer intéressante, notamment dans le cadre de systèmes *auto-stabilisants* [37].

Exemple 1 Système de réétiquetage SP de calcul d'arbre recouvrant

Le système calculant un arbre recouvrant est défini comme le triplet $SP = \{\{A, N, true, false\}, I_{SP}, R\}$ où :

- I_{SP} est l'ensemble des graphes connexes dont un unique sommet est étiqueté par A, les autres étant étiquetés par N, et dont toutes les arêtes sont étiquetés par $false$.
- Soit un sommet c, étiqueté par N, possédant un voisin v étiqueté par A tel que l'arête (c, v) est étiqueté par $false$. R transforme l'étiquette de c en A et l'étiquette de (c, v) en $true$.

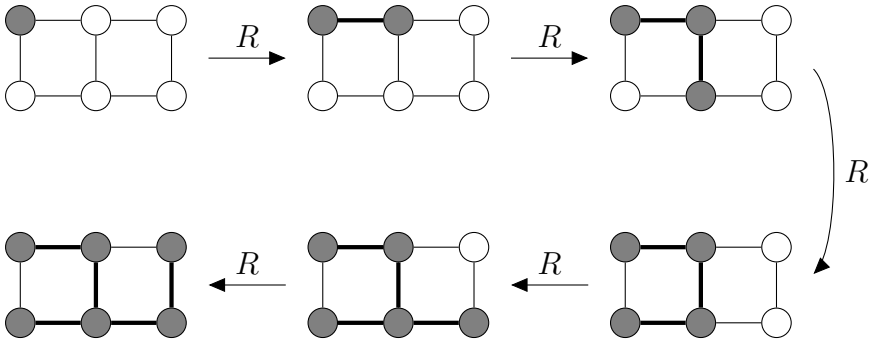
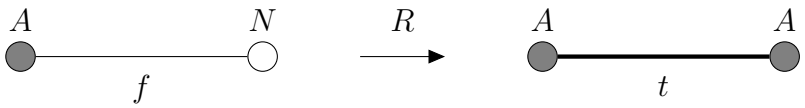


FIGURE 2.4 – Une *exécution* du système de calcul d'un arbre recouvrant (Exemple 1)

On décrira si possible une relation de réétiquetage en ne représentant que le sous-graphe affecté par la relation et en négligeant le contexte n'ayant pas d'influence sur son application. On nommera cette représentation une *règle de réétiquetage*. La règle 1 correspond par exemple à la relation de réétiquetage R de l'exemple 1.

Règle 1 Règle de calcul d'un arbre recouvrant



La relation engendrée par une telle *règle de calcul* correspond au *plongement* de celle-ci dans un graphe étiqueté donné. Nous noterons $\overline{R}$ le plongement de R dans G_s. En l'absence d'ambiguïté, nous utiliserons le nom d'une règle pour désigner à la fois la règle elle-même et son plongement dans un graphe. Nous laissons pour le moment de côté la définition formelle du plongement d'une règle, celle-ci sera donnée spécifiquement pour chacun des modes de synchronisation que nous étudierons par la suite. Dans les cas complexes, on divisera une relation de réétiquetage en plusieurs règles pour simplifier la représentation et modulariser notre raisonnement. La relation engendrée par l'ensemble des règles correspondra alors à l'union des relations engendrées par chacune.

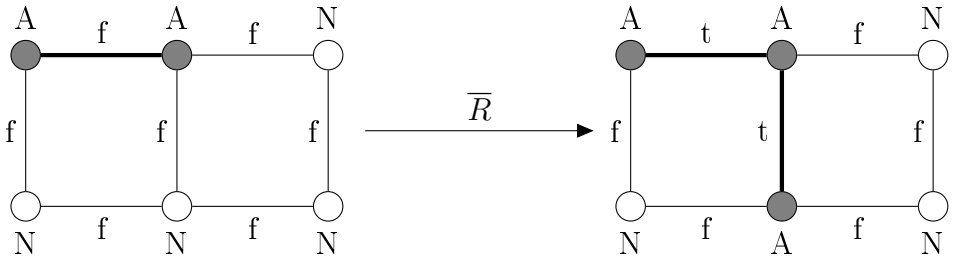


FIGURE 2.5 – Plongement $\overline{R}$ de R dans G_s

2.3.1 Relations de réétiquetages localement engendrées

Si l'on considère la règle 1 page précédente, on remarque que sa *portée* est limitée à un sous-graphe constitué de deux sommets adjacents. Dans la suite de ce document, nous limiterons notre étude aux relations de réétiquetage dont la portée n'excède pas une boule. Celles-ci correspondent à la représentation intuitive que nous nous faisons d'un algorithme distribué : seuls les processeurs reliés physiquement (par câble Ethernet, liaison radio, ou autres) peuvent communiquer entre eux. Outre cette contrainte de localité, il est souhaitable de pouvoir étudier les systèmes dits *anonymes* [35], où un processeur est indistinguable d'un autre par un système de réétiquetage (i.e. les processeurs ne possèdent pas d'identifiant unique, ou le système ne peut pas les utiliser). Dans le cadre du réétiquetage de graphe, cela revient à ne considérer que les relations dépendant uniquement de la topologie du graphe et de l'étiquetage.

Nous restreignons donc la classe de relations étudiées aux relations de réétiquetage dont la portée n'excède pas une boule et *stables par isomorphismes*. Cette classe de relations est appelée relations de réétiquetage *localement engendrées* [59]. Une relation de réétiquetage est localement engendrée si les propriétés suivantes sont respectées :

- Tout pas de réétiquetage affecte uniquement le voisinage d'un sommet.
- Le résultat de tout pas de réétiquetage est uniquement influencé par le voisinage d'un sommet.
- Tout pas de réétiquetage commute avec les isomorphismes de graphes.

ou plus formellement :

Définition 2.16. *Relation de réétiquetage localement engendrée*
Soit $\rightarrow$ une relation de réétiquetage.
Soient $G_s, G_{s'}$ et $H_t, H_{t'}$ quatre graphes étiquetés. Soient c un sommet de G et φ un isomorphisme de $B_G(c)$ vers $B_H(\varphi(c))$,
si $s|(G \backslash B_G(v)) = s'|(G \backslash B_G(v))$
et $B_G(c)_s$ (resp. $B_G(c)_{s'}$) est isomorphe à $B_H(\varphi(c))_t$ (resp. $B_H(\varphi(c))_{t'}$) par φ
alors $G_s \rightarrow G_{s'} \Leftrightarrow H_n \rightarrow H_{t'}$.

On désignera couramment les relations de réétiquetage localement engendrées par le terme de *systèmes de calculs locaux*. Cette notion de localité permet entre autres d'é-

tudier le comportement d'algorithmes dans le cas où une forme de symétrie (isomorphismes, revêtements) est présente dans un réseau. Les résultats d'Angluin [10] par exemple, utilisent la localité et la stabilité par isomorphisme des calculs locaux pour démontrer qu'il n'existe pas d'algorithme élisant un leader dans un graphe quelconque .

2.4 Tâches et réalisations

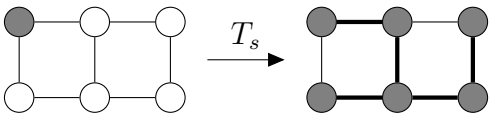
Une *tâche* sur un réseau [25, 22] est une relation de réétiquetage de graphe *globale*, spécifiant l'objectif d'un algorithme. Considérons par exemple le problème du calcul d'arbre recouvrant, présenté plus tôt : l'objectif est de construire, à partir d'une racine *connue a priori*, un arbre recouvrant d'un graphe connexe. Cette caractérisation du problème met en lumière deux éléments importants : les connaissances initiales nécessaires à la résolution du problème et le résultat souhaité. Ces deux ensembles de graphes peuvent ne pas être étiquetés par le même alphabet. Si l'on considère de nouveau le problème de calcul d'arbre recouvrant, les arêtes ne seront pas initialement étiquetées, alors que dans les états finaux, celles-ci seront étiquetées par un booléen.

Définition 2.17. *Tâche*
Soient L_i et L_o deux alphabets. Une tâche T est un sous ensemble de $\{(G_s, G_t) | G_s \in \mathcal{G}_{L_i} \wedge G_t \in \mathcal{G}_{L_o}\}$. On nommera les états initiaux *(ou domaine) de T le domaine de la relation T.*

Pour revenir à l'exemple du problème de l'arbre recouvrant, nous définissons la tâche correspondante de la manière suivante :

Tache 1 Tâche T_s de calcul d'un arbre recouvrant dans un graphe connexe

Le domaine de T_s est l'ensemble des graphes connexes dont un unique sommet est étiqueté par *true*. Les sommets et arêtes des graphes du domaine d'arrivée de T sont étiquetés par *true* ou $false$. Un couple (G_i, G_s) appartient à T_s si le sous-graphe engendré par les sommets et arêtes marqués *true* de G_s est un arbre recouvrant de G.



On dit qu'un système de réécriture *réalise* une tâche si celui-ci résout le problème que la tâche spécifie.

Définition 2.18. *Réalisation*
Un système de réétiquetage $S = (L, I, R)$ réalise une tâche $T \in \mathcal{G}_{L_i} \times \mathcal{G}_{L_o}$ ssi il existe deux fonctions ι de L_i vers L et π de L vers L_o telles que :

- *l'image par ι des étiquettes des sommets et arêtes d'un graphe étiqueté G_I du domaine de T résulte en un état initial (un graphe étiqueté appartenant à I).*
- *l'application de π à toutes les étiquettes des sommets et arêtes de toute forme normale G_f de R accessible depuis un état G_I de I résulte en un graphe étiqueté G_F image de G_I par T.*

$$\begin{array}{ccc} G_I & \xrightarrow{\quad T \quad} & G_F \\ \Big\downarrow{\scriptstyle \widehat{\iota}} & & \Big\uparrow{\scriptstyle \widehat{\pi}} \\ G_i & \xrightarrow{\quad R^* \quad} & G_f \end{array}$$

On dira encore qu'un triplet (ι, π, S) est une *réalisation* d'une tâche T. On nomme ι la *fonction d'initialisation* de la réalisation et *pi* la *fonction de projection du résultat*, ou *fonction de projection*. S'il existe une réalisation d'une tâche T on dira que celle-ci est *réalisable*. On illustre ces notions par l'exemple 1 page 23. Soit ι_s une fonction assignant à tout sommet étiqueté par *true* (resp. $false$) l'étiquette A (resp. N) et à toute arête l'étiquette $false$. Soit π_s une fonction associant à tout sommet étiqueté par A (resp. N) l'étiquette *true* (resp. $false$) et conservant l'étiquetage *true* ou $false$ des arêtes. On remarque que le système décrit par l'exemple 1 page 23, forme, avec ι_s et π_s une réalisation de la tâche T_s.

On remarque que les états initiaux d'un système de réétiquetage sont en général aisément déduisibles du domaine d'une tâche. Lorsque la tâche à réaliser est clairement définie, les *états initiaux* d'un système désigneront l'application de $\widehat{\iota}$ à tous les éléments du domaine de la tâche.

Si toute exécution d'un système de réétiquetage est finie, alors la notion de réalisation correspond à la *correction totale* d'un algorithme. Outre son utilisation dans le cadre de la preuve d'algorithmes concrets comme dans le cas de la section 3.8 page 48, la notion de réalisation s'avère utile pour démontrer l'impossibilité de résoudre un problème de manière distribuée. La distinction effectuée entre les alphabets des tâches et ceux des systèmes de réétiquetage permet en effet d'énoncer que quelque soit l'alphabet choisi, quelque soit la relation de réécriture employée, aucun système ne réalise une tâche. Cet aspect sera étudié plus en détail dans le chapitre 4.

2.5 Modes de détection de la terminaison

Contrairement à la notion de correction présentée ci-dessus, proche de celle que l'on peut trouver dans le cadre de l'algorithmique séquentielle, la notion de *terminaison* en

algorithmique distribuée est problématique. Dans le cadre séquentiel, une fonction termine lorsqu'elle retourne un résultat. Dans un cadre distribué le concept de terminaison est plus flou : un algorithme termine-t-il lorsqu'aucun message ne circule plus sur le réseau ? lorsque le résultat escompté a été calculé ? Dans un réseau dépourvu de structures de contrôle centrales, comment un processeur peut-il "prendre conscience" de la terminaison de l'application d'un algorithme sur le réseau ? Les auteurs de [47, 25] proposent un ensemble de réponses à ces questions ou *modes de détection de la terminaison*. Un mode de détection de terminaison spécifie (1) ce que signifie *la terminaison* d'un système de réétiquetage et (2) quels processeurs d'un réseau sont "conscients" de cette terminaison. Les définitions présentés ici sont celles issues de [22] et diffèrent quelques peut de celles proposées par Y. Métivier *et Al.*.

2.5.1 Détection locale de la terminaison locale (*LTD*)

Reprenons l'exemple 1 page 23 de calcul d'arbre recouvrant dans un graphe connexe. Rien n'indique à chaque sommet qu'une forme normale ait été atteinte. Par contre, lorsque l'étiquette d'un sommet devient A, celle-ci ne change plus, i.e. une fois un sommet ajouté à l'arbre, il n'en sera pas retiré. Lorsque l'algorithme atteint une forme normale, chaque sommet a atteint son état final et fait partie de l'arbre. En d'autres termes (i) un sommet détecte le fait que le résultat calculé localement (l'appartenance à l'arbre) est définitif, et (ii) à la fin d'une exécution, tout sommet a détecté le fait que son résultat est définitif. On nommera ce mode de détection de la terminaison la *détection locale de la terminaison locale* (ou mode LTD).

Définition 2.19. *Détection locale de la terminaison locale*
Soient $S = (L, I, R)$ un système de réétiquetage et (ι, π, S) une réalisation d'une tâche T. On dit que (ι, π, S) réalise T avec détection locale de la terminaison locale *s'il existe une fonction τ de L vers $\{true, false\}$ telle que*

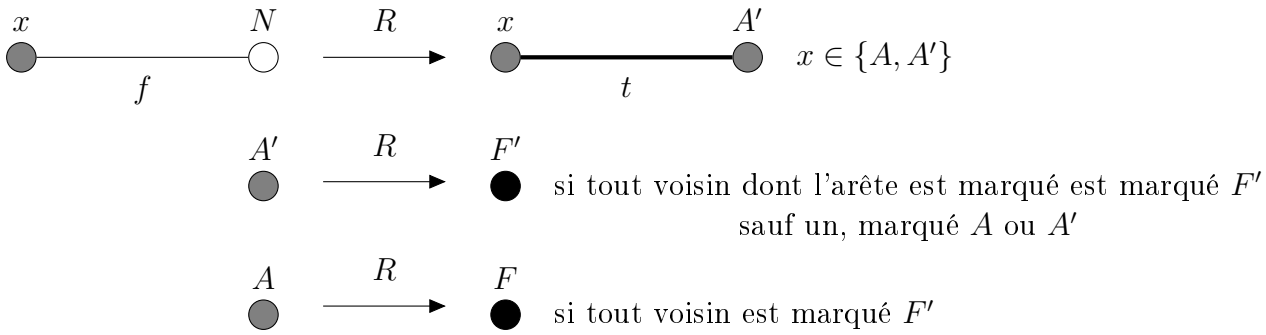
- *Pour tout sommet v d'un graphe étiqueté G_s, si $G_s \xrightarrow{R} G'_s$ et $\tau(s(c)) = true$ alors $\tau(s'(v)) = true$ et $\pi(s(c)) = \pi(s'(c))$.*
- *Soit G_s un état accessible depuis un graphe étiqueté de I, si G_s est une forme normale pour R, alors pour tout sommet v de G, $\tau(s(v) = true$.*

2.5.2 Détection locale de la terminaison globale (*GTD*)

Le mode LTD fournit peu d'information. Dans notre exemple, un processeur ne peut pas savoir si la construction de l'arbre est finie au niveau de son voisinage, et encore moins au niveau global. Il est néanmoins possible de modifier le système de réétiquetage originel pour que la racine soit avertie de la terminaison de tout calcul sur le réseau (voir règles 2 page suivante). Initialement, comme dans le système originel, la racine est étiquetée par A. Lorsqu'un sommet est ajouté à l'arbre, celui-ci prend l'étiquette A'. Une fois les feuilles de l'arbre atteintes, l'information de terminaison remonte au fur et à mesure vers la racine, à la manière de l'algorithme de Dijsktra-Schoelten [38]. Toute exécution de ce système (cf

figure 2.6) se conclut par un changement de l'étiquette de la racine de A en F. En d'autres termes, un sommet est marqué F si et seulement si une forme normale est atteinte. Ce mode de détection de terminaison, où au moins un sommet est averti de la terminaison globale d'un algorithme est nommé *détection locale de la terminaison globale* (ou mode GTD).

Règle 2 Calcul d'arbre recouvrant dans un graphe connexe, avec détection locale de la terminaison globale.



Définition 2.20. *Détection locale de la terminaison globale*
Soient $S = (L, I, R)$ un système de réétiquetage et (ι, π, S) une réalisation d'une tâche T. On dit que (ι, π, S) réalise T avec détection locale de la terminaison globale *s'il existe une fonction τ de L vers les booléens telle qu'un état G_s accessible depuis un graphe étiqueté appartenant à I est une forme normale pour R si et seulement si il existe un sommet v de G tel que $\tau(s(v)) = true$.*

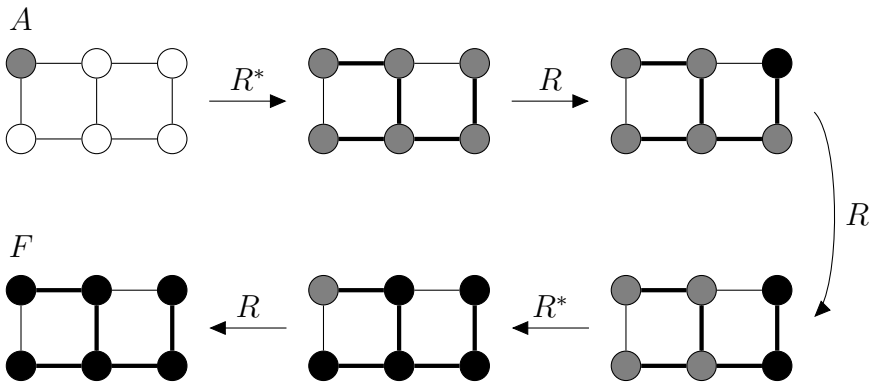


FIGURE 2.6 – Une exécution du système de calcul d'un arbre recouvrant avec détection locale de la terminaison globale (Règles 2)

2.5.3 Détection observée de la terminaison (*OTD*)

Les modes GTD et LTD sont deux modes extrêmes de la détection de terminaison : en mode GTD, un unique sommet détecte la terminaison de l'algorithme sur l'ensemble du

graphe. En mode LTD tout sommet détecte sa propre terminaison. Un mode de détection de la terminaison intermédiaire peut être considéré : la *détection observée de la terminaison* (ou mode OTD). Dans ce dernier, chaque sommet détecte lorsque le résultat calculé par l'ensemble des processeurs du réseau est final.

Définition 2.21. *Détection observée de la terminaison*
Soient $S = (L, I, R)$ un système de réétiquetage et (ι, π, S) une réalisation d'une tâche T. On dit que (ι, π, S) réalise T avec détection locale de la terminaison observée *s'il existe une fonction τ de L vers les booléens telle que pour tout état G_s accessible depuis un état de I :*

- *pour tout sommet v de G, si $\tau(s(v)) = true$ alors pour tout sommet v' de G, si $G_s \xrightarrow{R} G_{s'}$ alors $\tau(s'(v)) = true$ et $\pi(s(v')) = \pi(s'(v'))$.*
- *si G_s est une forme normale pour R alors pour tout sommet v de G, $\tau(s(v)) = true$.*

2.5.4 Hiérarchie des modes de détection de la terminaison

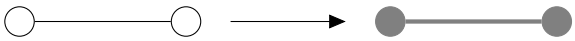
On remarque que tout système réalisant une tâche en mode *OTD* réalise aussi celle-ci en mode *LTD*. En effet, si un sommet détecte la terminaison pour tout sommet d'un graphe, alors il détecte par la même sa propre terminaison. Toute tâche réalisable par un système de réétiquetage *OTD* est donc réalisable par un système de réétiquetage *LTD*. Un résultat similaire peut être obtenu pour les systèmes *OTD* et *GTD*. On considère un algorithme *GTD*, une fois l'information de terminaison reçue par un sommet, celle-ci peut être retransmise à l'ensemble des sommets du graphe, transformant un algorithme *GTD* en algorithme *OTD* [47]. Nous étudierons plus en détail la preuve de cette transformation dans le cadre des relations **LC0** en 5.5 page 87.

2.6 Modes de synchronisation

La définition d'un calcul local donnée précédemment est large et ne correspond pas intuitivement à un unique paradigme de communication courant (client serveur, broadcast, etc). Des ensembles de relations plus spécifiques ont été introduits par J. Chalopin [23] pour représenter les primitives de communications communément employées. Ces travaux établissent une hiérarchie précise des modes de synchronisation. Nous nous concentrerons pour notre part sur les trois modes suivant : les calculs locaux *sur les arêtes*, les calculs locaux *sur les boules ouvertes*, et les calculs locaux *sur les boules*. Chacun de ces modes ont été implantés, dans ViSiDiA, à l'aide d'algorithmes probabilistes [15, 92]. Une implantation en *Coq* par Pierre Castéran est aussi en cours. Celle-ci se base sur la définition en *Gallina* d'un générateur de nombres pseudo-aléatoires ainsi que sur une interprétation fonctionnelle des différents modes de synchronisation.

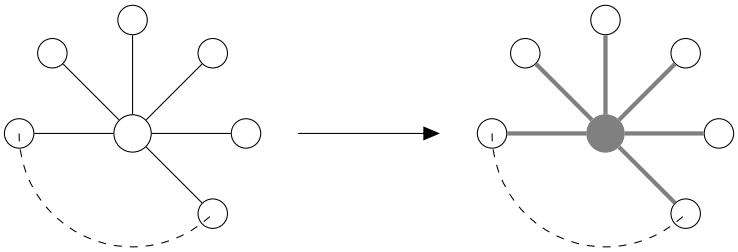
2.6.1 Calculs locaux sur les arêtes (LC0)

Ce premier mode de synchronisation, que l'on nommera **LC0** est illustré par la règle 1 page 23 de calcul d'arbre recouvrant. Ce mode correspond grossièrement au paradigme client-serveur de communication. Le réétiquetage concerne deux sommets adjacents et l'arête les reliant. Toute relation **LC0** est une relation de réétiquetage localement engendrée.



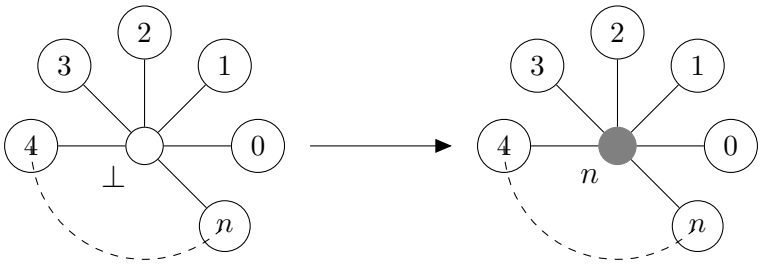
2.6.2 Calculs locaux sur les boules ouvertes (LC1)

On considère ici une boule étiquetée centrée sur un sommet c. Une règle de réétiquetage sur les boules ouvertes peut uniquement modifier l'étiquette du centre ainsi que les étiquettes des arêtes de la boule. Les étiquettes des voisins de c sont accessibles en lecture seulement.



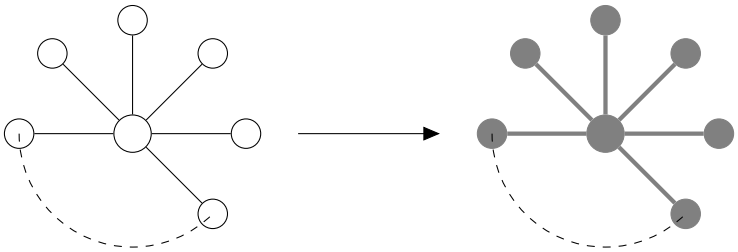
Contrairement à une relation **LC0** (voir 4.2 page 59), une relation **LC1** est capable de compter le cardinal de son voisinage avec détection locale de la terminaison locale. La règle **LC1** 3 calcul le degré de chaque sommet dans un graphe quelconque.

Règle 3 Calcul du degré de chaque sommet, avec détection locale de la terminaison locale



2.6.3 Calculs locaux sur les boules (LC2)

Comme précédemment, on considère une boule étiquetée centrée sur un sommet c. Une règle de réétiquetage sur les boules peut modifier les étiquettes du centre et de ses voisins, ainsi que les étiquettes des arêtes de la boule. La relation engendrée par les règles de réécriture 2 page 28 est une relation **LC2** : la première règle modifie l'étiquetage d'un sommet et d'un de ses voisins, et les deux suivantes lisent l'ensemble du voisinage d'un sommet.



2.6.4 Hiérarchie des modes de synchronisation

Nous remarquons immédiatement que pour toute relation **LC0** ou **LC1** il existe un équivalent **LC2**. Les travaux de J. Chalopin [23] établissent de plus qu'un algorithme **LC0**, avec connaissance initiale du degré de chaque sommet, peut simuler tout système **LC1** ([23], Proposition 3.42). La figure 2.7 représente les rapports entre les trois modes de synchronisation ainsi que les systèmes **LC0** avec connaissance initiale du degré de chaque sommet.

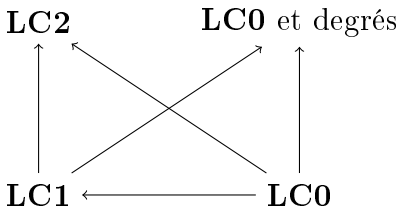


FIGURE 2.7 – Relation (inclusion) des différents modes de synchronisation entre eux

2.7 Conclusion

Nous avons défini dans ce chapitre le modèle des calculs locaux, tels que proposé par Métivier *et al.* Ce modèle de haut-niveau nous permet d'exprimer de manière concise des algorithmes distribués classique, tel qu'un calcul d'arbre recouvrant ou une élection. Nous avons vu que ce modèle nous permet de distinguer naturellement la topologie du réseau sur lequel un algorithme est exécuté de l'algorithme lui même. Nous verrons par la suite que cette propriété du modèle simplifiera la formalisation de preuves d'impossibilité.

Les spécifications d'un algorithme sont, dans ce modèle, décritent par deux notions : la tâche accomplie et la détection de terminaison. Nous proposons des définitions simplification de ces notions, dérivée des définitions proposées dans la littérature.

Chapitre 3

Sémantique des Systèmes de Calculs Locaux dans le Calcul des Constructions Inductives

Dans ce chapitre nous proposons une sémantique relationelle des systèmes de réétiquetages de graphes localement engendrés dans le calcul des constructions inductives. Nous montrons comment des sous-classes de relations de réétiquetage localement engendrées peuvent être définies à l'aide de *plongements*. Finalement nous proposons un schéma de preuve de correction pour les systèmes de calculs locaux.

Sommaire

3.1 Introduction

Dans ce chapitre nous proposons une sémantique des calculs locaux exprimée dans le Calcul des Constructions Inductives. Notre approche est guidée par un objectif principal : autoriser, dans un unique environnement, l'étude formelle de l'expressivité des calculs locaux ainsi que les preuves d'algorithmes exprimés dans ce modèle. Il nous faut, pour ce faire, être capable d'assigner un *type* aux objets manipulés : spécifications (tâches) et algorithmes (relations de réétiquetages). Nous montrons ici comment le calcul des constructions inductives permet ces définitions.

La méthodologie de formalisation proposée repose sur trois "couches". La première concerne la théorie des graphes et des graphes étiquetés. Celle-ci regroupe les définitions de graphes, de morphismes de graphes, ainsi que les lemmes intermédiaires les concernant. La seconde regroupe les définitions et résultats communs à l'ensemble des relations de réétiquetage de graphe, notamment les définitions en *CIC* d'un calcul local, des différents modes de détection de terminaison, d'un invariant, etc. Une troisième couche contient les définitions des modes de synchronisations **LC0**, **LC1** et **LC2** ainsi que leurs théorèmes "compagnons". Chacun de ces modes de synchronisation est caractérisé par un type et un *plongement* de ce type dans le type des relations de réétiquetages localement engendrées. Ce plongement définit la sémantique du mode de synchronisation en terme de relations de réétiquetage. Il prouve en plus que ces dernières sont bien des relations de réétiquetage localement engendrées. Nous pouvons ensuite raisonner sur la sémantique définie par le plongement pour obtenir des propriétés spécifiques à la synchronisation considérée, pour nous assister aux preuves de correction, ou étudier l'expressivité du mode.

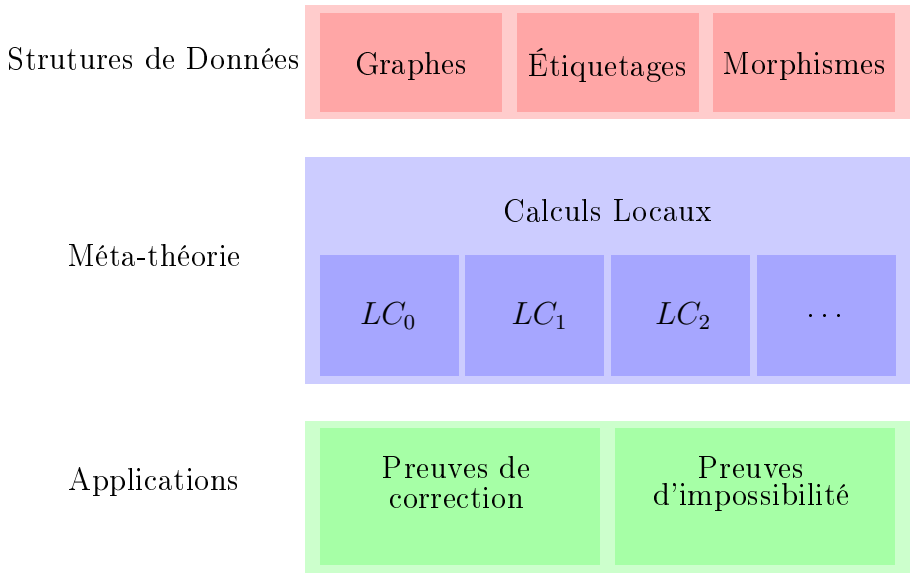


FIGURE 3.1 – Structure de notre bilbiothèque *Coq*

3.1.1 Organisation du chapitre

Nous commençons le chapitre par une rapide présentation de l'assistant de preuve *Coq*, en expliquant l'importance de l'expressivité de son système de types. Nous continuons en décrivant notre implantation de la théorie des graphes, en donnant les types et définitions principales. Nous proposons ensuite une sémantique relationnelle aux calculs locaux en *CIC* ainsi qu'un encodage des différents modes de synchronisation, des tâches, des invariants et des modes de détection de la terminaison. Finalement nous montrons comment cette implantation de la théorie des calculs locaux peut être utilisée pour démontrer formellement la correction d'un système de réétiquetage.

3.1.2 Travaux relatifs

Nous avons présenté dans le chapitre 1.1 les efforts de formalisation sur les modèles populaires de l'algorithmique distribué. Nous décrivons ici les travaux ayant trait aux calculs locaux en particulier, ainsi qu'aux formalisations de la théorie des graphes.

Le travail présenté ici se rapproche et s'inspire en partie de celui de M. Mosbah et R. Ossami [75, 76] : dans [75], ces derniers proposent un langage de programmation (LiDIA) dont l'expressivité correspond aux calculs locaux. LiDIA est essentiellement un langage de transitions gardées, où les gardes des transitions sont exprimés par des formules de la logique $\mathcal{L}^*_\infty$. Les transitions sont exprimées en utilisant des primitives de communication (*Send*, *SendAll*, etc). Il nous semble que l'utilisation de primitives de communication au sein d'une règle de réétiquetage annule le bénéfice de raisonner à un niveau abstrait. Toutefois, la logique $\mathcal{L}^*_\infty$ nous semble intéressante et l'étude de R. Ossami pourrait être utilisée à l'avenir pour parfaire les définitions en *Coq* des relations **LC1** et **LC2**.

M. Tounsi [92, 73, 93, 68] suit une approche différente de formalisation des calculs locaux. Ce dernier propose une méthodologie de preuve basée sur la méthode B-événementielle et l'approche "correcte par construction". Dans ces travaux, un système de calculs locaux est développé en partant de spécifications abstraites globales (correspondant à la notion de tâche). Les informations nécessaires à la réalisation de cette tâche sont localisées au fur et à mesure pour obtenir finalement une implantation déterministe du processus exécuté par chaque noeud du réseau. Une correspondance est établie entre les règles des systèmes de réétiquetages et les événements d'une machine B-événementielle. Cette méthode est efficace pour le développement d'algorithmes distribués (notamment en ce qui concerne l'automatisme des preuves) mais ne permet, pas selon, nous l'étude formelle de l'expressivité des systèmes de calculs locaux.

Les travaux de J. Duprat [39], C. Chou [31, 32] et B. Robillard [86] visent à formaliser la théorie des graphes à l'aide d'assistants de preuve. Dans la contribution de J. Duprat, les ensembles de sommets et d'arêtes sont représentés en *Coq* par des propositions pour lesquelles la relation d'appartenance n'est pas a priori décidable. Cette bibliothèque ne

contient donc pas d'aspects calculatoires, ce qui la rend moins pratique pour mettre en place des procédures de décision des propriétés de graphe formalisées. Or, la mise en place de telles procédures nous semble importante pour faciliter, voire, dans certains cas, automatiser les raisonnements sur des cas particuliers (dans le cadre de contre exemples et de preuves d'impossibilité notamment). C. Chou propose dans [31] une formalisation de la théorie des graphes à l'aide de l'assistant de preuve *HOL* et utilise cette dernière dans [32] pour vérifier la correction d'algorithmes distribués. Les définitions ensemblistes de C. Chou, pour les mêmes raisons que dans le développement de Duprat, ne nous semblent pas non plus appropriées aux calculs sur les graphes. La formalisation en *Coq* proposée par B. Robillard [86] semble celle qui se rapproche le plus de nos propres travaux. Ce dernier formalise la théorie des graphes en utilisant le système de module de *Coq* et en mettant l'accent sur la possibilité de calculer sur les types définis (avec comme objectif l'extraction de code). La bibliothèque en résultant est plus générique et flexible que celle que nous proposons ici : le type des graphes y est paramétré par un type ordonné représentant les sommets, là où nous avons choisi arbitrairement de représenter les sommets par des entiers. Toutefois nous préférons à l'utilisation de modules l'emploi de *classes de types* [89]. Un remplacement, dans notre formalisation, des modules de fonctions finies [42] de la bibliothèque standard de *Coq* par les représentations proposées par S. Lescuyer [57] a d'ailleurs été étudié par A. Fontaine.

