Des bisimulations pour la sémantique des systèmes réactifs

Sophie Pinchinat

THÈSE

présentée par

Sophie Pinchinat

pour obtenir le titre de Docteur
de l'Institut National Polytechnique de Grenoble

(Arrêté ministériel du 23 novembre 1988)

Spécialité : Informatique

Des Bisimulations pour la Sémantique des Systèmes Réactifs

Thèse soutenue le 8 janvier 1993 devant le jury :

J. Sifakis	président
P. Degano	
F. Vaandrager	rapporteurs
M. Hennessy	
Ph. Jorrand	
P. Schnoebelen	examinateurs

Thèse préparée au sein du Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle.

Résumé

Cette thèse contribue à l'étude des sémantiques pour la spécification et la vérification des systèmes réactifs. Plus précisément, nous comparons des sémantiques comportementales, basées sur la bisimulation avec celles induites par des logiques modales du temps arborescent.

Dans la première partie, nous considérons les modèles d'entrelacement pour les systèmes séquentiels non-déterministes. Plusieurs travaux récents ont montré que dans le cadre des systèmes à branchement infini (c-à-d. non-déterminisme infini), l'équivalence de bisimulation (forte), reconnue comme l'équivalence sémantique de base pour le temps arborescent, est strictement plus fine que celles induites par les logiques du temps arborescent (nous savons depuis longtemps qu'elles coïncident sous l'hypothèse de branchement fini). Nous utilisons les *Processus Ordinaux* de Klop, et, en dérivant une notion de *pouvoir de distinction* d'une équivalence sémantique, nous montrons dans un cadre parfaitement unifié que la bisimulation est plus fine que les logiques du temps arborescent, mais aussi que pour une large classe de combinateurs, les congruences engendrées par les logiques restent strictement plus faibles que la bisimulation.

Dans la deuxième partie de la thèse, nous considérons les modèles d'ordre partiel pour les systèmes parallèles, dans lesquels on dispose d'une définition satisfaisante de l'opération de raffinement de programme (cette notion est liée à la méthode classique de conception hiérarchique des programmes).

Parmi les équivalences sémantiques de la littérature, la history preserving bisimulation est particulièrement intéressante car c'est une congruence pour l'opération de raffinement quand les systèmes n'ont pas d'action invisible. En utilisant une caractérisation de cette équivalence en termes d'une bisimulation avant-arrière, nous exhibons deux caractérisations logiques, ainsi qu'un algorithme de traduction entre ces deux logiques. Nous étudions aussi plusieurs variantes de cette bisimulation avant-arrière.

Enfin, nous élargissons le champ de travail en considérant les modèles d'ordre partiel avec actions invisibles. Nous montrons que la bisimulation avant-arrière susmentionnée, adaptée à ce cadre, coïncide avec la branching bisimulation sur les arbres causaux, mais aussi avec deux nouvelles équivalences : l'équivalence de mixed-ordering branching et la history preserving branching bisimulation, que nous étudions.

Abstract

This thesis contributes to the study of semantics for the specification and the verification of reactive systems. More precisely, we compare behavioral equivalences, based on bisimulation with those induced by branching-time modal logics.

In the first part of the thesis, we consider interleaving models for sequential non-deterministic systems. Few recent works have shown that, in the framework of infinite branching systems (i.e. infinite non-determinism), bisimulation equivalence, acknowledged as the basic branching-time semantic equivalence, is strictly finer than those induced by branching-time logics (we already know that they coincide under the finite branching hypothesis). We use *Ordinal Processes* of Klop, and by deriving a notion of *distinguishing power* of a semantic equivalence, we show in a unified framework that bisimulation is finer than branching-time logics, but also that for a wide class of program combinators, the generated congruences by the logics are still weaker than bisimulation.

In the second part of the thesis, we consider partial order based models for concurrent systems, and in which we have a satisfactory definition of program refinement operation (this notion is related to the classical method of hierarchical design of programs).

Among the semantic equivalences of the literature, history preserving bisimulation is particularly interesting since it is a congruence w.r.t. the refinement of systems without invisible actions. We use a characterisation of this equivalence by means of a back and forth bisimulation to exhibit adequate modal logics, and we also give a translating algorithm between those logics.

We enlarge the field of our work by considering partial order based models with invisible actions. We show that the above-mentionned back and forth bisimulation, adapted to this framework, coincides with branching bisimulation over causal trees, but also with two new equivalences : the branching mixed-ordering equivalence and the history preserving branching bisimulation, we study in details.

Remerciements

Premier essai

Vraiment merci.

Deuxième essai

Je tiens à remercier les membres du jury:

- J. Sifakis qui a accepté de présider ce juryΓ
- M. Hennessy qui a bien voulu s'intéresser à ce travail et a accepté de participer au juryΓ
- P. Degano et F. Vaandrager qui ont accepté d'être rapporteurs. Cette tâche habituellement ingrate était encore alourdie par l'état inachevé du manuscrit.

Je veux aussi remercier Ph. Jorrand qui a encadré cette thèseΓ et surtout Ph. Schnoebelen qui m'a forméeΓassistée et supportée pendant toutes ces années de recherche. Sa disponibilité et son esprit critique (il n'y a qu'à lui demander ce qu'il pense de ces remerciements !) m'ont permis de réaliser ce travail. J'ai puΓgrâce à ses compétences scientifiques et ses qualités humainesΓprendre un immense plaisir à effectuer ma recherche.

Je tiens aussi à remercier J. Echague quiΓpar ses connaissances et son enthousiasme permanentΓm'a soutenue et aidée jusqu'à la fin.

Je remercie tous les membres de l'équipe "parallélisme" pour leur gentillesse et l'intérêt qu'ils m'ont porté.

EnfinΓje remercie RenéeΓJoëlΓKarimΓSami et tous ceux qui ont eu la patience de me soutenir (et de me supporter) dans la réalisation de ce travail.

Sommaire

Introduction

La vérification des programmes est une préoccupation majeure, surtout quand les programmes se substituent à l'homme pour des tâches délicates, e.g. le contrôle d'appareils d'assistance médicale, de régulation thermique de centrales nucléaires, etc. Mais, de nos jours, la complexité des programmes croît avec celle des tâches qui leur sont confiées. Pour aider à la conception et à la compréhension de systèmes complexes on utilise les méthodes de conception modulaire et/ou hiérarchique.

La conception hiérarchique consiste à concevoir le programme par analyse descendante. A chaque étape de cette analyse, un programme abstrait est remplacé par un programme plus complexe à un niveau d'abstraction plus concret. Cette méthode est connue sous le nom de *raffinement de programmes*.

La conception modulaire consiste à construire le programme en assemblant des sous-programmes et en décrivant la manière dont ils interagissent, par exemple en fonctionnant en parallèle, en communiquant par message, etc.

Lorsque l'on considère un programme, pris isolément, son exécution est influencée par l'environnement, e.g. l'exécution des autres sous-programmes, et elle influence elle-même cet environnement. Pnueli a introduit le concept de *Système Réactif* [Pnu77] pour qualifier les programmes dont le rôle est de maintenir une interaction avec leur environnement. Le problème de connaître la réaction de ces programmes vis à vis de leur environnement et la manière dont ils influencent cet environnement est très complexe. Cette complexité augmente encore lorsque le programme englobant est lui-même un sous-programme d'un programme plus large.

Pour prouver des propriétés sur le comportement d'un système réactif, il faut être capable de formaliser ce comportement. Habituellement, on utilise un domaine mathématique approprié d'*objets concrets* (ou encore de *modèles opérationnels*, où pour chaque objet du domaine, on dérive une définition formelle de la notion de

comportement. Chaque définition induit alors une identification sur les objets selon le critère "avoir le même comportement". Cette identification s'appelle une *équivalence sémantique* et on définit la *sémantique* d'un programme c'est à dire son comportement comme une classe d'équivalence.

La définition du comportement d'un système doit refléter les méthodes de construction modulaire et/ou hiérarchique qui sont utilisées pour concevoir les programmes c'est à dire que le comportement du système doit pouvoir être déduit des comportements de ses sous-systèmes. On veut donc définir une sémantique qui d'une part se comporte bien vis à vis des combinateurs de programmes (on dit qu'elle doit être *compositionnelle*) et qui d'autre part soit compatible avec le raffinement de programmes c'est à dire qu'elle autorise à changer le niveau de détail de description des programmes (on dit alors que cette sémantique doit être *compatible avec le raffinement de programmes*).

Donner une sémantique satisfaisante pour les systèmes réactifs n'est pas simple. Les premiers travaux sur ce sujet ont repris le cadre de la sémantique dénotationnelle qui a fait ses preuves pour les systèmes *transformationnels*[1]. Malheureusement ce type de sémantique se prête mal aux systèmes réactifs : la simple introduction du *non-déterminisme* dans un langage comme Pascal nécessite la construction d'outils mathématiques très compliqués et très lourds à manipuler (les "power-domains"); on peut en voir un exemple dans [Mos90]. L'approche dénotationnelle est toutefois encore étudiée (voir par exemple [BR92]).

Une approche adoptée aujourd'hui consiste à définir le comportement d'un programme en termes des actions observables de l'extérieur qu'il peut effectuer (une *action* représente n'importe quelle activité par exemple un acte de communication).

Dans la littérature les sémantiques pour les systèmes réactifs peuvent être ordonnées selon deux dimensions. La première dimension distingue les sémantiques d'*entrelacement* des sémantiques *basées sur la causalité* la deuxième dimension sépare les sémantiques *du temps linéaire* des sémantiques *du temps arborescent*. Prenons un exemple : définissons le comportement d'un système comme l'ensemble des suites d'actions qu'il peut effectuer (cette sémantique correspond à la *sémantique de traces* de Hoare [Hoa85]). La sémantique de traces ne fait aucune différence entre le calcul d'actions en parallèle et celui d'un entrelacement arbitraire de ces mêmes actions. C'est une sémantique d'*entrelacement* (d'autres exemples de sémantiques d'entrelacement sont la sémantique de *refus* de [BHR84] la sémantique de *readiness* de [OH86]). En opposition on pourrait explicitement représenter le parallélisme en décrivant le comportement du système au moyen d'une relation de causalité entre les actions effectuées ce qui correspond aux sémantiques *basées sur*

[1] Ces systèmes transforment l'état initial du programme en un état final.

la causalité [Pra86].

La sémantique de traces ne prend pas en compte le moment où le système effectue ses choix non-déterministes. On dit d'une telle sémantique qu'elle est *du temps linéaire*. Les sémantiques qui tiennent compte des calculs du système, mais aussi de ses alternatives de choix (ou d'une information partielle sur ces alternatives) au cours de ces calculs sont appelées les sémantiques *du temps arborescent*, comme par exemple la *sémantique de refus* de [BHR84] pour le langage CSP et la *bisimulation* de [Mil80] et [Par81] pour le langage CCS. [Pnu85] a mis en évidence la différence entre les sémantiques *du temps linéaire* et les sémantiques *du temps arborescent*.

L'existence de tous ces types de sémantiques montre que l'on ne cherche pas à définir *le* comportement d'un système. On choisit une sémantique plutôt qu'une autre en fonction des propriétés comportementales auxquelles on s'intéresse. Par exemple, si la propriété pertinente est "le système ne se bloquera jamais", il est inutile de tenir compte du moment où sont faits les choix internes du système.

Il existe aussi d'autres critères pour choisir une sémantique. Ces critères sont basés sur les propriétés théoriques attendue pour cette sémantique. Par exemple, l'équivalence observationnelle de Milner, dans sa version faible, c-à-d. avec prise en compte de pas silencieux du système, n'induit pas une "bonne" équivalence parce qu'elle n'est pas une congruence pour le combinateur de choix non-déterministe. La congruence observationnelle pose également des problèmes car elle ne peut être présentée sous la forme d'un système de réécriture canonique, c-à-d. à terminaison finie et confluent, etc.

La nécessité de choisir telle ou telle sémantique provient du fait que l'on ne sait pas définir *le* comportement d'un système réactif, auquel cas cette définition *du* comportement s'appliquerait toujours.

Parallèlement aux travaux visant à définir la sémantique des systèmes réactifs, se sont développés des langages formels de description de propriétés comportementales des programmes. Dans un programme séquentiel ordinaire, comme le tri d'une liste d'entiers, la correction s'exprime en termes d'une paire *Précondition/Postcondition* dans un formalisme tel que la *logique de Hoare*. On peut trouver dans [Apt85] une utilisation de la logique de Hoare dans un cadre de systèmes réactifs. Mais la logique est très lourde à manipuler dans le cadre des systèmes réactifs et ne permet pas de s'exprimer au bon niveau d'abstraction.

Comme l'a souligné Pnueli dans [Pnu77], les *logiques temporelles*, et plus généralement les *logiques modales*, sont un formalisme approprié pour exprimer les propriétés des systèmes réactifs : elles permettent d'inclure le Temps dans les raisonnements formels et d'exprimer des propriétés sur le comportement du système au cours du temps, comme les propriétés de sûreté, de vivacité, etc. (voir [MP92]). La Logique Temporelle a donné naissance à un domaine de recherche très actif pour de nombreux aspects de la conception des systèmes réactifs, tels que la spécification, la vérification ou la synthèse de programmes. Nous renvoyons à [Eme90] et [MP92]

pour l'étude théorique des logiques temporelles et leur utilisation pour des systèmes réactifs.

Les deux approches logique et sémantique comportementale ne sont pas opposées : lorsque l'on adopte une logique pour décrire les propriétés comportementales d'un programme il faut être certain qu'elle est cohérente avec la définition du comportement c'est à dire l'équivalence sémantique que l'on a choisie. Autrement dit la logique ne doit en aucun cas distinguer deux objets sémantiquement équivalents. On parle de *compatibilité* de la logique avec la sémantique comportementale.

Si la logique est suffisamment expressive pour distinguer des programmes qui ne sont pas sémantiquement équivalents on dit que la logique est *complètement expressive* pour la sémantique. Si les deux conditions compatibilité et expressivité complète sont réunies pour une logique on dit qu'elle *caractérise* la sémantique ou qu'elle est *adéquate* pour l'équivalence sémantique.

Il existe de nombreux résultats d'adéquation. Le plus célèbre est dû à [HM80] qui proposent une caractérisation de la bisimulation forte par la logique modale connue sous le nom de *logique de Hennessy-Milner* ou encore HML.

Toutes les approches pour définir une sémantique des systèmes réactifs ont donné naissance à une multitude d'équivalences parfois définies de façon tellement différente qu'il est difficile de savoir quelle sémantique est appropriée à quelle application. Le domaine de recherche appelé "*Comparative Concurrency Semantics*" qui s'est considérablement développé ces dernières années vise à classifier ces équivalences. Le principe est d'ordonner les équivalences sémantiques selon la relation "fait au moins autant d'identification que". On peut par exemple trouver dans [Gla90a] une classification des équivalences sémantiques d'entrelacement.

Dans cette thèse nous nous intéressons à une famille d'équivalences sémantiques du temps arborescent basées sur la *bisimulation*. Informellement l'existence d'une bisimulation entre deux systèmes signifie que les deux systèmes font les mêmes choix au même moment.

La première équivalence de bisimulation appelée *bisimulation forte* a été proposée par [Mil80] pour son langage CCS mais c'est la définition de [Par81] qu'on utilise habituellement. Aujourd'hui le nom "bisimulation" recouvre plus une famille d'équivalences qu'une équivalence particulière. Elles peuvent toutes être ramenées à une bisimulation forte sur des modèles dérivés.

Cette thèse est composée de deux parties. La première se situe dans le cadre des sémantiques d'entrelacement et la deuxième dans celui des sémantiques d'ordre partiel plus récemment étudiées que celles d'entrelacement. Les deux parties contiennent chacune une introduction qui en précise le contenu.

Partie I

Les modèles d'entrelacement du temps arborescent

Introduction

Les modèles d'entrelacement (traduit du terme anglais "*interleaving*") sont bien adaptés pour décrire les systèmes *séquentiels non-déterministes*.

ConceptuellementΓun système est dans un état donnéΓappelons le q. Chaque fois qu'il effectue une action aΓil évolue et "passe" dans un autre état q'. Nous notons $q \xrightarrow{a} q'$ ce changement d'étatΓaussi appelé *transition d'état*.

Un système est *séquentiel* s'il peut effectuer au plus une action à la fois.

Les *systèmes de transitions* sont reconnus comme étant des bons modèles pour représenter les systèmes séquentiels non-déterministes [Kel76]. L'utilisation de ces modèles mathématiques pour l'étude des systèmes réactifs s'est répandue depuis les travaux de Plotkin [Plo81] dans lesquels il propose une méthode simple pour définir la sémantique opérationnelle des langages de programmation. Cette méthode connue sous le nom *SOS* (pour Structural Operational Semantics) consiste à définir des règles structurelles sur la forme des programmes. On dérive du langage de programmation et des règles SOS associéesΓun système de transitions dont les états sont les termes d'un langage formel qui correspond au (ou étend le) langage de programmation auquel on veut donner une sémantique et les transitions entre les états sont déterminées par les règles SOS.

De nos joursΓla méthode de Plotkin est très largement utiliséeΓet on compte de nombreux langages munis d'une sémantique SOSΓe.g. pour CCS voir [Mil80]Γd'où l'intérêt général pour étudier les sémantiques basées sur les règles "à la Plotkin".

De façon tout à fait surprenanteΓpeu de travaux ont été développés pour étudier une théorie "du style de règles à la Plotkin" ([Sim85]Γ[BIM88] et [GV88]Γ[GV92]). A ce sujetΓ[Vaa89b] fait remarquer : *"Maybe one reason why semantists have almost paid no attention to a general theory of Plotkin style rules is the fact that they are*

so simple to use."

Mais les travaux de [GV88] (voir aussi [GV92]) démontrent que ce format de règles a des propriétés très intéressantes que nous mentionnons plus loin dans cette première partie.

Les équivalences sémantiques sur les modèles d'entrelacement ont déjà été beaucoup étudiées nous renvoyons à [Gla90a] où sont répertoriées les équivalences sémantiques différentes pour les systèmes séquentiels non-déterministes *concrets* à *branchement fini*. Un système est dit *concret* si tous ses calculs sont observables de l'extérieur. Un système est à *branchement fini* (ou encore *à non-déterminisme fini*) si à tout instant il n'a qu'un nombre fini d'alternatives pour effectuer une action donnée. La plupart des langages de programmation ne donnent lieu qu'à ce type de systèmes.

Dans cette première partie nous nous intéressons à la plus fine des équivalences sémantiques d'entrelacement (celle qui fait le moins d'identification) : la *bisimulation*. Cette équivalence est désormais reconnue comme la sémantique d'entrelacement de base du temps arborescent : elle se contente d'identifier les systèmes de transitions qui ont la même structure arborescente.

La bisimulation a été introduite par Milner pour son langage CCS [Mil80] sous le nom d'*équivalence observationnelle* mais c'est à Park [Par81] que l'on doit son nom ainsi que sa définition uniformément utilisée.

La bisimulation a de nombreuses propriétés. Nous en citons quelques unes.

Premièrement elle est compositionnelle. Ainsi lorsque les programmes sont construits de façon modulaire (e.g. par composition parallèle de sous-programmes) l'utilisation de la bisimulation permet de substituer à tout sous-programme un programme équivalent sans changer les propriétés comportementales du programme englobant. Plus généralement les travaux de [GV88] (voir aussi [GV92]) ont mis en évidence une classe très générale de combinateurs de programmes appelés *combinateurs tyft* contenant ceux du langage CCS pour lesquels la bisimulation est une congruence (Nous rappelons ce résultat au Chapitre 2).

Deuxièmement la bisimulation possède des caractérisations logiques. On connait les très classiques travaux de [HM85] qui ont montré la compatibilité et l'expressivité complète de la logique modale de Hennessy-Milner pour la bisimulation dans le cas où les systèmes sont à branchement fini (Nous le rappelons au Chapitre 3) Depuis ce résultat d'autres résultats d'adéquation entre une logique et la bisimulation ont été établis (voir p. ex. [Pnu85] [BCG87] [DV90]).

Enfin en utilisant la bisimulation comme équivalence sémantique on peut déterminer un représentant canonique minimal (en son nombre d'états) de chacune des classes d'équivalence modulo bisimulation ce qui d'un point de vue pratique permet certaines fois de simplifier les programmes.

La plupart des travaux en rapport avec la bisimulation font l'hypothèse de systèmes à branchement fini. A notre connaissance seuls [BR83] [BT85] [Her87] et [SP90] se sont attaqués au cas du branchement infini : [BR83] montre que l'adéquation des logiques HML [HM80] PDL [FL79] et RTL (de Rounds et Gurevich)[2] pour la bisimulation s'affaiblit en compatibilité et que la bisimulation est strictement plus fine que l'équivalence observationnelle de [Mil80]. [BT85] renforce le résultat pour la logique RTL en montrant qu'il en est de même pour la classe des logiques $\mathcal{C}-TL$. Les résultats de [BR83] et [BT85] sont basés sur des contre-exemples ad hoc et relativement compliqués.

[Her87] a étudié le μ-calcul propositionnel de [Koz83] ainsi que la logique PDL de [FL79] en utilisant un modèle simple qui fait partie de la famille des *Processus Ordinaux* de Klop [Klo88] (Il a comparé la bisimulation avec plusieurs variantes de l'équivalence observationnelle).

Ces modèles sont particulièrement intéressants car les classes d'équivalence modulo bisimulation sont des singletons.

[SP90] reprennent l'idée de Klop et utilisent les Processus Ordinaux pour montrer que la logique CTL de [CE81] n'est plus complètement expressive pour la bisimulation.

Dans cette partie nous utilisons le cadre unifié des *Processus Ordinaux* pour montrer que toutes les logiques citées plus haut ainsi que la logique CTL^* de [EH86] sont seulement compatibles avec la bisimulation. Bien entendu notre résultat ne portent que sur des logiques finitaires c-à-d. n'autorisant que des combinaisons booléennes finies. Dans le cas général des logiques infinitaires comme par exemple HML infinitaire considérée dans [Mil89] il n'est évidemment pas nécessaire de faire la moindre hypothèse sur le degré de branchement des systèmes pour garantir l'adéquation.

Les résultats de notre étude montrent que sauf à se restreindre au cas branchement fini il n'existe pas de résultat d'adéquation entre d'une part une sémantique basée sur la bisimulation et d'autre part des logiques (finitaires) comme HML CTL CTL^* le μ-calcul RTL etc... Il est alors naturel d'étudier les propriétés des équivalences induites par ces logiques pour déterminer si elles ne pourraient pas elles-mêmes être utilisées comme sémantique. Mais alors il faut vérifier que ces équivalences sont compositionnelles c-à-d. si comme la bisimulation elles sont des congruences pour les combinateurs de programmes.

Nous savons déjà partiellement répondre à la question pour la logique CTL : [Sch90a] a montré que l'équivalence qu'elle induit n'est pas une congruence pour la composition parallèle. Nous montrons qu'il en est de même pour la logique CTL^* et le μ-calcul.

Dans notre travail au lieu de considérer un ensemble particulier de combinateurs

[2] Nous ne connaissons pas de référence exacte pour la définition originelle de cette logique.

comme on le fait habituellementΓnous étudions les congruences engendrées par les logiques CTLΓCTL^* et le μ-calcul vis à vis de tous les combinateurs *tyft* de [GV88].

Bien que l'ajout de contextes *tyft* augmente de façon significative le pouvoir de distinction des équivalences considéréesΓnous établissons qu'aucune des congruences étudiées n'est aussi fine que la bisimulation.

FinalementΓnous isolonsΓau moyen d'un critère syntaxiqueΓune sous-classe des contextes *tyft*Γappelés contextes *sans copie*Γqui contient ceux de CCS. Intuitivementun contexte sans copie n'autorise pas la duplication du programme que l'on "teste" dans ce contexte. Nous montrons que la restriction aux contextes sans copie est réelleΓau sens où la congruences de traces pour ces contextes identifie plus de modèles que la congruence de traces pour les contextes *tyft*.

Chapitre 1

Rappels sur les Systèmes de Transitions

Nous introduisons les notions de *graphes d'états (étiquetés)* et de *systèmes de transitions (étiquetés)* [Kel76]. Un graphe étiqueté représente un système réactif (ou une machine) séquentiel non-déterministe dont les changements d'états sont accompagnés d'actions (e.g. communiquer avec un autre système ou récupérer de la mémoire vive, afficher un message à l'écran, recevoir un message, etc.

Dans cette thèse, nous appelons *système de transitions (étiqueté)* un graphe d'états enraciné (c-à-d. un graphe d'états dans lequel on privilégie un état, interprété comme l'état courant du système.

Dans certains contextes, certaines actions peuvent être considérées comme inobservables de l'extérieur. On parle aussi d'*action silencieuse*. C'est le cas, par exemple, de la récupération de mémoire dans un ordinateur lorsque le système est observé par le programmeur.

1.1 Graphes d'états, Systèmes de transitions (étiquetés)

Définition 1.1.1 (Graphe d'états (étiqueté)) *Soit un ensemble $A = \{a, \ldots\}$ de noms d'actions.*

Un graphe d'états (étiqueté sur A) *est une structure $G = (Q_G, A, \rightarrow_G)$, où*

- Q_G *est un ensemble* $\{q, q_1, \ldots\}$ *d'états,*
- $\rightarrow_G \subseteq Q_G \times A \times Q_G$, *est la relation de transition entre les états de* Q_G.

Pour faciliter la lecture on écrira souvent "graphe" au lieu de "graphe d'états".

Dans la suite l'ensemble d'actions A est le même pour tous les graphes que nous considérons. Nous notons $\mathbf{G}$ la classe des graphes d'états étiquetés sur A et nous écrivons plus simplement $G = (Q, \rightarrow)$ au lieu de $G = (Q, A, \rightarrow)$.

Pour $G = (Q, \rightarrow) \in \mathbf{G}$ un élément $(q, a, q') \in \rightarrow$ dénote un changement d'état de la machine représenté par G et donnant lieu à "l'événement" (ou encore action) a. Nous appelons ce changement d'état une *a-transition* que nous notons $q \stackrel{a}{\rightarrow} q'$. Nous notons $q \rightarrow q'$ lorsque $q \stackrel{a}{\rightarrow} q'$ pour une certaine action $a \in A$. Un état q n'ayant pas de successeur est appelé *état terminal.* Cette terminologie exprime que le système réactif dans cet état ne peut plus évoluer.

La relation $\rightarrow$ se prolonge naturellement en la relation $\rightarrow \subseteq Q \times A^* \times Q$: si $w \in A^*$ est le mot $a_1 a_2 \ldots a_n$ nous notons $q \stackrel{w}{\rightarrow} q'$ lorsqu'il existe une suite de transitions $q \stackrel{a_1}{\rightarrow} q_1 \stackrel{a_2}{\rightarrow} q_2 \ldots \stackrel{a_n}{\rightarrow} q_n = q'$. Par conséquent si ϵ désigne le mot vide de A^* $q \stackrel{\epsilon}{\rightarrow} q$ pour tout état q.

Nous présentons ici une opération de combinaison de graphes appelée *somme* qui nous servira dans la suite pour simplifier techniquement certaines définitions.

Définition 1.1.2 (Somme) *Pour tous* $G_i = (Q_i, \rightarrow_i) \in \mathbf{G}$, $i = 1, 2$, *tels que* $Q_1 \cap Q_2 = \emptyset$, *on définit*

$$G_1 \oplus G_2 \stackrel{def}{=} (Q_1 \cup Q_2, \rightarrow_1 \cup \rightarrow_2)$$

On distingue parfois dans l'ensemble des états d'un graphe G un état particulier qui correspond à l'état courant du système que représente G.

Définition 1.1.3 *Un* système de transitions (étiqueté sur A) *est une structure* $S = (G_S, r_S)$, *où*

- G_S *est un graphe d'états, appelé* graphe d'états de S,
- $r_S \in Q_{G_S}$, *est la* racine *ou encore l'*état initial *de* S.

Nous noterons en abrégé ST *pour "système de transitions (étiqueté sur A)".*

Dans la suite $\mathbf{ST}$ désigne la classe des ST.

Notations 1 *Soit* $S = ((Q_S, \rightarrow_S), r_S)$ *un ST de graphe d'états* $(Q_S, \rightarrow_S)$ *et de racine* r_S. *Nous le notons plus simplement* $(Q_S, \rightarrow_S, r_S)$ *ou encore* (G_S, r_S).

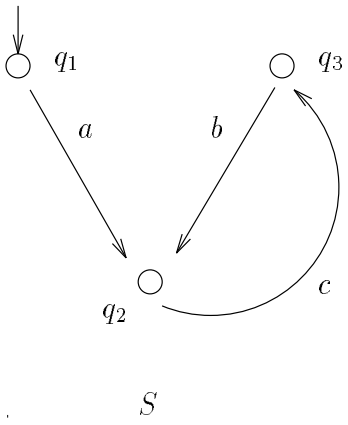


Figure 1.1: Un exemple de système de transitions

Exemple 1 *La Figure 1.1 représente graphiquement un ST S, sur un alphabet d'actions $A_S = \{a, b, c\}$. Son graphe d'états G_S a trois états. Formellement, $Q_S = \{q_1, q_2, q_3\}$ et $\rightarrow_S = \{(q_1, a, q_2), (q_2, b, q_3), (q_3, c, q_2)\}$.*

L'état initial de S est l'état q_1, ce que nous représentons graphiquement par une flèche incidente vers cet état.

Dans certaines des figures nous omettons de nommer certains états s'ils ne sont pas pertinents pour nos exemple.

1.2 Chemins et traces

Nous introduisons à présent la notion d'*exécution* dans un ST. Une *exécution* correspond à un calcul du système. Puisque les systèmes sont non-déterministesΓ plusieurs exécutions sont possibles. De plusΓpuisque l'activité du système n'a pas forcément pour vocation de se terminer un jourΓles exécutions peuvent être infinies.

Soit $G \in \mathbf{G}$Γet $q \in Q_G$.

Définition 1.2.1 (Chemins et exécutions) *Un* chemin *(ou* calcul*) dans G partant de q est une séquence de transitions de la forme $q \xrightarrow{a_1}_G q_1 \xrightarrow{a_2}_G \ldots$.*

Une exécution *partant de q est un chemin maximal partant de q. Il peut donc être fini (et s'achever dans un état terminal), ou infini.*

Nous désignons par $\Pi_G(q)$ (ou plus simplement $\Pi(q)$) l'ensemble de tous les chemins dans G qui partent de l'état q, et $Ex_G(q)$ (ou plus simplement $Ex(q)$) l'ensemble des exécutions qui partent de q. $Ex(G)$ désigne l'ensemble $\bigcup_{q \in Q_G} Ex_G(q)$.

Nous utilisons les symboles $\sigma, \sigma_1, \ldots$ pour désigner des chemins dans un graphes, et $\pi, \pi_1, \ldots$ pour désigner les exécutions.

Soit $S \in \mathbf{ST}$, $q \in Q_S$. Une exécution de S est une exécution π dans G_S partant de l'état initial de S, i.e. $\pi \in Ex_{G_S}(r_S)$. On note $Ex(S)$ l'ensemble des exécutions de S.

Etant donné $\sigma = q_0 \stackrel{a_1}{\rightarrow} q_1 \stackrel{a_2}{\rightarrow} \ldots$ un chemin dans SΓ$|\sigma|$ dénote sa longueurΓi.e. le nombre de transitions de σ; on pose $|\sigma| = \omega$ si σ est infini. Pour $i \leq |\sigma|$Γ$\sigma(i)$ dénote l'état q_i et σ^i le chemin $q_i \stackrel{a_{i+1}}{\rightarrow} q_{i+1} \stackrel{a_{i+2}}{\rightarrow} \ldots$ (c-à-d. le i-ème suffixe de σ).

Pour chaque calcul séquentiel effectué par le systèmeΓon extrait une observation externe constituée de la suite des actions qui ont eu lieu. Cette suite (éventuellement infinie) est appelée une *trace* et se définit formellement par :

Définition 1.2.2 (Trace) *Soit $G \in \mathbf{G}$. Etant donnée une exécution $\pi = q_0 \stackrel{a_1}{\rightarrow} q_1 \stackrel{a_2}{\rightarrow} \ldots$ dans G, nous définissons la* trace *de π, notée $tr(\pi)$, par*

$$tr(\pi) \stackrel{def}{=} a_1 a_2 \ldots$$

Lorsqu'il n'y a pas d'ambiguïtéΓnous notons $Tr(q)$ au lieu de $Tr_G(q)$.

Cette notion de trace est celle de [Hoa85] et de [Gla90b]Γmais elle diffère des *traces complètes* de [Gla90b] pour lesquelles on ne considère que les calculs achevésΓ i.e. $\pi \in Ex(S)$.

1.3 Notion de branchement infini

En pratiqueΓcomme par exemple dans le langage CCS avec récursivité gardéeΓde nombreux programmes ne donnent lieu qu'à un type restreint de non-déterminisme : le *branchement fini*Γqui a des propriétés mathématiques très intéressantes comme nous le verrons dans les Sections 2.5 et 3.4.

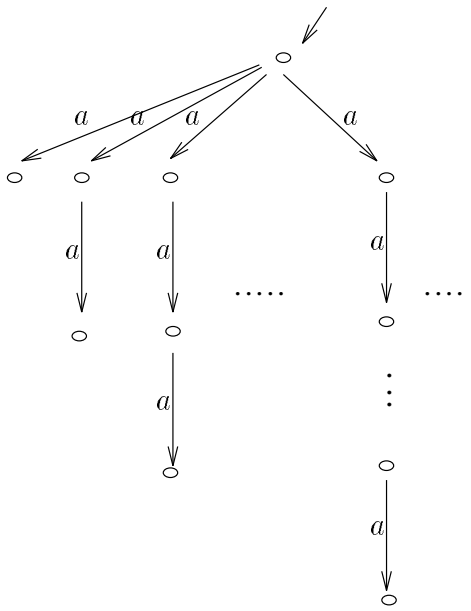
Définition 1.3.1 (Graphe d'états et Système de transitions à branchement fini) *$G \in \mathbf{G}$ est* à branchement fini[1], *noté* b.f. *en abrégé, si pour tous $q \in Q_G$, l'ensemble $\{q' \in Q_G \mid q \stackrel{a}{\rightarrow} q'\}$ est fini.*
$S \in \mathbf{ST}$ est à branchement fini si G_S est à branchement fini.

IntuitivementΓpour de tels systèmesΓle nombre de choix possibles pour effectuer une action donnée dans chaque état est fini.

Si S n'est pas à b.f. nous disons alors qu'il est *à branchement infini* (noté *b.i.*). C'est le cas par exemple du ST de la Figure 1.2.

Dans la suiteΓnous notons $\mathbf{G}_{bf}$ (resp. $\mathbf{ST}_{bf}$) la sous classe de $\mathbf{G}$ (resp. $\mathbf{ST}$) formée des graphes (resp. ST) à b.f.

[1] *La terminologie "à image finie" est utilisée dans [HM85] [Mil81].*



S_1

Figure 1.2: Un exemple de ST à branchement infini.

Chapitre 2

Equivalences comportementales

Nous rappelons les principales équivalences sémantiques du temps arborescent pour les ST. Parmi ces équivalences, l'équivalence de *bisimulation forte*, introduite par Milner, est reconnue comme l'équivalence sémantique de base : elle se contente d'identifier des systèmes qui ont la même structure arborescente.

Après avoir présenté la bisimulation forte et certaines de ses propriétés, nous définissons la classe de combinateurs de programmes *tyft* de [GV88] et nous rappelons la compositionalité de la bisimulation vis à vis de ces combinateurs.

Nous rappelons également que la bisimulation et l'équivalence observationnelle de Milner (ainsi qu'une variante) coïncident sur la classe des graphes à b.f..

Enfin, nous rappelons brièvement les variantes de l'équivalences de bisimulation proposées dans la littérature pour le cas où des actions silencieuses sont considérées.

La première section introduit l'*équivalence de traces* de [Hoa85] qui nous servira à plusieurs reprises dans la suite.

2.1 Notion d'équivalence de traces

Prenons un système S qui évolue par changement d'état, et que l'on peut interrompre à tout instant. L'observation d'un tel système consiste simplement en une séquence d'actions qu'il peut effectuer. Cette séquence peut être infinie si l'exécution n'est jamais interrompue. L'*équivalence de trace* identifie les systèmes qui admettent

le même ensemble d'observations.

Soit $G \in \mathbf{G}$. Pour tout $q \in Q_G$ nous définissons l'ensemble des traces dans G partant de q par

$$Tr_G(q) \stackrel{\text{def}}{=} \{tr(\sigma) \mid \sigma \in \Pi(q)\}$$

Pour tout $S \in \mathbf{ST}$ on définit l'ensemble des traces de S par

$$Tr(S) \stackrel{\text{def}}{=} Tr_{G_S}(r_S)$$

Définition 2.1.1 (équivalence de traces) *Deux ST, S_1, S_2 sont* équivalents de trace, *noté $S_1 \sim_{Tr} S_2$ si ils ont les mêmes traces, i.e. $Tr(S_1) = Tr(S_2)$.*

$\sim_{Tr}$ induit clairement une équivalence sur $\mathbf{ST}$.

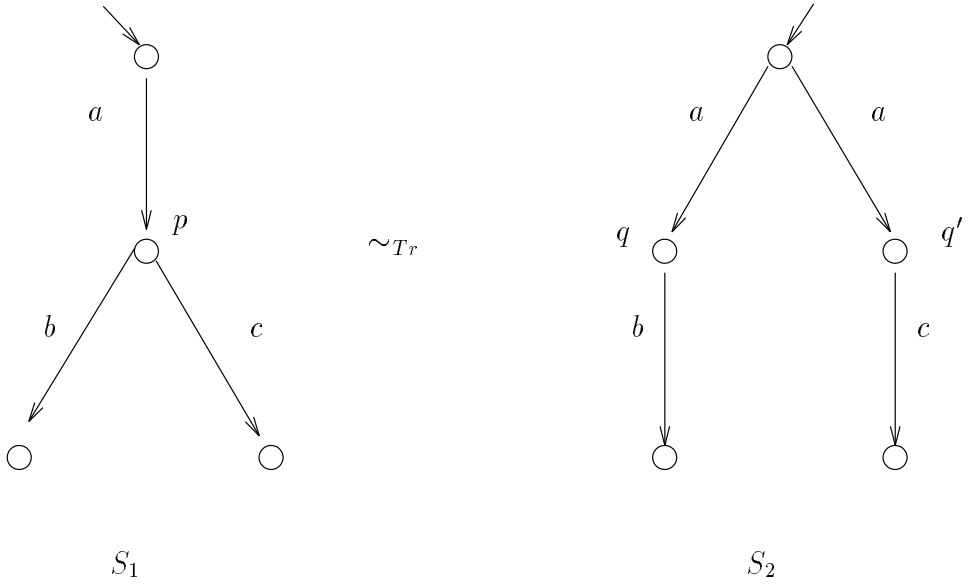


Figure 2.1: Le moment des choix.

Par exemple, les ST de la Figure 2.1 admettent le même ensemble de traces $\{\epsilon, a, ab, ac\}$, où ϵ dénote le mot vide. Cependant, il est clair que ces deux systèmes n'ont pas la même arborescence de choix : les deux systèmes peuvent choisir entre effectuer ab ou ac, mais ce choix ne se fait pas au même moment. S_1 choisit de faire ab ou ac après avoir effectué a, alors que S_2 choisit avant.

Les équivalences sémantiques du temps arborescent ont été introduites pour prendre aussi en compte la similitude des systèmes en termes des choix possibles dans chacun de leurs états. Autrement dit, ces équivalences respectent l'*arborescence*

de choix des systèmesΓaussi connue sous le nom de *structure de branchement* ou plus simplement *branchement*[1]. Elles sont en général plus fine que l'équivalence de trace.

2.2 Equivalence de bisimulation (forte)

La *bisimulation forte* a été introduite par Milner pour son langage CCS [Mil80]Γmais c'est la définition proposée par Park [Par81] que l'on considère habituellement.

Une bisimulation est une relation binaire entre les états de graphes d'états.

Définition 2.2.1 (Bisimulation (forte) sur les graphes)
Etant donnés $G_1, G_2 \in \boldsymbol{G}$, *une relation* $R \subseteq Q_{G_1} \times Q_{G_2}$ *est une* bisimulation *entre* G_1 *et* G_2, *noté* $R : G_1 \underline{\leftrightarrow} G_2$, *ssi pour tous* $q_1 R q_2$,

1. *pour toute transition* $q_1 \stackrel{a}{\rightarrow}_{G_1} q_1'$, *il existe un état* q_2' *de* G_2 *tel que* $q_2 \stackrel{a}{\rightarrow}_{G_2} q_2'$ *et* $q_1' R q_2'$, *et*
2. *réciproquement, pour toute transition* $q_2 \stackrel{a}{\rightarrow}_{G_2} q_2'$, *il existe un état* q_1' *de* G_1 *tel que* $q_1 \stackrel{a}{\rightarrow}_{G_1} q_1'$ *et* $q_1' R q_2'$.

Nous disons d'une relation R ayant les propriétés 1 et 2 de la Définitions 2.2.1 qu'elle a la *propriété de transfert* de la bisimulation forte.

Si $R : G \underline{\leftrightarrow} G$Γon dit que R est une *auto-bisimulation* de G.

Définition 2.2.2 (Bisimulation (forte) sur les ST) *Soient* $S_1, S_2 \in \boldsymbol{ST}$. *Une relation* R *est une bisimulation entre* S_1 *et* S_2, *noté* $R : S_1 \underline{\leftrightarrow} S_2$, *si* R *est une bisimulation entre* G_{S_1} *et* G_{S_2} *qui relie les états initiaux, i.e.*

- $R : G_{S_1} \underline{\leftrightarrow} G_{S_2}$, *et*
- $r_{S_1} R r_{S_2}$

On note $S_1 \underline{\leftrightarrow} S_2$ *si il existe une relation* R *telle que* $R : S_1 \underline{\leftrightarrow} S_2$, *et on dit que* S_1 *et* S_2 *sont* bisimilaires[2].

Exemple 2 *La relation* R *de la Figure 2.2 dessinée en lignes pointillées est une bisimulation entre* S_1 *et* S_2. *Par contre, les ST de la Figure 2.1 ne sont pas bisimilaires : l'état* p *de* S_1 *n'a pas d'équivalent possible dans* S_2 *puisque partant* p *on peut choisir entre effectuer l'action* b *ou l'action* c, *ce qui n'est pas vrai des états* q *et* q' *dans* S_2.

[1] Cette terminologie sera adoptée dans la suite

[2] *Cette terminologie est maladroite, on devrait plutôt dire* bisimulants, *mais c'est celle qui s'est imposer dans la pratique.*

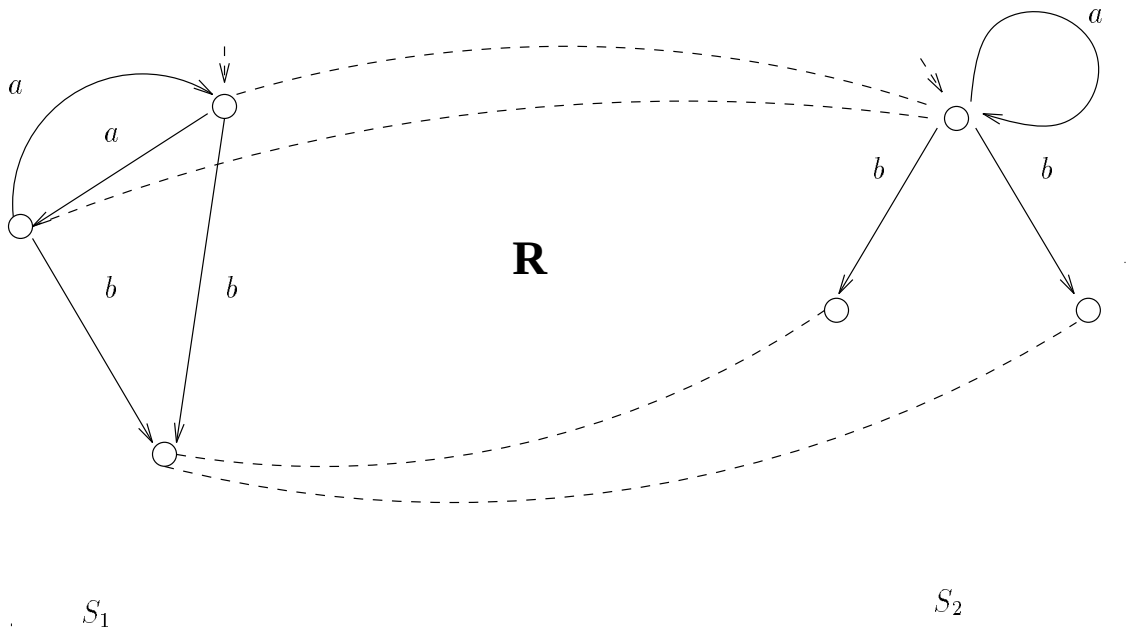


Figure 2.2: Un exemple de bisimulation forte.

Proposition 2.2.1 *Pour tous* $G, G_1, G_2, G_3 \in \mathbf{G}$ *:*

(1) $Id_{Q_G} : G \underline{\leftrightarrow} G$,

(2) $R : G_1 \underline{\leftrightarrow} G_2$ *implique* $R^{-1} : G_2 \underline{\leftrightarrow} G_1$,

(3) $R : G_1 \underline{\leftrightarrow} G_2$ *et* $R' : G_2 \underline{\leftrightarrow} G_3$ *impliquent* $(R' \circ R) : G_1 \underline{\leftrightarrow} G_3$,

(4) $R : G_1 \underline{\leftrightarrow} G_2$ *et* $R' : G_1 \underline{\leftrightarrow} G_2$ *impliquent* $(R \cup R') : G_1 \underline{\leftrightarrow} G_2$.

Ces propriétés sont des conséquences immédiates de la Définition 2.2.1. Il est facile de voir que les propriétés de (1) à (4) peuvent tout aussi bien être exprimées sur les ST. Ce qui nous montre que $\underline{\leftrightarrow}$ est une relation d'équivalence sur **ST**.

On peut clairement généraliser (4) à des unions arbitraires; autrement dit, pour I un ensemble quelconque,

$$\forall i \in I, R_i : G_1 \underline{\leftrightarrow} G_2 \text{ implique } \bigcup_{i \in I} R_i : G_1 \underline{\leftrightarrow} G_2$$

Comme conséquence, il existe une plus grande bisimulation entre G_1 et G_2. En particulier, on peut définir $\underline{\leftrightarrow}_G$, la plus grande auto-bisimulation de G par:

$$\underline{\leftrightarrow}_G \stackrel{\text{def}}{=} \bigcup \{R \subseteq Q_G \times Q_G \mid R : G \underline{\leftrightarrow} G\}$$

En général, une auto-bisimulation de G n'est pas forcément une équivalence sur Q_G. Cependant, d'après la Proposition 2.2.1 si $R : G \underline{\leftrightarrow} G$ alors la fermeture

symétriqueΓtransitive et réflexive de R est aussi une auto-bisimulation de G[3]Γet doncΓ$\underline{\leftrightarrow}_G$ est une équivalence sur Q_G.

Si q_1, q_2 sont deux états d'un même graphe GΓon dira qu'ils sont bisimilaires (noté $q_1 \underline{\leftrightarrow} q_2$) si $q_1 \underline{\leftrightarrow}_G q_2$. De même on dira que deux états $q_1 \in Q_1, q_2 \in Q_2$ de deux graphes différents G_1 et G_2 sont bisimilaires si ils sont bisimilaires dans $G_1 \oplus G_2$. Par conséquentΓpour tous $S_1, S_2 \in \mathbf{ST}$Γon a

$$R : S_1 \underline{\leftrightarrow} S_2 \text{ ssi } r_{S_1} \underline{\leftrightarrow}_{G_{S_1} \oplus G_{S_2}} r_{S_2},$$

qui montre que $\underline{\leftrightarrow}$ est une relation d'équivalence sur $\mathbf{ST}$.

La définition ci-dessus nous autoriseΓsans perdre de généralitéΓà exprimer les relations d'équivalences soit entre les états d'un grapheΓpouvant être compris comme la somme de deux graphesΓsoit entre deux graphes distincts. Suivant le contexteΓ pour faciliter la lectureΓnous adopterons l'un ou l'autre de ces points de vue.

Remarque 1 *Dans* ***ST****, Chaque classe d'équivalence modulo bisimulation possède des représentants canoniques. Nous ne rentrons pas dans les détails, et nous renvoyons par exemple à [AD89].*

La bisimulation forte peut aussi être vue comme une relations entre les chemins dans un graphe. FormellementΓsi $\sigma = q_0 \stackrel{a_1}{\rightarrow} q_1 \stackrel{a_2}{\rightarrow} q_2 \ldots q_n \ldots$ et $\sigma' = q'_0 \stackrel{a'_1}{\rightarrow} q'_1 \stackrel{a'_2}{\rightarrow} q'_2 \ldots q'_n \ldots$ sont deux chemins de même longueur dans un graphe GΓon dit que σ et σ' sont bisimilairesΓnoté $\sigma \underline{\leftrightarrow} \sigma'$Γsi $q_i \underline{\leftrightarrow} q'_i$ et $a_i = a'_i$ pour tout i.

Soit $G \in \mathbf{G}$Γ$q, q' \in Q_S$. Si $q \underline{\leftrightarrow} q'$Γalors pour tout chemin $\sigma \in \Pi(q)$Γon peut construire pas à pas un chemin $\sigma' \in \Pi(q')$ tel que $\sigma \underline{\leftrightarrow} \sigma'$.

Corollaire 2.2.2 *Pour tous* $S, S' \in \boldsymbol{ST}$*,*

$$S \underline{\leftrightarrow} S' \textit{ implique } S \sim_{Tr} S'$$

La réciproque est évidemment fausseΓvoir l'Exemple 2.

2.2.1 Propriétés de congruence de la bisimulation (forte)

Comme nous l'avons déjà dit dans notre introductionΓ la bisimulation est compositionnelle pour une large classe de combinateurs de programmesΓvoir [Sim85] et [GV88]. Plus précisémentΓla bisimulation est une congruence pour tous les combinateurs dont la sémantique opérationnelle est décrite au moyen de règles SOS à la Plokin qui respectent un format syntaxique appelé *format tyft*. En considérant l'exemple particulier d'un sous-langage $\mathcal{L}$ du langage CCS de Milner (avec les opérations de préfixageΓde composition parallèle asynchrone et de choix non-déterministe)Γnous énonçons la compatibilité de l'égalité sémantique induite par

[3] voir aussi [Sif83] pour une généralisation des propriétés algébriques des relations de simulation.

la bisimulation vis à vis du combinateur de mise en parallèle de CCSΓpuis nous définissons formellement le format *tyft* de [GV88].

Le langage $\mathcal{L}$ est défini par la grammaire suivante : soit Act un ensemble de noms d'actionsΓet nil une constante particulière n'appartenant pas à Act.

$$\mathcal{L}(\in p,q) ::= nil|a(p)|p+q|p \mid\mid q$$

où $a \in Act$.

La définition opérationnelle de $\mathcal{L}$ est donnée par les règles conditionnelles suivantes (il y a une règle pour chaque $a \in Act$):

$$(\textit{préfixage}) \quad a(p) \xrightarrow{a} p$$

$$(\textit{choix non déterministe}) \quad \frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{a} q'}{p+q \xrightarrow{a} q'}$$

$$(\textit{composition parallèle}) \quad \frac{p \xrightarrow{a} p'}{p \mid\mid q \xrightarrow{a} p' \mid\mid q} \qquad \frac{q \xrightarrow{a} q'}{p \mid\mid q \xrightarrow{a} p \mid\mid q'}$$

Ces règles définissent les transitions d'un terme de $\mathcal{L}$ en fonction des transitions possibles de ses sous-termes. C'est en ce sens qu'elles sont conditionnelles. L'expression $p \xrightarrow{a} p'$ se lit "le programme p peut effectuer l'action a et après exécution de cette actionΓil se comporte comme le programme p'". Par exempleΓla première règle de *composition parallèle* se lit "si un programme p peut effectuer une action a et devenir le programme p'Γalors la composition parallèle de ce programme p et de n'importe quel autre programme q peut effectuer cette même action a et se transformer en $p' \mid\mid q$". L'expression $p \xrightarrow{a} p'$ est appelée la *prémisse* (ou encore *condition*) de la règleΓet l'expression $p \mid\mid q \xrightarrow{a} p' \mid\mid q$ sa *conclusion.*

On peut associer au langage $\mathcal{L}$ un graphe d'états $G_{\mathcal{L}}$ étiqueté sur Act défini par :

- l'ensemble des états de $G_{\mathcal{L}}$ est l'ensemble des termes du langage $\mathcal{L}\Gamma$
- la relation de transition dans $G_{\mathcal{L}}$ est l'ensemble des transitions $q \xrightarrow{a} q'\Gamma$ où $a \in Act$ et $q, q' \in \mathcal{L}\Gamma$ que l'on peut prouver en utilisant les règles opérationnelles de *préfixage*Γde *choix non-déterministe* et de *composition parallèle.*

Il est alors facile d'associer à chaque terme $p \in \mathcal{L}$ un ST$\Gamma ST(p) \stackrel{\text{def}}{=} (G_{\mathcal{L}}, p)$.