3.2 Sur l'expressivité du Calcul des Constructions Inductives.

L'assistant de preuve *Coq* est une implantation du calcul des constructions inductives : un lambda calcul dont le système de type autorise les familles de types (ou types d'ordre supérieur), le polymorphisme et les types dépendants. Un type peut lui même appartenir à une des trois *sortes* suivantes : *Prop*, la sorte des propositions, *Set* la sorte des spécifications et *Type* la sorte de tous les types. En *Coq*, l'énoncé d'un théorème est un type de sorte *Prop*. Prouver ce théorème revient à construire un terme du type donné par l'énoncé. On écrira par exemple en *CIC* l'énoncé de logique propositionelle "Pour toutes propositions A et B, si A implique B et A alors B." comme :

$$\forall (A\ B\ :\ Prop), (A \to B) \to A \to B$$

Nous pouvons interpréter le type ci dessus comme : soient deux propositions A et B, si nous connaissons (i) une fonction transformant une preuve de A en une preuve de B et (ii) une preuve de A, alors nous pouvons construire une preuve de B. Le terme de preuve correspondant est :

$$\lambda\ (A\ B\ :\ Prop)(f\ :\ A \to B)(a\ :\ A) \Rightarrow\ (f\ a)$$

On remarque l'utilisation de deux constructeurs de types : la flèche ($\rightarrow$), correspondant aux types des fonctions, et le produit dépendant ($\forall$) [1]. En *Coq*, le produit dépendant nous permet par exemple de typer les objets (i) (ii) et (iii), où `state Lv Le` est le type d'un étiquetage. On distingue dans la formalisation le type des étiquettes des sommets, noté `Lv` et celui des étiquettes des arêtes, noté `Le`.

```
(i)   LC0_relabeling : Type → Type → Type

(ii)  LC0_Step_At_center : ∀ (Lv Le:Type)(R:LC0_relabeling Lv Le),
        Graph_t → vertex → state Lv Le→ state Lv Le→ Prop

(iii) LC0_Localy_generated : ∀ (Lv Le:Type)(R:LC0_relabeling Lv Le),
        Locally_generated (LC0_Step_At_center Lv Le  R)
```

Dans (i), `LC0_relabeling` est la famille des types des relations de réétiquetages **LC0**.

Dans (ii), `LC0_Step_At_center` est une fonction prenant en paramètre deux types, une relation **LC0** et retournant une *relation de réétiquetage localisée* (cf Definition 3.5 page 41). On dira que `LC0_Step_At_center` est le *plongement* des relations **LC0** dans les relations de réétiquetage localisées. Comme dans (i), le type de `LC0_Step_At_center` est un type d'ordre supérieur.

Dans (iii), `LC0_localy_generated` est une fonction prenant en paramètre deux types, une relation **LC0** et retournant une *preuve* que le plongement de cette relation dans les relations de réétiquetage localisées est localement engendré.

Lorsque nous étudions une relation de réétiquetage particulière, nous travaillons sur une instances de ces schémas de types. Lorsque nous raisonnons sur l'expressivité des calculs locaux, nous conservons au contraire la quantification sur les types et relations, à la manière de (iii).

3.3 Graphes étiquetés

Nous décrivons ici une formalisation originale de la théorie des graphes et graphes étiquetés en *Coq*, basée sur la librairie d'ensembles et fonctions finies développée par P. Letouzey et J.C. Filliâtre [42]. Cette formalisation n'a pas pour objectif d'être exhaustive, notre sujet d'étude principal étant la théorie des calculs locaux. Nous nous concentrons donc sur les notions nécessaires à sa représentation.

1. En pratique la flèche est un cas particulier du produit.

3.3.1 Graphes

Nous définissons le type des graphes comme une structure contenant deux ensembles : un ensemble de sommets V et un ensemble d'arêtes E. Nous utilisons l'implantation des ensembles finis en *Coq* de J.C. Filliatre et P. Letouzey, la librairie *FSet*. La librairie *FSet*[42] a été initialement conçue pour vérifier en *Coq* les implantations des arbres *AVL* utilisées dans la librairie standard Ocaml [2] pour manipuler les ensembles finis. Elle a ensuite été intégrée à la bibliothèque standard *Coq* comme une alternative calculatoire à la représentation pré-existante des ensembles par des propriétés.

Nous représentons par la suite les sommets comme des entiers. Une arête est une paire de sommets. On notera par la suite *vertex* (un *alias* pour les entiers naturels) le type des sommets et *edge* le type des arêtes. Le choix des entiers comme représentants des sommets a peu d'incidence par la suite, étant donné que l'on traite de relations de réétiquetage indépendantes du nommage des sommets. Dans de futures versions de la formalisation, il serait néanmoins intéressant de rendre le type des graphes polymorphes, i.e. de paramétrer ce dernier par un type pour les sommets, à la manière de [86]. Ceci permettrait d'implanter de manière intuitive certaines constructions, comme par exemple l'énumération de l'ensemble des revêtements d'un graphe telle que définie par Gross et Tucker [48].

Définition 3.1. *Type des graphes [Coq 0]*

```
Class Graph_t :Type := {
  V_of : Vertices.t;
  E_of : Edges.t }.
```

Ce type n'est pas assez restrictif : rien ne force l'appartenance des extrémités d'une arête à l'ensemble des sommets. De plus, comme nous définissons les arêtes comme des paires de sommets, cette définition permet de représenter les graphes orientés (une paire est, par définition, orientée), ainsi que les graphes contenant des boucles. Nous restreignons donc l'ensemble de graphes que nous utilisons à l'aide d'un prédicat `Graph`, spécifiant que :

(i) Les extrémités d'une arête d'un graphe appartiennent à l'ensemble des sommets de ce graphe.

(ii) l'ensemble des arêtes ne contient que des couples de la forme (x, y) tels que $x < y$.

(i) nous assure que la structure manipulée est bien un graphe. (ii) nous assure que ce graphe est non orienté et qu'il ne contient pas de boucles. On trouvera une définition proche dans [31] (Definition 1 page 4).

3.3.2 Étiquetages

Un *type d'étiquette* est un type dont l'égalité est décidable. Cette définition correspond à la restriction communément trouvée dans les travaux sur les calculs locaux, restreignant

les étiquetages à des alphabets récursivement énumérables. Pour des raisons de typage, nous décomposons l'étiquetage d'un graphe en deux fonctions finies : l'une associant une étiquette aux sommets (notée *lambda* dans le code source) et l'autre aux arêtes (notée *rho*). Dans ce cadre les étiquettes des sommets et arêtes ne sont pas nécessairement du même type. Pour simplifier la lecture, on notera si possible par la suite L un couple de types d'étiquette et Σ_L le type des étiquetages.

3.3.3 Morphismes de graphes et de graphes étiquetés

Comme exposé dans le chapitre précédent, les morphismes de graphes jouent un rôle fondamental dans la définition d'un calcul local. Ceux-ci permettent en outre de raisonner sur les symétries locales dans un graphe (revêtements), utilisées pour démontrer l'impossibilité d'élire dans un graphe quelconque. Nous proposons ici des définitions formelles des morphismes de graphes et graphes étiquetés. On considérera pour ce faire des fonctions totales sur les ensembles finis. Soit G et H deux graphes. Un morphisme de graphe peut être défini formellement comme une fonction totale de $V(G)$ vers $E(G)$ préservant la relation d'adjacence :

Définition 3.2. *Morphisme de graphes [Coq 1]*

```
Class morphism (G G':Graph_t)(f: vertex->vertex) :=
  {
    V_morphism   :  Total_fun (V_of G) (V_of G') (Trivial_morphism f);
    V_respectful :  forall x y,
      Edge_of (mk_edge x y) G->Edge_of (mk_edge (f x) (f y)) G'
  }.
```

Un morphisme de fonction finie est une fonction préservant la valeur associée à chaque élément. En d'autres termes, soient m une fonction finie, on note $Inv(y, m)$ l'ensemble des éléments ayant pour image y par m. f est un morphisme de m vers m' si pour tout y, f préserve l'appartenance à $Inv(y, m')$. Ou encore, f est un morphisme de m vers m' si pour tout y, f est une fonction totale de $Inv(y, m)$ vers $Inv(y, m')$.

Définition 3.3. *Morphisme de fonctions finies*

```
Definition Inv elt y (m:t elt) : Ensemble E.t :=
  fun x=>  x ↦  y ∈ m.

Definition morphism elt (m m':t elt)
  (f:E.t->E.t)(M:Proper (E.eq==>E.eq) f) :=
  forall y, Total_fun (Inv y m) (Inv y m') M.
```

3.4 Relations de réétiquetages et calculs locaux

Dans cette section, nous proposons une sémantique des relations de réétiquetage localement engendrées en *CIC*. Nous utilisons par la suite deux formes de relations de réétiquetages. Dans les preuves de systèmes de réétiquetage nous faisons usage de relations de réétiquetage générales (définition 3.4). Dans les énoncés faisant appel à la notion de relation de réétiquetage localement engendrée, nous utilisons des relations de réétiquetage *localisées* (definition 3.5) *i.e.* des relations de réétiquetage paramétrées par un sommet (le centre de la boule). On notera que le passage de l'une à l'autre peut être fait en quantifiant sur le sommet. L'utilisation de la seconde forme facilite la définition des relations de réétiquetage localement engendrées. On remarque que les deux types de relations sont paramétrés par un graphe et deux étiquetages. En effet une relation de réétiquetage dépend en général de la topologie du graphe sur lequel elle est appliquée, d'où la nécéssité de mentionner celle-ci. Toutefois, cette topologie n'étant pas affectée par un réétiquetage, il n'est pas nécéssaire de préciser la topologie du graphe *après* une transition.

Définition 3.4. *Types des relations de réétiquetage*
Soit $\mathcal{L}$ un couple de types dont l'égalité est décidable. Une relation de réétiquetage est une relation de type :

$$Graph\_t \to \Sigma_L \to \Sigma_L \to Prop$$

Définition 3.5. *Types des relations de réétiquetages localisées*
Soit $\mathcal{L}$ un couple de types dont l'égalité est décidable. Une relation de réétiquetage localisée est une relation de type :

$$Graph\_t \to vertex \to \Sigma_L \to \Sigma_L \to Prop$$

On notera en général c le sommet auquel est appliquée une relation de réétiquetage localisée. Si $(G, c, s, s') \in R$ on dira qu'il existe un R-réétiquetage en c entre G_s et $G_{s'}$, que l'on note $G_s \xrightarrow[c]{R} G_{s'}$. On dira qu'une relation de réétiquetage localisée est *localement engendrée* si

- cette relation est stable par isomorphisme de boule,
- lorqu'un R-réétiquetage a lieu en un sommet c, ce réétiquetage ne change pas l'étiquetage en dehors de la boule centrée en c.

Définition 3.6. *Relation de réétiquetage localement engendrée[Coq 2]*
Soit R une relation de réétiquetage, R est localement engendrée ssi, pour tous graphes étiquetés G_s et $G_{s'}$, pour tout sommet c de G, si il existe un R-réétiquetage en c entre G_s et $G_{s'}$ alors :

- *tous sommets et arêtes en dehors de $B_G(c)$ conservent leurs étiquettes dans $G_{s'}$.*
- *Pour tous graphes étiquetés H_n et $H_{n'}$, pour toute fonction $\varphi \in V(G) \to V(H)$, si*
  - *φ est un isomorphisme entre $B_G(c)_s$ et $B_H(\varphi(c))_n$*
  - *φ est un isomorphisme entre $B_G(c)_{s'}$ et $B_H(\varphi(c))_{n'}$*

– *tous sommets et arêtes en dehors de* $B_H(\varphi(c))$ *conservent leurs étiquettes dans* $H_{n'}$.

alors il existe une transition de R *en* $\varphi(c)$ *entre* H_n *et* $H_{n'}$.

Cette définition diffère de celle courament trouvée dans la littérature. Considérons par exemple la définition proposée dans [59] :

Définition 3.7. *(Définition 1.6.3 page 22 de [59])*
Soit R une relation de réétiquetage. R est localement engendrée ssi, pour tous graphes étiquetés G_s, $G_{s'}$, H_t et $H_{t'}$, pour tous sommets $c \in V(G)$ et $c' \in V(H)$ tels que les boules $B_G(c)$ et $B_H(c')$ sont isomorphes via φ, avec $\varphi(c) = \varphi(c')$ et si :
(i) $\forall x \in B_G(c), s(x) = t(\varphi(x)) \wedge s'(x) = t'(\varphi(x)) \wedge$
(ii) $\forall x \notin B_G(c), s(x) = s'(x) \wedge$
(ii) $\forall x \notin B_H(c'), t(x) = t'(x)$,
alors $G_s \xrightarrow{R} G_{s'} \Leftrightarrow H_t \xrightarrow{R} H_{t'}$.

On remarque que certaines relations **LC0** échappent à cette définition des calculs locaux.

Remarque 3.1 (Informelle)**.** *Soient L un alphabet P un prédicat sur L. Soit R la relation de réétiquetage engendrée par la règle :*

R n'est pas localement engendrée si l'on considère la définition de [59].

Démonstration. Soient quatre graphes étiquetés G_s, $G_{s'}$, H_t et $H_{t'}$ tels que décris par la figure 3.2. On suppose que $\neg P(\alpha)$, $\neg P(\beta)$ et $P(\gamma)$. Nous choisissons $c = a$ et $c' = d$. On définit φ comme la fonction associant a à d et b à e. Les boules $B_G(a)$ et $B_H(d)$ sont évidement isomorphes. De même, les conditions (i), (ii) et (iii) de la définition sont vérifiées. Si R est une relation localement engendrée, on doit donc avoir l'équivalence $G_s \xrightarrow{R} G_{s'} \Leftrightarrow H_t \xrightarrow{R} H_{t'}$. Or ce n'est pas ici le cas puisque $G_s \xrightarrow{R} G_{s'}$ (car $P(\gamma)$) mais pas $H_t \xrightarrow{R} H_{t'}$ (car $\neg P(\alpha)$ et $\neg P(\beta)$). Donc R n'est pas localement engendrée d'après la définition de [59]. □

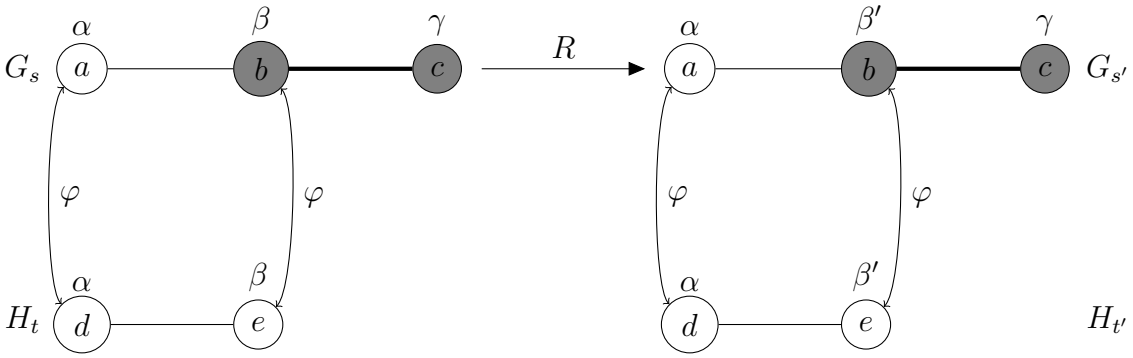


FIGURE 3.2 – Illustration de la preuve de la remarque 3.1.

On note que la définition 3.6 page 41 évite le problème souligné par la remarque 3.1 page précédente en précisant sur quels sommets sont appliqués les relations de réétiquetage. Dans notre implantation, nous nous assurons d'ailleurs formellement que toutes les relations de réétiquetage **LC0** sont des relations de réétiquetage localement engendrées, au sens de la définition 3.6 page 41.

Pour chacun des modes de synchronisation étudiés nous proposons un plongement dans l'ensemble des relations de réétiquetages localement engendrées. Ce plongement se divise en deux parties : la première est une transformation des relations originelles en une relation de réétiquetage de graphes, la seconde une preuve que les relations ainsi obtenues sont effectivement des relations de réétiquetage localement engendrées. Les résultats démontrés pour les calculs locaux sont ainsi valides pour chaque mode de synchronisation et peuvent être réutilisés dans les preuves de propriétés de ces derniers. Ces preuves permettent aussi de nous assurer de la correction des définitions formelles proposées.

Raisonner sur les transformations permet de démontrer des propriétés pour l'ensemble des relations d'un mode de synchronisation. Nous utilisons notamment cette technique pour faciliter la mécanisation de preuves concernant un mode en particulier. Pour simplifier, par exemple, les preuves de terminaison de systèmes de réétiquetage, nous fournissons pour chaque mode de synchronisation une preuve que la décroissance d'un variant local (une mesure) lors d'une transition assure la terminaison de ce dernier.

3.4.1 Calculs locaux sur les arêtes (LC0)

Nous proposons ici un type et un plongement pour les calculs locaux sur les arêtes étiquetées. On rappelle que les calculs de ce mode de synchronisation s'effectuent sur l'étiquetage d'un couple de sommets adjacents. Le nombre d'étiquettes pris en compte lors de calculs **LC0** étant fixe, nous représentons ceux-ci comme des relations liant l'étiquetage des deux sommets et de l'arête les reliant avant et après chaque pas de calcul. Plus formellement :

Définition 3.8. *Type des relations **LC0** [Coq 3]*
*Soient $\mathcal{L}_v$ et $\mathcal{L}_e$ deux types d'étiquettes. Une relation de réétiquetage **LC0** est une relation de type :*

$$\mathcal{L}_v \to \mathcal{L}_v \to \mathcal{L}_e \to \mathcal{L}_v \to \mathcal{L}_v \to \mathcal{L}_e \to Prop$$

dont les arguments représentent de gauche à droite l'étiquetage des deux sommets et celui de l'arête avant et après un pas de calcul.

La relation **LC0** R définie par $R\ A\ N\ false\ A\ A\ true$ correspond par exemple à la règle 1 page 23 de calcul d'arbre recouvrant.

Deux transformations de ce type sont fournies : la première vers le type des relations de réétiquetage de graphes [Coq 4] et la seconde vers les relations de réétiquetage de graphes *localisées* [Coq 5]. La première abstrait le sommet centre sur lequel le réétiquetage a lieu. La seconde est utilisée pour démontrer l'inclusion des relations de réétiquetages **LC0** dans les relations de réétiquetages localement engendrées [Coq 6]. Ces deux transformations ne posent aucune difficulté majeure.

3.4.2 Calculs locaux sur les boules ouvertes (LC1)

Nous considérons ici des relations affectant l'étiquetage d'une boule. Plus précisément, on rappelle que les relations **LC1** peuvent lire l'intégralité de l'étiquetage d'une boule, et modifier l'étiquetage du centre et des arêtes de la boule. Les étiquettes des sommets périphériques sont inchangées après un pas de calcul.

Pour représenter l'étiquetage des sommets d'une boule nous considérons ce que nous nommons une *couronne* [Coq 7] : une couronne contient l'ensemble des sommets d'une boule à l'exception du centre, ainsi que les étiquettes de ses sommets et des arêtes les reliant au centre. Intuitivement, une couronne est une fonction finie associant à chaque sommet v adjacent au centre c d'une boule son étiquette ainsi que l'étiquette de l'arête (c, v). Nous représentons une relation **LC1** par le type suivant :

Définition 3.9. *Type des relations **LC1** [Coq 8]*
*Soient $\mathcal{L}_v$ et $\mathcal{L}_e$ deux types d'étiquettes. Soit C le type des fonctions finies des sommets vers $(\mathcal{L}_v \times \mathcal{L}_e)$. Une relation de réétiquetage **LC1** est une relation de type :*

$$\mathcal{L}_v \to \mathcal{L}_v \to C \to C \to Prop$$

dont les arguments représentent de gauche à droite l'étiquetage du centre avant et après réétiquetage et la couronne avant et après réétiquetage.

Tel quel, ce type permet de représenter un ensemble de relations plus large que les relations **LC1**. Premièrement ce type mentionne l'étiquetage des sommets adjacents au centre après un pas de réétiquetage. Or, une relation **LC1** ne peut modifier ces étiquettes. Deuxièmement, rien ne spécifie dans le type que les modifications sont apportées uniquement sur des arêtes ayant pour extrémité le centre de la boule. Finalement, ce type mentionne les noms des sommets adjacents au centre de la boule. Or, une relation localement engendrée (et par extension une relation **LC1**) est, par définition, indépendante des noms de sommets.

Les deux premiers problèmes sont traités dans le plongement dans les relations de réétiquetages [Coq 9]. Cette transformation assure premièrement que les changements d'étiquetage des sommets de la couronne ne seront pas pris en compte ; deuxièmement que seules les étiquettes des arêtes adjacentes au centre de la boule peuvent être modifiées.

Le troisième problème est traité par la restriction des relations **LC1** par un prédicat. Plus précisément, une relation de type [Coq 8] est une relation **LC1** si celle-ci est stable par isomorphisme de couronne [Coq 10]. En restreignant ainsi les relations étudiées, on prouve que leur transformation en une relation de réétiquetage de graphes est bien une relation localement engendrée [Coq 11].

Une solution alternative pour empêcher la modification de l'étiquetage des sommets par une relation **LC1**, serait de considérer un type dont le dernier argument mentionnerait uniquement les étiquettes des arêtes. Néanmoins, dans la version proposée, les relations **LC1** et **LC2** partagent le même type, ce qui permet de réutiliser une grande partie de l'infrastructure mise en place pour les relations **LC1** pour les relations **LC2**. Les deux derniers problèmes peuvent être résolus en définissant le type des relations **LC1** comme un *type dépendant*. Ces solutions n'ont pas été explorées dans la formalisation accompagnant ce mémoire, mais sont des pistes d'améliorations considérée dans la version en cours de développement par P. Castéran.

3.4.3 Calculs locaux sur les boules (LC2)

Les relations de réétiquetages sur les boules ou relations **LC2** partagent leur type avec les relations **LC1**. On les restreints aussi, comme dans le cas des relations **LC1**, aux relations stables par isomorphisme de couronnes [Coq 12]. La seule différence notable avec les relations **LC1** réside dans le plongement dans les relation de réétiquetage de graphes [Coq 13] : ici on tient compte des changements d'étiquettes des sommets de la couronne.

3.5 Invariants

Nous exprimons les propriétés de systèmes de réétiquetage comme des *invariants* de ces systèmes. On rappelle qu'un invariant est une propriété des états d'un système toujours vérifiée au cours d'une exécution de ce dernier. Nous choisissons de paramétrer un invariant par un état initial. Cela permet, lors de preuves, de préserver certaines informations, comme par exemple un sommet ayant initié un calcul (racine d'un arbre) ou autres. Plus formellement :

Définition 3.10. *Invariant [Coq 14]*
Soient L un couple de types d'étiquette, G un graphe et I un ensemble d'états initiaux. On dira qu'une relation $J \in \mathcal{G} \times \Sigma_L \times \Sigma_L$ est un invariant d'un système de réétiquetage R depuis I ssi pour tout état i appartenant à I, pour tout état s accessible depuis i par R alors $(G, i, s) \in J$.

3.6 Tâches

Nous proposons ici un type *coq* pour les tâches telles que décrites par la définition 2.17 page 25. On rappelle qu'une tâche est composée d'un ensemble de graphes étiquetés initiaux et d'une relation entre un état initial et un état final. Nous considérons que les états initiaux et finaux peuvent être étiquetés par des alphabets différents. On note I (resp. O) le couple des types des étiquettes (sommets et arêtes) des états initiaux (resp. finaux).

Définition 3.11. *Type d'une tâche [Coq 15]*

```
Class Task:  Type :=  {
  Input   : Graph_t → Σ_I → Prop;
  BA      : Graph_t → Σ_I → Σ_O → Prop}.
```

L'ensemble des graphes étiquetés initiaux est représenté comme un prédicat sur un graphe et son étiquetage. Ce prédicat correspond à la notion de *connaissances à priori* : le degré de chaque sommet, l'existence d'un identifiant unique pour chaque sommet, etc. La relation "avant-après" entre graphes initiaux et finaux est représentée comme un prédicat sur un graphe et deux étiquetages pouvant être de types différents. Comme nous supposons que les systèmes étudiés sont des systèmes de réétiquetages, ceux-ci ne changent pas le graphe sous-jacent. Il n'est donc pas nécessaire de préciser de nouveau le graphe final dans le prédicat "avant-après". On remarque que le fait de repréciser le domaine de la tâche (`Input`) est redondant et n'est pas strictement nécéssaire. Cela s'avère toutefois pratique lorsqu'une tâche est paramétrée par une classe d'états initiaux (voir A.3 page 127).

On rappelle qu'un système de réécriture R réalise une tâche T s'il existe deux paires de fonctions (ι_v, ι_e) et (π_v, π_e) telles que l'image par (π_v, π_e) d'une forme normale G_s accessible par R depuis l'image par (ι_v, ι_e) d'un état $G_{i'}$ appartenant au domaine de T est en relation avec ce dernier. Nous pouvons formuler la notion de réalisation sous la forme d'un invariant :

Définition 3.12. *Invariant de réalisation[Coq 16]*
Soient T une tâche, $\iota = (\iota_v, \iota_e)$ et $\pi = (\pi_v, \pi_e)$. Soit $\widehat{(f_v, f_e)}(s)$ l'image par (f_v, f_e) d'un l'étiquetage s. On note $nf(R)$ l'ensemble des formes normales de R. Nous définissons l'invariant de réalisation J_r comme :
$J_r = \{(G, i, s) | \exists i' \in dom(T), \widehat{\iota}(i') = i \Rightarrow G_s \in nf(R) \Rightarrow (G_{i'}, G_{\widehat{\pi}(s)}) \in T\}$

Nous remplaçons, dans la formalisation, la quantification existentielle sur l'état du domaine d'une tâche par un paramêtre. Cette modification permet une manipulation plus aisée de la notion de réalisation dans les preuves formelles.

On remarque que le type des tâches est indépendant du mode de synchronisation considéré, ainsi que des types des étiquettes des sommets et arêtes utilisés par un hypothétique système de réétiquetage réalisant la tâche. Cela nous permet non seulement de réutiliser aisément des spécifications d'un système à l'autre, mais aussi de produire des preuves d'impossibilité générales. En effet, il est possible de produire des preuves d'impossibilité quelque soit la classe de système de réétiquetage considérée, mais aussi quelque soit le type des étiquettes des sommets. Cette partie sera discutée plus en détail par la suite (voir chapitre 4 page 57).

3.7 Modes de détection de la terminaison

Nous définissons ici les invariants de systèmes correspondant aux modes de détection de terminaison décrits précédemment (voir section 2.5 page 26). On rappelle que l'on note τ le *statut de terminaison* d'un sommet. τ est une fonction associant à une étiquette un booléen, valant *true* si la terminaison à été détéctée par le sommet et $false$ sinon. On note π_v la fonction associant à une étiquette sa *valeur de sortie*.

Cette représentation diffère quelque peu de la représentation choisie par Y. Métivier, E. Godard et G. Tel. Dans [47] ces derniers définissent l'étiquette d'un sommet comme un triplet $(mem, out, term)$, ou mem représente l'état courant d'un sommet, out représente la valeur de sortie du sommet et $term$ est le statut de terminaison, un booléen décrivant si oui ou non le sommet a détecté sa terminaison. D'expérience, cette notation nous est apparue redondante : la valeur de sortie et le statut de terminaison peuvent en général tout deux être déduits de l'étiquette d'un sommet.

Pour chaque mode de détection de terminaison nous définissons un invariant. Comme le reste des prédicats de spécifications (réalisation de tâche et terminaison) nous utilisons ceux-ci sous deux formes : Comme une obligation de preuve pour les systèmes "concrets" dont nous souhaitons prouver la correction, et comme hypothèse lorsque nous raisonnons sur la réalisabilité d'une tâche.

3.8 Preuves de systèmes de réétiquetage : un exemple commenté

Nous décrivons ici les techniques de preuve mises en place pour démontrer la correction totale d'un système de calculs locaux. Une liste détaillée des systèmes étudiés est fournie en annexe. Ces études de cas nous permettent de tester la correction de nos définitions sur des exemples concrets. Nous décomposons la preuve de correction d'un système en trois lemmes :

- Une preuve de réalisation de tâche c'est à dire une preuve que la projection de toute forme normale accessible depuis un état initial du système respecte le prédicat avant-après défini par la tâche à réaliser.
- Une preuve de terminaison c'est à dire une preuve que toute exécution du système depuis un état initial est finie.
- Une preuve que le mode de terminaison choisi est respecté.

Comme c'est en général le cas dans la littérature[78], les preuves de ces théorèmes seront effectuées par récurrence sur les exécutions. Pour illustrer la méthodologie mise en place, nous commentons un exemple simple : la preuve de correction totale d'un algorithme de calcul d'arbre recouvrant.

3.8.1 Spécifications

Nous commençons par définir en *Coq* la tâche associée à un calcul d'arbre recouvrant, que l'on nommera T_{span}, telle que décrite par l'exemple 1 page 25.

On rappelle qu'un sommet *racine* est connu *à priori*. Nous considérons des graphes initiaux simples, connexes et non nuls, dont les sommets sont étiquetés par des booléens. L'unique sommet étiqueté par *true* représente la racine de l'arbre à construire. Nous représentons l'abscence d'informations spécifiques sur les arêtes en supposant que celles-ci sont étiquetées par l'unique représentant du type singleton *unit*. Le type *st_in* est le type des états initiaux de T_{span} et le prédicat *Input* définit l'ensemble des états initiaux.

Exemple 2 Definition des états initiaux de la tâche T_{span}

```
Notation st_in := (state L_bool L_unit).

Definition Input  G (i:st_in ) := Not_Null G ∧
                                  Connected G  ∧
                                  For_one_vertex v of G, lambda  i v = true.
```

Nous considérons des états finaux où les sommets et arêtes peuvent être marqués ou non (étiquetages booléens). Nous représentons le sous-graphe construit comme le résultat de l'application du filtre sélectionnant uniquement les arêtes et sommets étiquetés par *true*, noté *marked_subgraph*. *Postcond* définit la relation avant-après de T_{span}. On remarque que dans ce cas précis, l'étiquetage final ne dépend pas de l'étiquetage initial, ce dernier reste donc inutilisé dans le prédicat *Postcond*.

Exemple 3 Définition de la relation avant-après de la tâche T_{span}

```
Notation   st_out := (state L_bool L_bool).

Definition marked_subgraph G (s_out : st_out) :=
   graph_simple_filter (fun b:bool ⇒ b) (fun b:bool ⇒ b) s_out G.

Definition Postcond  G (i : st_in)(s : st_out): Prop :=
   Spanning_Tree (marked_subgraph G s ) G.

Definition SP_Task := Build_Task  Input  Postcond.
```

3.8.2 Définition du système de réétiquetage

Le système de réétiquetage étudié est celui présenté par l'exemple 1 page 23. Pour correspondre à la définition de ce système de réétiquetage proposée par *ViSiDiA*, nous étiquetons les sommets par A (marqué) ou N (non marqué), et des arêtes étiquetées par des booléens. Par la suite nous considérons la "vue" résultant de l'application du filtre sélectionnant les sommets marqués A et les arêtes marquées N. Cette vue dépendant de l'étiquetage, celle-ci est dynamique, contrairement au graphe sous-jacent qui lui, est statique.

On rappelle que lors de chaque transition, un sommet du graphe et une arête sont marqués i.e. ceux-ci sont "ajoutés" à la vue en cours de construction. L'exécution se poursuit ainsi jusqu'à ce qu'aucun sommet du graphe ne puisse plus être marqué. [2]

Exemple 4 Définition d'un système **LC0** réalisant T_{span}

```
Inductive R0 : A_N → A_N → bool → A_N → A_N → bool → Prop :=
    R0_intro : R0 A N false A A true.
```

Nous obtenons une réalisation en définissant les fonctions d'initialisation et de projection du résultat pour correspondre à la tâche T_{span}. Ces définitions sont immédiates. à l'étiquette A (resp. N) est associée la valeur de sortie $true$ (resp. $false$). À chaque arête est initialement attribuée l'étiquette $false$. La fonction d'initialisation des étiquettes des sommets est l'inverse de la fonction de calcul du résultat. Dans le cas des arêtes celle-ci est simplement l'identité.

2. L'algorithme ViSiDIA correspondant peut être trouvé dans Spanning_Tree_Degree_LC0.java, dans les fichiers accompagnant ce mémoire.

Exemple 5 Fonctions d'initialisation et de calcul de la valeur de sortie associées au système de l'exemple 4 page ci-contre

```
Definition inj_v (b : bool) := if b then A else N.
Definition inj_e (_ : unit) := false.

Definition is_A (l:A_N):= match l with
                            | A ⇒ true
                            | _ ⇒ false
                          end.

Definition pi_v := is_A.
Definition pi_e (b:bool) := b.
```

3.8.3 Invariants et variants du système

La preuve du système peut être grossièrement résumée ainsi : nous considérons le sous-graphe des sommets étiquetés A et des arêtes étiquetées par $true$. Ce sous-graphe contient initialement un unique sommet (la racine). Il est donc initialement un arbre. A chaque pas de calcul, l'étiquetage d'un triplet sommet-sommet-arête devient $A-A-true$. Cette arête et ses deux extrémités appartient donc, dans le nouvel état, au sous-graphe calculé. Ce sous-graphe reste donc un arbre. Une fois le calcul terminé, tous les sommets sont marqués A et appartiennent donc à l'arbre calculé. Ce dernier est donc un arbre recouvrant. De cette ébauche de preuve nous déduisons les invariants suivants :

(1) Le filtre sélectionnant les sommets marqués A et les arêtes marquées $true$ est un arbre.

(2) Toute arête dont une extrémité au moins est étiqueté par N est étiquetée par $false$.

(3) Il existe un sommet marqué A.

Le premier invariant nous permet de montrer que nous calculons un arbre. Les invariants (2) et (3) permettent de montrer que lorsqu'un état irréductible est atteint, l'arbre calculé est recouvrant. Les invariants (1), (2) et (3) (dont les définitions en *Coq* sont données par l'exemple 6 page suivante) sont prouvés de manière usuelle i.e. on montre que ces derniers sont vrais pour tout état initial et conservés par tout pas de calcul du système de réétiquetage de graphe engendré par $R0$.

Il nous reste à prouver que toute exécution conduit à une forme normale, ou en d'autres termes que le système de réétiquetage généré par la (ou les dans le cas général) règles de calcul est nœtherien. Comme nous le proposions plus tôt, nous associons à chaque état une *mesure de progression* ou variant. Dans ce cas-ci nous choisissons un variant entier : à A nous associons la valeur 1 et à N la valeur 0. L'étiquetage des arêtes peut ici être

Exemple 6 Définitions *Coq* des invariants (1), (2) et (3).

```
(1) Tree(A_Subgraph s)
(2) For_each_edge e of G,
    λs(e_fst e) = N ∨ λs(e_snd e) = N → ρs(e) = false.
(3) For_some_vertex o of G, λs(o) = A
```

ignoré, nous associons donc 0 à toute étiquette des arêtes. Grâce aux résultats génériques associés aux plongements générant la relation de réétiquetage, la preuve de terminaison est triviale.

3.8.4 Format général de preuve

Quelque soit le mode de synchronisation étudié, une preuve de réalisation sera structurée de la manière suivante :

- On commencera par démontrer un ensemble de lemmes techniques permettant de simplifier les preuves d'invariants. En effet, ceux-ci seront en général définis de manière à être lisible mais leur manipulation dans des preuves peut s'avérer plus simple sous d'autres formes.
- Dans une première section, on considére un état initial G_i, et on montre que celui-ci respecte les invariants.
- Dans une sous-section dîte "courante", nous supposons l'existence d'un état G_s accessible depuis G_i pour lequel les invariants sont vérifiés.
- Dans une sous-section de la section "courante", nous supposons l'existence d'un troisième état $G_{s'}$ tel qu'il existe un pas de calcul par la relation de réétiquetage engendrée par la règle étudiée entre G_s et $G_{s'}$. Nous montrons que les invariants sont valides pour $G_{s'}$. Nous montrons également ici que le variant défini plus tôt décroît entre G_s et $G_{s'}$. Certaines propriétés liées aux détections de terminaison LTD et OTD seront démontrées ici.
- Dans une autre sous-section de la section "courante", nous supposons que G_s est une forme normale et montrons que celui-ci respecte la postcondition de la tâche considérée. Le reste des propriétés liées à la détection de terminaison est démontré ici.
- Dans une section de conclusion, les lemmes généraux sont appliqués pour déduire des résultats prouvés plus tôt la correction totale du système de réétiquetage engendré par les règles de calcul.

L'exemple 7 page 55 illustre par des extraits de la preuve du système de calcul d'arbre recouvrant le format de preuve mis en œuvre. Le schéma de découpage de preuve exposé ci-dessus s'inspire du schéma utilisé par la méthode *B-événementielle* [5]. Une spécification *B-événementielle*, où machine, se compose de différentes sections : variables, invariants, et

évènements. Un évènement spécifique correspond à l'initialisation de la machine. L'outil *Rodin* [6] génère, à partir d'une machine, les obligations de preuves assurant le respect de l'invariant au cours d'une exécution. La création d'un outil similaire à *Rodin* ou à *Why* [40], prenant en entrée la définition *Coq* d'un système de réétiquetage et de ses invariants, et générant une instance du schéma de preuve ci-dessus faciliterait les preuves de corrections de systèmes de réétiquetage en *Coq*.

3.9 Conclusion et travaux futurs

Dans ce chapitre nous avons définis, dans le calcul des constructions inductives, une sémantique relationnelle des systèmes de caluls locaux. En nous basant sur celle-ci, nous avons mise en place une méthodologie de preuve, que nous avons utilisée pour démontrer la correction de plusieurs systèmes de réétiquetages. La table 3.1 donne la liste des systèmes dont la correction à été démontrée en *Coq*. Pour chacun, nous précisons le mode de synchronisation, le mode de détection de la terminaison, et les connaissances initiales. On note "racine" si un unique sommet du graphe est distingué, "degrés" si le degré de chaque sommet est connu.

Tâche	Synchronisation	Terminaison	états initiaux
Degrés	**LC0**	ITD	uniformes
Degrés	**LC1**	LTD	uniformes
Arbre recouvrant	**LC0**	LTD	graphes connexes, racine
Arbre recouvrant	**LC0**	GTD	graphes connexes, racine, degrés
Arbre recouvrant	**LC1**	GTD	graphes connexes, racine
Election	**LC0**	GTD	arbres, degrés
Election	**LC1**	GTD	arbres

TABLE 3.1 – Récapitulatifs des preuves de réalisation mécanisées

Cette méthodologie de preuve n'est pas fondamentalement originale et peut être retrouvée dans la plupart des travaux concernant la preuve d'algorithmes, distribués ou non. Notre contribution principale réside ici plutôt dans la formalisation et l'étude des plongements de règles de calculs dans les systèmes de calculs locaux. Nous poursuivons cette étude dans les chapitres suivants, en nous intéressant plus précisément aux propriétés des plongements et en utilisant celles-ci pour démontrer à l'aide de *Coq* des résultats d'impossibilité.

Cette implantation relationnelle des calculs locaux permet d'étudier l'ensemble des exécutions possibles et est donc adaptée aux preuves de réalisation (et d'impossibilité comme nous le verrons par la suite). Il peut néanmoins être intéressant de considérer une implantation fonctionnelle des différents modes de synchronisation, ce qui permettrait de tester les systèmes de réétiquetage à la manière de [95] et de visualiser leur exécutions.

Un plongement de la sémantique fonctionnelle vers la sémantique relationnelle proposée ici pourrait être défini, permettant ainsi de réutiliser une preuve du modèle relationnel pour un objet fonctionnel. Pour compléter cette chaîne de développement d'algorithmes, il serait aussi intéressant de formaliser le passage du modèle des calculs locaux aux modèles par passages de messages [15].

L'étude des plongements des modes de synchronisation nous a montré qu'il existe un large potentiel pour l'automatisation des preuves d'invariants. Définir une tactique *Ltac* pour chaque mode de synchronisation résolvant certains buts courant (étiquettes inchangées en dehors d'une boule, etc) permettrait de rendre ces preuves plus concises. Ce genre de tactiques est illustré par une tactique permettant de prouver la terminaison des systèmes de réétiquetage **LC0**. Une solution alternative serait d'utiliser une logique de séparation [79] pour simplifier les raisonnements sur les modifications de l'état global.

Exemple 7 Squelette de la preuve du calcul d'arbre recouvrant

```
Section Proof.

  Variable G:Graph_t.
  Hypotheses (GG  : Graph G).
  Notation transition_at  :=   (LC0_Step_At R0 G).
  [...]
  Section Initials.

    Variable input : st_in.
    Hypothesis Hi : Input G input.
    Hypothesis Covers_input : Covers G input.
    Let i := inject SP_Algorithm input.
    [...]
    Section current.

      Variable s : st_mem.
      Hypothesis Hr : reachable_from (SP G) i s.
      Hypothesis Inv_s : Inv G i s.
      Hypothesis Cs : Covers G s.
      [...]
      Section Step.

        Variables ( s':st_mem)
                  (c v :vertex)
                  (H_c : Vertex_of c G)
                  (H_v : Vertex_of v G)
                  (Hcv : Adjacent G c v).
        Hypothesis Step: transition_at  c v s s'.
        Hypothesis Cs' : Covers G s'.
      [...]
      End Step.

      Section Irreducible.

        Hypothesis Hirr : irreducible transition s.
        [...]
      End Irreducible.
  [...]
  End current.
[...]
End Initials.
[...]
End Proof.
```

Chapitre 4

Étude Formelle de l'Expressivité de Classes de Calculs Locaux

Dans ce chapitre nous exposons la méthodologie mise en oeuvre pour démontrer formellement l'irréalisabilité d'une tâche par une certaine classe de calcul. Cette méthodologie repose sur des *invariants de classes*, des propriétés caractérisant l'expressivité d'une classe de systèmes. Nous introduisons deux nouveaux invariants de classes pour les systèmes de calculs sur les arêtes, inspiré par les travaux de J. Chalopin [23], et illustrons leur utilisation en fournissant de nouvelles preuves d'impossibilité pour les problèmes de l'élection, du calcul d'arbre recouvrant et du calcul du degré. Enfin nous présentons une formalisation d'une preuve de l'impossibilité d'élire dans un graphe quelconque par un système de calculs locaux, issue des travaux de D. Angluin [10] et de E. Godard et Y. Métivier [45].

Sommaire

4.1 Introduction

La puissance d'expression d'un ensemble de systèmes de calculs locaux est caractérisée par l'ensemble des tâches que celui-ci permet de réaliser. Nous avons vu, dans le chapitre précédent, qu'il nous était possible de démontrer formellement qu'un système de réétiquetage A réalise une tâche T. Ces preuves de réalisation nous fournissent une première indication sur l'expressivité de la synchronisation choisie pour le système témoin : nous répondons en effet ainsi à la question "Existe t'il un algorithme avec synchronisation X réalisant la tâche T ?".

Nous étudions ici la possibilité de prouver formellement qu'étant donnée une *classe* de système de calculs locaux $\mathcal{C}$ et une tâche T, aucun système de $\mathcal{C}$ ne réalise T. Pour ce faire, nous adoptons un schéma standard de preuve par l'absurde. Nous supposons l'existence d'un système A quelconque de la classe $\mathcal{C}$ réalisant T, puis construisons une *exécution fautive* de A. On rappelle qu'une *exécution correcte* de A vis à vis de T est une exécution de A dont l'état initial et l'état final respectent la relation *avant-après* de T. Notre objectif est donc ici de construire une exécution de A dont l'état initial et l'état final contredisent le prédicat avant-après de T.

Nous dépendons, pour construire une exécution fautive, de la synchronisation choisie, de l'ensemble d'états initiaux de la tâche considérée et du mode de détection de la terminaison supposé. Il s'agit de dégager, pour chaque synchronisation, une ou plusieurs propriétés rendant possible la constructions de telles exécutions. Un contre-exemple sera généralement construit comme suit : on considérera en premier lieu un algorithme A, que l'on suppose réaliser une tâche T avec un certain mode de détection de terminaison. On choisira ensuite un graphe adéquat vis à vis de la propriété de synchronisation à employer et supposerons l'existence d'une exécution correcte de A sur ce graphe. À l'aide de la propriété de synchronisation considérée et du mode de détection supposé, on construira ensuite un second graphe et une exécution de A contredisant l'hypothèse de réalisation.

Historiquement, la première de ces propriétés de synchronisation (ou invariants de classes) est le lemme de *relèvement*, tel que défini par Angluin [10] et adapté par la suite par E. Godard et Y. Métivier aux calculs locaux [45]. Ce premier lemme permet de démontrer qu'il n'existe pas de système de calcul local permettant d'élire dans tout graphe. Nous proposons, pour les calculs locaux sur les arêtes étiquetés, deux invariants de classe. Ces deux lemmes, dit *d'extension* et de *composition*, s'inspirent des travaux de J. Chalopin [26, 23] concernant l'expressivité des différentes classes de calculs locaux. Nous utilisons le premier pour démontrer l'irréalisabilité du calcul du degré avec un système de calcul sur les arêtes, ainsi que l'impossibilité de construire un arbre recouvrant avec détection globale de la terminaison. Le lemme de composition, un corollaire du lemme d'extension, nous permet de définir un outil proche du lemme de relèvement spécifique aux systèmes de calcul sur les arêtes. Nous verrons néanmoins que ces deux résultats, bien que d'utilisation similaire, concernent des classes de graphes différentes.

4.1.1 Organisation du chapitre

Nous commençons par étudier la réalisabilité de problèmes simples dans le cadre de systèmes **LC0**, tels que le calcul du degré de chaque sommet, ou la construction d'un arbre recouvrant. À cette fin, nous introduisons un invariant des systèmes **LC0** : le lemme d'extension. Nous continuons par étudier la réalisabilité de l'élection : nous présentons tout d'abord une formalisation du lemme de relèvement et du résultat d'impossibilité général d'élection pour l'ensemble des systèmes de calculs locaux. En nous basant sur un corollaire du lemme d'extension, nous proposons une preuve alternative de l'impossibilité d'élire un leader par un système **LC0**. Nous généralisons enfin de manière informelle (sans preuve *Coq*) ces résultat, en restreignant la classe des graphes dans laquelle l'élection est possible pour la synchronisation **LC0**.

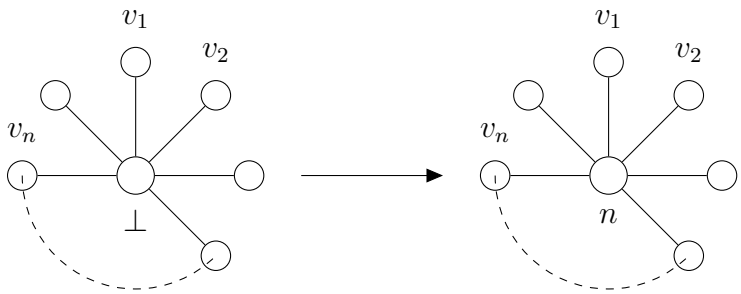
4.2 Impossibilité de calculer le degré en synchronisation LC0 avec détection de la terminaison LTD

Nous considérons ici la tâche T_d de calcul du degré de chaque sommet. Nous partons d'un état initial *quelconque*, et souhaitons obtenir un graphe où chaque sommet est étiqueté par son degré. Cette tâche est réalisable sans détection de la terminaison (voir section A.1.1 page 119). Démontrer l'irréalisabilité avec détection de la terminaison de cette tâche par un système de calcul sur les arêtes n'en reste pas moins d'une importance particulière : en effet, J. Chalopin a démontré [23] que tout système de calcul sur les arêtes avec connaissance a priori du degré est équivalent à un système de calcul sur les étoiles. En d'autres termes, si cette tâche était réalisable par un système de calcul sur les arêtes avec détection de la terminaison (que ce soit GTD, OTD ou LTD), alors les systèmes de calcul sur les arêtes seraient équivalents aux systèmes de calcul sur les étoiles.

De plus réaliser une telle tâche avec détection locale de la terminaison locale à l'aide un système sur les étoiles est trivial (voir règle 4 page suivante). Le résultat d'irréalisabilité que nous fournit le théorème 4.1 est donc une preuve alternative du fait que les systèmes de calculs sur les arêtes sont strictement moins puissants que les systèmes de calcul sur les étoiles.

Comme décrit dans l'introduction nous procédons, pour démontrer le théorème 4.1 page suivante, par une preuve par l'absurde. Nous supposons l'existence d'un algorithme D de calcul sur les arêtes réalisant T_d avec détection locale de la terminaison locale. La démonstration se base sur cette simple observation : un sommet ne peut, avec un système de calcul sur les arêtes, embrasser la totalité de son voisinage en un unique pas de calcul. Considérons par exemple un sommet, sans connaissances topologiques a priori, comptant et marquant ses voisins un à un. La sémantique des calculs sur les arêtes assure que, même si le prochain sommet choisi a déjà été marqué, rien ne permet de décider qu'il n'existe plus aucun sommet non marqué dans son voisinage. En d'autres termes, un calcul sur

Règle 4 Un système de réétiquetage **LC1** réalisant la tâche T_d avec détection locale de la terminaison locale. Celle-ci est détectée si l'étiquette du sommet est différente de $\perp$.



les arêtes peut "oublier" une partie du voisinage d'un sommet. Cette observation peut être généralisée pour tout système R de calcul sur les arêtes : si une séquence de pas de réétiquetage par R est applicable dans un graphe étiqueté G_s, alors celle-ci peut être appliquée à l'identique dans tout graphe étiqueté H_n dont G_s est un sous-graphe.

Lemme 4.1. ***Extension*** *[Coq 17]*
Soient R un système de calcul sur les arêtes, et G_s et H_n deux graphes étiquetés tels que G_s est un sous-graphe étiqueté de H_n. Pour tout état $G_{s'}$ accessible depuis G_s par R, il existe un état $H_{n'}$ accessible depuis H_n dont $G_{s'}$ est un sous-graphe. De plus, tout sommet de H n'appartenant pas à G conserve son étiquetage originel dans $H_{n'}$.

Le lemme 4.1 est un invariant de classe nous permettant de construire une preuve simple du théorème 4.1. Cet invariant caractérise plus précisément que le *lemme de relèvement* l'expressivité des calculs locaux sur les arêtes, comme nous le verrons par la suite (c.f. section 4.4 page 63). La formalisation en *Coq* de la preuve du théorème 4.1 suit le schéma de la figure 4.2.

Théorème 4.1. *[Coq 18]*
Il n'existe pas d'algorithme de calcul sur les arêtes réalisant T_d avec détection locale de la terminaison locale.

Démonstration. Soit D un système de réétiquetage que l'on suppose réaliser la tâche T_d avec détection locale de la terminaison locale. On suppose de plus que toute exécution de D est finie. Partant du graphe étiqueté G_s de la figure 4.1 page ci-contre, nous pouvons construire à l'aide de ces hypothèses une exécution correcte de D. Nous déduisons de l'hypothèse de réalisation que cette exécution transforme G_s en un graphe étiqueté $G_{s'}$ ou chaque sommet est étiqueté par son degré (en l'occurrence 1). Nous savons en outre que chaque sommet, dans un état final, a détecté sa propre terminaison. Considérons maintenant le graphe H_t, dont G_s est un sous-graphe. Par le lemme 4.1, il existe un état $H_{t'}$, dont $G_{s'}$ est un sous-graphe, tel que $H_{t'}$ est accessible par D depuis H_t. Or nous savons que pour tout sommet de $G_{s'}$, et donc de $H_{t'}$, le degré calculé est 1 et la terminaison à été détectée. Le degré calculé ne changera donc plus, ce qui résulte en une forme normale ou le

degré calculé pour le sommet u du graphe H sera 1. Hors, H_t étant un état initial pour T_d, le fait que le degré calculé pour u soit incorrect contredit l'hypothèse de réalisation. □

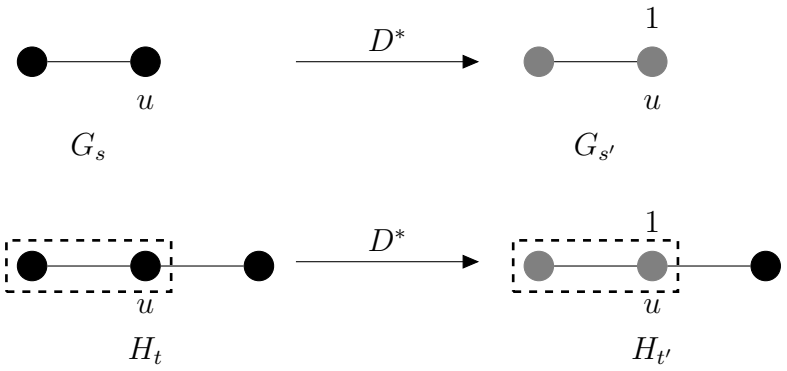


FIGURE 4.1 – Impossibilité de calculer le degré avec détection locale de la terminaison locale : les sommets gris ont détecte leur terminaison.

Ce raisonnement dépend à la fois de l'ensemble d'états initiaux de la tâche et du mode de détection de terminaison considéré. Nous proposons une caractérisation plus précise de la classe d'états initiaux pour laquelle la preuve ci-dessus reste valide. Ce théorème n'a pas été démontré en *Coq*, mais sa preuve est une simple généralisation à celle du théorème 4.1.

Théorème 4.2 (Informel)**.** *Soit I_d l'ensemble d'état initiaux de T_d, si, pour tout graphe étiqueté H_t appartenant à I_d, il existe un sous-graphe strict G_s de H_t appartenant à I_d, alors T_d est irréalisable par un système de calcul sur les arêtes avec détection locale de la terminaison locale.*

Démonstration. On suppose que T_d est réalisable par un système de calcul **LC0** pour l'ensemble d'états initiaux I_d. Soit H_t un graphe appartenant à I_d. D'après notre hypothèse, il existe un sous-graphe étiqueté stricte G_s de H_t appartenant à I_d. Il existe donc un état irreductible $H_{t'}$ accessible depuis H_t où le degré de chaque sommet à été calculé et où chaque sommet à détecté la terminaison. Par le lemme 4.1, il existe donc un état $H_{t'}$ accessible depuis H_t pour lequel : (i) le degré calculé pour chaque sommet x est égale à $D_G(x)$, (ii) la terminaison locale à été détecté. La terminaison locale ayant été détecté dans $H_{t'}$, dans toute forme normale $H_{t''}$, accessible depuis $H_{t'}$, le degré calculé pour chaque sommet x sera donc égal $D_G(x)$. Hors G est un sous-graphe stricte de H, donc il existe un sommet x tel que $D_G(x) \neq D_H(x)$, pour lequel le degré calculé est incorrect. L'hypothèse de réalisation est donc contredite. □

Cette classe d'ensembles d'états initiaux exclu la connaissance *à priori* du degré, ainsi que les connaissances topologiques n'étant pas stables pour la relation *être un sous-graphe*. Par contre celle-ci n'exclu pas l'étiquetage de chaque sommet par un identifiant unique.

```
Remark extension : Extends G H s t.

Section Extension.

   Variable (s' t': state Lv Le)
   Hypothesis Reach: reachable_from  (D G)  s s'.
   Hypothesis Reach: reachable_from  (D G)  t t'.
   Hypothesis Irr  : irreducible (tr (D G)) s'.
   Hypothesis extension' : Extends G H s' t'.

   Let u := 0.

   Remark tau_u_s'   : tau (lambda s' u) = true.

   Remark proj_u_s'  : pi_v (lambda x u)  = degree G u.

   Remark tau_u_t'   : tau (lambda t' u) = true.

   Remark proj_u_t'  : pi_v (lambda t' u) = degree G u.

   Remark proj_u_f   : forall f,
     reachable_from (D H) t' f →
     irreducible (tr (D H)) f →
     pi_v (lambda f u ) = degree H u.

   Remark proj_u_f'  : forall f,
     reachable_from (D H) t' f →
     irreducible (tr (D H)) f →
     pi_v (lambda f u) = degree G u.

End Extension.

Theorem degree_not_LC0_computable : False.
```

FIGURE 4.2 – Extrait de la formalisation en *Coq* de la preuve du théorème 4.1

En ce qui concerne le mode de détection de la terminaison, Y. Métivier *et al.* ayant démontré [47] l'inclusion du mode *LTD* dans tous les autres modes, nous en déduisons qu'il est impossible de résoudre T_d avec détection de la terminaison, quelque-soit le mode choisi. On note que notre implantation *Coq* contenant la preuve de ces inclusions, il est aussi simple d'en déduire une preuve formelle.

4.3 Impossibilité de calculer un arbre recouvrant en synchronisation LC0 avec détection de la terminaison GTD

La méthodologie de preuve développée pour l'exemple ci-dessus peut être réutilisée à l'identique pour étudier la réalisabilité d'une autre tâche : le calcul d'un arbre recouvrant, avec détection locale de la terminaison globale. Nous noterons cette tâche T_s. Les états initiaux de T_s sont les états dans lesquels un unique sommet est marqué. Les états finaux de T_s sont définis comme les graphes étiquetés dont les sommets et arêtes marqués forment un arbre couvrant. Divers systèmes de calcul sur les arêtes (voir section 3.8 page 48), réalisent cette tâche. Néanmoins, ceux-ci détectent au plus la terminaison locale. Nous montrons ici, en utilisant le lemme 4.1 page 60, qu'aucun système de calcul sur les arêtes ne peut réaliser cette tâche avec détection locale de la terminaison globale. À notre connaissance, cette preuve n'apparaît pas dans d'autres travaux.

Théorème 4.3. *[Coq 19]*
Il n'existe pas d'algorithme de calcul sur les arêtes réalisant T_s avec détection locale de la terminaison globale.

Pour démontrer le théorème 4.3, nous procédons de la même manière que dans le cas du calcul du degré.

Démonstration. Nous considérons le graphe étiqueté G_s de la figure 4.3 page suivante, dont un unique sommet est marqué. On suppose l'existence d'un système S de calcul sur les arêtes réalisant T_s. Nous pouvons construire une exécution de S menant depuis G_s à un état $G_{s'}$, une forme normal pour S. D'après l'hypothèse de détection de terminaison, il existe un sommet de $G_{s'}$ ayant détecté la terminaison globale. Nous considérons maintenant le graphe H_t de la figure 4.3 page suivante, dont G_s est un sous-graphe. On note que seul le sommet marqué dans G_s est marqué dans H_t, ce qui fait de H_t un état initial pour T_s. Par le lemme 4.1 page 60, il existe un graphe étiqueté $H_{n'}$ accessible par S depuis H_t dont $G_{s'}$ est un sous-graphe et dont les sommets n'appartenant pas à G ont conservé leur étiquette initiale. Ces sommets n'étant pas initialement marqués, l'arbre des sommets et arêtes marquées de $H_{t'}$ ne peut être *recouvrant*. Or, un sommet a détecté la terminaison globale dans $G_{s'}$, et par extension dans $H_{t'}$. $H_{t'}$ est donc, d'après la définition de la détection locale de la terminaison globale, une forme normale pour S, accessible depuis un état initial H_t pour T_s dont les sommets et arêtes marqués ne forment pas un arbre recouvrant, ce qui contredit l'hypothèse de réalisation. □

4.4 Élection

Nous étudions maintenant le problème de l'élection pour différentes classes de calculs locaux. Nous considérons une variante du problème pour laquelle aucune connaissance

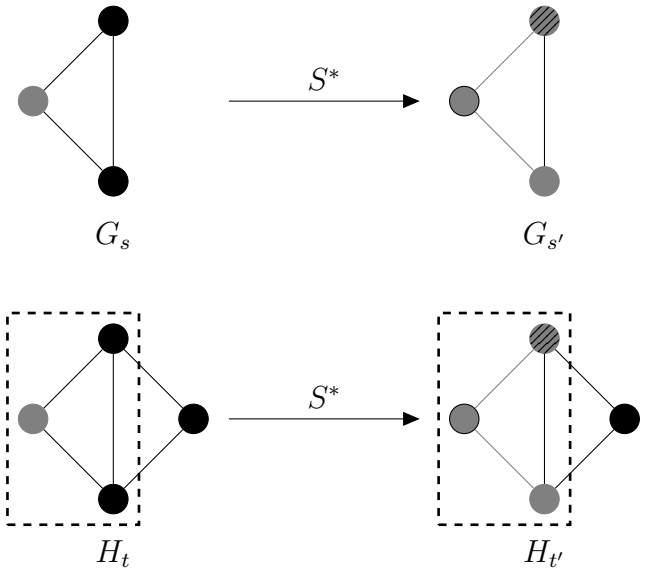


FIGURE 4.3 – Impossibilité de calculer un arbre couvrant avec détection globale de la terminaison.

topologique n'est initialement disponible.

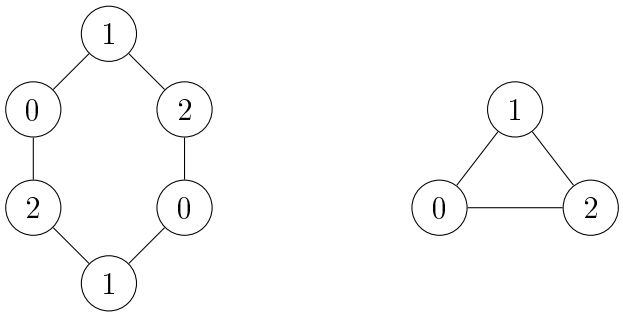
Nous commençons par proposer une formalisation du lemme de relèvement et de la preuve d'irréalisabilité de l'élection dans les graphes non minimaux pour les revêtements dans le cadre des calculs locaux.

Nous étudions ensuite le cas de l'élection dans les arbres par les systèmes de calcul sur les arêtes. Le choix de cette classe de graphe n'est pas fortuite : en effet les arbres sont *minimaux* pour les revêtements, et ne sont donc pas des candidats pour l'application du lemme de *relèvement*. Nous lui substituons un corollaire du lemme d'*extension*. Ce problème a déjà été étudié par J. Chalopin dans sa thèse[23]. La preuve d'irréalisabilité proposée se base néanmoins sur la notion d'isomorphisme, qui, comme nous le montrons, n'est pas nécessaire et ne reflète pas clairement selon nous les propriétés sous-jacentes aux systèmes de calcul sur les arêtes permettant l'obtention de ce résultat.

Dans la suite, sauf indication contraire, nous considérons le problème de l'élection tel que défini dans [45]. L'ensemble des états initiaux pouvant varier, ceux-ci ne sont pas précisés ici.

Tache 2 T_e : Election

Les états finaux consistent en un étiquetage des sommets par *battu* ou *élu*. Une fois étiqueté *élu* ou *battu*, un sommet ne peut changer d'étiquette. Un état final contient un unique sommet élu.



H G

FIGURE 4.4 – H est un revêtement stricte de G [10]

4.4.1 Une formalisation en *Coq* des résultats d'Angluin

Nous rappelons la forme usuelle [10, 45, 26] d'une preuve d'irréalisabilité de l'élection. Comme précédemment, on suppose l'existence d'un système de calculs locaux réalisant une élection. On considérera ensuite un graphe et deux de ses sous-graphes, en général isomorphes, puis on montrera qu'il existe une exécution menant à l'élection d'un sommet dans un de ces sous-graphes, et donc par symétrie, dans l'autre. Une exécution est ainsi construite où deux sommets sont élus, ce qui va à l'encontre de la définition de la tâche de l'élection.

Le raisonnement ébauché ci-dessus se base sur la notion de "voisinages similaires", correspondant à la notion formelle de revêtement. On rappelle qu'un revêtement est un morphisme de graphe surjectif localement bijectif (Definition 2.5 page 18). On nommera revêtement *strict* un revêtement n'étant pas un isomorphisme de graphe. La figure 4.4 illustre la notion de revêtement strict. On dira qu'un graphe est *minimal* pour les revêtements (et par extension pour les revêtements stricts) s'il n'existe aucun revêtement depuis ce graphe vers un autre.

Définition 4.1. *Graphe minimal pour les revêtements (stricts)*
Un graphe étiqueté G_s est dit minimal pour les revêtements (stricts) s'il n'existe aucun graphe H_n tel que G_s est revêtement (strict) de H_n.

La notion de graphe minimal pour les revêtements stricts caractérise une classe de graphes dans laquelle l'élection est impossible[10, 23]. Nous prouvons en *coq* un résultat plus faible : la non existence d'un système de calcul local réalisant l'élection pour n'importe quel état initial.

Théorème 4.4. *[Coq 20]*
Il n'existe pas de relation de réétiquetage localement engendrée réalisant universellement la tâche T_e.

Nous proposons dans la suite une formalisation de ce théorème et du résultat principal permettant sa démonstration : le lemme dit de *relèvement*.

4.4.1.1 Lemme de relèvement

Le lemme de relèvement [10, 23] peut être énoncé ainsi :

Lemme 4.2. *Lemme de relèvement [Coq 21]*
Soient $H_t, H_{t'}$ et G_s trois graphes étiquetés. Soit φ un revêtement de G_s vers H_t. Soit R une relation de réétiquetage localement engendrée. Si $H_t \xrightarrow{R} H_{t'}$ alors il existe un graphe étiqueté $G_{s'}$ tel que
- *$G_s \xrightarrow{R^*} G_{s'}$ et*
- *φ est un revêtement de $G_{s'}$ vers $H_{t'}$.*

La figure 4.5 page ci-contre illustre ce lemme et donne une ébauche de sa preuve. Intuitivement, le lemme de relèvement dit que toute séquence de pas de réétiquetage applicable sur une boule d'un graphe étiqueté H_n est applicable à chaque boule d'un graphe G_s étant revêtement de H_n. Nous définissons un revêtement en *coq* comme un morphisme de graphe surjectif (épimorphisme), qui, restreint aux boules, est un isomorphisme :

Définition 4.2. *Revêtement [Coq 22]*
Une fonction φ est un revêtement d'un graphe G vers un graphe H ssi :
- *φ est un épimorphisme de G vers H.*
- *pour tout sommet c de H les boules $B_H(c)$ et $B_G(\varphi(c))$ sont isomorphes.*

Un revêtement de graphe étiqueté est à la fois un revêtement de graphe et un morphisme pour les étiquetages [Coq 23]. La définition 2.5 page 18 diffère quelque peu de la définition formelle proposée, celle-ci permettant des preuves plus directes que la première. La bibliothèque *Coq* construite propose en effet de nombreux lemmes sur les épimorphismes et isomorphismes de graphe (inverse, etc) qui seraient plus difficilement utilisables ("destruction" de définitions, etc) si la définition 2.5 page 18 était implémentée "telle-quelle". On peut toutefois aisément vérifier que ces deux définitions sont équivalentes.