Considérons l'égalité sémantique des programmes de $\mathcal{L}$ basée sur la bisimulation forteΓc-à-d. deux programmes p_1 et p_2 de $\mathcal{L}$ sont dits "égaux" ssi $ST(p_1) \underline{\leftrightarrow} ST(p_2)\Gamma$ ce que nous notons $p_1 \underline{\leftrightarrow} p_2$. Cette égalité sémantique est compositionnelle pour le combinateur de programmes $\mid\mid \Gamma$i.e. pour tout $q \in \mathcal{L}\Gamma$

$$p_1 \underline{\leftrightarrow} p_2 \text{ implique } (p_1 \mid\mid q) \underline{\leftrightarrow} (p_2 \mid\mid q)$$

Ce type de résultat est très classique : en utilisant qu'il existe une bisimulation $R : ST(p_1) \underline{\leftrightarrow} ST(p_2)$Γon montre qu'il existe $R' : ST(p_1 \mid\mid q) \underline{\leftrightarrow} ST(p_2 \mid\mid q)$Γen raisonnant sur les transitions possibles des programmes $p_1 \mid\mid q$ et $p_2 \mid\mid q$Γqui sont complètement décrites par les règles de *composition parallèle*.
Cette technique de preuve s'applique aussi aux autres combinateurs de programmes de $\mathcal{L}$Γainsi qu'à tous les opérateurs du langage CCS.
Comme l'on montré [GV88] (voir aussi la version journal [GV92])Γces preuves se généralisent à une large classe de combinateurs de programmes (contenant évidemment les combinateurs de CCS) : il suffit que ces combinateurs soient décrits au moyen de règles SOS à la Plotkin respectant le format *tyft* que nous définissons formellement dans la suite.

Nous supposons donné un ensemble (dénombrable) X de *variables*Γd'éléments typiques $x, y, x_1, y_1, \ldots$.

Notations 2 *Soit une signature (à une sorte)* $\Sigma = (F, \nabla)$ *où* F *est un ensemble de* noms de fonctions *disjoint de* X, *et* $\nabla : F \to N$ *est l'arité des noms de fonctions de* F.

- T_Σ *dénote l'ensemble des termes* clos *(i.e. ne contenant pas de variable) sur la signature* Σ.
- *Nous notons* $T_\Sigma(X) \stackrel{def}{=} T_{\Sigma \cup X}$ *l'ensemble des termes sur* Σ *pouvant contenir des variables de* X.
- *Si* $t \in T_\Sigma(X)$, $Var(t) \subseteq X$ *dénote l'ensemble des variables qui apparaissent dans le terme* t.

De façon généraleΓon peut montrer que pour une très large classe d'ensembles de règles SOSΓil est possible de dériver un graphe (d'états). Nous appelons ici de tels ensembles de règles des *spécifications de graphe* (voir aussi la terminologie *spécification de systèmes de transitions* dans [GV88]). Comme l'ont étudié [GV88] et [Gro90]Γles spécifications utilisant des règles SOS contenant des prémisses négativesΓ comme on le trouve dans le format $ntyxt/ntyft$ de [Gro90] ou encore dans le format *GSOS* de [BIM88]Γposent de sérieux problèmes pour montrer l'existence d'un ST canonique associé à cette spécification.
Dans notre thèse nous ne considérons que des règles sans prémisses négatives.

Définition 2.2.3 (Spécification de graphe (d'états)) *Une* spécification de graphe (d'états) *(SG) est une structure* $P = (\Sigma, A, R)$ *où*

- Σ *est une signature,*
- A *est un ensemble de noms d'actions,*

- *R est un ensemble de règles de la forme*

$$\frac{\{t_i \xrightarrow{a_i} t'_i \mid i \in I\}}{t \xrightarrow{a} t'}$$

où I est un ensemble fini, $t_i, t'_i, t, t' \in T_\Sigma(X)$, $a, a_i \in A$.

Les expressions $t_i \xrightarrow{a_i} t'_i$ s'appellent les prémisses *de la règle, et $t \xrightarrow{a} t'$ sa* conclusion. *Les règles sans prémisses sont appelées des* axiomes.

Nous supposons connue la notion de *preuve* d'une transition $t \xrightarrow{a} t'$ (où t et t' sont des termes clos) à partir de P : la preuve est basée sur les règles de R et sur des instantiations des variables qui apparaissent dans ces règles.
Classiquement, nous notons $P \vdash t \xrightarrow{a} t'$ si il existe une preuve de la transition $t \xrightarrow{a} t'$. Nous étendons canoniquement le sens de l'expression $P \vdash \ldots$ à toute séquence de transitions de la forme $t \xrightarrow{a_1} t_1 \xrightarrow{a_2} \ldots$.

Soit $P = (\Sigma, A, R)$ une SG. Le *graphe d'états spécifié par P* est le graphe d'états étiqueté sur A, noté $G(P)$, d'ensemble d'états T_Σ et de relation de transition $\rightarrow_P \subseteq T_\Sigma \times A \times T_\Sigma$ définie par : pour tous $t, t' \in T_\Sigma$,

$$t \xrightarrow{a}_P t' \text{ iff } P \vdash t \xrightarrow{a} t'$$

Dans la suite, nous écrivons plus simplement $\rightarrow$ au lieu de $\rightarrow_P$.
Ainsi, à tout terme $t_0 \in T_\Sigma$ on associe le ST noté $ST(t_0)$, de graphe d'états $G(P)$ et d'état initial t_0. Clairement, si π est une exécution dans $G(P)$ partant de t_0, alors, par définition des transitions dans $G(P)$, il existe une preuve de π. Cette preuve est infinie si π est infinie.

Les travaux de [GV88] ont mis en évidence un critère syntaxique sur les règles d'une SG, appelé *format tyft (pur)* qui permet d'énoncer :

Théorème 2.2.3 [GV88] *Si $P = (\Sigma, A, R)$ est une SG en format* tyft, *alors la bisimulation forte est une congruence pour tous les noms de fonctions, i.e. pour tout $f \in F$ et tous termes clos $u_i, v_i \in T_\Sigma$ $(1 \leq i \leq \nabla(f))$,*

$$\forall i, u_i \underline{\leftrightarrow}_{G(P)} v_i \text{ impliquent } f(u_1, \ldots, u_n) \underline{\leftrightarrow}_{G(P)} f(v_1, \ldots, v_n)$$

Nous définissons formellement le format *tyft* de la façon suivante :

Définition 2.2.4 *Une règle est en* format tyft *si elle est de la forme :*

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, \ldots, x_n) \xrightarrow{a} t}$$

où I est un ensemble fini d'indices, $x_j (1 \leq j \leq n), y_i (i \in I)$ sont des variables distinctes, $a, a_i \in A$, $t_i, t \in T_\Sigma(X)$.

On remarquera en particulier que les règles de CCS sont toutes de ce format. Il est important de mentionner que [GV92] ne se restreignent pas à un ensemble de prémisses finiesΓmais autorisent un ensemble I infini.

Il faut cependant rajouter quelques contraintes syntaxiques sur les règles en format *tyft* pour ne définir que des combinateurs compatibles avec la bisimulation. Ces contraintes sont en particulier respectées par les règles de CCS :

Nous notons $Var(r)$ l'ensemble des variables qui apparaissent dans la règle r. Une variable dans $Var(r)$ est *libre* si elle n'apparaît pas dans le membre gauche de la conclusion de r ou dans le membre droit d'une prémisse de r.

L'Exemple 3Γtiré de [GV88]Γest un exemple de règle dite *circulaire* dans laquelle y_1 dépend de y_2 et réciproquement. [GV88] excluent de telles règles pour lesquelles l'existence d'un ST canonique (en accord avec un ensemble de règles donné) est difficile à prouver.

Exemple 3 *Circularité.*

$$\frac{f(x, y_2) \xrightarrow{a} y_1 \quad g(x', y_1) \xrightarrow{b} y_2}{x \xrightarrow{c} x'}$$

Dans [GV88]Γ la circularité d'une règle est définie par le biais du *graphe de dépendance* de celle-ci. Nous définissons formellement cette notion qui nous servira aussi au Chapitre 4.

Pour $t \xrightarrow{a} t'$ une transition dans $G(P)$Γnous notons $Var(t \xrightarrow{a} t')$ l'ensemble de variables $Var(t) \cup Var(t')$. La Définition 2.2.5 est extraite de [GV88].

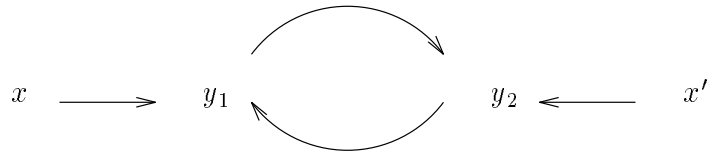


Figure 2.3: Un exemple de graphe de dépendance de règle SOS

Définition 2.2.5 (Graphe de dépendance et Circularité) *Soit $P = (\Sigma, A, R)$ une SG et soit $S \stackrel{def}{=} \{t_i \xrightarrow{a} t'_i | i \in I\}$ l'ensemble des transitions dans $G(P)$. Le* graphe de dépendance *de S est le graphe (dirigé, non étiqueté) dont l'ensemble des états est $\bigcup_{i \in I} Var(t_i \xrightarrow{a} t'_i)$ et l'ensemble des transitions est donné par $x \rightarrow y$ si il existe un $i \in I$ tel que $x \in Var(t_i)$ et $y \in Var(t'_i)$.*

Un ensemble de transitions est circulaire *si son graphe de dépendance contient un cycle.*

Une règles est circulaire *si l'ensemble de ses prémisses est circulaire.*

Un ensemble de règles est circulaire *si une des règles est circulaire.*
Enfin, une SG est circulaire *si l'ensemble de ses règles est circulaire.*

La Figure 2.3 représente le graphe de dépendance de la règle de l'Exemple 3.

Définition 2.2.6 (SG en format* tyft*) *Une SG $P = (\Sigma, A, R)$ est en* format *tyft si pour toute $r \in R$, r est une règle en format* tyft *et r est* pure, *c-à-d. r ne contient pas de variable libre et n'est pas circulaire.*

Au Chapitre 4, nous proposons une sous-classe syntaxique du format *tyft*, peu restrictive dans la mesure où l'on peut encore exprimer tous les combinateurs de CCS. Ce format est appelé *tyft sans copie* ou plus simplement *sans copie*. Nous montrons que cette restriction syntaxique est réelle au sens où la congruence de trace induite par les contextes *tyft* est strictement plus fine que celle induite par les contextes *sans copie*.

2.3 Définition par point-fixe de la bisimulation (forte)

La bisimulation forte de la Définition 2.2.1 peut être caractérisée comme un point-fixe. Nous adoptons les notations de [Mil88].

Soit $G = (Q, \rightarrow) \in \mathbf{G}$. On note $(2^{Q\times Q}, \subseteq)$ le treillis complet des relations de Q dans Q et ordonné par l'inclusion. Nous définissons $\mathcal{F} : 2^{Q\times Q} \rightarrow 2^{Q\times Q}$ par :
$q_1 \mathcal{F}(R) q_2$ ssi pour tout $a \in A$,

1. si $q_1 \stackrel{a}{\rightarrow} q_1'$, alors il existe $q_2 \stackrel{a}{\rightarrow} q_2'$ telle que $q_1' R q_2'$, et
2. si $q_2 \stackrel{a}{\rightarrow} q_2'$, alors il existe $q_1 \stackrel{a}{\rightarrow} q_1'$ telle que $q_1' R q_2'$.

La Définition 2.2.1 exprime qu'une auto-bisimulation R de G vérifie $R \subseteq \mathcal{F}(R)$. Il est facile de vérifier que la fonction $\mathcal{F}$ est monotone (i.e. $R_1 \subseteq R_2$ implique $\mathcal{F}(R_1) \subseteq \mathcal{F}(R_2)$); la plus grande relation R telle que $R \subseteq \mathcal{F}(R)$ est donc

$$\bigcup \{R | R \subseteq \mathcal{F}(R)\}$$

autrement dit, d'après le Théorème de Tarski [Tar55], le plus grand point-fixe de $\mathcal{F}$.

Par conséquent, si *pgpf* f dénote le plus grand point fixe d'une application f, alors

$$\underline{\leftrightarrow}_G = pgpf\,\mathcal{F}$$

On peut également définir une fonction $\mathcal{G} : 2^{Q\times Q} \rightarrow 2^{Q\times Q}$ en étendant la relation $\rightarrow$ aux mots de A^* dans les clauses 1. et 2. de la définition de $\mathcal{F}$. Cette fonction $\mathcal{G}$ nous servira à comparer la bisimulation forte à une variante de l'équivalence observationnelle définie en Section 2.4. On peut montrer que $\mathcal{G}$ admet des propriétés similaires à $\mathcal{F}$ et que

$$\underline{\leftrightarrow}_G = pgpf\mathcal{G}$$

Dans [HM85]Γil est montré que si le graphe G est à b.f. alors la transformation $\mathcal{F}$ est $\cap$-continue. Ce résultat se démontre aussi pour $\mathcal{G}\Gamma$car la propriété d'être à b.f. reste vraie lorsque la relation de transition est étendue aux mots de A^*. Nous rappelons qu'une fonction f est $\cap$-continue si pour toute suite décroissante $R_0 \supseteq R_1 \supseteq \ldots R_n \ldots\Gamma$

$$f(\bigcap_{i<\omega} R_i) = \bigcap_{i<\omega} f(R_i)$$

Dans la section suivanteΓ nous présentons l'équivalence observationnelle telle que [Mil80] l'a définieΓainsi qu'une varianteΓque nous comparons à la bisimulation.

2.4 Equivalences projectives

Les équivalences projectives que sont l'*équivalence simple* et l'*équivalence double*Γ sont aussi basées sur un principe de simulation entre les systèmes. La première est l'équivalence observationnelle de Milner [Mil80] ; la seconde en est une variante. Les définitions et les résultats sont extraits de [SP90].

Définition 2.4.1 (Équivalence simple) *Soit $G = (Q, \rightarrow) \in \mathbf{G}$. On définit les relations $\equiv_n \subseteq Q_G \times Q_G$ pour $n < \omega$ par:*

- $q_1 \equiv_0 q_2$ *pour tous* $q_1, q_2 \in Q$,
- $q_1 \equiv_{n+1} q_2$ *ssi*
  - *pour tout* $q_1 \stackrel{a}{\rightarrow} q_1'$, *il existe* $q_2 \stackrel{a}{\rightarrow} q_2'$ *tel que* $q_1' \equiv_n q_2'$,
  - *et réciproquement, pour tout* $q_2 \stackrel{a}{\rightarrow} q_2'$, *il existe* $q_1 \stackrel{a}{\rightarrow} q_1'$ *tel que* $q_1' \equiv_n q_2'$.

On définit $\equiv$ par

- $q_1 \equiv q_2$ *ssi* $q_1 \equiv_n q_2$ *pour tout* $n < \omega$.

Les relations $\equiv_n$ sont des équivalences sur Q.
Si $S_1, S_2 \in \mathbf{ST}\Gamma$on pose

$$S_1 \equiv_n S_2 \stackrel{\text{def}}{=} r_{S_1} \equiv_n r_{S_2} \text{ dans } G_{S_1} \oplus G_{S_2}$$

$$S_1 \equiv S_2 \stackrel{\text{def}}{=} r_{S_1} \equiv r_{S_2} \text{ dans } G_{S_1} \oplus G_{S_2}$$

IntuitivementΓl'équivalence $\equiv$ de la Définition 2.4.1 distingue deux états q_1 et q_2 s'ils peuvent être distingués à un niveau n finiΓe.g. si q_1 peut effectuer une transition $q_1 \stackrel{a}{\rightarrow} q_1'$ et si q_1' est distingué au niveau $n-1$ de tous les a-successeurs de q_2. En particulierΓ$\equiv_1$ distingue des états à partir desquels il n'est pas possible d'exécuter

les mêmes actions.

Une variante immédiate de l'équivalence simple peut être proposée en considérant la relation de transition étendue aux mots dans la Définition 2.4.1.

Définition 2.4.2 (Équivalence double)
Soit $G = (Q, \rightarrow) \in \boldsymbol{G}$. On définit les relations $\approx_n$ pour $n < \omega$ par:

- $q_1 \approx_0 q_2$ *pour tous* $q_1, q_2 \in Q$,
- $q_1 \approx_{n+1} q_2$ *ssi*
  - pour tout $q_1 \stackrel{w}{\rightarrow} q_1'$ *(où* $w \in A^*$*), il existe* $q_2 \stackrel{w}{\rightarrow} q_2'$ *tel que* $q_1' \approx_n q_2'$
  - et réciproquement, pour tout $q_2 \stackrel{w}{\rightarrow} q_2'$ *(où* $w \in A^*$*), il existe* $q_1 \stackrel{w}{\rightarrow} q_1'$ *tel que* $q_1' \approx_n q_2'$.

On définit $\approx$ *par*

- $q_1 \approx q_2$ *ssi* $q_1 \approx_n q_2$ *pour tout* $n < \omega$.

De même que $\equiv$ la relation $\approx$ définie une équivalence entre les états d'un graphe Γ et peut être étendue de la même façon en une équivalence sur **ST** par :

$$S_1 \approx S_2 \stackrel{\text{def}}{-} r_{S_1} \approx r_{S_2} \text{ dans } G_{S_1} \oplus G_{S_2}$$

Comme l'équivalence simple Γ l'équivalence double distingue deux états q_1 et q_2 s'ils sont distingués à un niveau fini Γ mais la notion de niveau change. Ici Γ deux états q_1 et q_2 sont équivalents au niveau n si ils ont les mêmes préfixes de traces et si pour chaque mot w qui soit un préfixe fini d'une trace partant de $q_1 \Gamma$ et pour tout w-successeur q_1' de $q_1 \Gamma$ on peut trouver un w-successeur de q_2 équivalent à q_1' au niveau $n-1$.
Nous verrons dans la Section 2.5 que $\approx$ est plus fine que $\equiv$.

La bisimulation forte est en général plus fine que ses deux variantes $\equiv$ et $\approx$ que nous venons de présenter. C'est ce que nous rappelons dans la section qui suit.

2.5 Bisimulation et équivalences projectives

Nous étudions les liens entre la bisimulation et les équivalences projectives de la Section 2.4. Elles coïncident dans les classes $\mathbf{G}_{bf}$ des graphes à b.f. Les résultats qui relient ces équivalences sont classiques Γ nous renvoyons par exemple à [Mil88]:

Proposition 2.5.1 *Pour tous* $S_1, S_2 \in \boldsymbol{G}$, *on a*

$$S_1 \underline{\leftrightarrow} S_2 \;\Rightarrow\; S_1 \approx S_2 \;\Rightarrow\; S_1 \equiv S_2 \tag{2.1}$$

si de plus $S_1, S_2 \in \boldsymbol{G}_{bf}$,

$$S_1 \equiv S_2 \;-\; S_1 \approx S_2 \;-\; S_1 \underline{\leftrightarrow} S_2 \tag{2.2}$$

Preuve D'après les Définitions 2.4.1 et 2.4.2Γsi $S_1 \approx S_2$Γalors $S_1 \equiv S_2$.

Soit $G \in \mathbf{G}$. Nous utilisons les assertions suivantes :

(1) Pour tout $n, \underline{\leftrightarrow}_G = \mathcal{F}^n(\underline{\leftrightarrow}_G)$
(2) $\equiv = \bigcap_{n<\omega} \mathcal{F}^n(Q \times Q)$
(3) $\approx = \bigcap_{n<\omega} \mathcal{G}^n(Q \times Q)$
(4) Si $G \in \mathbf{G}_{bf}, \mathcal{F}$ et $\mathcal{G}$ sont $\cap$-continues.

L'assertion (1) se justifie par le fait que $\underline{\leftrightarrow}_G = pgpf\mathcal{F}$. (2) et (3) sont respectivement les définitions de $\equiv$ et $\approx$. Nous renvoyons à [HM85] pour le point (4).

Les points (1) et (3) montrent que si $q \underline{\leftrightarrow} q'$Γalors $q \approx q'$.

Supposons maintenant $G \in \mathbf{G}_{bf}$. D'après (4) $\mathcal{F}$ est $\cap$-continueΓet donc $\equiv = \cap_{n<\omega}\mathcal{F}^n(Q \times Q) = \mathcal{F}(\bigcap_{n<\omega} \mathcal{F}^n(Q \times Q)) = \mathcal{F}(\equiv)$. D'après le Théorème de KleeneΓ $\equiv$ est le plus grand point-fixe de $\mathcal{F}$Γc-à-d. $\underline{\leftrightarrow}_G$Γce qui prouve le Point (2.2) de la proposition. □

Remarque 2 • *Il existe une autre preuve du Point (2.2) de la Proposition 2.5.1, proposée dans [BBK87], n'utilisant pas la définition par point-fixe de $\underline{\leftrightarrow}$. Dans ce même article, il est montré que le Point (2.2) reste vrai lorsque l'un des deux graphes seulement est à b.f.*

L'hypothèse de graphes à b.f. est fondamentale pour montrer que les équivalences $\underline{\leftrightarrow}$Γ$\equiv$ et $\approx$ coïncident. En effetΓconsidérons l'exemple désormais classique de la Figure 2.4 : S_1 et S_2 ne sont pas bisimilaires puisqu'ils n'ont pas les mêmes traces (S_2 admet la trace infinie a^ωΓce qui n'est pas le cas pour S_1). En revancheΓ $S_1 \equiv S_2$. Il suffit de remarquer que la branche a^ω de S_2 peut être simulée à chaque niveau n par une des branches de S_1.

L'équivalence double $\approx$ ne coïncide pas non plus avec $\underline{\leftrightarrow}$ lorsque les graphes sont à b.i. Le premier contre-exemple proposé se trouve dans [San82] page 117 et reste très compliqué. Nous renvoyons au Chapitre 4 qui propose un cadre plus simple pour établir ce résultat.

2.6 Bisimulations modulo τ

Dans les systèmes que nous avons considérés jusqu'à présentΓune transition ne peut avoir lieu sans être observée.

Nous introduisons dans cette section le concept classique d'action invisible de l'extérieur (ou encore d'action "silencieuse")[4]. Par exempleΓdans l'algèbre de processus ACP_τ de [BK85]Γl'abstraction apparaît par le renommage des actions en une

[4]Nous traiterons également l'introduction d'actions invisibles dans les structures d'événements, voir la Partie 2.

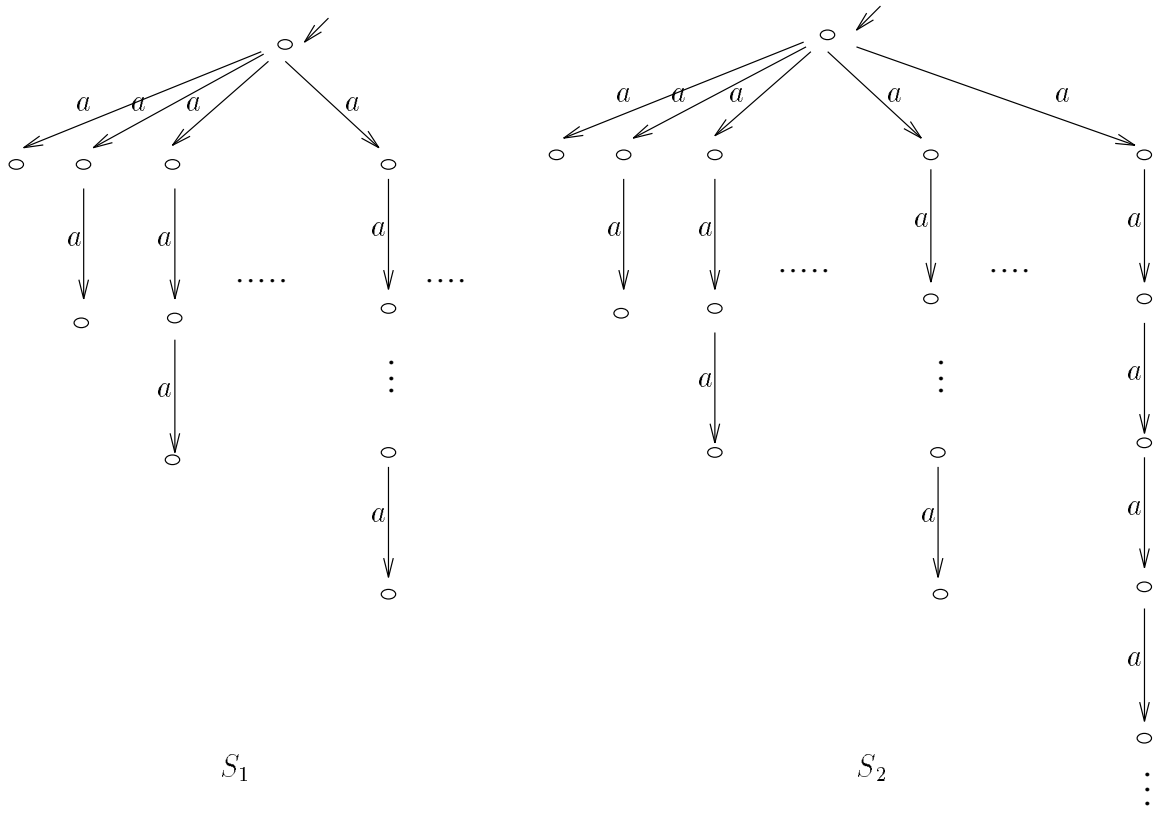


Figure 2.4: $S_1 \equiv S_2$ mais $S_1 \not\underline{\leftrightarrow} S_2$.

action silencieuse appelée τ. Dans le calcul CCS de Milner [Mil81]Γla transition silencieuse résulte de la synchronisation dans la communication des processus.

Pour cela nous ajoutons à l'alphabet d'actions A un élément particulier τ pour dénoter les actions invisibles[5].

Nous notons $A_\tau = \{\nu, \ldots\}$ l'ensemble $A \cup \{\tau\}$ et nous considérons dans la suite de ce chapitre la classe des graphes (resp. ST) sur l'ensemble d'actions A_τΓque nous notons encore **G** (resp. **ST**).

Nous supposons donné un élément $\epsilon \notin A_\tau$Γet nous notons A_ϵ l'ensemble $A \cup \{\epsilon\}$. Les élément typiques de A_ϵ seront notés $\lambda, \lambda', \ldots$

Soit $G \in \mathbf{G}$. Pour tout $\nu \in A_\tau$ nous notons $q \stackrel{\nu}{\Rightarrow} q'$ lorsqu'il existe $n, m \geq 0$ tels que $q \stackrel{\tau^n \nu \tau^m}{\rightarrow} q'$. IntuitivementΓune transition $\stackrel{\nu}{\Rightarrow}$ permet d'absorber un nombre fini de τ avant ou après exécution de l'action ν; nous écrivons $q \stackrel{\epsilon}{\Rightarrow} q'$ lorsque $q \stackrel{\tau^n}{\rightarrow} q'$ pour $n \geq 0$.

[5]Il est clair qu'il n'est pas nécessaire d'introduire une constante pour chaque action invisible, voir [Mil81].

L'introduction du $\tau\Gamma$ c-à-d. des action silencieusesΓ dans les modèles conduit naturellement à définir de nouvelles équivalences comportementales.

2.6.1 τ-bisimulation, équivalence observationnelle

La τ-bisimulation de [BK85] (ou encore équivalence observationnelle de Milner) est une bisimulation qui prend en compte l'invisibilité de certaines des actions. Pour celaΓon définit sa propriété de transfert pour les relations de transition $\stackrel{\lambda}{\Rightarrow}\Gamma$où $\lambda \in A_\epsilon$ au lieu de $\stackrel{\nu}{\rightarrow}\Gamma$où $\nu \in A_\tau\Gamma$comme pour la bisimulation forte.

Définition 2.6.1 (τ-bisimulation) *Etant donné $G \in \mathbf{G}$, une relation $R \subseteq Q_G \times Q_G$ est une* τ-bisimulation *de G, noté $R : G \underline{\leftrightarrow}_\tau G$, ssi pour tous $q_1 R q_2$,*

1. *pour toute transition $q_1 \stackrel{\lambda}{\Rightarrow} q_1'$, il existe un état q_2' de G_2 tel que $q_2 \stackrel{\lambda}{\Rightarrow} q_2'$ et $q_1' R q_2'$,*
2. *réciproquement, pour toute transition $q_2 \stackrel{\lambda}{\Rightarrow} q_2'$, il existe un état q_1' de G_1 tel que $q_1 \stackrel{\lambda}{\Rightarrow} q_1'$ et $q_1' R q_2'$.*

Une τ-bisimulation entre S_1 et S_2 ($\in \mathbf{ST}$) est une relation R telle que :

- $R : G_{S_1} \oplus G_{S_2} \underline{\leftrightarrow}_\tau G_{S_1} \oplus G_{S_2}$, *et*
- $r_{S_1} R r_{S_2}$,

ce que nous notons $R : S_1 \underline{\leftrightarrow}_\tau S_2$. Nous notons $S_1 \underline{\leftrightarrow}_\tau S_2$ si il existe une relation R telle que $R : S_1 \underline{\leftrightarrow}_\tau S_2$.

Des propriétés analogues à celles de la Proposition 2.2.1 sont faciles à obtenir ; il existe toujours une plus grande τ-bisimulation entre les états d'un graphe G donné. Cette τ-bisimulation est une équivalence sur les états de $G\Gamma$ce qui garantit que $\underline{\leftrightarrow}_\tau$ est une relation équivalence sur la classe **ST**.

La définition de $\underline{\leftrightarrow}_\tau$ est clairement moins stricte que celle de $\underline{\leftrightarrow}$ puisque nous exigeons simplement d'une action τ qu'elle soit simulée par zéro ou plusieurs actions τ (alors que pour $\underline{\leftrightarrow}$ chaque action $\lambda \in A_\tau$ doit être simulée par exactement une action λ même si il s'agit d'un τ). Remarquons que si G ne contient pas de τ-transitionΓ $R : G \underline{\leftrightarrow}_\tau G$ ssi $R : G \underline{\leftrightarrow} G$.

Exemple 4 *La Figure 2.5 nous montre deux ST τ-bisimilaires. En effet, il suffit de vérifier que la transition $q \stackrel{a}{\rightarrow} q'$ de S_2 est peut être imitée dans S_1, puisque c'est la seule différence entre les deux graphes. Clairement, cette transition est imitée dans S_1 par la suite de transitions $p \stackrel{\tau a}{\rightarrow} p'$, et on a bien p' bisimilaire à q' puisque ces deux états sont terminals.*

Comme le souligne [GW89a]Γ la τ-bisimulation ne préserve pas la structure arborescente des systèmes. C'est la raison pour laquelle la *branching bisimulation* a été proposée.

La section qui suit indique les bisimulation "avec τ" les plus connues.

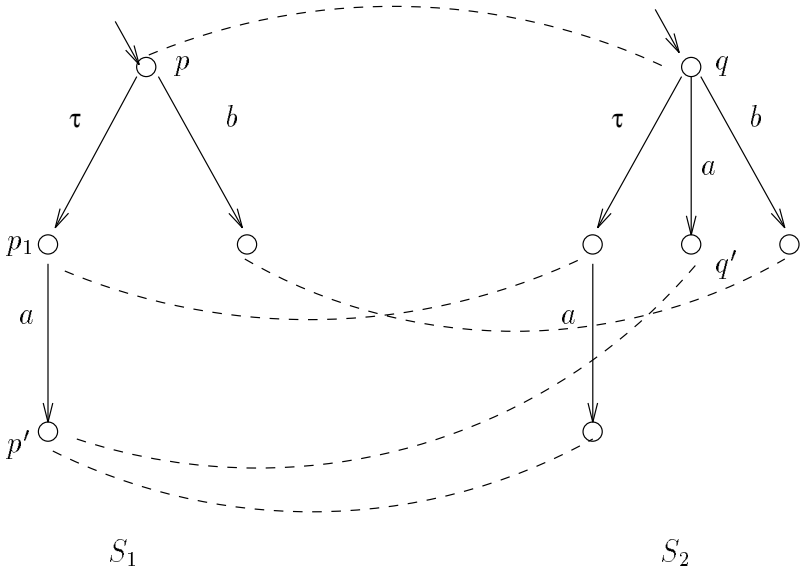


Figure 2.5: Un exemple de τ-bisimulation.

2.6.2 Bisimulations modulo τ

Les bisimulations modulo τ proposées dans la littérature sont les suivantes :

- La *tau*-bisimulationΓou encore *équivalence observationnelle* de [Mil80] et [BK85]Γ
- La *branching bisimulation* [GW89a]Γque nous notons $\underline{\leftrightarrow}_b$Γ
- La η*-bisimulation* [BG87]Γ
- La Δ*-bisimulation* [Wei89] ou *delay-bisimulation* [Wal88]Γdéjà considérée dans [Mil81].
- La *quasi-branching* bisimulation de [Che92b].

Nous renvoyons à [Wei89] pour une comparaison exhaustive des bisimulations modulo τ.
CependantΓnous définissons formellement la *branching bisimulation* de [GW89a] qui nous servira au Chapitre 9.

Définition 2.6.2 (Branching bisimulation) *Etant donné $G \in \boldsymbol{G}$, une relation symétrique $R \subseteq Q \times Q$ est une* branching bisimulation *de G, notée $R : G \underline{\leftrightarrow}_b G$, ssi pour tous $q_1 R q_2$, si $q_1 \xrightarrow{\lambda} q_1'$, alors*

1. *$\lambda = \tau$ et $q_1' R q_2$, ou*
2. *il existe $q_2 \stackrel{\epsilon}{\Rightarrow} p_2 \xrightarrow{\lambda} q_2'$ tels que $q_1 R p_2$ et $q_1' R q_2'$.*

Soient $S_1, S_2 \in \boldsymbol{ST}$. *On pose* $R : S_1 \underline{\leftrightarrow}_b S_2$ *ssi :*

- $R : G_{S_1} \oplus G_{S_2} \underline{\leftrightarrow}_b G_{S_1} \oplus G_{S_2}$, *et*
- $r_{S_1} R r_{S_2}$.

On note $S_1 \underline{\leftrightarrow}_b S_2$ *lorsque qu'il existe* $R : S_1 \underline{\leftrightarrow}_b S_2$.

L'équivalence $\underline{\leftrightarrow}_b$ induit une équivalence sur **ST** et il existe toujours une plus grande branching bisimulation.

Intuitivement, cette équivalence autorise d'imiter une transition visible, étiquetée par exemple par a, par une suite de transitions comprenant d'abord un nombre fini de τ-transitions (éventuellement aucune) puis une a-transition. La différence majeure avec la τ-bisimulation est que l'état que l'on atteint juste avant d'imiter l'action a (dans la Définition 2.6.2, il s'agit de l'état p_2) doit être bisimilaire à l'état de départ. En fait, on montre facilement que lors de l'exécution des τ-transitions qui précédent la a-transition, les états intermédiaires sont tous équivalents en terme de leur branchement, d'où la terminologie "branching".

Exemple 5 *Dans la Figure 2.5,* $S_1 \not\underline{\leftrightarrow}_b S_2$. *En effet, pour imiter la transition* $q \xrightarrow{a} q'$ *de* S_2, *le système* S_1 *passe par l'état intermédiaire* p_1 *; mais* p_1 *n'est pas équivalent à* q *puisque seul* q *admet un b-successeur.*

Chapitre 3

Equivalences logiques

Comme l'a souligné Pnueli dans [Pnu77]Γles *logiques temporelles*Γet plus généralement les *logiques modales* constituent un formalisme approprié pour exprimer les propriétés des systèmes réactifs (voir p. ex. [MP92] pour une synthèse). [Lam83] fait remarquer qu'une propriété exprimée en logique temporelle a un sens à n'importe quel niveau d'abstraction sur le programme.

Dans cette thèseΓnous nous intéressons aux logiques modales et temporelles en temps que langages de description de propriétés comportementales des systèmesΓla notion de comportement étant induite par l'équivalence sémantiqueΓdans notre cas la bisimulation. L'approche classique pour montrer qu'un langage logique donné exprime des propriétés comportementales consiste à comparer l'équivalence sémantique à l'équivalence induite par la logiqueΓque l'on appelle plus simplement *équivalence logique* dans la suite (Nous rappelons que deux systèmes sont équivalents pour la logique s'ils satisfont les mêmes formules. Voir la Section 3.3 pour les définitions formelles). Selon les résultats de cette comparaison on peut en déduire l'intérêt d'utiliser ce langage de propriétés : par exempleΓsi on montre que l'équivalence sémantique "implique" l'équivalence logiqueΓnous sommes certains que les formules de la logique expriment bien les propriétés comportementales des systèmes. On dit que *la logique est compatible avec la sémantique*. Si ce n'est pas le casΓla logique est trop expressiveΓc-à-d. qu'elle permet de "parler" de certaines différences entre

les systèmes que l'on a déclarés superflues par la sémantique.

Ou encore, si les deux équivalences coïncident, la logique est suffisamment expressive pour distinguer deux systèmes dont les comportements diffèrent. On dit alors que *la logique est adéquate pour la sémantique* ou encore que *la logique caractérise l'équivalence sémantique* (voir [Pnu85] par exemple). On peut alors expliquer dans la logique pourquoi deux programmes ne sont pas sémantiquement égaux (voir p. ex. les travaux de [Cle90], [Hil87]).

3.1 La logique HML

Dans cette section, nous présentons la logique modale HML introduite par Hennessy et Milner dans [HM80], ainsi que sa version pour la présence d'actions silencieuses, HML_τ. Puis nous définissons l'équivalence logique induite par HML, et nous rappelons ses liens avec l'équivalence sémantique de bisimulation forte.

3.1.1 Syntaxe de HML

Nous supposons donné un ensemble $A = \{a, \ldots\}$ de noms d'actions.

Définition 3.1.1 (Syntaxe de HML) *Les formules de HML sont données par la grammaire suivante :*

$$\varphi, \psi \ (\in HML) ::= \top \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle a \rangle \varphi$$

où $a \in A$.

L'ensemble des formules de HML est stratifié en fonction de la profondeur $d(\varphi)$ des formules, suivant la définition inductive :

$$\begin{array}{ll} d(\top) = 0 & d(\neg\varphi) = d(\varphi) \\ d(\varphi \wedge \psi) = max(d(\varphi), d(\psi)) & d(\langle a \rangle \varphi) = d(\varphi) + 1 \end{array}$$

Par exemple, $d(\langle a \rangle(\langle b \rangle \top \vee \langle c \rangle \top)) = 2$. Intuitivement, la profondeur de φ est définie comme le nombre maximum d'emboîtements de modalités de la forme $\langle . \rangle$ dans φ.

Nous noterons HML_n l'ensemble des formules de HML de profondeur au plus égale à n, i.e. $HML_n \stackrel{\text{def}}{=} \{\varphi \in HML | d(\varphi) \leq n\}$.

3.1.2 Sémantique de HML

Une formule de HML s'interprète sur les états dans un graphe étiqueté sur l'alphabet d'actions A. C'est donc une logique "du temps arborescent" au sens de [Lam80] ; on dit aussi *logique d'états*.

HML contient les connecteurs standards de la logique propositionnelle: $\top$ correspond à la proposition "vrai"Γ$\neg$ à la négation et $\wedge$ à la conjonction. Les opérateurs modaux sont les $\langle a\rangle$Γoù $a \in A$ dont voici la sémantique informelle: la formule $\langle a\rangle\varphi$ est vraie d'un état $q \in Q$ si il existe un a-successeur de q dans lequel la formule φ est vraie.

FormellementΓétant donné $G = (Q, A, \rightarrow) \in \mathbf{G}$Γnous définissons une relation de satisfaction $\models_G \subseteq Q \times HML$. On écrit $q \models_G \varphi$ (ou $q \models \varphi$ lorsque G est sous-entendu) et on dit "q satisfait la formule φ" ou encore que "la formule φ est vraie dans l'état q". La définition de $\models_G$ se fait par induction sur la structure des formules :

Définition 3.1.2 (Sémantique de** HML**) *Pour tous $q \in Q$ et $\varphi, \psi \in HML$,*

- $q \models_G \top$ *toujours,*
- $q \models_G \varphi \wedge \psi$ *ssi* $q \models_G \varphi$ *et* $q \models_G \psi$
- $q \models_G \neg\varphi$ *ssi* $q \not\models_G \varphi$
- $q \models_G \langle a\rangle\varphi$ *ssi il existe une transition dans G de la forme $q \xrightarrow{a} q'$ telle que $q' \models_G \varphi$.*

Pour tout $S \in ST$, on dérive une notion de "S satisfait φ", noté $S \models \varphi$, par

$$S \models \varphi \stackrel{def}{=} r_S \models_{G_S} \varphi$$

Notations 3

- $\varphi \vee \psi \stackrel{def}{=} \neg(\neg\varphi \wedge \neg\psi)$ *est la disjonction entre les formules φ et ψ.*
- *Pour tout $a \in A$, $[a]\varphi \stackrel{def}{=} \neg\langle a\rangle\neg\varphi$ est l'opérateur dual de $\langle a\rangle$. Un état satisfait $[a]\varphi$ si tous ses a-successeurs satisfont φ.*

Exemple 6 *Les ST de la Figure 2.1 ne satisfont pas les mêmes formules de HML. En effet, soit $\varphi \stackrel{def}{=} \langle a\rangle\neg(\langle b\rangle\top \wedge \langle c\rangle\top)$, alors $S_2 \models \varphi$ mais $S_1 \not\models \varphi$. La formule φ exprime qu'il existe une a-transition atteignant un état qui n'a pas à la fois un b-successeur et un c-successeur.*

3.2 La logique HML_τ

On étend naturellement la définition de HML en incluant l'action silencieuse τ dans le comportement. La logique obtenueΓnotée HML_τΓa la syntaxe suivante :

$$\varphi, \psi\ (\in HML_\tau) ::= \top \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle\lambda\rangle\varphi$$

où $\lambda \in A_\epsilon$.

Les modalités $\langle\lambda\rangle$ sont interprétées modulo τ. Formellement :

$q \models_G \langle\lambda\rangle\varphi \in HML_\tau$ ssi il existe une transition $q \stackrel{\lambda}{\Rightarrow} q'$ telle que $q' \models_G \varphi$

3.3 Equivalence induite par une logique

Etant donnée une logique d'états LΓ$G = (Q, \rightarrow) \in \mathbf{G}$Γ$q \in Q$Γon définit la *$L$-théorie* de q par

$$\mathcal{T}h_L(q) \stackrel{\text{def}}{=} \{\varphi \in L \mid q \models \varphi\}$$

Toute logique d'états L définit alors une équivalence sur les états de G par : deux états sont équivalents ssi ils ont la même L-théorieΓc-à-d. s'ils satisfont exactement le même ensemble de formules de L[1].

FormellementΓcette équivalenceΓappelée *L-équivalence*Γest définie par

$$q_1 \sim_L q_2 \stackrel{\text{def}}{=} \mathcal{T}h_L(q_1) = \mathcal{T}h_L(q_2)$$

Nous étendons à tout sous-ensemble de formules $\Phi \subseteq L$Γla définition de $\sim_L$ par

$$q_1 \sim_\Phi q_2 \stackrel{\text{def}}{=} \text{ pour toute } \varphi \in \Phi(q_1 \models \varphi \text{ ssi } q_2 \models \varphi)$$

Lorsque Φ est un singleton $\{\varphi\}$Γnous écrivons plus simplement $\sim_\varphi$ au lieu de $\sim_{\{\varphi\}}$.

3.4 Caractérisation logique de la bisimulation

Les premiers travaux portant sur la comparaison d'une équivalence induite par une logique modale avec une équivalence comportementale se trouvent dans [HM80] et étudient précisément le cas de la bisimulation forte et de l'équivalence induite par HML. Nous en rappelons les résultats.

Théorème 3.4.1 [HM80] *Pour tout $G = (Q, \rightarrow) \in \boldsymbol{G}_{bf}$, et tous états $q_1, q_2 \in Q$,*

$$q_1 \underline{\leftrightarrow} q_2 \text{ ssi } q_1 \sim_{HML} q_2$$

Preuve Nous adoptons la preuve de [HM85] qui est basée sur le théorème suivant:

Théorème 3.4.2 [HM85] *Pour tout ST $G = (Q, \rightarrow) \in \boldsymbol{G}_{bf}$ et états $q, q' \in Q$,*

$$q \equiv_n q' \text{ ssi } q \sim_{HML_n} q' \tag{3.3}$$

Preuve La preuve se fait par induction sur n. Dans cette preuveΓon notera $\mathcal{T}h_n(q)$ au lieu de $\mathcal{T}h_{HML_n}(q)$ pour plus de lisibilité.

- $n = 0$: les seules formules de HML (aux opérations booléennes près) sont $\top$ et $\neg\top$. AinsiΓpour tous $q, q' \in Q$Γ$\mathcal{T}h_0(q) = \mathcal{T}h_0(q)$Γet comme $\equiv_0 = Q \times Q$Γla Propriété (3.3) est vraie pour $n = 0$.

[1] C'est la notion classique de structures "élémentairement équivalentes" [Kei77]

- $q \equiv_{n+1} q'$ implique $\mathcal{T}h_{n+1}(q) = \mathcal{T}h_{n+1}(q')$: on montre par induction structurelle sur les formules de $\varphi \in HML_{n+1}$ que si $q \models \varphi$Γalors $q' \models \varphi$.
  - l'induction est claire si φ est de la forme $\neg\psi$ ou de la forme $\psi_1 \wedge \psi_2$.
  - pour le cas où $\varphi = \langle a\rangle\psi$ avec $\psi \in HML_n$. $q \models \varphi$ signifie qu'il existe $q \xrightarrow{a} p$ tel que $p \models \psi$. Comme $q \equiv_{n+1} q'$Γil existe (d'après la Définition 2.4.1) $q' \xrightarrow{a} p'$ tel que $p \equiv_n p'$. Par hypothèse d'induction sur nΓ$p \sim_{HML_n} p'$; par conséquent $p' \models \psi$Γdonc $q' \models \varphi$.
- $q \not\equiv_{n+1} q'$ implique $\mathcal{T}h_{n+1}(q) \neq \mathcal{T}h_{n+1}(q')$: puisque $q \not\equiv_{n+1} q'$Γon peut supposer sans perdre de généralitéΓqu'il existe une transition $q \xrightarrow{a} p$ telle que pour tout $q' \xrightarrow{a} p'$Γ$p' \not\equiv_n q'$. Puisque G est à b.f. l'ensemble $\{p' | q' \xrightarrow{a} p'\}$ est fini. Notons le $\{p'_1, \ldots, p'_k\}$. Par hypothèse d'induction sur nΓ$\mathcal{T}h_n(p) \neq \mathcal{T}h_n(p'_i)$ pour $i = 1, \ldots k$. Il existe donc $\{\psi_1, \ldots, \psi_k\} \subseteq HML_n$ telles que $p \models \psi_i$ et $p'_i \not\models \psi_i$ pour tout i. Par conséquentΓla formule $\psi \stackrel{\text{def}}{=} \psi_1 \wedge \ldots \wedge \psi_k$ est telle que $p \models \psi$ et $p'_i \not\models \psi$ pour tout i. Donc $q \models \langle a\rangle\psi$ et $q' \not\models \langle a\rangle\psi$Γpour une formule ψ telle que $d(\langle a\rangle\psi) \leq n+1$; ce qui montre que $\mathcal{T}h_{n+1}(q) \neq \mathcal{T}h_{n+1}(q')$.

□

Le Théorème 3.4.2 montre que dans le cas b.f. $\equiv$ et $\sim_{HML}$ coïncident. Le Théorème 3.4.1 découle alors de ce point et du point 2.2 de la Proposition 2.5.1. □

Remarque 3 • *L'hypothèse de branchement fini n'intervient dans la preuve du Théorème 3.4.1 que pour montrer que l'équivalence HML implique $\equiv$. Nous verrons dans le Chapitre 4 les liens entre ces équivalences (et des équivalences induites par des logiques plus riches que HML, e.g. CTL [CE81], CTL* [EH86], L_μ [Koz83]) sans l'hypothèse de branchement fini.*

- *Lorsque l'ensemble A des noms d'actions est fini, on peut aussi montrer que $\sim_{HML}$ et $\equiv$ coïncident dans le cas b.i. Ce résultat est classique, nous renvoyons à [HM85] pour sa preuve.*

Il est clair que la preuve du Théorème 3.4.1 peut être adaptée pour la logique HML_τ et la τ-bisimulation. Ce résultat ne peut se montrer que dans le cas où les graphes considérés sont à b.f. pour la relation $\Rightarrow$. Nous qualifions de graphe

Théorème 3.4.3 [HM80] *Pour tout $G = (Q, \rightarrow)$ tel que $(Q, \Rightarrow)$ soit à b.f. et tous états $q_1, q_2 \in Q$,*

$$q_1 \underline{\leftrightarrow}_\tau q_2 \text{ ssi } q_1 \sim_{HML_\tau} q_2$$

Preuve Elle est similaire à celle du Théorème 3.4.1. □

Le Chapitre suivant est consacré à l'étude des liens entre la bisimulation forte et des logiques (du temps arborescent) sans l'hypothèse de branchement fini.

Chapitre 4

Etude du branchement infini

Nous étudions à présent les liens entre les logiques du temps arborescent et la bisimulation dans le cadre général des systèmes à branchement infini. Notre principale motivation pour ce travail réside dans le fait que peu de travaux portent sur l'utilisation des logiques pour les systèmes à b.i. alors que ces systèmes sont très utiles soit parce qu'ils existent déjà avec leur non-déterminisme infini soit parce que le b.i. apparaît après l'application d'une opération sur un système a b.f. (e.g. le renommage) soit parce que le b.i. découle d'une abstraction que l'on veut faire sur le système (e.g. lorsque l'on abstrait la valeur des message sur un canal) etc.

Dans le cas général du b.i. les Théorèmes d'adéquation 3.4.1 et 3.4.3 sont faux; on conserve seulement la compatibilité de HML pour la bisimulation. Dans [BR83] le cas de HML et d'autres logiques sont étudiés. Les contre-exemples pour prouver que les logiques considérées ne sont pas adéquates pour la bisimulation (mais seulement compatibles) sont souvent compliqués et ad hoc.

Dans ce chapitre nous proposons une méthode simple pour engendrer des contre-exemples eux-mêmes très simples. Nous utilisons une classe particulière de graphes appelés *graphes ordinaux* [Klo88]. Ces graphes sont étiquetés par une seule action et leur relation de transition est transitive et bien fondée. Dans ce cadre les classes d'équivalence modulo bisimulation sont réduites à des singletons.

La classe de *processus ordinaux* contient une infinité de modèles à b.i. dont on peut extraire les contre-exemples qui prouvent que les équivalences induites par les

logiques du temps arborescent sont plus faibles que la bisimulation.

Plus précisément nous traitons le cas des équivalences induites par les logiques HML et CTL [CE81] et CTL^* [EH86] et le μ-calcul [Koz83]. Pour les logiques HML et CTL le résultat est extrait de [SP90]. Pour le μ-calcul nous retrouvons les résultats obtenus par [Her87]. Nous considérons également la classe des logiques modales $\mathcal{C}$–TL de [BT85] qui contient la logique RTL (*Regular Trace Logic*) de Rounds et Gurevich ainsi que le fragment de RTL appelé PDL (pour *Propositional Dynamic Logic*) de [FL79] (la version propositionnelle de la *Dynamic Logic* de [Pra76]). Pour ces logiques nous montrons des résultats qui rejoignent les conclusions de [BR83] et [BT85] à savoir que PDL et la classe des logiques $\mathcal{C}$–TL ne sont pas adéquates pour la bisimulation dans le cas b.i.