Remarque 4.1 (informelle)**.** *Soit G et H deux graphes quelconques.*
- *(i) φ est un morphisme de G vers H et*
- *(ii) $\forall x \in V(H), \exists y \in V(G), \varphi(y) = x$ et*
- *(iii) $\forall c \in V(G), \forall v\ v' \in B_G(c), \varphi(v) = \varphi(v') \Rightarrow v = v'$*

équivaut à
- *(iv) φ est un épimorphisme de G vers H et*
- *(v) pour tout sommet c de H les boules $B_H(c)$ et $B_G(\varphi(c))$ sont isomorphes.*

Démonstration. La conjonction des prédicats (i) et (ii) équivaut au prédicat (iv) ; le prédicat (v) est équivalent à la conjonction des prédicats (i) et (iii). □

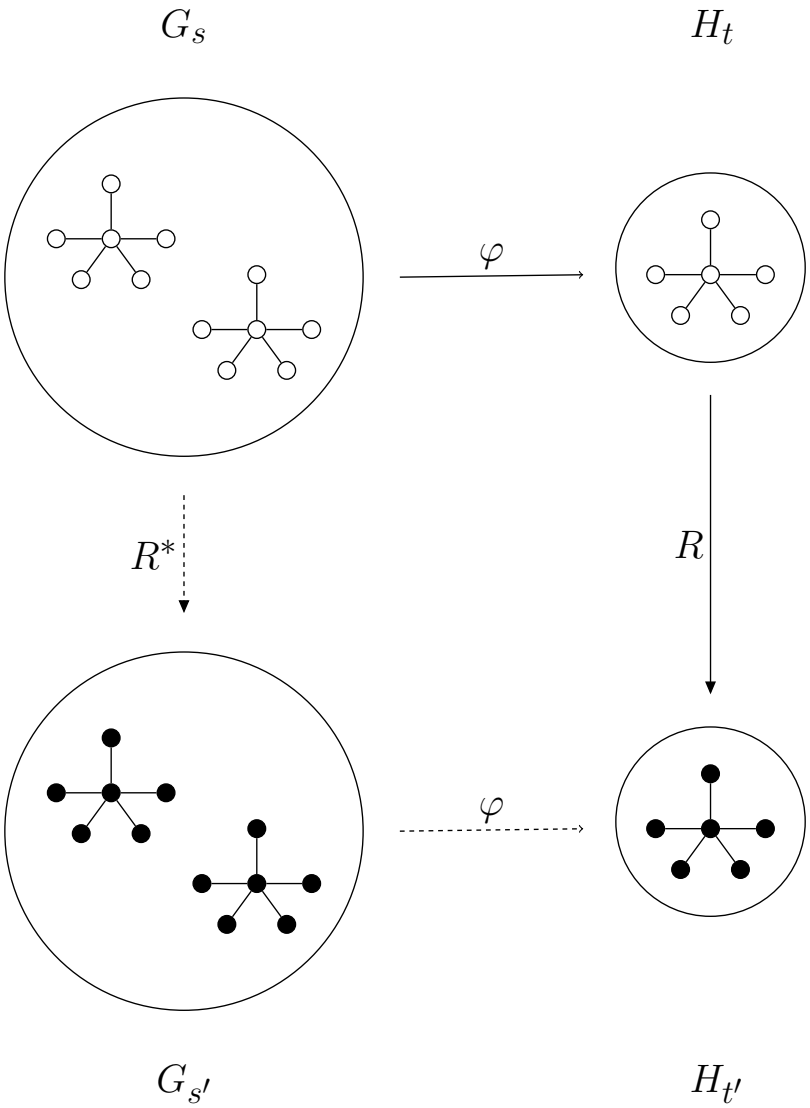


FIGURE 4.5 – Lemme de relèvement : les transformations induites par un pas de R sur une boule de G_s peuvent être reproduites sur toutes les images inverses de cette boule dans H_t. Le graphe étiqueté $H_{t'}$ ainsi obtenu est un revêtement de $G_{s'}$ et est accessible depuis H_t par R.

Nous définissons un revêtement strict comme suit :

Définition 4.3. *Revêtement strict [Coq 24]*
Une fonction φ est un revêtement strict d'un graphe G vers un graphe H ssi :
- *φ est un revêtement de G vers H.*
- *tout sommet de H possède au moins deux antécédents par φ dans G.*

Comme dans le cas de la définition 4.2 page précédente, la définition implantée en *Coq* diffère de la définition standard. La définition implémentée permet en effet un raison-

nement plus direct sur le nombre d'antécédents d'un sommet, et simplifie ainsi les preuves d'impossibilité (déduire que deux sommets sont élus est immédiat). Encore une fois, les deux définitions sont équivalentes.

Remarque 4.2 (informelle)**.** *Soit φ un revêtement de G vers H, (i) φ n'est pas un isomorphisme G et H ssi (ii) tout sommet de H a au moins deux antécédents par φ dans G.*

Démonstration.
$(i) \Rightarrow (ii)$: par contraposée. S'il existe au moins un sommet de H ayant un unique antécédent dans G, alors chaque sommet de H possède un unique antécédent dans G (cf. [23] proposition 2.6 page 20). φ est donc un isomorphisme.
$(ii) \Rightarrow (i)$: de la même manière. □

Nous considérons par la suite trois graphes étiquetés $H_n, H_{n'}$ et G_s, un sommet c ainsi qu'une fonction φ tels que cette dernière est un revêtement de G_s vers H_n et $H_n \xrightarrow[c]{R} H_{n'}$.

Nous construisons maintenant un nouvel état s' de sorte que $G_{s'}$ soit accessible depuis G_s et que φ soit un revêtement de $G_{s'}$ vers $H_{n'}$. On rappelle que nous considérons des graphes *finis*. L'égalité sur les sommets est de plus décidable (ceux-ci étant représentés par des entiers). Il nous est donc possible de calculer l'ensemble des sommets de G ayant pour image un sommet c de H par φ. On notera $\varphi_G^{-1}(c)$ cet ensemble (et par extension $\varphi_G^{-1}(x,y)$ l'ensemble des images inverses d'une arête (x,y) de H). Nous définissons s' comme l'application des changements d'étiquettes de n' à s. Plus formellement :

$$[\text{Coq 25}] \quad s' = \{x \mapsto y | \forall z \in B_H(c), x \in \varphi_G^{-1}(z) \wedge n'(z) = y\} \uplus s$$

φ est bien un revêtement de $G_{s'}$ vers $H_{n'}$ [Coq 26] : en effet, les graphes G et H ne changeant pas, φ reste un revêtement de G vers H ; de plus, de par le fait qu'un pas de R en c ne modifie pas n en dehors de $B_H(c)$ et de par la définition de s', φ est un morphisme d'étiquetage de s' vers n'. Nous déduisons du fait que φ est un revêtement de $G_{s'}$ vers $H_{n'}$ que toute boule étiquetée $B_G(v)_{s'}$ dont le centre v appartient à $\varphi_G^{-1}(c)$ est isomorphe à $B_G(c)_{n'}$ [Coq 27] (de même pour s et n [Coq 28]). De par la définition des calculs locaux nous déduisons donc qu'un pas de calcul est possible dans chacune de ces boules :

Lemme 4.3. *[Coq 29]*
Soit v un sommet quelconque appartenant à $\varphi_G^{-1}(c)$. On note $s'' = (s'|B_G(v)) \uplus s$. Un pas de calcul de R en v est possible depuis G_s vers $G_{s''}$.

De plus nous savons que l'intersection de boules dont les centres sont les images inverses d'un même sommet par un revêtement sont disjointes [Coq 30]. Nous en déduisons donc que $G_{s'}$ est accessible depuis G_s en appliquant un pas de R sur chacune des boules dont le centre appartient à $\varphi_G^{-1}(c)$ [Coq 31]. $G_{s'}$ est donc accessible depuis G_s et un revêtement de $H_{n'}$, ce qui prouve le lemme de relèvement.

4.4.1.2 Application à l'élection

Nous appliquons maintenant le lemme 4.2 page 66 pour démontrer le théorème 4.5. Deux preuves *Coq* en sont proposées. La première [Coq 20] montre que s'il existe une relation de réétiquetage localement engendrée réalisant la tâche T_e pour n'importe quel état initial, alors, pour tout graphe non minimal pour les revêtements stricts, il existe une exécution de R contredisant l'hypothèse de relation de T_e. Celle-ci peut être résumée comme suit :

Théorème 4.5. *[Coq 20]*
Il n'existe pas de relation de réétiquetage localement engendrée réalisant la tâche T_e pour tout graphe.

Démonstration. On suppose l'existence d'une relation R réalisant la tâche T_e pour n'importe quel graphe étiqueté initial. Soit G_s et H_n deux graphes étiquetés tels que G_s soit un revêtement strict de H_n par une fonction φ. Par hypothèse, il existe une exécution de R sur H_n menant à l'élection d'un sommet [Coq 32]. Or G_s étant un revêtement strict de H, par le lemme 4.2 page 66, il existe une exécution de R sur G_s menant à l'élection de deux sommets [Coq 33], ce qui contredit l'hypothèse de réalisation de T_e [Coq 20]. □

La seconde preuve [Coq 34] est une application du théorème 4.5 au graphe de la figure 4.4 page 65. Cette dernière nous assure de la correction de nos définitions, particulièrement en ce qui concerne les revêtements.

On notera qu'on ne prouve pas ici que l'élection est impossible pour la classe des graphes non minimaux pour les revêtements, mais uniquement que ceux-ci sont des contres-exemples à l'existence d'une relation localement engendrée d'élection pour tout graphe. Compléter la preuve formelle revient à prouver que l'élection est impossible pour l'ensemble des graphes non minimaux :

Théorème 4.6 (informel)**.** *([45] Proposition 12)*
Soit R une relation de réétiquetage localement engendrée réalisant T_e pour l'ensemble des graphes non minimaux pour les revêtements stricts. Alors pour tout graphe G n'étant pas minimal pour les revêtements stricts, il existe une exécution de R dans G conduisant à l'élection de deux sommets.

La formalisation en *Coq* du théorème 4.6 reste à effectuer, néanmoins celle-ci correspond en grande partie à la preuve du théorème 4.5 et ne pause aucun problème majeur. Il n'est pas possible dans ce cas d'assumer l'existence d'une exécution de R conduisant à un sommet élu dans le graphe dont G est revêtement. En effet, il est possible que ce graphe soit minimal pour les revêtements et ne correspond donc pas à l'ensemble des graphes pour lesquels R réalise T_e. Pour formaliser le théorème 4.6, il sera donc nécessaire de démontrer en *Coq* l'inverse du lemme de relèvement (lemme 4.4 page suivante), démonstration qui selon nous ne posera pas de problèmes majeurs.

Lemme 4.4 (informel). *Soient H_n et $G_s, G_{s'}$ trois graphes étiquetés. Soit φ un revêtement de G_s vers H_n. Soit R une relation de réétiquetage localement engendrée. Si $G_s \xrightarrow{R} G_{s'}$ alors il existe un graphe étiqueté $H_{n'}$ tel que*

- *$H_n \xrightarrow{R^*} H_{n'}$ et*
- *φ est un revêtement de $G_{s'}$ vers $H_{n'}$.*

Démonstration. Découle trivialement de la définition des calculs locaux. □

4.4.2 Synchronisation LC0 et élection

Nous avons étudié, dans la section précédente, la réalisabilité de l'élection par une relation de réétiquetage localement engendrée. Nous nous intéressons maintenant à la réalisabilité de l'élection par un système de calcul sur les arêtes. J. Chalopin et Y. Métivier [26] ont montré que l'élection est irréalisable par un système de calcul sur les arêtes dans un graphe non minimal pour les revêtements. Dans sa thèse de doctorat [23] (Prop. 3.44, p. 76), J. Chalopin utilise l'exemple de l'élection pour démontrer que les systèmes **LC0**, sans connaissance à priori du degré, sont strictement moins puissants que les systèmes **LC0** avec connaissance du degré. Il démontre pour ce faire l'irréalisabilité de l'élection par un système de calcul sur les arêtes sans connaissance du degré.

Nous proposons ici une version simplifiée et formelle de la preuve de J. Chalopin reposant sur un corollaire du lemme 4.1 page 60. Finalement, nous élargissons la classe de graphes étiquetés pour laquelle l'élection est irréalisable par un système de calculs locaux sur les arêtes.

4.4.2.1 Un contre-exemple

Nous commençons par montrer qu'il n'existe pas de système de calcul sur les arêtes réalisant l'élection pour tout graphe étiqueté. Pour illustrer la classe de graphe que nous définirons par la suite nous considérons un contre-exemple précis, et non pas une application du théorème 4.5 page précédente. Nous étudions les exécutions d'un hypothétique algorithme d'élection dans la *chaîne* de taille 4, représentée par la figure 4.6 page ci-contre. Pour ce faire nous utilisons un corollaire du lemme 4.1 page 60 :

Lemme 4.5. *Corollaire du lemme 4.1 page 60 [Coq 35]*
Soient R un système de calcul sur les arêtes et G_s un graphe étiqueté. Soient H_m et I_n deux sous-graphes étiquetés disjoints de G_s. Soient m' et n' tels que $H_m \xrightarrow{R^*} H_{m'}$ et $I_n \xrightarrow{R^*} I_{n'}$. Alors il existe s' un état de G tel que $G_s \xrightarrow{R^*} G_{s'}$ et $H_{m'}$ et $I_{n'}$ sont deux sous-graphes étiquetés de $G_{s'}$.

Le lemme 4.5 permet d'exhiber, à la manière du lemme de relèvement, une exécution fautive d'un système de calcul sur les arêtes supposé réaliser T_e. Nous pouvons donc prouver d'une manière alternative (et plus simple d'après notre expérience) qu'il n'existe pas de système de réétiquetage réalisant l'élection pour n'importe quel graphe étiqueté.

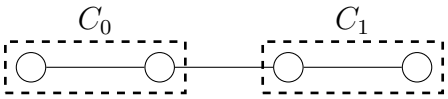
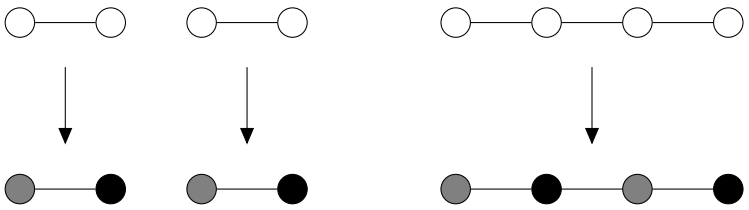


FIGURE 4.6 – La chaîne de taille 4 contient deux sous chaînes C_0 et C_1.

La preuve peut être résumée ainsi : considérons la chaîne de taille quatre de la figure 4.6 ; celle-ci est composée de deux chaînes disjointes de taille deux (C_0 et C_1). Supposons maintenant qu'il existe un système de réétiquetage sur les arêtes R réalisant la tâche T_e à partir de n'importe quel état. Alors il existe, pour C_0 et C_1 une exécution de R menant à l'élection d'un sommet dans chaque chaîne (Fig 4.7a). Par application du lemme 4.5 page ci-contre, il existe donc une exécution de R sur la chaîne de taille 4 menant a l'élection de deux sommets [Coq 36] ce qui contredit l'hypothèse de réalisation de T_e.



(a) Exécutions de R sur C_0 et C_1

(b) Application du lemme 4.5 page ci-contre

FIGURE 4.7 – Application du lemme 4.5 page ci-contre à la chaîne de taille 4. Les sommets noirs sont élus, les sommets gris sont battus.

4.4.2.2 Une nouvelle classe de graphe où l'élection est irréalisable par un système LC0

Nous proposons ici une généralisation du contre-exemple ci-dessus. En nous basant sur celui-ci, nous pouvons définir une classe de graphes étiquetés différente des graphes non minimaux pour les revêtements stricts où la tâche T_e de l'élection est irréalisable avec détection locale de la terminaison locale. Cette partie n'est pas prouvée en *Coq*. Il nous semble toutefois intéressant de mentionner cette conséquence des lemmes issus des travaux de formalisations accomplis.

Théorème 4.7 (informel)**.** *Soit $\mathcal{I}$ une classe de graphes étiquetés de plus de quatre sommets stables par passage aux sous-graphes. L'élection est irréalisable dans $\mathcal{I}$ par un système de calcul sur les arêtes.*

Démonstration. On considère un système de calcul sur les arêtes noetherien et décidable R réalisant T_e pour tout graphe de $\mathcal{I}$. Soit G_s un graphe étiqueté de $\mathcal{I}$. Nous savons qu'il existe au moins deux sous-graphes étiquetés F_l et H_n de taille supérieure ou égale à deux (une règle **LC0** y est donc applicable). Par hypothèse, F_l et H_n appartiennent à $\mathcal{I}$, il existe

donc deux graphes étiquetés $F_{l'}$ et $H_{n'}$. Dans chacun, un sommet est élu. Par application du lemme 4.5 page 70, il existe donc un graphe étiqueté $G_{s'}$ accessible depuis G_s dans lequel deux sommets sont élus. Or, nous savons que les sommets élus ne peuvent changer de statut. De plus, R étant noetherien et décidable, il existe une forme normale G_f pour R accessible depuis $G_{s'}$ dans laquelle deux sommets sont élus, ce qui contredit l'hypothèse de réalisation de T_e. □

Ce résultat nous permet de montrer que même avec des étiquetages initiaux "forts", l'élection est irréalisable par un système de calcul sur les arêtes. Considérons par exemple un graphe où chaque sommet est étiqueté par un identifiant unique : ceux-ci appartiennent à la classe $\mathcal{I}$ et l'élection au sens traditionnel n'y est pas réalisable par un système de calcul sur les arêtes.

Théorème 4.8 (informel)**.** *T_e est irréalisable par un système de réétiquetage sur les arêtes pour les graphes dont chaque sommet est étiqueté par un identifiant unique.*

Démonstration. Soit G_s un graphe où chaque sommet est étiqueté par un identifiant unique. Alors pour tout sous-graphe H_t de G_s, chaque sommet de H_t est étiqueté par un identifiant unique. G_s appartient donc à $\mathcal{I}$. Par application du théorème 4.7 page précédente, T_e y est donc irréalisable. □

Nous montrons ainsi que $\mathcal{I}$ est un ensemble de graphe étiqueté différent des graphes non minimaux pour les revêtements stricts. En effet, les graphes dont chaque sommet est étiqueté par un identifiant unique sont minimaux pour les revêtements.

Lemme 4.6 (informel)**.** *Soit G_s un graphe dont chaque sommet est étiqueté par un identifiant unique. G_s est minimal pour les revêtements stricts.*

Démonstration. On suppose qu'il existe un graphe H_t dont G_s est revêtement strict par une fonction φ. Soit v un sommet de H_t étiqueté par x. Comme G_s est revêtement strict de H_t, $|\varphi^{-1}(v)| \geq 2$. Il existe donc deux sommets de G_s dont l'étiquette est x. Contradiction. □

4.5 Conclusion et Perspectives

Nous avons montré dans ce chapitre que l'assistant de preuve *Coq* nous a permis d'implanter une preuve d'impossibilité bien connue dans le domaine de l'algorithmique distribuée : l'inexistence d'un algorithme universel d'élection. À notre connaissance, l'implantation proposée est la première preuve formelle du résultat de D. Angluin.

Cette implantation repose de plus sur la formalisation de concepts fondamentaux de la théorie des graphes tels que les morphismes, isomorphismes de graphes et revêtements. Ces derniers seront, nous l'espérons, réutilisés et modifiés dans de nouvelles preuves d'impossibilités et/ou d'autres études formelles de la théorie des graphes.

4.4.5 Conclusion et Perspectives

L'utilisation d'un assistant de preuve nous a permis en outre de mettre au jour de nouvelles propriétés de la synchronisation **LC0**. Ces dernières ont été utilisées dans trois preuves d'impossibilités. Les preuves d'impossibilités formalisées sont résumés dans le tableau 4.1.

Enfin, la bibliothèque mise en place nous permet de raisonner sur les spécifications de deux façons : soit en prouvant que celles-ci sont réalisables, soit en démontrant leur irréalisabilité. À notre connaissance, notre bibliothèque est la seule à permettre de manipuler dans un unique environnement formel les preuves de corrections, concrètes et les d'impossibilités, abstraites.

Tâche	Synchronisation	Terminaison	états initiaux
Calcul des degrés	**LC0**	LTD	uniformes
Arbre Couvrant	**LC0**	GTD	racine
Élection	**LC0**	LTD	uniformes
Élection	*	LTD	*

TABLE 4.1 – Récapitulatifs des preuves d'irréalisabilité mécanisées. "racines" représente la distinction à priori d'un sommet. "*" symbolise n'importe quel mode de synchronisation et n'importe quelle forme d'états initiaux.

Chapitre 5

Transformations des Systèmes de Calculs Locaux et de leurs Preuves de Correction

Nous étudions ici les transformations de systèmes de calculs locaux. Nous montrons qu'il est possible de transformer les preuves existantes de systèmes en utilisant les notions de *simulations en avant* et de *commutation* des règles de calcul. Nous utilisons ces outils, ainsi que des opérateurs sur les relations **LC0**, afin de démontrer formellement la correction de la transformation des relations **LC0** *GTD* en relations **LC0** *OTD* proposé par Métivier *et al.*

Sommaire

5.1 introduction

Nous avons étudié, dans les deux chapitres précédents, la possibilité d'utiliser l'assistant de preuve *Coq* pour formaliser les preuves de réalisabilité et d'irréalisabilité de tâches par des systèmes de calculs locaux. Le système de types de *Coq* nous permet de manipuler ces preuves comme des objets de premier ordre et de les passer en paramètre à d'autres objets. Cette possibilité est utile dans deux cas que nous étudions ici : lorsque l'on souhaite démontrer la correction d'un algorithme proche d'un algorithme certifié existant, ou lorsque l'on souhaite définire une *transformation certifiée de système de réétiquetage*. Cette transformation est une fonction prenant en paramètre un système de réétiquetage et sa preuve de correction et retournant un nouveau système, accompagné d'une nouvelle preuve de correction.

Pour illustrer le premier cas, nous considérons une relation de réétiquetage réalisant l'élection dans un arbre. Il est souhaitable de ne pas avoir à refaire la totalité de la preuve de correction lorsque cette relation est modifiée pour s'appliquer à un arbre recouvrant d'un graphe connu à priori.

A cette fin, nous étudions et combinons plusieurs techniques pour réutiliser des preuves existantes, et en particulier la notion de *forward simulation* telle qu'utilisée dans les travaux de N. Lynch et autres [63, 72] sur les automates d'entrée-sortie. Nos travaux s'inspirent aussi des travaux d'Abrial [4], D. Méry, D. Cancell, M. Mosbah et M. Tounsi [73, 93, 68] sur le *raffinement*, une forme de simulation. Dans ces articles, D. Méry, D. Cancell et M. Tounsi utilisent le raffinement pour dériver d'une spécification abstraite et globale, un système de calcul localement engendré correct. La notion de spécification abstraite qu'ils considèrent correspond grossièrement à la notion de tâche telle que nous l'avons exposée plus tôt.

Les travaux exposés ci-dessus se concentrent sur les preuves de correction. Les techniques de formalisation que nous avons mises en œuvre nous permettent d'utiliser les transformations de systèmes de réétiquetage et de leurs preuves pour étudier l'expressivité des calculs locaux. Par exemple la transformation d'un système GTD en un système OTD telle que définie par E. Godard, Y. Métivier et G. Tel dans [47] fournie une preuve constructive du fait que toute tâche réalisable par un système GTD l'est aussi par un système OTD. Nous proposons une implantation de cette dernière à l'aide des bibliothèques *Coq* décrites plus tôt.

5.1.1 Comparaisons avec les travaux relatifs

Les travaux de M. Tounsi [73, 93, 68], se basant sur la méthode *B événementielle* [5], visent à produire, à partir d'une spécification abstraite et globale, une relation de réétiquetage localement engendrée. Pour ce faire, une forme de simulation, le *raffinement* est utilisée, pour produire des spécifications de plus en plus *déterministes* et de plus en

plus *locales*, en supprimant au fur à mesure les occurrences de variables globales. Là où M. Tounsi dérive une relation de réétiquetage localement engendrée en plusieurs pas de raffinement à partir d'une spécification abstraite et globale, correspondant dans notre cas à une tâche, nous montrons en une unique étape la correspondance d'une relation de réétiquetage avec une tâche. L'approche de M. Tounsi peut être décrite comme une approche "de bas en haut", la nôtre de "haut en bas". De plus, la logique sur laquelle se base B-événementiel ne nous semble pas permettre de formaliser des preuves d'impossibilités comme nous l'avons fait au chapitre précédent. Cette limitation est toutefois compensée par une automatisation des preuves supérieure à ce que nous avons produit avec *Coq*. On peut, pour résoudre ce problème, envisager la création d'un outil de type *Why* [41], permettant de générer des obligations de preuves aux systèmes de calculs locaux et résolvant si possible celles-ci à l'aide de prouveurs de théorèmes automatiques. Finalement le langage de modélisation du B-événementiel ne permet pas, à notre connaissance, la représentation des transformations de modèles au sein même du langage. Les travaux que nous présentons dans la suite sont donc difficilement réalisables en B-événementiel et/ou demanderaient un méta-raisonnement sortant du cadre de la logique sous-jacente. On note que l'établissement de *patterns* [14, 18] correspondant aux transformations certifiées que nous proposons par la suite pourrait être intéressant.

Parmi les travaux cités ci-dessus, les travaux de Mitra et Archer [72] définissent dans le système *PVS* [43] des stratégies de preuves permettant l'utilisant des notions de simulations et de raffinement au sein de l'environnement *TAME* [11, 12, 72] pour les automates d'entrée-sortie. Un de leurs objectifs principaux est de limiter les raisonnements sur le raffinement à l'application d'un ensemble fini de "pas" de preuves indépendant des deux automates considérés. Contrairement à la leur, notre approche de simplification des preuves de simulation (voir section 5.4 page 83) s'appuie explicitement sur la définition des relations de réétiquetages étudiées. Au delà des techniques de preuves utilisées, les méthodes de formalisation employées diffèrent aussi. Ces différences sont principalement dues à certaines restrictions techniques de *PVS* : ce dernier ne supportant pas le *polymorphisme paramétrique*, les auteurs de [72] ne peuvent définir de type pour les automates d'entrée-sortie. Ces derniers se basent alors, pour formaliser la notion de raffinement, sur la possibilité de paramétrer une théorie *PVS* par une autre, à la manière du système de modules de *Coq*. Dans notre cas, *Coq* supportant des types plus complexes, nous sommes capables - comme montré plus tôt- de définir un type polymorphe pour les relations de réétiquetage. Nous pouvons donc définir la simulation comme une propriété de deux relations de réétiquetages, sans faire appel au système de modules de *Coq*.

Toujours dans le domaine des automates d'entrée-sortie, Win et autres [95] proposent entre autres d'aider à la démonstration de simulation en exécutant deux automates d'entrée-sortie en même temps pour vérifier qu'une relation de simulation proposée par l'utilisateur est maintenue. Pour le moment nos travaux ne permettent pas l'utilisation de ce genre de techniques, les relations de réétiquetages étant représentées sous formes relationnelles, non exécutables. Toutefois, dans des travaux récents, P. Castéran définit

des règles de réétiquetage comme des fonctions partielles *Coq*. L'aboutissement de ces travaux, en combinaison avec l'outil de visualisation ViSiDiA permettrait de créer un outil se rapprochant des travaux de tendant vers les travaux de T. N. Win [95].

5.1.2 Organisation du chapitre

Nous commençons par présenter une adaptation de la "forward simulation" aux systèmes de calculs locaux. Nous appliquons celle-ci à un premier exemple : la réutilisation de la preuve de correction d'un système réalisant une élection dans un arbre pour prouver la correction d'un système réalisant une élection dans un graphe dont un arbre recouvrant est connu.

Nous montrons ensuite comment factoriser les preuves de simulation pour les systèmes **LC0** en définissant des transformations courantes de systèmes **LC0**. Nous utilisons finalement ces définitions pour prouver formellement que toute tâche réalisable par un système **LC0** avec détection locale de la terminaison globale est réalisable par un système **LC0** avec détection observée de la terminaison.

5.2 Simulation et relations de réétiquetage

La notion de simulation a été popularisée par les travaux de D. Park [80] et ceux de R. Milner [70]. Une simulation peut être grossièrement décrite ainsi : soit A et B deux systèmes, B simule A si tout comportement de B peut être "rapporté" à un comportement de A. De cette définition informelle nous faisons la déduction suivante : si toute exécution de A est correcte vis à vis d'une spécification alors toute exécution de B l'est aussi. Cette observation simple fait de la simulation un outil majeur de preuves de correction d'algorithmes et d'algorithmes distribués en particulier [63, 61, 95]. Une notion de simulation est utilisée par exemple dans [62] par N. Lynch pour démontrer la correction d'une variation du protocole du bit alterné. Les notions de simulation et de bisimulation jouent un rôle essentiel dans les travaux de Milner sur les algèbres de processus [71].

Plus récemment une forme de simulation, le *raffinement*, a été utilisé par J.R. Abrial et autres [4, 6] pour concevoir une approche de développement de programme *corrects par construction*. Cette approche "de bas en haut" consiste à dériver, à partir de spécifications abstraites, un programme correct par raffinements successifs. Chaque programme de la chaîne ainsi créé étant une simulation du programme précédent, cette méthode assure que le programme final obtenu implémente les spécifications abstraites originelles. Cette méthode a été utilisée par D. Méry et D. Cansell [17] pour certifier un algorithme de comptage de références. Dernièrement, M. Tounsi, D. Méry et M. Mosbah ont travaillé à l'adaptation de cette méthode à la construction de relations de réétiquetages localement engendrées [73, 93, 68, 92].

Dans le cadre de la bibliothèque *Coq* produite, nous introduisons la simulation non seulement pour simplifier la preuve de systèmes de réétiquetage, comme c'est le cas dans les travaux de M. Tounsi, mais aussi pour raisonner sur leurs transformations.

5.2.1 Definitions

Nous utilisons une adaptation de la *forward simulation* (simulation en avant) de Lynch et Vaandrager ([61], Page 10) aux relations de réétiquetage de graphe. La différence principale entre les deux définitions réside dans le fait qu'une transition d'un système de réétiquetage de graphe n'est pas étiquetée par une *action* comme c'est le cas pour les automates d'entrée-sortie.

Soient A et B deux relations de réétiquetage de graphe, et S une relation sur $\Sigma_A \times \Sigma_B$. On dit que B simule A par S si S est préservée pour la clôture réflexive et transitive de B par un pas de calcul par A (c.f. figure 5.1). Plus formellement :

Définition 5.1. *Simulation En Avant [Coq 37]*
On dit que B simule A par S (noté $B \leq_S A$) si pour tous graphes G et H on a :

$$\forall s\ s'\ t, G_s \xrightarrow{A} G_{s'} \wedge (G_s, H_t) \in S \Rightarrow \exists t', H_t \xrightarrow{B^*} H_{t'} \wedge (G_{s'}, H_{t'}) \in S$$

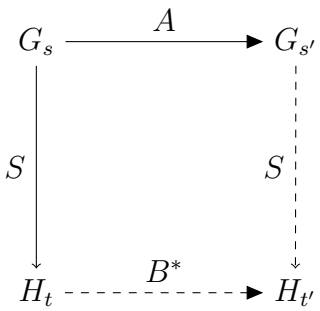


FIGURE 5.1 – Simulation

Nous étendons la notion de *simulation en avant* des relations de réétiquetages aux systèmes de réétiquetages.

Définition 5.2. *Simulation en avant et système de réétiquetage[Coq 37] :*
Soit $S_A = (\Sigma_A, I, A)$ et $S_B = (\Sigma_B, I', B)$ deux systèmes de réétiquetages. On dira, par extension de la définition 5.1, que S_B simule S_A pour une relation S (noté $S_B \leq_S S_A$) ssi $B \leq_S A$ et si, pour tous graphes H et G, tout état $H_{i'} \in I'$, il existe un état $G_i \in I$ tel que $(G_i, H_{i'}) \in S$.

5.2.2 Simulation et invariants

L'intérêt principal de la simulation réside dans la réutilisation des propriétés du système simulé pour démontrer la correction du système simulant. Considérons une propriété P_A du système simulé et une propriété P_B du système simulant. Si P_A "implique P_B modulo la relation de simulation", alors P_B est un invariant du système. Cette propriété de la simulation nous permet d'éviter les démonstrations par récurrence sur une exécution, potentiellement fastidieuses, exposées dans la section 3.8 page 48.

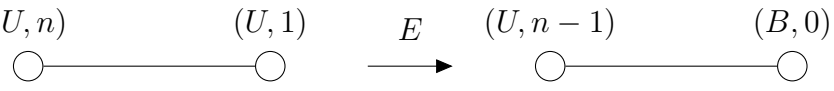
Théorème 5.1. *[Coq 38]*
Soient $S_A = (\Sigma_A, I, A)$ et $S_B = (\Sigma_B, I', B)$ deux systèmes de réétiquetages tels que $S_B \leq_S S_A$. Soient P_A un invariant de S_A et P_B une relation sur $\Sigma_B \times \Sigma_B$. Pour tous états G_s et H'_s accessible depuis un état $G_i \in I$ et $H_{i'} \in I'$ respectivement, si $(G_s, H_{s'}) \in S \wedge (G_i, G_s) \in P_A \Rightarrow (H_{i'}, H_{s'}) \in P_B$, alors P_B est un invariant de S_B.

Les propriétés de réalisation de tâches et de détection de terminaison étant exprimées sous formes d'invariants, le théorème ci-dessus permet de réutiliser ces dernières lors d'une transformation d'un système de réétiquetage. Comme décrit dans l'introduction de ce chapitre, cette forme de théorème est couramment mentionnée dans la littérature, et est principalement une adaptation des travaux de N. Lynch [61, 95] et J.R Abrial sur les méthodes B [4] et B-événementielle [6] aux relations de réétiquetages de graphe.

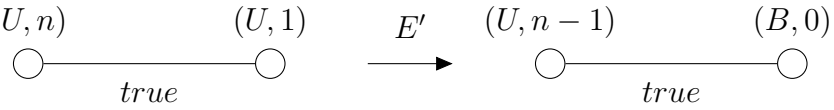
5.3 Simulation : une première application

Nous illustrons ici l'emploi de la simulation en vue de réutiliser la preuve d'un système de réétiquetage concret. Nous considérons l'exemple, mentionné en introduction, de deux systèmes de réétiquetage : le premier réalisant une élection dans un arbre, le second réalisant une élection dans un graphe connexe pour lequel un arbre recouvrant est connu *à priori*. Les systèmes de réétiquetage correspondants sont décrits par les règles 5 et 6 page suivante. Dans le cas du système de réétiquetage défini par la règle E, les états initiaux sont des arbres où chaque sommet est étiqueté par son degré. Dans le cas du système défini par E', les états initiaux sont des graphes connexes où chaque sommet est étiqueté par son degré, et le résultat de l'application du filtre sélectionnant les arêtes marquées est un arbre recouvrant.

Règle 5 Élection dans un arbre



Règle 6 Élection dans un graphe connexe dont un arbre recouvrant est connu



On rappelle le fonctionnement du système E (voir le fichier `Election_Tree_LC0.java`) : chaque sommet est étiqueté initialement par son degré. Les feuilles (i.e. les sommets étiquetés par 1) sont élagués, et le degré de leur voisin est diminué. Finalement l'unique sommet restant dont l'étiquette est zéro est élu. Le système E' se comporte de manière similaire sur l'arbre recouvrant. La figure 5.2 page ci-contre donne un exemple d'exécution pour chaque système.

On aura remarqué que les exécutions de la figure 5.2 page suivante sont "équivalentes" si l'on considère uniquement le sous graphe engendré par les arêtes marquées dans la seconde. De fait, nous pouvons définir une relation de (bi)simulation S_e entre les deux relations de réétiquetages :

Exemple 8 Relation de simulation entre l'election dans un arbre et l'election dans un sous arbre recouvrant [Coq 39]

G_s et $H_{s'}$ sont en relation par S_e ssi les composantes de s et s' concernant les sommets sont égales et si le résultat de l'application du filtre selectionnant les arêtes marquées à $H_{s'}$ est égale au graphe G.

Il est aisé de vérifier que la relation S_e de l'exemple 8 est bien une simulation (voir figure 5.2 page suivante). Une fois ce fait établi, et en supposant prouvée la correction de la relation de réétiquetage E, la preuve de E' revient grossièrement à appliquer le théorème 5.1 page précédente.

Utiliser la simulation nous permet de raccourcir significativement les preuves. Nous pouvons évaluer le gain en comparant le nombre de lignes de code *Coq* nécessaire à la démonstration de E (sans simulation) (957) avec le nombre de lignes nécessaires à la preuve de E' (539), le raisonnement dans les deux cas étant quasiment identique. La taille de la preuve est donc divisée par deux en utilisant la simulation. Toutefois, dans ce cas particulier, les deux preuves étant quasiment identiques, un meilleur résultat pourrait probablement être obtenu en améliorant la technique utilisée. L'utilisation des techniques de factorisation de preuve que nous présentons ci-dessous permettrait sans doute de simplifier encore la démonstration.

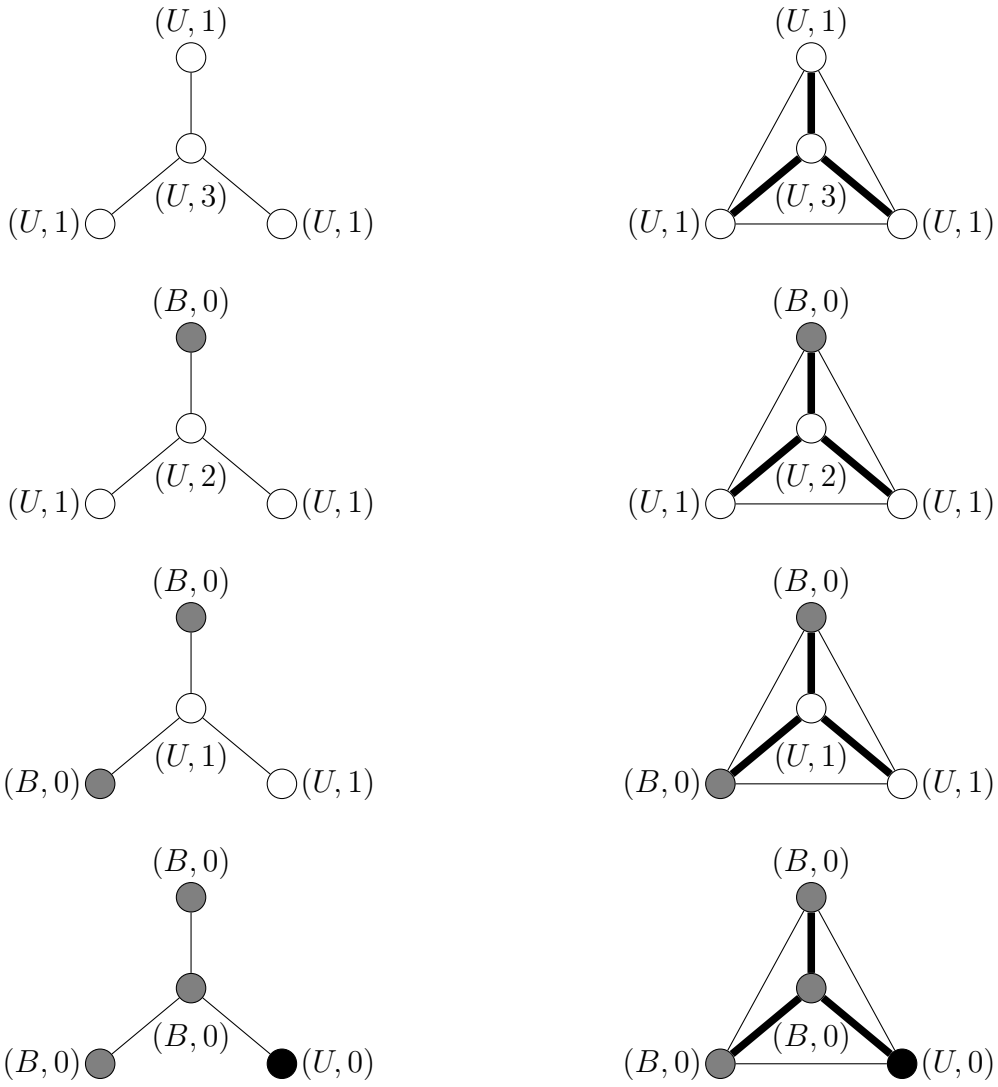


FIGURE 5.2 – Exécutions des relations d'élection dans un arbre et d'élection dans graphe dont un arbre recouvrant est connu.

5.4 Transformations courantes de règles LC0

Nous avons, dans la section précédente, décrit la notion de simulation implémentée en *Coq* et montré comment l'utiliser dans le cadre de deux systèmes de réétiquetage "intuitivement proches". L'utilisation de la simulation peut être simplifiée à l'aide *d'opérateurs de transformation*, implémentant les transformations courantes pouvant être appliquées à une relation de réétiquetage. Cette standardisation des transformations permet de factoriser les preuves de simulations.

Nous proposons par la suite une illustration de ce concept en décrivant six transformations courantes de règles **LC0**. Pour chacune de ces transformations, un ensemble de lemmes concernant la simulation est fourni. Une représentation de ces lemmes sous forme de règles d'inférence est fournie. Nous utilisons ces résultats dans les cas traités par la suite : la transformation de toute relation **LC0** avec détection locale de la terminaison globale en une relation **LC0** avec détection de la terminaison observée (voir section 5.5 page 87), et dans la définition d'opérateurs de composition de relations **LC0** (voir chapitre 6 page 93).

On note que ces opérateurs représentent les cas de transformation souvent traités par la suite. Ils ne visent pas à être exhaustifs, mais à permettre la factorisation des preuves de simulations.

5.4.1 Identité LC0

On notera dans la suite *Skip* la règle **LC0** d'identité, ne produisant aucun changement sur l'étiquetage . Cette dernière sera utilisée en combinaison avec les transformations décrites. On remarque que l'identité simule n'importe quelle relation **LC0**.

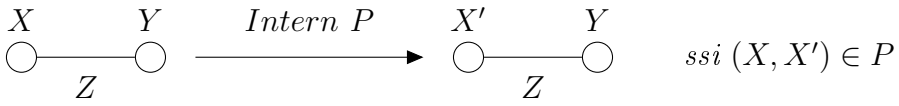
Lemme 5.1. *Simulation et identité. [Coq 40]*
*Soient R et R' deux relations **LC0**. Si un pas de R simule un pas de Skip par la relation S, alors un pas de R simule un pas de R' par S.*

$$Skip\ \frac{R \leq_S Skip}{R \leq_S R'}$$

5.4.2 Transition interne

Une *transition interne* représente le changement d'étiquetage d'un sommet sans consultation ni modification de celui d'aucun voisin. Cette construction n'est pas une transformation de règle à proprement parler mais une facilité syntaxique. Elle permet de factoriser les preuves de non modification du voisinage. On prendra soin de ne pas confondre cette forme de règle avec les transitions dites "silencieuses" que l'on peut trouver dans d'autres types de systèmes de transitions : si une transition interne n'accède pas à l'étiquetage de son voisinage, celle-ci n'en reste pas moins une règle **LC0** et nécessite l'existence d'au moins un voisin pour s'appliquer.

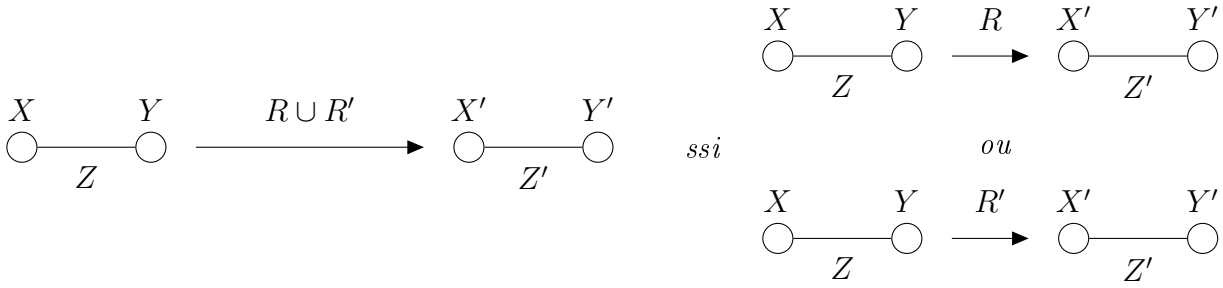
Définition 5.3. *Transition interne [Coq 41]*



5.4.3 Union

La première transformation définie ici correspond à l'union de relations. Sa définition correspond à la définition ensembliste de l'union. On notera $R \cup R'$ l'union de deux relations **LC0** R et R'.

Définition 5.4. *Union de relation **LC0** [Coq 42]*



Une relation de réétiquetage étant en général définie comme l'union de plusieurs règles de calculs, le lemme 5.2 permet de diviser la preuve de simulation d'une relation R'' par un système R en des preuves concernant les composants de R et d'appliquer "récursivement" les résultats associés aux transformations de relation **LC0**.

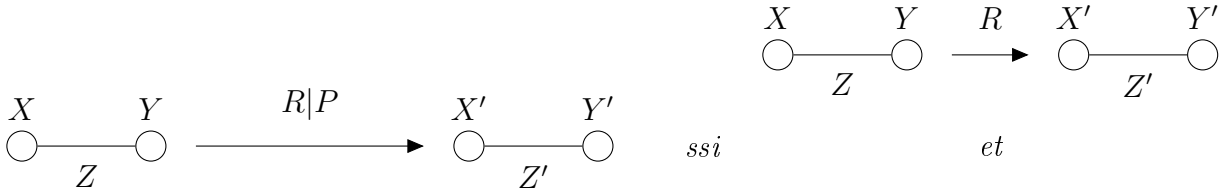
Lemme 5.2. *Union et simulation [Coq 43]*
*Soient R, R' et R'' trois relations **LC0** telles que R et R' manipulent le même type d'états. Soit S une relation entre état de R (et de R') et état de R''. Si un pas de R simule un pas de R'' par S et si un pas de R' simule un pas de R'' par S alors un pas par l'union de R et de R' simule un pas de R par S.*

$$Union\ \frac{R \leq_S R'' \qquad R' \leq_S R''}{R \cup R' \leq_S R''}$$

5.4.4 Renforcement de garde

La seconde transformation présentée ici, dite *renforcement de garde* ajoute un condition à l'application d'une règle. Cette condition est représentée par un prédicat sur les étiquettes des deux sommets adjacents et de l'arête avant réétiquetage. Cet opérateur permet de distinguer *par définition* la garde d'une règle.

Définition 5.5. *Renforcement de garde [Coq 44]*



$$(X, Y, Z) \in P$$

Lemme 5.3. *Simulation et renforcement de garde [Coq 45]*
*Soient R et R' deux relations **LC0** telles qu'un pas de R' simule un pas de R par une relation S. Alors, pour toute condition Q, Un pas de $R'|Q$ simule un pas de R par S.*

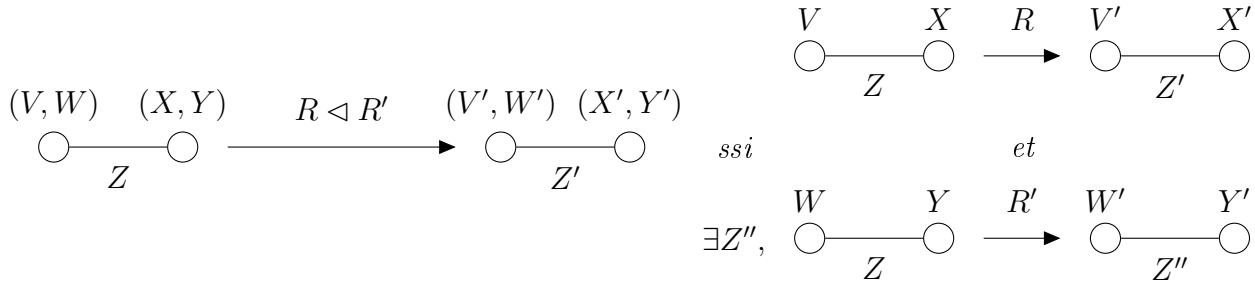
$$Grd\,\frac{R \leq_S R'}{R|P \leq_S R'}$$

5.4.5 Produit Gauche

Les opérateurs présentés jusqu'ici ne modifient pas le type d'une relation **LC0**. Or, changer les types d'étiquette sur lesquels opère une règle de réécriture peut, dans le cadre de la réutilisation de composants prouvés, être une opération utile. Nous proposons ici une transformation permettant de changer le type des étiquettes des sommets sur lequel des règles opèrent en conservant le type des arêtes. En effet nous n'avons rencontré que des transformations de relations où seul le type des étiquettes des sommets est composé.

Nous nommons cette transformation *produit gauche*. Ce dernier est par la suite utilisé avec l'identité ou une relation ne modifiant pas l'étiquetage de l'arête. Pour éviter de définir un nouveau type de relations (i.e. une relation ne concernant que l'étiquetage des sommets), la nouvelle étiquette de l'arête est quantifiée existentiellement dans l'application du terme droit dans le produit.

Définition 5.6. *Produit gauche **LC0** [Coq 46]*



Lemme 5.4. *Simulation et produit gauche [Coq 47]*
*Soient R et R' deux relations **LC0**.*
Soit S la relation $S = \{(G_s, H_n) | G = H \wedge s = \widehat{fst}(n)\}$ où $\widehat{fst}(n)$ est l'état n où les seconds composants des étiquettes des sommets ont été supprimés.
Un pas de $R \lhd R'$ simule un pas de R par S.

$$LP\,\frac{}{R \lhd R' \leq_S R}$$

5.5 Transformations courantes de règles LC0 : application à la transformation d'un système GTD en un système OTD

Dans [47], E. Godard, Y. Métivier et G. Tel donnent une preuve succincte de l'inclusion de la classe des relations à détection locale de la terminaison globale dans la classe des relations à détection observée de la terminaison. Cette preuve peut être résumée ainsi : soit R un système de réétiquetage localement engendré quelconque. On suppose que ce système détecte localement la terminaison globale i.e. un noeud détecte la terminaison si et seulement si l'état courant est une forme normale. Dans cet article, l'ajout d'une règle de diffusion de l'information de terminaison permet de transformer R en une relation de réétiquetage OTD.

À l'aide des transformations de règles précédemment définies, nous définissons en *Coq* une fonction de transformation d'une relation de réétiquetage GTD en une relation OTD pour la synchronisation **LC0**. Cette transformation est accompagnée de théorèmes prouvant sa correction ainsi que son appartenance à la classe de détection de terminaison OTD. On notera que nous avons tenu à respecter la synchronisation de la relation originelle : une relation **LC0** sera transformée en une relation **LC0**. Cette stabilité peut être utile dans le cas d'une éventuelle utilisation de cette transformation pour étudier l'expressivité des diverses classes de calculs locaux.

Notre approche est la suivante : nous commençons par définir la transformation à l'aide des opérateurs de transformation de règles définis précédemment. Nous montrons ensuite, à l'aide des résultats de simulation génériques concernant les opérateurs de transformations que le système obtenu simule bien le système originel et utilisons cette propriété pour démontrer la correction du système obtenu.

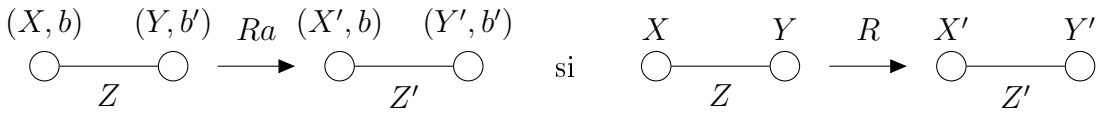
5.5.1 Définition

Nous notons R le système de réétiquetage originel et R' le système issu de la transformation. On supposera que R traite des étiquettes de type L_v pour les sommets et L_e pour les arêtes. Dans R' une composante booléenne est ajoutée aux étiquettes des sommets pour signifier la réception de l'information de terminaison. R' opère donc sur des étiquettes de type $L_v \times bool$. Le type des étiquettes des arêtes est inchangé. À l'initialisation, chaque sommet reçoit une étiquette dont la composante gauche correspond à l'initialisation de la relation originelle, et dont la composante droite est égale à $false$ (i.e. initialement aucune information de terminaison n'a été reçue).

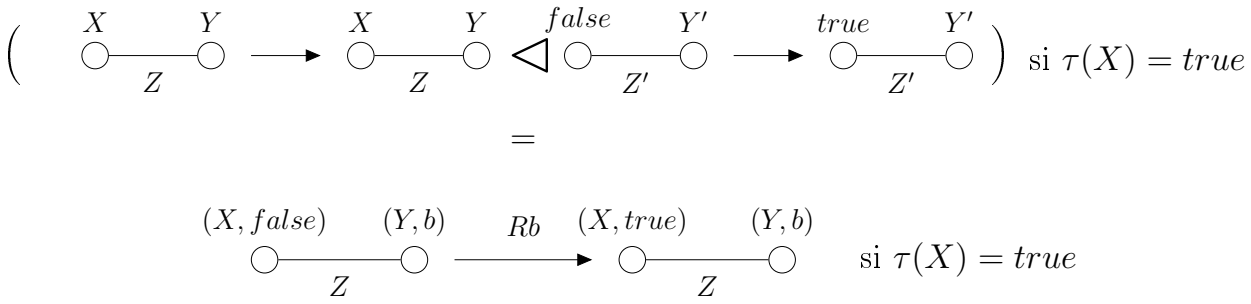
Nous décomposons la définition de R' en trois règles : la première, Ra, simulant R sur le nouvel étiquetage, la seconde, Rb, initialisant la phase de diffusion de l'information

de terminaison et la troisième, Rc, assurant la diffusion. Pour chacune des règles de réétiquetage, nous donnons sa définition accompagnée de sa représentation graphique.

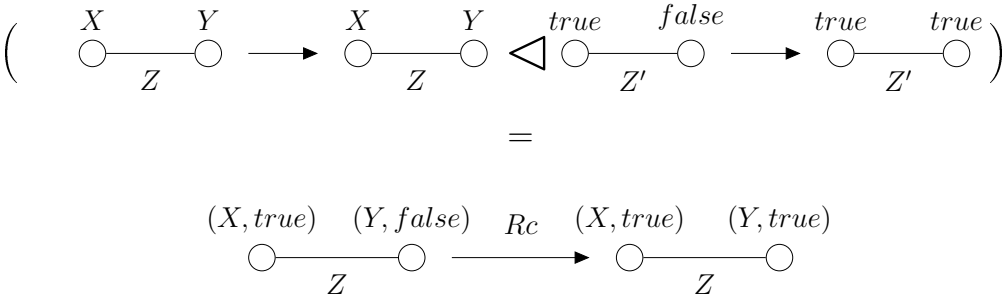
Nous définissons le premier composant de R', simulant R, comme le produit gauche de R avec l'identité sur les booléens. Plus formellement :



La définition du second composant se base sur l'opérateur de transition "interne" et illustre la construction d'une règle à l'aide d'une combinaison des opérateurs proposés. Sa définition peut être résumée ainsi : lorsque la terminaison globale est détectée sur un sommet, alors l'information est inscrite dans la seconde composante de l'étiquette de ce sommet. On remarquera que — comme dans le cas de la règle suivante — cette relation est construite comme le produit gauche de la relation **LC0** identité avec une autre relation, et ne changera donc pas l'étiquetage des arêtes.



La règle de diffusion consiste simplement à changer la seconde composante de l'étiquetage d'un sommet en *true* si un de ses voisins a reçu l'information de terminaison.



La règle **LC0** engendrant le système R' est simplement l'union des règles Ra, Rb et Rc. La figure 5.3 page ci-contre représente une exécution de la relation de réétiquetage

engendrée par R' ou les transitions sont étiquetées par les règles appliquées. Par souci de lisibilité, les étiquettes des arêtes ne sont pas représentées.

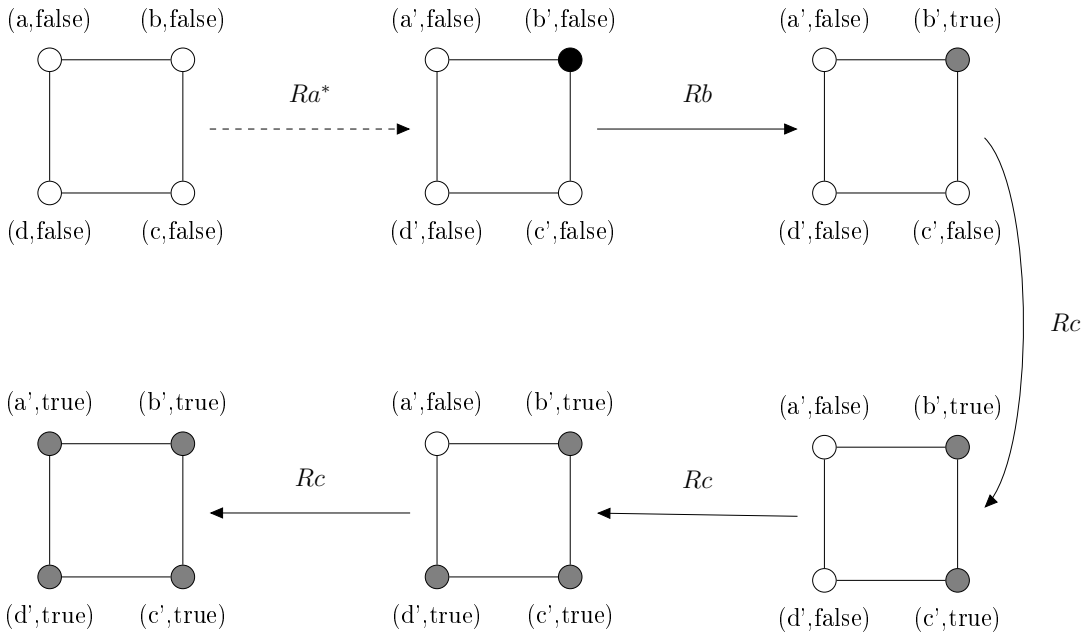


FIGURE 5.3 – Une exécution de R'. Le sommet noir a détecté la terminaison globale de R. Les sommets gris ont reçu l'information de terminaison.

On supposera par la suite que la relation de réétiquetage R réalise une tâche T. i.e. nous supposons l'existence de quatre fonctions ι_v, ι_e, π_v et pi_e telles que (ι, π, R') est une réalisation de T. On supposera de plus que les graphes sous-jacents des états initiaux de cette tâche sont connexes et contiennent au moins deux sommets, cette dernière condition étant nécessaire pour appliquer *au moins une fois* une règle **LC0**. On notera G_i un état initial pour R et G'_i un état initial pour R'.

5.5.2 Réalisation

Nous montrons maintenant que pour tout système R et toute tâche T réalisée par R, le système R' réalise T. La correction de R' est une conséquence de la simulation de R par R'.

Lemme 5.5. *[Coq 48]*
Soit S une relation définie par $S = \{(G_s, H_n) | G = H \wedge \widehat{fst}(n) = s\}$, ou $fst(x)$ est le composant gauche de l'étiquette x et $\widehat{fst}(s)$ est l'application de fst à toutes les étiquettes de s. $R' \leq_S R$.

Démonstration.
Nous donnons ici l'arbre de preuve résumant les applications des lemmes de la section

5.4. On note C la relation $\{(false, true)\}$, B la relation de diffusion de l'information de terminaison et P la propriété "la terminaison de R a été détectée". La preuve de simulation peut donc être représentée comme :

$$\dfrac{\dfrac{}{R \lhd Skip \leq_S R}\,\text{LP} \qquad \dfrac{\dfrac{\dfrac{}{Skip \lhd B \leq_S Skip}\,\text{LP}}{Skip \lhd B \leq_S R}\,\text{Skip}}{(Skip \lhd B)|P \leq_S R}\,\text{Grd} \qquad \dfrac{\dfrac{}{Skip \lhd C \leq_S Skip}\,\text{LP}}{Skip \lhd C \leq_S R}\,\text{Skip}}{(((Skip \lhd B)|P) \cup (Skip \lhd C)) \leq_S R}\,\text{Union}}{(R \lhd Skip) \cup (((Skip \lhd B)|P) \cup (Skip \lhd C)) \leq_S R}\,\text{Union}$$

□

On remarque dans la preuve du lemme 5.5 page précédente l'importance des définitions des règles engendrant le système de réétiquetage R' : ces définitions, combinées aux lemmes génériques sur l'union et le produit gauche de règle **LC0** permettent de rendre la démonstration de simulation *Coq* intuitive.

Nous déduisons du lemme 5.5 page précédente que R' réalise la tâche T.

Théorème 5.2. *[Coq 49]*
Soient les fonctions

$\iota'_v(x) = (\iota_v(x), false)$),
$\iota'_e = \iota_e$,
$\pi'_v(x) = \pi_v \circ fst(x)$,
$\pi'_e = \pi_e$,

(ι', π', R') *est une réalisation de* T.

Démonstration. Soit G_s une forme normale de R' accessible depuis $G_{i'}$ par applications répétées du lemme 5.5 page précédente, $G_{\widehat{fst}(s)}$ est accessible depuis un état initial de R. De plus, R' étant défini comme l'union de Ra, Rb et Rc, G_s est une forme normale pour Ra. Ra étant défini comme le produit gauche de R avec l'identité, $G_{\widehat{fst}(s)}$ est une forme normale pour R. De par l'hypothèse de réalisation de R, $\hat{\pi}(G_{\widehat{fst}(s)})$ appartient au domaine d'arrivée de T. Or, $\widehat{\pi}(G_{\widehat{fst}(s)}) = \widehat{\pi'}(G_s)$. Donc $\widehat{\pi'}(G_s)$ appartient au domaine d'arrivée de T, (ι', π', R') est donc bien une réalisation de T. □

Ici encore, la définition des règles de calcul à l'aide d'opérateurs (union et produit gauche) permet de réduire le raisonnement sur les formes normales à des résultats génériques factorisant les preuves, ce qui simplifie la démonstration du théorème 5.2.

5.5.3 Détection de terminaison *OTD*

On rappelle qu'un système de réétiquetage est dit *à détection observée de la terminaison* si :

(1) lorsqu'un sommet détecte la terminaison, la valeur de sortie de chaque sommet est finale,

(2) en forme normale, tout sommet a détecté la terminaison.

Dans R', nous dirons qu'un sommet détecte la terminaison s'il est étiqueté $(X, true)$, quelque soit X. Nous prouvons (1) en montrant que les trois propriétés suivantes sont des invariants du système :

(A) Si dans un état G_s un sommet a détecté la terminaison pour R, alors G_s est une forme normale pour Ra.

(B) S'il existe un sommet dont l'étiquette est $(X, true)$ (pour n'importe quel X), alors il existe un sommet ayant détecté la terminaison de R.

(C) Si la seconde composante de l'étiquette d'un sommet est $true$ dans un état G_s, alors dans tout état successeur de G_s par R' celle-ci gardera la valeur $true$.

Nous en déduisons maintenant une démonstration de (1) et de (2) :

Démonstration. (1) [Coq 50] :
On suppose G_s l'état courant, accessible depuis un état initial de R'. D'après (B) si un sommet détecte la terminaison de R' dans G_s, alors il existe un sommet ayant détecté la terminaison de R. D'après (A) G_s est donc une forme normale pour Ra. Or Ra est la seule règle à modifier les valeurs de sortie des sommets. Ce raisonnement peut être appliqué à tout successeur de G_s grâce à (C). □

Démonstration. (2) [Coq 51] :
Soit G_s l'état courant, accessible depuis un état initial de R'. On suppose que G_s est une forme normale de R'. Nous savons qu'un sommet, que l'on nommera o, a détecté la terminaison de R. Si o est étiqueté par $(X, false)$, alors un pas de calcul est possible par Rb ce qui contredit le fait que G_s soit une forme normale pour R'. o est donc étiqueté par $(X, true)$. Supposons qu'il existe un sommet de G_s étiqueté par $(Y, false)$. Dans ce cas, G étant connexe, par récurrence sur les chemins, il existe au moins deux sommets adjacents de G_s étiquetés chacun par $(X, true)$ et $(Y, false)$. Un pas de calcul par Rc est donc possible ce qui contredit le fait que G_s soit une forme normale pour R'. □

5.5.4 Terminaison

Pour cette dernière partie de la preuve, nous remarquons simplement que l'union de deux relations bien fondées commutant entre elles est bien fondée. Nous déduisons de la bonne fondation de R la bonne fondation de Ra. La bonne fondation de $Rb \cup Rc$ peut être prouvée à l'aide d'un variant correspondant au nombre de sommets dont la seconde

composante de l'étiquette est $false$. La commutativité de Ra et $Rb \cup Rc$ entre elles est une conséquence directe des invariants (A) et (B).

5.6 Conclusion et Perspectives

Dans ce chapitre nous avons proposé une adaptation et implantation en *Coq* du concept de *forward simulation* aux relations de réétiquetages de graphes. Nous avons montré comment cette notion nous permet de réutiliser les preuves de systèmes de réétiquetages.

Nous avons introduit, un ensemble de *transformations courantes* de règles de réétiquetage **LC0** pour simplifier l'usage de la simulation dans nos preuves. À l'aide de ces transformations, nous avons proposé une formalisation de la transformation d'une relation de réétiquetage **LC0** avec détection locale de la terminaison globale en une relation de réétiquetage **LC0** avec détection observée de la terminaison.

La notion de simulation est générique et peut être appliquée pour tout type de synchronisation (**LC0**, **LC1**, **LC2**). Nous l'avons d'ailleurs utilisée pour proposer une transformation GTD vers OTD pour les relations **LC1**. La preuve de la version **LC1** est similaire à celle de la version **LC0**. Les deux dernières règles de réétiquetage ajoutées modifient en effet l'étiquette d'un unique sommet lors d'un pas de calcul et sont donc facilement traduisibles en une version **LC1**. Toutefois il est nécessaire, dans la version **LC1**, d'ajouter à la preuve le fait que les règles diffusant l'information de terminaison respectent bien les prédicats définissant la synchronisation **LC1** (voir section 3.4.2 page 45).

Une transformation GTD vers OTD spécifique à la synchronisation **LC2** reste a réaliser. Toutefois, celle-ci reposant sur le même type que la synchronisation **LC1**, les changements à apporter à la preuve seraient minimes (les modifications porteraient principalement sur les prédicats définissant les relations **LC2**).

Les opérateurs de transformations proposés gagneraient à être généralisés et définis pour les synchronisations **LC1** et **LC2**. Ces opérateurs devraient selon nous être utilisés comme des *constructeurs* de règles de calculs, définissant ainsi un langage de règle et de permettant de factoriser toutes formes de preuves, et pas uniquement les preuves de simulation comme c'est ici le cas. Des travaux en cours par P. Castéran définissent ces constructeurs pour une représentation fonctionnelle des relations de réétiquetage.

Finalement, il serait intéressant de formaliser la hiérarchie définie par J. Chalopin (cf. section 2.6.4 page 31). Les transformations de systèmes et de preuves nécessaires, par exemple, à la démonstration de l'équivalence entre calculs locaux cellulaires sur les arêtes et calculs locaux sur les arêtes sont plus complexe que les cas traités ici et permettraient de tester plus en profondeur les méthodologies de preuve mises en place.

Chapitre 6

Composition Localement Séquentielle LC0

Nous décrivons ici deux propositions de composition des systèmes de calculs sur les arêtes (**LC0**). Nous montrons comment raisonner sur la composition de systèmes en utilisant une technique basée sur la commutation des relations de réétiquetage et la simulation en avant.

Sommaire

6.1 Introduction

Lors de la conception d'un programme séquentiel, il est d'usage de distinguer les différentes tâches accomplies par le système en les répartissant dans des sous-routines. Cette modularité permet d'accroître la lisibilité du programme, ainsi que d'en réutiliser les composants. Dans le cadre d'un développement formel, ce découpage est d'autant plus important du fait que la preuve de composants est longue et requiert une main d'œuvre spécialisée.

Dans ce chapitre, nous définissons deux techniques de composition, que l'on dira *localement séquentielle*, pour les systèmes de calculs locaux. La mise au point de ces techniques est motivée par l'étude de réalisabilité d'un système **LC0** de calcul d'arbre recouvrant avec détection locale de la terminaison globale. Nous avons vu, dans la section 4.3 page 63, que cette tâche est irréalisable avec détection locale de la terminaison globale si l'on considère des état initiaux où seule une racine est connue. Nous définissons, par composition, un système de réétiquetage résolvant ce problème lorsque le degré de chaque sommet est connu à priori.

La connaissance initiale des degrés nous offre deux choix de réalisations. Le premier choix consiste à formaliser en *Coq* la transformation d'un système **LC1** en un système **LC0** avec connaissance du degré proposée par J. Chalopin ([23], section 3.6.3 page 70). Connaissant une réalisation **LC1** (voir Annexe A.2.2 page 123) du calcul d'arbre recouvrant avec détection de la terminaison globale, l'application de la transformation de J. Chalopin nous fournie une réalisation **LC0**.