Certains des résultats de ce chapitre sont extraits de [Pin91] et [Pin92].

4.1 Les processus ordinaux

Nous supposons que le lecteur connait les ordinaux. Nous renvoyons par exemple à [Ros82].

Les *Processus Ordinaux* de Klop sont des ST étiquetés par une seule action (que nous considérons fixé pour la suite du chapitre). Nous rappelons leur définition ainsi que leurs principales propriétés.

Définition 4.1.1 (Graphes et Processus Ordinaux) *A tout ordinal λ, on associe le* graphe ordinal $G_\lambda \stackrel{def}{=} (\lambda+1, >)$. *Autrement dit, la relation de transition de G_λ est définie par : pour tous $\alpha, \beta \leq \lambda$ (i.e. $\alpha, \beta \in \lambda+1$), $\alpha \rightarrow \beta$ ssi $\alpha > \beta$.*
On note ***GO*** *la classe des graphes ordinaux.*
Le processus ordinal *associé à λ est le ST $S_\lambda \stackrel{def}{=} (G_\lambda, \lambda)$. Nous notons* ***PO*** *la classe des processus ordinaux.*

Exemple 7 *A l'ordinal 3 correspond le ST $S_3 = (\{0,1,2,3\}, \leq, 3)$ dessiné en Figure 4.1.*

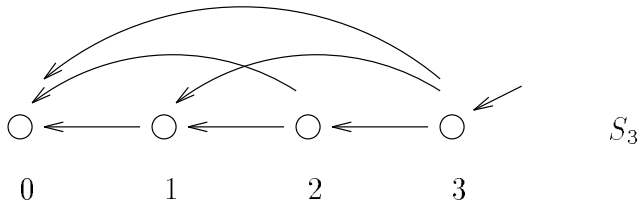


Figure 4.1: Le processus ordinal S_3

A l'ordinal ω (premier ordinal infini) correspond le ST à b.i. $S_\omega \stackrel{def}{=} (\{0,1,\ldots\} \cup \{\omega\}, \leq, \omega)$ *dont la représentation graphique est donnée en Figure 4.2.*

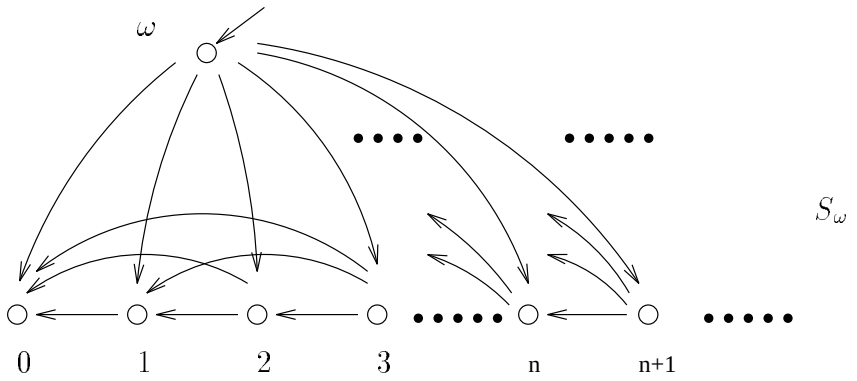


Figure 4.2: Le processus ordinal $\S_\omega$

Il faut remarquer que dès que $\lambda \geq \omega$Γle graphe G_λ est à b.i. De plusΓla relation de transition de G_λ étant l'ordre $<$ sur les ordinaux $\alpha \leq \lambda$Γon en déduit :

- $\rightarrow$ est transitiveΓet
- il n'existe aucun chemin infini dans G_λ (puisque $<$ est bien fondée).

Nous rappelons des ordinaux particuliers qui apparaissent dans nos résultats.

- $\omega * 2$ dénote l'ordinal $\omega + \omega$Γet plus généralement $\omega * n$ dénote $\omega + \ldots + \omega$Γn fois. Ces ordinaux sont tous dénombrables.
- ω^2 est $\omega + \ldots$Γω foisΓi.e. $\omega^2 = \bigcup_{n \in \omega} \omega * n$; il est dénombrable.
- ω_1 est le plus petit ordinal non dénombrableΓi.e. $\omega_1 = \bigcup\{\alpha | \alpha$ dénombrable$\}$

4.2 Notion de pouvoir de distinction

A toute relation d'équivalence $\sim$ sur la classe **PO** des processus ordinauxΓon associe un ordinal appelé *pouvoir de distinction* de $\sim$. D'une certaine façonΓcet ordinal mesure l'équivalence $\sim$.

Définition 4.2.1 (Pouvoir de distinction) *Soit $\sim$ une relation d'équivalence sur* ***PO****. Le* pouvoir de distinction *(d.p.)*[1] *de $\sim$, noté $dp(\sim)$, est le plus petit ordinal γ tel que pour tous ordinaux $\gamma \leq \alpha, \beta$, $\alpha \sim \beta$.*

[1] *Pour "distinguishing power".*

Remarque 4 *Dans la Définition 4.2.1, α et β sont vus comme les processus ordinaux qu'ils induisent canoniquement, c'est à dire S_α et S_β.*

Dans le cas où l'équivalence que nous considérons est celle induite par une logique LΓnous notons $dp(L)$ au lieu de $dp(\sim_L)$Γet nous disons plus simplement le "pouvoir de distinction de L" au lieu du "pouvoir de distinction de l'équivalence induite par L".

Remarque 5 *Le terme "pouvoir de distinction" est un abus de langage, car en réalité nous mesurons seulement ce qu'une équivalence identifie dans la classe* ***PO*** *et non dans la classe* ***ST*** *toute entière.*

Par exempleΓle d.p. de la relation triviale sur **PO** (i.e. celle qui identifie tous les processus ordinaux) est 0.

Il est important de noter que le d.p. d'une équivalence n'existe pas toujours. C'est le cas par exemple de la bisimulation forte dont les classes d'équivalences sur **PO** sont des singletons :

Proposition 4.2.1 [Klo88] [SP90] *Pour tous ordinaux α_0, α_1,*

$$S_{\alpha_0} \underline{\leftrightarrow} S_{\alpha_1} \text{ ssi } \alpha_0 = \alpha_1$$

Preuve Soient deux ordinaux distincts $\alpha_0 > \alpha_1$. Plaçons nous dans un graphe ordinal G_λ contenant α_0 et α_1Γi.e. $\alpha_1 < \alpha_0 \leq \lambda$. AinsiΓpour tous $\alpha, \beta \leq \lambda$Γon a S_α et S_β et on a $\alpha \underline{\leftrightarrow}_{G_\lambda} \beta$ ssi $S_\alpha \underline{\leftrightarrow} S_\beta$.

Supposons que $\alpha_0 \underline{\leftrightarrow} \alpha_1$; la transition $\alpha_0 \rightarrow \alpha_1$ (cette transition existe puisque $\alpha_0 > \alpha_1$) est imitée par une transition $\alpha_1 \rightarrow \alpha_2$ telle que $\alpha_1 \underline{\leftrightarrow} \alpha_2$. En répétant ce raisonnementΓon peut construire pas à pas un chemin infini $\alpha_0 \rightarrow \alpha_1 \rightarrow \alpha_2 \rightarrow \ldots$Γ ce qui est absurde puisque $\rightarrow$ est bien fondée. □

Corollaire 4.2.2 *Soit $\sim$ une relation d'équivalence sur* ***PO****. Si $dp(\sim)$ existe, alors l'équivalence $\sim$ n'est pas plus fine que la bisimulation.*

Preuve Soit $\gamma = dp(\sim)$. Par définition du d.p.Γon a $S_\gamma \sim S_{\gamma+1}$Γmais d'après la Proposition 4.2.1Γ$S_\gamma \not\underline{\leftrightarrow} S_{\gamma+1}$. □

L'introduction de la notion de pouvoir de distinction permet parfois de montrer que deux équivalences sont distinctes : Prenons deux équivalences $\sim$ et $\sim'$ de d.p. respectifs γ et γ'. Si $\gamma < \gamma'$Γalors $\gamma \sim \gamma'$Γmais $\gamma \not\sim' \gamma'$.

Toujours dans [Klo88] (et [SP90])Γil est montré que

Proposition 4.2.3 *$dp(\equiv) = \omega$ et $dp(\approx) = \omega^2$.*

Preuve Nous renvoyons par exemple à [SP90]. □

Corollaire 4.2.4 *Les équivalences $\underline{\leftrightarrow}$, $\approx$ et $\equiv$ qui coïncident dans la classe des graphes à b.f. sont en général distinctes.*

Concernant le pouvoir de distinction d'une logique LΓil est possible de le reformuler en termes du pouvoir de distinction des formules de la logiqueΓque nous appelons la *hauteur*.

Définition 4.2.2 (Hauteur d'une formule) *Soit L une logique d'états. Nous appelons* hauteur *de la formule f, notée $h(f)$, le d.p. de l'équivalence $\sim_f$, autrement dit le plus petit ordinal γ tel que pour tous $\gamma \leq \alpha \leq \beta$, $\alpha \sim_f \beta$.*

Dans ce casΓon voit clairement que

$$dp(L) = \bigcup_{f \in L} h(f)$$

Dans la Section 4.3.3Γnous étudions en détail la hauteur des formules des logiques HMLΓCTLΓCTL^*Γdu μ-calcul propositionnelΓetc.

4.3 Pouvoir de distinction des logiques d'états

Les logiques temporelles et modales utilisées pour raisonner sur les systèmes réactifs peuvent être classifiées selon différents axes. Nous renvoyons à [Eme90] pour une étude générale des logiques pour le parallélisme.

Lorsqu'on définit une logique temporelleΓil existe deux façons de représenter la structure du temps. La première approche est d'attribuer au temps une structure linéaire. Cela signifie qu'à chaque instantΓil n'existe qu'un seul futur possible. Les logiques basées sur ce principe sont appelées *logiques du temps linéaire*. La seconde approche est d'attribuer au temps une structure arborescenteΓce qui signifie qu'à chaque instantΓil existe plusieurs futurs possibles. Les logiques basées sur ce principe sont appelées *logiques du temps arborescent*.

Dans l'utilisation que l'on fait de ces logiquesΓla nature du temps est induite par les équivalences sémantiques que l'on adopte. Si cette équivalence est du temps arborescent (resp. linéaire) on choisira d'utiliser une logique du temps arborescent (resp. linéaire). Dans cette thèseΓnotre attention est centrée sur la bisimulationΓ par conséquentΓnous ne considérerons que des logiques du temps arborescent. Les logiques que nous utilisons sont toutes propositionnelles.

Nous étudions d'abord le cas des logiques HMLΓCTLΓCTL^* et du μ-calcul qui caractérisent toutes la bisimulation sous l'hypothèse de b.f. comme nous le rappelons. Nous montrons que le d.p. de HMLΓCTLΓCTL^* et du μ-calcul est ωΓ en utilisant la preuve que les hauteurs des formules de ces logiques sont des ordinaux finis. Il est alors immédiat que dans le cas généralΓces logiques ne sont pas adéquates pour la bisimulation.

Une démarche analogue est adoptée dans la Section 4.3.5 pour montrer que le d.p. des logiques $\mathcal{C}$–TL est au plus égal à ω^2. Nous en déduirons que ces logiques ne sont pas adéquates pour la bisimulation.

4.3.1 Les logiques CTL et CTL^*

La logique CTL (Computation Tree Logic) de [CE81] et [CES83] est très proche des logiques proposées par [EC80] et [QS83]. Elle contient les opérateurs de base $\forall$ ("pour tous les futurs") ou $\exists$ ("il existe un futur") qui permettent de quantifier sur les futurs possibles et les modalités temporelles X ("à l'instant suivant") ou U ("jusqu'à"). Pour la logique CTL la manière de combiner les opérateurs est restreinte : les modalités X et U sont toujours immédiatement précédées d'un quantificateur $\forall$ ou $\exists$. Cette restriction syntaxique limite considérablement le pouvoir d'expressivité de CTL; en particulier on ne peut l'utiliser pour exprimer certaines propriétés d'équité [EH86]. Le relâchement de cette contrainte syntaxique (i.e. les quantificateurs sur les exécutions sont suivis de formules arbitraires) permet de mélanger la logique CTL et la logique du temps linéaire PTL. On obtient la logique CTL^* de [EH86]. CTL constitue néanmoins une logique relativement expressive puisqu'elle est capable d'exprimer les propriétés les plus importantes de blocage (ou d'absence de blocage) de vivacité etc (voir par exemple [CE81]).

Formellement la syntaxe de CTL^* est donnée par la grammaire suivante :

$$(CTL^* \ni)\quad f, g ::= \top \mid f \wedge g \mid \neg f \mid X\, f \mid f\, U\, g \mid \forall f$$

Notations 4 *Nous introduisons les abréviations classiques suivantes :*

- *Soit $\exists$ l'opérateur défini par $\exists f \stackrel{def}{=} \neg\forall\neg f$.*
- *On note $F\, f \stackrel{def}{=} \top\, U\, f$ et $G\, f \stackrel{def}{=} \neg\, F\, \neg f$.*

Définition 4.3.1 *Pour tout sous-ensemble de formules $F \subseteq CTL^*$ nous définissons $SF(F)$ le plus petit ensemble de formules de CTL^* qui contient F et qui soit fermé par sous-formules.*
Nous définissons $BC(F)$ comme le plus petit ensemble de formules (modulo équivalence sémantique, au sens de (a) page 54) qui soit fermé par sous-formules et combinaison booléennes.
Lorsque l'ensemble F est un singleton $\{f\}$, nous notons $SF(f)$ (resp. $BC(f)$) au lieu de $SF(\{f\})$ (resp. $BC(\{f\})$).

On peut remarquer que $SF(F) \subseteq BC(F)$ et que lorsque F est fini les ensembles $SF(F)$ et $BC(F)$ le sont aussi.

Exemple 8 *Prenons l'ensemble de formules* $F_0 = \{\top, X\,X\,\top\}$*, alors* $SF(F_0) = \{\top, X\,\top, X\,X\,\top\}$ *et* $BC(F_0) = \{\top, X\,\top, X\,X\,\top, \neg\top, \neg X\,\top, \neg X\,X\,\top, X\,\top \wedge \neg X\,X\,\top, \neg(X\,\top \wedge \neg X\,X\,\top)\}$.

La formule $X\,\top \wedge X\,X\,\top$ *ne figure pas dans l'ensemble* $BC(F_0)$ *puisqu'elle est équivalente à* $X\,X\,\top$.

Les formules de CTL^* sont interprétées sur les exécutions d'un graphe $G = (Q, \rightarrow)$. La sémantique formelle de CTL^* est donnée par la *relation de satisfaction* $\models \subseteq Ex(G) \times CTL^*$ définie par induction sur les formules. La phrase $\pi \models f$ où $\pi \in Ex(G)$ et $f \in CTL^*$ se lit "f est vraie dans l'exécution π" ou encore "l'exécution π satisfait la propriété f". Nous définissons $\pi \models f$ par induction sur la structure des formules de CTL^* :

Pour tous $\pi \in Ex(G)$ et $f, g \in CTL^*$

- $\pi \models \top$ toujours
- $\pi \models f \wedge g$ ssi $\pi \models f$ et $\pi \models g$
- $\pi \models \neg f$ ssi $\pi \not\models f$
- $\pi \models X\,f$ ssi $\pi^1 \models f$ (où π^1 est le premier préfixe de π)
- $\pi \models f\,U\,g$ ssi il existe $n \in N$ tel que $\pi^n \models g$ et $\pi^i \models f$ pour tout $i < n$
- $\pi \models \forall f$ ssi pour toute exécution π' de G partant de l'état $\pi(0)$ $\pi' \models f$

Intuitivement, si $\pi \models \forall f$ alors toutes les exécutions partant de $\pi(0)$ satisfont f; $\pi \models \exists f$ si il existe une exécution partant de l'état $\pi(0)$ qui satisfait f. Une exécution π satisfait $F\,f$ si il existe un moment le long de π où la propriété f est satisfaite; une exécution π satisfait $G\,f$ si à tout moment le long de π la propriété f est vraie. Par exemple, la propriété $G\,F\,f$ exprime que la formule f est vraie une infinité de fois le long de l'exécution. De telles formules permettent d'exprimer des propriétés d'équité (faibles, fortes, etc.).

Si une exécution π satisfait une formule de la forme $\forall f$ ou $\exists f$, il est clair que satisfaire une telle propriété ne dépend que de $\pi(0)$, de l'état de départ de l'exécution π, et de G. Les formules de la forme $\forall f$ et $\exists f$ sont appelées des formules d'états.

On peut alors facilement interpréter CTL^* sur les états d'un graphe par

$$q \models f \stackrel{\text{def}}{-} \text{pour tout } \pi \in Ex_G(q), \pi \models f$$

Soit CTL le fragment syntaxique de CTL^* constitué des formules d'états dans lesquelles tous les opérateurs X et U sont immédiatement précédés d'un opérateur $\forall$ ou $\exists$. Par exemple, si $f, g \in CTL$, $\forall f\,U\,g \in CTL$ mais $\exists X\,X\,(f \wedge g) \notin CTL$.

Habituellement pour la logique CTL on note A et E les opérateurs $\forall$ et $\exists$. Nous utilisons ces notations pour définir sa syntaxe.

$$(CTL\ \ni)\quad f,g ::= \top \mid f \wedge g \mid \neg f \mid E\,X\,f \mid E\,f\,U\,g \mid A\,f\,U\,g$$

Intuitivement les formules de CTL peuvent être comprises de la façon suivante :

- $E\,X\,f$ se lit "il existe un état successeur qui satisfait f". Par exemple la formule $EX\top$ caractérise les états qui possèdent des successeurs dans le graphe et donc $\neg EX\top$ est vraie de tous les états terminaux du graphe.
- $E\,f\,U\,g$ signifie qu'"il existe un chemin le long duquel f est vraie jusqu'à ce que g soit vraie". La formule $E\top U\,f$ exprime qu'il est possible d'atteindre (en zéro ou plusieurs étapes) un état qui satisfait f.
- $A\,f\,U\,g$ se lit "pour tous les chemins la formule f est vraie le long de ce chemin jusqu'à ce que g le soit". Par exemple $A\top U\neg EX\top$ est vraie d'un état à partir duquel on ne peut atteindre que des états terminaux ; autrement dit tous les chemins qui partent de cet état sont finis. Ainsi $\neg A\top U\neg EX\top$ caractérise les états du graphes qui appartiennent à un chemin infini.

La sémantique formelle de CTL peut être définie sur les états d'un graphe G par pour tous $q, q_0 \in Q_G$ et $f, g \in CTL$

- $q \models \top, q \models f \wedge g, q \models \neg f$ ont des définitions évidentes
- $q \models EX\,f$ ssi il existe une transition $q \rightarrow_G q'$ telle que $q' \models f$
- $q_0 \models E\,f\,U\,g$ ssi il existe une exécution partant de q_0 $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \ldots$ et un $n \in N$ tel que $q_n \models g$ et $q_i \models f$ pour tout $i < n$
- $q_0 \models A\,f\,U\,g$ ssi pour toute exécution partant de q_0 $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \ldots$ il existe un $n \in N$ tel que $q_n \models g$ et $q_i \models f$ pour tout $i < n$.

Pour les graphes étiquetés sur un alphabet d'actions de cardinal 1 comme par exemple les graphes ordinaux on peut voir la logique HML de [HM85] du Chapitre 3 comme le fragment de CTL formé des opérateurs $\top$ $\wedge$ $\neg$ et EX. En effet si G est un graphe étiqueté sur l'alphabet $A = \{a\}$ la modalité $\langle a \rangle$ de HML est sémantiquement équivalente à la modalité $E\,X$ de CTL.

Habituellement on compare des logiques par leur expressivité ou à travers les équivalences qu'elles induisent.

La notation $L \leq L'$ signifie que la logique L n'est *pas plus expressive* que la logique L ce qui se définit formellement par :

(a) pour toute formule $f \in L$ il existe une formule $f' \in L'$ sémantiquement équivalente à f i.e. telle que pour tout $G \in \mathbf{G}$ et tout état $q \in Q_G$ $q \models f$ ssi $q \models f'$.

Par inclusion syntaxique, il est clair que $HML \leq CTL \leq CTL^*$.

Nous notons $L < L'$ si $L \leq L'$ et $L \not\leq L'$, et nous disons que L *est strictement moins expressive que* L'. En particulier, $HML < CTL < CTL^*$. Ce résultat est classique (voir [Eme90] pour une classification des logiques temporelles selon leur expressivité). Par exemple, la logique HML ne permet pas d'exprimer l'existence de chemins infinis dans un graphe, ce que CTL peut faire. On peut trouver dans [Eme90], un exemple d'une formule de CTL^* qui n'est pas exprimable en CTL.

Il est clair que

$$L \leq L' \text{ implique } \sim_L \subseteq \sim_{L'} \tag{4.4}$$

Dans [BCG87], il est montré que si deux états sont bisimilaires, alors ils satisfont les mêmes formules de CTL^*. De [BCG87] et de l'équation (4.4) on en déduit

$$\underline{\leftrightarrow} \subseteq \sim_{CTL^*} \subseteq \sim_{CTL} \subseteq \sim_{HML} \tag{4.5}$$

Les inclusions de l'équation (4.5) se réduisent à des égalités lorsque l'on se restreint à la classe $\mathbf{G}_{bf}$ des graphes à b.f. : nous savons, d'après le Théorème d'adéquation 3.4.1 de Hennessy-Milner, que dans ce cadre, les équivalences $\sim_{HML}$ et $\underline{\leftrightarrow}$ coïncident. On en déduit : pour tout graphe $G \in \mathbf{G}_{bf}$, et tous états $q, q' \in Q_G$,

$$q \underline{\leftrightarrow} q' \text{ ssi } q \sim_{CTL^*} q' \text{ ssi } q \sim_{CTL} q' \text{ ssi } q \sim_{HML} q'$$

Les égalités ci-dessus montre que la réciproque de (4.4) est fausse en général.

4.3.2 Le μ-calcul

Le μ-calcul propositionnel, que nous notons L_μ dans cette thèse, a été introduit par [SB69], et étudié entre autre par [Pra81] et [Koz83]. Dans cette logique, les modalités ne sont pas définies à l'avance, mais peuvent être exprimées par des opérateurs de point-fixe. Le μ-calcul existe sous les deux formes arborescentes et linéaires. En ce qui nous concerne, c'est sa version arborescente qui nous intéresse.

Nous supposons donné un ensemble $\mathcal{V} = \{Y, Z, \ldots\}$ de variables. La syntaxe de la logique L_μ est donnée par la grammaire suivante :

$$(L_\mu \ni) \quad f, g ::= \top \mid f \wedge g \mid \neg f \mid \Diamond f \mid Y \mid \mu Y.f(Y)$$

où $Y \in \mathcal{V}$ et $\mu Y.f(Y)$ désigne la plus petite solution de $Y \equiv f(Y)$. En considérant le symbole μ comme un quantificateur, nous utilisons, comme pour le calcul des prédicats du premier ordre, la terminologie de variable *libre* et de variable *liée*. Soit $\hat{Y} = (Y_1, Y_2, \ldots, Y_n) \in \mathcal{V}^n$. La notation $f(\hat{Y})$ signifie que les variables ayant des occurrences libres dans f appartiennent à $\hat{Y}$. Une formule de L_μ est dite *fermée* si

elle ne contient pas d'occurrence d'une variable libre.

Nous n'autorisons pour L_μ que les formules f pour lesquelles toute occurrence de la variable Y est :

- *gardée* c-à-d. dans la portée d'au moins un opérateur $\Diamond$ et
- *positive* c-à-d. dans la portée d'un nombre pair d'opérateurs $\neg$.

Les formules de L_μ s'interprètent sur les états d'un graphe $G = (Q, \rightarrow)$. Les opérateurs $\top$ $\neg$ et $\wedge$ ont leur sens habituel. L'opérateur $\Diamond$ à le même sens que l'opérateur EX de CTL. Toute formule fermée de L_μ peut être interprétée comme un élément du treillis complet $(2^Q, \subseteq)$ formé des parties de Q et ordonné par l'inclusion : par exemple la formule $\top$ correspond à 2^Q. Les formules non fermées sont alors vues comme des fonctions totales sur $(2^Q, \subseteq)$: soit $\hat{Y} = (Y_1, Y_2, \ldots, Y_n) \in \mathcal{V}^n$. La formule $f(\hat{Y})$ est interprétée comme une fonction de $(2^Q)^n \rightarrow 2^Q$ et la lettre μ désigne l'opérateur de plus petit point fixe. Ainsi la formule $\mu Y.f(Y)$ représente l'élément de $(2^Q, \subseteq)$ qui est le plus petit point-fixe de la fonction f. L'existence de $\mu Y.f(Y)$ est garantie par la restriction syntaxique d'occurrences positives des variables libres : en effet dans ce cas $f(Y)$ est une fonction monotone en Y et d'après le Théorème de Knaster-Tarski [Tar55] le plus petit point-fixe de f existe.

Les sous-formules fermées d'une formule fermée f de L_μ se définissent à partir de la fermeture de Fischer-Ladner de f [FL79].

Définition 4.3.2 (Fermeture de Fischer-Ladner) *Soit $f \in L_\mu$, une formule fermée. La* fermeture de Fischer-Ladner *de f, est le plus petit ensemble, notée $FL(f)$, tel que*

$$FL(\top) = \{\top\} \qquad FL(\neg f) = \{\neg f\} \cup FL(f)$$
$$FL(f \wedge g) = \{f \wedge g\} \cup FL(f) \cup FL(g) \qquad FL(\Diamond f) = \{\Diamond f\} \cup FL(f)$$
$$FL(\mu Y.f(Y)) = \{\mu Y.f(Y)\} \cup FL(f(\mu Y.f(Y)))$$

Par exemple, $FL(\mu Y.h \vee g \wedge \Diamond Y) = \{\mu Y.h \vee g \wedge \Diamond Y, h, g, \Diamond(\mu Y.h \vee g \wedge \Diamond Y)\}$.

Pour permettre des raisonnement par induction sur les formules de L_μ nous définissons un ordre bien fondé : $\prec$ est le plus petit ordre tel que

$$f \prec \neg f, \quad f \prec f \wedge g, \quad f \prec g \wedge f, \quad f(\mu Y.f(Y)) \prec \mu Y.f(Y)$$

Puisque les formules de L_μ sont gardées le fait que nous ne demandions pas que "$f \prec \Diamond f$" nous garantit que $\prec$ est bien fondé : en effet il n'est pas possible de déplier indéfiniment une formule (gardée) (i.e. d'utiliser infiniment souvent que $f(\mu Y.f(Y)) \prec f$) sans jamais rencontrer une occurrence de la forme $\Diamond g$.

La sémantique formelle de L_μ est donnée pour une valuation $v : \mathcal{V} \rightarrow 2^Q$ par la relation de satisfaction $\models_v \subseteq Q \times L_\mu$ suivante : pour tous $q \in Q$ et $f, g \in L_\mu$

- $q \models_v \top, q \models_v f \wedge g, q \models_v \neg f$ ont des définitions évidentesΓ
- $q \models_v Y$ ssi $q \in v(Y)\Gamma$
- $q \models_v \Diamond f$ ssi il existe une transition $q \rightarrow_G q'$ telle que $q' \models_v f\Gamma$
- $q \models_v \mu Y.f(Y)$ ssi $q \models_{v'} f$ pour toute valuation v' telle que

  (1) $v'(Z) = v(Z)$ si $Z \neq Y\Gamma$et

  (2) $v'(Y) \subseteq f(v'(Y))$.

La logique L_μ est très expressive. Elle contient trivialement CTL en remarquant que

$$E\,f\,U\,g = \mu Y.(g \vee f \wedge \Diamond Y) \qquad A\,f\,U\,g = \mu Y.(g \vee f \wedge \Diamond\top \wedge \neg\Diamond\neg Y)$$

[Dam90] propose une traduction de la logique CTL^* dans $L_\mu\Gamma$avec une complexité exponentielle.

L_μ est strictement plus expressive que CTL^*. Notre preuve est une adaptation de celle de [Koz83] où il est montré que L_μ est strictement plus expressive que la logique ΔPDL [Str81] (un fragment syntaxique de la "Propositional Dynamic Logic" de [FL79]). Considérons le graphe d'états infini de la Figure 4.3.

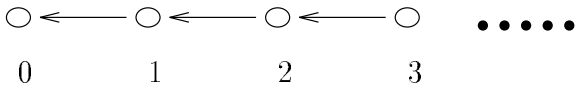


Figure 4.3:

On montre facilement que :

Lemme 4.3.1 *La formule $\mu Y.(\neg\Diamond\top \vee \Diamond\Diamond Y)$ caractérise les états de G de la forme $2 * n$ où $n \in N$.*

Nous montrons que cette formule n'a pas d'équivalent en CTL^* en utilisant :

Lemme 4.3.2 *Pour toute formule $f \in CTL^*$, l'ensemble des états de G qui satisfont f est soit fini, soit cofini (i.e. de complémentaire fini).*

Preuve La preuve se fait par induction sur la structure de la formule. Remarquons que dans $G\Gamma$une formule $\forall f$ est équivalente à $f\Gamma$car de chaque état ne part qu'une seule exécution. Nous notons π_n l'unique exécution partant de l'état n. Dans cette preuveΓ nous posons $Sat(f) \stackrel{\text{def}}{=} \{n \in Q_G | \pi_n \models f\}$. Nous disons d'une formule f qu'elle est *finie* si $Sat(f)$ est finiΓet qu'elle est *cofinie* si $Q_G \setminus Sat(f)$ est fini. Nous traitons à titre d'exemple le cas des formules $f\,U\,g$. Pour celaΓil suffit de considérer trois cas.

1. si g est cofinieΓcomme $Sat(g) \subseteq Sat(f\,U\,g)$Γ$f\,U\,g$ est cofinie.
2. si g et f sont finiesΓcomme $Sat(f\,U\,g) \subseteq Sat(f) \cup Sat(g)$Γ$f\,U\,g$ est finie.
3. enfinΓsi g est finie et f est cofinieΓnous définissons k le plus petit entier tel que pour tout $n \geq k$Γ$\pi_n \models f \wedge \neg g$. Si $k = 0$Γalors $Sat(f\,U\,g) = \emptyset$ et on a terminé. Sinon $(k \geq 1)$Γil est facile de voir que que selon que $\pi_{k-1} \models f\,U\,g$ ou nonΓla formule $f\,U\,g$ est finie ou cofinie.

Les autres cas sont simplesΓnous omettons la preuve. □

Corollaire 4.3.3 $CTL^* < L_\mu$.

On peut de plus montrer que l'équivalence $\sim_{L_\mu}$ contient $\underline{\leftrightarrow}$. On en déduira que dans le cas de graphes à b.f. L_μ caractérise $\underline{\leftrightarrow}$ puisque c'est déjà le cas pour la logique CTL^*Γmoins expressive que L_μ.

Proposition 4.3.4 $\underline{\leftrightarrow} \subseteq \sim_{L_\mu}$

Preuve Nous utilisons l'idée de [HS85] qui introduisent une version infinitaire de la logique HML. Nous notons HML^∞ l'extension de la logique HML dans laquelle on autorise des conjonctions et disjonctions arbitraires (i.e. éventuellement non dénombrables). Dans [HS85]Γil est montré que pour tout graphe G et tous états q, q' de G

$$q \underline{\leftrightarrow} q' \text{ ssi } q \sim_{HML^\infty} q'$$

Il suffit donc de montrer que toute formule de L_μ est équivalente à une formule de HML^∞. Seul le cas des formules de L_μ de la forme $\mu Y.f(Y)$ est à traiter en détail. Pour une telle formuleΓsi f est interprétée comme la fonction de 2^Q vers 2^Q qui à Y fait correspondre $f(Y)$Γle plus petit point fixe de f est contenu dans $\bigcup_\alpha f^\alpha(\emptyset)$Γ où f^β est défini par :

- $f^0(\emptyset) = \emptyset$Γ
- $f^{\beta+1}(\emptyset) = f(f^\beta(\emptyset))$Γ
- $f^\lambda(\emptyset) = \bigcup_{\beta<\lambda} f^\beta(\emptyset)$

De plusΓpour chaque graphe GΓil existe un ordinal α tel que $\mu Y.f(Y) = f^\alpha(\emptyset)$. Par conséquentΓpour tout état q de GΓ$q \models \mu Y.f(Y)$ ssi $q \models f^\alpha(\bot)$Γoù $\bot \stackrel{\text{def}}{=} \neg\top$. Autrement ditΓla formule $\mu Y.f(Y)$ est équivalente à la conjonction infinie $f_1 \equiv \bigcup_\alpha f^\alpha(\bot)$. La formule f_1 infinitaire contient un opérateur μ de moins que la formule $\mu Y.f(Y)$. En itérant ce procédéΓon aboutit à une formule f_n ne contenant aucun opérateur μΓc-à-d. à une formule $f_n \in HML^\infty$. □

4.3.3 Hauteurs des formules

Pour chacune des logiques HML, CTL, CTL^* et L_μ, nous étudions une borne pour la hauteur des formules. Les Propositions 4.3.5 et 4.3.12 nous montrent que les hauteurs de formules sont des ordinaux finis.

Considérons dans un premier temps les logiques HML et CTL. Nous généralisons à la logique CTL la notion de profondeur d'une formule telle que proposée pour HML au Chapitre 3 (i.e. $d(f)$ est égale au nombre maximum d'opérateurs EX emboîtés dans f). Formellement, nous définissons $d : CTL \to N$ par induction sur la structure des formules de CTL :

$$\begin{array}{ll} d(\top) = 0 & d(EX\,f) = d(f) + 1 \\ d(f \wedge g) = max(d(f), d(g)) & d(E\,f\,U\,g) = max(d(f), d(g)) \\ d(\neg f) = d(f) & d(A\,f\,U\,g) = max(d(f), d(g)) \end{array}$$

Proposition 4.3.5 [SP90] *Pour toute $f \in CTL$, $h(f) \leq d(f)$.*

Preuve Par définition de $h(f)$, il faut montrer que pour tous ordinaux $\alpha, \beta \geq d(f)$, $\alpha \sim_f \beta$. Pour cela, il suffit de montrer que l'ordinal $d(f)$ est $\sim_f$-équivalent à tout $\alpha \geq d(f)$. La preuve se fait par induction sur la structure de f :

- Les cas $\top$, $\neg f$ et $f \wedge g$ sont évidents.
- Supposons que f soit de la forme $EX\,g$. Soit $n = d(g)$ et $\alpha \geq d(f) = n + 1$.
  Si $n + 1 \models EX\,g$, il existe $n + 1 \to \beta$ tel que $\beta \models g$ (où $\beta < n + 1 \leq \alpha$). Comme $\beta < \alpha$, on a $\alpha \to \beta$, ce qui prouve que $\alpha \models EX\,g$.
  Sinon, $n + 1 \not\models EX\,g$, c-à-d. qu'aucun des successeurs de $n + 1$ ne satisfait g. En particulier $n \not\models g$. Par hypothèse d'induction, pour tout $\beta \geq n$, $\beta \not\models g$ car $\beta \geq n = d(g)$. Par conséquent, $\alpha \not\models EX\,g$.
- Supposons f de la forme $E\,g\,U\,h$. Soit $n = d(f)$ et $\alpha \geq n$.
  Si $n \models f$, alors ou bien $n \models g$ ou bien $n \models h$.
  - si $n \models h$, par hypothèse d'induction $\alpha \models h$, donc $\alpha \models f$.
  - sinon $n \models g$ et il existe $k < n$ tel que $k \models h$ (puisqu'il existe $n \to k_1 \to \ldots \to k_n$ tels que $k_n \models h$ et $k_i \models g$ pour tout $1 \leq i < n$, et que $\to$ est transitive). Comme $\alpha \geq n \geq d(g)$ et que $n \models g$, par hypothèse d'induction $\alpha \models g$. De plus, comme $\alpha \to k$, on obtient $\alpha \models f$.
- Sinon $n \not\models f$, donc $n \not\models h$ qui implique, par hypothèse d'induction que $\alpha \not\models h$. Par conséquent, $\alpha \models f$ implique que $\alpha \models g$ et qu'il existe $\alpha \to \beta$ tel que $\beta \models h$. Comme $\beta < n$ (sinon on aurait $n \models h$), nous avons $n \to \beta$, ce qui contredit $n \not\models f$.
- Un raisonnement similaire permet de traiter les formules f de la forme $A\,g\,U\,h$.

□

Il est clair que $d(f)$ est la meilleur borne que l'on puisse atteindre car pour les formules de la forme $EX^n\top$ qui contient n opérateurs EX emboîtés. Notons que ces formules appartiennent à la logique HML.

Avant de traiter la hauteur des formules de CTL^* et de L_μ nous fixons quelques définitions et notations.

Définition 4.3.3 *Un graphe $G = (Q, \rightarrow)$ est* transitif *si $\rightarrow$ est transitive.*
Dans un tel graphe, si q_1, q_2 sont deux états tels que $q_1 \rightarrow q_2$, nous notons $[q_1, q_2]$ l'ensemble $\{q \in Q | q_1 \rightarrow^* q \rightarrow^* q_2\}$, où $\rightarrow^*$ est la fermeture réflexive de $\rightarrow$. Un ensemble de la forme $[q_1, q_2]$ est appelé un intervalle.

Les logiques CTL^* et L_μ vérifient le Théorème de l'Intervalle qui exprime que les propriétés temporelles sont préservées dans les intervalles d'un graphe transitif. Plus précisément, pour toute formule f, pour tout état q_1 d'un graphe transitif, si il existe $q_1 \rightarrow q_2$ tels que $q_1 \sim_F q_2$ pour un ensemble F de sous-formules de f, fermé par sous-formules, alors tout état intermédiaire $q \in [q_1, q_2]$ vérifie $q_1 \sim_F q$. Nous le prouvons d'abord pour la logique L_μ :

Théorème 4.3.6 (Théorème de l'Intervalle pour L_μ) *Pour tout graphe transitif $G = (Q, \rightarrow)$, toute formule $f \in L_\mu$, tous états $q_1, q_2 \in Q$,*

$$q_1 \sim_{FL(f)} q_2 \text{ implique } q_1 \sim_{FL(f)} q, \text{ pour tout } q \in [q_1, q_2]$$

Preuve Il suffit de montrer que pour toute formule $g \in FL(f)$, $q_1 \sim_{FL(g)} q_2$ implique pour tout $q \in [q_1, q_2]$, $q_1 \sim_g q$. Soit $q \in [q_1, q_2]$. Nous proposons une preuve par induction sur la formule g (pour l'ordre $\prec$) :

- Les cas $\top$, $f \wedge g$ et $\neg g$ sont évidents.
- $\mu Y.g(Y)$: $q_1 \sim_{FL(\mu Y.g(Y))} q_2$ implique $q_1 \sim_{FL(g(\mu Y.g(Y)))} q_2$, qui par hypothèse d'induction, entraîne $q_1 \sim_{g(\mu Y.g(Y))} q$. Comme les formules $g(\mu Y.g(Y))$ et $\mu Y.g(Y)$ sont équivalentes, on en déduit que $q_1 \sim_{\mu Y.g(Y)} q$.
- $\Diamond g$: l'hypothèse d'un graphe transitif s'utilise à cet endroit de la preuve.
  Si $q_2 \models \Diamond g$, alors il existe $q_2 \rightarrow q_2'$ tel que $q_2' \models g$. Comme $q \in [q_1, q_2]$, et comme $\rightarrow$ est transitive, on a $q \rightarrow q_2'$ et donc $q \models \Diamond g$.
  Sinon, $q_2 \not\models \Diamond g$, et donc $q_1 \not\models \Diamond g$. Puisque tous les successeurs de q dans G sont des successeurs de q_1, on obtient $q \not\models g$.

□

Le Théorème de l'Intervalle pour la logique CTL^* se démontre de manière similaire, mais la preuve est compliquée par l'interprétation des formules de CTL^* sur les états : en général, $q_1 \sim_f q_2$ n'implique pas $q_1 \sim_{\neg f} q_2$, car $q_1 \models \neg f$ n'est pas équivalent à $q_1 \not\models f$. Par exemple, l'état 2 du graphe ordinal G_3 ne satisfait ni $X\,X\top$ ni $\neg X\,X\top$.

Théorème 4.3.7 *Pour tout graphe transitif $G = (Q, \rightarrow)$, tout ensemble $F \subseteq CTL^*$ fermé par sous formules, tous états $q_1, q_2 \in Q$,*

$$q_1 \sim_F q_2 \text{ implique } \forall q \in [q_1, q_2], q_1 \sim_F q$$

Preuve Nous montrons d'abord le Lemme 4.3.8 qui établit des propriétés sur les exécutions dans un graphe transitif G.

Lemme 4.3.8 *Pour tout graphe transitif $G = (Q, \rightarrow)$, toute $f \in CTL^*$, pour tous $\pi, \pi' \in Ex(G)$, si $\pi \not\sim_f \pi'$ et $\pi^1 \sim_{SF(f)} \pi'^1$, alors il existe une sous-formule de f de la forme $\forall g$ telle que $\pi(0) \not\sim_g \pi'(0)$*

Preuve La preuve se fait par induction sur la structure de f.

- Les cas $\top \Gamma \neg f$ et $f \wedge g$ sont évidents.
- $f = X\, g$ est excluΓcar dans ce cas il est impossible que π^1 et π'^1 satisfassent les mêmes sous-formules de f. En effetΓsi par exemple $\pi \models X\, g$ et $\pi' \not\models X\, g\Gamma$alors $\pi^1 \models g$ mais ou bien π'^1 n'existe pasΓou bien $\pi'^1 \not\models g\Gamma$et donc $\pi \not\sim_{SF(f)} \pi'^1$.
- $f = g\, U\, h$: comme f est équivalente à $h \vee g \wedge X\, f\Gamma$si $\pi \not\sim_f \pi'$ et $\pi^1 \sim_{SF(f)} \pi'^1\Gamma$ en particulier $\pi \sim_{f\, U\, g} \pi'$. Donc $\pi \not\sim_f \pi'$ implique ou bien (1) $\pi \not\sim_h \pi'\Gamma$ou bien (2) $\pi \not\sim_g \pi'$. Autant pour (1) que pour (2)Γil suffit d'appliquer l'hypothèse d'induction pour conclure.
- $f = \forall g$: par définition de l'opérateur $\forall$ et de la sémantique des formules de CTL^* sur les états d'un grapheΓf est la sous-formule cherchée.

□

Nous pouvons maintenant prouver le Théorème 4.3.7 par induction sur la structure des formules de F : soit $f \in F$ et supposons $q_1 \models f$. Etant donné $q \in [q_1, q_2]\Gamma$ nous montrons que pour tout $\pi \in Ex(q)\Gamma\pi \models f$.

Puisque $q \in [q_1, q_2]\Gamma$pour tout $\pi \in Ex(q)\Gamma$il existe $\pi_1 \in Ex(q_1)$ tel que $\pi^1 = \pi_1^1$ (il suffit de prendre $\pi_1 = q_1.\pi^1$). Par hypothèse $\pi_1 \models f$ (car $q_1 \models f$). Supposons $\pi \not\models f$. D'après le Lemme 4.3.8Γon peut exhiber une sous-formule g de f telle que $q_1 \not\sim_g q\Gamma$ce qui contredit l'hypothèse d'induction.
De mêmeΓon montre que $q \models f$ implique $q_2 \models f\Gamma$et donc $q_1 \models f$. □

Que ce soit pour la logique CTL^* ou pour la logique $L_\mu\Gamma$ le Théorème de l'Intervalle permet d'établir que pour toute formule $f\Gamma$ l'ensemble des états d'un graphe ordinal peut être partitionné en parties convexes qui caractérisent les classes d'équivalences induites par l'ensemble des sous-formules de f.

Notations 5 *Etant donné deux ordinaux α et β tels que $\alpha < \beta$, nous posons* $[\alpha, \beta[\stackrel{def}{=} \{\gamma | \alpha \leq \gamma < \beta\}$.

Proposition 4.3.9 *Soit G_λ un graphe ordinal.*
Pour toute formule fermée $f_0 \in L_\mu$, l'ensemble des états de G_λ peut être partitionné en $[0,\lambda[= [\lambda_0,\lambda_1[+[\lambda_1,\lambda_2[+\ldots+[\lambda_k,\lambda+1[$ avec :

- $0 = \lambda_0 < \lambda_1 < \ldots < \lambda_k \leq \lambda$,
- $\lambda_i \sim_{FL(f_0)} \lambda_j$ *ssi* $i = j$,
- *pour tout* $i = 0,\ldots,k-1$, *et tous* $x, y \in [\lambda_i, \lambda_{i+1}[$, $x \sim_{FL(f_0)} y$,
- *pour tous* $x, y \in [\lambda_k, \lambda+1[$, $x \sim_{FL(f_0)} y$

Si $f_0 \in CTL^*$, il faut remplacer l'ensemble $FL(f_0)$ par $BC(f_0)$.

Preuve **Pour** L_μ La relation $\sim_{FL(f_0)}$ induit au plus $2^{|FL(f_0)|}$ classes d'équivalences non vides. Puisque G_λ est un graphe transitifΓle Théorème de l'Intervalle 4.3.6 nous dit que ces classes d'équivalences sont convexes et nous les ordonnons canoniquement.

Pour CTL^* L'argument est le même que précédemmentΓmais en utilisant le Théorème 4.3.7.

□

La Proposition 4.3.9 exprime que la formule f_0 est toujours vraie ou toujours fausse "au dessus" de l'ordinal λ_k. Ce qui nous fournit un majorant pour $h(f_0)$. En réalitéΓle majorant λ_k est précisément égal à kΓcomme le montre la Proposition 4.3.10.

Proposition 4.3.10
- *Pour tout $f_0 \in L_\mu$, $i \sim_{FL(f_0)} \lambda_i$, pour tout $i \in \{0,\ldots,k\}$.*
- *Pour tout $f_0 \in CTL^*$, $i = \lambda_i$, pour tout $i \in \{0,\ldots,k\}$.*

Preuve
- $f_0 \in L_\mu$: nous montrons que $i \sim_f \lambda_i$ pour tout $f \in FL(f_0)$Γpar une double induction sur i et sur f (pour l'ordre $\prec$).
  Le cas $i = 0$ est clair puisque $0 = \lambda_0$.
  Les cas des formules $f \wedge g$Γ$\neg f$ et $\mu Y.f(Y)$ sont simples puisque leurs fermetures de Fischer-Ladner s'expriment en termes des fermetures de Fischer-Ladner de formules plus petites.
  Il reste donc à traiter le cas de $\Diamond f$. Si $i \models \Diamond f$Γalors $\lambda_i \models \Diamond f$ car $i \leq \lambda_i$. SinonΓi.e. $i \not\models \Diamond f$Γon a pour tout $j < i$Γ$j \not\models f$. Par hypothèse d'inductionΓ $\lambda_j \not\models f$Γpour tout $j < i$Γimpliquant $\lambda_i \not\models \Diamond f$.

- $f_0 \in CTL^*$: nous disons d'un ensemble F de formules qu'il est *finiment engendré* si toute combinaison booléenne de formules de F est équivalente à une combinaison booléenne finie de formules de F.

Lemme 4.3.11 *Soient $G = (Q, \rightarrow)$ un graphe d'états, et F un ensemble de formules de CTL^* fermé par sous-formules, par combinaisons booléennes, et finiment engendré. Pour tous états $q, q' \in Q$, si $q \sim_F q'$ alors pour toute exécution $\pi \in Ex(q)$, il existe $\pi' \in Ex(q')$ telle que $\pi \sim_F \pi'$.*

Preuve Supposons qu'un tel π' n'existe pas. Alors pour tout $\pi' \in Ex(q')$ il existe une formule $f_{\pi'}$ telle que $\pi \not\sim_{f_{\pi'}} \pi'$. On peut supposer que $\pi \not\models f_{\pi'}$ et $\pi' \models f_{\pi'}$ (car F est fermé pour l'opération $\neg$). Par construction des formules $f_{\pi'}$ on a $q' \models \bigvee_{\pi'} f_{\pi'}$ et $q \not\models \bigvee_{\pi'} f_{\pi'}$. Cette disjonction peut-être infinie mais comme F est finiment engendré il existe une formule $g \in F$ équivalente à $\bigvee_{\pi'} f_{\pi'}$. Mais alors $q \not\sim_g q'$ contredit l'hypothèse. □

Nous montrons que $i = \lambda_i$ par induction sur i et sur le nombre de formules de la forme $\forall h$ dans $BC(f_0)$.
On a bien $0 = \lambda_0$.
Supposons qu'il existe i tel $i < \lambda_i$ et prenons le minimal. Il est clair que $i \neq 0$. Alors $i \in [i-1, \lambda_i[$ et par construction de la partition $i \sim_{BC(f_0)} i-1$ et $i \not\sim_{BC(f_0)} \lambda_i$. Il existe donc $f \in BC(f_0)$ telle que $i \not\sim_f \lambda_i$. Comme l'ensemble $BC(f_0)$ est fermé pour l'opérateur $\neg$ on peut choisir f telle que $i \models f$ et $\lambda_i \not\models f$. Puisque $\lambda_i \not\models f$ il existe une exécution $\pi \in Ex(\lambda_i)$ telle que $\pi \not\models f$. Puisque $\pi^1(0) < \lambda_i$ il existe $j \leq i-1$ tel que $\pi^1(0) \sim_{BC(f_0)} j$. Comme $BC(f_0)$ est finiment engendré le Lemme 4.3.11 nous garantit l'existence d'une exécution $\sigma \in Ex(j)$ telle que $\sigma \sim_{BC(f_0)} \pi^1$. Puisque $i \models f$ l'exécution $\pi' \stackrel{\text{def}}{=} i.\sigma \models f$. Mais alors en appliquant le Lemme 4.3.8 à π et π' on peut exhiber une formule de la forme $\forall g \in BC(f_0)$ telle que $\pi(0) \not\sim_g \pi'(0)$ i.e. $i \not\sim_g \lambda_i$. On en déduit que $i \not\sim_{G_0} \lambda_i$ pour un sous-ensemble $G_0 \subseteq BC(f_0)$ contenant moins de formules de la forme $\forall h$ ce qui contredit l'hypothèse.

□

Les Propositions 4.3.9 et 4.3.10 nous permettent d'affirmer que :

Corollaire 4.3.12
- *Pour toute formule $f_0 \in CTL^*$, $h(f_0) \leq 2^{|BC(f_0)|}$*
- *Pour toute formule fermée $f_0 \in L_\mu$, $h(f_0) \leq 2^{|FL(f_0)|}$,*

La borne $2^{|FL(f_0)|}$ pour L_μ peut être grandement améliorée en considérant les Lemmes 4.3.13 et 4.3.14 qui suivent.

Lemme 4.3.13 *Pour tout $i = 1, \ldots k$, il existe une formule g_i telle que*

- $\Diamond g_i \in FL(f_0)$,
- $i - 1 \not\models \Diamond g_i$,
- $i \models \Diamond g_i$.

Preuve Puisque $i \not\sim_{FL(f_0)} i-1$Γil existe une formule $g \in FL(f_0)$ telle que $i \not\sim_g i-1$. Prenons g minimale pour $\prec$. Alors g ne peut être de la forme $\top$Γ $\neg f$Γ $f \wedge g$Γ ou $\mu Y.f(Y)$Γdonc g est une formule $\Diamond g_i$ et doncΓ$i \models \Diamond g_i$ et $i-1 \not\models \Diamond g_i$. □

Lemme 4.3.14 *Les formules g_i sont toutes distinctes.*

Preuve Supposons que $g_i = g_j$ avec $j < i$. Mais alorsΓd'une partΓ$j \models \Diamond g_i$ car $g_i = g_j$Γet d'autre partΓ$j \not\models \Diamond g_i$ car $j \leq i-1$ et $i-1 \not\models \Diamond g_i$ par construction des g_iΓce qui mène à une contradiction. □

En énumérant les formules g_iΓon en déduit que :

Corollaire 4.3.15 *Pour tout $f_0 \in L_\mu$, $h(f_0) \leq |\{\Diamond g \in FL(f_0)\}|$*

[Her87] montre que pour le μ-calcul positif (i.e. ne contenant pas l'opérateur $\neg$Γmais l'opérateur $\Box$ dual de $\Diamond$ et la constante $\bot$) la hauteur d'une formule f_0 est bornée par $Max(|\{\Diamond g \in FL(f_0)\}|, |\{\Box g \in FL(f_0)\}|)$.

Concernant la hauteur d'une formule f_0 de CTL^*Γ nous n'avons pas actuellement de meilleur majorant que $2^{|BC(f_0)|}$. Nous donnons l'exemple d'une suite de formules $(f_n)_{n \in N}$ telle que $h(f_n)$ soit strictement plus grande que le nombre maximum d'opérateurs X emboîtés dans f_nΓnoté $d(f_n)$.

Notations 6 *Nous introduisons les abréviations suivantes :*

f	*définition*	$d(f)$
$(\geq n)$	$\exists X^n \top$	n
$(< n)$	$\neg(\geq n)$	n
$(= n)$	$(\geq n) \wedge \neg(\geq n+1)$	$n+1$
$(\exists n)$	$\top\, U\, (= n)$	$n+1$

Pour tout $n > 0$Γon pose $f_n \stackrel{\text{def}}{=} \exists(n-1) \vee X\,(< n-1)$. ClairementΓpour tout $n > 0$Γ

$$n \models \forall f_n \text{ et } n+1 \not\models \forall f_n$$

Par conséquent $h(f_n) \geq n+1$Γet pourtant $d(f_n) = n$.

4.3.4 Pouvoir de distinction de HML, CTL, CTL^* et L_μ

Les résultats de la section précédente montrent que la hauteur des formules de HMLΓ CTLΓ CTL^* et L_μ est un ordinal fini. Par conséquentΓpour toute logique $L \in \{HML, CTL, CTL^*, L_\mu\}$Γ $dp(L) \leq \omega$.

Comme nous l'avons déjà remarquéΓon peut trouver des formules de HML de hauteur arbitrairement grande : il suffit de considérer les formules $EX^n \top$ de hauteur nΓd'où le corollaire :

Corollaire 4.3.16 *Pour toute logique $L \in \{HML, CTL, CTL^*, L_\mu\}$, $dp(L) = \omega$*

Les résultats de cette section mettent donc en évidence que dans le cas général des ST à branchement infini aucune des logiques HML, CTL, CTL^* et L_μ n'est adéquate pour la bisimulation.

En réalité, il nous aurait suffit de prouver ce résultat pour la logique L_μ pour borner le d.p. de toutes les logiques moins expressives (e.g. HML, CTL et CTL^*). Nous pensons cependant que nos résultats présentent un intérêt en eux-mêmes dans la mesure où ils fournissent une notion de complexité des formules et indiquent les coûts des traduction entre ces logiques, lorsque les traductions existent. Par exemple, si on peut montrer qu'il existe des formules de CTL^* pour lesquelles la hauteur atteint la borne du Corollaire 4.3.12, on en déduira que toute traduction de CTL^* dans L_μ est au moins de complexité doublement exponentielle.

4.3.5 Résultats brefs pour les logiques $\mathcal{C}$–TL et PDL

Les logiques $\mathcal{C}$–TL de [BT85] sont des généralisations de la logique RTL de Rounds et Gurevich. [BR83] a montré que pour les systèmes à b.i., la logique RTL n'est pas adéquate pour la bisimulation (mais compatible seulement). Les travaux de [BT85] généralisent ce résultat à toutes les logiques $\mathcal{C}$–TL, en montrant que l'équivalence double (voir le Chapitre 2) (moins fine que la bisimulation) est strictement plus fine que les équivalence induites par ces logiques. Dans tous ces travaux, les contre-exemples utilisés sont ad hoc. Nous proposons ici d'utiliser les processus ordinaux pour établir ces résultats.

Nous définissons une logique TL (pour *Trace Logic*) qui contient toutes les logiques $\mathcal{C}$–TL de [BT85] lorsque les graphes sont étiquetés par une seule action. Dans le cas général les logiques $\mathcal{C}$–TL sont beaucoup plus riches. Nous renvoyons à cet article pour les définitions exactes de ces logiques.

La logique TL a la syntaxe suivante :

$$TL(\ni f, g) ::= \top | \neg f | f \wedge g | \forall P \langle f \rangle | \forall P[f]$$

où $P \subseteq N$.

Les formules de TL s'interprètent dans un graphe $G = (Q, \rightarrow)$. Pour $p \in N$ et tous $q, q' \in Q$, nous notons $q \stackrel{p}{\Rightarrow} q'$ si il existe une suite $q = q_0 \rightarrow q_1 \rightarrow \ldots q_p = q'$. En particulier on a $q \stackrel{0}{\Rightarrow} q$. La relation de satisfaction $\models \subseteq Q \times TL$ est définie par :

- $\top$, $\neg f$, $f \wedge g$ ont des définitions évidentes,
- $q \models \forall P \langle f \rangle$ ssi pour tout $p \in P$, il existe $q \stackrel{p}{\Rightarrow} q'$ tel que $q' \models f$,
- $q \models \forall P[f]$ ssi pour tout $p \in P$ et tout $q \stackrel{p}{\Rightarrow} q'$, $q' \models f$.

La logique RTL est le fragment de TL obtenu en restreignant la classe des ensembles $P \subseteq N$: il doit exister un langage régulier L tel que $P \stackrel{\text{def}}{=} \{|w| \, | w \in L\}$.

Par exemple RTL admet la modalité $\forall N\langle . \rangle$ car $N = \{|\, w \,| \, |w \in \{a\}^*\}$ où $\{a\}^*$ est le langage (régulier) des mots finis sur l'alphabet $\{a\}$.

Plus généralement et de manière adaptée à notre cadre [BT85] définissent pour chaque classe $\mathcal{C}$ de parties de N la logique $\mathcal{C}$–TL dans laquelle les ensembles P des modalités $\forall P\langle . \rangle$ et $\forall P[.]$ doivent appartenir à $\mathcal{C}$. Par exemple lorsque la classe $\mathcal{C}$ est formée de toutes les parties de N on retrouve la logique TL (appelée $U - TL$ dans [BT85]).

La logique PDL de [FL79] est le fragment de RTL obtenu en enlevant les formules de la forme $\forall P\langle f \rangle$. En posant $\exists P\langle f \rangle \stackrel{\text{def}}{=} \neg\forall P[\neg f]$ on a $q \models \exists P\langle f \rangle$ ssi il existe $p \in P$ et $q \stackrel{p}{\Rightarrow} q'$ tels que $q' \models f$. Les formules $EX^n\top$ de CTL se traduisent donc facilement en PDL par $\exists\{n\}\langle \top \rangle$ ou $\exists\{1\}\langle \exists\{1\} \ldots \exists\{1\}\langle \top \rangle \ldots \rangle$.

Nous montrons que le d.p. de la logique TL est égale à ω^2 et que celui de son fragment PDL est ω.

Pour borner le d.p. de TL nous procédons comme pour la logique CTL en définissant pour chaque formule $f \in TL$ une notion de profondeur et en montrant une proposition similaire à la Proposition 4.3.5. La profondeur de f notée $d(f)$ dépend du nombre maximum de modalités $\forall P\langle . \rangle$ et $\forall P[.]$ emboîtées dans f. Formellement :

$$\begin{array}{ll} d(\top) = 0 & d(\neg f) = d(f) \\ d(f \wedge g) = max(d(f), d(g)) & \\ d(\forall P\langle f \rangle) = d(f) + \omega & d(\forall P[f]) = d(f) + Min\{p \in P\} \end{array}$$

Proposition 4.3.17 *Pour tout $f \in TL$ et tout ordinal $\alpha \geq d(f)$, $\alpha \sim_f d(f)$.*

Preuve Elle se fait par induction sur la structure de f :

- Les cas $\top$ $\neg f$ et $f \wedge g$ sont évidents.
- $f = \forall P\langle g \rangle$: soit $\gamma = d(g) + \omega$ et $\alpha \geq \gamma$.
  Puisque $\rightarrow$ est transitive pour tout $p \in N$ $\gamma \stackrel{p}{\Rightarrow} \gamma'$ implique $\alpha \stackrel{p}{\Rightarrow} \gamma'$. On en déduit que $\gamma \models \forall P\langle g \rangle$ implique $\alpha \models \forall P\langle g \rangle$.
  Supposons $\gamma \not\models \forall P\langle g \rangle$. Il existe donc $p \in P$ tel que pour tout $\gamma \stackrel{p}{\Rightarrow} \gamma'$ $\gamma' \not\models g$. En particulier $\gamma \stackrel{p}{\Rightarrow} d(g)$ (car $\gamma \geq d(g) + \omega$) implique $d(g) \not\models g$. Par hypothèse d'induction pour tout $\beta \geq d(g)$ $\beta \not\models g$ donc $\alpha \not\models \forall P\langle g \rangle$.
- $f = \forall P[g]$: soit $p_0 \stackrel{\text{def}}{=} Min\{p \in P\}$ $\gamma = d(g) + p_0$ et $\alpha \geq \gamma$.
  Si $\gamma \models \forall P[g]$ alors pour tout $p \in P$ et tout $\gamma \stackrel{p}{\Rightarrow} \gamma'$ $\gamma' \models g$. De plus $p_0 \in P$ et $\gamma \stackrel{p_0}{\Rightarrow} d(g)$ impliquent $d(g) \models g$. Donc par hypothèse d'induction pour tout $\beta \geq d(g)$ $\beta \models g$. Donc $\alpha \models \forall P[g]$.
  Sinon $\gamma \not\models \forall P[g]$. Il existe donc $p \in P$ et $\gamma \stackrel{p}{\Rightarrow} \gamma'$ tels que $\gamma' \not\models g$. Comme on a aussi $\alpha \stackrel{p}{\Rightarrow} \gamma'$ on conclut facilement.

□

Remarquons que pour le fragment PDLΓla hauteur des formules est bornée par ω. Comme de plusΓla hauteur de la formule $f_n \stackrel{\text{def}}{=} \exists\{n\}\langle\top\rangle\ PDL$ est égale à n (car elle est équivalente à la formule $EX^n\top \in HML$)Γon en déduit

Corollaire 4.3.18 *$dp(PDL) = \omega$*

Remarque 6 *Nous savons (voir [EH86]) que le fragment ΔPDL [Str81] de PDL est strictement plus expressif que CTL^*, ce qui fournit une autre preuve pour le d.p. de CTL^*.*

La Proposition 4.3.17 montre que la hauteur d'une formule f de TL est bornée par sa profondeurΓelle même bornée par ω^2.

Nous montrons maintenant que pour tout $n \in N$ Γil existe une formule f_n de TL qui distingue jusqu'à $\omega * n$. Soit $(f_n)_{n\in N}$ la suite de formules définie par $f_0 \stackrel{\text{def}}{=} \top$Γ $f_{n+1} \stackrel{\text{def}}{=} \forall N\,\langle f_n\rangle$. AlorsΓ

- $\omega * n \models f_n$Γcar $d(f) = \omega * n$
- pour tout ordinal αΓ$\alpha \not\models f_n$ implique $\alpha \not\models f_{n+1}$ (car $0 \in N$ et $\alpha \stackrel{0}{\Rightarrow} \alpha$).

Nous montrons par induction sur n que pour tout $\beta < \omega * n$Γ$\beta \not\models f_n$. Le cas $n = 0$ est évident. Soit $\beta < \omega * n + \omega$. Si $\beta < \omega * n$Γalors par hypothèse d'induction $\beta \not\models f_n$Γdonc $\beta \not\models f_{n+1}$. SinonΓ$\omega * n \leq \beta < \omega * n + \omega$. Alors β est de la forme $\omega * n + k$Γmais alors pour tout $\beta \stackrel{k+1}{\Rightarrow} \beta'$Γ$\beta' < \omega * n$Γet par hypothèse d'induction $\beta' \not\models f_n$Γdonc $\beta \not\models f_{n+1}$.
Remarquons que seule la modalité $\forall N\,\langle.\rangle$ est utilisée.

Corollaire 4.3.19 *Pour toute classe $\mathcal{C}$ de parties de N, si $N \in \mathcal{C}$, alors $dp(\mathcal{C}-TL) = \omega^2$. En particulier $dp(RTL) = dp(TL) = \omega^2$.*

Les résultats de cette section montrent que sauf à se restreindre au cas du branchement finiΓ il n'existe pas de résultat d'adéquation entre d'une part une sémantique basée sur la bisimulation et d'autre part les logiques d'étatsΓcomme HMLΓCTLΓCTL^*ΓL_μΓPDLΓRTLΓetc...

4.4 Pouvoir de distinction des congruences engendrées par les logiques d'états

Suite aux résultats obtenus jusqu'iciΓ il est naturel de se demander si les équivalences induites par les logiques ne pourraient être elles-mêmes utilisées comme équivalences sémantiques. Dans l'esprit de trouver des sémantiques compatibles

avec la construction modulaire des programmes, l'utilisation d'une de ces équivalences n'est intéressante que si l'on montre que cette équivalence est en fait une congruence pour les combinateurs classiques de programmes, tels que ceux de CCS, CSP, ... Dans [Sch90a], il est établi que l'équivalence induite par la logique CTL n'est pas une congruence pour la composition parallèle "à la CSP". Dans cette section, nous répondons globalement pour les équivalences induites par CTL et CTL^*, en montrant qu'aucune d'entre elles n'est une congruence. La preuve repose sur le fait que le d.p. des plus grandes congruences induites par les logiques (vis à vis des combinateurs *tyft* de [GV88]) est strictement plus grand que ω, c-à-d. que le d.p. des équivalences logiques.

De plus nous étudions une sous-classe des combinateurs *tyft*, appelés combinateurs *sans copie* (qui contient tous les combinateurs usuels de CCS), vis à vis de laquelle les congruences induites par CTL et CTL^* ont encore un d.p. strictement plus grand que ω.

Enfin, nous montrons que la congruence de trace induite par les contextes *tyft* est strictement plus fine que celle induite par la sous-classe des contextes *sans copie*. Ce dernier résultat met en évidence une différence d'expressivité entre les formats des règles SOS qui décrivent les contextes *tyft* et les contextes *sans copie*.

4.4.1 Notre démarche

Comme indiqué ci-dessus, nous considérons dans cette section deux classes de combinateurs : celle des combinateurs *tyft* définie précédemment en Section 2.2.1, et sa sous-classe formée des combinateurs *sans copie*, définie plus loin en Définition 4.4.1. Ces combinateurs sont utilisés pour construire des contextes dans lesquels seront placés les processus ordinaux. Informellement, si $\mathcal{C}[.]$ est un contexte à 1 place (c-à-d. un terme avec 1 trou) construit à partir de combinateurs *tyft*, et si α est un ordinal, le terme $\mathcal{C}[\alpha]$ dénote le processus qui peut évoluer par des transitions que l'on prouve, d'une part, au moyen des règles qui décrivent les combinateurs de $\mathcal{C}[.]$ et d'autre part, en utilisant que $\alpha \rightarrow \beta$ pour tout $\beta < \alpha$.

Etant donnée une équivalence sémantique $\sim$, nous étudions le d.p. s'il existe, de la plus grande congruence contenue dans $\sim$ relativement à tous les combinateurs *tyft* (resp. *sans copie*). Cette congruence identifie deux ordinaux α et α' ssi pour tout contexte *tyft* (resp. *sans copie*) $\mathcal{C}[.]$, $\mathcal{C}[\alpha] \sim \mathcal{C}[\alpha']$.

Informellement notre démarche est la suivante : soit α un ordinal placé dans un contexte *tyft* $\mathcal{C}[.]$ et π une exécution partant de $\mathcal{C}[\alpha]$. A cette exécution est associée une preuve $\mathcal{P}_\pi$, construite à partir de transitions entre ordinaux et d'instances des règles SOS. Nous expliquons ensuite comment, à partir de $\mathcal{P}_\pi$, il est possible de construire une preuve $\mathcal{P}_{\pi'}$ d'une exécution qui imite π et qui part d'un état de la forme $\mathcal{C}[\alpha']$, à condition que l'ordinal α' soit "suffisamment" grand.

Un tel résultat nous permet :

1) de borner le d.p. des congruences de trace induites par les contextes *tyft* et les contextes *sans copie*Γ

2) de définir une nouvelle notion de profondeur des formules de CTL et CTL^*Γ à partir de laquelle nous déduirons une borne pour le d.p. des congruences induites par CTL et CTL^*.

Nous n'avons pas traité le cas de L_μ qui s'avère beaucoup moins trivial.

4.4.2 Définitions préliminaires

Soit $\Sigma = (F, ar)$ une signature (à une sorte). Nous définissons $\tilde{\Sigma}$ l'extension de Σ obtenue en ajoutant toutes les constantes ordinales. Un terme de l'algèbre $T_{\tilde{\Sigma}}$ est construit à partir des symboles de fonction de Σ et d'ordinaux. Par exempleΓsi $\mathcal{C}[.,\ldots,.]$ est un contexte *tyft* à n placesΓc-à-d. un terme de T_Σ à n trousΓet que $\alpha_1, \alpha_2, \ldots, \alpha_n$ sont n ordinauxΓle terme $t = \mathcal{C}[\alpha_1, \alpha_2, \ldots, \alpha_n]$ appartient à $T_{\tilde{\Sigma}}$.

Si $P = (\Sigma, A, R)$ est une SGΓnous notons $\tilde{P}$ l'extension de P obtenue en étendant la signature Σ en $\tilde{\Sigma}$ et en ajoutant à R tous les axiomes $\alpha \to \beta$ (pour $\beta < \alpha$). En reprenant l'exemple ci-dessusΓ le terme $t = \mathcal{C}[\alpha_1, \ldots, \alpha_n]$ est un état du graphe spécifié par $\tilde{P}$Γet les transitions partant de t sont toutes les transitions de la forme $t \xrightarrow{a}_{\tilde{P}} t'$ que l'on pourra prouver en utilisant les règles de R et les axiomes $\alpha \to \beta$.

Pour des raisons de lisibilitéΓet lorsqu'il n'y a pas d'ambiguïtéΓnous noterons P au lieu de $\tilde{P}$.

Exemple 9 *Considérons les règles* tyft *suivantes :*

$$\frac{x \to y}{g(x) \to f(x,y)} \qquad \frac{x \to y}{f(x,z) \to f(x,y)} \qquad \frac{x \to y}{f(x,z) \to f(f(x,z),y)} \qquad f(x,y) \to y$$

On peut facilement prouver la transition $g(\omega) \to f(\omega, 0)$, en utilisant la première règle, et l'axiome $\omega \to 0$. Il est facile de voir que l'on peut aussi prouver l'existence d'un chemin infini $g(\omega) \to f(\omega,0) \to f(f(\omega,0),0) \to f(f(f(\omega,0),0),0)\ldots$ Une preuve de ce chemin est bien sûr infinie.

L'Exemple 9 précédent nous montre que des ordinaux placés dans des contextes *tyft* peuvent avoir des comportements infinisΓmais aussi que le nombre d'occurrences d'ordinaux dans les termes le long d'un chemin peut augmenter. Nous définissons dans la suite une sous-classe des combinateurs *tyft* pour laquelle ce nombre ne peut que stagner ou diminuer le long de toute exécution.

Définition 4.4.1 (Règles sans copie) *Soit r une règle* tyft *(pure) de la forme*

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, \ldots, x_n) \xrightarrow{a} t}$$

La règle r est sans copie *si pour chaque x_k il existe au plus un $y \in Var(t)$ atteignable à partir de x_k dans le graphe de dépendance de r.*

Nous disons d'une SG $P = (\Sigma, A, R)$ qu'elle est sans copie si toute règle $r \in R$ est sans copie.

Remarque 7 • *Informellement, une règle sans copie ne permet pas de produire des duplicata de l'état du système placé dans le contexte.*

- *Une règle est sans copie si et seulement si son graphe dépendance ne contient aucun branchement.*
- *Toutes les règles de la Section 2.2.1 et plus généralement toutes celles de CCS sont sans copie.*
- *Les axiomes $\alpha \to \beta$ sont des règles sans copie. Il en résulte que si P est une SG sans copie, alors $\tilde{P}$ est aussi sans copie.*

4.4.3 Résultats préliminaires

Nous considérons un exemple simple qui nous servira pour illustrer nos définitions et nos résultats.

Exemple 10 *Soit P une SG en format* tyft *dont les règles sont les suivantes :*

$$\frac{x \to y \quad x \to z \quad z \to u}{f(x) \to g(y,z)} \qquad \frac{x \to y \quad x \to z}{h(x) \to l(z,y)}$$

Considérons maintenant l'ordinal 4 placé dans le contexte $h(f(.))$. On peut alors prouver la transition $h(f(4)) \to l(g(2,2), g(3,1))$ en utilisant par exemple la preuve $\mathcal{P}_0$ suivante :

$$\frac{\dfrac{4 \to 2 \quad 4 \to 2 \quad 2 \to 0}{f(4) \to g(2,2)} \quad \dfrac{4 \to 3 \quad 4 \to 2 \quad 2 \to 1}{f(4) \to g(3,1)}}{h(f(4)) \to l(g(2,2), g(3,1))}$$

Il est clair que si dans cette preuve on remplace toutes les occurrences de l'ordinal 4 par l'ordinal 6, on obtient une preuve de la transition $h(f(6)) \to l(g(2,2), g(3,1))$. Il en serait de même de tout ordinal $\beta \geq 4$.

Nous étudions dans la suite un résultat très généralΓénoncé en Proposition 4.4.1Γ qui exprime les liens qu'il faut exiger entre deux ordinaux α et α' pour que dans tout contexte *tyft*$\mathcal{C}$[.]Γsi on peut prouver une exécution π partant de $\mathcal{C}[\alpha]$ alors on peut aussi prouver une exécution π' partant de $\mathcal{C}[\alpha']$ qui dans un certain sens est "similaire" à π.

Avant d'énoncer ce résultatΓnous introduisons une famille d'équivalences sur les ordinaux qui permet de parler de pouvoir de distinctionΓsans pour autant désigner une équivalence particulière.

Définition 4.4.2 *A tout ordinal γ, nous associons la relation $\overset{\gamma}{\smile}$ sur les ordinaux, définie par :*

$$\alpha \overset{\gamma}{\smile} \beta \overset{def}{=} \alpha = \beta \text{ ou } \alpha, \beta \geq \gamma$$

$\alpha \overset{\gamma}{\smile} \beta$ se lit "α et β sont égaux jusqu'à γ".

Clairement $dp(\overset{\gamma}{\smile}) = \gamma$ et si $\gamma \geq \gamma'$ alors $\alpha \overset{\gamma}{\smile} \beta$ implique $\alpha \overset{\gamma'}{\smile} \beta$.

Nous étendons la définition de $\overset{\gamma}{\smile}$ sur $T_{\tilde{\Sigma}}$ et sur les exécutions de $G(P)$ par :

- pour tous $f, g \in \Sigma$

$$f(u_1, \ldots, u_n) \overset{\gamma}{\smile} g(v_1, \ldots, v_n) \text{ ssi } f = g \text{ et } u_i \overset{\gamma}{\smile} v_i \text{ pour tout } i = 1, \ldots n.$$

- si π et π' sont des exécutions de $G(P)$

$$\pi \overset{\gamma}{\smile} \pi' \text{ ssi } tr(\pi) = tr(\pi') \text{ et } \pi(i) \overset{\gamma}{\smile} \pi'(i), \text{ pour tout } i < |\pi|.$$

La Proposition 4.4.1 qui suit exprime que pourvu que des ordinaux α_i et α'_i $i = 1, \ldots, n$ soient assez grands (ou égaux) pour tout contexte *tyft* $\mathcal{C}[., \ldots, .]$ à n-places toute exécution partant de $\mathcal{C}[\alpha_1, \ldots, \alpha_n]$ peut "être imitée modulo $\overset{\gamma}{\smile}$" en partant de $\mathcal{C}[\alpha'_1, \ldots, \alpha'_n]$.

Proposition 4.4.1 *Soit $\mathcal{C}[., \ldots, .]$ un contexte* tyft *à n places et γ un ordinal.*

1. *Si $t_0 = \mathcal{C}[\alpha_1, \ldots, \alpha_n] \overset{\gamma+\omega_1}{\smile} \mathcal{C}[\alpha'_1, \ldots \alpha'_n] = t'_0$, alors pour tout $\pi \in Ex(t_0)$, il existe $\pi' \in Ex(t'_0)$ tel que $\pi \overset{\gamma}{\smile} \pi'$.*
2. *Si de plus on se restreint à des contextes sans copie, ω_1 peut être remplacé par ω si π est finie, et par $\omega * 2$ sinon.*

Pour prouver la Proposition 4.4.1 nous montrons comment il est possible de modifier la preuve de l'exécution π pour obtenir une preuve de π'.

Supposons pour simplifier que le contexte $\mathcal{C}[., \ldots, .]$ est à 1 place. Soit $\mathcal{P}$ une preuve de π. Nous extrayons de cette preuve $\mathcal{P}$ un arbre de dépendance T étiqueté par les occurrences des ordinaux qui apparaissent dans la preuve $\mathcal{P}$ et qui caractérise les dépendances entre ces occurrences.

Par exemple nous extrayons de la preuve $\mathcal{P}_0$ de l'Exemple 10 l'arbre T_0 de la Figure 4.4.

Il est facile de généraliser la construction d'un tel arbre dans le cas général même si l'exécution π est infinie. Nous ne donnons pas la définition formelle de cette construction puisque premièrement elle n'apporte rien d'essentiel et que deuxièmement sa description serait de toute façon très lourde à formaliser. Il faut seulement bien comprendre que pour construire cet arbre il faut se souvenir de toutes les règles utilisées dans la preuve $\mathcal{P}$ et des substitutions qui leur sont appliquées afin de reconnaître les occurrences d'un même ordinal dans $\mathcal{P}$. Remarquons simplement que l'arbre T est défini de façon univoque modulo l'ordre de ses branches.

Dans le cas d'un contexte à n place une preuve $\mathcal{P}$ de $\pi \in Ex(\mathcal{C}[\alpha_1, \ldots, \alpha_n])$ engendre non plus un arbre mais une forêt d'arbres $T_1, \ldots, T_n$ chaque T_i ayant pour racine l'ordinal α_i.

Nous définissons dans la suite la notion d'α-arbre où α est un ordinal qui sert techniquement à prouver la Proposition 4.4.1.

Définition 4.4.3 (α-arbre) *Un α-arbre est une structure $T = (L, \rightarrow, E)$ où*

- *$L \subseteq N^*$ est fermé par préfixe, $\epsilon \in L$ est le mot vide, appelé la racine de T,*
- *$\rightarrow \subseteq L \times L$ est telle que pour tous $p, q \in L, (p, q) \in \rightarrow$ si $\exists n, q = p.n$,*
- *$E : L \rightarrow \alpha + 1$ est telle que $E(\epsilon) = \alpha$ et $E(p) \geq E(q)$ pour tous $p, q \in L$ tels que $(p, q) \in \rightarrow$.*

Informellement, un α-arbre $T = (L, \rightarrow, E)$ est un arbre dont la racine est étiquetée par α et dont les autres nœuds (i.e. les éléments de $L \setminus \{\epsilon\}$) sont étiquetés par des ordinaux inférieurs ou égaux à α, qui ne peuvent que décroître ou stagner le long d'un chemin dans l'arbre. Puisque $<$ l'ordre sur les ordinaux est bien fondé, pour tout chemin infini $p_1 \rightarrow \ldots \rightarrow p_n \rightarrow \ldots$ dans T, il existe un entier m tel que $E(p_k) = E(p_m)$ pour tout $k \geq m$.

Reprenons maintenant l'Exemple 10. La preuve $\mathcal{P}_0$ peut être modifiée en une preuve de la transition $h(f(6)) \rightarrow l(g(2,2), g(3,1))$. Il suffit de renommer dans $\mathcal{P}_0$ toutes les occurrences de 4 par 6. On obtient alors la preuve :

$$\dfrac{\dfrac{6 \rightarrow 2 \quad 6 \rightarrow 2 \quad 2 \rightarrow 0}{f(6) \rightarrow g(2,2)} \quad \dfrac{6 \rightarrow 3 \quad 6 \rightarrow 2 \quad 2 \rightarrow 1}{f(6) \rightarrow g(3,1)}}{h(f(6)) \rightarrow l(g(2,2), g(3,1))}$$

En fait, il est suffisant pour construire cette preuve de donner un nouvel étiquetage de l'arbre T_0 (associé à la preuve $\mathcal{P}_0$) de sorte que 4, la racine de T_0, est renommée en 6 et que ce nouvel étiquetage respecte la décroissance des étiquettes le long des branches de T_0. Si un tel étiquetage existe, nous disons que l'arbre T_0 *peut être étiqueté en dessous de* 6. Plus formellement :

Définition 4.4.4 (Reétiquetage[2] d'un α-arbre en dessous de β modulo γ) *Soient $T = (L, \rightarrow, E)$ un α-arbre, β et γ deux ordinaux. L'α-arbre T* peut être

[2] ***Au sens où l'on donne un autre étiquetage.***

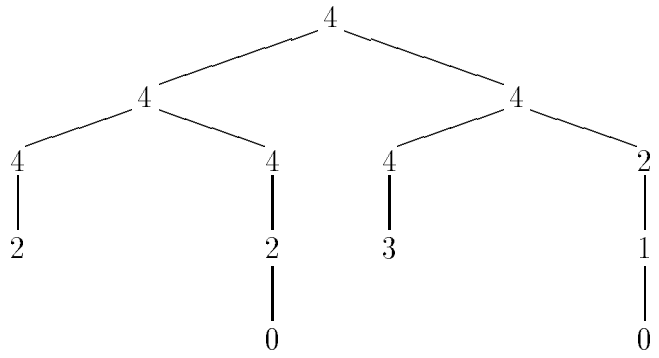


Figure 4.4: L'arbre T_0 extrait de la preuve $\mathcal{P}_0$ de l'Exemple 10

reétiqueté sous β modulo γ *(par E'), en abrégé r.s. β mod γ (par E') si il existe une fonction d'étiquetage $E' : L \to \beta + 1$ telle que pour tous $p, q \in L$,*

1. $E'(p) \leq \beta$ *(sous β)*
2. $E'(p) > E'(q)$ *ssi* $E(p) > E(q)$ *(respect des décroissances strictes dans T)*
3. $E'(p) \overset{\gamma}{\smile} E(p)$ *(modulo $\overset{\gamma}{\smile}$)*

D'après la Définition 4.4.4Γ

- Si T peut être r.s. β mod γ par E'Γalors $T' \overset{\text{def}}{=} (L, \to, E')$ est un β-arbre.
- Si $\beta' \geq \beta$ et si T peut être r.s. β mod γΓalors que T peut être r.s. β' mod γ.

L'arbre T_0 de l'Exemple 10 peut être r.s. 6 mod 0Γmais aussi r.s. β mod 0 pour tout $\beta \geq 4$.

Dans la suite nous étudions sous quelles hypothèsesΓun α-arbre peut être reétiqueté. Remarquons que lorsqu'un arbre est extrait d'une preuve à partir de règles sans copieΓcet arbre contient au plus un chemin infini.

Définition 4.4.5 *Un α-arbre est* primaire *si il contient au plus un chemin infini.*

Proposition 4.4.2 *Soient α, β, γ des ordinaux.*

1. *Si $\alpha \overset{\gamma+\omega_1}{\smile} \beta$, alors tout α-arbre T peut être r.s. β mod γ.*
2. *On peut remplacer ω_1 par $\omega * 2$ si on ne considère que des α-arbres primaires,*
3. *On peut remplacer ω_1 par ω si on ne considère que des α-arbres primaires finis.*

Preuve 1. On peut supposer sans perdre de généralité que $\alpha > \beta > \geq \gamma + \omega_1$. Soit $T = (L, \to, E)$ un α-arbre. Puisque $\beta > \gamma + \omega_1$Γil est suffisant de montrer que T peut être r.s. $\gamma + \omega_1$ mod γ. Ce qui peut se faire en montrant qu'il existe un ordinal dénombrable η $(< \omega_1)$ tel que T peut être r.s. $\gamma + \eta$ mod γ par une fonction d'étiquetage E'.
Nous définissons E' par:

- $E'(p) = E(p)$ pour tout p tel que $E(p) < \gamma$Γ
- Il ne reste alors qu'un ensemble dénombrable $\mathcal{R}$ de positions dans l'arbre T sur lequel E' n'est pas encore définie. Soit δ le sous-ordre qui correspond aux étiquettes (par E) $\{\alpha_0, \alpha_1 \ldots\}$ associé à $\mathcal{R}$. Soit $\{\delta_0, \delta_1 \ldots\}$ la représentation de δ. Nous posons alors $E'(p) = \gamma + \delta_i$ si $E(p) = \alpha_i$. Par constructionΓ $E'(\epsilon) < \gamma + \omega_1$.

2. Il est suffisant de montrer que chaque α-arbre primaire $T = (L, \to, E)$ peut être r.s. $\omega + k$ mod γ où $k \in N$. Considérons l'unique chemin infini de T partant de sa racine $p_0 \to p_1 \ldots$. Il existe $m \in N$ minimal tel que $E(p_i) = E(p_m)$Γ pour tout $i \geq m$. Mais alors il est facile de voir que le sous-arbre

de T à la position p_m peut être r.s. ω mod γ par un E' (car toutes les autres branches sont finies). Il reste à étendre E' sur un nombre fini k de positions dans Tce qui peut facilement être faitdonnant un étiquetage de T sous $\omega+k$ modulo γ.
Si l'α-arbre T est finion voit que ω suffit.

□

Nous pouvons maintenant prouver la Proposition 4.4.1. Pour simplifier l'exposé de la preuvenous ne considérons qu'un contexte $\mathcal{C}[.]$ à 1 place. Nous supposons donnés α_0 et α'_0 tels que $\alpha_0 \overset{\gamma+\omega_1}{\smile} \alpha'_0$.

1. Soit $\mathcal{P}$ une preuve de $\pi = \mathcal{C}_0[\alpha_0] \to t_1 \to \ldots$ (où les t_i sont des éléments de $T_{\tilde{\Sigma}}$). Nous extrayons de $\mathcal{P}$ un α_0-arbre $T = (L, \to, E)$. D'après la Proposition 4.4.2 comme $\alpha_0 \overset{\gamma+\omega_1}{\smile} \alpha'_0$ T peut être r.s. α'_0 mod γ par une fonction d'étiquetage E'. On obtient un α'_0-arbre T' tel que $E'(p) \overset{\gamma}{\smile} E(p)$ pour tout $p \in L$. En utilisant l'arbre T' et les règles utilisés dans la preuve $\mathcal{P}$on peut construire une preuve $\mathcal{P}'$ d'une exécution π' partant de $\mathcal{C}[\alpha'_0]\pi' = \mathcal{C}_0[\alpha'_0] \to t'_1 \to \ldots$. Par construction de $T't_i \overset{\gamma}{\smile} t'_i$pour tout i (toutes les instances des variables dans t_i et t'_i sont des ordinaux $\overset{\gamma}{\smile}$-équivalents). Comme les règles utilisées dans la preuve $\mathcal{P}'$ sont les mêmes que dans $\mathcal{P}$on a $tr(\pi) = tr(\pi')$d'où on déduit $\pi \overset{\gamma}{\smile} \pi'$.

2. Un raisonnement analogue s'applique aux contextes sans copieen remarquantd'après la Proposition 4.4.2 que l'ordinal $\omega * 2$ peut être ramené à ω si l'exécution est finiecar alors T est fini.

La Proposition 4.4.1 est un résultat fondamental pour la suite de notre étude. Elle permet de savoir sous quelles conditions deux ordinaux placés dans les mêmes contextes *tyft* (et sans copie) peuvent (1) engendrer les mêmes traces (c'est l'objet de la section suivante)et (2) satisfaire les mêmes formules des logiques CTL et CTL^* (c'est l'objet de la Section 4.4.5).

4.4.4 Congruences de traces

Nous établissons ici quelques résultats supplémentaires qui nous serviront par la suite. Nous en déduirons en particulier le d.p. des deux congruences de traces qu'induisent respectivement les contextes *tyft* et sans copie.

Définition 4.4.6 *Un α-arbre T est* à largeur bornée *si il contient au plus un nombre fini de chemins infinis. En particulier les arbres primaires sont à largeur bornée.*

Dans la suite nous utilisons les définitions suivantes :

Définition 4.4.7 *Soit α un ordinal.*

- $U(\alpha)$ *dénote le plus petit ordinal* δ *tel que tout* α*-arbre peut être r.s.* δ *mod* 0.
- $U'(\alpha)$ *dénote le plus petit ordinal* δ *tel que tout* α*-arbre à largeur bornée peut être r.s.* δ *mod* 0.

Lemme 4.4.3 *Pour tout ordinal* α,

1. $U(\alpha) = min(\alpha, \omega_1)$,
2. $U'(\alpha) = min(\alpha, \omega * 2)$

Preuve 1. Pour chaque $\alpha < \omega_1$ nous exhibons un α-arbre (binaire) T_α et nous montrons par induction sur les ordinaux la propriété $(*)$ suivante : T_α ne peut être reétiqueté sous un ordinal plus petit que α ce qui prouve que $U(\alpha) = \alpha$. Comme α est dénombrable nous pouvons le noter $\{\alpha_0, \alpha_1, \alpha_2, \ldots\}$. T_α est alors l'arbre étiqueté par α à toutes les positions 1^k $(k \in N)$ et tel que le sous-arbre à la position 1^k0 est T_{α_k}.
Clairement la propriété $(*)$ est vraie pour $\alpha = 0$.
Supposons $\alpha \neq 0$ et qu'il existe $\gamma < \alpha$ tel que T_α peut être r.s. γ mod 0 par E'. L'ordinal γ est un α_i. Nous pouvons supposer que c'est α_0. Comme $\alpha > \gamma$ le nouvel étiquetage E' de T_α doit respecter les décroissances dans T_α c-à-d- $\gamma = E'(\epsilon) > E'(0)$. Par conséquent $E'(0)$ est un ordinal $\beta < \gamma$ tel que T_γ peut être r.s. β mod 0 ce qui contredit l'hypothèse d'induction. Donc $U(\alpha) = \alpha$.

Pour tout $\alpha \geq \omega_1$ nous montrons d'abord que pour chaque α-arbre T il existe un ordinal β_T dénombrable tel que T peut être r.s. β_T mod 0. Soit T un α-arbre. Le sous-ordre de α formé des occurrences d'ordinaux dans T est bien évidemment dénombrable puisqu'il est borné par le nombre de position dans l'arbre T. Son type d'ordre nous définit l'ordinal β_T cherché. Donc $U(\alpha) \leq \omega_1$. Clairement si $\alpha \geq \omega_1$ alors pour tout $\beta < \omega_1$ on peut construire un α-arbre T'_β qui ne peut être r.s. β mod 0; il suffit de poser $T'_\beta \stackrel{\text{def}}{=} \alpha.T_\beta$ où T_β est l'arbre associé à l'ordinal dénombrable β comme indiqué plus haut. Donc $U(\alpha) = \omega_1$.

2. Remarquons que dans le cas où $\alpha < \omega * 2$ l'α-arbre T_α est à largeur bornée.
Si $\alpha < \omega * 2$ la preuve de $U(\alpha) = \alpha$ pour α dénombrable s'applique.
Si $\alpha \geq \omega * 2$ soit $T = (L, \rightarrow, E)$ un α-arbre à largeur bornée et soit l le nombre de chemins infinis de T. En descendant suffisamment bas le long de chacun des l chemins infinis de T il existe une position p_i dans T telle que $T_i \stackrel{\text{def}}{=} T/p_i$ le sous-arbre de T à la position p_i ne contient qu'un seul chemin infini et tel que l'étiquetage le long de ce chemin soit constant. On vérifie facilement que chaque T_i peut être r.s. ω mod 0. En remarquant qu'il ne reste à étiqueter qu'un nombre fini k_T de positions dans T il est clair que T peut être r.s. $\omega + k_T$ mod 0 ce qui implique que T peut être r.s. $\omega * 2$ mod 0. Donc

$U'(\alpha) \leq \omega * 2$. De plus, il est clair pour chaque $\beta < \omega * 2$, on peut construire un α-arbre qui ne peut être r.s. β mod 0.

□

Le Lemme 4.4.3, nous permet d'exhiber des arbres qui différencient les ordinaux. A partir de ces arbres, nous savons construire des contextes *tyft* qui les "codent".

Soient deux ordinaux $\alpha_0 < \alpha'_0$, tels que $\alpha_0 \not\sim \alpha'_0$. Alors $\alpha_0 < U(\alpha'_0) = Min(\alpha, \omega_1)$ et le Lemme 4.4.3 impliquent qu'il existe un α'_0-arbre $T = (L, \rightarrow, E)$ tel que T ne peut pas être r.s. α_0 mod 0. Sans perdre de généralité, nous pouvons supposer que T est un arbre binaire complet, i.e. $L = \{0,1\}^*$. A partir de T nous construisons un contexte à une place $\mathcal{C}_T[.]$ tel que $t'_0 = \mathcal{C}_T[\alpha'_0] \not\sim_{Tr} \mathcal{C}_T[\alpha_0] = t_0$ car t'_0 possède une trace infinie, mais pas t_0.

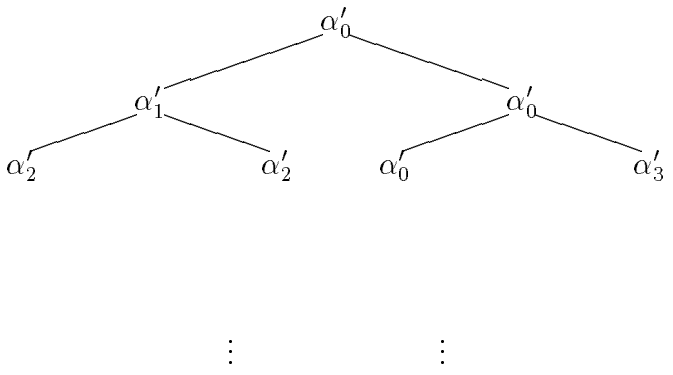


Figure 4.5: L'α'_0-arbre T

Supposons, par exemple, que T a la forme de la Figure 4.4.4, où $\alpha'_2 < \alpha'_1 < \alpha'_0$ et $\alpha'_3 < \alpha'_0$. Nous définissons la famille d'opérateurs $(O_i)_{i \in N}$ par les règles *tyft* $(R_i)_{i \in N}$ suivantes :

$$R_0 \qquad \frac{x_1 \rightarrow y_1}{O_0(x_1) \xrightarrow{a} O_1(y_1, x_1)}$$

$$R_1 \qquad \frac{x_1 \xrightarrow{a} y_1 \quad x_2 \xrightarrow{a} y_2}{O_1(x_1, x_2) \xrightarrow{a} O_2(y_1, y_1, x_2, y_2)}$$

$$R_2 \qquad \ldots$$

où l'ensemble des prémisses de R_i pour $O_i(x_1, \ldots, x_{2^i}) \xrightarrow{a} O_{i+1}(x_1, y_1, \ldots, x_{2^i}, y_{2^i})$ est donné par les lien de dépendances entre les occurrences d'ordinaux dans l'arbre T entre le niveau i et le niveau $i+1$. Par construction, seul $O_0(\alpha'_0)$ admet la trace infinie $aaa\ldots$.

Un raisonnement similaire peut être appliqué au cas des arbres à largeur bornée, dont on peut alors exhiber des contextes sans copie.

Nous pouvons ainsi montrer que l'ajout des contextes *tyft* mais aussi des contextes sans copie peut ajouter strictement du pouvoir de distinction. C'est le cas pour l'équivalence de trace $\sim_{Tr}$. Il est clair que $dp(\sim_{Tr}) = \omega$: deux entiers $n < m$ n'ont pas les mêmes traces (seul m admet une trace de longueur m). Par contre tous les ordinaux à partir de ω admettent des traces de longueur arbitrairement longues. On peut cependant déduire de ce qui précède que des ordinaux au-delà de ω peuvent être distingués par cette équivalence si on les place dans des contextes.

Proposition 4.4.4 1) *Le d.p. de la congruence de trace engendrée par les contextes* tyft *est égal à* ω_1,

2) *Le d.p. de la congruence de trace engendrée par les contextes sans copie est égal à* $\omega * 2$.

3) *La congruence de trace engendrée par les contextes* tyft *est strictement plus fine que celle engendrée par les contextes sans copie.*

Preuve Pour les contextes *tyft* (resp. sans copie) le pouvoir de distinction est borné par ω_1 (resp. $\omega * 2$) comme le montre la Proposition 4.4.1 dans le cas où γ vaut 0. Ce pouvoir de distinction atteint au moins ω_1 (resp. $\omega * 2$) comme le montre la construction des opérateurs O_i ci-dessus. □

Ces premiers résultats indiquent une réelle différence d'expressivité entre les contextes *tyft* et les contextes sans copie. Dans la section suivante nous considérons les congruences induites par les logiques CTL et CTL^* pour les deux classes de contextes *tyft* et sans copie et nous montrons que ces congruences n'atteignent pas la bisimulation.

4.4.5 Congruences induites par CTL et CTL^*

Nous introduisons les notations suivantes :

Définition 4.4.8 (Congruence induite par une logique) *Soit L une logique d'états. Nous définissons les équivalences $\simeq_L$ et $\cong_L$ entre les états d'un graphe (obtenu par une SG) par : pour tous $q, q' \in Q_G$,*

$$q \simeq_L q' \stackrel{def}{=} \text{pour tout contexte tyft } \mathcal{C}[.], \mathcal{C}[q] \sim_L \mathcal{C}[q']$$

$$q \cong_L q' \stackrel{def}{=} \text{pour tout contexte sans copie } \mathcal{C}[.], \mathcal{C}[q] \sim_L \mathcal{C}[q']$$

Clairement, $q \simeq_L q'$ implique $q \cong_L q'$. D'autre part, si L et L' sont deux logiques d'états telles que l'équivalence $\sim_{L'}$ est plus fine que l'équivalence $\sim_L$, on a

$$q \simeq_{L'} q' (\text{resp. } q \cong_{L'} q') \text{ implique } q \simeq_L q' (\text{resp. } q \cong_L q')$$

Nous étudions tout d'abord le d.p. des congruences $\simeq_{CTL}$ et $\simeq_{CTL^*}$ en montrant qu'il vaut $\omega_1 * \omega$ voir la Proposition 4.4.5. Pour cela il suffit d'établir (1) que cette valeur est une borne pour le d.p. de $\simeq_{CTL^*}$ puis (2) nous exhibons des contextes *tyft* et des formules CTL pour montrer que cette borne est atteinte.

Une démarche similaire est adoptée pour les congruences $\cong_{CTL}$ et $\cong_{CTL^*}$ montrant le résultat de la Proposition 4.4.7.

Proposition 4.4.5 $dp(\simeq_{CTL}) = dp(\simeq_{CTL^*}) = \omega_1 * \omega$

Preuve Nous définissons par induction sur les formules de CTL^* une notion de profondeur notée δ par :

$$\begin{aligned}
&\delta(\top) = 0\\
&\delta(f \wedge g) = max(\delta(f), \delta(g))\\
&\delta(\neg f) = \delta(f)\\
&\delta(X\, f) = \delta(f)\\
&\delta(f\, U\, g) = max(\delta(f), \delta(g))\\
&\delta(\forall f) = \delta(f) + \omega_1
\end{aligned}$$

Par définition de δ on a $\delta(f) \leq \omega_1 * n$ où n est le nombre maximum d'opérateurs $\forall$ emboîtés dans f. Par conséquent pour toute formule $f \in CTL^*$ $\delta(f) < \omega_1 * \omega$.

Lemme 4.4.6 *Soit G_λ un graphe ordinal. Pour toute $f \in CTL^*$, et toutes exécutions π et π' de G_λ,*

$$\pi \overset{\delta(f)}{\smile} \pi' \text{ implique } \pi \sim_f \pi'$$

Preuve On le montre par induction sur la structure de f. Les cas $\top$ $\neg f$ $f \wedge g$ sont évidents.

- $f = X\, g$: soit $\gamma = \delta(f)$. Si $\pi \models X\, g$ alors π^1 existe et satisfait g. Comme $\pi \overset{\gamma}{\smile} \pi'$ alors par définition de $\overset{\gamma}{\smile}$ sur les exécutions π'^1 existe et $\pi^1 \overset{\gamma}{\smile} \pi'^1$. Comme $\gamma = \delta(g)$ par hypothèse d'induction $\pi'^1 \models g$ donc $\pi' \models X\, g$.
- Un raisonnement analogue s'applique au cas $g\, U\, h$.
- $f = \forall g$: soit $\gamma = \delta(g)$ et $\pi \overset{\gamma+\omega_1}{\smile} \pi'$. Supposons que $\pi \not\models \forall g$ alors il existe une exécution π_0 partant de $\pi(0)$ telle que $\pi_0 \not\models g$. Mais comme $\pi(0) \overset{\gamma+\omega_1}{\smile} \pi'(0)$ d'après la Proposition 4.4.1 il existe une exécution π'_0 partant de $\pi'(0)$ telle que $\pi_0 \overset{\gamma}{\smile} \pi'_0$. Par hypothèse d'induction $\pi'_0 \not\models g$ donc $\pi' \not\models \forall g$.

□

On montre très facilement en utilisant le lemme précédent que pour toute $f \in CTL^*$ et tous ordinaux α_i, α'_i $(i = 1, \ldots, n)$

$$\alpha_i \overset{\delta(f)}{\smile} \alpha'_i \text{ pour tout } i = 1, \ldots, n \text{ implique } \mathcal{C}[\alpha_1, \ldots, \alpha_n] \sim_f \mathcal{C}[\alpha'_1, \ldots, \alpha'_n]$$

Une formule f de CTL^* ne peut donc pas distinguer des ordinaux au dessus de $\delta(f)$ même lorsqu'on les place dans des contextes *tyft*. Par conséquent on a

$$dp(\simeq_{CTL}) \leq dp(\simeq_{CTL^*}) \leq \omega_1 * \omega$$

Nous montrons maintenant que cette borne est atteinte en exhibant une suite de contextes $\mathcal{C}_n^\alpha[.]$ $(n \geq 0)$ et de formules f_n de CTL tels que que pour tout n et tout ordinal $\alpha < \omega_1$

$$\mathcal{C}_n^\alpha[\omega_1 * (n+1)] \models f_n \tag{4.6}$$

$$\mathcal{C}_n^\alpha[\beta] \not\models f_n, \textbf{ pour tout } \beta \leq \omega_1 * n + \alpha \tag{4.7}$$

Pour faciliter la présentation nous enrichissons la logique CTL par des propositions atomiques de la forme $after\,a$ où $a \in A$ et A est un ensemble de noms d'actions. La sémantique de ces propositions est donnée par :

$$q \models after\,a \text{ ssi } \text{ il existe un état } q' \text{ tel que } q' \xrightarrow{a} q$$

D'après le Lemme 4.4.3 pour tout $\alpha < \omega_1$ il existe un ω_1-arbre T^α qui ne peut être r.s. β mod 0 quel que soit $\beta \leq \alpha$ (car $U(\omega_1) = \omega_1$). On peut supposer comme en section précédente que l'arbre T^α est binaire et complet. Nous lui associons la famille d'opérateurs $\{O_{0,i}^\alpha[.]\}_{i<\omega}$ dont les règles $R_{0,i}$ décrivent les dépendances des ordinaux à chaque étage de l'arbre T^α (chaque règle $R_{0,i}$ est calquée sur la règle R_i de la page 76). Par construction des opérateurs $O_{0,i}^\alpha$ il est clair que dans le contexte $\mathcal{C}_0^\alpha[.] \stackrel{\textbf{def}}{=} O_{0,0}^\alpha(.)$ on a :

- $O_{0,0}^\alpha(\omega_1)$ n'admet qu'une exécution $aaaa\ldots$ qui est infinie en appliquant la suite de règles $R_{0,0}, R_{0,1}, \ldots$ et
- pour tout $\beta < \alpha$ $O_{0,0}^\alpha(\beta)$ n'admet qu'une exécution qui elle est finie car une des règles $R_{0,p}$ n'est plus applicable.