La transformation de J. Chalopin s'avère toutefois complexe, autant en termes de définition qu'en termes de nombre d'applications de règles de réétiquetage : la simulation de l'application d'une règle **LC1** nécessite en effet au minimum d applications d'une règle **LC0** pour un sommet de degré d. Nous proposons donc une solution alternative, dont le principe correspond à celui de l'algorithme **LC1** réalisant le calcul d'arbre recouvrant : on supposera initialement que le degré de chaque sommet est connu, et qu'un sommet est désigné comme racine. Nous calculons un arbre recouvrant, dans lequel chaque sommet est étiqueté par son degré *au sein de l'arbre calculé*. Détecter la terminaison globale revient alors à effectuer une élection sur le sous arbre calculé (voir section 5.3 page 81) : l'information de terminaison remonte le long de l'arbre construit jusqu'à une racine, qui détecte la terminaison globale. On remarque que contrairement a la réalisation **LC1**, le sommet détectant la terminaison globale n'est pas obligatoirement le même que la racine originel de l'arbre.

La solution choisie repose sur deux systèmes de réétiquetage que nous avons déjà étudier : d'un côté un calcul d'arbre recouvrant et de l'autre une élection dans un graphe dont un arbre recouvrant est connu. Notre objectif est d'esquisser un principe de *composition* de ces deux systèmes pour atteindre le résultat souhaité. Nous fixons pour ce principe

de composition les exigences suivantes : premièrement, ce principe de composition doit permettre d'appliquer les deux systèmes de réétiquetages en parallèle dans le graphe i.e. l'exécution de l'élection doit pouvoir commencer alors que le calcul de l'arbre recouvrant n'est pas terminé. Deuxièmement, la solution proposée doit être généralisable, même si certains détails techniques restent dans un premier temps spécifiques au problème abordé. Dans la suite de ce chapitre nous proposons et étudions deux solutions potentielles.

6.1.1 Travaux connexes

L'étude de la composition d'algorithmes distribués étant un vaste champ d'étude, nous ne produirons pas un résumé exhaustif de ce dernier. L'inspiration principale pour ce chapitre vient des travaux d'Yves Métivier *et al.* [46, 47, 25] : pour prouver l'existence de passerelles entre classes de détection de terminaison, un principe de composition est défini ([46], Definition 11). Nos deux propositions tentent de généraliser cette idée et d'étudier la structure des preuves nécessaires à sa validation.

Cette définition de composition diffère quelque peu de celle trouvée dans la littérature classique, par exemple dans les travaux de N. Lynch [64] ou ceux sur les algèbres de processus : Dans le cadre des automates d'entrée-sortie, composer deux automates revient (grossièrement) dans notre modèle à exécuter les deux automates sur deux noeuds adjacents. Dans notre modèle, chaque noeud exécute le même algorithme. Composer deux algorithmes revient à exécuter ces deux algorithmes d'une certaine manière sur chaque noeud. On s'intéresse donc ici aux conséquences globales d'un composition locale.

Nos définitions se rapprochent de celles de P.C Héam et O. Kouchnarenko [49], qui proposent d'utiliser des notions de simulation et de composition pour étudier l'interchangeabilité de composants et comparer leurs qualité de service. La définition de composition séquentielle qu'ils proposent, plus simple, ne permet néanmoins pas de passer le résultat de l'exécution d'un algorithme pour le passer en paramètre à un second.

6.1.2 Organisation du chapitre

Dans une première section, nous commençons par définir les notations et concepts généraux utilisés dans le reste du chapitre. Nous justifions notamment les restrictions auxquels nous faisons face concernant le réétiquetage de arêtes.

Nous poursuivons, en présentant dans la section 6.3 une première proposition de composition, pleinement formalisée. Nous décrivons ses propriétés utiles et ses faiblesses, notamment en terme de parallélisme d'exécution des systèmes.

Ces faiblesses nous ont poussé, en collaboration avec M. Tounsi, à proposer une seconde composition de relation de réétiquetage **LC0**, que nous décrivons dans la section 6.4. Là

où dans notre première proposition, l'application d'une règle du second système est conditionnée par la détection de la terminaison locale du premier sur les deux extrémités d'une arête, on ne requiert ici que la détection de la terminaison locale sur une des extrémités. Le système résultant applique plus tôt le second système que celui issu de la première proposition. Toutefois, cette définition donne naissance à de nouvelles obligations de preuves. Nous concluons en discutant des possibles améliorations de la notion de composition pour les calculs locaux.

6.2 Notations et définitions générales

Nos deux propositions de composition se basent sur l'observation suivante : si deux relations de réétiquetage opèrent sur des ensembles de variables disjoints, raisonner sur leur composition est simple. Nos deux propositions assurent donc que les deux relations de réétiquetage modifient chacune une composante de l'étiquetage des sommets. Cette définition a l'avantage de préserver les résultats du calcul de chacune des relations.

On considérera ici deux relations de réétiquetage **LC0** A et B réalisant respectivement deux taches T_A et T_B. Nous supposons que A et B terminent, et que A détecte localement la terminaison locale. Nous définissons le type des étiquettes des sommets manipulées par la composition de A et B comme étant le type produit $L = (L_A \times bool \times L_B)$. On supposera que les deux algorithmes manipulent des étiquettes d'arêtes du même type. Nous verrons par la suite que cette hypothèse est nécessaire.

On définit la fonction $fst : L \to L_A$ comme le composant gauche d'une étiquette (i.e. $fst((x, y, z)) = x$. De la même manière, on définit $third : L \to L_B$ comme $third((x, y, z)) = z$ et $snd : L \to bool$ comme $snd((x, y, z)) = y$.

Dans nos deux propositions, la transmission des résultats des calculs de A à B est assurée par une relation calculant l'image de chaque étiquette des sommets d'un état par une certaine fonction avec détection locale de la terminaison locale. Nous appellerons par la suite cette relation *l'opérateur de copie*. On note que l'opérateur de copie ne modifie pas les étiquettes des arêtes : en effet, il est impossible dans le cas général (sans connaissance initiale spécifique) de détecter localement la terminaison locale d'une relation calculant l'image des étiquettes de toutes les arêtes par une fonction.

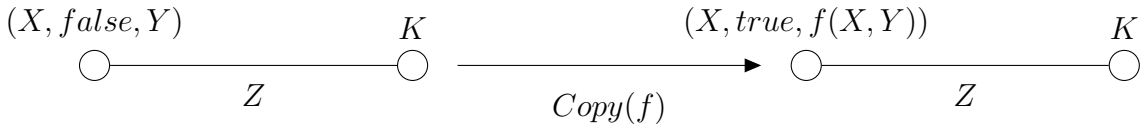
Théorème 6.1 (informel)**.** *Soit $T_C(f, g)$ la tâche consistant à appliquer la fonction f (resp. g) à toute étiquette de sommet (resp. d'arête) sur un graphe étiqueté initial quelconque. $T_C(f, g)$ est irréalisable avec détection locale de la terminaison locale.*

Démonstration. Similaire à la preuve du théorème 4.1 page 60, en utilisant le lemme *d'extension.* □

Le calcul de l'image de l'étiquette de chaque sommet et arête d'un graphe étiquetés est donc irréalisable par une relation **LC0**, sans connaissances initiales spécifiques (par exemple le degré). On remarque toutefois que si l'on ne considère qu'une transformation des étiquettes des sommets, la tâche est réalisable. Bien que la transformation des étiquettes des sommets *et* arêtes soit réalisable par une relation **LC1** (cf. règle 4 page 60), nous choisissons de définir l'opérateur de copie comme une relation **LC0**. Ce choix a l'avantage de préserver le mode de synchronisation, ce qui peut s'avèrer utile si l'on souhaite étudier l'expressivité de celle-ci via la composition.

Nous définissons *l'opérateur de copie* utilisé dans nos deux propositions de composition comme suivant :

Définition 6.1 (informelle)**.** *Soient L_A et L_B deux types. Soit f une fonction $f : L_A \to L_B \to L_B$. Nous nommons opérateur de copie la règle de réétiquetage suivante :*



On note que cette définition de l'opérateur de copie est une généralisation d'une définition proposé dans l'implantation *Coq*. La version formelle [Coq 52], dont la correction à été prouvée, est plus restreinte et ne prend pas en compte le troisième composant d'une étiquette lors d'une copie.

La relation de copie sera appliquée une unique fois sur chaque sommet du graphe, grâce au passage de la composante booléenne de l'état à *true*. On remarque de plus que le même mécanisme permet la détection locale de la terminaison locale. Une exécution est présentée en figure 6.1 page suivante.

Lemme 6.1 (informel)**.** *Soit $T_C(f)$ la tâche consistant à calculer l'image par f des étiquettes de tous sommets d'un graphe étiqueté G_s connexe tel que $|V(G)| > 2$. Soit id la fonction identité, $(id, id, Copy(f))$ est une réalisation de $T_C(f)$ avec détection locale de la terminaison locale.*

Démonstration. triviale. □

6.3 Une première proposition

6.3.1 Definitions

On rappelle que nous souhaitons composer les relations de réétiquetage A et B de telle manière que les résultats des calculs de A puissent être réutilisés dans les calculs de B.

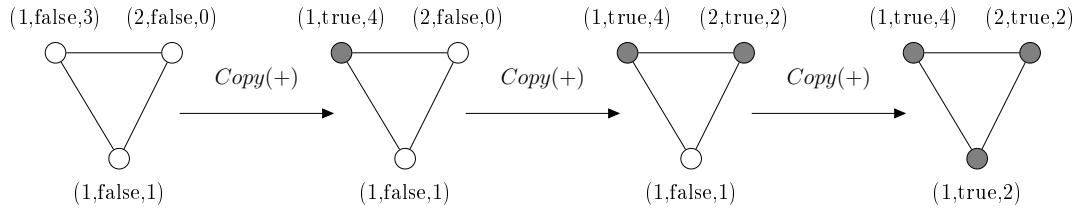


FIGURE 6.1 – Une execution de l'opérateur de copie où le paramètre est la fonction somme. Les sommets gris on appliqué la rêgle et detecté la terminaison locale.

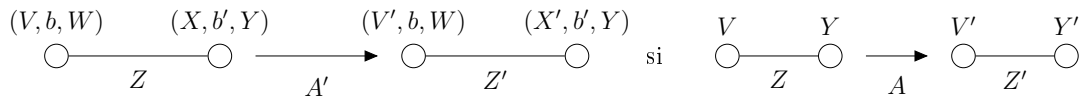
On suppose que A (resp. B) réalise une tâche T_A (resp T_B). On supposera que les états du domaine de T_A (resp. T_B) sont étiquetés par des objets de type I_{AV} (resp. I_{BV} pour les sommets et I_{AE} (resp. I_{BE}) pour les arêtes. Pour les types concernant les domaines d'arriver des tâches, on remplacera I par O (O_{AV}, O_{AE} etc.). Pour les raisons exposées plus tôt (voir proposition 6.1 page 97) nous supposons que $O_{EA} = I_{EB}$. On requiert que nous soient fournies deux réalisations — (ι_A, π_A, A) une réalisation de A et (ι_B, π_B, B) une réalisation de B — ainsi qu'une fonction $f : O_{AV} \to I_{BV}$ que l'on nommera *fonction de coopération*.

Pour nous assurer du bon comportement de la composition de A et B nous faisons l'hypothèse suivante, dite hypothèse *de compatibilité*, concernant T_A et T_B :

Définition 6.2. Hypothèse de compatibilité [Coq 53] :
Soit G_s un état appartenant au domaine d'arrivée de T_A. Le graphe étiqueté résultant de l'application de f à l'ensemble des étiquettes des sommets de G_s appartient au domaine de T_B.

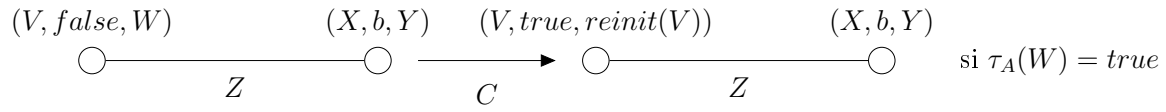
En d'autres termes f transforme un état final de T_A en un état initial de T_B. Nous définissons la composition de A et B comme l'union de trois règles de calculs : la première, que nous noterons A', reproduit le comportement de A sur des étiquettes de type $(L_A \times bool \times L_B)$. Nous utilisons pour définir ces trois règles les opérateurs introduits au chapitre précédent (voir section 5.4 page 83).

Règle 7 Règle A' de simulation de A



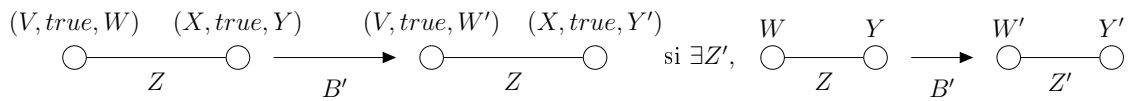
La seconde règle du système, que l'on notera C, applique l'opérateur de copie avec en argument la fonction *reinit*, associant à une étiquette de A son image par $\iota_B \circ f \circ \pi_A$. Cette règle est appliquée seulement si la terminaison de A a été détectée.

Règle 8 Règle C d'application de l'opérateur de copie.



La troisième et dernière règle du système applique B sur une arête lorsque l'opérateur de copie à été appliqué à ses deux extrémités. Cette règle est définie comme un renforcement de la garde du produit gauche de l'identité avec B. On remarque qu'à cause de cette définition, cette règle ne peut modifier l'étiquetage de l'arête, même si c'est le cas dans B. Dans l'exemple étudié, cette définition simplifie les raisonnements puisque ainsi l'union des trois règles simule A. Toutefois, si l'on souhaite généraliser cette méthode, un produit droit (le quantificateur existentiel passe sur la règle gauche) doit être utiliser.

Règle 9 Règle B' de simulation de B



Nous définissons la composition de A et de B, que l'on notera $A \oplus B$, comme l'union de ces trois règles.

6.3.2 Propriétés générales

Nous étudions tout d'abord le comportement général de $A \oplus B$. Nous montrons en particulier que $A \oplus B$ réalise la tâche réalisée par A, et que sous certaines hypothèses, $A \oplus B$ hérite de la détection de terminaison de B. Nous utilisons pour ce faire la notion de simulation introduite au chapitre précédent, ainsi que la commutation des règles A', C et B' deux à deux.

6.3.2.1 Réalisation

Comme dans le cas de la transformation d'un système GTD en un système OTD, nous pouvons utiliser les résultats sur les transformations courantes pour montrer que $A \oplus B$ simule A.

Lemme 6.2. *[Coq 54] Soit S la relation définie comme $S = \{(G_s, H_n) | G = H \wedge s = \widehat{fst}(n)\}$, alors S est une relation de simulation : $A \oplus B \leq_S A$.*

Démonstration.
On remarque que l'opérateur de copie ne modifie pas la première composante des étiquettes des sommets. Donc celui-ci simule *Skip* par S quelque soit la fonction passée

en paramètre. Nous donnons la structure de la preuve comme l'arbre d'application des lemmes de simulations, de la même manière que pour le lemme 5.5 page 89 :

$$\dfrac{\text{LP}\ \dfrac{}{(A \lhd R) \leq_S A} \qquad \text{Union}\ \dfrac{\text{Grd}\ \dfrac{\text{Skip}\ \dfrac{(Copy\ reinit) \leq_S Skip}{(Copy\ reinit) \leq_S A}}{((Copy\ reinit)|P) \leq_S A} \qquad \dfrac{\text{LP}\ \dfrac{}{(Skip \lhd B) \leq_S Skip}}{(Skip \lhd B) \leq_S A}\ \text{Skip}}{(((Copy\ reinit)|P) \cup (Skip \lhd B)) \leq_S A}}{(A \lhd R) \cup (((Copy\ reinit)|P) \cup (Skip \lhd B)) \leq_S A}\ \text{Union}$$

□

De la même manière que pour le théorème 5.5 page 89, nous pouvons déduire du lemme 6.2 page ci-contre que $A \oplus B$ réalise la tâche T_A. Cette démonstration, de part la définition de $A \oplus B$ ne demande aucune hypothèse supplémentaire sur le comportement de A ou B.

Théorème 6.2. *[Coq 55]*
Soit l_B une valeur quelconque de type L_B et soient les fonctions
$\iota'_v(x) = (\iota_{Av}(x), false, l_B))$,
$\iota'_e = \iota_{Ae}$,
$\pi'_v(x) = \pi_{Av} \circ fst(x)$,
$\pi'_e = \pi_{Ae}$,
alors $(\iota', \pi', A \oplus B)$ est une réalisation de T_A.

Démonstration. Similaire à la démonstration du théorème 5.5 page 89. □

6.3.2.2 Terminaison

Comme dans le cas de la transformation d'une relation GTD en une relation OTD, la terminaison de $A \oplus B$ peut être déduite du fait que A', B' et C commutent deux à deux.

Théorème 6.3. *[Coq 56] Toute exécution de $A \oplus B$ est finie.*

Démonstration. Soit f la fonction qui à un état x d'un sommet de $A \oplus B$ associe 1 si $snd(x) = true$ et 0 sinon. f est un variant pour $C \cup B'$. Toute exécution de $C \cup B'$ est donc finie.
On remarque de plus que B' et C commutent avec A', donc $C \cup B'$ commute avec A'. Toute exécution de A' est finie par hypothèse, donc, $C \cup B'$ commutant avec A', toute exécution de $A' \cup (C \cup B')$ est finie. □

6.3.2.3 Détection de terminaison

On suppose maintenant que B détecte localement la terminaison globale. Nous montrons que, si certaines conditions sont respectées, $A \oplus B$ hérite de cette propriété. En plus de l'hypothèse de compatibilité (voir définition 6.2 page 99), nous faisons à cette fin les hypothèses suivantes :

(i) Si la terminaison de A à été détectée sur un sommet, alors ce dernier ne participe plus à aucun pas de réétiquetage.
(ii) B détecte localement la terminaison globale.
(iii) B ne modifie pas les étiquettes des arêtes.
(iv) Les étiquettes apparaissant dans un état initial de B n'apparaissent pas dans une forme normale de B.

On rappelle que $A \oplus B = A' \cup (C \cup B')$. Notre raisonnement se base sur l'observation suivante : à toute exécution de $A \oplus B$ *correspond* une exécution de $A \oplus B$ où A', C et B' sont exécutés de manière *globalement séquentielle*. Par globalement séquentielle nous voulons dire que si l'on considère une exécution de $A \oplus B$, si un réétiquetage par C a lieu, alors aucun réétiquetage par A' n'aura plus lieu, et si un pas de B' à lieu, alors aucun réétiquetage par C ou par A' n'aura plus lieu. Raisonner sur une exécution globalement séquentielle nous permet d'utiliser l'hypothèse de coopération (cf définition 6.2 page 99).

Définition 6.3 (informelle)**.** *Séquence de réétiquetage globalement séquentielle.*
Soient A et B deux relations de réétiquetage. Soit $G_{s_1}, G_{s_2}, \ldots, G_{s_n}$ une suite d'états tels que $\forall i, 1 < i < n \Rightarrow G_{s_i} \xrightarrow{A \cup B} G_{s_{i+1}}$. La séquence de réétiquetage $G_{s_1}, \ldots, G_{s_n}$ est dîtes globalement séquentielle *ssi $\exists j, \forall i, 1 < i < j \Rightarrow G_{s_i} \xrightarrow{A} G_{s_{i+1}} \wedge \forall i \geq j, G_{s_i} \xrightarrow{B} G_{s_{i+1}}$*

Pour démontrer qu'un état accessible par une exécution de $A \oplus B$ est accessible par une exécution globalement séquentielle de $A \oplus B$, nous nous basons sur le fait que chacune des règles A C et B commutent deux à deux. Nous ne détaillons pas la preuve suivante, celle-ci revenant à appliquer la définition de la commutativité de manière récurrente.

Lemme 6.3. *[Coq 57] Soient G_i et G_s deux états de $A \oplus B$ tels que $G_i \xrightarrow{A \oplus B^*} G_s$. Il existe deux états G_a et G_c tels que $G_i \xrightarrow{A'^*} G_a \xrightarrow{C^*} G_c \xrightarrow{B'^*} G_s$.*

Démonstration. par récurrence sur la séquence de réétiquetage $G_i \xrightarrow{A \oplus B^*} G_s$. □

On remarque enfin que puisque B ne modifie pas l'étiquetage des arêtes par l'hypothèse (iii), il existe une relation de simulation S' tel que $B' \leq_{S'} B$.

Lemme 6.4. *[Coq 58]*
Soit S' la relation définie comme $S = \{(G_s, H_n) | G = H \wedge s = \widehat{third}(n)\}$. Alors S' est une relation de simulation : $B' \leq_{S'} B$.

Démonstration. B' étant un renforcement de garde du produit gauche de l'identité et de B, celui-ci modifie la troisième composante de l'étiquette des sommets de la même manière que B. De plus, par l'hypothèse (iii), B' ne modifie pas les étiquettes des arêtes. Donc $B' \leq_{S'} B$. □

Nous montrons maintenant que $A \oplus B$ hérite de la détection de terminaison de B. On rappelle que le système de réétiquetage $A \oplus B$ est dit GTD si l'existence d'un sommet ayant détecté la terminaison dans un état équivaut au fait que cet état est une forme normale. Dans le cas de $A \oplus B$, on dit qu'un sommet a détecté la terminaison si ce sommet à détecté la terminaison de B. Nous commençons par prouver la seconde direction de l'équivalence :

Lemme 6.5. *[Coq 59]*
Soit G_s une forme normale de $A \oplus B$ accessible depuis un état initiale G_i. Il existe dans G_s un sommet ayant détecté la terminaison de $A \oplus B$.

Démonstration. Nous savons que $G_i \xrightarrow{A \oplus B} G_s$. Du lemme 6.3 page ci-contre, nous déduisons que : $\exists G_t\ G_{t'}, G_i \xrightarrow{A'*} G_t \xrightarrow{C} G_{t'} \xrightarrow{B'*} G_s$.
On remarque de plus qu'un pas de réétiquetage par B' préserve les formes normales pour C et A' et qu'un pas de réétiquetage par C préserve les formes normales par A'.
$G_{t'}$ est donc une forme normale pour C et G_t est une forme normale pour A'.
Nous déduisons de l'hypothèse de compatibilité qu'il existe un état $G_{u'}$ tel que $(G_{u'}, G_{t'}) \in S'$ et $G_{u'}$ est un état initial pour B.
Par le lemme 6.4 page précédente, il existe un état $G_{u''}$ tel que $(G_{u''}, G_s) \in S'$ et $G_{u'} \xrightarrow{B^*} G_{u''}$.
Si G_s est une forme normale pour $A \oplus B$, alors G_s est une forme normale pour B', et donc $G_{u''}$ est une forme normale pour B.
Or B étant GTD, il existe un sommet de $G_{u''}$ ayant détecté la terminaison pour B.
Comme $(G_{u''}, G_s) \in S'$, il existe donc un sommet de G_s ayant détecté la terminaison pour B, et par définition pour $A \oplus B$. □

Pour démontrer la seconde direction de l'équivalence, nous procédons par contradiction. Nous supposons qu'un sommet d'un état G_s à détecté la terminaison et que G_s n'est pas une forme normale pour $A \oplus B$. nous construisons une séquence de pas de $A \oplus B$ à partir G_s et montrons que l'existence de cette séquence est impossible.

Lemme 6.6. *[Coq 60]*
Soit G_s un état de $A \oplus B$ accessible depuis un état initial G_i. S'il existe un sommet de G_s ayant détecté la terminaison, alors G_s est un forme normale pour $A \oplus B$.

Démonstration. Soient G_s et G_i deux états de $A \oplus B$ tels que G_i est un état initial pour $A \oplus B$ et $G_i \xrightarrow{A \oplus B^*} G_s$.
On suppose qu'il existe un sommet v de G_s d'étiquette x telle que $\tau_B(third(x)) = true$) i.e. v a détecter la terminaison.

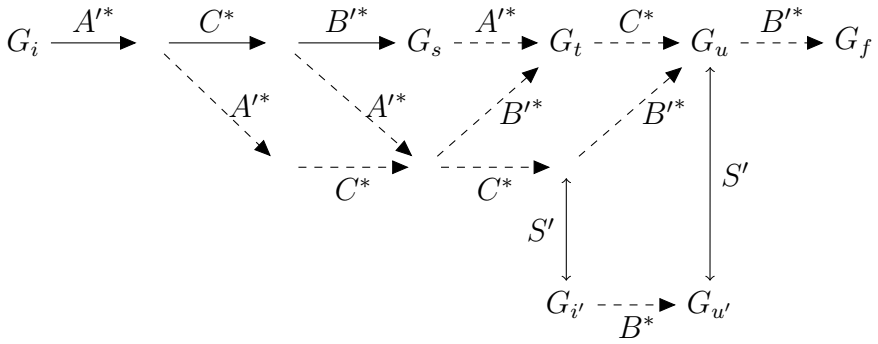


FIGURE 6.2 – Construction de l'exécution de B simulée par une partie de l'execution de $A \oplus B$.

Supposons que G_s ne soit pas une forme normale pour $A \oplus B$. Par le théorème 6.3 page 101, toute séquence de pas de $A \oplus B$ est finie, il existe donc une forme normale G_f, accessible par $A \oplus B$ depuis G_s.

Par le lemme 6.3 page 102, il existe deux états G_t et G_u tels que $G_s \xrightarrow{A'^*} G_t \xrightarrow{C^*} G_u \xrightarrow{B'^*} G_f$. On note que si $\tau_B(third(x)) = true)$ dans G_s, si y est l'étiquette de v dans G_f, $\tau_B(third(y)) = true)$.

À l'aide de l'hypothèse de compatibilité et du lemme 6.4 page 102, nous pouvons construire une exécution de B $G_{i'} \xrightarrow{B^*} G_{u'}$ où $G_{i'}$ est un état initial pour B et $(G_{u'}, G_u) \in S'$ (voir figure 6.2).

Comme $(G_{u'}, G_u) \in S'$, v a détecté la terminaison de B. B étant GTD, $G_{u'}$ est une forme normale pour B. Nous en déduisons que G_u est une forme normale pour B', donc $G_u = G_f$.

De plus $G_t \xrightarrow{C^*} G_u$, donc $G_t \xrightarrow{C^*} G_f$.

Si un pas de réétiquetage par C a lieu entre G_t et G_f, alors un sommet de G_f possède une étiqueté apparaissant dans un état initial pour B. $(G_{f'}, G_f) \in S'$, donc un sommet de $G_{f'}$ possède une étiquette apparaissant dans un état initial pour B. $G_{f'}$ étant une forme normale pour B, cela contredit l'hypothèse (iv). Donc $G_t = G_u = G_f$.

Finalement $G_s \xrightarrow{A'^*} G_t$ donc $G_s \xrightarrow{A'^*} G_f$.

Hors nous savons que G_f est une forme normale de $A \oplus B$. Tout sommet de G_f doit donc posséder une étiquette de la forme $(X, true, Y)$. Hors, $G_s \xrightarrow{A'^*} G_f$ et A' ne changeant pas la seconde composante des étiquettes, tout sommet de G_s possède une étiquette de la forme $(X, true, Y)$.

Hors on remarque que, d'après la définition de $A \oplus B$, si le second composant d'une étiquette est $true$, alors le sommet portant cette étiquette à détecté la terminaison de A. Pour tout sommet de G_s la terminaison de A a été détecté, G_s est donc une forme normale pour A et par extension pour A'.

$G_s = G_f$. G_s est donc une forme normale pour $A \oplus B$. □

Nous en concluons que $A \oplus B$ est un système GTD.

Théorème 6.4. *[Coq 61] Si les conditions (i), (ii), (iii) et (iv) sont remplies, alors $A \oplus B$ est un système de réétiquetage **LC0** GTD.*

Démonstration. Par application des lemmes 6.6 page 103 et 6.6 page 103. □

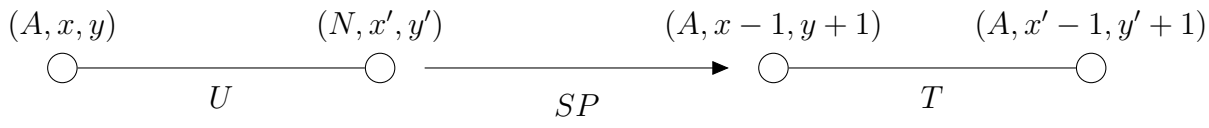
6.3.3 Application au calcul d'arbre recouvrant avec détection locale de la terminaison globale

Nous appliquons maintenant la composition en remplaçant le système A par un calcul d'arbre recouvrant, et le système B par le système d'élection décrit en section 5.3 page 81.

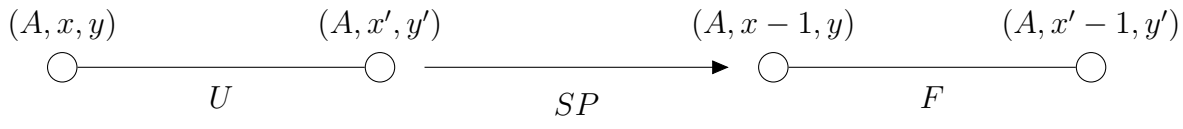
6.3.3.1 Calcul d'arbre recouvrant et des degrés dans l'arbre

Le système de calcul d'arbre recouvrant considéré, que l'on nommera SP', est défini comme suivant : soit $L_S = \{A, N\}$, les étiquettes des sommets appartiennent à l'ensemble $L_S \times \mathbb{N} \times \mathbb{N}$. Le premier composant d'une étiquette représente l'appartenance à l'arbre en construction, le second le nombre de voisins restant à visiter, le dernier le degré du sommet dans l'arbre en construction. Les arêtes sont étiquetées par U, T ou F. Initialement chaque sommet v est étiqueté par $(N, d(v), 0)$ où $d(v)$ est le degré de v dans le graphe considéré, sauf un sommet racine r étiqueté par $(A, d(v), 0)$. Le système est composé de deux règles, l'une (règle 10) ajoutant un sommet et une arête à l'arbre en cours de construction, l'autre (règle 11) éliminant une arête ne pouvant être ajoutée à l'arbre. Lorsque le nombre de voisins restant à traiter d'un sommet atteint zéro, celui-ci détecte la terminaison locale. L'implantation *Coq* de ce système et sa preuve de correction sont discutées en annexe (voir section A.2.3 page 125).

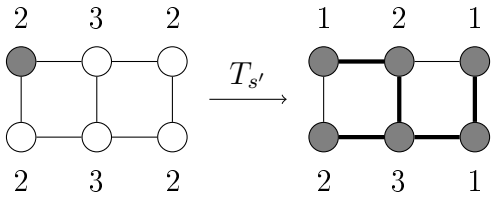
Règle 10 Calcul d'arbre recouvrant avec calcul du degré de chaque sommet dans l'arbre construit : ajout d'une arête.



Règle 11 Calcul d'arbre recouvrant avec calcul du degré de chaque sommet dans l'arbre construit : élimination d'une arête.



Tache 3 $T_{S'}$: Calcul d'arbre recouvrant avec calcul du degré de chaque sommet dans l'arbre construit

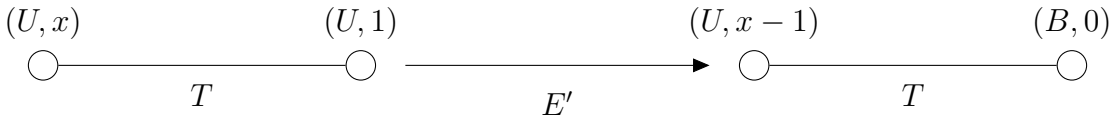


Ce système réalise une tâche $T_{S'}$ ou initialement le degré de chaque sommet est connu et un sommet est distingué. Les états finaux de $T_{s'}$ sont les états ou un arbre recouvrant à été calculé et le degré de chaque sommet dans cet arbre est connu.

6.3.3.2 Election dans un graphe dont un arbre recouvrant est connue *a priori*

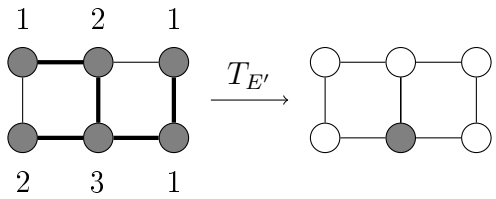
On rappelle la définition du système E d'élection dans un graphe dont un arbre recouvrant est connue, déjà présenté en section 5.3 page 81 : chaque sommet est étiqueté par son degré et le fait qu'il soit battu ou non. Les applications répétées de la règle 15 page 111 élague l'arbre jusqu'à ce qu'il ne reste qu'un unique sommet. L'implantation et la preuve de correction *Coq* sont discutées en section A.3.2 page 128.

Règle 12 Élection dans un graphe dont un arbre recouvrant est connu.



Ce système réalise une tâche $T_{E'}$ (tache 4) où un arbre recouvrant est connu, ainsi que le degré de chacun des sommets dans cet arbre. Les états finaux de $T_{s'}$ sont les états où un unique sommet est élu.

Tache 4 $T_{E'}$: Élection dans un graphe dont un arbre recouvrant est connu



6.3.3.3 Résultat de la composition : un calcul d'arbre recouvrant avec détection locale de la terminaison globale.

On considère ici un graphe connexe contenant plus d'un sommet. Le théorème 6.2 page 101 nous assure immédiatement que la composition de SP et de E' réalise un calcul d'arbre recouvrant :

Théorème 6.5. *[Coq 62] $SP \oplus E'$ réalise la tâche $T_{S'}$.*

Démonstration. Par application du théorème 6.2 page 101 □

De plus, les définitions de $T_{s'}$ et $T_{E'}$ assurent la validité de *l'hypothèse de compatibilité* : un état final de la tâche $T_{s'}$ est un état initial de la tâche $T_{E'}$. Il est de plus trivial de montrer, à l'aide de variants, que SP et E' terminent . Nous montrons maintenant que les autres conditions permettant l'héritage de la détection de terminaison de E' sont remplies.

Lemme 6.7. *[Coq 63] condition (i)*
Si la terminaison de SP a été détectée sur un noeud, celui-ci ne participe plus à aucun pas de réétiquetage.

Démonstration. La terminaison de SP est détectée pour un sommet lorsque la seconde composante de son étiquette est zéro. Hors la seconde composante de l'étiquette d'un sommet correspond au nombre de voisins restant a traité. Si celui-ci est nul, alors ce sommet ne participera plus à aucun pas de réétiquetage. □

Lemme 6.8. *[Coq 64] condition (ii)*
E' détecte localement la terminaison globale.

Démonstration. Par simulation du système d'élection correspondant à la règle 5 page 81. □

Lemme 6.9. *[Coq 65] condition (iii)*
E' ne modifie pas les étiquettes des arêtes.

Démonstration. Par définition de E'. □

Lemme 6.10. *[Coq 66] condition (iv)*
Un étiquette apparaissant dans un état initiale de E' n'apparaît pas dans une forme normale pour E'.

Démonstration. Initialement tout sommet est étiqueté par son degré dans l'arbre recouvrant. Ce dernier étant un arbre, donc connexe, le degré de chaque sommet est supérieur ou égale à 1. Hors dans une forme normale de E', la seconde composante de l'étiquette de tout sommet est zéro. □

Théorème 6.6. *[Coq 67] $SP \oplus E'$ réalise la tâche $T_{S'}$ avec détection locale de la terminaison globale.*

Démonstration. Les conditions (i) à (iv) étant remplies, par application du théorème 6.4 page 105. □

6.3.4 Discussion

Deux améliorations pourraient être apportées à cette première proposition de composition. Premièrement l'hypothèse (iv) est peu intuitive et semble "artificielle". Celle-ci est en effet directement issue de la preuve de l'héritage de la propriété GTD du second algorithme. Il serait intéressant soit de fournir une preuve alternative ne nécessitant pas l'hypothèse (iv), soit d'exprimer cette dernière sous une forme plus intuitive.

La seconde amélioration concerne les exécutions de la composition. Comme nous l'avons fait remarquer plus tôt, l'application du second système sur une arête est conditionné par la terminaison du premier sur les deux extrémités de celle-ci. Cela donne en général lieu, dans le cas de la composition du calcul d'un arbre recouvrant avec l'élection d'un leader à des *exécutions globalement séquentielles.*

6.4 Une seconde proposition

Dans cette section, nous présentons le travail accompli avec M. Tounsi [92] sur la composition. Ce travail à débouché sur une seconde notion de composition des relations **LC0**. Celle-ci se base sur le fait que la relation **LC0** d'élection définie en 6.3.3.2 page 106 peut être appliquée dès qu'un sommet détecte la terminaison du calcul d'arbre recouvrant.

6.4.1 Definitions

La notion de composition proposée précédemment permet principalement d'hériter de la détection de terminaison de la seconde relation de réétiquetage. Nous nous intéressons ici à la réutilisation des résultats du second algorithme. Là où dans le premier cas nous prenons comme exemple un système de calcul d'arbre recouvrant avec détection globale de la terminaison, nous étudions ici un système calculant un arbre recouvrant et y réalisant l'élection d'un leader. Nous commençons par définir la notion de *composition de tâches.* Grossièrement, la composition de deux tâches T_A et T_B est une tâche $T_{A\otimes B}$ dont le domaine est le domaine de T_A et dont le domaine d'arrivée est le domaine d'arrivée de T_B. Plus précisément :

Définition 6.4. *Composition de deux tâches [Coq 68]*
Soit T_A et T_B deux tâches opérant sur des graphes étiquetés par les mêmes types. On définit la composition des tâches T_A et T_B comme la tâche $T_{A\otimes B} = \{(G_i, G_f) | G_i \in dom(T_A) \wedge \exists G_{i'} \in dom(T_B), (G_{i'}, G_f) \in T_B\}$

Nous simplifions ici le contexte en supposant que les états initiaux et finaux des tâches T_A et T_B sont des graphes dont les étiquettes appartiennent aux mêmes ensembles. Cette supposition ne pose pas de réels problèmes en ce qui concerne la réutilisation des preuves des algorithmes que l'on souhaite composer et permet de simplifier l'hypothèse de compatibilité :

Définition 6.5. *Nouvelle hypothèse de compatibilité [Coq 69]*
Soient deux tâches T_A et T_B. T_B est compatible avec T_A si $\forall (G_i, G_s) \in T_A, G_s \in dom(T_B)$.

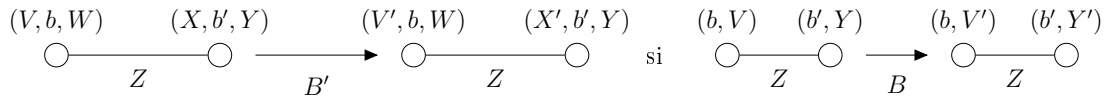
Cette nouvelle hypothèse de compatibilité représente un changement fondamental par rapport à la proposition de composition précédente. Dans la version de la section 6.3 page 98, la transformation des états finaux faisait partie de l'hypothèse de compatibilité. Ici on considérera trois tâches et deux compatibilités : la tâche T_A, une tâche de copie T_c (réalisée par l'opérateur de copie) compatible avec T_A, et enfin la tâche T_B, compatible avec T_c. Ce changement permet de fournir une preuve simple de la réalisation de la composition de deux tâches.

Lemme 6.11. *[Coq 70] Soient deux tâches T_A et T_B telles que T_B est compatible avec T_A. Soit A (resp. B) un système de réétiquetage réalisant T_A (resp. B). Si B commute avec A et si les formes normales de A sont préservées par B, alors $A \cup B$ réalise $T_{A \otimes B}$.*

Démonstration. Soient deux états G_i et G_s tels que
G_i est un état initiale de A,
G_s est une forme normale pour $A \cup B$ et
$G_i \xrightarrow{(A \cup B)^*} G_s$.
B commutant avec A, il existe un état G_t tel que $G_i \xrightarrow{A^*} G_t \xrightarrow{B^*} G_s$. Les formes normales de A étant stables par B, G_t est une forme normale de A, accessible depuis un état initial de A. A réalisant T_A et T_B étant compatible avec T_A, il existe un état $G_{t'}$ appartenant au domaine de T_B correspondant à G_t. B réalisant T_B, il existe un état $G_{s'}$ correspondant à G_s tel que $(G_{t'}, G_{s'}) \in T_B$. La tâche $T_{A \otimes B}$ est donc bien réalisée par $A \cup B$. □

Nous considérons dans la suite les trois systèmes de réétiquetage A, $Copy(f)$ et B réalisant respectivement les tâches T_A, $T_{C(f)}$ et T_B. Nous supposons que ces trois tâches concernent des graphes dont les sommets sont étiquetés par $L_A \times bool \times L_B$. On suppose que A manipule des étiquettes de sommet de type L_A et B manipule des étiquettes de sommets de types $bool \times L_B$. Le composant booléen permet d'indiquer à B si l'opération de copie a été appliquée. Dans tous les systèmes de réétiquetage on considère qu'un unique type L_E d'étiquetage des arêtes est manipulé. Les règles A' et $Copy$ sont définies comme précédemment. La règle B' permettant de simuler B devient :

Règle 13 Nouvelle règle B' de simulation de B



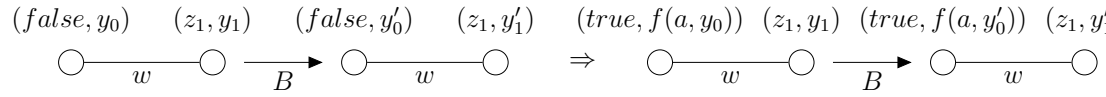
Dans le reste du chapitre nous montrons que l'union de A', C et B' réalise la tâche $T_{A \otimes (C(f) \otimes B)}$, i.e. une tâche ayant pour état initiaux les états initiaux de T_A et pour état finaux les états finaux de T_B, $T_{C(f)}$ étant la tâche correspondant à l'application de la fonction f à tout sommet d'un graphe étiqueté étant un état final pour T_A.

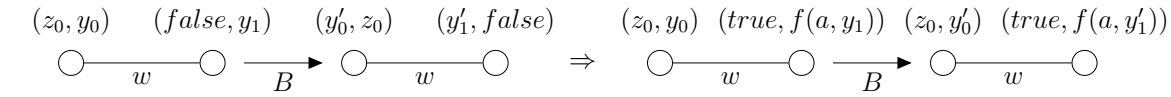
6.4.2 Commutativité des règles deux à deux et préservation des formes normales

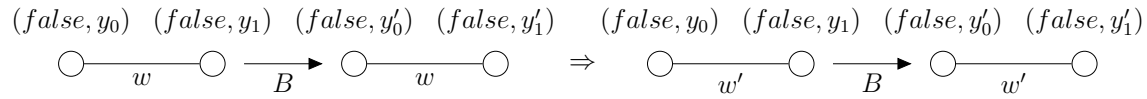
Nous reprenons ici la technique de preuve utilisée dans la section précédente : nous montrons que toute exécution de $A' \cup (C \cup B')$ est équivalente à une exécution globalement séquentielle de $A' \cup (C \cup B')$ et nous nous concentrons sur l'étude de ce type d'exécutions. Ce raisonnement se base sur la commutation des règles de réétiquetage deux à deux. Toutefois, ici seule le fait que B' commute avec A' peut être déduit directement de leurs définitions. Pour assurer la commutation de C avec A', il est nécessaire de nous assurer de la propriété suivante :

> (v) lorsque la terminaison de A à été détectée sur un sommet, l'étiquette de ce dernier, ainsi que de toute arête adjacente ne sera plus modifiée.

Pour nous assurer que la fonction passée en paramètre à l'opérateur de copie et les opérations effectuées par B commutent, nous devons vérifier que les propriétés suivantes sont assurées (on rappelle que f correspond au paramètre de l'opérateur de copie) :

(vi) 

(vii) 

(viii) 

En ce qui concerne la préservation des formes normales, $Copy(f)$ ne modifiant pas la première composante de l'étiquetage de sommet, ni l'étiquetage des arêtes, ce dernier préserve les formes normales de A'. De la même manière, B' ne modifiant ni le premier ni la seconde composante booléenne de l'étiquetage, B' préserve à la fois les formes normales de A' et de $Copy(f)$.

6.4.3 Réalisation

Une fois la commutativité deux à deux des règles et la préservation des formes normales prouvées, et sachant que A' (resp. B') simule A (resp. B) et donc réalise T_A resp. T_B, il nous suffit d'appliquer deux fois le lemme 6.11 page précédente pour prouver que $A' \cup (C \cup B')$ réalise $T_{A \otimes (C(f) \otimes B)}$. Comme dans le chapitre précédent, nous supposons que l'application de f à tout les sommets d'un état final de T_A transforme celui-ci en un état initial de T_B. On notera cette hypothèse (ix).

Lemme 6.12. *[Coq 71] La tache $T_C(f)$ est compatible avec la tache T_A.*

Démonstration. $T_{C(f)}$ étant la tâche correspondant à l'application de la fonction f à tout sommet d'un graphe étiqueté étant un état final pour T_A, celle-ci est compatible par définition avec T_A. □

Lemme 6.13. *[Coq 72] La tache T_B est compatible avec la tache $T_C(f)$.*

Démonstration. $T_{C(f)}$ étant la tâche correspondant à l'application de la fonction f à tout sommet d'un graphe étiqueté étant un état final pour T_A, par (ix) $T_{C(f)}$ est compatible avec T_B. □

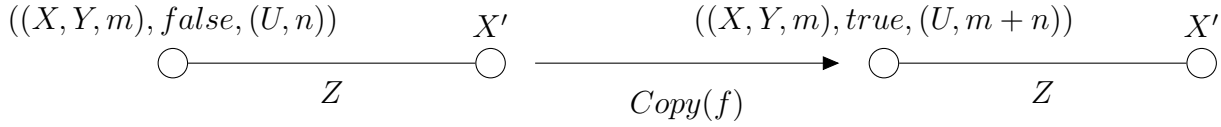
Théorème 6.7. *[Coq 73] Si les propriétés (v) à (ix) sont vérifiés, alors $A' \cup (C \cup B')$ réalise $T_{A \otimes (C(f) \otimes B)}$.*

Démonstration. B' commute avec C et préserve les formes normales de ce dernier, et T_B est compatible avec $T_{C(f)}$. Par le lemme 6.11 page 109 $(C \cup B')$ réalise $T_{(C(f) \otimes B)}$.
De plus, B' et C commutent avec A' et préservent les formes normales de ce dernier. Enfin T_A est compatible avec $T_{C(f)}$, donc avec $T_{(C(f) \otimes B)}$. Par le lemme 6.11 page 109, $A' \cup (C \cup B')$ réalise $T_{A \otimes (C(f) \otimes B)}$. □

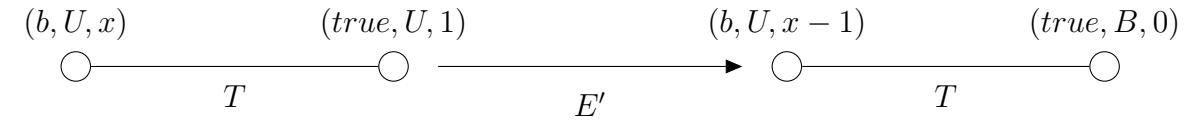
6.4.4 Application à l'exemple et discussion

Nous considérons le calcul d'arbre recouvrant décrit en A.2.3 page 125. Nous devons toutefois adapter le système d'élection de leader, ainsi que la fonction a laquelle sera appliqué l'opérateur de copie. Celle-ci est définie comme la somme du degré du sommet dans l'arbre en cours de construction avec la valeur entière de l'état de l'algorithme d'élection.

Règle 14 Opérateur de copie appliqué à la fonction somme.



Règle 15 Election dans un graphe dont un arbre recouvrant est connu.



On remarque que pour conserver l'information de construction de l'arbre recouvrant en utilisant le théorème 6.7 page précédente, chaque tâche doit mentionner cette information dans ses états finaux. On remarque aussi que le théorème 6.7 page précédente apporte moins d'information que le théorème 6.4 page 105 concernant la détection de terminaison : le premier sens de l'équivalence définissant le mode de détection de terminaison GTD est aisément déductible du théorème 6.7 page précédente, toutefois le second sens doit faire l'objet, dans l'état actuel de nos travaux, d'un raisonnement manuel.

Prouver la compatibilité des tâches est ici trivial. De même les obligations (v) à (viii) sont remplie puisque l'addition est commutative. La difficulté est d'adapter la seconde relation de réétiquetage pour prendre en compte la terminaison de la première. La réalisation de T_B par cette adaptation doit aussi être prouvée, toutefois la simulation permet de réduire l'effort de preuve nécessaire.

On remarquera finalement que bien que moins détaillée au niveau des lemmes compagnons que la version précédente, cette nouvelle composition n'est pas restreinte par la topologie du graphe pour exécuter les deux systèmes en parallèle.

6.5 Conclusion et travaux futurs

Dans ce chapitre, nous avons étudié deux propositions de composition pour les systèmes **LC0** et avons appliqués celles-ci à la preuve d'un système **LC0** de calcul d'arbre recouvrant avec détection locale de la terminaison globale. Pour chacune de nos deux définitions, nous étudions les propriétés des exécutions montrant que celles-ci sont équivalentes à des exécutions *globalement séquentielles*, où les systèmes de réétiquetage sont appliqués les uns après les autres.

La première définition de composition proposé à l'avantage de permettre la deduction du mode de détection de la terminaison de celui du second algorithme. Toutefois, celle-ci ne permet que peu d'exécutions réellement parallèles des deux algorithmes.

La seconde proposition, définie en collaboration avec M. Tounsi, permet des executions parallèles des deux systèmes sur une plus large classe de graphes. Celle-ci requiert toutefois une adaptation plus importante des systèmes que l'on souhaite composer. La définition du second système dépend en effet de la detection de la terminaison. Dans le cas étudié, l'election doit par exemple être modifiée pour s'assurer que lors de l'application, le degré calculé dans l'arbre ne changera plus. On note que cette adaptation est toutefois facilité par les techniques décrites au chapitre précédent.

Une extension de ces travaux aux modes de synchronisation **LC1** et **LC2** serait souhaitable et permettrait de tester ces techniques sur un plus grand nombre d'exemples. Toutefois, dans le cas des règles **LC2**, prouver la commutativité de deux systèmes s'avèrera sans

doute fastidieux. L'utilisation de constructeurs de relations permettrais sans doute de simplifier ces preuves de commutativité.

Il est intéressant de noter que la composition de systèmes ouvre la porte à une technique d'étude formelle de calculs locaux sur les graphes dynamiques [21, 20, 19]. Si l'on considère le calcul d'arbre recouvrant étudié dans ce chapitre, le système d'élection est exécuté sur un sous-arbre variant au cours d'une exécution. Nous pourrions, de la même manière, considérer la composition de deux systèmes, l'un simulant un système de réétiquetage sur un sous-graphe engendré par un certain étiquetage, l'autre modifiant le sous-graphe sur lequel le premier système est appliqué. On note que cette technique permettrait en outre d'étudier la correction d'un algorithme en fonction de diverses hypothèses de mobilité, en changeant le système de modification du sous-graphe.

Chapitre 7

Conclusion

Nous avons défini dans ce mémoire une sémantique des systèmes de calculs locaux dans le *calcul des constructions inductives*. La formalisation en *Coq* proposée nous permet d'étudier, au sein du même environnement la correction de systèmes de réétiquetage ainsi que l'expressivité de sous-classe de systèmes de calculs locaux. Cette formalisation nous a permis de nous assurer de la correction des résultats existant, tout en éliminant, en discutant avec leurs auteurs, les ambiguïtés des définitions "papiers". La définition de schémas de preuves de réalisation ou d'impossibilité permet de simplifier de futurs preuves, que ce soit dans un cadre pédagogique ou à des fins de recherche. Dans le cadre de l'enseignement de l'algorithmique distribuée, l'utilisation de notre bibliothèque et des schémas de preuves définis permettrait selon nous de guider les étudiants dans des raisonnements élégants mais pouvant paraître peu intuitif au premier abord, comme les preuves d'impossibilités. Dans le cadre de la recherche, nos preuves formelles d'impossibilités mettent en valeur les connections entre topologie, modes de synchronisation et détection de terminaison déjà soulignées par Y. Métivier *et al.* Nos efforts de formalisation nous ont permis d'ajouter à la "boîte à outils" théorique des calculs locaux les lemmes d'extension et de composition pour les systèmes **LC0**, et de définir à partir de ceux-ci des schémas de preuves d'impossibilité simples, ne nécessitant pas l'utilisation de morphismes de graphes. Nous résumons dans la suite de ce chapitre ces travaux et proposons des pistes de recherche.

7.1 Sémantique des calculs locaux dans le *CIC*

Nous avons vu dans le chapitre 3 que l'expressivité du *CIC* autorise la construction de types pour les principales structures manipulées dans le cadre de la théorie des calculs locaux. L'approche de formalisation proposée dans ce chapitre est la suivante : nous avons formalisé le sous ensemble de la théorie des graphes formant la base des calculs locaux en *CIC*. Ce sous-ensemble comprend notamment les graphes, graphes étiquetés ainsi que les morphismes spécifiques à ces deux objets. Nous utilisons ces notions pour définir les types des *systèmes de réétiquetage* et *systèmes de réétiquetages localement engendrés*. Nous montrons que la définition des relations de réétiquetages localement engendrées fournie par les travaux ultérieurs est ambiguë, et proposons une définition alternative.

Nous définissons, pour chacun des modes de synchronisation **LC0**, **LC1** et **LC2**, un type de relation spécifique. Pour chacun de ces types, un plongement dans les relations de réétiquetages localement engendrées est fourni. Ces plongements se composent :

- d'une fonction transformant les relations correspondantes aux mode de synchronisations en une relation de réétiquetage.
- d'une preuve que cette fonction produit uniquement des relations de réétiquetage localement engendrées.

Les notions d'invariant de système de réétiquetages, de réalisation et de modes de détection de la terminaison sont ensuite définies. La réalisation et les différents modes de détection de la terminaison sont définis à l'aide d'invariants. Finalement nous proposons un schéma de preuve des systèmes de calculs locaux, en nous basant sur l'exemple d'un système **LC0** de calcul d'arbre recouvrant.

7.2 Étude formelle de l'expressivité des systèmes de calculs locaux

Nous montrons, dans le chapitre 4, comment la sémantique proposée peut être utilisée pour raisonner sur l'expressivité des systèmes de calculs locaux, ainsi que sur les modes de synchronisation. Nous prouvons en *Coq* deux invariants des systèmes **LC0**, inspirés par les travaux de J. Chalopin : le lemme *d'extension* et son corollaire, le lemme de *composition*. Ces deux lemmes, dont l'utilisation se rapproche du lemme de pompage de la théorie des automates finies, nous permettent de prouver formellement l'irréalisabilité de plusieurs tâches par les systèmes **LC0** : le calcul du degré de chaque sommet avec détection locale de la terminaison locale, la construction d'un arbre recouvrant avec détection locale de la terminaison globale et l'impossibilité d'élire un leader. À l'aide du lemme de *composition*, nous proposons une preuve informelle de la restriction de la classe de graphe dans laquelle une élection est impossible par un système **LC0**.

Nous proposons également dans ce chapitre une formalisation de la preuve que l'élection est irréalisable par un système de réétiquetage localement engendrée dans un graphe quelconque. Cette preuve se fonde sur une formalisation du lemme de *relèvement*, lui même basé sur les revêtements, dont nous donnons une définition en *Coq*.

7.3 Transformation des systèmes de calculs locaux et de leurs preuves

Dans le chapitre 5 nous proposons une définition en *CIC* de la notion de *forward simulation*. Nous utilisons celle-ci pour démontrer la correction d'un algorithme d'élection dans un graphe connexe dont un arbre recouvrant est connu à priori. Nous montrons ensuite

comment un ensemble de transformations de systèmes de réétiquetage **LC0** réutilisables permet de simplifier les preuves de simulation. Nous utilisons enfin ces transformations pour formaliser la transformation de systèmes **LC0** avec détection locale de la terminaison globale (GTD) en système **LC0** avec détection observée de la terminaison.

Nous terminons ce mémoire en exposant dans le chapitre 6 les travaux effectués en collaboration avec M. Tounsi sur la composition de systèmes de réétiquetage, et plus particulièrement de système **LC0**. Nous proposons et implantons en *Coq* une technique de preuve fondée sur la commutativité des règles de calcul d'un système pour transformer ces exécutions en exécutions globalement séquentielles et faciliter l'étude d'exécution entrelacé. Nous appliquons cette technique à deux formes de composition de systèmes **LC0**.

7.4 Travaux futurs

Le volet "preuve de systèmes de calculs locaux" de ces travaux manque d'automatisation par rapport aux autres travaux portant sur la preuve formelle d'algorithmes distribués. La création d'un générateur d'obligations de preuve et l'intégration de prouveurs automatiques, à la manière de *Rodin* ou *Why* simplifierais les preuves de corrections dans notre environnement. La définition de tactiques spécifiques à chaque mode de synchronisation permettrait aussi de réduire la taille des preuves. Une généralisation du style de preuve mis en place dans le cadre de la simulation simplifierait l'écriture de ces tactiques.

Dans des travaux récents, P. Castéran étudie en particulier des règles de calculs déterministes, représentées par des fonctions *Coq*, pour lesquelles la définition d'une notion de composition est simplifiée. Même si les schémas de preuves restent à définir dans cette interprétation des calculs locaux, cette piste nous semble prometteuse : elle permet l'exécution et le test de systèmes de réétiquetage au sein même de *Coq*, tout en autorisant l'extraction de ceux-ci. Il nous semble en effet difficile, dans l'interprétation relationnelle fournie ici, d'utiliser les facilités d'extraction de *Coq* pour produire un algorithme distribué exécutable par ViSiDIA par exemple. L'interprétation relationnelle pourrait quand à elle être étendue pour autoriser l'étude d'exécutions infinies et des systèmes auto-stabilisants, à l'aide des types co-inductifs de *Coq*.

Les travaux présentés dans les deux derniers chapitres doivent être simplifiés et étendus aux relations **LC1** et **LC2**. L'utilisation de la composition dans le cadre de preuves d'impossibilité est aussi une piste de recherche intéressante. La composition de systèmes séquentiels est couramment utilisée pour réduire un problème de décision à un autre. Il serait intéressant de pouvoir produire des preuves d'impossibilité de ce style dans le cadre des systèmes de calculs locaux. Enfin une combinaison des notions de composition et de simulation permet selon nous de représenter et d'étudier les systèmes de réétiquetage sur les graphes dynamiques sans changer fondamentalement de modèle de calcul. Cette

représentation des systèmes de réétiquetage sur graphe dynamique permettrait la comparaison formelle de l'expressivité des deux modèles de calculs au sein du même environnement. Il serait aussi intéressant de formaliser la preuve d'équivalence entre les systèmes de réétiquetage localement engendrés d'autres modèles : les agents mobiles, les protocoles de populations, les modèles par passage de messages, etc. La construction de passerelles formalisées entre modèles nous semble être une tâche importante dans un domaine où les représentations abondent.

Annexe A

Preuves Formelles de Systèmes de Calculs Locaux Formalisées

Nous listons ici les preuve de réalisation formalisées à l'aide de nos bibliothèques *Coq*. Pour chacune des preuve de réalisation, nous donnons une définition du système sous forme de règles de calculs et de terme *Coq* et une brève explication de son fonctionnement. Nous résumons les invariants utilisé et l'idée générale de preuve. Pour chaque système de calcul, le nom du fichier java contenant son implémentation *ViSiDIA* est donné.

A.1 Calcul du degré de chaque sommet

Tache 5 Tâche de calcul du degré de chaque sommet sans connaissances initiales

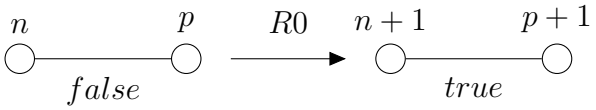
```
Definition st_in := state L_unit L_unit.
Definition st_out := state L_nat L_unit.

Definition Postcond G (i:st_in) (s:st_out)  :=
  For_each_vertex v of G, degree G v = lambda s v.

Definition Degree_Task  :=
  Build_Task   (fun G i => True)  Postcond.
```

A.1.1 Calcul des degrés LC0 avec détection de terminaison implicite

On donne ici une réalisation **LC0** de la tâche 5. On définit un état initial du système comme un état ou chaque sommet est étiqueté par 0, et chaque arête par $false$. Le système

de réétiquetage **LC0** étudié ici traite les arêtes une à une, incrémentant au fur à mesure de leur traitement l'étiquette de chacune de leurs extrémités.

Règle 16 Système **LC0** de calcul du degré de chaque sommet. [Degree_LC0.java]




```
Inductive R0 : LC0_relabeling nat bool :=
  R0_intro : forall n p:nat, R0 n p false (S n) (S p) true.
```

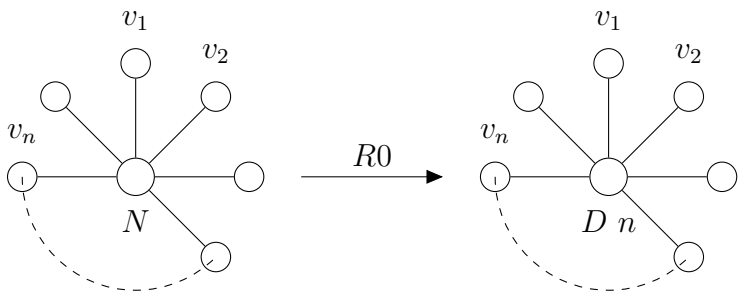
La preuve de ce système se base sur l'invariant suivant : l'étiquette d'un sommet est égale au nombre des arêtes lui étant adjacentes marquées par *true*. Dans une forme normale, si il existe un sommet dont l'étiquette est inférieur a son degré, un pas de calcul est possible ce qui contredit le fait que l'on considère une forme normale. La terminaison peut être montrée en remarquant que le nombre d'arêtes étiquetés par $false$ décroît à chaque pas de calcul.

A.1.2 Calcul du degré LC1 avec détection locale de la terminaison locale

On donne ici une réalisation de la tâche 5 page précédente par un système de réétiquetage **LC1**. On considère un étiquetage des sommets appartenant au type $N|D\ x$ ou x est un entier naturel. Lors d'un pas de calcul, le centre de la boule prend pour étiquetage $D\ n$ où n est son degré.

Ici la preuve se base sur le fait que pour tout sommet c étiqueté par $D\ n$, n est égale au degré de c. La terminaison peut être prouvée en montrant que le nombre de sommets étiquetés par N décroît à chaque transition.

Règle 17 Système **LC1** de calcul du degré avec détection locale de la terminaison locale. [Degree_LC1.java]




```
Inductive R0  : LC1_relabeling nat unit :=
  R0_intro: forall loc,  R0 N (D (crown_size loc)) loc VLabel.empty.
```

A.2 Calcul d'un arbre recouvrant

On considère ici la tâche de calcul d'un arbre recouvrant dans un graphe connexe. On supposera qu'un sommet "racine" initie le calcul. La racine est distingué dans le domaine de la tâche par un sommet étiqueté par *true*. On suppose que les arêtes ne portent aucune information.

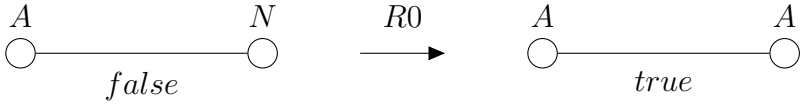
Tache 6 Calcul d'arbre recouvrant

```
Notation st_in := (state L_bool L_unit).
Notation  st_out := (state L_bool L_bool).

Definition Input  G (i:st_in ) :=
  Not_Null G ∧
  Connected G  ∧
  For_one_vertex v of G, lambda  i v = true.

Definition marked_subgraph G (s_out : st_out) :=
  graph_simple_filter  (fun b:bool => b) (fun b:bool => b) s_out G.

Definition Postcond  G (i : st_in)(s : st_out): Prop :=
  Spanning_Tree (marked_subgraph G s ) G.

Definition SP_Task :=   Build_Task  Input  Postcond.
```

A.2.1 Calcul LC0 d'arbre recouvrant avec détection de la terminaison locale

Nous étudions ici un système **LC0** de calcul d'arbre recouvrant. Les sommets sont étiquetés par A ou N, les arêtes par un booléen. Initialement un unique sommet est étiqueté par A et toutes les arêtes sont étiquetés par $false$. A chaque pas de calcul un sommet est ajouté à l'arbre.

Règle 18 Algorithme de calcul d'arbre recouvrant avec détection locale de la terminaison locale [Spanning_Tree_LC0.java]




```
Inductive R0 : LC0_relabeling A_N bool :=
  R0_intro : R0 A N false A A true.
```

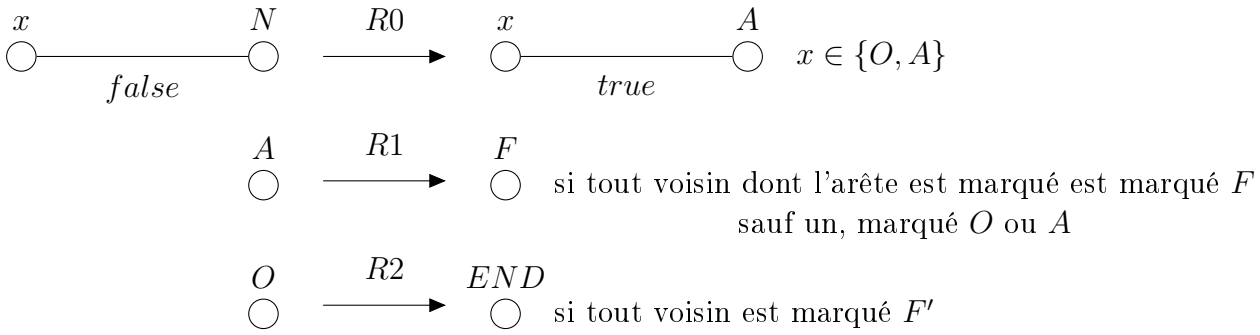
La preuve de ce système se fonde sur le fait que le sous graphe des sommets marqués par A et des arêtes marquées par *true* est un arbre (pour plus de détail sur les invariants de ce système se référer à la section 3.8.3 page 51). Si un sommet est marqué par N on prouve qu'un pas de calcul est possible. Une forme normal ne contient donc que des sommets étiquetés A. On obtient bien un arbre recouvrant. La terminaison peut aisément être prouvée en montrant qu'à chaque transition, le nombre de sommets marqués par N décroît.

A.2.2 Calcul d'arbre recouvrant LC1 et détection locale de la terminaison globale

On considère ici un système de réétiquetage fonctionnant de manière similaire au système précédent. On ajoute une phase de report de la terminaison à la manière de l'algorithme de Dijkstra-Scholten. Un sommet est étiqueté par un des symboles O, A, N F ou N. Les arêtes sont étiquetés par *true* ou $false$. Initialement la racine est étiqueté par O. Tous les autres sommets sont étiquetés par N. Toutes les arêtes sont étiquetés par $false$.

La correction partielle de ce système peut être déduite en remarquant que celui-ci simule le système **LC0** de calcul d'arbre recouvrant. La terminaison peut elle aussi être simplement prouvée en assignant une valeur entière aux étiquettes de sommets ($\{N \mapsto 3, O \mapsto 2, A \mapsto 2, F \mapsto 1, END \mapsto 1\}$). Malheureusement, démontrer que ce système détecte localement la terminaison globale ne peut être déduit directement par la simulation. Pour simplifier, nous prouvons par récurrence sur les exécutions que si il existe dans le graphe un sommet marqué O, A ou N, alors un pas de calcul est possible. Nous prouvons qu'au moins un sommet du graphe est marqué par O ou END. Ainsi nous savons que dans une forme normale un sommet est marqué par END. Nous montrons de plus que si un sommet est étiqueté par END, tout autre sommet est étiqueté par F, et donc plus aucun pas de calcul n'est possible. Nous obtenons ainsi la seconde direction de l'équivalence définissant le mode de détection locale de la terminaison globale.