Nous ajoutons à ce système de règles les familles de règles suivante ;

$$R_{n,i} \quad \frac{x_1 \rightarrow y_{11}\ldots}{O_{n,i}^\beta[x_1,\ldots] \xrightarrow{a} O_{n,i+1}^\beta[y_{11}\ldots]} \qquad (0 < n < \omega, i < \omega, \beta < \omega_1)$$

$$D_n \quad \frac{x_j \rightarrow y_j}{O_{n,i}^\beta[x_1,\ldots,x_{2^i}] \xrightarrow{\surd} O_{n-1,0}^\beta[x_j]} \quad (0 < n < \omega, i < \omega, j \in \{1,\ldots,2^i\}, \beta < \omega_1)$$

Pour chaque α et chaque $n \geq 1$ les règles $R_{n,i}$ pour les opérateurs $O_{n,i}^\alpha$ sont calqués sur les règles $R_{0,i}$ des opérateurs $O_{0,i}^\alpha$ construits à partir de l'arbre T^α.
Pour chaque α les opérateurs $O_{n,i}^\alpha$ sont stratifiés selon $n \geq 0$.
Pour tout $n \geq 1$ fixé l'état $O_{n,0}^\alpha(\omega_1 * (n+1))$ admet une l'exécution infinie $aaa\ldots$ en

appliquant la suite de règles $R_{n,0}, R_{n,1}, \ldots$ (construite à partir des dépendances des occurrences d'ordinaux dans l'arbre T^α). De plus dans tout état le long de cette exécution (i.e. de la forme $O^\alpha_{n,i}(.,\ldots,.)$) et pour tout $\surd$-successeur de cet état (i.e. de la forme $O^\alpha_{n-1,0}(.)$) il existe aussi une exécution infinie $aaa\ldots$ telle que dans tout état le long de cette exécution et pour tout $\surd$-successeur etc...

Nous proposons donc les formules f_n $(n \geq 0)$ définies récursivement par :

$$f_0 \stackrel{\text{def}}{=} AG(EX\top)$$

où la modalité $AG(.)$ est une abréviation pour $\neg E\top U \neg(.)$. Elle appartient à la logique CTL. Intuitivement $AG\, f$ signifie que la formule f est globalement vraie. Par conséquent f_0 exprime qu'il n'existe pas de chemin fini.

Pour $n \geq 1$ on pose

$$f_n \stackrel{\text{def}}{=} EG\,(EX\top \wedge AX\,((after\ \surd) \Rightarrow f_{n-1})$$

où la modalité $EG(.)$ est une abréviation pour $\neg A\top U \neg(.)$. Elle appartient à la logique CTL. Intuitivement $EG\, f$ signifie qu'il existe un chemin le long duquel la formule f est toujours vraie. La connective $\Rightarrow$ est l'implication classique i.e.— $f \Rightarrow g$ est équivalente à $\neg f \vee g$.

Par construction chaque état $O^\alpha_{n,0}(\omega_1 * n + \omega_1)$ satisfait la formule f_n.

Nous montrons par induction sur $n \geq 0$ qu'un état de la forme $O^\alpha_{n,0}(\beta)$ où $\beta \leq \omega_1 * n + \alpha$ ne satisfait pas f_n. Nous avons déjà constaté que c'est le cas pour $n = 0$ (c'est l'argument de la section précédente pour les traces).

Soit $n \geq 1$. Pour un état $O^\alpha_{n,0}(\beta)$ avec $\beta \leq \omega_1 * n + \alpha$ l'application des règles $R_{n,i}$ conduit forcément à un état de la forme $O^\alpha_{n,p}(\beta_1,\ldots,\beta_{2^p})$ où l'un des β_j est strictement inférieur à $\omega_1 * (n-1)$ (puisque α ne peut étiqueté T^α à la racine par construction). Pour un tel état le $\surd$-successeur $O^\alpha_{n-1,0}(\beta_j)$ ne satisfait pas f_{n-1} par hypothèse d'induction. Par conséquent $O^\alpha_{n,0}(\beta) \not\models f_n$.

Il suffit pour conclure de poser $\mathcal{C}^\alpha_n[.] \stackrel{\textbf{def}}{=} O^\alpha_{n,0}(.)$ pour tout $n \geq 0$. Ainsi les Assertions (4.6) et (4.7) sont vérifiées. □

Une proposition similaire à la Proposition 4.4.7 permet de caractériser les congruences induites par les contextes sans copie.

Proposition 4.4.7 $dp(\cong_{CTL}) = dp(\cong_{CTL^*}) = \omega^2$

Preuve Le fait de ne considérer que des contextes sans copie nous permet d'après la Proposition 4.4.1 de définir une nouvelle notion de profondeur δ' des formules de CTL^* (moins grande que précédemment) pour laquelle le Lemme 4.4.6 reste vrai. δ' est définie comme δ dans la preuve précédente sauf que $\delta'(\forall f) = (f) + \omega * 2$ suffit. Nous pouvons énoncer :

Lemme 4.4.8 *Soit G_λ un graphe ordinal.*
Pour toute $f \in CTL^*$, et toutes exécutions π et π' de G_λ

$$\pi \overset{\delta(f)}{\smile} \pi' \text{ implique } \pi \sim_f \pi'$$

Nous omettons la preuve qui est similaire à celle du Lemme 4.4.6.

La profondeur des formules étant bornée par ω^2 nous terminons notre preuve en montrant qu'il existe une suite de contextes sans copie $(\mathcal{C}_p[.])_{p\in N}$ et de formules $(f_p)_{p\in N} \in CTL$ telles que pour tout $p \geq 0$

$$\mathcal{C}_p[\omega * (p+1)] \models f_p \textbf{ mais } \mathcal{C}_p[\omega * p] \not\models f_p \tag{4.8}$$

Pour cela nous utilisons des contextes CCS construits à partir de nil et des combinateurs $a(.)$ et $+$ définis en Section 2.2.1. Nous supposons donnée une fonction $\times : A \times A \to A$ telle que pour tous $a, a', b, b' \in A$ $a \times b = a' \times b'$ ssi $a = a'$ et $b = b'$. Nous introduisons le combinateur de composition parallèle défini par les règles sans copie suivantes :

$$\frac{x \overset{e}{\to} x'}{x \,|\, y \overset{e}{\to} x' \,|\, y} \qquad \frac{y \overset{e}{\to} y'}{x \,|\, y \overset{e}{\to} x \,|\, y'} \qquad \frac{x \overset{e}{\to} x' \;\; y \overset{e'}{\to} y'}{x \,|\, y \overset{e\times e'}{\to} x' \,|\, y'}$$

Nous supposons que les graphes ordinaux sont étiquetés sur l'alphabet $A = \{a\}$. Nous définissons les termes

- $G_0 \overset{\text{def}}{=} nil$
- $G_p \overset{\text{def}}{=} \Sigma_{n\in N}\, a^n(b(G_{p-1}))$ pour tout $p \geq 1$.

Nous posons alors

- $\mathcal{C}_0[.] \overset{\textbf{def}}{=} nil$ et $f_0 \overset{\text{def}}{=} \top$ et
- $\mathcal{C}_p[.] \overset{\textbf{def}}{=} G_p \,|\, .$ et $f_p \overset{\text{def}}{=} AX\,[E\,(after\, a \times c)\, U\, ((after\, b \times c) \wedge f_{p-1})]$ pour $p \geq 1$

On vérifie facilement l'Assertion (4.8) par induction sur p. Nous omettons cette preuve. □

La Proposition 4.4.7 implique que pour toute $L \in \{CTL, CTL^*, L_\mu\}$

$$dp(L) < dp(\cong_L) < dp(\simeq_L)$$

Corollaire 4.4.9 *L'équivalence induite par la logique $L \in \{CTL, CTL^*, L_\mu\}$ n'est ni une congruence pour les contextes* tyft *ni pour les contextes sans copie.*

4.4.6 Congruences induites par les logiques $\mathcal{C}$–TL

Nous n'avons pas étudié en détail les congruences induites par les logiques $\mathcal{C}$–TL de [BT85]. Dans le cas où l'on considère des contextesΓles graphes considérés ne sont plus forcément étiquetés sur un alphabet singleton. Il faut alors prendre en compte la définition d'origine de ces logiques. Quoiqu'il en soitΓil est semble possible de traduire leurs formules en des conjonctions ou disjonctions dénombrables de formules plus élémentaires pour lesquelles on sait définir une notion de profondeur permettant de borner le d.p. de ces logiques (par des techniques similaires à celles que nous avons utilisées).

Conclusion

Dans cette première partie de la thèse, nous avons montré comment la bisimulation et les logiques du temps arborescent sont étroitement (mais imparfaitement) liées. Dans le cadre général du branchement infini, les logiques que nous avons considérées induisent des équivalences plus faibles que la bisimulation. La démarche utilisée pour montrer nos résultats est exclusivement basée sur l'utilisation des *Processus Ordinaux* de Klop, dans le même esprit que [SP90].

On peut résumer les derniers résultats du dernier chapitre par : plus la logique utilisée est expressive (en restant compatible avec la bisimulation), et plus on considère un large jeu d'opérateurs (pour lequel la bisimulation reste une congruence), plus la congruence associée à la logique utilisée est proche de la bisimulation. Il faut noter que nous avons considéré toute la classe des combinateurs *tyft*, qui ne peut être étendue de façon triviale. De ce point de vue, notre travail est optimal et il généralise les travaux de [Sch90b] dans lesquels, seules la logique CTL et une sous-classe des combinateurs *tyft* sont considérées.

Les techniques de preuves que nous avons utilisées pour traiter les différentes logiques sont sensiblement les mêmes. La méthode consiste à trouver la "bonne" définition de la profondeur d'une formule, pour borner sa hauteur. Nous pensons que cette méthode s'applique à beaucoup d'autres logiques.

Nous aimerions interpréter ce résultat de façon plus générale, en faisant abstraction du fait que les équivalences étudiées sont induites par des logiques. En effet, le fait que le pouvoir de distinction d'une équivalence existe devrait suffir à montrer que la congruence associée (vis à vis de la classe de combinateurs *tyft* de [GV88]) admet elle aussi un pouvoir de distinction d'où nous déduirons qu'elle reste plus faible que la bisimulation. Dans cette optique, il paraît nécessaire de généraliser

aux chemins les définitions des équivalences et de leur pouvoir de distinction pour tirer avantage de la Proposition 4.4.1 qui constitue un élément clé de nos résultats.

D'autres perspectives pour généraliser nos travaux peuvent être envisagées. Par exemple en considérant le format GSOS de [BIM88] ou plus généralement le format *ntyft/ntyxt* de [Gro89] (qui autorisent des prémisses négatives dans les règles SOS *tyft*). Dans le cas du branchement fini la congruence de traces induite par les contextes *ntyft/ntyxt* coïncide avec la bisimulation. On peut alors envisager d'étudier ce format de règles dans le cadre plus général du branchement infini.

Partie II

Les modèles d'ordre partiel

Introduction

Le *raffinement* des programmes fait partie des opérations que l'on utilise naturellement pour concevoir les programmes : il consiste à remplacer une unité "élémentaire" à un niveau d'abstraction élevé par un programme plus complexe à un niveau d'abstraction moins élevé[3]. Les modèles d'entrelacement présentés en Partie 1 sont incompatibles avec le concept intuitif que l'on cherche à modéliser ou tout du moins il est nécessaire d'imposer des restrictions sur les raffinements pour que les sémantiques d'entrelacement soient compositionnelles pour cette opération. Ces considérations ont été observées par [CMP87] qui ont mis en évidence l'intérêt d'utiliser des sémantiques d'ordre partiel.

Le principe général des modèles d'ordre partiel est basé sur l'idée de Petri. On représente explicitement la dépendance causale entre les actions effectuées et par complémentarité l'indépendance entre les actions c'est à dire le parallélisme. Les modèles les plus connus sont les *traces de Mazurkievich* [AR88] les *structures d'événements* de [NPW81] les *réseaux de Petri* (voir [Rei85] pour une introduction sur les réseaux de Petri) et les *systèmes de transitions asynchrones* de [Bed87]. Dans certains de ces modèles comme par exemple les structures d'événements on représente aussi explicitement les choix non-déterministes du système.

Les *structures d'événements* [NPW81][WN92] sont des modèles de base pour les sémantiques d'ordre partiel du temps arborescent. Il existe plusieurs catégories de structures d'événements ([NPW81][DD92][BC88][Win89][NPW81][Lan92]). Dans cette thèse pour simplifier les preuves nous ne considérons que les structures d'événements les plus élémentaires c-à-d. les structures d'événements *premières*. Nous les présentons au Chapitre 5. Nous justifions le choix des structures

[3] C'est la *construction hiérarchique* de systèmes ou l'*analyse descendante* en génie logiciel.

d'événements premières par le fait qu'elles sont les modèles de base pour étudier en théorie les liens entre les sémantiques d'ordre partiel et les sémantiques du temps arborescent. Nous sommes cependant convaincus que nos résultats se généralisent aux structures d'événements *stables*.

Au Chapitre 6 nous nous plaçons dans le cadre simple des structures d'événements sans action invisible. Nous présentons les équivalences classiques qui tentent de combiner les bonnes propriétés de la bisimulation et les sémantiques d'ordre partiel bien adaptées au raffinement de programmes. Les équivalences les plus naïves comme la *pomset bisimulation* de [BC87] ne sont pas satisfaisantes.

Nous introduisons ensuite une combinaison entre bisimulation et ordre partiel particulièrement intéressante: la *history preserving bisimulation* de [GG89a]. Cette équivalence a été originellement introduite par [RT88] pour les réseaux de Petri sous le nom de *Behavior Structure bisimulation* (voir aussi la *Fully Concurrent bisimulation* de [BDKP91]). Nous rappelons que cette équivalence sémantique coïncide avec

- l'*équivalence de mixed-ordering* de [DDM89]
- l'*équivalence causale* de [DD89]
- la *pomset bisimulation avant-arrière* (sur les histoires des calculs) de [Che92a].

La pomset bisimulation avant-arrière de [Che92a] a été proposée très récemment et n'a donc pas beaucoup été étudiée. Nous montrons au Chapitre 7 comment cette équivalence permet de dériver une caractérisation de la history preserving bisimulation par une logique "à la HML" avec des modalités du passé. A notre connaissance il s'agit de la première caractérisation logique de la history preserving bisimulation. De plus nous exhibons un algorithme de traduction entre notre logique notée L_{pbf} et la logique L_P de [DF90]. Pour traduire L_P dans L_{pbf} nous utilisons une technique de séparation "à la Gabbay" des formules de L_P. Le fait que la logique L_P s'injecte dans L_{pbf} prouve qu'elle ne caractérise pas la weak history preserving bisimulation comme le prétendent [DF90].

Le Chapitre 8 étudie des variantes de la pomset bisimulation avant-arrière. Nous obtenons une classification des bisimulations avant-arrière pour les structures d'événements premières sans action invisible.

Finalement au Chapitre 9 nous étendons la pomset bisimulation avant-arrière de [Che92a] en la pomset τ-bisimulation avant-arrière qui prend en compte les actions invisibles. Nous montrons que cette nouvelle équivalence coïncide avec la branching bisimulation sur les *arbres causaux* de [DD89]. De plus nous introduisons deux nouvelles équivalences : l'*équivalence de mixed-ordering branching* et la history preserving branching bisimulation et nous montrons qu'elles coïncident avec la pomset τ-bisimulation avant-arrière.

Finalement nous étudions plus en détail la *history preserving branching bisimulation*Γ en particulier en la comparant aux versions déjà existantes de la history preserving bisimulation telles que dans [Ace91] et [Vog91].

Chapitre 5

Rappels sur les Structures d'Événements (premières)

Les structures d'événements premières[1] ont été introduites par [NPW81].

Dans ces modèles, les calculs sont modélisés par des occurrences d'actions, appelées *événements*, entre lesquelles on décrit une relation de dépendance, appelée *causalité*, qui est simplement formalisée par un ordre partiel. Une fonction d'étiquetage permet d'associer à chaque événement l'action à laquelle il correspond. De plus, on modélise le non-déterminisme par une relation entre les événements, appelée *conflit*. Cette relation est héritée entre les conséquences de deux événements en conflit[2].

5.1 Définitions

Dans toute la suite, nous supposons donné un ensemble $Act = \{a, b, c, \ldots\}$ de noms d'actions. Nous utilisons les notations de [Win89].

[1] La terminologie *première* est due au fait qu'il existe d'autres classes de structures d'événements, voir l'introduction de la Partie 2.

[2] Cette propriété est caractéristique des structures d'événements premières.

Définition 5.1.1 *Une* structure d'événements première étiquetée sur *Act est une structure* $\mathcal{E} = (E_\mathcal{E}, \leq_\mathcal{E}, \#_\mathcal{E}, l_\mathcal{E})$ *composée d'un ensemble* $E_\mathcal{E}$ *d'*événements, *d'un ordre partiel* $\leq_\mathcal{E} \subseteq E_\mathcal{E} \times E_\mathcal{E}$ *appelé relation de* causalité, *et d'une relation binaire symétrique, irréflexive* $\#_\mathcal{E} \subseteq E_\mathcal{E} \times E_\mathcal{E}$, *appelée relation de* conflit *telle que*

$$\begin{array}{ll} (1) & \leq \cap \# = \emptyset \\ (2) & \{e' \mid e' \leq e\} \text{ est fini} \\ (3) & e\#e' \leq e'' \text{ implique } e\#e'' \end{array}$$

pour tous $e, e', e'' \in E_\mathcal{E}$. *La Condition* (2) *s'appelle l'*hypothèse de causes finies, *et la Condition* (3) *exprime la propriété d'héritage de la relation de conflit.*

$l : E_\mathcal{E} \to Act$ *est la fonction d'étiquetage.*

Dans toute la suite **SE** désigne la classe des SE premières (étiquetées sur *Act*). Nous écrivons plus simplement SE au lieu de structure d'événements premières étiquetées sur *Act*.

Nous dirons que deux événements e et e' sont *concurrents*Γ noté e *co* e'Γ si $\neg(e \leq e' \vee e' \leq e \vee e\#e')$Γ c-à-d. s'ils ne sont ni causalement dépendants ni en conflit.

On dit que $\mathcal{E} \in$ **SE** est *sans-conflit* si la relation $\#_\mathcal{E}$ est vide.

Notations 7
- $\emptyset_{\mathbf{SE}}$ *désigne la SE vide.*
- *Dans la représentation graphique des SE, nous écrivons les noms des événements ainsi que leurs étiquettes selon* $e(a)$, *où* $e \in E$ *et* $a = l(e)$. *Seuls les conflits immédiats, et pas ceux obtenus par hérédité, sont indiqués. Enfin, la relation de causalité est représentée par des arcs orientés, en omettant les causes dérivées par transitivité.*

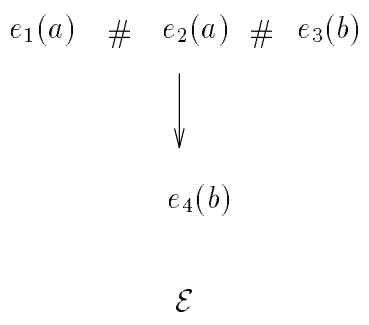


Figure 5.1: Un exemple de structure d'événements: $\mathcal{E} = a + (a; b) + b$

Exemple 11 *La Figure 5.1 représente une SE dans laquelle les événements* e_1 *et* e_3 *sont concurrents. Ces deux événements sont en conflit avec l'événement* e_2 *et,par hérédité du conflit, avec l'événement* e_4 *dont* e_2 *est la cause.*

HabituellementΓon souhaite faire abstraction du nom des événements d'une SE donnée : seule la structure sous-jacente a de l'importance. FormellementΓon définit la notion d'isomorphisme par

Définition 5.1.2 *Soient $\mathcal{E}, \mathcal{F} \in$ **SE**. Un* isomorphisme *de $\mathcal{E}$ vers $\mathcal{F}$ est une bijection $f : E_{\mathcal{E}} \to E_{\mathcal{F}}$ qui préserve les relations de causalité, de conflit et la fonction d'étiquetage, c-à-d. pour tous $e, e' \in E_{\mathcal{E}}$,*

- $e \leq_{\mathcal{E}} e'$ *ssi* $f(e) \leq_{\mathcal{F}} f(e')$
- $e \#_{\mathcal{E}} e'$ *ssi* $f(e) \#_{\mathcal{F}} f(e')$
- $l_{\mathcal{F}}(f(e)) = l_{\mathcal{E}}(e)$

5.2 Comportement des SE

Un état de calcul d'une SE est un ensemble C d'événements qui ont eu lieuΓque l'on nomme habituellementΓune *configuration*. Si un événement a eu lieuΓalors toutes ses causes doivent également avoir eu lieu. Autrement ditΓl'ensemble C est fermé à gauche pour $\leq$. De plusΓdeux événements en conflit (c-à-d. s'excluant mutuellement) ne peuvent appartenir au même état de calcul CΓalors C est sans-conflit. FormellementΓsi

Définition 5.2.1 (Configurations) *Une* configuration *de $\mathcal{E} \in$ **SE** est un ensemble $C \subseteq E_{\mathcal{E}}$ tel que*

- *(fermé à gauche): si $e \in C$ et $e' \leq e$ alors $e' \in C$, et*
- *(sans-conflit): si $e, e' \in C$ alors $\neg(e \# e')$*

Nous notons $\mathcal{C}(\mathcal{E})$ l'ensemble de toutes les configurations de $\mathcal{E}$.

L'ensemble d'événements vide $\emptyset$ est toujours une configuration de $\mathcal{E}$ qui correspond à l'état initial du système.

Remarque 8 *Toute $C \in \mathcal{C}(\mathcal{E})$ est canoniquement munie d'un ordre partiel et d'une fonction d'étiquetage - en prenant $\leq_{|C \times C}$ et $l_{|C}$ les restrictions[3] de $\leq$ et de l à C. Par conséquent, toute configuration C peut être vue comme une SE sans-conflit.*

Nous définissons à présent l'évolution d'une SEΓc-à-d. la manière dont on passe d'un état de calcul à un autreΓce qui se décrit par des transitions entre les configurations (c'est le même principe que dans les Systèmes de Transition).

Définition 5.2.2 (Transitions d'une SE) *Soit $\mathcal{E} \in$ **SE**. Une* transition *dans $\mathcal{E}$ est un triplet $(C, \theta, C') \in \mathcal{C}(\mathcal{E}) \times \mathcal{P}(E_{\mathcal{E}}) \times \mathcal{C}(\mathcal{E})$ tel que*

[3] *La notation $f_{|X}$ dénotera la restriction de la fonction f à l'ensemble X.*

- $C \subset C'$, *et*
- $\theta = C' \setminus C$

Il faut remarquer que dans la Définition 5.2.2, les configurations C et C' sont distinctes.

Nous notons $C \xrightarrow{\theta}_{\mathcal{E}} C'$ si (C, θ, C') est une transition de $\mathcal{E}$, et $C \rightarrow_{\mathcal{E}} C'$ s'il existe $\theta \subseteq E$ tel que $C \xrightarrow{\theta}_{\mathcal{E}} C'$.

Notations 8 *Pour simplifier la lecture, nous écrivons $C \xrightarrow{e}_{\mathcal{E}} C'$ au lieu de $C \xrightarrow{\{e\}}_{\mathcal{E}} C'$, et nous omettons l'indice $\mathcal{E}$ de $\rightarrow_{\mathcal{E}}$ lorsqu'il n'y a pas d'ambiguïté.*

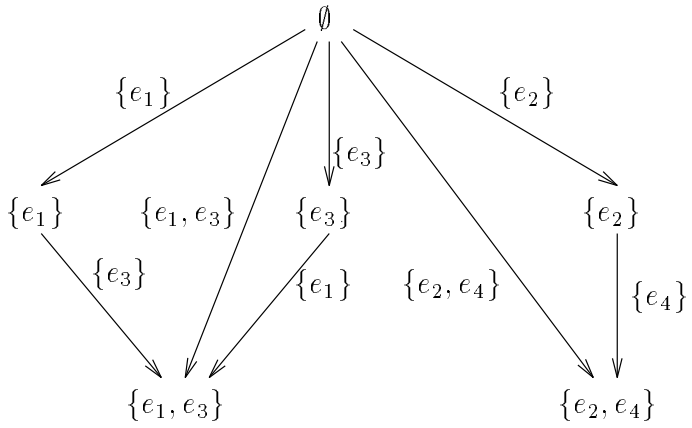


Figure 5.2: Un exemple de ST des configurations

Définition 5.2.3 (Système de transitions des configurations)

*Soit $\mathcal{E} \in$ **SE**.* Le ST des configurations *de $\mathcal{E}$ est défini par* $Conf(\mathcal{E}) \stackrel{def}{=} (\mathcal{C}(\mathcal{E}), \mathcal{P}(E_{\mathcal{E}}), \rightarrow_{\mathcal{E}}, \emptyset)$ *où $\rightarrow_{\mathcal{E}}$ est décrite en Définition 5.2.2.*

Exemple 12 *La Figure 5.2 nous montre le ST des configurations de la SE $\mathcal{E}$ de la Figure 5.1. Le système décrit par $\mathcal{E}$ se comporte de la façon suivante : il peut effectuer les événements e_1 et e_3 en parallèle par la transition $\emptyset \xrightarrow{\{e_1,e_3\}} \{e_1, e_3\}$, ainsi que les deux entrelacements de ces deux événements $\emptyset \xrightarrow{\{e_1\}} \{e_1\} \xrightarrow{\{e_3\}} \{e_1, e_3\}$ et $\emptyset \xrightarrow{\{e_3\}} \{e_3\} \xrightarrow{\{e_1\}} \{e_1, e_3\}$; il peut aussi effectuer l'événement e_2. Dans ce cas, les possibilités d'effectuer e_1 ou e_3 sont exclues. Si le système commence par e_2, l'unique événement suivant possible est e_4.*

L'évolution d'une SE étant définie, il faut maintenant définir l'observation extérieure de cette évolution. L'observation d'un événement e est son action correspondante $l_{\mathcal{E}}(e)$ définie par la fonction d'étiquetage.

Dans l'Exemple 12, l'observation du calcul "e_2 suivi de e_4" est "a suivi de b". Dans la section suivante, nous formalisons cette notion d'observation.

5.3 Observation des transitions, Pomsets

L'observation d'un calcul (ou d'une transition) est représentée par un "multiset partiellement ordonné" [Pra86]Γplus simplement appelé *pomset*[4] (ou encore "partial word dans [Gra81])Γc-à-d. la donnée d'étiquettes et d'une relation de causalité entre ces étiquettes. FormellementΓ

Définition 5.3.1 *Un* pomset[5] *est une classe d'isomorphisme de SE sans-conflit. Si $\mathcal{E}$ est sans-conflit, $pom(\mathcal{E})$ désigne sa classe d'isomorphisme, et on note $\emptyset$ le pomset vide.*

Une configuration C étant une SE sans conflitΓsi on abstrait le nom des événements de CΓl'objet obtenu est le multiset des actions correspondant aux événementsΓ reliées par la relation de causalité. De façon généraleΓpour une transition $C \xrightarrow{\theta}_{\mathcal{E}} C'$Γ la classe d'isomorphisme $pom(\theta)$ de θ correspond à l'observation du calcul effectué lors de ce changement d'état.

Notations 9

- *Dans la suite, $a;b$ dénote la classe d'isomorphisme de la SE sans conflit $\mathcal{E} = (\{e_1, e_2\}, \{(e_1, e_2), (e_2, e_1)\}, \emptyset, \{(e_1, a), (e_2, b)\})$. C'est donc un pomset qui décrit un calcul dans lequel l'action a est une cause de l'action b.*
- *$a \,|\, b$ dénote la classe d'isomorphisme de la SE d'ensembles d'événements $E_{\mathcal{E}} = \{e_1, e_2\}$, sans lien de causalité entre eux, et d'étiquettes respectives a et b. Le pomset $a \,|\, b$ exprime que les actions a et b ont lieu en parallèle. Nous renvoyons à la Section 5.4 pour la définition de l'opérateur $|$ dans les SE.*

Nous utiliserons les lettres $p, p', q, q', \ldots$ pour désigner les pomsetsΓ et nous noterons $C \xrightarrow{p} C'$ si $C \xrightarrow{\theta} C'$ avec $pom(\theta) = p$.

Exemple 13 *Considérons le ST des configurations de la SE, Figure 5.2. On peut dériver de ce ST, un ST dans lequel les transitions sont étiquetées par les pomsets. Nous obtenons alors le graphe de la Figure 5.3.*

Certains pomsets présentent des caractéristiques particulièresΓcomme par exemple de ne contenir aucun lien de causalité entre ses éléments. On les appelle des *step* (ou encore *pas*). Par exemple "$a \,|\, b \,|\, c$" est un step. L'origine du mot "step" et due à la théorie des réseaux de Petri ; il dénote un ensemble (ou un multi-ensemble) de transitions exécutables en parallèle. Un pomset de cardinal 1Γc-à-d. une actionΓ est un cas particulier de step.

Dans la suiteΓles lettres $s, s', \ldots$ désignent des step.

[4] *pomset* vient de l'anglais pour "partially ordered multiset".

[5] *Pomset signifie* partially ordered multiset.

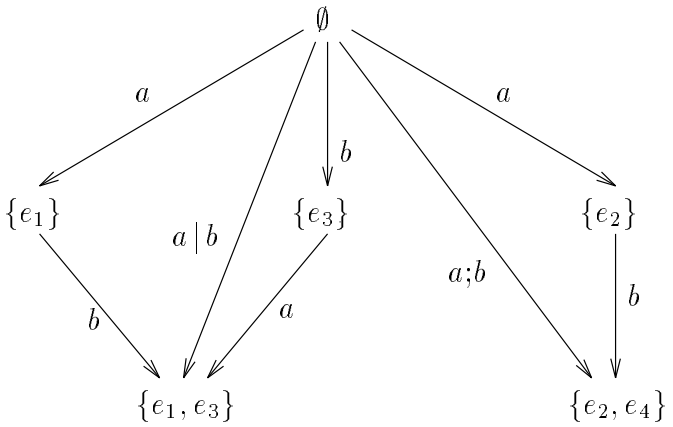


Figure 5.3:

5.4 Opérations sur les SE, raffinement d'actions

Nous ne décrivons que brièvement deux opérations sur les SE : celle de composition parallèle et celle de choix-déterminisme, qui sont utiles pour nos exemples.

Nous renvoyons par exemple à [Win89] qui propose une sémantique dénotationnelle dans les SE pour un langage de programmes parallèles.

Dans ce qui suit, si A et B sont des ensembles, $A \uplus B$ dénote l'union disjointe de A et B.

Nous supposons données $\mathcal{E}, \mathcal{F} \in SE$.

La mise en parallèle de $\mathcal{E}$ et $\mathcal{F}$, notée $\mathcal{E} \,|\, \mathcal{F}$, est définie par

$$\mathcal{E} \,|\, \mathcal{F} \stackrel{\text{def}}{=} (E_\mathcal{E} \uplus E_\mathcal{F}, \leq_\mathcal{E} \uplus \leq_\mathcal{F}, \#_\mathcal{E} \uplus \#_\mathcal{F}, l_\mathcal{E} \uplus l_\mathcal{F})$$

de sorte que tous les éléments de $E_\mathcal{E}$ sont en parallèle avec tous ceux de $E_\mathcal{F}$.

Le choix non-déterministe entre $\mathcal{E}$ et $\mathcal{F}$, noté $\mathcal{E} + \mathcal{F}$ est défini par

$$\mathcal{E} + \mathcal{F} \stackrel{\text{def}}{=} (E_\mathcal{E} \uplus E_\mathcal{F}, \leq_\mathcal{E} \uplus \leq_\mathcal{F}, \#_{\mathcal{E}+\mathcal{F}}, l_\mathcal{E} \uplus l_\mathcal{F})$$

où $\#_{\mathcal{E}+\mathcal{F}} \stackrel{\text{def}}{=} \#_\mathcal{E} \uplus \#_\mathcal{F} \uplus (E_\mathcal{E} \times E_\mathcal{F}) \uplus (E_\mathcal{F} \times E_\mathcal{E})$.

Ainsi, le système représenté par $\mathcal{E} + \mathcal{F}$ n'évolue que comme $\mathcal{E}$ ou que comme $\mathcal{F}$.

Nous présentons maintenant l'opération de *raffinement d'actions* [CMP87] dans les SE (premières) en utilisant la définition de [GG89b]. Cette opération permet de descendre le niveau de description en remplaçant une action soit par une séquence de sous-actions, soit par des sous-actions s'exécutant en parallèle, soit par un pomset quelconque. Il est par contre exclu de remplacer a par une structure contenant

des conflits car en général la SE obtenue n'est plus première. Ce problème technique peut être surmonté mais la formalisation d'un tel raffinement étant très lourde on a plutôt intérêt à se placer dans une classe plus large de SE par exemple les SE *à flot* de [BC88]. Nous renvoyons à [DG92] pour un exemple de sémantique dénotationnelle et opérationnelle d'un langage simple dans les SE *à flot* (voir aussi [GG89b] pour la définition formelle du raffinement dans les SE *à flot* et dans les réseaux de Petri).

Nous n'autorisons pas non plus les raffinements dits *d'oubli* qui remplacent certaines actions par la SE vide $\emptyset_{\mathbf{SE}}$.

Un raffinement d'action est formellement défini comme une fonction *ref*[6] spécifiant pour chaque action a une SE $ref(a)$ que l'on substitue à a. Dans la pratique on ne raffine que certaines actions ; cependant pour simplifier la définition nous considérons comme dans [GG89b] des raffinements où toutes les actions sont raffinées certaines ne l'étant que par elles-mêmes.

Soient $\mathcal{E} \in \mathbf{SE}$ et *ref* une fonction de raffinement. La SE $ref(\mathcal{E})$ est alors construite en remplaçant chaque événement $e \in E_{\mathcal{E}}$ d'étiquette a par une copie disjointe $\mathcal{E}_e$ de $ref(a)$. Les relations de causalité et de conflit sont alors héritées de $\mathcal{E}$.

La définition qui suit est empruntée à [GG89b].

Définition 5.4.1 (Raffinement d'actions) *Une fonction* $ref : Act \to \mathbf{SE}$ *est un* raffinement d'actions *si pour toute action* $a \in Act$, $ref(a)$ *est non vide, finie et sans conflit.*

Soient $\mathcal{E} \in \mathbf{SE}$ *et ref un raffinement d'actions, alors le SE* $ref(\mathcal{E})$ *est définie par :*

- $E_{ref(\mathcal{E})} \stackrel{def}{=} \{(e, e') | e \in E_{\mathcal{E}}, e' \in E_{ref(l_{\mathcal{E}}(e))}\}$,
- $(d, d') \leq_{ref(\mathcal{E})} (e, e')$ *ssi* $d < e$ *ou* $(d = e$ *et* $d' \leq_{ref(l_{\mathcal{E}}(d))} e')$,
- $(d, d') \#_{ref(\mathcal{E})} (e, e')$ *ssi* $d \#_{\mathcal{E}} e$,
- $l_{ref(\mathcal{E})}(e, e') = l_{ref(l_{\mathcal{E}}(e))}(e')$.

Nous renvoyons à [GG89b] où il est montré que le raffinement est une opération bien définie sur **SE** (premières) et que l'isomorphisme entre SE est une congruence pour cette opération.

[6] "*ref*" provient de "refinement of actions" en anglais.

Chapitre 6

Equivalences comportementales sans action invisible

Les équivalences sémantiques qui ont été étudiées sur les SE cherchent à combiner les bonnes propriétés de la bisimulation et les sémantiques d'ordre partiel qui s'adaptent bien au raffinement d'actions.

Les motivations pour définir des équivalences sémantiques *compatibles avec le raffinement d'actions* (on dit aussi *préservées par le raffinement d'actions*) sont multiples. Ces équivalences s'appliquent dans le cas où les programmes sont conçus à différents niveaux d'abstraction car elles sont insensibles au niveau d'atomicité des actions.

Les équivalences où l'on combine "naïvement" bisimulation et sémantique d'ordre partiel (e.g. [BC87]) ne possèdent pas les propriétés attendues. Les travaux de [GG89a] ont permis d'exhiber pour les SE sans action invisible une combinaison particulièrement intéressante : la *history preserving bisimulation.*

A l'origine cette équivalence a été proposée par [RT88] pour les réseaux de Petri sous le nom de *Behavior Structure bisimulation* (ou encore *BS-bisimulation*). [GG89a] ont rephrasé la définition pour les SE (premières) et ont prouvé sa compatibilité avec le raffinement d'actions[1].

[1] [BDKP91] généralisent ce résultat dans les réseaux de Petri

Ce chapitre est consacré à introduire la history preserving bisimulation qui est l'équivalence principalement étudiée dans les chapitres suivants de cette thèse.

Après avoir rappelé quelques équivalences de la littérature ainsi que leurs défauts majeursΓnous considérons la history preserving bisimulationΓpuis nous rappelons qu'elle coïncide avec l'équivalence de mixed-ordering de [DDM89]Γla pomset bisimulation avant-arrière de [Che92a] et l'équivalence causale de [DD89].

6.1 Bisimulations d'ordre partiel

La première équivalence basée à la fois sur les ordres partiels et sur la bisimulation est la *bisimulation de pomset* (ou encore *pomset bisimulation*) de [BC87]. Elle correspond à la combinaison la plus directe entre la bisimulation et les ordres partiels. IntuitivementΓcette équivalence identifie deux SE dont les ST des configurationsΓ étiquetés par les pomsetsΓsont fortement bisimilaires.

Deux variantes affaiblies de l'équivalence de [BC87] peuvent être définies en exigeant une propriété de transfert dans laquelle les transitions à imiter sont étiquetées par des steps ou des par actions. On parlera alors de *bisimulation de step* et de *bisimulation d'action* (ou de *step bisimulation* et de *bisimulation interleaving*).

Définition 6.1.1 (Bisimulations de pomset, de step et d'action) *Soient $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$. Une relation symétrique $R \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{F}) \cup \mathcal{C}(\mathcal{F}) \times \mathcal{C}(\mathcal{E})$ est une* bisimulation de pomset *(resp.* de step, d'action*) entre $\mathcal{E}$ et $\mathcal{F}$ si $(\emptyset, \emptyset) \in R$ et pour tout $(C, D) \in R$ et tout pomset (resp. step, action) θ, si $C \xrightarrow{\theta} C'$, alors il existe $D \xrightarrow{\theta'} D'$ tel que $pom(\theta) = pom(\theta')$ et $(C', D') \in R$.*

On note $R : \mathcal{E} \approx_p \mathcal{F}$ (resp. $\approx_s$, $\approx_i$) si R est une bisimulation de pomset (resp. step, action) entre $\mathcal{E}$ et $\mathcal{F}$.

On note $\mathcal{E} \approx_p \mathcal{F}$ si $R : \mathcal{E} \approx_p \mathcal{F}$ pour une relation R, et de même pour les équivalences $\approx_s$ et $\approx_i$.

On vérifie facilement que $\approx_p$Γ$\approx_s$ et $\approx_i$ sont des relations d'équivalence sur **SE**.

L'équivalence $\approx_i$ généralise la bisimulation forte des modèles d'entrelacement.

Exemple 14 *Les SE de la Figure 6.4 et de la Figure 6.1 sont pomset équivalentes [GG89a].*

Il est clair que tous $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$Γ$\mathcal{E} \approx_p \mathcal{F}$ implique $\mathcal{E} \approx_s \mathcal{F}$ implique $\mathcal{E} \approx_i \mathcal{F}$.

De plus ces implications sont strictes comme le montrent les Figures 6.3 et 6.2.

L'équivalence $\approx_p$ est largement criticable : [GV87] remarque qu'elle ne respecte pas la combinaison entre la causalité et l'arborescence de choix ; en effetΓdans la

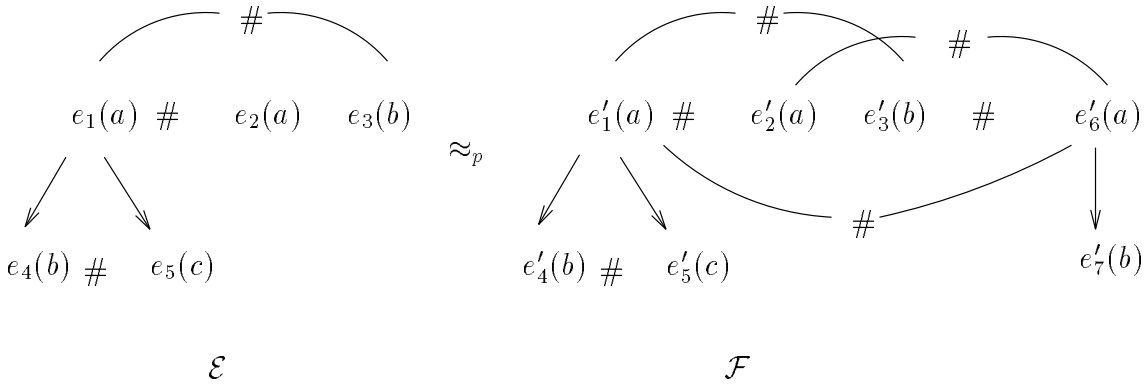


Figure 6.1: Un exemple de SE pomset bisimilaires

Figure 6.2: $\approx_i$ n'implique pas $\approx_s$. $\mathcal{E} = a \,|\, a\Gamma\mathcal{F} = a;a$

Figure 6.1Γseul le premier système a la propriété que toute action a qui précède causalement une action b précède également un choix entre cette action b et une action c.

Ces premières lacunes grossières de $\approx_p$ ont été corrigées par [GV89] qui proposent la *pomset bisimulation généralisée*. MalheureusementΓni $\approx_p$ ni sa version généralisée ne sont préservées par le raffinement d'actionsΓau sens où l'on peut exhiber deux SE équivalentes qui ne le sont plus lorsqu'on les raffine. Pour le cas de l'équivalence $\approx_p$Γ nous renvoyons au cas de l'exemple de la Figure 6.1 dans lequel on raffine l'action a par $a_1;a_2$ et pour l'équivalence pomset généraliséeΓnous signalons l'Exemple 4.2 page 225 de [Gla90a].

Nous définissons formellement la propriété "d'être préservée par le raffinement d'actions" par

Définition 6.1.2 *Soit $\sim$ une relation d'équivalence sur* ***SE****. On dit que $\sim$ est* préservée par le raffinement d'actions *si pour toute fonction de raffinement d'actions*

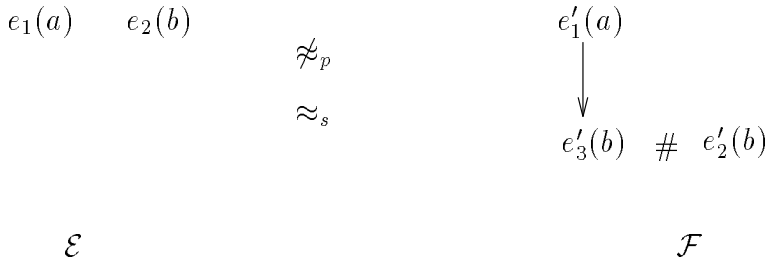


Figure 6.3: $\approx_s$ n'implique pas $\approx_p$

ref et pour toutes $\mathcal{E}, \mathcal{F} \in \boldsymbol{SE}$,

$$\mathcal{E} \sim \mathcal{F} \textit{ implique } \textit{ref}(\mathcal{E}) \sim \textit{ref}(\mathcal{F})$$

Notons qu'il existe une définition plus générale de la propriété "d'être préservé par le raffinement" : au lieu d'appliquer le même raffinement à $\mathcal{E}$ et $\mathcal{F}$Γon applique deux raffinements *ref* et *ref'* équivalents pour $\sim$. Deux raffinements *ref* et *ref'* sont équivalents pour $\sim$ si pour toute action aΓon a $\textit{ref}(a) \sim \textit{ref}'(a)$.

Dans la section suivante nous introduisons la *history preserving bisimulation.* Cette équivalence a l'avantage incontestable d'être compatible avec le raffinementΓ tout en étant contenu dans la pomset bisimulation. C'est en ce sens qu'elle a de bonnes propriétés de base.

6.2 La history preserving bisimulation

Une history preserving bisimulation entre $\mathcal{E}$ et $\mathcal{F}$ est une relation entre les configurations de $\mathcal{E}$ et $\mathcal{F}$ qui met en jeu un isomorphisme entre les configurations reliées. Pour chaque événement d'une des configurationsΓcet isomorphisme indique l'événement correspondant dans l'autre configuration par lequel il a été imité dans l'histoire du calcul. Cette histoire n'est pas explicite dans la définition. FormellementΓ

Définition 6.2.1 (History preserving bisimulation) *Soient* $\mathcal{E}, \mathcal{F} \in \boldsymbol{SE}$. *Une relation symétrique*[2] $R \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{F}) \times \mathcal{P}(E_\mathcal{E} \times E_\mathcal{E}) \bigcup \mathcal{C}(\mathcal{F}) \times \mathcal{C}(\mathcal{E}) \times \mathcal{P}(E_\mathcal{F} \times E_\mathcal{E})$ *est une* history preserving bisimulation (h.p.b.) *entre* $\mathcal{E}$ *et* $\mathcal{F}$, *noté* $R : \mathcal{E} \approx_{hp} \mathcal{F}$, *si* $(\emptyset_\mathcal{E}, \emptyset_\mathcal{F}, \emptyset) \in R$ *et si pour tout* $(C, D, f) \in R$,

- $f : C \to D$ *est un isomorphisme, et*
- *si* $C \xrightarrow{e} C'$, *il existe* D', f' *tels que* $D \to D'$, $f'_{|C} = f$ *et* $(C', D', f') \in R$.

On note $\mathcal{E} \approx_{hp} \mathcal{F}$ *lorsque* $R : \mathcal{E} \approx_{hp} \mathcal{F}$ *pour une relation* R, *et on dit que* $\mathcal{E}$ *et* $\mathcal{F}$ *sont* history preserving bisimilaires.

[2] *"symétrique" signifie* $(C, D, f) \in R$ *implique* $(D, C, f^{-1}) \in R$

Il est facile de vérifier que $\approx_{hp}$ est une relation d'équivalence sur **SE**.

La propriété de transfert de la Définition 6.2.1 ne porte pas sur des transitions quelconques mais étiquetées par des actions; il est cependant facile de démontrer que les deux variantes (transitions d'action ou transition quelconque) coïncident. Notre choix n'est ici motivé que par des considérations techniques.

En revanche la clause qui exige que l'isomorphisme f' soit une extension de f est fondamentale. Observons au passage que c'est justement cette clause qui permet de prouver que les deux variantes susmentionnées sont équivalentes. Lorsque cette clause d'extension de l'isomorphisme est retirée de la définition nous obtenons alors la *weak history preserving bisimulation* de [GG89a] ou *NMS partial ordering* de [DDM87]. Cette équivalence qui contient $\approx_{hp}$ est alors incomparable avec $\approx_p$ et en général elle n'est pas préservée par le raffinement (voir par exemple [GG89a]).

La history preserving bisimulation est une équivalence très fine mais toutefois distincte de l'isomorphisme comme le montre l'Exemple 15. Cet exemple est souvent désigné sous le nom de "Loi d'absorption".

Notons que [GKP92] ont récemment défini une équivalence plus fine que $\approx_{hp}$ qui en particulier distingue les SE de l'Exemple 15. Mais pour cet équivalence [GKP92] ne proposent aucun exemple non trivial.

Exemple 15 *([Gla90a] page 228)*

$$(a \,|\, (b+c)) + ((a+c) \,|\, b) \approx_{hp} (a \,|\, (b+c)) + ((a+c) \,|\, b) + (a \,|\, b)$$

La "branche" $a \mid b$ n'est présente que dans la SE de droite. Si la transition à imiter provient de cette branche, il y a 3 cas à considérer, dont deux symétriques : la transition de la forme $\xrightarrow{a}$ (et symétriquement $\xrightarrow{b}$) et la transition de la forme $\xrightarrow{a\not b}$. Dans le dernier cas, la transition peut être indifféremment imitée par l'une des branches $(a \mid (b+c))$ ou $((a+c) \mid b)$. Par contre la transition $\xrightarrow{a}$ (resp. $\xrightarrow{b}$) doit obligatoirement être imitée par la branche $(a+c) \mid b$ (resp. $a \mid (b+c)$) afin exclure toute possibilité d'exécuter une transition $\xrightarrow{c}$ qui n'aurait pas de correspondant dans la branche $a \mid b$.

Proposition 6.2.1 [GW89b] *Pour toutes $\mathcal{E}, \mathcal{F} \in$ **SE**,*

$$\mathcal{E} \approx_{hp} \mathcal{F} \text{ implique } \mathcal{E} \approx_p \mathcal{F}$$

Nous omettons la preuve qui est simple. Elle est basée sur l'assertion :
si $R : \mathcal{E} \approx_{hp} \mathcal{F}$ alors $\{(C,D) | (C,D,f) \in R\} : \mathcal{E} \approx_p \mathcal{F}$
Nous renvoyons à [GW89b] pour les détails; il y est également montré que la weak history preserving bisimulation et $\approx_p$ ne sont pas comparables.

La Figure 6.4 nous montre que l'implication de la Proposition 6.2.1 est stricte. En effet la transition $\emptyset_{\mathcal{F}} \xrightarrow{a}_{\mathcal{F}} \{e_1''\}$ de $\mathcal{E}$ doit être imitée par $\emptyset_{\mathcal{E}} \xrightarrow{a}_{\mathcal{E}} \{e_1\}$. Si maintenant

$\mathcal{E}$ effectue $\{e_1\} \xrightarrow{b}_{\mathcal{E}} \{e_1, e_3\}$, seule la transition $\{e_1''\} \xrightarrow{b}_{\mathcal{F}} \{e_1'', e_2'\}$ peut l'imiter, mais alors les configurations $\{e_1, e_3\}$ et $\{e_1'', e_2'\}$ ne sont pas isomorphes.

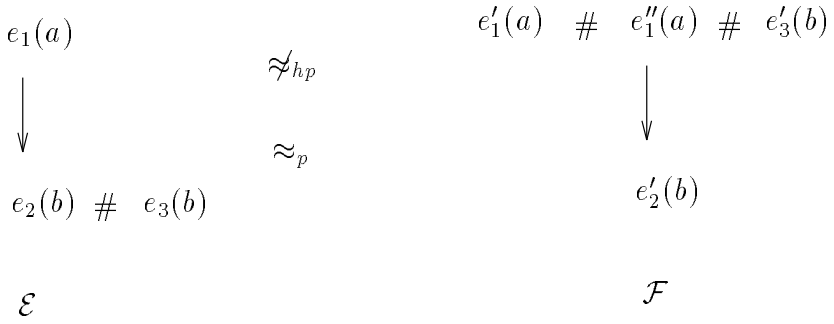


Figure 6.4: $\approx_p$ n'implique pas $\approx_{hp}$

6.3 Les caractérisations de $\approx_{hp}$

La Définition 6.2.1 de la history preserving bisimulation peut parfois sembler lourde et peu intuitive. En réalité, cette équivalence est très naturelle, surtout si on considère les quatre formulations de cette dernière qui existent dans la littérature. Dans cette section, nous présentons d'autres alternatives pour définir $\approx_{hp}$. Nous établissons également l'égalité entre ces multiples définitions.

Plus précisement, nous considérons l'équivalence de mixed-ordering, $\approx_{mo}$, de [DDM89], l'*équivalence causale*, $\approx_c$, de [DD89], et la pomset bisimulation avant-arrière, $\approx_{pbf}$, de [Che92a].

Dans [Vaa89a], il est montré que $\approx_{hp}$ et $\approx_c$ coïncident dans les SE premières, puis dans le cadre plus général des SE stables avec actions invisibles, [Ace91] prouve que $\approx_{hp}$, $\approx_c$ et $\approx_{mo}$ sont égales. Enfin, nous renvoyons à [Che92a] où $\approx_{hp}$ est caractérisée par une bisimulation avant-arrière [Che92a] le long des histoires du calcul. Dans cette thèse, cette équivalence s'appelle la *pomset bisimulation avant-arrière* et est notée $\approx_{pbf}$.

6.3.1 L'équivalence de mixed-ordering

Considérons une relation $R : \mathcal{E} \approx_{hp} \mathcal{F}$ et un triplet $(C, D, f) \in R$. L'isomorphisme f est une manière de désigner pour chaque événement de la configuration C, par quel événement de la configuration D il a été imité dans l'histoire du calcul qui a mené à C et D.