Règle 19 Système **LC1** de calcul d'arbre recouvrant GTD [Spanning_Tree_LC1.java]




```
Definition neighbours_card_ge n (p:Lv→bool→bool) m :bool :=
  Compare_dec.leb n
    (VLabel.cardinal (VLabelFacts.filter (fun v l=> p (fst l) (snd l)) m)).

Definition O_or_A_neighbour lv le :=
  match lv,le with O, true => true
                 | A,true => true
                 | _,_    => false
end.

Inductive R0: LC0_relabeling Lv bool :=
  |R0_O : R0 O N false O A true
  |R0_A : R0 A N false A A true.


Inductive R1: LC1_relabeling Lv bool :=
  R1_intro : forall occ,
    No_neighbour_is  occ (N,false) →
    neighbours_card_ge 2 O_or_A_neighbour occ = false →
    R1 A F occ occ.

Inductive R2: LC1_relabeling Lv bool :=
  R2_intro : forall occ,
    (∀ x, forall l, VLabel.find x occ = Some l → fst l = F) →
    R2 O END occ occ.
```

A.2.3 Calcul d'arbre recouvrant LC0 avec calcul du degré de chaque sommet dans l'arbre, avec détection locale de la terminaison locale

Nous considérons ici une variation de la tâche 6 page 122, ou dans le domaine de la tâche, le degré de chaque sommet est connu. Dans les états finaux, tous les sommets connaissent leur degré dans l'arbre construit. On remarque que dans la définition de la tâche 7 le terme *Input* mentionné à l'intérieur de la définition de *Input* fait référence au domaine de la tâche 6 page 122.

Tache 7 Calcul d'arbre recouvrant avec calcul des degrés des sommets dans l'arbre

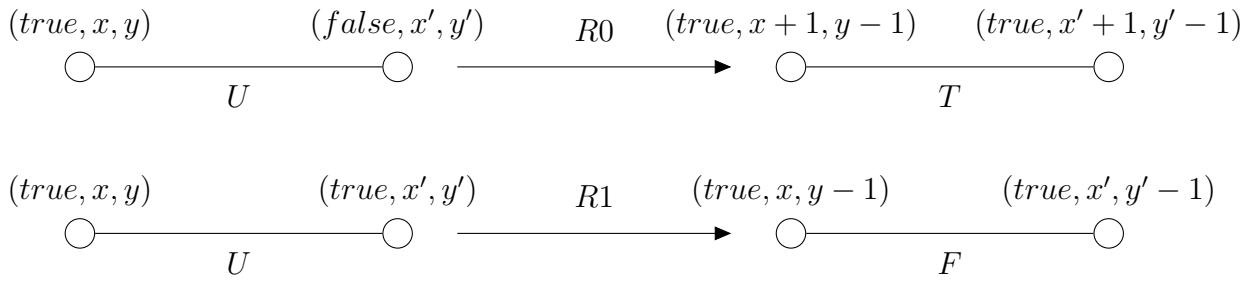
```
Definition Input G (i:state (L_product L_bool L_nat) L_unit) :=
  Input G (state_map (fun x=>fst x) (fun x=>x) i) ∧
  (forall v,  Vertex_of v G → snd (lambda  i v) = degree G v).

Definition Output G (i:state (L_product L_bool L_nat) L_unit)
                    (s:state (L_product L_bool L_nat) L_bool) :=
  let Span := (graph_simple_filter
    (fun b:(bool*nat) => fst b) (fun b:bool => b) s G) in
  Spanning_Tree Span G ∧
  forall v, Vertex_of v Span→
    snd (lambda s v) = degree Span v.

Definition SP_Task_Degree := Build_Task Input Output.
```

Le système réalisant la tâche 7 procède de la manière suivante : initialement chaque sommet v est étiqueté par $(false, d(v), 0)$ ou $d(v)$ est le degré de v, sauf la racine r qui elle est étiqueté par $(true, d(r), 0)$. Dans l'étiquette (X, n, m) d'un sommet, X correspond à l'appartenance à l'arbre, m au nombre d'arêtes adjacentes non traité et n au degré du sommet dans l'arbre en cours de construction. Toute les étiquettes sont initialement étiquetés par U.

Lorsqu'une arête peut être ajoutée à l'arbre (si ses extrémités sont étiquetées respectivement par *true* et $false$), son étiquette devient T, l'appartenance à l'arbre de l'extrémité droite est modifiée, et pour chacune des extrémités le degré de chaque sommet dans l'arbre est incrémenté et le degré des arêtes restant à traiter est décrémenté. Lorsque les deux extrémités d'une arête étiquetée U sont dans l'arbre, l'étiquette de celle-ci devient F, et le nombre d'arêtes restant à traité de chacune des extrémités est décrémenté.

Nous réalisons la preuve de ce système par étape de simulations successives, à la manière d'un développement B. Une première version du système remplace les étiquettes A et N

Règle 20 Système **LC0** de calcul d'arbre recouvrant avec calcul du degré de chaque sommet dans l'arbre construit. [Spanning_Tree_Degree_LC0.java]




```
Definition Lv := (bool*nat*nat)%type.
Inductive  Le := U|T|F.

Inductive R0 : LC0_relabeling Lv Le :=
  | R0_intro: forall x x' y y',
    R0 (true,x,S y) (false, x',S y') U
       (true, S x, y) (true, S x', y' ) T.

Inductive R1 : LC0_relabeling Lv Le :=
  | R1_intro: forall x y x' y',
    R1 (true,x,S y) (true,x',S y') U
       (true, x, y) (true, x', y') F.
```

par des booléens. La seconde version introduit le calcul du degré de chaque sommet dans l'arbre. Dans une troisième version, les trois différents états pour les arêtes sont introduits. On montre ici que la valeur calculé par les sommets correspond au degré dans le sous graphe contenant les sommets étiqueté par $(true, x)$ et les arêtes étiquetés par T. Finalement, dans la version présenté ci-dessus, la détection locale de la terminaison locale est ajouté. L'utilisation ici d'une technique "à la B" n'est pas réellement utile, les preuves d'invariants n'étant que peu simplifiées. Toutefois, cela montre que nos bibliothèque permette l'utilisation de cette technique de preuve.

A.2.4 Calcul d'arbre recouvrant LC0 avec détection locale de la terminaison globale

À l'aide de la première notion de composition présentée dans le chapitre 6, nous proposons un système réalisant la tâche 7 page précédente. La définition du système et sa

preuve sont fournis dans ce chapitre.

A.3 Élection d'un leader

Nous considérons maintenant le problème de l'élection d'un leader dans un graphe. On note que la tâche définie ici est paramétrée par un domaine, nous permettant de faire varier facilement les connaissances initiales d'une preuve à l'autre. On note que L_result est le type d'étiquette correspondant au type *result*. Pour simplifier l'énoncé, sa définition n'est pas donnée ici.

Tache 8 Définition paramétrique de la tâche d'Élection d'un leader

```
Variables (in_v in_e : Type)
          (L_in_v : LabelType in_v)
          (L_in_e : LabelType in_e).

Inductive result : Set := elected | beaten.

Notation st_in := (state L_in_v L_in_e).
Notation st_out := (state L_result L_unit).

Variable (Pre : Graph_t → st_in → Prop).

Definition Postcond (G:Graph_t)(i : st_in )(s:st_out) :=
  For_one_vertex v of G,  lambda  s v = elected.

Definition Election_Task := Build_Task Pre  Postcond.
```

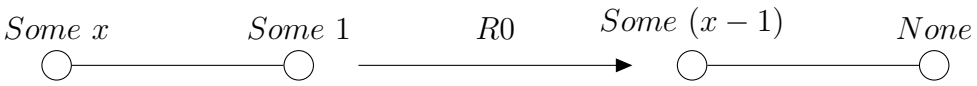
A.3.1 Élection LC0 dans un arbre dont le degré de chaque sommet est connue, avec détection globale de la terminaison

Nous considérons ici le problème de l'élection dans un arbre contenant au moins deux sommets où chaque sommet est initialement étiqueté par son degré. La tâche correspondante est la tâche 9 page suivante. Pour réaliser celle-ci, nous proposons la règle 22 page 129. Les sommets sont étiquetés par le type *option nat* (i.e. soit un entier, soit une valeur par défaut). Les arêtes ne portent aucune information. Initialement tout sommet est étiqueté par son degré. L'algorithme élague l'arbre (en supprime les feuilles) au fur et à mesure de l'exécution. A la fin un unique sommet, le sommet élue, est étiqueté par *Some* 0, tout les autres étant étiquetés par *None*.

Tache 9 Élection d'un leader dans un arbre avec connaissance du degré de chaque sommet

```
Notation in_v  := nat.
Notation in_e  := unit.
Notation st_in := (state L_nat L_unit).

Definition Labeled_Tree G (i:st_in) :=
  Not_Null G ∧ Tree G ∧
  (For_each_vertex v of G, lambda i v = (degree G v)) ∧
  exists x : Vertices.elt,
  exists y : Vertices.elt, Vertex_of x G ∧ Vertex_of y G ∧ x <> y.

Definition Election_Trees_Task := Election_Task   Labeled_Tree.
```

Règle 21 Système **LC0** d'élection dans un arbre dont le degré de chaque sommet est connue [Election_Tree_LC0.java]




```
Inductive R0  : LC0_relabeling (option nat) unit :=
  R0_intro : forall (x :nat), R0 (Some (S x)) (Some 1) tt (Some x) None tt.
```

On appel un sommet étiqueté par *None* un sommet battu. La preuve de ce système se base sur le fait que chaque sommet est étiqueté par son degré dans le sous arbre des sommets non battus. On peut ainsi montré que lorsqu'une forme normale est atteinte (i) un sommet est étiqueté par *Some* 0 et (ii) chaque sommet étant étiqueté par son degré dans le sous arbre des sommets non battu, tout autre sommet est battu.

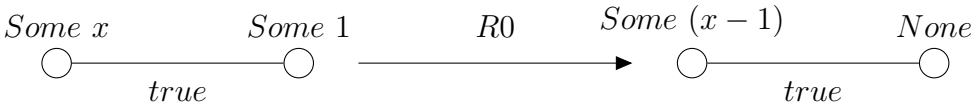
A.3.2 Élection LC0 dans un graphe dont un arbre recouvrant est connu

On considère ici une variation de la tâche 9. Le domaine de la nouvelle tâche est défini comme l'ensemble des graphes connexes dont un arbre recouvrant est connu. On suppose que chaque sommet est étiqueté par son degré dans cet arbre. Le système réalisant cette tâche ne change que très peu par rapport au système réalisant la tâche 9 : seul une condition sur les arêtes est ajoutée. La règle de réétiquetage s'applique maintenant uniquement sur le sous arbre engendré par les arêtes étiqueté par *true*. Nous discutons de la preuve de ce système en 5.3 page 81.

Tache 10 Election dans un graphe dont un arbre recouvrant est connu. Les sommets sont étiquetés par leur degré dans cette arbre.

```
Definition Spannig_In_Input G (i:state (L_product L_bool L_nat) L_bool) :=
  let Span := (graph_simple_filter
    (fun b:(bool*nat) => true) (fun b:bool => b) i G) in
    Not_Null G ∧
    Spanning_Tree Span G ∧
    (exists x : Vertices.elt,
      exists y : Vertices.elt,
        Vertex_of x G ∧  Vertex_of y G ∧ x <> y) ∧
    For_each_vertex v of G, snd (lambda i v) = degree Span v.

Definition Election_Subtree_Task := Election_Task Spanning_In_Input.
```

Règle 22 Système **LC0** d'élection dans un graphe connexe dont un arbre recouvrant et le degré de chaque sommet dans cet arbre est connue [Election_SubTree_LC0.java]




```
Inductive R0: LC0_relabeling (option nat) bool :=
  | R0_intro: forall x, R0 (Some (S x)) (Some 1) true (Some x) None true.
```

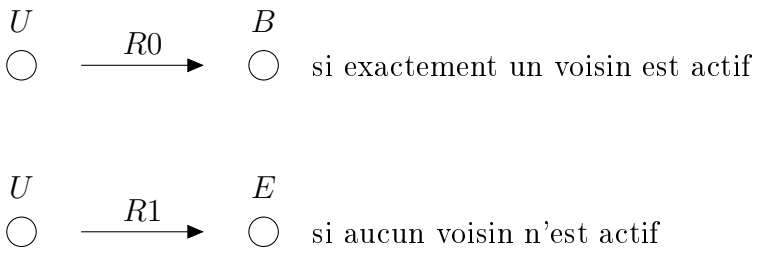
A.3.3 Élection LC1 dans les arbres

Ici nous considérons la tâche de l'élection d'un leader dans un arbre sans aucune connaissance initiale. Encore une fois Nous redéfinissons uniquement le domaine de l'élection. Nous considérons une réalisation dans laquelle les sommets sont étiquetés par U (statut inconnue), B (sommet battu) ou E (sommet élu). Les arêtes ne sont pas étiquetées. On dit qu'un sommet est *actif* si il n'est ni battu ni élu. Le système procède, comme dans les cas précédent, par élagage. Si un sommet c possède exactement un voisin actif, c est éliminé. Si un sommet c ne possède pas de voisins actif, alors c est élu.

Tache 11 Election dans arbre sans connaissance initiale.

```
Definition Unlabeled_Tree G (i:state L_unit L_unit) :=
  Not_Null G / Tree G.

Definition Election_Trees_Uniform_Task := Election_Task Unlabeled_Tree.
```

Règle 23 Système **LC1** d'élection dans un arbre. [Election_Tree_LC1.java]




```
Inductive Lv := E:Lv | B:Lv | U:Lv.
Definition Le:= unit.

Definition active_neighbourhood (s:crown_state Lv Le) :=
  VLabelFacts.filter
    (fun k l => match (fst l) with U=>true | _ => false end)  s.

Inductive R0: LC1_relabeling Lv Le :=
  | R0_intros : forall s,
    (VLabel.cardinal (active_neighbourhood s)) = 1 →
    R0 U B s s.

Inductive R1 : LC1_relabeling Lv Le :=
  |R1_intro: forall s ,
    (VLabel.cardinal (active_neighbourhood s)) = 0 →
    R1 U E s s.
```

Référence des termes Coq

[Coq 0] graph_t, Graphs/Graph.v.
[Coq 1] morphism, Graphs/Graph.v.
[Coq 2] locally_generated, GRS/GraphRelabeling.v.
[Coq 3] LC0_relabeling, GRS/LC0.v.
[Coq 4] LC0_Step, GRS/LC0.v.
[Coq 5] LC0_Step_at_center, GRS/LC0_locally_generated.v.
[Coq 6] LC0_locally_generated, GRS/LC0_locally_generated.v.
[Coq 7] crown_state, Graphs/StateOperations.v.
[Coq 8] LC1_relabeling, GRS/LC1.v.
[Coq 9] LC1_Step, GRS/LC1.v.
[Coq 10] lc1, GRS/LC1.v.
[Coq 11] LC1_Locally_generated, GRS/LC1_Locally_generated.v.
[Coq 12] lc2, GRS/LC2.v.
[Coq 13] LC2_Step, GRS/LC2.v.
[Coq 14] invariant_i, GRS/GraphRelabeling.v.
[Coq 15] Task, GRS/Tasks.v.
[Coq 16] partial_correctness_invariant, GRS/Tasks.v.
[Coq 17] extension_lemma, GRS/LC0.v.
[Coq 18] degree_not_LC0_computable, Examples/No_Degree_LC0_LTD.v.
[Coq 19] No_Spanning_Tree_LC0_GTD, Examples/No_Spanning_LC0_GTD.
[Coq 20] Contradiction, Examples/No_Election_Covering.v.
[Coq 21] Lifting, GRS/Lifting.v.
[Coq 22] Covering, Graphs/Graph.v.
[Coq 23] LabeledGraph_Covering, Graphs/Labeling.v.
[Coq 24] Strict_Covering, Examples/No_Election_Covering.v.
[Coq 25] lift, GRS/Lifting.v.
[Coq 26] lift_covering, GRS/Lifting.v.

[Coq 27] Lifting_Steps_aux_2, GRS/Lifting.v.
[Coq 28] Lifting_Steps_aux_1, GRS/Lifting.v.
[Coq 29] Lifting_Steps, GRS/Lifting.v.
[Coq 30] Covering_Balls_Empty, Graphs/GraphFacts.v.
[Coq 31] Composition, GRS/Composition.v.
[Coq 32] Election_in_H, Examples/No_Election_Covering.v.
[Coq 33] Election_in_G, Examples/No_Election_Covering.v.
[Coq 34] Application, Examples/No_Election_Covering.v.
[Coq 35] composition_lemma, GRS/LC0.v.
[Coq 36] Two_elected, Examples/No_Election_LC0.v.
[Coq 37] Forward_Simulation, GRS/Simulation_alt.v.
[Coq 38] Simulation_System_Invariant, GRS/Simulation_alt.v.
[Coq 39] Refinement_relation, Election_Subtree_LC0_GTD.v.
[Coq 40] FS_Id, GRS/Simulation_alt.v.
[Coq 41] LC0_internal_trans, GRS/LC0_operators.v.
[Coq 42] LC0_or, GRS/LC0_operators.v.
[Coq 43] FS_Union, GRS/Simulation_alt.v.
[Coq 44] LC0_guard, GRS/LC0_operators.v.
[Coq 45] FS_Guard, GRS/Simulation_alt.v.
[Coq 46] LC0_vertices_product_alt_left, GRS/LC0_operators.v.
[Coq 47] FS_LProduct, GRS/Simulation_alt.v.
[Coq 48] GRc_Simulates_GR, GRS/GTD_OTD_LC0.v.
[Coq 49] Realizes_T_GRc, GRS/GTD_OTD_LC0.v.
[Coq 50] tau_otd_stable, GRS/GTD_OTD_LC0.v.
[Coq 51] Possible_step, GRS/GTD_OTD_LC0.v.
[Coq 52] DMap_alt, GRS/DMap.v.
[Coq 53] Cooperativity_0, GRS/LC0_composition.v.
[Coq 54] composition_Refines_A, GRS/LC0_composition.v.
[Coq 55] composition_Realizes_TA, GRS/LC0_composition.v.
[Coq 56] LC0_composition_terminates_from, GRS/LC0_composition.v.
[Coq 57] Sequentialize_aux, GRS/GraphRelabeling.v.
[Coq 58] B'_Strongly_Refines_B, GRS/LC0_composition.v.
[Coq 59] GTD_dir_1, GRS/GraphRelabeling.v.
[Coq 60] GTD_dir_2, GRS/GraphRelabeling.v.

[Coq 61] Is_Gtd_composition, GRS/GraphRelabeling.v.
[Coq 62] SP_GTD_Partially_Correct, Examples/SpanningTree_LC0_GTD_Degree.v.
[Coq 63] Strong_Ltd, Examples/SpanningTree_LC0_GTD_Degree.v.
[Coq 64] Election_Subtree_GTD, Refinement_Examples/Election_Subtree_LC0_GTD.v.
[Coq 65] Edges_dont_change, Examples/SpanningTree_LC0_GTD_Degree.v.
[Coq 66] Each_label_changes, Examples/SpanningTree_LC0_GTD_Degree.v.
[Coq 67] Is_Gtd_SP_GTD, Examples/SpanningTree_LC0_GTD_Degree.v.
[Coq 68] T_Compose, GRS/papier_decomp.v.
[Coq 69] Compatibility, GRS/papier_decomp.v.
[Coq 70] Realizes_TC, GRS/papier_decomp.v.
[Coq 71] Ta_Task_copy_compat, GRS/papier_decomp.v.
[Coq 72] Task_copy_Tb_compat, GRS/papier_decomp.v.
[Coq 73] Realizes_A_copy_b, GRS/papier_decomp.v.

Index des Notions

References

[1] The coq users' contributions. http://coq.inria.fr/pylons/contribs/index.

[2] Le langage ocaml. http://caml.inria.fr, 2011.

[3] Martín Abadi, Ricardo Corin, and Cédric Fournet. Computational secrecy by typing for the pi calculus. In *APLAS*, pages 253–269, 2006.

[4] Jean-Raymond Abrial. *The B-book : assigning programs to meanings*. Cambridge University Press, New York, NY, USA, 1996.

[5] Jean-Raymond Abrial. *Modeling in Event-B : System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

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

[7] Flemming Andersen. *A Theorem Prover for Unity in Higher Order Logic*. PhD thesis, Technical University of Denmark, 1992.

[8] Flemming Andersen, Kim Dam Petersen, and Jimmi S. Pettersson. Program verification using hol-unity. In *Higher Order Logic Theorem Proving and Its Applications : HUG '93, LNCS 780*, pages 1–15. Springer, 1994.

[9] David P. Anderson. Boinc : A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[10] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *STOC*, pages 82–93, 1980.

[11] Myla Archer. Tame : Using PVS strategies for special-purpose theorem proving. *Annals of Mathematics and Arti cial Intelligence*, 2000.

[12] Myla Archer, Constance Heitmeyer, and Elvinia Riccobene. Proving invariants of I/O automata with TAME, 2002.

[13] Myla Archer, Constance Heitmeyer, and Steve Sims. TAME : A PVS interface to simplify proofs for automata models. In *In Proc. User Interfaces for Theorem Provers 1998 (UITP '98*, 1998.

[14] Elisabeth Ball and Michael Butler. Event-B patterns for specifying fault-tolerance in multi-agent interaction. In Michael Butler, Cliff Jones, Alexander Romanovsky, and

Elena Troubitsyna, editors, *Methods, Models and Tools for Fault Tolerance*, volume 5454 of *Lecture Notes in Computer Science*, pages 104–129. Springer Berlin Heidelberg, 2009.

[15] Michel Bauderon, , Yves Métivier, and Affif Mosbah, Mohamed Sellami. From local computations to asynchronous message passing systems, 2002.

[16] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art : The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004.

[17] Dominique Cansell and Dominique Méry. Formal and incremental construction of distributed algorithms : on the distributed reference counting algorithm. *Theor. Comput. Sci.*, 364(3) :318–337, November 2006.

[18] Dominique Cansell, Dominique Méry, and Joris Rehm. Time constraint patterns for Event-B development. In Jacques Julliand and Olga Kouchnarenko, editors, *B 2007 : Formal Specification and Development in B*, volume 4355 of *Lecture Notes in Computer Science*, pages 140–154. Springer Berlin / Heidelberg, 2006.

[19] A. Casteigts, S. Chaumette, and A. Ferreira. Characterizing topological assumptions of distributed algorithms in dynamic networks. In *Proc. of 16th Intl. Conference on Structural Information and Communication Complexity (SIROCCO'09)*, volume 5869 of *Lecture Notes in Computer Science*, pages 126–140, Piran, Slovenia, May 2009. Springer-Verlag.

[20] Arnaud Casteigts. *Contribution à l'Algorithmique Distribuée dans les Réseaux Mobiles Ad Hoc - Calculs Locaux et Réétiquetages de Graphes Dynamiques*. PhD thesis, University of Bordeaux, 2007.

[21] Arnaud Casteigts and Serge Chaumette. Dynamicity aware graph relabeling systems (da-grs), a local computation based model to describe manet algorithms. In *IASTED PDCS'05*, pages 231–236, 2005.

[22] Pierre Castéran and Vincent Filou. Tasks, types and tactics for local computation systems. *Studia Informatica Universalis*, 9.1 :39–86, 2011.

[23] Jérémie Chalopin. *Algorithmique Distribuée, Calculs Locaux et Homomorphismes de Graphes*. PhD thesis, Université de Bordeaux 1, LaBRI, 2006.

[24] Jérémie Chalopin, Emmanuel Godard, and Yves Métivier. Local terminations and distributed computability in anonymous networks. In *DISC*, pages 47–62, 2008.

[25] Jérémie Chalopin, Emmanuel Godard, and Yves Métivier. Local terminations and distributed computability in anonymous networks. In *Proceedings of the 22nd international symposium on Distributed Computing*, DISC '08, pages 47–62, Berlin, Heidelberg, 2008. Springer-Verlag.

[26] Jérémie Chalopin and Yves Métivier. Election and local computations on edges. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures*, volume 2987 of *Lecture Notes in Computer Science*, pages 90–104. Springer Berlin / Heidelberg, 2004.

[27] K. Mani Chandy. *Parallel program design : a foundation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[28] Michel Charpentier. *Assistance à la répartition de systèmes réactifs*. PhD thesis, Institut National Polytechnique de Toulouse, novembre 1997.

[29] Michel Charpentier, Mamoun Filali, Philippe Mauran, Gerard adiou, and Philippe Quinnec. Tailoring unity to distributed program design. In *IN INTERNATIONAL WORKSHOP ON FORMAL METHODS FOR PARALLEL PROGRAMMING : THEORY AND APPLICATIONS (FMPPTA98), VOLUME 1388 OF LECTURE NOTES IN COMPUTER SCIENCE*, pages 820–832. Springer Verlag, 1997.

[30] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. A tla+ proof system. In *LPAR Workshops*, 2008.

[31] Ching-tsun Chou. A formal theory of undirected graphs in higher-order logic, 1994.

[32] Ching-tsun Chou. Mechanical verification of distributed algorithms in higher-order logic. *The Computer Journal*, 38 :158–176, 1995.

[33] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2) :345–363, April 1936.

[34] Alberto Ciaffaglione, Matthew Hennessy, and Julian Rathke. Proof methodologies for behavioural equivalence in distributed pi-calculus. In *Proc. Formal Techniques for Networked and Distributed Systems - FORTE 2005*, volume 3731 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[35] Bruno Codenotti, Peter Gemmell, and Janos Simon. Symmetry breaking in anonymous networks, 1995.

[36] Thierry Coquand and Gerard Huet. The calculus of constructions. *Inf. Comput.*, 76(2-3) :95–120, February 1988.

[37] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11) :643–644, November 1974.

[38] Edsger W. Dijkstra and C.S. Scholten. Termination detection for diffusing computations. 1980.

[39] J. Duprat. A coq toolkit for graph theory, rapport de recherche 2001-15. Technical report, 2001.

[40] Jean-Christophe Filliâtre. Proof of Imperative Programs in Type Theory. In *Proceedings of the TYPES'98 workshop*, volume 1657 of *Lecture Notes in Computer Science*. Springer, 1998.

[41] Jean-Christophe Filliâtre. *Preuve de programmes impératifs en théorie des types*. PhD thesis, Université Paris-Sud, July 1999.

[42] Jean-Christophe Filliâtre and Pierre Letouzey. Functors for proofs and programs. In David Schmidt, editor, *Programming Languages and Systems*, volume 2986 of *Lecture Notes in Computer Science*, pages 370–384. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-24725-8_26.

[43] SRI FormalWare. *PVS Specifications and Verification System.* http ://pvs.csl.sri.com/.

[44] John R. W. Glauert. Asynchronous mobile processes and graph rewriting. In *Proceedings of the 4th International PARLE Conference on Parallel Architectures and Languages Europe*, PARLE '92, pages 63–78, London, UK, UK, 1992. Springer-Verlag.

[45] Emmanuel Godard and Yves Métivier. A characterization of families of graphs in which election is possible. In *Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture notes in computer science*, pages 159–171, Spain, 2002. Springer.

[46] Emmanuel Godard, Yves Métivier, Mohamed Mosbah, and Afif Sellami. Termination detection of distributed algorithms by graph relabelling systems. In *Proceedings of the First International Conference on Graph Transformation*, ICGT '02, pages 106–119, London, UK, UK, 2002. Springer-Verlag.

[47] Emmanuel Godard, Yves Métivier, and Gerard Tel. Termination detection of local computations. Technical report, 2009.

[48] Jonathan L. Gross and Thomas W. Tucker.

[49] Pierre-Cyrille Héam, Olga Kouchnarenko, and Jérôme Voinot. Component simulation-based substitutivity managing qos and composition issues. *Sci. Comput. Program.*, 75 :898–917, October 2010.

[50] Matthew Hennessy. *A distributed Pi-calculus.* 2007.

[51] Matthew Hennessy. *A Distributed Pi-Calculus.* Cambridge University Press, New York, NY, USA, 2007.

[52] Matthew Hennessy, Massimo Merro, and Julian Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science*, 322 :615–669, 2004.

[53] C. A. R. Hoare. Communicating sequential processes, 2004.

[54] Sara Kalvala. A formulation of tla in isabelle. In E. Thomas Schubert, Philip Windley, and James Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Applications*, volume 971 of *Lecture Notes in Computer Science*, pages 214–228. Springer Berlin / Heidelberg, 1995.

[55] Leslie Lamport. The temporal logic of actions, 1993.

[56] Leslie Lamport. *Specifying Systems : The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[57] Stéphane Lescuyer. First class containers in coq. *Studia Informatica*, 9(1) :87–127, July 2010.

[58] Huimin Lin. PAM : A process algebra manipulator. In *In Proc. Third Workshop on Computer Aided Verification*, pages 136–146. Springer-Verlag, 1993.

[59] Igor Litovsky, Yves Métivier, and Eric Sopena. Graph relabelling systems and distributed algorithms. In *Handbook of graph grammars and computing by graph transformation*, pages 1–56. World scientific, 2001.

[60] Nancy Lynch and Mark Tuttle. An introduction to input/output automata. 1988.

[61] Nancy Lynch and Frits Vaandrager. Forward and backward simulations part i : Untimed systems (replaces tm-486). Technical report, Cambridge, MA, USA, 1994.

[62] Nancy A. Lynch. Multivalued possibilities mappings. In *Stepwise Refinement of Distributed Systems, volume LNCS 430*, pages 519–543. Springer, 1989.

[63] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, PODC '87, pages 137–151, New York, NY, USA, 1987. ACM.

[64] Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2 :219–246, 1989.

[65] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. A peer-to-peer framework for supporting mapreduce applications in dynamic cloud environments. In Nick Antonopoulos, Lee Gillam, and A. J. Sammes, editors, *Cloud Computing*, volume 0 of *Computer Communications and Networks*, pages 113–125. Springer London, 2010. 10.1007/978-1-84996-241-4_7.

[66] Abadi Martín. Secrecy by typing insecurity protocols. In *TACS*, pages 611–638, 1997.

[67] Antoni Mazurkiewicz. Solvability of the asynchronous ranking problem. *Inf. Process. Lett.*, 28(5) :221–224, August 1988.

[68] Dominique Méry, Mohamed Mosbah, and Mohamed Tounsi. Refinement-based verification of local synchronization algorithms. In Michael Butler and Wolfram Schulte, editors, *FM 2011 : Formal Methods*, volume 6664 of *Lecture Notes in Computer Science*, pages 338–352. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-21437-0_26.

[69] Stephan Merz. http ://www.loria.fr/ merz/projects/isabelle-tla/index.html, 2011.

[70] Robin Milner. An algebraic definition of simulation between programs. Technical report, Stanford, CA, USA, 1971.

[71] Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[72] Sayan Mitra and Myla Archer. Reusable proof strategies for proving abstraction properties of I/O automata. In *Fifth Workshhop on Strategies in Automated Deduction*, pages 79–92, 2004.

[73] Tounsi Mohamed, Mosbah Mohamed, and Mery Dominique. Proving distributed algorithms by combining refinement and local computations. *Automated Verification of Critical Systems*, 35, 2010.

[74] Faron Moller. The edinburgh concurrency workbench. http://homepages.inf.ed.ac.uk/perdita/cwb/.

[75] Mohamed Mosbah and Rodrigue Ossamy. A programming language for local computation graphs. In *MCSEAI'04, Eighth Maghrebian Conference on Software Engineering and Artificial Intelligence*, Sousse, Tunisia, May 2004.

[76] Mohamed Mosbah and Rodrigue Ossamy. A programming language for local computations in graphs : Computational completeness. In *ENC '04 : Proceedings of the Fifth Mexican International Conference in Computer Science*, pages 12–19, Washington, DC, USA, 2004. IEEE Computer Society.

[77] Olaf Muller. *A Verification Environment for I/O Automata Based on Formalized Meta-Theory*. PhD thesis, Technische Universität München, September 1998.

[78] Yves Métivier, Mohamed Mosbah, and Affif Sellami. Proving distributed algorithms by graph relabelling systems : Examples of trees in networks with processor identities. In *AGT2002, ETAPS*, Grenoble, France, April 2002.

[79] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Proceedings of the 15th International Workshop on Computer Science Logic*, CSL '01, pages 1–19, London, UK, UK, 2001. Springer-Verlag.

[80] David Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, 1981.

[81] Christine Paulin-Mohring. Inductive definitions in the system coq rules and properties. In Marc Bezem and Jan Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer Berlin / Heidelberg, 1993.

[82] Randy Pollack. The lego proof assistant. http://www.dcs.ed.ac.uk/home/lego/, 1998.

[83] The PRL Project. Nuprl. http://www.nuprl.org/, 2012.

[84] James Riely and Matthew Hennessy. A typed language for distributed mobile processes (extended abstract). In *POPL*, pages 378–390, 1998.

[85] James Riely and Matthew Hennessy. Distributed processes and location failures. *Theor. Comput. Sci.*, 266(1-2) :693–735, 2001.

[86] Benoit Robillard. Formalization of a generic graph library for coq : Application to the formal verification of dijkstra's shortest path algorithm , rapport de recherche 2012. Technical report, 2012.

[87] Antony Rowstron and Peter Druschel. Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems, 2001.

[88] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. 2001.

[89] Matthieu Sozeau and Nicolas Oury. First-class type classes. In *TPHOLs*, pages 278–293, 2008.

[90] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. pages 149–160, 2001.

[91] "Coq Development Team". *The Coq Proof Assistant Reference Manual*. coq.inria.fr.

[92] Mohamed Tounsi. *Preuve d'Algorithmes Distribués par Raffinement*. PhD thesis, Université de Bordeaux, LaBRI, July 2012.

[93] Mohamed Tounsi, Ahmed Hadj Kacem, Mohamed Mosbah, and Dominique Méry. A refinement approach for proving distributed algorithms : Examples of spanning tree problems. In *Integration of Model-based Formal Methods and Tools - IM_FMT'2009 - in IFM'2009*, Düsseldorf Allemagne, 02 2009.

[94] Projet ViSiDiA. http://visidia.labri.fr.

[95] Toh Ne Win, Michael D. Ernst, Stephen J. Garland, Dilsun Kirli, and Nancy A. Lynch. Using simulated execution in verifying distributed algorithms. *Software Tools for Technology Transfer*, 6 :283–297, 2003.