L'*histoire*[3] du calcul d'un système consiste en la liste des événements qui ont eu

[3] On trouvera aussi dans la littérature le nom de *trace* ou *séquence d'événements*.

lieu dans leur ordre d'apparition. Plus formellement une histoire est alors représentée par un ordonnancement d'événements (ayant eu lieu) c-à-d. un ordre linéaire et cet ordre est bien entendu cohérent avec l'ordre partiel de causalité.

Définition 6.3.1 (Histoires et exécutions) *Une* histoire *de $\mathcal{E}$ est une séquence $\sigma = e_1 \dots e_n \in E_{\mathcal{E}}^*$ telle que $\emptyset_{\mathcal{E}} \stackrel{e_1}{\rightarrow}_{\mathcal{E}} \{e_1\} \stackrel{e_2}{\rightarrow}_{\mathcal{E}} \dots \stackrel{e_n}{\rightarrow}_{\mathcal{E}} \{e_1, \dots, e_n\} \stackrel{def}{=} C_\sigma$.*

$Hist(\mathcal{E})$ désigne l'ensemble des histoires de $\mathcal{E}$, et $\lambda_{\mathcal{E}}$ l'histoire vide.

Une histoire π est une exécution *si π est une histoire maximale, c-à-d. que la configuration C_π est un état terminal dans $Conf(\mathcal{E})$.*

Les lettres $\sigma, \sigma', \rho, \rho', \dots$ désignent les histoires, alors que $\pi, \pi', \eta, \dots$ désignent les exécutions.

Les histoires correspondent finalement à des chemins dans le ST des configurations. Ceci nous permet dans la suite d'utiliser toutes les notations introduites en Section 1.2 pour les chemins et les exécutions.

L'isomorphisme f dans la Définition 6.2.1 doit par définition pouvoir s'étendre au fur et à mesure que les deux systèmes s'imitent tout comme l'histoire du calcul est une extension de l'histoire précédente. C'est en ce sens que l'isomorphisme préserve l'histoire. La history preserving bisimulation peut ainsi être reformulée en termes des histoires de calcul des SE et de leur possibilités d'évolution. C'est précisément ce qu'exprime l'équivalence de mixed-ordering.

L'équivalence de mixed-ordering de [DDM89] est basée sur une relation entre les histoires de SE. Elle exprime une propriété de nature statique : deux histoires reliées ont la même structure et une propriété dynamique qui garantit que les deux systèmes ont les mêmes possibilités d'évolution.

Notations 10 *Si $\sigma \in Hist(\mathcal{E})$ et $\rho \in Hist(\mathcal{F})$ sont telles que $\mid\sigma\mid = \mid\rho\mid = n$, alors f_σ^ρ dénote la fonction de graphe $\{(\sigma(i), \rho(i)) \mid 1 \leq i \leq n\}$.*

Définition 6.3.2 (Equivalence de mixed-ordering)

Une (équivalence de) mixed-ordering *entre $\mathcal{E}$ et $\mathcal{F}$ est une relation symétrique $R \subseteq Hist(\mathcal{E}) \times Hist(\mathcal{F}) \bigcup Hist(\mathcal{F}) \times Hist(\mathcal{E})$, notée $R : \mathcal{E} \approx_{mo} \mathcal{F}$, telle que :*

- $\lambda_{\mathcal{E}} R \lambda_{\mathcal{F}}$,
- *pour tous $\sigma R \rho$,*
  - $f_\sigma^\rho : C_\sigma \rightarrow C_\rho$ *est un isomorphisme,*
  - *si $\sigma' = \sigma.e \in Hist(\mathcal{E})$, il existe $\rho' = \rho.e' \in Hist(\mathcal{F})$ tel que $\sigma' R \rho'$*

Nous notons $\mathcal{E} \approx_{mo} \mathcal{F}$, lorsqu'il existe une équivalence de mixed-ordering entre $\mathcal{E}$ et $\mathcal{F}$.

Il est facile de vérifier que la relation $\approx_{mo}$ est une relation équivalence sur **SE**.

Observons que l'isomorphisme f^{ρ}_{σ} de la Définition 6.3.2 est précisément le même que celui de la Définition 6.2.1Γà la différence près qu'il est cette fois complètement déterminé. Ceci n'est pas étonnant puisque l'on dispose explicitement des histoires σ et ρ qui ont menées aux configurations C_σ et C_ρ. On peut ainsi établir

Proposition 6.3.1 *Pour tous* $\mathcal{E}, \mathcal{F} \in$ ***SE****,*

$$\mathcal{E} \approx_{hp} \mathcal{F} \text{ ssi } \mathcal{E} \approx_{mo} \mathcal{F}$$

Preuve La preuve est basée sur les deux assertions suivantes (dont nous omettons la preuve) :

- si $R : \mathcal{E} \approx_{mo} \mathcal{F}$Γalors $S \stackrel{\text{def}}{=} \{(C_\sigma, C_\rho, f^{\rho}_{\sigma}) | \sigma R \rho\}$ est une history preserving bisimulation.
- si $S : \mathcal{E} \approx_{hp} \mathcal{F}$Γalors $R \stackrel{\text{def}}{=} \{(\sigma, \rho) | (C_\sigma, C_\rho, f^{\rho}_{\sigma}) \in S\}$ est une équivalence de mixed-ordering.

□

Il faut remarquer que si deux histoires σ et ρ sont reliées par une équivalence de mixed-orderingΓil est possible de mettre en évidence des propriétés très intéressantes sur le passé du calcul.
Soient donc σ et ρ deux histoires équivalentes. On peut montrer que si l'on remonte au même moment dans les histoires σ et ρΓles pomsets calculés par les deux SE entre ce moment passé et le moment présent sont égaux. Dans la suiteΓnous dirons que "l'équivalence de mixed-ordering préserve les pomsets en arrière"; ce qu'exprime le Lemme 6.3.2 qui suit.

Lemme 6.3.2 *Soient* $R : \mathcal{E} \approx_{mo} \mathcal{F}$ *et* $\sigma.\omega R \rho.\omega'$. *Si* $|\omega| = |\omega'|$, *alors* $pom(\omega) = pom(\omega')$.

Preuve La preuve est immédiate par définition d'une équivalence de mixed-ordering : en effetΓsi $\sigma.\omega R \rho.\omega'$Γalors par définition $f^{\rho.\omega'}_{\sigma.\omega}$ est un isomorphismeΓet en particulierΓsa restriction à ω est aussi un isomorphisme dont le codomaine est évidemment ω'. □

L'équivalence de mixed-ordering préservant les pomsets en arrièreΓil semble assez naturel de proposer une nouvelle définition pour cette équivalence dans laquelle la clause sur l'isomorphisme serait remplacée par une clause de propriété de transfert pour les transitions en arrièreΓle long de l'histoire des calculs. En procédant ainsiΓ nous obtenons la définition de la pomset bisimulation avant-arrière de [Che92a].

6.3.2 Pomset bisimulation avant-arrière

Le principe est similaire à l'approche développée par [DMV90] qui introduisent les "back and forth bisimulations" (que nous traduisons de l'anglais par *bisimulations avant-arrière*) dans le cadre des systèmes de transitions. Les bisimulations avant-arrière sont des relations entre les histoires. Elles possèdent une propriété de transfert pour les transitions vers l'avant l'on dira aussi "transitions en avant" mais aussi pour les transitions "en arrière" le long des histoires. Remarquons que les transitions en arrière sont déterministes.

Notations 11 *Par abus de notation, nous notons* $\sigma \xrightarrow{p}_{\mathcal{E}} \sigma'$ *lorsque* $C_\sigma \xrightarrow{p}_{\mathcal{E}} C_{\sigma'}$.

Définition 6.3.3 (Pomset bisimulation avant-arrière) *Soient* $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$. *Une relation symétrique* $R \subseteq Hist(\mathcal{E}) \times Hist(\mathcal{F}) \bigcup Hist(\mathcal{F}) \times Hist(\mathcal{E})$ *est une* pomset bisimulation avant-arrière *entre* $\mathcal{E}$ *et* $\mathcal{F}$, *noté* $R : \mathcal{E} \approx_{pbf} \mathcal{F}$ ($\approx_{pbf}$ *signifie "pomset back and forth"), si*

- $\lambda_{\mathcal{E}} R \lambda_{\mathcal{F}}$, *et*
- *pour tous* $\sigma R \rho$,
  - *(en avant): si* $\sigma \xrightarrow{a} \sigma'$ *alors il existe* $\rho \xrightarrow{a} \rho'$ *tel que* $\sigma' R \rho'$,
  - *(en arrière): si* $\sigma' \xrightarrow{p} \sigma$ *alors il existe* $\rho' \xrightarrow{p} \rho$ *tel que* $\sigma' R \rho'$.

On note $\mathcal{E} \approx_{pbf} \mathcal{F}$ *si il existe une relation* R *telle que* $R : \mathcal{E} \approx_{pbf} \mathcal{F}$.

On vérifie facilement que $\approx_{pbf}$ est une relation d'équivalence sur **SE** et qu'il existe toujours une plus grande pomset bisimulation avant-arrière entre deux SE.

Notons que la Définition 6.3.3 est différente de celle de [Che92a] dans laquelle les clauses "en avant" et "en arrière" portent sur des transitions étiquetées par des pomsets. Toutefois comme pour le cas des équivalences $\approx_{hp}$ et $\approx_{mo}$ il suffit d'exiger la propriété de transfert "en avant" pour des transitions étiquetées par des actions (Nous renvoyons à la Proposition 8.2.2 du Chapitre 8).

Exemple 16 *Dans la Figure 6.4, les SE* $\mathcal{E}$ *et* $\mathcal{F}$ *ne sont pas* $\approx_{pbf}$*-équivalentes. En effet, supposons qu'il existe* $R : \mathcal{E} \approx_{pbf} \mathcal{F}$. *Alors en particulier* $\lambda_{\mathcal{E}} R \lambda_{\mathcal{F}}$ *et la transition* $\lambda_{\mathcal{F}} \xrightarrow{a}_{\mathcal{F}} e_1''$ *ne peut être imitée dans* $\mathcal{E}$ *que par* $\lambda_{\mathcal{E}} \xrightarrow{a}_{\mathcal{E}} e_1$. *Puis la transition* $e_1 \xrightarrow{b}_{\mathcal{E}} \sigma = e_1.e_3$, *forcément imitée dans* $\mathcal{F}$ *par* $e_1'' \xrightarrow{b}_{\mathcal{F}} \rho = e_1''.e_2'$, *nous conduit à une contradiction puisque pour les transitions en arrière* $\lambda \xrightarrow{a\|b}_{\mathcal{E}} \sigma$ *et* $\lambda \xrightarrow{a;b}_{\mathcal{F}} \rho$ *nous observons des pomsets différents.*

Proposition 6.3.3 *Pour tous* $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$,

$$R : \mathcal{E} \approx_{mo} \mathcal{F} \text{ ssi } R : \mathcal{E} \approx_{pbf} \mathcal{F}$$

Preuve $\Rightarrow$) : Immédiate d'après la Proposition 6.3.2.
$\Leftarrow$) : Nous reprenons l'idée de la preuve de [Che92a]. Soit $R : \mathcal{E} \approx_{pbf} \mathcal{F}$. Il suffit de montrer que pour tous $\sigma R \rho$Γla fonction f^{ρ}_{σ} est un isomorphisme. Si ce n'est pas le cas pour un couple (σ, ρ)Γ nous définissons $n \stackrel{\text{def}}{=} \mid \sigma \mid$ et $j \stackrel{\text{def}}{=} Max\{i | (\sigma(i) \leq_{\mathcal{E}} \sigma(n)$ et $\rho(i) \not\leq_{\mathcal{F}} \rho(n))$ ou $(\sigma(i) \not\leq_{\mathcal{E}} \sigma(n)$ et $\rho(i) \leq_{\mathcal{F}} \rho(n))\}$. Par constructionΓ $pom(\sigma^j) \neq pom(\rho^j)$ et $pom(\sigma^{j+1}) = pom(\rho^{j+1})$. Mais alors pour les transitions en arrière $\sigma(1) \ldots \sigma(j-1) \stackrel{p}{\rightarrow} \sigma$ et $\rho(1) \ldots \rho(n) \stackrel{q}{\rightarrow} \rho$Γ$p \neq q$Γce qui contredit $\sigma R \rho$. □

Les Propositions 6.3.1 et 6.3.3 nous permettent alors d'énoncer que

Corollaire 6.3.4 [Che92a] *Pour tous* $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$,

$$\mathcal{E} \approx_{pbf} \mathcal{F} \text{ ssi } \mathcal{E} \approx_{hp} \mathcal{F}$$

La caractérisation de $\approx_{hp}$ par $\approx_{pbf}$ est montrée dans [Che92a] par une preuve directe de l'égalité entre ces deux équivalences. Mais en réalitéΓcette preuve constructive s'appuie sur une relation intermédiaire entre $\approx_{hp}$ et $\approx_{pbf}$ qui est précisement une équivalence de mixed-ordering.

6.3.3 L'équivalence causale

La history preserving bisimulation se caractérise aussi par l'*équivalence causale* de [DD89]. Elle est basée sur les *arbres causaux* qui sont des variantes des arbres de synchronisation de Milner : l'étiquetage est enrichi de sorte qu'il permet de retrouver les indications de dépendances causales entre les actions qui ont lieu. Cette approche a le gros avantage d'utiliser des modèles d'entrelacementΓqui ont été beaucoup étudiés dans la littératureΓet pour lesquels on dispose de nombreuses équivalences sémantiques.

A toute $\mathcal{E} \in \mathbf{SE}$ on associe un arbre causal $CT(\mathcal{E})$Γbasé sur les histoiresΓet qui contient toute l'information pour retrouver la SE (à isomorphisme près)[4].

La définition qui suit est extraite de [DD89].

Définition 6.3.4 (Arbre causal et équivalence causale) *Soit* $\mathcal{E} \in \mathbf{SE}$. *L'*arbre causal *de* $\mathcal{E}$, *noté* $CT(\mathcal{E})$, *est défini de la façon suivante :*

- *l'ensemble des nœuds de* $CT(\mathcal{E})$ *est* $Hist(\mathcal{E})$,
- *les arcs sont les paires* $(\sigma, \sigma.e)$,
- *l'étiquetage d'un arc* $(\sigma, \sigma.e)$ *est* $(l(e), C(e, \sigma))$, *où* $C(e, \sigma)$ *est l'ensemble des* références relatives *des causes de* e *dans* σ, *où*

la référence relative *de* e_i *dans* $e_1 \ldots e_i \ldots e_n$ *est le nombre d'événements du facteur* $e_i \ldots e_n$.

[4]Cette approche se généralise aux SE dites *free* qui généralisent les SE premières.

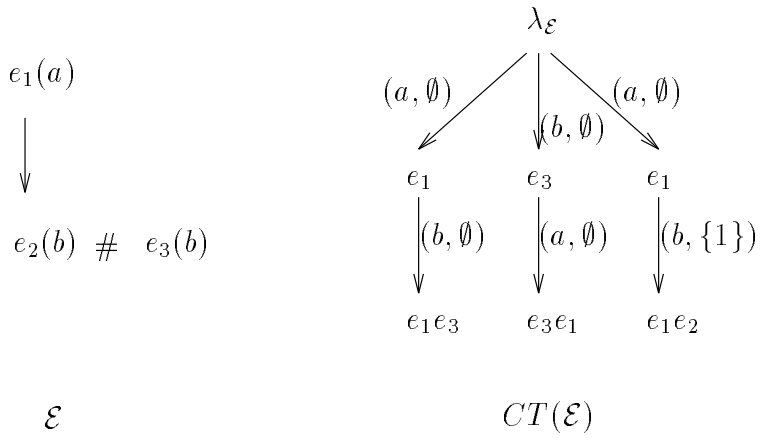


Figure 6.5: Un arbre example d'arbre causal

L'ensemble $C(e,\sigma)$ doit être compris comme un ensemble de pointeurs en arrière dans l'arbre $CT(\mathcal{E})$ vers les causes de e.

*L'*équivalence causale *sur* ***SE*** *est alors définie par :*

$$\mathcal{E} \approx_c \mathcal{F} \stackrel{def}{=} CT(\mathcal{E}) \underline{\leftrightarrow} CT(\mathcal{F})$$

où $\underline{\leftrightarrow}$ est la bisimulation forte sur les ST.

Dans la Figure 6.5Γnous avons représenté l'arbre causal de la SE $\mathcal{E}$ de la Figure 6.4.

Proposition 6.3.5 [Vaa89a] *Pour tous $\mathcal{E}, \mathcal{F} \in SE$,*

$$\mathcal{E} \approx_{hp} \mathcal{F} \ ssi \ \mathcal{E} \approx_c \mathcal{F}$$

6.4 Conclusion

Dans ce chapitreΓnous avons rappelé les différentes manières d'exprimer la history preserving bisimulation. Plus précisémentΓil est rappelé que pour tous $\mathcal{E}, \mathcal{F} \in$ **SE**Γ

$$\mathcal{E} \approx_{hp} \mathcal{F} \text{ ssi } \mathcal{E} \approx_{mo} \mathcal{F} \text{ ssi } \mathcal{E} \approx_{pbf} \mathcal{F} \text{ ssi } \mathcal{E} \approx_c \mathcal{F} \tag{6.9}$$

Concernant les équivalences $\approx_{hp}$Γ$\approx_{mo}$ et $\approx_c$Γle résultat a aussi été établi par [Ace91] dans le cadre plus général des SE stables avec actions invisibles.

Au Chapitre 9 nous montrons que les équivalences de [Ace91] ne sont pas les bonnes généralisations de $\approx_{hp}$Γ$\approx_{mo}$ et $\approx_c$. En effetΓnous établissons que la quatrième équivalence $\approx_{pbf}$ lorsqu'on l'étend trivialement à la présence de τ (l'action

invisible) est alors strictement plus fine que celles de [Ace91].

Les résultats du Chapitre 9 montrent clairement l'intérêt de disposer d'une définition de l'équivalence $\approx_{hp}$ en termes d'une bisimulation avant-arrière. Dans le chapitre suivant, le Chapitre 7, nous exploitons d'une autre façon la bisimulation avant-arrière de [Che92a]. En effet, cette équivalence à l'avantage de n'être définie que sur des critères comportementaux, ou encore observationnels, tout comme $\approx_c$, mais pas $\approx_{hp}$ ou $\approx_{mo}$. De plus, l'observation utilisée pour $\approx_{pbf}$ est basée sur des actions locales, les pomsets, ce qui rend cette équivalence plus "naturelle" et ce qui nous permet d'en dériver une logique adéquate.

Chapitre 7

Caractérisations logiques

Une multitude de logiques modales ont été proposées pour les modèles d'ordre partiel[1]. CependantΓ toutes ces logiques ne sont pas utilisées dans la même optique. Certaines sont introduites pour fournir un formalisme de description de la structure des modèles (*e.g.* *ESL* [Pen88]Γ *DESL* [Pen89]Γ *ESL[δ]* [Pen91]Γ *ESL[C]* de [MT89])Γ d'autres pour décrire la structure des calculs (*e.g.* *POTL* [PW84]Γ *POTL[U,S]* de [KP86]Γ *ISTL* [KP87])Γ ou encore pour décrire les propriétés comportementales des systèmes (*e.g.* L_T et L_P [DF90]Γ *POL* [Sin90]).

Notre utilisation des logiques se base sur les mêmes motivations que dans la première partie de la thèse. A savoir que les logiques nous fournissent un formalisme bien adapté à la définition sémantique du comportement. Puisque notre intérêt porte sur les sémantiques du temps arborescentΓ nous justifions dans la suite l'exclusion de nombreuses logiques modales qui existent dans la littérature.

Considérons les deux SE :

$$e_1(a)\#e_2(a) \qquad\qquad e(a)$$

$$\mathcal{E} \qquad\qquad \mathcal{F}$$

[1] Nous excluons de la discussion les logiques définies pour d'autres modèles que les Structures d'Evénements (premières), *e.g.* la logique $\mathcal{F}(B)$ de Reisig pour les réseaux de Petri ou encore *SESL* de [LT87] pour les agents séquentiels

Il est clair que du point de vue comportementalΓ$\mathcal{E}$ et $\mathcal{F}$ sont les mêmes : la seule exécution possible est d'effectuer l'action a. Nous n'autorisons donc pas aux logiques qui nous intéressent de distinguer ces deux modèles. Ceci est un argument suffisant pour exclure les logiques *ESL* [Pen88]Γ *DESL* [Pen89]Γ *ESL[δ]* [Pen91]Γ...Γ dans lesquelles on peut exprimer qu'il existe deux événements en conflit.

Les logiques comme *POTL* [PW84]Γ *POTL[U,S]* de [KP86]Γ *ISTL* [KP87] sont exclues car elles font partie des logiques du temps linéaire.

En réalitéΓil n'existe pas dans la littérature beaucoup d'exemples de logiques ayant été proposées pour étudier leur lien avec des équivalences sémantiques. A notre connaissanceΓseuls [DF90] et [GKP92] ont abordé cette question.

[DF90] proposent deux logiques L_T et L_P. La première caractérise la weak history preserving bisimulation et la secondeΓ la history preserving bisimulation comme nous le montrons en Section 7.4).

[GKP92] étudie des logiques temporelles existantes en comparant les équivalences qu'elles induisentΓet en établissant des liens entre ces équivalences et des équivalences comportementales.

Les résultats de cette partie sont extraits de [CLP92]. Nous proposons une logique avec des opérateurs du passéΓL_{pbf}Γdérivée de la logique HML du Chapitre 3Γ de sorte qu'elle caractérise la pomset bisimulation avant-arrièreΓ$\approx_{pbf}$Γde la Section 6.3.2. Ce résultat est prouvé en Section 7.2 Théorème 7.2.1. Comme corollaire immédiat de ce théorèmeΓnous obtenons une caractérisation logique par L_{pbf} de la history preserving bisimulation. Nous proposons en Section 7.4Γune comparaison de l'expressivité de L_{pbf} et de la logique L_P de [DF90]. Il est montré que leur pouvoir d'expression est le même puisque l'on peut traduire toute formule de L_{pbf} en une formule équivalente de L_PΓet réciproquement.

7.1 La logique L_{pbf}

Nous visons à définir une logique modale qui permette d'exprimer les propriétés des histoires dans une SEΓet dont le pouvoir de distinction coïncide avec celui de la pomset bisimulation avant-arrière.

Dans la littératureΓon retrouve de multiples versions du Théorème de Hennessy-Milner (voir Théorème 3.4.1 du Chapitre 3 qui relie une bisimulation à une logique "à la HML"). En réalitéΓce théorème supporte toute version dans laquelle la nature de l'observation que l'on considère pour la bisimulation (*e.g.* une actionΓune action modulo τΓun pomset) est transportée fidèlement dans les opérateurs de la logique. Si de plus la bisimulation est avant-arrièreΓil suffit de rajouter à HML des modalités du passé. C'est ce que nous nous proposons d'utiliser pour définir L_{pbf}.

La syntaxe de L_{pbf} est définie par :

$$\varphi, \psi \ (\in L_{pbf}) ::= \top \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle \leftarrow q \rangle \varphi \mid \langle a \rangle \varphi$$

où $a \in Act$ et q est un pomset sur Act.

Pour tout $A \subseteq Act$ et $\varphi \in L_{pbf}$, nous utilisons les abréviations $\langle \leftarrow A\rangle\varphi$ pour $\bigvee_{a\in A}\langle \leftarrow a\rangle\varphi$, $\langle A\rangle\varphi$ pour $\bigvee_{a\in A}\langle a\rangle\varphi$ et $[A]\varphi$ pour $\neg\langle A\rangle\neg\varphi$.

Une formule de L_{pbf} s'interprète sur les histoires d'une SE. Intuitivement, $\langle a\rangle\varphi$ se lit "il est possible d'effectuer l'action a en atteignant un état dans lequel la formule φ est vraie". $\langle \leftarrow q\rangle\varphi$ se lit "il y a eu un moment dans l'histoire où la formule φ était vraie et tel que le pomset q est le calcul effectué entre ce moment dans le passé et notre présent".

Formellement, étant donnée $\mathcal{E} \in \mathbf{SE}$, nous définissons $\models_{\mathcal{E}} \subseteq Hist(\mathcal{E}) \times L_{pbf}$, la relation de satisfaction par :

- $\sigma \models_{\mathcal{E}} \top$ toujours
- $\sigma \models_{\mathcal{E}} \neg\varphi$ ssi $\sigma \not\models_{\mathcal{E}} \varphi$
- $\sigma \models_{\mathcal{E}} \varphi \wedge \psi$ ssi $\sigma \models_{\mathcal{E}} \varphi$ et $\sigma \models_{\mathcal{E}} \psi$
- $\sigma \models_{\mathcal{E}} \langle \leftarrow q\rangle\varphi$ ssi il existe $\sigma' \xrightarrow{q}_{\mathcal{E}} \sigma$ tel que $\sigma' \models_{\mathcal{E}} \varphi$
- $\sigma \models_{\mathcal{E}} \langle a\rangle\varphi$ ssi il existe $\sigma \xrightarrow{a}_{\mathcal{E}} \sigma'$ tel que $\sigma' \models_{\mathcal{E}} \varphi$

Nous disons alors que $\mathcal{E}$ ($\in \mathbf{SE}$) satisfait φ ($\in L_{pbf}$), noté $\mathcal{E} \models \varphi$, si $\lambda_{\mathcal{E}} \models_{\mathcal{E}} \varphi$. Pour alléger les notations, nous omettrons l'indice $\mathcal{E}$ dans $\models_{\mathcal{E}}$ quand il n'y a pas d'ambiguïté.

Comme dans la Section 3.3, nous dérivons de notre logique une équivalence sur $\mathbf{SE}$ par :

$$\mathcal{E} \sim_{L_{pbf}} \mathcal{F} \text{ ssi pour toute } \varphi \in L_{pbf}\ (\mathcal{E} \models \varphi \text{ ssi } \mathcal{F} \models \varphi)$$

Exemple 17 *Les deux SE de la Figure 6.4, page 104 ne sont pas équivalentes pour $\sim_{L_{pbf}}$. En effet, seule $\mathcal{F}$ satisfait la formule $\langle a\rangle[b]\langle \leftarrow a;b\rangle\top$. Elle exprime qu'il existe une action a qui est une cause de toutes les actions b qui suivent.*

7.2 Caractérisation de $\approx_{pbf}$ par L_{pbf}

Nous présentons le Théorème 7.2.1, une variante du Théorème de Hennessy-Milner. Pour que notre bisimulation coïncide avec notre logique, il nous faut considérer une hypothèse "de branchement fini". Les SE que nous considérons sont finies, ce qui nous garantit qu'il n'existe qu'un nombre fini de transitions possibles à chaque étape. Ce nombre est borné par le nombre d'événements de la SE.

Cependant, il est possible d'établir la preuve sous des hypothèses moins strictes : il suffit en fait d'exiger que pour tout $n \in N$, la troncature de la SE au niveau n soit finie. Intuitivement, cette hypothèse nous limite à des SE possédant un branchement fini pour leur non-déterminisme et leur parallélisme. Soit $\mathbf{SE}_f$ la sous-classe de $\mathbf{SE}$ formée des SE finies.

Théorème 7.2.1 (Adéquation) *Soient* $\mathcal{E}, \mathcal{F} \in \mathbf{SE}_f$

$$\mathcal{E} \approx_{pbf} \mathcal{F} \text{ ssi } \mathcal{E} \sim_{L_{pbf}} \mathcal{F}$$

Preuve

- $\Rightarrow$): Soit $R : \mathcal{E} \approx_{pbf} \mathcal{F}$. Nous montrons par induction sur la structure des formules de L_{pbf} que $\sigma R \rho$ implique ($\sigma \models_{\mathcal{E}} \varphi$ ssi $\rho \models_{\mathcal{F}} \varphi$) pour toute $\varphi \in L_{pbf}$.
  Les cas $\top, \neg\varphi, \varphi \wedge \psi$ sont évidents. Supposons $\sigma \models_{\mathcal{E}} \langle \leftarrow q \rangle \varphi$. Il existe donc $\sigma' \xrightarrow{q}_{\mathcal{E}} \sigma$ tel que $\sigma' \models \varphi$. Puisque $\sigma R \rho$Γil existe une transition en arrière $\rho' \xrightarrow{q}_{\mathcal{F}} \rho$ avec $\sigma' R \rho'$. Par hypothèse d'inductionΓ$\rho' \models_{\mathcal{F}} \varphi$. Donc $\rho \models \langle \leftarrow q \rangle \varphi$.
  Le cas des formules de la forme $\langle a \rangle \varphi$ se traite de façon analogue.
  FinalementΓ$\lambda_{\mathcal{E}} R \lambda_{\mathcal{F}}$ implique $\mathcal{E} \sim_{L_{pbf}} \mathcal{F}$.
- $\Leftarrow$): Notons $\sigma \sim \rho$ lorsque σ et ρ satisfont les même formules de L_{pbf}. Nous montrons que $\sim : \mathcal{E} \approx_{pbf} \mathcal{F}$.
  1. $\mathcal{E} \sim_{L_{pbf}} \mathcal{F}$ implique $\lambda_{\mathcal{E}} \sim \lambda_{\mathcal{F}}$.
  2. Supposons que $\sigma \sim \rho$Γet $\sigma \xrightarrow{a}_{\mathcal{E}} \sigma'$. Soit $\{\varphi_1, \varphi_2, \ldots\}$ une énumération des formules de L_{pbf}. Nous définissons $(\psi_i)_{i \in N}$ par :

$$\begin{aligned} \psi_0 &\stackrel{\text{def}}{=} \top \\ \psi_{i+1} &\stackrel{\text{def}}{=} \psi_i \wedge \varphi_i \quad \text{si} \quad \sigma' \models_{\mathcal{E}} \varphi_i \\ \psi_{i+1} &\stackrel{\text{def}}{=} \psi_i \wedge \neg\varphi_i \quad \text{si} \quad \sigma' \not\models_{\mathcal{E}} \varphi \end{aligned}$$

  Par constructionΓ$\sigma \models_{\mathcal{E}} \langle a \rangle \psi_i$ pour tout i. Puisque $\sigma \sim \rho$Γ$\rho \models_{\mathcal{F}} \langle a \rangle \psi_i$. Donc pour chaque iΓil existe un ρ_i tel que $\rho \xrightarrow{a}_{\mathcal{F}} \rho'_i$ et $\rho'_i \models_{\mathcal{F}} \psi_i$. Puisque $\mathcal{F} \in \mathbf{SE}_f$Γ les ρ_i sont en nombre fini. Il existe donc $\rho' \in Hist(\mathcal{F})$ tel que $\rho \xrightarrow{a}_{\mathcal{F}} \rho'$ et $\rho' \models_{\mathcal{F}} \psi_i$ pour une infinité de iΓde sorte que $\sigma' \sim \rho'$.
  3. La propriété de transfert en arrière se démontre de manière analogueΓmais cette foisΓl'hypothèse de branchement fini n'intervient pas puisque que les prédécesseurs sont uniques le long de l'histoire.

□

Remarque 9 *Le Théorème 7.2.1 n'utilise l'hypothèse de b.f. que pour montrer que* L_{pbf} *implique* $\approx_{pbf}$.

Le Corollaire 6.3.4 du Chapitre 6 et le Théorème 7.2.1 nous permettent d'énoncer que

Corollaire 7.2.2 *Dans les SE finies (ou plus généralement à branchement fini), la logique* L_{pbf} *est adéquate pour l'équivalence de history preserving bisimulation.*

Notons que dans la littératureΓnous disposons déjà "gratuitement" d'une caractérisation modale très simple de la history preserving bisimulation au moyen de sa reformulation en la bisimulation forte sur l'arbre causal dérivé de la SE. En effetΓpour le faireΓil suffit de considérer la logique HML (présentée au Chapitre 3)

dans laquelle les modalités $EX\,a$ sont substituées par des modalités de la forme $EX\,(a,K)$Γ où les (a,K) sont les étiquettes des branches de l'arbre causalΓ puis d'utiliser le très classique théorème d'adéquation de [HM85] entre HML et la bisimulation forte. Il est d'ailleurs intéressant de noter que cette logique étendue est bien une généralisation de la logique HML de [HM85] (les étiquettes des arbres causaux dérivés des ST - qui sont séquentiels - sont toutes de la forme $(a,1)$Γavec $a \in Act$).

En étudiant les autres résultats de la littératureΓnous avons cherché à comparer L_{pbf} avec d'autres logiques qui caractérisent $\approx_{hp}$. [DF90] propose la logique L_P que nous rappelons dans la Section 7.3. En réalitéΓdans cet articleΓil est affirmé que L_P caractérise non pas $\approx_{hp}$ mais la weak history preserving bisimulation. Nous prouvons dans ce qui suit que cette assertion est fausse.

Nous n'avons pas traité dans cette thèse la comparaison entre L_{pbf} et la logique HML "étendue" avec les modalités de la forme $EX\,(a,K)$ comme indiqué plus haut. Il est clair que ces deux logiques ont le même pouvoir de distinction puisqu'elles caractérisent la même équivalence sémantique. Une comparaison de leur expressivité reste à poursuivre.

7.3 La logique L_P

La logique L_P de [DF90] est une logique "à la CTL^*" (voir le Chapitre 4)Γà laquelle on a rajouté des modalités du passé. Notre présentation de la logique L_P est légèrement différente de celle d'origine : nous considérons des propositions atomiques. CependantΓil est immédiat de vérifier que notre définition et celle de [DF90] sont équivalentes.

7.3.1 Définition de L_P

La syntaxe de L_P est la suivante :

$$\Phi, \Psi\ (\in L_P) ::= \top \mid q \mid \neg\Phi \mid \Phi \wedge \Psi \mid X^{-1}\Phi \mid X\Phi \mid \forall\Phi$$

où q est un pomset sur Act.

Comme pour CTL^*Γune formule de L_P s'interprète dans un état le long d'une exécution du systèmeΓi.e. un futur choisi. De plusΓpuisque l'on considère des modalités du passéΓil est nécessaire de garder l'information de l'histoire du calcul qui a mené à l'état courant. Par conséquentΓpour $\mathcal{E} \in \mathbf{SE}$Γles formules de L_P sont interprétées sur des couples de la forme (π,σ)Γoù $\sigma \in Hist(\mathcal{E})$Γet $\pi \in Hist(\mathcal{E})$ est une extension maximale de σΓi.e. une exécution extension de σ.

IntuitivementΓ X^{-1} est la modalité permettant d'exprimer des propriétés du prédécesseur immédiat de l'état courant C_σ dans l'histoire σ. X est la modalité "symétrique"; elle fait référence au successeur de C_σ le long du futur π. Le quantificateur $\forall$ munit la logique L_P du pouvoir d'expression des logiques arborescentesΓ

en quantifiant sur tous les futurs possiblesΓi.e. toutes les extensions possibles de l'histoire σ. La proposition atomique q signifie qu'il a existé un état dans l'histoire σ tel que le pomset calculé entre cet état et l'état C_σ soit q.

Soit $\mathcal{E} \in \mathbf{SE}\Gamma\sigma \in Hist(\mathcal{E})\Gamma$et $\pi \in Hist(\mathcal{E})$ une exécution telle que $\sigma \leq \pi$. Nous définissons $\models_\mathcal{E}$ la relation de satisfaction par :

- $(\pi, \sigma) \models_\mathcal{E} \top$ toujoursΓ
- $(\pi, \sigma) \models_\mathcal{E} q$ ssi $\sigma = \sigma'.\theta$ avec $pom(\theta) = q\Gamma$
- $(\pi, \sigma) \models_\mathcal{E} \neg\Phi$ ssi $(\pi, \sigma) \not\models_\mathcal{E} \Phi\Gamma$
- $(\pi, \sigma) \models_\mathcal{E} \Phi \wedge \Psi$ ssi $(\pi, \sigma) \models_\mathcal{E} \Phi$ et $(\pi, \sigma) \models_\mathcal{E} \Psi$
- $(\pi, \sigma) \models_\mathcal{E} X^{-1}\Phi$ ssi $\sigma = \sigma'.e$ et $(\pi, \sigma') \models_\mathcal{E} \Phi\Gamma$
- $(\pi, \sigma) \models_\mathcal{E} X\Phi$ ssi $\pi = \sigma.e.\theta$ et $(\pi, \sigma.e) \models_\mathcal{E} \Phi\Gamma$
- $(\pi, \sigma) \models_\mathcal{E} \forall\Phi$ ssi pour toute exécution π' telle que $\sigma \leq \pi'\Gamma(\pi', \sigma) \models_\mathcal{E} \Phi$.

Notations 12 *On note $\exists\Phi$ pour $\neg\forall\neg\Phi$ et X^{-n} (resp. X^n) pour un emboîtement de n opérateurs X^{-1} (resp. X).*

On interprète les formules de L_P sur les histoires (c-à-d. les états de calcul avec passé) selon :

$$\sigma \models_\mathcal{E} \Phi \stackrel{\text{def}}{-} (\pi, \sigma) \models_\mathcal{E} \Phi \text{ pour toutes les exécutions } \pi \text{ telles que } \sigma \leq \pi.$$

On note $\mathcal{E} \models \Phi$ lorsque $\lambda_\mathcal{E} \models_\mathcal{E} \Phi$. L'équivalence induite par L_P sur **SE** est alors définie par :

$$\mathcal{E} \sim_{L_P} \mathcal{F} \stackrel{\text{def}}{-} (\mathcal{E} \models \Phi \text{ ssi } \mathcal{F} \models \Phi) \text{ pour toute } \Phi \in L_P$$

Exemple 18 *[RT88] L'exemple des SE de la Figure 7.1 montre que L_P caractérise une équivalence strictement plus fine que la weak history preserving bisimulation. En effet, on peut montrer que $\mathcal{E}$ et $\mathcal{F}$ sont équivalentes pour cette bisimulation, mais seule $\mathcal{E}$ satisfait la formule $\Phi \equiv \forall X(\exists X \exists X a;b) \in L_P$. Cette formule exprime que quelque soit le premier pas de calcul (ici, c'est toujours le pomset a), le système pourra toujours effectuer le pomset $a;b$, ce qui n'est pas vrai de $\mathcal{F}$ si on effectue d'abord l'événement e'_2.*

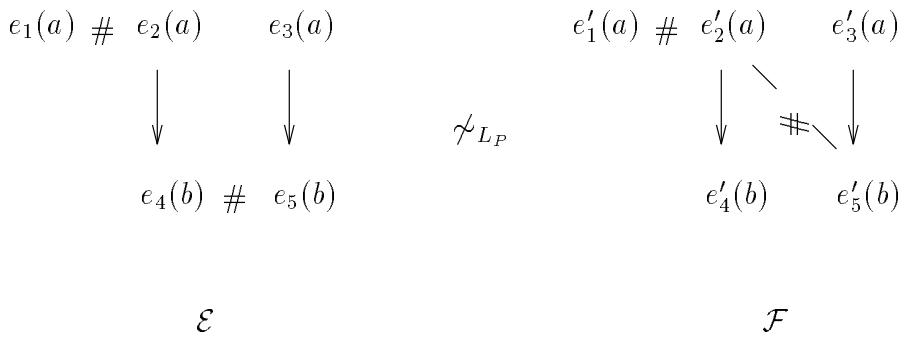


Figure 7.1: $\sim_{L_P}$ est strictement plus fine que la weak history preserving bisimulation

7.3.2 Une traduction simple de L_{pbf} vers L_P

Théorème 7.3.1 *Toute $\Phi \in L_{pbf}$ peut être transformée en une formule $\Phi^* \in L_P$ équivalente, i.e.*

$$\sigma \models_{\mathcal{E}} \Phi \text{ ssi } \sigma \models_{\mathcal{E}} \Phi^*$$

pour toutes $\mathcal{E} \in \boldsymbol{SE}$ et $\sigma \in Hist(\mathcal{E})$.

Preuve Dans cette preuve$\Gamma|\,q\,|$ dénote la taille du pomset $q\Gamma$i.e. le cardinal du multiset q.

Nous exhibons la transformation $(\ )^* : L_{pbf} \to L_P\Gamma$définie par induction sur la structure des formules de L_{pbf} :

$$\begin{aligned}
&(\top)^* = \top \\
&(\neg\Phi)^* = \neg(\Phi)^* \\
&(\Phi \wedge \Psi)^* = (\Phi)^* \wedge (\Psi)^* \\
&(\langle \leftarrow q\rangle\Phi)^* = q \wedge X^{-|q|}(\Phi)^* \\
&(\langle a\rangle\Phi)^* = \exists X((\Phi)^* \wedge a)
\end{aligned}$$

Il est immédiat de montrer que cette transformation vérifie le Théorème 7.3.1. □

Exemple 19 *En utilisant la transformation $(\ )^*$, la formule $\langle a\rangle[b]\langle \leftarrow a;b\rangle\top \in L_{pbf}$ de l'Exemple 17 se traduit en $\exists X(a \wedge \forall X(a;b))$.*

Un corollaire immédiat du Théorème 7.3.1 est que la logique L_P est au moins aussi discriminante que L_{pbf} et donc que $\approx_{hp}$. Ce qui démontre que l'assertion du Théorème 4.19 de [DF90] est bien fausse.

7.4 Une traduction de L_P vers L_{pbf}

Dans cette section nous montrons que la logique L_P caractérise exactement la history preserving bisimulation. Pour y parvenirΓ nous prouvons (Corollaire 7.4.7) qu'on peut traduire toute formule de L_P en une formule équivalente de L_{pbf}.

La traduction de L_P vers L_{pbf} n'est pas triviale dans la mesure où L_P n'apparaît pas comme un fragment de L_{pbf}. Si l'on espère transformer les formules de L_P dans L_{pbf}Γil est clair que l'on ne considérera que des formules qui ne dépendent pas d'un futur choisi; nous les appelons formules *d'histoire*.

FormellementΓ$\Phi \in L_P$ est *d'histoire* si pour tout (π, σ)Γ

$$(\pi, \sigma) \models \Phi \text{ ssi } (\pi, \sigma) \models \forall\Phi$$

autrement dit si la valeur de vérité de Φ en (π, σ) ne dépend que de σ.

Une formule $\Phi \in L_P$ est *arborescente* si toutes les occurrences de la modalité X dans Φ sont immédiatement précédées d'un quantificateur $\forall$ ou $\exists$. C'est le cas de la formule de l'Exemple 18.

Nous désignons par L_P^{arb} le sous-ensemble de L_P formé des formules arborescentes.

Par définitionΓL_P^{arb} ne contient que des formules d'histoire. Ce fragment de L_P est facile à traduire dans L_{pbf}.

Le problème de traduire L_P dans L_{pbf} peut donc être ramené à deux sous-problèmes : le premier consiste à traduire L_P^{arb} dans L_{pbf}Γce que nous faisons dans la section suivante. Le second problème consiste à montrer comment toute formule d'histoire de L_P peut être traduite dans L_P^{arb}Γce que nous traitons en Section 7.4.2.

7.4.1 De L_P^{arb} vers L_{pbf}

Proposition 7.4.1 *Toute formule $\Phi \in L_P^{arb}$ est équivalente à une formule $\varphi \in L_{pbf}$.*

Preuve Il suffit de remplacer dans $\Phi \in L_P$ toute occurrence

- des proposition atomique q par $\langle \leftarrow q \rangle \top$Γ
- de X^{-1} par $\langle \leftarrow Act \rangle$Γ
- de $\forall X$ par $[Act]$ et $\exists X$ par $\langle Act \rangle$.

□

Puisque les formules de L_P^{arb} se traduisent facilement en formules de L_{pbf}Γnous allons montrer comment transformer toute formule (d'histoire) de L_P en une formule de L_P^{arb} équivalente.

7.4.2 De L_P vers L_P^{arb}

Notations 13 *Nous écrivons δ au lieu $\neg X \top$. Ainsi, $\sigma \models_{\mathcal{E}} \delta$ ssi σ est une exécution.*

Définition 7.4.1 (Formules linéaires, du passé, du futur)

- *Une formule de L_P est* linéaire *si elle ne contient aucun $\forall$ (ni $\exists$).*
- *Une formule de L_P est* du passé *(resp.* futur*) si elle ne contient aucun X (resp. X^{-1}).*
- *Une formule de L_P est du* futur strict *si elle est du futur et si toutes les propositions atomiques q apparaissent sous la portée d'un X.*

Les noms "du futur" et "du passé" n'expriment que des propriétés syntaxiques des formules. Il ne faut pas leur attribuer un sens sémantique selon que la formule décrit une propriété dans le passéΓdans le présent ou dans le futur de l'état courant.

CependantΓtoute formule du passé est une formule d'histoire ; comme de plus une formule du passé ne contient aucun opérateur XΓelle est évidemment arborescente.

La méthode que l'on utilise pour transformer toute formule d'histoire L_PΓi.e. de la forme $\forall\Phi$Γen une formule arborescente est de remplacer Φ par une formule équivalente $\tilde{\Phi}$ dans laquelle l'opérateur $\forall$ se propage devant toutes les modalités X.

Le Théorème 7.4.3 permet de traiter le cas où Φ est une formule linéaire Γ : il est basé sur le Lemme de séparation 7.4.2 qui transforme Γ en une combinaison booléenne $\tilde{\Gamma}$ de formules du passé et du futur. L'opérateur $\forall$ étant sans effet sur les formules du passé (puisqu'elles sont d'histoire)Γil se propage trivialement dans les formules du futur (voir les Lemmes 7.4.4 et 7.4.5).

Lemme 7.4.2 *Toute formule linéaire $\Gamma \in L_P$ est équivalente à une formule linéaire $\tilde{\Gamma}$ de la forme*

$$\bigwedge_i (\bigvee_j \Gamma^-_{i,j} \vee \bigvee_k \Gamma^+_{i,k})$$

où les $\Gamma^-_{i,j}$ sont du passé et les $\Gamma^+_{i,k}$ sont du futur.

Preuve Nous utilisons des règles de réécriture qui séparent les modalités du passé de celles du futur :

$$\begin{array}{lcll}
X X^{-1}\Gamma & \longrightarrow & X\top \wedge \Gamma & (1)\\
X \neg X^{-1}\Gamma & \longrightarrow & X\top \wedge \neg\Gamma & (2)\\
X^{-1} X\Gamma & \longrightarrow & X^{-1}\top \wedge \Gamma & (3)\\
X^{-1} \neg X\Gamma & \longrightarrow & X^{-1}\top \wedge \neg\Gamma & (4)\\
 & & & \\
X(\Gamma \wedge \Psi) & \longrightarrow & X\Gamma \wedge X\Psi & (5)\\
X(\Gamma \vee \Psi) & \longrightarrow & X\Gamma \vee X\Psi & (6)\\
X^{-1}(\Gamma \wedge \Psi) & \longrightarrow & X^{-1}\Gamma \wedge X^{-1}\Psi & (7)\\
X^{-1}(\Gamma \vee \Psi) & \longrightarrow & X^{-1}\Gamma \vee X^{-1}\Psi & (8)\\
\neg(\Gamma \wedge \Psi) & \longrightarrow & \neg\Gamma \vee \neg\Psi & (9)\\
\neg(\Gamma \vee \Psi) & \longrightarrow & \neg\Gamma \wedge \neg\Psi & (10)\\
\neg\neg\Gamma & \longrightarrow & \Gamma & (11)
\end{array}$$

Nous considérons un ordre lexicographique $(\leq_1, \leq_2, \leq_3)$ sur le triplet (M_1, M_2, n_3) où :

1. M_1 est le multiset du nombre de modalités X^{-1} et X dans chaque branche maximale de la formuleΓ
2. M_2 est le multiset des profondeurs des connectifs $\wedge$ et $\vee$ dans la formuleΓ
3. n_3 est le nombre d'opérateurs $\neg$ dans la formule.

Toute règle du système de réécriture défini ci-dessus fait décroître strictement cet ordre. En effetΓles règles 1 à 4 font décroître strictement M_1 ; les règles 5 à 10 laissent inchangé M_1 et font décroître strictement M_2 ; enfinΓla règle 11 n'accroît ni $M_1\Gamma$ni $M_2\Gamma$et fait décroître strictement n_3.
L'ordre choisi est bien fondé[2] car les ordres $\leq_1\Gamma\leq_2$ et $\leq_3$ le sont. Par conséquentΓ le système termine.

Il reste à montrer sa complétudeΓ c-à-d. que toute formule dans laquelle un opérateur X (resp. X^{-1}) est dans la portée d'un X^{-1} (resp. X) est réductible par une des règles. Pour celaΓon montre que toute formule irréductible est séparée.

Considérons Φ une formule irréductible. Les règles de 5 à 11 garantissent qu'il ne peut y avoir de $\wedge$ ou de $\vee$ dans la portée d'un X ou d'un X^{-1}. Φ est donc une combinaison booléenne de formules composées des seuls opérateurs $X\Gamma X^{-1}\Gamma\neg\Gamma q$ et $\top$. Les règles 1 à 4 excluent le cas où Φ contient des alternances de modalités du futur et de modalités du passé. Φ est donc séparéeΓce qui termine la preuve. □

Le Lemme 7.4.2Γ qui décompose une formule en parties du passé et du futurΓ est inspiré des techniques de preuves utilisées dans le "théorème de séparation" de [Gab87]Γmais il porte sur des logiques du temps arborescent au lieu du temps linéaire.

Théorème 7.4.3 *Toute formule linéaire $\forall\Gamma \in L_P$, est équivalente à une formule arborescente de la forme*

$$\bigwedge_i \bigvee_j (\Gamma^-_{i,j} \vee \Psi_i]$$

où les $\Gamma^-_{i,j}$ sont du passé et les Ψ_i sont du futur et arborescentes.

Preuve Grâce au Lemme 7.4.2Γon peut supposer Γ de la forme $\bigwedge_i(\bigvee_j \Gamma^-_{i,j}\vee\bigvee_k \Gamma^+_{i,k})$. Puisque l'opérateur $\forall$ se distribue sur les conjonctionsΓon sait que

$$\forall\Gamma \equiv \bigwedge_i \forall(\bigvee_j \Gamma^-_{i,j} \vee \bigvee_k \Gamma^+_{i,k}) \tag{7.10}$$

D'autre partΓle fait que chaque $\Gamma^-_{i,j}$ soit une formule d'histoireΓ puisque c'est une formule du passéΓle $\forall$ devant chaque disjonction peut être propagé devant les formules du futur comme suit :

[2] i.e. il n'existe pas de suite infinie décroissante.

$$\forall\Gamma \equiv \bigwedge_i (\bigvee_j \Gamma^-_{i,j} \vee \forall \bigvee_k \Gamma^+_{i,k}) \tag{7.11}$$

En effetΓon peut montrer que

Lemme 7.4.4 *Pour toute formule Φ et toute formule d'histoire Φ^-,*

$$\forall(\Phi^- \vee \Phi) \equiv \Phi^- \vee \forall\Phi$$

Preuve Soit $(\pi,\sigma) \models \forall(\Phi^- \vee \Phi)$. Cela signifie que pour chaque exécution π'Γou bien $(\pi',\sigma) \models \Phi^-$Γou bien $(\pi',\sigma) \models \forall\Phi$.

Si il existe π_1 telle que $(\pi_1,\sigma) \models \Phi^-$Γalors comme $\Phi^- \equiv \forall\Phi^-$Γ$(\pi',\sigma) \models \Phi^-$ pour tout π'Γdonc en particulier (π,σ)Γce qui implique $(\pi,\sigma) \models \Phi^- \vee \forall\Phi$.

SinonΓtoutes les exécution π' vérifient $(\pi',\sigma) \models \forall\Phi$Γen particulier pour π et on conclut facilement.

Nous omettons la preuve pour l'autre directionΓqui est plus simple. □

Il reste à montrer comment on peut transformer chaque $\forall \bigvee_k \Gamma^+_{i,k}$ en une formule arborescente. Il suffit d'utiliser le lemme qui suit :

Lemme 7.4.5 *Toute formule $\forall\Psi$, où Ψ est du futur, est équivalente à une formule arborescente.*

Preuve La preuve se fait par induction sur la profondeur de la formule ΨΓi.e. le nombre maximal d'opérateurs X emboîtés dans ΨΓqu'on note $d(\Psi)$ (Cette notion a déjà été introduite dans le Chapitre 3 pour les formules de HML).

Le cas $d(\Psi) = 0$ est évident. Soit Ψ avec $d(\Psi) > 0$. La formule peut être représentée sous la forme générale :

$$\Psi \equiv \bigwedge_r (q_{r,1} \vee q_{r,2} \vee \ldots \vee \neg q'_{r,1} \vee \neg q'_{r,2} \vee \ldots \vee X\Psi_{r,1} \vee X\Psi_{r,2} \vee \ldots \vee \neg X\Psi'_{r,1} \vee \neg X\Psi'_{r,2} \vee \ldots)$$

où les $q_{r,k}$ sont des propositions atomiquesΓet donc des formules d'histoire. En utilisant le fait que $\neg X\Psi'_{r,i} \equiv \delta \vee X\neg\Psi'_{r,i}$Γet que $X\Phi \vee X\Phi' \equiv X(\Phi \vee \Phi')$Γon obtient :

$$\Psi \equiv \bigwedge_s (q_{s,1} \vee q_{s,2} \vee \ldots \vee \neg q'_{s,1} \vee \neg q'_{s,2} \vee \ldots \vee X(\Psi_{s,1} \vee \Psi_{s,2} \vee \ldots))$$

où certains $q_{s,i}$ sont éventuellement égaux à δ.

AlorsΓen appliquant le Lemme 7.4.4 on obtient :

$$\forall\Psi \equiv \bigwedge_s (q_{s,1} \vee q_{s,2} \ldots \vee \forall X\forall(\Psi_{s,1} \vee \Psi_{s,2} \vee \ldots))$$

Puisque $d(\Psi_{s,1} \vee \Psi_{s,2} \ldots) = Max\{d(\Psi_{s,i})\} < d(\Psi)$Γnous pouvons utiliser l'hypothèse d'induction. $\forall(\Psi_{s,1} \vee \Psi_{s,2} \vee \ldots)$ est alors équivalente à une formule arborescente Ψ_sΓ pour chaque s. Ainsi $\forall\Psi \equiv \bigwedge_s (q_{s,1} \vee \ldots \vee \forall X\Psi_s)$ qui est arborescente. □

Les formules $\Gamma^{-}_{i,j}$ et Ψ_i du Théorème 7.4.3 sont donc arborescentes, ce qui termine la preuve. □

Le Théorème 7.4.3, muni d'une stratégie d'application, permet de transformer toute formule d'histoire $\forall\Phi$ en une formule arborescente. C'est l'objet du Théorème 7.4.6.

Pour transformer $\forall\Phi$, quand Φ n'est pas linéaire, on applique le Théorème 7.4.3 à toutes les sous-formules $\forall\Phi_i$ de Φ, telles que Φ_i soit linéaire. Soit Ψ_i la formule de L_P^{arb} équivalente à $\forall\Phi_i$ obtenue. Ψ_i peut alors être temporairement considérée comme une proposition atomique p_i; ceci permet de d'appliquer le Théorème 7.4.3 à une formule contenant moins d'opérateurs $\forall$. Cette formule est $\forall\Phi$ dans laquelle les sous-formules Ψ_i sont remplacées par les propositions p_i.

Théorème 7.4.6 *Toute formule $\forall\Phi \in L_P$ est équivalente à une formule arborescente.*

Preuve On procède par induction sur les sous-formules de la forme $\forall\Phi'$.
Si Φ ne contient sous-formule de la forme $\forall\Phi'$, alors Φ est une formule linéaire et il suffit d'appliquer le Théorème 7.4.3.
Sinon, Φ contient des sous-formules $\forall\Phi_i$, $i = 1, \ldots$. Ces $\forall\Phi_i$ sont des formules d'histoire et peuvent donc être considérées comme des nouvelles propositions atomiques p_i. Le Théorème 7.4.3 s'applique alors à $\forall\Phi$, car Φ est considérée comme linéaire, grâce aux p_i. Nous obtenons alors une formule arborescente Ψ modulo les sous-formules $\forall\Phi_i$. Par hypothèse d'induction, chaque $\forall\Phi_i$ est équivalente à une formule arborescente Ψ_i. En remplaçant dans Ψ chaque $\forall\Phi_i$ par Ψ_i, on construit une formule Ψ' équivalente à $\forall\Phi$ et arborescente. Ce qui termine la preuve. □

D'après les Théorèmes 7.4.1 et 7.4.6, on peut énoncer :

Corollaire 7.4.7 *Toute formule $\Phi \in L_P$ peut être transformée en une formule $\Phi^* \in L_{pbf}$ de sorte que*

$$\sigma \models_{\mathcal{E}} \Phi \text{ ssi } \sigma \models_{\mathcal{E}} \Phi^*$$

pour toutes $\mathcal{E} \in \boldsymbol{SE}$ et $\sigma \in Hist(\mathcal{E})$.

La logique L_{pbf} est très simple. Elle pourrait par exemple être utilisée comme langage d'explication dans un vérificateur automatique pour $\approx_{hp}$. En effet, d'après le Théorème 7.2.1, si deux systèmes ne sont pas équivalents pour $\approx_{hp}$, i.e. $\approx_{pbf}$, alors il existe une formule de L_{pbf} qui les distinguent.

En revanche, la logique L_P semble plus naturelle pour la spécification de propriétés. Comme nous disposons d'algorithmes de traduction de L_P vers L_{pbf} (et vice versa), la logique L_P peut être utilisée comme le langage d'entrée d'un éventuel vérificateur automatique basé sur la logique L_{pbf}.

Chapitre 8

Bisimulations avant-arrière

La pomset bisimulation avant-arrière $\approx_{pbf}$ définie en Section 6.3.2 possède la propriété de transfert "en avant" pour des transitions étiquetées par des actions et "en arrière" pour des transitions étiquetées par des pomsets. En fait cette définition bien qu'apparemment plus faible que celle de [Che92a] est équivalente.

Dans ce chapitre nous introduisons quinze variantes de la Définition 6.3.3 de $\approx_{pbf}$. Dans ces variantes les propriétés de transfert (en avant et en arrière) portent sur des pomsets restreints comme par exemple des steps ou des actions. Nous obtenons finalement une classification des bisimulations avant-arrière illustrée par la Figure 8.3 page 127.

8.1 Définitions et résultats préliminaires

Notations 14 *Soient p et q deux pomsets. On dira que q* est moins séquentiel que *p, noté $q \sqsubseteq p$, si le pomset q contient moins de liens de causalité que p. Par exemple $a \,|\, b \sqsubseteq a;b$.*

La Définition 8.1.1 regroupe seize définitions d'équivalences dont la Définition 6.3.3 de $\approx_{pbf}$ et celle de [Che92a].

Définition 8.1.1 (Bisimulations x-avant y-arrière) *Soient $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$. Une relation symétrique $R \subseteq Hist(\mathcal{E}) \times Hist(\mathcal{F}) \bigcup Hist(\mathcal{F}) \times Hist(\mathcal{E})$ est une* bisimu-

lation x-avant y-arrière *entre* $\mathcal{E}$ *et* $\mathcal{F}$, *noté* $R : \mathcal{E} \approx_{xbyf} \mathcal{F}$, *pour* $x, y \in \{i, s, pw, p\}$, *si* $\lambda_{\mathcal{E}} R \lambda_{\mathcal{F}}$, *et si pour tous* $\sigma R \rho$,

- *(en avant): si* $\sigma \xrightarrow{\theta} \sigma'$ *où selon* x :
  - $x = i$: $pom(\theta)$ *est une action*[1]
  - $x = s$: $pom(\theta)$ *est un step*

  alors il existe ρ' *et* θ' *tels que* $\rho \xrightarrow{\theta'} \rho'$ *avec* $\sigma' R \rho'$ *et selon* x :
  - $x \in \{i, s, p\}$: $pom(\theta) = pom(\theta')$
  - $x = pw$: $pom(\theta') \sqsubseteq pom(\theta)$
- *(en arrière): si* $\sigma' \xrightarrow{\theta} \sigma$ *où selon* y :
  - $y = i$: $pom(\theta)$ *est une action,*
  - $y = s$: $pom(\theta)$ *est un step*

  alors il existe ρ' *et* θ' *tels que* $\rho' \xrightarrow{\theta'} \rho$ *avec* $\sigma' R \rho'$ *et selon* y :
  - $y \in \{i, s, p\}$: $pom(\theta) = pom(\theta')$
  - $y = pw$: $pom(\theta') \sqsubseteq pom(\theta)$

On montre facilement que pour tous $x, y \in \{i, s, pw, p\}\Gamma \approx_{xbyf}$ est une relation d'équivalence sur **SE** et que pour toutes $\mathcal{E}, \mathcal{F} \in$ **SE** et tous $x, y \in \{i, s, pw, p\}$ il existe une plus grande relation $R : \mathcal{E} \approx_{xbyf} \mathcal{F}$.

Certaines équivalences de la Définition 8.1.1Γ ont déjà été proposées dans la littérature. Nous pouvons déjà reconnaître en $\approx_{pbpf}$ la pomset bisimulation avant-arrière de [Che92a].

Nous verrons plus loin que l'équivalence $\approx_{ibpwf}$ n'est autre que la *partial word bisimulation* de [Vog91] (voir la Définition 8.1.3). Cette équivalence est une version affaiblie de la pomset bisimulation de [BC87] puisque dans l'imitation d'une transition il n'est pas nécessaire d'observer exactement le même pomsetΓmais un pomset éventuellement plus "parallèle"Γ c-à-d. contenant moins de dépendances causales entre ces occurrences d'actions.

Nous donnons tout d'abord la définition d'un homomorphisme que nous utilisons en Définition 8.1.3.

Définition 8.1.2 *Soient* $\mathcal{E} = (E_{\mathcal{E}}, \leq_{\mathcal{E}}, \#_{\mathcal{E}}, l_{\mathcal{E}})$ *et* $\mathcal{F} = (E_{\mathcal{F}}, \leq_{\mathcal{F}}, \#_{\mathcal{F}}, l_{\mathcal{F}})$ *deux SE sans-conflit. Un* homomorphisme *entre* $\mathcal{E}$ *et* $\mathcal{F}$ *est une bijection* $f : E_{\mathcal{E}} \mapsto E_{\mathcal{F}}$ *qui préserve l'étiquetage, i.e. pour tout* $e \in E_{\mathcal{E}}$, $l_{\mathcal{F}}(f(e)) = l_{\mathcal{F}}(e)$, *et la relation de causalité, i.e. pour tous* $e_1, e_2 \in E_{\mathcal{E}}$, $e_1 \leq_{\mathcal{E}} e_2$ *implique* $f(e_1) \leq_{\mathcal{F}} f(e_2)$.

[1] *Les transitions étiquetées par une action sont celles que l'on considère habituellement dans les modèles d'entrelacement ou encore "interleaving", d'où le label* i *correspondant.*

Définition 8.1.3 (Partial word bisimulation) [Vog91] *Soient $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$. Une relation symétrique $R \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{F}) \times \mathcal{P}(E_{\mathcal{E}} \times E_{\mathcal{F}}) \bigcup \mathcal{C}(\mathcal{F}) \times \mathcal{C}(\mathcal{E}) \times \mathcal{P}(E_{\mathcal{F}} \times E_{\mathcal{E}})$ est une* partial word bisimulation *entre $\mathcal{E}$ et $\mathcal{F}$, ce que l'on note $R : \mathcal{E} \approx_{pw} \mathcal{F}$, si*

- $(\emptyset, \emptyset, \emptyset) \in R$,
- *pour tout $(C, D, f) \in R$, $f : C \to D$ est une bijection qui respecte l'étiquetage,*
- *si $C \to C'$ alors il existe D', f' tels que $D \to D'$, $(C', D', f') \in R$, $f'_{|C} = f$ et $f'^{-1}_{|D'-D}$ est un homomorphisme de $D' - D$ vers $C' - C$.*

Nous écrivons $\mathcal{E} \approx_{pw} \mathcal{F}$ si $R : \mathcal{E} \approx_{pw} \mathcal{F}$ pour une relation R.

Dans la Définition 8.1.3Γ la bijection f'^{-1} se réduit à un homomorphisme de $D' - D$ vers $C' - C$. Cette condition exprime exactement que $pomset(D' - D) \sqsubseteq pomset(C' - C)$.

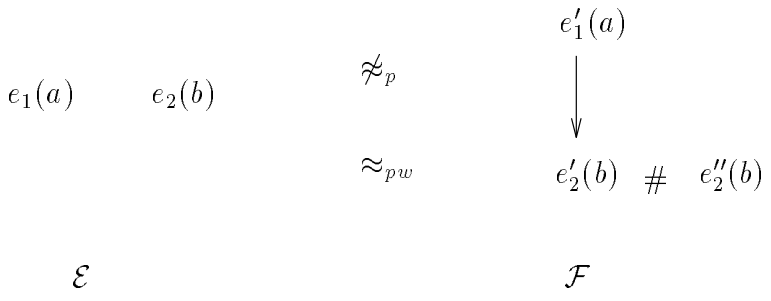


Figure 8.1: $\approx_{pw}$ n'implique pas $\approx_p$

Nous rappelons au passage les résultats de [Vog91] qui situent $\approx_{pw}$ par rapport aux équivalences de la Section 6.1.

Proposition 8.1.1 [Vog91] *Pour tous $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$,*

$$\mathcal{E} \approx_p \mathcal{F} \text{ implique } \mathcal{E} \approx_{pw} \mathcal{F} \text{ implique } \mathcal{E} \approx_s \mathcal{F}$$

[Vog91] montre que ces implications sont strictes. On peut voir en Figures 8.1 et 8.4.

8.2 Classification des bisimulations avant-arrière

Par définitionΓles équivalences introduites en Définition 8.1.1 peuvent être organisées de la manière suivante : pour simplifier l'énoncé de l'Equation (8.12)Γ nous introduisons un ordre sur l'ensemble $\{p, pw, s, i\}$. Soit $\leq$ le plus petit ordre contenant $i \leq s \leq pw \leq p$. Alors pour tous $x, x', y, y' \in \{i, s, pw, p\}$Γ

$$x \leq x' \text{ et } y \leq y' \text{ impliquent } \approx_{x'by'f} \subseteq \approx_{xbyf} \tag{8.12}$$

et de plusΓ

Proposition 8.2.1 *[Che92c] Pour tous $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$,*

$$\mathcal{E} \approx_{pbpf} \mathcal{F} \textit{ ssi } \mathcal{E} \approx_{pwbpwf} \mathcal{F} \textit{ implique } \mathcal{E} \approx_{sbsf} \mathcal{F} \textit{ implique } \mathcal{E} \approx_{ibif} \mathcal{F}$$

où les deux dernières implications sont strictes. De plus,

$$\mathcal{E} \approx_{ibif} \mathcal{F} \textit{ ssi } \mathcal{E} \approx_{i} \mathcal{F}$$

La Proposition 8.2.2 qui suit exprime que si des SE peuvent effectuer les mêmes actions en avant et les mêmes pomsets (resp. partial wordsΓsteps) en arrièreΓalors elles peuvent effectuer les mêmes pomsets (resp. partial wordsΓsteps) en avant. Un aspect pratique de ce résultat est qu'il suffit désormais pour montrer que deux SE sontΓpar exempleΓ$\approx_{pbpf}$ de ne vérifier "en avant" que les transitions à une action.

Proposition 8.2.2 *Pour tout $x \in \{i, s, pw, p\}$, et tous $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$,*

$$\mathcal{E} \approx_{xbif} \mathcal{F} \textit{ implique } \mathcal{E} \approx_{xbxf} \mathcal{F}$$

Preuve Le résultat est évident pour $x = i$.

Nous traitons à titre d'exemple le cas $x = p$.
Soit $R : \mathcal{E} \approx_{pbif} \mathcal{F}$. On montre que $R : \mathcal{E} \approx_{pbpf} \mathcal{F}$. Il suffit de vérifier la clause "en avant" de la Définition 8.1.1. Soit $\sigma R \rho$ et supposons $\sigma \xrightarrow{p} \sigma'$. La suite de transitions qui mènent de σ à σ' peut être pas à pas imitée par une suite de transitions partant de ρ et menant à un certain ρ' tel que $\sigma' R \rho'$. Il faut vérifier que le pomset calculé de ρ à ρ' est bien p : c'est clairement le cas en utilisant la propriété de transfert "en arrière" de R.

Des raisonnements analogues permettent de traiter les deux derniers cas $x = s$ et $x = pw$. □

Il nous sera utile dans la suite de notre étude de remarquer que les équivalences $\approx_{ibif}$Γ$\approx_{ibsf}$Γ$\approx_{ibpf}$ et $\approx_{ibpwf}$ coïncident respectivement avec les équivalences $\approx_i$Γ$\approx_s$Γ $\approx_p$ et $\approx_{pw}$. Nous omettons cette preuve qui est une généralisation de [DMV90] où il est montré que la bisimulation forte coïncide avec sa version avant-arrière. Ce résultat exprime que l'on ne gagne rien à exiger de pouvoir imiter les transitions d'actions en arrière si on peut déjà les imiter en avant.

Les résultats obtenus jusqu'ici sont résumés par la Figure 8.2 où les boîtes en pointillés regroupent les équivalences qui coïncident et les flèches expriment les implications immédiates (celles obtenues par transitivité ne sont pas représentées). Nous simplifions cette figure en la Figure 8.3 pour laquelle nous montrons qu'aucune flèche ne peut y être ajoutée (hormis celles obtenues par transitivité).

Quand nous le pourronsΓnous utiliserons le formalisme de la logique L_{pbf} (présentée au Chapitre 7) pour expliquer nos contre-exemples.

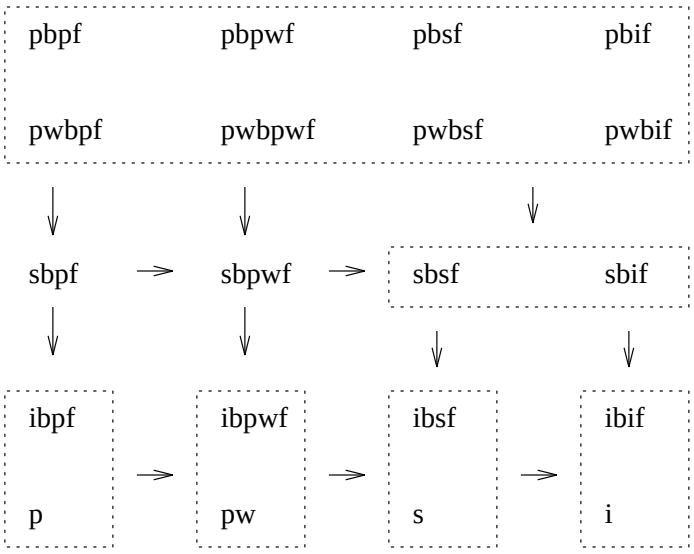


Figure 8.2: Les liens entre les bisimulations avant-arrière

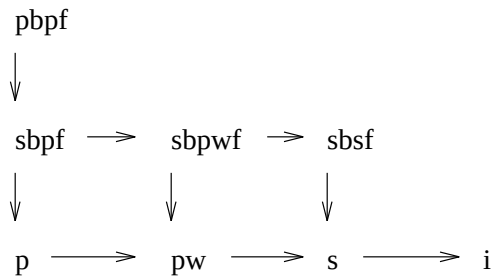


Figure 8.3: Classification des bisimulations avant-arrière

- D'après [Vog91]Γles implications qui relient $\approx_p$Γ$\approx_{pw}$Γ$\approx_s$ et $\approx_i$ sont strictes.
- La Figure 6.4 montre que $\approx_p$ n'implique pas $\approx_{sbsf}$. En effetΓ $\mathcal{E} \approx_p \mathcal{F}$Γmais seule la SE $\mathcal{E}$ satisfait la formule de L_{pbf} $[a]\langle b\rangle\langle \leftarrow a \mid b\rangle\top$ qui montre que $\mathcal{E} \not\approx_{sbsf} \mathcal{F}$.

  Nous en déduisons qu'aucune flèche dans la Figure 8.3 ne peut remonter de la dernière ligne à l'avant-dernière ligne.
- La Figure 8.4 prouve que $\approx_{sbsf}$ n'implique pas $\approx_{pw}$.

  On en déduit alors que $\approx_{sbsf}$ n'implique pas $\approx_{sbpwf}$.
- La Figure 8.5 montre que $\approx_{sbpwf}$ n'implique pas $\approx_p$. En effetΓseule $\mathcal{E}$ peut effectuer le pomset $(a \mid b);c$.

Cet exemple montre également que $\approx_{sbpwf}$ n'implique pas $\approx_{sbpf}$.

Des deux points précédentsΓnous déduisons qu'aucune autre flèche ne peut descendre vers la dernière ligne de la Figure 8.3.

- Observons que les SE de la Figure 8.6 sont $\approx_{sbpf}$-équivalentes mais que seule $\mathcal{E}$ satisfait la formule de L_{pbf} $[a;a]\langle b\rangle\langle\leftarrow (a;a) \mid b\rangle\top$. Par conséquentΓ$\approx_{sbpf}$ n'implique pas $\approx_{pbpf}$.

  On peut alors déduire que $\approx_{sbpwf}$ n'implique pas $\approx_{pbpf}$ et que $\approx_{sbsf}$ n'implique pas $\approx_{pbpf}$ (ce qui a déjà été établi par [Che92c]).

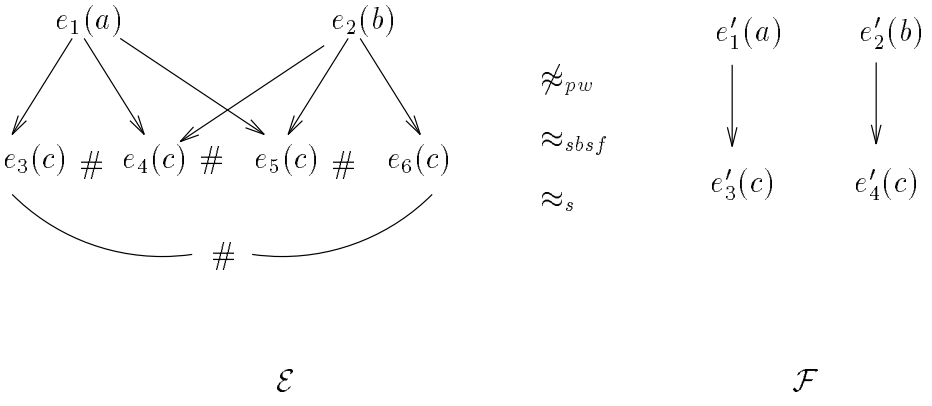


Figure 8.4: $\approx_{sbsf}$ n'implique pas $\approx_{pw}$

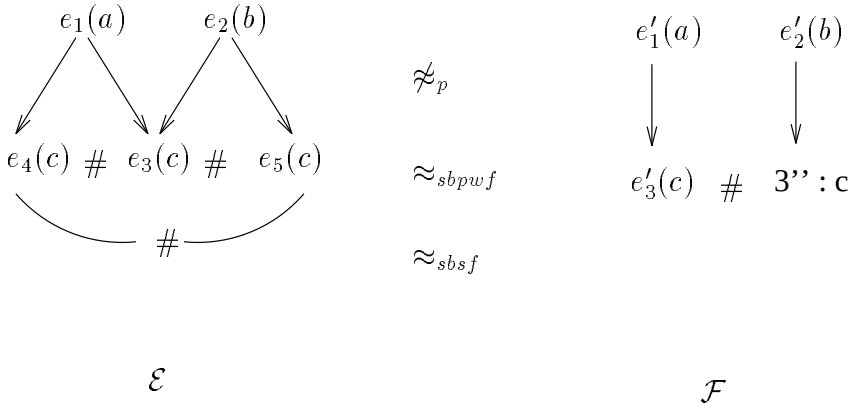


Figure 8.5: $\approx_{sbpwf}$ n'implique pas $\approx_p$

Nous venons de classifier les bisimulations avant-arrière dans les structures d'événements (premières) sans action invisible.

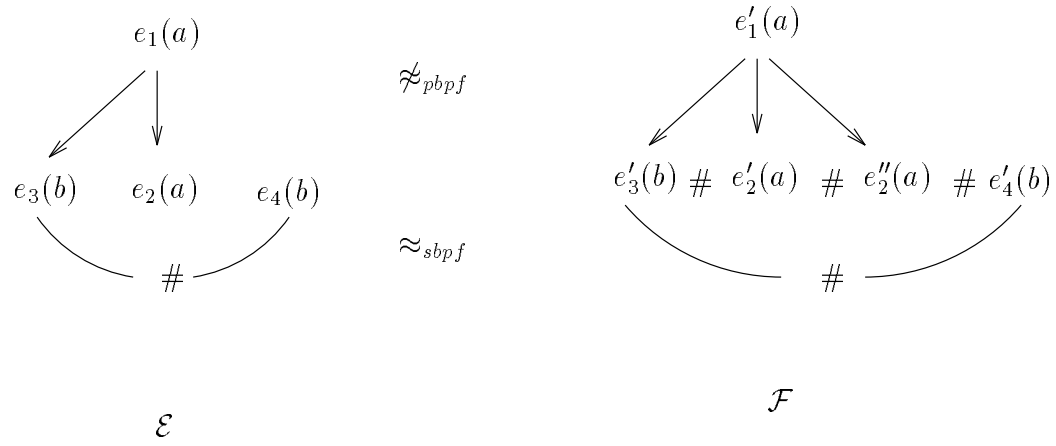


Figure 8.6: $\approx_{sbpf}$ n'implique pas $\approx_{pbpf}$

Sur les 16 équivalences considérées dans la Définition 8.1.1Γ nous obtenons 8 équivalences distinctes. Dans la section suivanteΓ nous étudions les propriétés de ces 8 équivalences vis-à-vis du raffinement d'actions.

8.3 Raffinement d'actions

Cette classification des bisimulations avant-arrière sur les SE (sans action invisible) nous éclaire un peu plus sur la situation générale. La plupart de ces équivalences existent déjà dans la littérature et ont été très largement étudiéesΓet il est possible de répondre si oui ou non elle est préservée par le raffinement d'actions. Seules les deux équivalences $\approx_{sbpf}$ et $\approx_{sbpwf}$ n'ont pasΓà notre connaissanceΓété définies dans la littérature.

Rappelons que dans [Vog91]Γles équivalences $\approx_i$Γ$\approx_s$Γ$\approx_{pw}$Γ$\approx_p$ et $\approx_{hp}$ (i.e. $\approx_{pbf}$) sont étudiées pour le raffinement d'actions. Il est montré que seule $\approx_{hp}$ se comporte bien vis-à-vis de cette opération. De plusΓ[Che92c] a montré que $\approx_{sbsf}$ n'est pas préservée par le raffinement et nous empruntons son contre-exemple pour notre étude.

Proposition 8.3.1 *L'équivalence $\approx_{sbpwf}$ n'est pas préservée par le raffinement d'actions.*

Preuve Reprenons l'exemple de [Che92c] dessiné en Figure 8.7. Ces SE sont aussi $\approx_{sbpwf}$-équivalentes. On peut facilement vérifier que les transitions en arrièreΓoù seuls des steps sont considérésΓsont toutes imitables dans l'une et l'autre des SE. Le seul cas à vérifier est celui où $\mathcal{E}$ calcule le pomset $q \stackrel{\text{def}}{=} (a \mid b);c$. Si $\mathcal{E}$ effectue

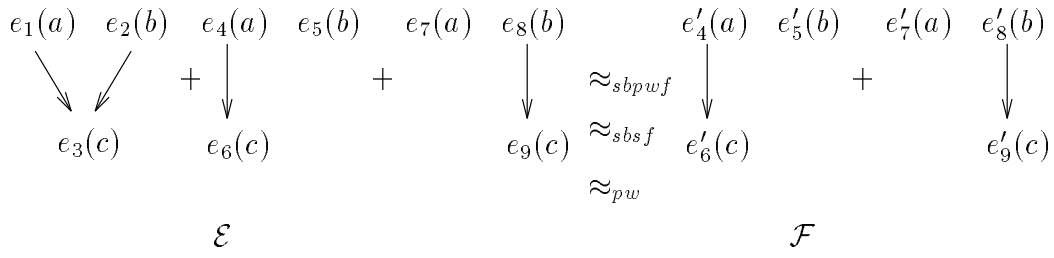


Figure 8.7: $\mathcal{F} = ((a;c) \,|\, b) + (a \,|\, (b;c))$ et $\mathcal{E} = \mathcal{F} + ((a \,|\, b);c)$

$\lambda_{\mathcal{E}} \xrightarrow{q} e_1e_2e_3\Gamma$alors $\mathcal{F}$ peut l'imiter par le pomset moins séquentiel $\lambda_{\mathcal{F}} \overset{a\|(b;c)}{\longrightarrow} e'_1e'_2e'_3$. Sinon$\Gamma$c-à-d. si $\mathcal{E}$ effectue $\lambda_{\mathcal{E}} \xrightarrow{q} e_2e_1e_3\Gamma\mathcal{F}$ l'imite par $\lambda_{\mathcal{F}} \overset{(a;c)\|b}{\longrightarrow} e'_2e'_1e'_3$.
Si maintenant on raffine $\mathcal{E}$ et $\mathcal{F}$ par $ref(a) = a_1;a_2\Gamma$ alorsΓ comme [Che92a] le remarque$\Gamma ref(\mathcal{E}) \not\approx_i ref(\mathcal{F})\Gamma$et donc $ref(\mathcal{E}) \not\approx_{sbpwf} ref(\mathcal{F})$. □

Corollaire 8.3.2 *Aucune équivalence comprise entre $\approx_{sbpwf}$ et $\approx_i$ n'est compatible avec le raffinement d'actions.*

Le tableau ci-dessus récapitule les propriétés des bisimulations avant-arrière sur **SE** vis à vis du raffinement d'actions :

	est préservée par le raffinement
$\approx_{pbf}$	oui
$\approx_{sbpf}$	?
$\approx_{sbpwf}$	non
$\approx_{sbsf}$	non
$\approx_{p}$	non
$\approx_{pw}$	non
$\approx_{s}$	non
$\approx_{i}$	non

Nous n'avons pas résolu la question pour l'équivalence $\approx_{sbpf}$. Nous conjecturons qu'elle n'est pas préservée par le raffinement d'actions.

Chapitre 9

Equivalences comportementales avec action invisible

Dans ce Chapitre nous généralisons l'équivalence $\approx_{pbf}$ du Chapitre 6 au cas de la présence d'actions inobservables. L'équivalence obtenue est appelée la *τ-pomset bisimulation avant-arrière*[1] et est notée $\approx^{\tau}_{bf}$.

Comme nous le verrons l'équivalence $\approx^{\tau}_{bf}$ ne correspond à aucune équivalence déjà proposée dans la littérature.

A partir de cette équivalence nous établissons une équation similaire à l'Equation (6.9) du Chapitre 6 : nous généralisons de façon adéquate les équivalences $\approx_c$ et $\approx_{mo}$ et $\approx_{hp}$ en respectivement $\approx^b_c$ et $\approx^b_{mo}$ et $\approx^b_{hp}$ voir la Section 9.2. Par exemple la généralisation de l'équivalence $\approx_c$ est la branching bisimulation [GW89a] (voir le Chapitre 2) sur les arbres causaux notée $\approx^b_c$. Pour les généralisations des équivalences $\approx_{hp}$ et $\approx_{mo}$ nous introduisons leur version "branching" obtenant l'*équivalence de mixed-ordering branching* $\approx^b_{mo}$ et la *history preserving branching bisimulation* $\approx^b_{hp}$.

Concernant la *history preserving branching bisimulation* nous pensons que cette équivalence est d'une certaine manière la bonne généralisation de $\approx_{hp}$ dans la mesure où elle satisfait l'Equation (9.13) qui généralise (6.9).

Nous consacrons le reste du chapitre à son étude. Dans un premier temps nous

[1] que l'on pourrait traduire en anglais par *pomset weak bisimulation.*

situons cette nouvelle équivalence par rapport aux généralisations déjà existantes de la history preserving bisimulation.

Il est par exemple immédiat de voir que $\approx_{hp}^{b}$ est plus fine que la *history preserving τ-bisimulation* de [Ace91] en comparant soit directement leurs définitions soit les équivalences causales associées.

En revanche nous établissons que $\approx_{hp}^{b}$ et la history preserving ST bisimulation de [Vog91] ici notée $\approx_{hp}^{ST}$ sont incomparables. Le fait que $\approx_{hp}^{ST}$ ne soit pas plus fine que $\approx_{hp}^{b}$ est assez naturel. Par contre le résultat inverse est surprenant si l'on sait que dans le cas des systèmes séquentiels l'équivalence de [Vog91] aussi connue comme la *maximality preserving bisimulation* [Dev90] coïncide avec la delay bisimulation elle-même contenue dans la branching bisimulation.

Nous tenons à préciser que le contre-exemple de la Figure 9.8, qui montre que $\approx_{hp}^{b}$ n'est pas plus fine que l'équivalence $\approx_{hp}^{ST}$ de [Vog91], a été proposé par Robert van Glabbeek après la soutenance de la thèse ([Gla93].

A partir de ce contre-exemple il est immédiat de déduire que $\approx_{hp}^{b}$ n'est pas compatible avec le raffinement car sinon elle impliquerait la plus grande congruence pour le raffinement contenue dans la history preserving τ-bisimulation à savoir l'équivalence $\approx_{hp}^{ST}$ (voir [Vog91]).

De plus le contre-exemple pour le raffinement est obtenu en appliquant un raffinement très simple aux SE de la Figure 9.8.

Nous considérons dans la suite un alphabet d'actions Act auquel nous ajoutons l'action invisible τ. Act_τ désigne l'ensemble $Act \cup \{\tau\}$.

Soit $\mathcal{E} \in \mathbf{SE}$ et $\theta \subseteq E_\mathcal{E}$. Nous définissons la partie visible de θ par

$$vis(\theta) \stackrel{\text{def}}{=} \{e \in \theta | l(e) \neq \tau\}$$

L'ensemble $vis(\theta)$ est canoniquement muni de l'ordre $\leq_\mathcal{E}$.

Notations 15 *Pour alléger les notations, nous posons*

- $vis_\mathcal{E} \stackrel{def}{=} vis(E_\mathcal{E})$, *et*
- $pomvis(\theta) \stackrel{def}{=} pom(vis(\theta))$.

Nous définissons $\Rightarrow \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{P}(vis_\mathcal{E}) \times \mathcal{C}(\mathcal{E})$ la *relation de transition modulo τ* dans $\text{Conf}(\mathcal{E})$ par : pour tous $C, C' \in \mathcal{C}(\mathcal{E})$

$$C \stackrel{\mu}{\Rightarrow} C' \text{ ssi } C \stackrel{\theta}{\rightarrow} C' \text{ et } vis(\theta) = \mu$$

et pour tout pomset p sur Act

$$C \stackrel{p}{\Rightarrow} C' \text{ ssi } C \stackrel{\theta}{\rightarrow} C' \text{ et } pomvis(\theta) = p$$

Lorsqu'il n'y a pas d'ambiguïté, nous écrivons $C \Rightarrow C'$ au lieu de $C \stackrel{\emptyset}{\Rightarrow} C'$.
Nous étendons canoniquement les définitions de vis, $\stackrel{\mu}{\rightarrow}$ et $\stackrel{p}{\Rightarrow}$ aux histoires.

9.1 La pomset τ-bisimulation avant-arrière

La définition de la pomset τ-bisimulation avant-arrière généralise canoniquement celle de $\approx_{pbf}$: les propriétés de transfert "en avant" et "en arrière" portent cette fois sur les transitions modulo τ.

Définition 9.1.1 (pomset τ-bisimulation avant-arrière) *Soient $\mathcal{E}, \mathcal{F} \in$ **SE**. Une relation symétrique $R \subseteq Hist(\mathcal{E}) \times Hist(\mathcal{F}) \bigcup Hist(\mathcal{F}) \times Hist(\mathcal{E})$ est une* pomset τ-bisimulation avant-arrière *entre $\mathcal{E}$ et $\mathcal{F}$, noté $R : \mathcal{E} \approx^{\tau}_{bf} \mathcal{F}$, si :*

- $\lambda_{\mathcal{E}} R \lambda_{\mathcal{F}}$,
- *pour tous $\sigma R \rho$,*
  - *(en avant): si $\sigma \stackrel{a}{\Rightarrow} \sigma'$ alors il existe $\rho \stackrel{a}{\Rightarrow} \rho'$ tel que $\sigma' R \rho'$*
  - *(en arrière): si $\sigma' \stackrel{p}{\Rightarrow} \sigma$ alors il existe $\rho' \stackrel{p}{\Rightarrow} \rho$ tel que $\sigma' R \rho'$,*

On note $\mathcal{E} \approx^{\tau}_{bf} \mathcal{F}$ lorsqu'il existe $R : \mathcal{E} \approx^{\tau}_{bf} \mathcal{F}$.

Remarque 10 *On peut proposer une définition équivalente à la Définition 9.1.1 en autorisant dans la clause "en avant" des transitions (modulo τ) quelconques. Ce résultat généralise la Proposition 8.2.2.*

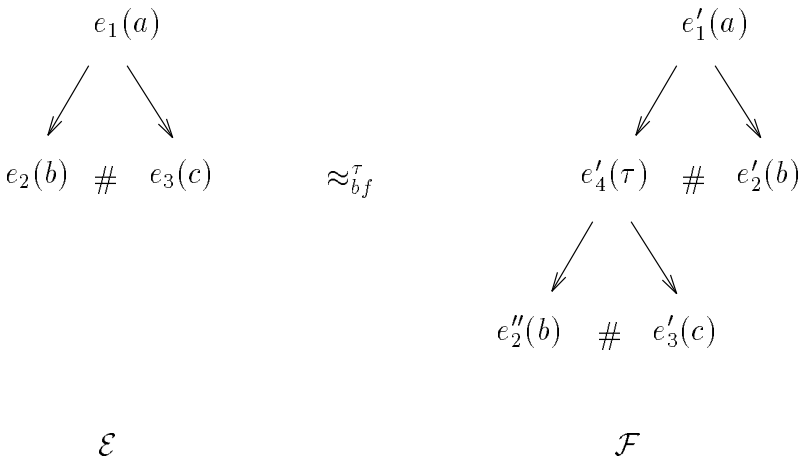


Figure 9.1: Un exemple de SE équivalentes pour $\approx^{\tau}_{bf}$.

Il est facile de montrer que $\approx^{\tau}_{bf}$ est une relation d'équivalence sur **SE**, et qu'il existe toujours une plus grande pomset τ-bisimulation avant-arrière entre deux SE.

Nous notons simplement $\approx^{\tau}_{bf}$ la plus grande pomset τ-bisimulation avant-arrière entre deux SE. Ainsi, dans la suite comme par exemple dans l'énoncé du Lemme 9.1.1, une expression telle que $\sigma \approx^{\tau}_{bf} \rho$ signifie que les histoires σ et ρ sont reliées par la plus grande pomset τ-bisimulation avant-arrière (entre $\mathcal{E}$ et $\mathcal{F}$).

Exemple 20 *Les SE de la Figure 9.1 sont équivalentes pour $\approx^{\tau}_{bf}$. Par contre celles de la Figure 9.7 sont distinguées par $\approx^{\tau}_{bf}$. En effet, la transition $\lambda_{\mathcal{F}} \stackrel{\emptyset}{\rightarrow}_{\mathcal{F}} e''_1$ est forcément imitée dans $\mathcal{E}$ par $\lambda_{\mathcal{E}} \stackrel{\emptyset}{\rightarrow}_{\mathcal{E}} e_1 e_2$. Mais alors la transition en arrière $e_1 \stackrel{\emptyset}{\rightarrow} e_1 e_2$ n'a pas de correspondant dans $\mathcal{F}$:*

- *elle ne peut pas être imitée dans $\mathcal{F}$ sans bouger, car e_1 et e''_1 ne sont pas des histoires bisimilaires ; e_1 peut évoluer en $e_1 \stackrel{b}{\Rightarrow} e_1 e_3$ qui n'est pas imitable en partant de e''_1.*
- *elle ne pas non plus être imitée par la transition arrière $\lambda_{\mathcal{F}} \stackrel{\emptyset}{\rightarrow} e''_1$ car e_1 et $\lambda_{\mathcal{F}}$ ne sont pas bisimilaires ; seule $\lambda_{\mathcal{F}}$ peut effectuer une a-transition.*

Dans la suite nous prouvons que l'équivalence $\approx^{\tau}_{bf}$ possède la propriété du *X-lemma* (voir la X-propriété dans [DMV90] ou encore le *stuttering lemma* dans [GW89a]) tout comme la plupart des équivalences modulo τ. Cette propriété nous servira pour simplifier certaines preuves. Elle est illustrée en Figure 9.2.

Intuitivement, la propriété du X-lemma exprime que l'on peut identifier des états appartenant à une même boucle de τ.

Lemme 9.1.1 (X-lemma)

Soit $R : \mathcal{E} \approx^{\tau}_{bf} \mathcal{F}$. Pour tous $\sigma, \sigma' \in Hist(\mathcal{E})$ et tous $\rho, \rho' \in Hist(\mathcal{F})$, si $\sigma \stackrel{\emptyset}{\Rightarrow} \sigma'$, $\rho \stackrel{\emptyset}{\Rightarrow} \rho'$, $\sigma R \rho'$ et $\sigma' R \rho$, alors $\sigma \approx^{\tau}_{bf} \rho$ et $\sigma' \approx^{\tau}_{bf} \rho'$.

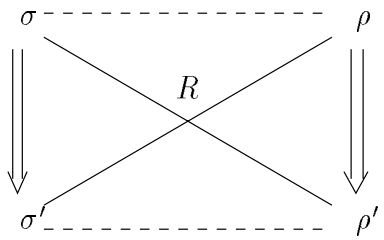


Figure 9.2: X-lemma

Preuve Elle est classique, nous renvoyons par exemple à [DMV90] pour la technique de preuve. □

9.2 Caractérisations de $\approx^{\tau}_{bf}$

Nous proposons dans cette section d'autres façons d'exprimer l'équivalence $\approx^{\tau}_{bf}$. En particulier il est possible de la caractériser par une équivalence causale : la branching bisimulation de [GW89a] dans les arbres causaux. Pour établir ce résultat nous décomposons la preuve en deux étapes.

La première étape consiste à montrer qu'elle coïncide avec l'équivalence de *mixed-ordering branching* que nous introduisons à cette occasion. Cette approche est très similaire aux travaux de [DMV90] pour les systèmes de transitions.

La deuxième étape établit l'égalité entre l'équivalence de *mixed-ordering branching* et la branching bisimulation dans les arbres causaux.

Finalement l'équivalence $\approx^{b}_{mo}$ conduit naturellement à proposer la history preserving branching bisimulation que nous étudions ensuite.

9.2.1 L'équivalence de mixed-ordering branching

Notations 16 *Si $\sigma \in Hist(\mathcal{E})$ et $\rho \in Hist(\mathcal{F})$ sont telles que $|vis(\sigma)|=|vis(\rho)|=n$, g^{ρ}_{σ} dénote la fonction de $vis(C_\sigma)$ dans $vis(D_\rho)$ de graphe $\{(vis(\sigma)(i), vis(\rho)(i))|1 \leq i \leq n\}$. Dans le cas sans τ, g^{ρ}_{σ} coïncide avec f^{ρ}_{σ}, la fonction définie en Section 6.3.1.*

La Définition qui suit est la version "branching" de l'équivalence de mixed-ordering de [Ace91].

Définition 9.2.1 (L'équivalence de mixed-ordering branching)
Une équivalence de mixed-ordering branching *entre $\mathcal{E}$ et $\mathcal{F}$ est une relation symétrique $R \subseteq Hist(\mathcal{E}) \times Hist(\mathcal{F}) \cup Hist(\mathcal{F}) \times Hist(\mathcal{E})$ telle que $\lambda_{\mathcal{E}} R \lambda_{\mathcal{F}}$, et pour tout $\sigma R \rho$,*

- $g^{\rho}_{\sigma} : vis(C_\sigma) \rightarrow vis(D_\rho)$ *est un isomorphisme,*
- *si $\sigma \xrightarrow{e} \sigma'$, alors*
  - *ou bien $l(e) = \tau$ et $\sigma' R \rho$,*
  - *ou bien il existe $\rho \Rightarrow \rho'' \rightarrow \rho'$ tels que $\sigma R \rho''$ et $\sigma' R \rho'$.*

On note $R : \mathcal{E} \approx^{b}_{mo} \mathcal{F}$, lorsque R est une équivalence de mixed-ordering branching entre $\mathcal{E}$ et $\mathcal{F}$, et $\mathcal{E} \approx^{b}_{mo} \mathcal{F}$ si il existe R telle que $R : \mathcal{E} \approx^{b}_{mo} \mathcal{F}$.

Notons que dans la Définition 9.2.1 nous avons forcément $\rho \stackrel{\emptyset}{\Rightarrow} \rho''$.

Proposition 9.2.1 *Pour tous $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3 \in$ **SE** :*

(1) $Id_Q : \mathcal{E} \approx^{b}_{mo} \mathcal{E}$,

(2) $R : \mathcal{E}_1 \approx^{b}_{mo} \mathcal{E}_2$ *implique* $R^{-1} : \mathcal{E}_2 \approx^{b}_{mo} \mathcal{E}_1$,

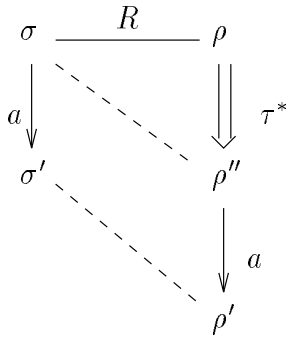


Figure 9.3: Propriété de transfert de $\approx^b_{mo}$.

(3) $R : \mathcal{E}_1 \approx^b_{mo} \mathcal{E}_2$ *et* $R' : \mathcal{E}_2 \approx^b_{mo} \mathcal{E}_3$ *impliquent* $(R' \circ R) : \mathcal{E}_1 \approx^b_{mo} \mathcal{E}_3$,
(4) $R : \mathcal{E}_1 \approx^b_{mo} \mathcal{E}_2$ *et* $R' : \mathcal{E}_1 \approx^b_{mo} \mathcal{E}_2$ *impliquent* $(R \cup R') : \mathcal{E}_1 \approx^b_{mo} \mathcal{E}_2$.

La propriété (4) se généralise aux unions arbitraires. Par conséquent il existe toujours une plus grande équivalence de mixed-ordering branching que l'on note simplement $\approx^b_{mo}$.

Proposition 9.2.2 *Pour tous* $\mathcal{E}, \mathcal{F} \in \mathbf{SE}$,

$$\mathcal{E} \approx^b_{mo} \mathcal{F} \text{ ssi } \mathcal{E} \approx^\tau_{bf} \mathcal{F}$$

Preuve Nous introduisons les notations suivantes :
Pour toute $\mathcal{E} \in SE$ et toute $\sigma = e_1 \ldots e_n \in Hist(\mathcal{E})$ nous notons

$$\overline{\sigma} \stackrel{\text{def}}{=} \{\sigma' \in Hist(\mathcal{E}) | \sigma \approx^b_{mo} \sigma'\}$$

la classe d'équivalence de σ dans $Hist(\mathcal{E})$ pour la plus grande équivalence de mixed-ordering branching de $\mathcal{E}$ dans $\mathcal{E}$.

Nous définissons alors *la trace de* σ par :

$$t(\sigma) \stackrel{\text{def}}{=} \lambda.l(e_1).\overline{\sigma^1}.l(e_2).\overline{\sigma^2} \ldots \overline{\sigma^{n-1}}.l(e_n).\overline{\sigma^n}$$

En abstrayant les τ "inoffensifs" c-à-d. en enlevant de $t(\sigma)$ les sous-chaînes $\tau.\overline{\sigma^i}$ pour lesquelles $\sigma^{i-1} \approx^b_{mo} \sigma^i$ nous obtenons la *trace de* σ *modulo stuttering* notée $stutt(\sigma)$.

Cette notion est exactement celle de la *trace colorée* dans [DMV90] où il est montré que (dans les systèmes de transitions) la τ-bisimulation avant-arrière et la branching bisimulation coïncident.

Nous pouvons maintenant démontrer la Proposition 9.2.2.

$\Rightarrow$) : Soit $R : \mathcal{E} \approx^{b}_{mo} \mathcal{F}$ et soient $\sigma \in Hist(\mathcal{E})$ et $\rho \in Hist(\mathcal{F})$. Si $stutt(\sigma) = \mathcal{C}_1.a_1.\mathcal{C}_2 \ldots \mathcal{C}_{n-1}.a_n.\mathcal{C}_n$ et $stutt(\rho) = \mathcal{C}'_1.b_1.\mathcal{C}'_2 \ldots \mathcal{C}'_{m-1}.b_m.\mathcal{C}'_m$Γnous notons $stutt(\sigma) \sim stutt(\rho)$ si :

- $n = m$Γ
- $a_i = b_i$ pour tout $i = 1, \ldots, n$Γ
- pour tous $\sigma' \in \mathcal{C}_i$Γ$\rho' \in \mathcal{C}'_i$Γ$\sigma' \approx^{b}_{mo} \rho'$.

Nous montrons alors que $S \stackrel{\text{def}}{=} \{(\sigma, \rho) | stutt(\sigma) \sim stutt(\rho)\} : \mathcal{E} \approx^{\tau}_{bf} \mathcal{F}$.

Par constructionΓ$S \subseteq R$ et $\lambda_{\mathcal{E}} S \lambda_{\mathcal{F}}$.
Soient $\sigma S \rho$. Puisque $S \subseteq R$ΓS a évidemment la propriété de transfert en avant. Supposons maintenant donnée une transition en arrière $\sigma' \stackrel{p}{\Rightarrow} \sigma$. Alors $\sigma' \in \mathcal{C}_i$Γ pour un $i \in \{1, \ldots, n\}$. Puisque $stutt(\sigma) \sim stutt(\rho)$Γla classe d'équivalence $\mathcal{C}_i$ a un correspondant $\mathcal{C}'_k$ dans $stutt(\rho)$Γet donc le k-ième préfixe de ρ satisfait $\sigma' S \rho^k$. Il reste à vérifier que dans la transition arrière $\rho^k \stackrel{q}{\Rightarrow} \rho$ le pomset q est égal à p. Pour celaΓil suffit d'utiliser que g^{ρ}_{σ} et $g^{\rho^k}_{\sigma'}$ sont des isomorphismes.

$\Leftarrow$) : Nous reprenons l'idée de la preuve du Théorème 2.3.3 de [DMV90].
Nous montrons que si $R : \mathcal{E} \approx^{\tau}_{bf} \mathcal{F}$Γalors $R : \mathcal{E} \approx^{b}_{mo} \mathcal{F}$.

Evidemment $\lambda_{\mathcal{E}} R \lambda_{\mathcal{F}}$.
Soient $\sigma R \rho$ et $\sigma \stackrel{e}{\rightarrow} \sigma'$. Nous distinguons deux cas :

1) $l(e) \neq \tau$. Puisque $\sigma R \rho$Γil existe $\rho \stackrel{f_1 \ldots f_n}{\Rightarrow} \rho_1 \stackrel{f}{\rightarrow} \rho_2 \stackrel{f'_1 \ldots f'_m}{\Rightarrow} \rho'$Γtels que $\sigma' R \rho'$Γ$l(f) = a$ et $l(f_i) = l(f'_j) = \tau$ pour tous $1 \leq i \leq n, 1 \leq j \leq m$.
   La transition arrière $\rho' \Rightarrow \rho_2$ est forcément imitée à partir de σ' en ne faisant rienΓdonc $\sigma' R \rho_2$. Il reste à montrer que $\sigma R \rho_1$. Pour celaΓon considère la transition en arrière $\rho_1 \stackrel{f}{\rightarrow} \rho_2$. Elle est imitée en partant de σ' par $\hat{\sigma} \stackrel{\emptyset}{\Rightarrow} \sigma \stackrel{e}{\rightarrow} \sigma'$ avec $\hat{\sigma} R \rho_1$. En appliquant le X-Lemme 9.1.1 à $\sigma R \rho$ et $\hat{\sigma} R \rho_1$Γnous obtenons $\sigma R \rho_1$.
2) $l(e) = \tau$: La transition est imitée en partant de ρ par un certain nombre de τ-transitions $(\rho =) \rho_0 \stackrel{\tau}{\rightarrow} \rho_1 \stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow} \rho_n = (\rho')$. Si $n = 0$Γnous avons prouvé la propriété de transfert. Si $n > 0$Γon peut effectuer la transition en arrière $\rho_{n-1} \stackrel{\tau}{\rightarrow} \rho_n$. Une première possibilité est que σ' simule cette transition en ne faisant rien. Dans ce casΓou bien $n = 1$ et c'est terminéΓou bien on peut effectuer la transition arrière $\rho_{n-2} \stackrel{\tau}{\rightarrow} \rho_{n-1}$. En itérant le raisonnementΓ soit $\sigma' R \rho_0$Γalors la propriété de transfert de l'équivalence de mixed-ordering branching est vérifiéeΓsoit il existe $m > 0$ avec $\sigma' R \rho_m$ pour lequel la transition arrière $\rho_{m-1} \stackrel{\tau}{\rightarrow} \rho_m$ est imitée par $\hat{\sigma} \stackrel{\emptyset}{\Rightarrow} \sigma \stackrel{\tau}{\rightarrow} \sigma'$ avec $\hat{\sigma} R \rho_{m-1}$. Le

(X-)Lemme 9.1.1 nous permet de conclure que $\sigma R \rho_{m-1}$Γce qu'illustre la Figure 9.4.

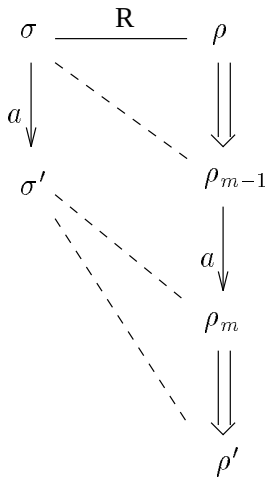


Figure 9.4:

□

9.2.2 La branching équivalence causale

[DD89] définissent les arbres causaux dans le cas où les SE considérées contiennent des actions invisibles. IntuitivementΠl'arbre causal est construit selon le même principe qu'en Section 6.3.3Γsauf que les références en arrière vers les causes d'un événement le long du chemin dans l'arbre ne portent que sur les événements visibles. Par conséquentΓpour les événements d'étiquette τΓaucune référence en arrière n'est indiquée et aucune référence en arrière ne pointe vers eux. FormellementΓ(la définition qui suit est extraite de [DD89])

Définition 9.2.2 *Si $\mathcal{E}$ est une SE sur l'alphabet d'actions Act_τ, l'arbre causal de $\mathcal{E}$, $CT(\mathcal{E})$, est défini par :*

- *l'ensemble des nœuds de $CT(\mathcal{E})$ est $Hist(\mathcal{E})$,*
- *les arcs sont les paires $(\sigma, \sigma.e)$,*
- *la fonction d'étiquetage des arcs vérifie*

$$\begin{array}{ll} l(\sigma,\sigma.e) = \tau & \text{si } l(e) = \tau \\ l(\sigma,\sigma.e) = (l(e), C(e,\sigma)) & \text{sinon} \end{array}$$

où $C(e,\sigma)$ est l'ensemble des références en arrière des causes visibles de e le long de σ.

Exemple 21 *L'arbre causal correspondant à la SE de droite de la Figure 9.1 a la forme de la Figure 9.5.*

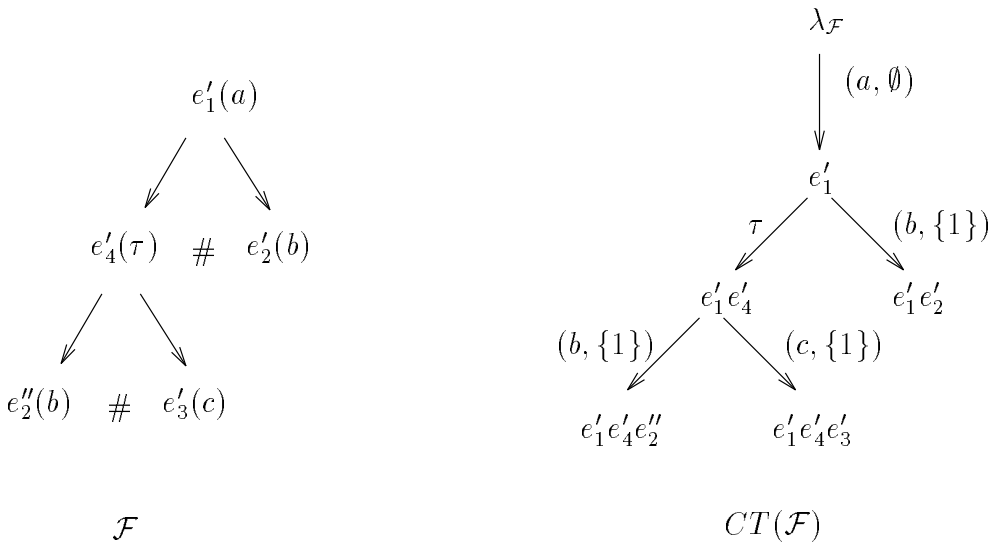


Figure 9.5: Un example d'arbre causal avec τ.

Les arbres causaux pour les SE avec actions invisibles sont donc des ST avec actions invisibles. On peut ainsi définir toutes les bisimulations modulo τ présentées au Chapitre 2 dans la Section 2.6.2.

Par exemple, la branching équivalence causale dérivée de la branching bisimulation sur les ST (voir la Section 2.6.2) est définie par :

Définition 9.2.3 (branching équivalence causale)
*Soient $\mathcal{E}, \mathcal{F} \in$ **SE**. On pose*

$$\mathcal{E} \approx_c^b \mathcal{F} \stackrel{def}{=} CT(\mathcal{E}) \underline{\leftrightarrow}_b CT(\mathcal{F})$$

Proposition 9.2.3 *Pour tous $\mathcal{E}, \mathcal{F} \in$ **SE**,*

$$\mathcal{E} \approx_c^b \mathcal{F} \text{ ssi } \mathcal{E} \approx_{mo}^b \mathcal{F}$$

Preuve Nous l'omettons, mais elle est basée principalement sur le fait qu'il existe une bijection entre l'ensemble des nœuds de $CT(\mathcal{E})$ et $Hist(\mathcal{E})$.

□

Suite aux Propositions 9.2.2 et 9.2.3 que nous venons d'établir, nous obtenons le résultat suivant :

Corollaire 9.2.4 *Pour tous $\mathcal{E}, \mathcal{F} \in$ **SE**,*

$$\mathcal{E} \approx_{bf}^{\tau} \mathcal{F} \text{ ssi } \mathcal{E} \approx_c^b \mathcal{F}$$

9.2.3 La history preserving branching bisimulation

En nous inspirant des résultats précédentsΓnous définissons la history preserving branching bisimulation. Cette équivalence coïncide trivialement avec la history preserving bisimulation dans le cas sans τ.

Définition 9.2.4 (history preserving branching bisimulation) *Soient $\mathcal{E},\mathcal{F} \in$ **SE**. Une relation symétrique*[2] *$R \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{F}) \times \mathcal{P}(vis_{\mathcal{E}} \times vis_{\mathcal{F}}) \bigcup \mathcal{C}(\mathcal{F}) \times \mathcal{C}(\mathcal{E}) \times \mathcal{P}(vis_{\mathcal{F}} \times vis_{\mathcal{E}})$ est une* history preserving branching bisimulation *entre $\mathcal{E}$ et $\mathcal{F}$, noté $R : \mathcal{E} \approx^b_{hp} \mathcal{F}$, si $(\emptyset,\emptyset,\emptyset) \in R$ et pour tout $(C,D,g) \in R$,*

- *$g : vis(C) \rightarrow vis(D)$ est un isomorphisme,*
- *si $C \stackrel{e}{\rightarrow} C'$, alors*
  - *ou bien $l(e) = \tau$ et $(C',D,g) \in R$*
  - *ou bien il existe D', D'', g' tels que $D \Rightarrow D'' \rightarrow D'$, $g'_{|vis(C)} = g$, $(C,D'',g) \in R$ et $(C',D',g') \in R$*

Nous notons $\mathcal{E} \approx^b_{hp} \mathcal{F}$ lorsque $R : \mathcal{E} \approx^b_{hp} \mathcal{F}$ pour une relation R, et nous disons que $\mathcal{E}$ et $\mathcal{F}$ sont history preserving branching bisimilaires.

On vérifie facilement que $\approx^b_{hp}$ est une relation d'équivalence sur **SE**.

Notons que dans la Définition 9.2.4Γnous avons forcément $D \stackrel{\emptyset}{\Rightarrow} D''$.

Proposition 9.2.5 *Pour tous $\mathcal{E},\mathcal{F} \in$ **SE**,*

$$\mathcal{E} \approx^b_{hp} \mathcal{F} \text{ ssi } \mathcal{E} \approx^b_{mo} \mathcal{F}$$

Preuve $\Rightarrow$) : Soit $S : \mathcal{E} \approx^b_{hp} \mathcal{F}$Γon définit $R \subseteq Hist(\mathcal{E}) \times Hist(\mathcal{F}) \bigcup Hist(\mathcal{F}) \times Hist(\mathcal{E})$ par :

$$R \stackrel{\text{def}}{=} \{(\sigma,\rho) | (C_\sigma, D_\rho, g^\rho_\sigma) \in S\}$$

On montre facilement que $R : \mathcal{E} \approx^b_{mo} \mathcal{F}$.

$\Leftarrow$) : Soit $R : \mathcal{E} \approx^b_{mo} \mathcal{F}$Γon définit $S \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{F}) \bigcup \mathcal{C}(\mathcal{F}) \times \mathcal{C}(\mathcal{E})$ par :

$$S \stackrel{\text{def}}{=} \{(C_\sigma, D_\rho, g^\rho_\sigma) | \sigma R \rho\}$$

On vérifie facilement que $S : \mathcal{E} \approx^b_{hp} \mathcal{F}$. □

En résumé de cette sectionΓnous avons établi le résultat suivant : pour toutes $\mathcal{E},\mathcal{F} \in$ **SE**Γ

$$\mathcal{E} \approx^b_{hp} \mathcal{F} \text{ ssi } \mathcal{E} \approx^b_{mo} \mathcal{F} \text{ ssi } \mathcal{E} \approx^\tau_{bf} \mathcal{F} \text{ ssi } \mathcal{E} \approx^b_c \mathcal{F} \qquad (9.13)$$

l'Equation (9.13) généralise l'Equation (6.9) du Chapitre 6Γ ce qui justifie de considérer l'équivalence de history preserving branching bisimulation. La section suivante est consacrée à son étude.

[2] $(C,D,f) \in R$ *implique* $(D,C,f^{-1}) \in R$

9.3 Les history preserving bisimulations

L'équivalence de history preserving bisimulation a déjà été étudiée dans les SE avec τΓvoir [Vog91] et [Ace91]. Suivant la façon de généraliser la définition pour abstraire les actions invisiblesΓon obtient différentes équivalences. Notre équivalence $\approx_{hp}^{b}$ en est un exemple.

[Ace91] considère la history preserving τ-bisimulation (qui est la généralisation la plus immédiate de $\approx_{hp}$)Γnotée $\sim_{hp}$ dans la suite. Il montre qu'elle coïncide avec la τ-bisimulation sur les arbres causauxΓainsi qu'avec l'équivalence de mixed-ordering dans sa version faibleΓc-à-d. comme τ-bisimulation.

Définition 9.3.1 (history preserving τ-bisimulation)
Soient $\mathcal{E},\mathcal{F} \in \mathbf{SE}$. Une relation symétrique[3] *$R \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{F}) \times \mathcal{P}(vis_{\mathcal{E}} \times vis_{\mathcal{F}}) \bigcup \mathcal{C}(\mathcal{F}) \times \mathcal{C}(\mathcal{E}) \times \mathcal{P}(vis_{\mathcal{F}} \times vis_{\mathcal{E}})$ est une* history preserving τ-bisimulation *entre $\mathcal{E}$ et $\mathcal{F}$, noté $R : \mathcal{E} \sim_{hp} \mathcal{F}$, si $(\emptyset,\emptyset,\emptyset) \in R$ et si pour tout $(C,D,g) \in R$,*

- *$g : vis(C) \rightarrow vis(D)$ est un isomorphisme, et*
- *si $C \stackrel{e}{\Rightarrow} C'$, il existe D', g' tels que $D \Rightarrow D'$, $g'_{|vis(C)} = g$ et $(C',D',g') \in R$.*

Exemple 22 *Les SE de la Figure 9.6 sont $\sim_{hp}$ équivalentes. Observons cependant qu'elles ne sont plus équivalentes si on raffine l'action a en $a_1;a_2$. En effet, les SE obtenues ne sont alors même plus équivalentes pour la τ-bisimulation : $\mathcal{F}$ peut effectuer $e'_4(a_1)$ imité dans $\mathcal{E}$ par $e_1(a_1)$. Mais les configurations atteintes ne sont pas bisimilaires puisque à partir de $\{e_1\}$ on a toujours la possibilité dans le futur d'effectuer un b ce qui n'est pas le cas en partant de $\{e'_4\}$.*

L'exemple de la Figure 9.6 est emprunté à [Vog90] quiΓpour pallier les lacunes de l'équivalence $\sim_{hp}$ vis-à-vis du raffinementΓpropose sa version "ST"Γappelée *history preserving ST bisimulation*. Nous la notons $\approx_{hp}^{ST}$ dans la suite.

Les ST-bisimulations ont été introduites par [GV87]. Contrairement aux équivalences considérées jusqu'iciΓelles ne sont pas basées sur l'*atomicité* des actions. Nous renvoyons à [Gla90c] pour leur définitionsΓoù il est de plus montré qu'elles sont préservées par le raffinement d'actions.

La Définition de $\approx_{hp}^{ST}$ est basée sur la notion de ST-configurations. Nous ne jugeons pas nécessaire de rappeler la définition formelle de ST-configurations puisque nous disposons dans [Dev90] d'une définition équivalente de $\approx_{hp}^{ST}$Γappelée *maximality preserving bisimulation*Γqui est directement définie sur les configurations.

Définition 9.3.2 [Dev90] (hST bisimulation) *Soient $\mathcal{E},\mathcal{F} \in \mathbf{SE}$. Une relation symétrique $R \subseteq \mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{F}) \times \mathcal{P}(vis_{\mathcal{E}} \times vis_{\mathcal{F}}) \bigcup \mathcal{C}(\mathcal{F}) \times \mathcal{C}(\mathcal{E}) \times \mathcal{P}(vis_{\mathcal{F}} \times vis_{\mathcal{E}})$ est une* history preserving ST bisimulation *entre $\mathcal{E}$ et $\mathcal{F}$, noté $R : \mathcal{E} \approx_{hp}^{ST} \mathcal{F}$, si $(\emptyset,\emptyset,\emptyset) \in R$ et si pour tout $(C,D,g) \in R$,*

[3] *$(C,D,g) \in R$ implique $(D,C,g^{-1}) \in R$*

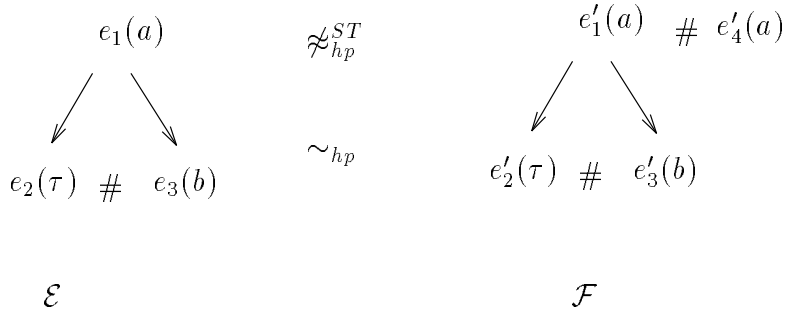


Figure 9.6: $\mathcal{E} = a;(\tau + b)$ et $\mathcal{F} = \mathcal{E} + a$

- g : $vis(C) \to vis(D)$ *est un isomorphisme, et*
- *si* $C \Rightarrow C'$, *il existe* D', g' *tels que*
  - $D \Rightarrow D'$, $(C', D', g') \in R$ *et* $g'_{|vis(C)} = g$,
  - *si* $e \in vis(C')$ *est un élément maximal de* C', *alors* $g'(e)$ *est maximal dans* D' *ou* $e \in C$ *et* $g(e)$ *n'est pas maximal dans* D.

[Vog91] remarque que dans la Définition 9.3.2 on peut restreindre les transitions $C \Rightarrow C'$ au cas où $C' \setminus C$ contient au plus un événement visible.

Nous rappelons la propriété principale de la history preserving ST bisimulation :

Proposition 9.3.1 [Vog90]
L'équivalence $\approx^{ST}_{hp}$ *est la plus grande congruence pour le raffinement d'actions qui soit contenue dans* $\sim_{hp}$.

En particulierΓl'équivalence $\approx^{ST}_{hp}$ coïncide avec la history preserving bisimulation $\approx_{hp}$ dans le cas sans τ. L'Exemple 22 suffit à prouver qu'en général $\approx^{ST}_{hp}$ est strictement plus fine que $\sim_{hp}$: les SE de la Figure 9.6 ne sont pas $\approx^{ST}_{hp}$-équivalente. En effetΓla transition $\emptyset \to_{\mathcal{F}} \{e'_4\}$ est forcément imitée par $\emptyset \to_{\mathcal{E}} \{e_1, e_2\}$ avec l'isomorphisme qui associe e_1 à e'_4 ; mais alors e'_4 est un élément visible maximalΓce qui n'est pas le cas de e_1 puisque $e_1 \leq e_2$.

Les rappels sur les history preserving bisimulations avec τ étant terminésΓnous nous proposons dans la section qui suit d'étudier la compatibilité de notre équivalence $\approx^{b}_{hp}$ vis-à-vis du raffinement d'actions.

9.4 Raffinement d'actions

Dans cette section nous établissons que $\approx^{b}_{hp}$ est strictement plus fine que $\sim_{hp}$ et qu'elle est incomparable avec $\approx^{ST}_{hp}$Γce qui nous permettra d'énoncer queΓen généralΓ

l'équivalence $\approx^b_{hp}$ n'est pas préservée par le raffinement d'actions.

Par définition$\Gamma \approx^b_{hp}$ implique $\sim_{hp}$; on peut aussi retrouver ce résultat en considérant les équivalences causales associées qui sont respectivement la branching bisimulation (voir section précédente) et la τ-bisimulation [Ace91].

Remarquons au passage que le fait de pouvoir caractériser des équivalences d'ordre partiel dans les arbres causauxΓ qui sont des modèles d'entrelacementΓ donne un excellent moyen pour reconnaître ces équivalences dans la sous-famille des systèmes séquentiels. Par exempleΓpour de tels systèmes$\Gamma \approx^b_{hp}$ et $\sim_{hp}$ coïncident respectivement avec la branching bisimulation (voir section précédente) et la τ-bisimulation [Ace91]. Il est alors facile d'exhiber des contre-exemples qui prouvent que l'implication de $\approx^b_{hp}$ vers $\sim_{hp}$ est stricteΓtout simplement en les empruntant à la littérature qui traite les équivalences des modèles d'entrelacement. Dans notre casΓil suffit de trouver deux modèles τ-bisimilaires mais pas branching bisimilaires; c'est par exemple le cas des deux ST de la Figure 2.5 du Chapitre 2.

De même pour la history preserving ST bisimulationΓ[Vog91] remarque qu'elle coïncide avec la delay bisimulation pour le cas des systèmes séquentiels. Nous pouvons alors déduire que $\approx^{ST}_{hp}$ n'implique pas $\approx^b_{hp}$. Par exempleΓles SE de la Figure 9.7 ne sont pas $\approx^b_{hp}$-équivalentes. En revancheΓelles sont $\approx^{ST}_{hp}$-équivalentes puisque tous les événements visibles sont maximaux dans toutes les histoires.

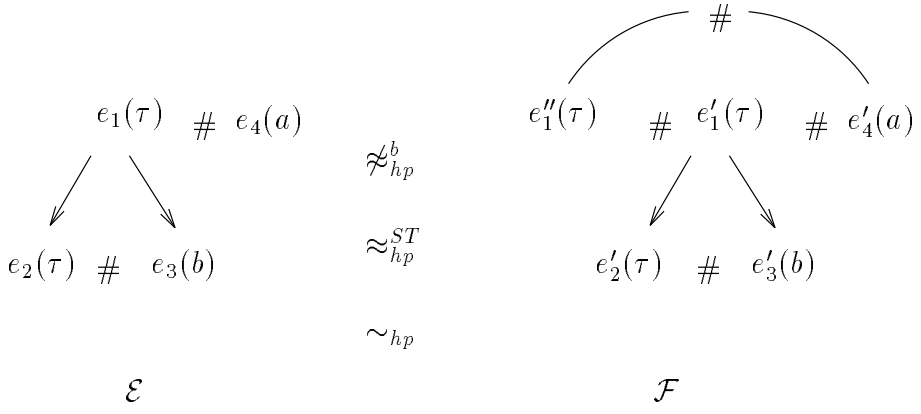


Figure 9.7: $\mathcal{E} = (\tau;(\tau + b)) + a$ et $\mathcal{F} = \mathcal{E} + \tau$

Curieusement nous n'avons pas non plus $\approx^b_{hp}$ implique $\approx^{ST}_{hp}$. Nous devons ce contre-exemple à Robert van Glabbeek [Gla93].

Considérons les deux SE de la Figure 9.8. Elles ne sont pas $\approx^{ST}_{hp}$-équivalentes. En effetΓsi $\mathcal{F}$ effectue l'événement "c" le plus à droiteΓalors $\mathcal{E}$ l'imite en effectuant son unique événement étiqueté par c. Si maintenant $\mathcal{E}$ poursuit en effectuant son

événement a le plus à droite, il faut dans $\mathcal{F}$ exclure toute possibilité d'exécuter une action b dans le futur : pour cela il est nécessaire d'exécuter une action τ, conséquence immédiate de c dans $\mathcal{F}$. Mais alors, la maximalité de c dans $\mathcal{E}$ n'est pas préservée dans $\mathcal{F}$. Les deux SE ne sont donc pas $\approx_{hp}^{ST}$-équivalentes.

La preuve que ces deux SE de la Figure 9.8 sont $\approx_{hp}^{b}$-équivalentes s'appuie sur les observations suivantes : premièrement, les configurations de $\mathcal{E}$ et $\mathcal{F}$ ne peuvent contenir deux événements ayant la même étiquette. Deuxièmement, les dépendances causales entre les événements sont complètement déterminées par les étiquettes de ces événements - tout événement étiqueté par a ou par c n'a aucune cause et tout événement étiqueté par b a une unique cause étiquetée par a.

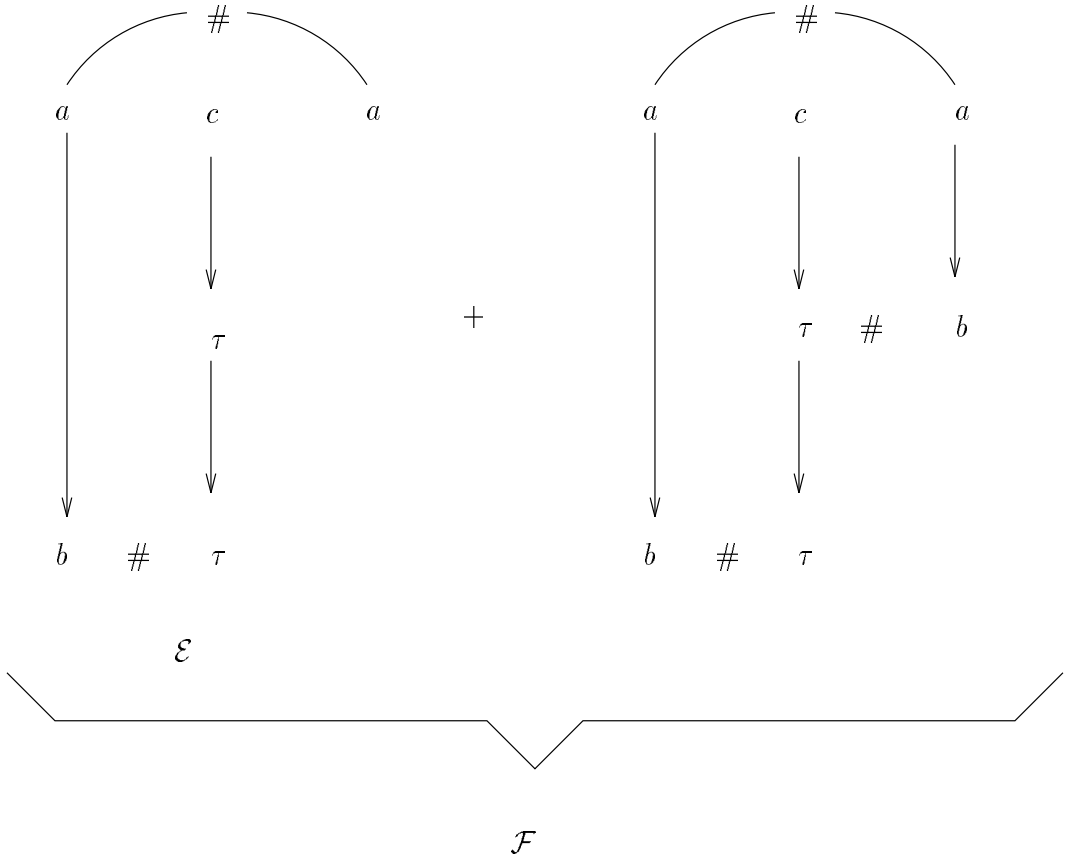


Figure 9.8: $\approx_{hp}^{b}$ n'implique pas $\approx_{hp}^{ST}$

Lemme 9.4.1 *Les SE de la Figure 9.8 sont history preserving branching bisimilaires si et seulement si elles sont interleaving branching bisimilaires.*

Preuve Le sens $\Rightarrow$ est évident.

Pour l'autre sens il suffit de remarquer que si R est une branching bisimulation entre $\mathcal{E}$ et $\mathcal{F}$ c-à-d. une relation binaire entre leurs configurations alors R restreinte aux paires de configurations ayant les mêmes ensembles d'événements visibles est encore une branching bisimulation qui peut de plus être relevée en une relation de history preserving branching bisimulation entre $\mathcal{E}$ et $\mathcal{F}$.

□

Le Lemme ci-dessus montre qu'il suffit de prouver que $\mathcal{E}$ et $\mathcal{F}$ sont (interleaving) branching bisimilaires. Nous omettons cette démonstration qui n'est pas très difficile.

En résumé nous venons d'établir que les équivalences $\approx_{hp}^{ST}$ et $\approx_{hp}^{b}$ sont incomparables. De là et à partir de la Proposition 9.3.1 nous pouvons énoncer

Corollaire 9.4.2 *L'équivalence $\approx_{hp}^{b}$ n'est pas compatible avec le raffinement d'actions.*

On peut aussi donner une preuve directe du corollaire : considérons les SE $\mathcal{E}$ et $\mathcal{F}$ de la Figure 9.8 et soit ref la fonction de raffinement qui remplace l'action c par $c_1;c_2$.

Alors les SE $ref(\mathcal{E})$ et $ref(\mathcal{F})$ ne sont mêmes plus interleaving τ-bisimilaires. Pour l'expliquer nous donnons la formule de HML $\Phi \stackrel{\text{def}}{=} [c_1]\langle a\rangle[b]\neg\top$.

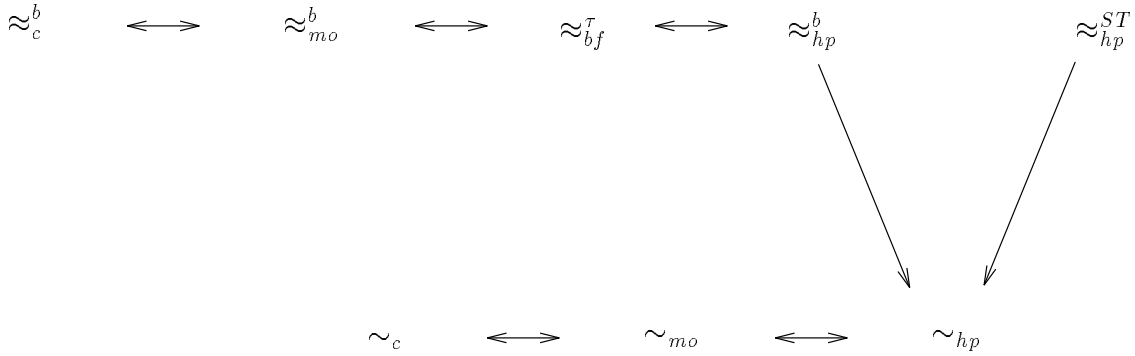
Nous avons $ref(\mathcal{E}) \models \Phi$ mais $ref(\mathcal{F}) \not\models \Phi$.

Remarquons qu'avant de leur appliquer le raffinement les deux SE satisfont la formule $[c]\langle a\rangle[b]\neg\top$. Dans $\mathcal{F}$ il suffit d'effectuer "rapidement" un τ juste avant le a.

De cet exemple nous pouvons aussi déduire

Corollaire 9.4.3 *Aucune équivalence contenue entre la history preserving branching bisimulation et la (interleaving) τ-bisimulation n'est préservée par le raffinement d'actions.*

La Figure 9.4 résume les résultats obtenus dans ce chapitre.


$\approx_c^b$
$\approx_{mo}^b$
$\approx_{bf}^\tau$
$\approx_{hp}^b$
$\approx_{hp}^{ST}$
$\sim_c$
$\sim_{mo}$
$\sim_{hp}$

Conclusion

Dans cette deuxième partie de la thèse, nous avons contribué à éclaircir les liens entre les différentes équivalences sémantiques pour les structures d'événements premières.

Nous avons proposé au Chapitre 6 une présentation unifiée des équivalences $\approx_{hp}$ de [GG89a], $\approx_{mo}$ de [DDM89], $\approx_{c}$ de [DD89] et $\approx_{pbf}$ de [Che92a].

En utilisant l'équivalence $\approx_{pbf}$, nous avons dérivé une caractérisation de la history preserving bisimulation par la logique L_{pbf}. Ce résultat est inspiré du Théorème d'adéquation de Hennessy-Milner qui relie une bisimulation (avant-arrière) à une logique (avec des modalités du passé).

Nous avons utilisé la logique L_{pbf} pour montrer que la logique L_P de [DF90] caractérise elle aussi la history preserving bisimulation, contrairement à ce qu'affirme [DF90]. Notre méthode est basée sur une technique "à la Gabbay" de séparation des formules de la logique L_P en une combinaison booléenne de formules du futur et du passé. A notre connaissance, cette technique n'a été utilisée jusqu'ici que pour des logiques temporelles. Nous pensons que cette méthode s'applique à d'autres cas. Par exemple, elle peut servir à montrer que les formules d'une logique modale avec des opérateurs du passé sont initialement équivalentes à des formules sans opérateur du passé.

Le Chapitre 8 établit une classification des bisimulations avant-arrière pour les structures d'événements premières sans action invisible, qui complète celle de [Che92c].

Enfin, en considérant les actions invisibles dans les structures d'événements, nous avons étudié la pomset τ-bisimulation avant-arrière, qui généralise la pomset bisimulation avant-arrière. Nous avons montré que la pomset τ-bisimulation avant-

arrière coïncide avec la branching bisimulation dans les arbres causaux, et nous avons proposé deux nouvelles caractérisations : l'équivalence de mixed-ordering branching et la history preserving branching bisimulation. Ces deux équivalences n'ont pas, à notre connaissance été proposées dans la littérature.

A la soutenance de la présente thèse, nous n'étions pas en mesure de dire si l'équivalence $\approx_{hp}^{b}$ était ou non préservée par le raffinement d'actions.

Depuis, et grâce à un travail en collaboration avec Robert van Glabbeek, nous avons pu exhiber un exemple de deux SE équivalentes mais qui ne le sont plus lorsqu'on les raffine.

Remarquons que ce résultat négatif a un caractère surprenant si l'on sait que, et la history preserving bisimulation (dans les SE sans τ) et la branching bisimulation (dans les systèmes de transitions) sont compatibles avec le raffinement d'actions.

Les résultats de cette partie offrent plusieurs perspectives de travail.

Tout d'abord, nous pensons que nos résultats peuvent être généralisés aux structures d'événements *stables* de [Win89], tout comme [Ace91] l'a fait pour la history preserving τ-bisimulation. Nous travaillons actuellement dans des modèles plus généraux, les *Asynchronous Transition Systems* [Bed87], dans lesquels nous étudions une généralisation de la pomset bisimulation avant-arrière et de l'équivalence de mixed-ordering.

D'autre part, nous n'avons pas étudié de caractérisations logiques de la pomset τ-bisimulation avant-arrière, mais il paraît clair qu'une généralisation avec τ de la logique L_{pbf} est adéquate.

L'équivalence $\approx_{hp}^{b}$ n'étant pas compatible avec le raffinement, il semble naturel d'en définir sa version "ST". L'équivalence alors obtenue, que nous appelons la *history preserving branching ST bisimulation* et que nous notons $\approx_{hp}^{bST}$, serait évidemment préservée par le raffinement et en s'inspirant des résultats de [Vog91] nous obtiendrons la plus grande congruence qui contienne $\approx_{hp}^{b}$ (voir la Figure 9.9).

D'autres approches pour définir des bisimulations avant-arrière ont été considérées dans la littérature :
[GKP92] proposent une bisimulation avant-arrière dans le graphe des configurations, où les transitions en arrière se font aussi le long de l'histoire du calcul, mais pour une autre notion d'histoire. Cette équivalence est définie dans les structures d'événements sans auto-concurrence (l'absence d'auto-concurrence garantit le déterminisme pour les transitions en arrière dans le graphe des configurations).

[GKP92] ont montré que dans ce cadre, l'équivalence qu'ils définissent est une congruence pour le raffinement et qu'elle est plus fine que la history preserving bisimulation. Il reste à étudier, dans le cadre des structures d'événements avec

auto-concurrenceΓla différence entre l'approche de [GKP92] et celle de [Che92a].

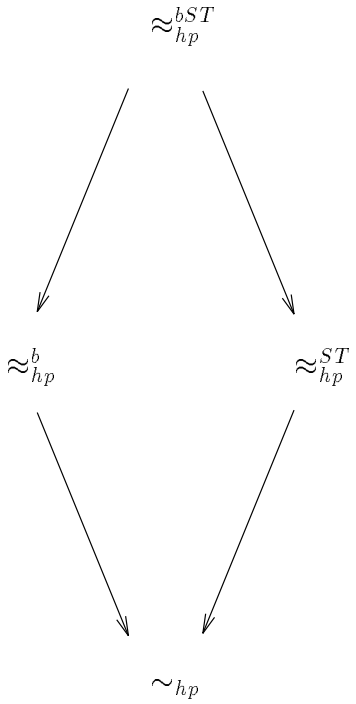


Figure 9.9:

Bibliographie

[Ace91] L. Aceto. History preserving, causal and mixed-ordering equivalence for stable event structures. Tech. Report HPL-PSC-91-28, Hewlett-Packard Laboratories, Pisa Science Center, Pisa, 1991. To appear in *Fundamenta Informaticae*.

[AD89] A. Arnold and A. Dicky. An algebraic characterization of transition systems equivalences. *Information and Computation*, 82(2):198–229, August 1989.

[Apt85] K. R. Apt. Proving correctness of CSP programs. A tutorial. In M. Broy, editor, *Control Flow and Data Flow: Concepts of Distributed Programming*, pages 441–474. Springer-Verlag, 1985.

[AR88] I. J. Aalbersberg and G. Rozenberg. Theory of traces. *Theoretical Computer Science*, 60:1–82, 1988.

[BBK87] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51(1):129–176, 1987. Available as CWI Report CS-R8511.

[BC87] G. Boudol and I. Castellani. On the semantics of concurrency: Partial orders and transition systems. In *Proc. CAAP'87, Pisa, LNCS 249*, pages 123–137. Springer-Verlag, March 1987.

[BC88] G. Boudol and I. Castellani. Concurrency and atomicity. *Theoretical Computer Science*, 59(1):25–84, 1988.

[BCG87] M. C. Browne, E. M. Clarke, and O. Grümberg. Characterizing Kripke structures in temporal logic. In *Proc. CAAP'87, Pisa, LNCS 249*, pages 256–270. Springer-Verlag, March 1987.

[BDKP91] E. Best, R. Devillers, A. Kiehn, and L. Pomello. Concurrent bisimulations in Petri nets. *Acta Informatica*, 28:231–264, 1991.

[Bed87] M. A. Bednarczyk. *Categories of Asynchronous Systems*. PhD thesis, Univ. Sussex, October 1987. Available as CS R 1/88.

[BG87] J. C. M. Baeten and R. J. van Glabbeek. Another look at abstraction in process algebra (extended abstract). In *Proc. 14th ICALP, Karlsruhe, LNCS 267*, pages 84–94. Springer-Verlag, July 1987.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, July 1984.

[BIM88] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced: preliminary report. In *Proc. 15th ACM Symp. Principles of Programming Languages, San Diego, CA*, pages 229–239, January 1988.

[BK85] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[BR83] S. D. Brookes and W. C. Rounds. Behavioural equivalence relations induced by programming logics. In *Proc. 10th ICALP, Barcelona, LNCS 154*, pages 97–108. Springer-Verlag, July 1983. Available as CMU Report CMU-CS-83-112.

[BR92] J.W. de Bakker and J. J. M. M. Rutten, editors. *Ten years of concurrency semantics : selected papers of Amsterdam Concurrency Group*. World Scientific, 1992.

[BT85] S. L. Bloom and D. R. Troeger. A logical characterization of observation equivalence. *Theoretical Computer Science*, 35:43–53, 1985.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Logics of Programs Workshop, Yorktown Heights, LNCS 131*, pages 52–71. Springer-Verlag, May 1981.

[CES83] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications: A practical approach. In *Proc. 10th ACM Symp. Principles of Programming Languages, Austin, Texas*, pages 117–126, January 1983.

[Che92a] F. Cherief. Back and forth bisimulations on prime event structures. In *Proc. PARLE'92, Paris, LNCS 605*, pages 843–858. Springer-Verlag, June 1992.

[Che92b] F. Cherief. *Contributions à la Sémantique du Parallélisme: Bisimulations pour le Raffinement et le Vrai Parallélisme*. Thèse de Doctorat, I.N.P. de Grenoble, France, October 1992.

[Che92c] F. Cherief. Investigations of back and forth bisimulations on prime event structures. *Computers and Artificial Intelligence*, 11(5):481–496, 1992.

[Cle90] R. Cleaveland. On automatically explaining bisimulation inequivalence. In *Proc. CAV'90, New Brunswick, NJ, LNCS 531*, pages 364–372. Springer-Verlag, June 1990.

[CLP92] F. Cherief, F. Laroussinie, and S. Pinchinat. Modal logics with past for true concurrency. Internal Report, May 1992.

[CMP87] L. Castellano, G. De Michelis, and L. Pomello. Concurrency vs interleaving: an instructive example. *EATCS Bull.*, 31:12–15, February 1987.

[Dam90] M. Dam. Translating CTL* into the modal μ-calculus. Research Report ECS-LFCS-90-123, Lab. for Foundations of Computer Science, Edinburgh, November 1990.

[DD89] Ph. Darondeau and P. Degano. Causal trees. In *Proc. 16th ICALP, Stresa, LNCS 372*, pages 234–248. Springer-Verlag, July 1989.

[DD92] Ph. Darondeau and P. Degano. Refinement of actions in event structures and causal trees. To appear in TCS, 1992.

[DDM87] P. Degano, R. De Nicola, and U. Montanari. Observational equivalences for concurrency models. In M. Wirsing, editor, *Formal Description of Programming Concepts - III, Proc. of the third IFIP WG 2.1 working conf., Ebberup 1986*, pages 105–129. North-Holland, 1987.

[DDM89] P. Degano, R. De Nicola, and U. Montanari. Partial orderings descriptions and observations of nondeterministic concurrent processes. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Noordwijkerhout, LNCS 354*, pages 438–466. Springer-Verlag, 1989.

[Dev90] R. Devillers. Maximality preserving bisimulation. Tech. Report LIT-214, Lab. Informatique Théorique, Université Libre de Bruxelles, March 1990.

[DF90] R. De Nicola and G. L. Ferrari. Observational logics and concurrency models. In *Proc. 10th Conf. Found. of Software Technology and Theor. Comp. Sci., Bangalore, India, LNCS 472*, pages 301–315. Springer-Verlag, December 1990.

[DG92] P. Degano and R. Gorrieri. An operational definition of action refinement. Tech. Report 28/92, Univ. Pisa, Dept. of Computer Science, 1992.

[DMV90] R. De Nicola, U. Montanari, and F. Vaandrager. Back and forth bisimulations. In *Proc. CONCUR'90, Amsterdam, LNCS 458*, pages 152–165. Springer-Verlag, August 1990.

[DV90] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation (extended abstract). In *Proc. 5th IEEE Symp. Logic in Computer Science, Philadelphia, PA*, pages 118–129, June 1990.

[EC80] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th ICALP, Noordwijkerhout, LNCS 85*, pages 169–181. Springer-Verlag, July 1980.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B*, chapter 16, pages 995–1072. Elsevier Science Publishers, 1990.

[FL79] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[Gab87] D. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Proc. Temporal Logic in Specification, Altrincham, UK, LNCS 398*, pages 409–448. Springer-Verlag, April 1987.

[GG89a] R. J. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions. In *Proc. Math. Found. Comp. Sci., Porabka-Kozubnik, LNCS 379*, pages 237–248. Springer-Verlag, August 1989.

[GG89b] R. J. van Glabbeek and U. Goltz. Refinement of actions in causality based models. In *Stepwise Refinement of Distributed Systems. Models, Formalisms, Correctness, Mook, LNCS 430*, pages 267–300. Springer-Verlag, May 1989.

[GKP92] U. Goltz, R. Kuiper, and W. Penczek. Propositional temporal logics and equivalences. In *Proc. CONCUR'92, Stony Brook, NY, LNCS 630*, pages 222–236. Springer-Verlag, August 1992.

[Gla90a] R. J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions.* PhD thesis, Free Univ. of Amsterdam, May 1990.

[Gla90b] R. J. van Glabbeek. The linear time - branching time spectrum. Research Report CS-R9029, CWI, July 1990.

[Gla90c] R. J. van Glabbeek. The refinement theorem for ST-bisimulation semantics. Research Report CS-R9002, CWI, January 1990.

[Gla93] R. J. van Glabbeek, November 1993. Private communication.

[Gra81] J. Grabowski. On partial languages. *Fundamenta Informaticae*, IV(2):427–498, 1981.

[Gro89] J. F. Groote. Transition system specifications with negative premisses. Research Report CS-R8950, CWI, December 1989.

[Gro90] J. F. Groote. Transition system specifications with negative premisses. In *Proc. CONCUR'90, Amsterdam, LNCS 458*, pages 332–341. Springer-Verlag, August 1990.

[GV87] R. J. van Glabbeek and F. Vaandrager. Petri net models for algebraic theories of concurrency. In *Proc. PARLE'87, vol. II: Parallel Languages, Eindhoven, LNCS 259*, pages 224–242. Springer-Verlag, June 1987.

[GV88] J. F. Groote and F. W. Vaandrager. Structured operational semantics and bisimulation as a congruence. Research Report CS-R8845, CWI, November 1988.

[GV89] R. J. van Glabbeek and F. Vaandrager. Modular specifications in process algebra. In *Proc. 9th IFIP WG 6.1 Int. Symp. Protocol Specification, Testing and Verification, Enschede, NL*, June 1989.

[GV92] J. F. Groote and F. W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, October 1992.

[GW89a] R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in process algebra. In *Information Processing 89, Proc. IFIP 11th World Computer Congress, San Francisco*, pages 613–618. North-Holland, August 1989.

[GW89b] R. J. van Glabbeek and W. P. Weijland. Refinement in branching time semantics. In *Proc. AMAST Conf., Iowa City*, pages 197–201, 1989. Available as CWI Report CS-R8922.

[Her87] B. Herwig. Equivalence in PDL and $L\mu$. In *Proc. 5th Easter Conf. Model Theory, Berlin*, pages 46–56, April 1987.

[Hil87] M. Hillerström. Verification of CCS processes. M.Sc. Thesis, Aalborg University, 1987.

[HM80] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *Proc. 7th ICALP, Noordwijkerhout, LNCS 85*, pages 299–309. Springer-Verlag, July 1980.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, January 1985.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall Int., 1985.

[HS85] M. Hennessy and C. Stirling. The power of the future perfect in program logics. *Information and Control*, 67:23–52, 1985.

[Kei77] H. J. Keisler. Fundamentals of model theory. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 47–103. North-Holland, 1977.

[Kel76] R. M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, July 1976.

[Klo88] J. W. Klop. *Bisimulation Semantics.* Lectures given at the REX School/Workshop, Noordwijkerhout, NL, May 1988.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[KP86] Y. Kornatzky and S. S. Pinter. A model checker for partial order temporal logic. EE PUB 597, Department of Electrical Engineering, Technion-Israel Institute of Technology, 1986.

[KP87] S. Katz and D. Peled. Interleaving set temporal logic. In *Proc. 6th ACM Symp. Principles of Distributed Computing, Vancouver, B.C., Canada*, pages 178–190, August 1987.

[Lam80] L. Lamport. "Sometimes" is sometimes "Not Never". In *Proc. 7th ACM Symp. Principles of Programming Languages, Las Vegas, Nevada*, pages 174–185, January 1980.

[Lam83] L. Lamport. What good is temporal logic ? In *Proc. Information Processing (IFIP) 83*, pages 657–668. North-Holland, September 1983.

[Lan92] R. Langerak. *Transformations and Semantics for LOTOS.* PhD thesis, Department of Computer Science, University of Twente, November 1992.

[LT87] K. Lodaya and P. S. Thiagarajan. A modal logic for a subclass of event structures. In *Proc. 14th ICALP, Karlsruhe, LNCS 267*, pages 290–303. Springer-Verlag, July 1987.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, 1980.

[Mil81] R. Milner. A modal characterisation of observable machine-behaviour. In *Proc. CAAP'81, Genoa, LNCS 112*, pages 25–34. Springer-Verlag, March 1981.

[Mil88] R. Milner. Operational and algebraic semantics of concurrent processes. Research Report ECS-LFCS-88-46, Lab. for Foundations of Computer Science, Edinburgh, February 1988.

[Mil89] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *Information and Computation*, 81(2):227–247, 1989.

[Mos90] P. D. Mosses. Denotational semantics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B*, chapter 11, pages 575–631. Elsevier Science Publishers, 1990.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurent Systems*, volume I: Specification. Springer-Verlag, 1992.

[MT89] M. Munkund and P. S. Thiagarajan. An axiomatization of event structures, 1989. Submitted to FSTTCS in Bangalore, India.

[NPW81] M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains, Part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

[OH86] E.-R. Olderog and C. A. R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conf. on Th. Comp. Sci., LNCS 104*, pages 167–183. Springer-Verlag, March 1981.

[Pen88] W. Penczek. A temporal logic for event structures. *Fundamenta Informaticae*, 11(3):297–326, 1988.

[Pen89] W. Penczek. A concurrent branching time temporal logic. In *Proc. 3rd Workshop on Computer Science Logic, Kaiserslautern, LNCS 440*, pages 337–354, October 1989.

[Pen91] W. Penczek. Branching time and partial order in temporal logics. Tech. Report UMCS-91-??, Dept. of Computer Science, Univ. of Manchester, 1991.

[Pin91] S. Pinchinat. Ordinal processes in comparative concurrency semantics. In *Proc. 5th Workshop on Computer Science Logic, Bern, LNCS 626*, pages 293–305. Springer-Verlag, October 1991.

[Pin92] S. Pinchinat. Ordinal processes in comparative concurrency semantics. Tech. Report 81, LIFIA-IMAG, Grenoble, April 1992.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Lect. Notes, Aarhus University, Aarhus, DK, 1981.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. Foundations of Computer Science, Providence*, pages 46–57, November 1977.

[Pnu85] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *Proc. 12th ICALP, Nafplion, LNCS 194*, pages 15–32. Springer-Verlag, July 1985.

[Pra76] V. R. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proc. 17th IEEE Symp. Foundations of Computer Science, Houston*, pages 109–121, 1976.

[Pra81] V. R. Pratt. A decidable mu-calculus: Preliminary report. In *Proc. 22nd IEEE Symp. Foundations of Computer Science, Nashville*, pages 421–427, 1981.

[Pra86] V. R. Pratt. Modeling concurrency with partial orders. *Int. J. Parallel Programming*, 15(1):33–71, 1986.

[PW84] S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations (extended abstract). In *Proc. 3rd ACM Symp. Principles of Distributed Computing, Vancouver, B.C., Canada*, pages 28–37, 1984.

[QS83] J. P. Queille and J. Sifakis. Fairness and related properties in transition systems. A temporal logic to deal with fairness. *Acta Informatica*, 19:195–220, 1983.

[Rei85] W. Reisig. *Petri Nets. An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[Ros82] J. G. Rosenstein. *Linear Orderings*. Academic Press, 1982.

[RT88] A. Rabinovich and B. A. Trakhtenbrot. Behavior structures and nets. *Fundamenta Informaticae*, 11(4):357–404, 1988.

[San82] M. T. Sanderson. *Proof Techniques for CCS*. PhD thesis, Univ. Edinburgh, November 1982. Available as Report CST-19-82.

[SB69] D. Scott and J. W. de Bakker. A theory of programs. IBM Seminar, Vienna, August 1969. Reprinted in "J. W. de Bakker, 25 Jaar Semantiek", CWI, Amsterdam, April 1989.

[Sch90a] Ph. Schnoebelen. Congruence properties of the process equivalence induced by temporal logic. Research Report 831-I, LIFIA-IMAG, Grenoble, October 1990.

[Sch90b] Ph. Schnoebelen. *Sémantique du parallélisme et logique temporelle. Application au langage FP2*. Thèse de Doctorat, I.N.P. de Grenoble, France, June 1990.

[Sif83] J. Sifakis. Property preserving homomorphisms of transition systems. In *Proc. Logics of Programs Workshop, Pittsburgh, LNCS 164*, pages 458–473. Springer-Verlag, June 1983.

[Sim85] R. De Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[Sin90] A. Sinachopoulos. Partial order logics for elementary net systems: State- and event-approaches. In *Proc. CONCUR'90, Amsterdam, LNCS 458*, pages 442–455. Springer-Verlag, August 1990.

[SP90] Ph. Schnoebelen and S. Pinchinat. On the weak adequacy of branching-time temporal logic. In *Proc. ESOP'90, Copenhagen, LNCS 432*, pages 377–388. Springer-Verlag, May 1990.

[Str81] R. Streett. Propositional dynamic logic of looping and converse. In *Proc. 13th ACM Symp. Theory of Computing, Milwaukee*, pages 375–383, 1981.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5:285–309, 1955.

[Vaa89a] F. Vaandrager. An explicit representation of equivalence classes of the history preserving bisimulation. Unpublished manuscript, CWI-Amsterdam, 1989.

[Vaa89b] F. W. Vaandrager. *Algebraic Techniques for Concurrency and their Applications.* PhD thesis, Univ. of Amsterdam, 1989.

[Vog90] W. Vogler. Bisimulation and action refinement. SFB-Bericht 342/10/90, Institut für Informatik, TUM, Munich, May 1990.

[Vog91] W. Vogler. Bisimulation and action refinement. In *Proc. STACS'91, Hamburg, LNCS 480*, pages 309–321. Springer-Verlag, February 1991.

[Wal88] D. J. Walker. Bisimulations and divergence. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, July 1988.

[Wei89] W. P. Weijland. *Synchrony and Asynchrony in Process Algebra.* PhD thesis, Univ. Amsterdam, June 1989.

[Win89] G. Winskel. An introduction to event structures. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Noordwijkerhout, LNCS 354*, pages 364–397. Springer-Verlag, 1989.

[WN92] G. Winskel and M. Nielsen. Models for concurrency. Research Report DAIMI PB-492, Comp. Sci. Dept., Aarhus Univ., November 1992. To appear in the Handbook of Logic in Computer Science, ed. S. Abramsky, D. M. Gabbay and T. S. E. Maibaum.