VÉRIFICATION FORMELLE DE SYSTÈMES D'INFORMATION

par

Raphaël Chane-Yack-Fa

Thèse présentée au Département d'informatique
en vue de l'obtention du grade de philosophiæ doctor (Ph.D.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 9 janvier 2018

Le 9 janvier 2018,

le jury a accepté la thèse de Monsieur Raphaël Chane-Yack-Fa dans sa version finale.

Membres du jury

Professeur Marc Frappier,
Directeur de recherche,
Département d'informatique, Faculté des sciences,
Université de Sherbrooke

Amel Mammar,
Codirectrice de recherche,
Télécom SudParis

Professeur Richard St-Denis,
Membre interne,
Département d'informatique, Faculté des sciences,
Université de Sherbrooke

Professeure Nadia Tawbi,
Membre externe,
Département d'informatique et de génie logiciel,
Université Laval

Professeur Pierre-Marc Jodoin,
Président-rapporteur,
Département d'informatique, Faculté des sciences,
Université de Sherbrooke

Résumé

Cette thèse s'intéresse à l'étude des méthodes formelles de spécification et de vérification dans le cadre des systèmes d'information. Les systèmes d'informations sont des systèmes dynamiques constitués d'entités et d'associations représentées par la composition en parallèle de processus répliqués issus de différentes classes. De plus, ces systèmes font partie de la classe des systèmes paramétrés. On propose un modèle de spécification de systèmes paramétrés, nommé PASTD, qui est adapté aux systèmes d'information et qui est basé sur la notation des diagrammes états-transitions algébriques (ASTD). Puis, on étudie le problème de sûreté pour les PASTD, à travers la méthode de vérification de couverture pour les systèmes de transitions bien structurés (WSTS). Cette méthode repose sur trois conditions principales : la monotonie, le beau préordre et la pred-base effective. Les PASTD sont montrés comme étant monotones et on définit une sous-classe vérifiant la propriété de beau préordre. Enfin, on décrit une nouvelle méthode, adaptée aux systèmes paramétrés, qui explicite un ensemble de conditions permettant de prouver la pred-base effective. Ces conditions définissent une nouvelle classe appelée RMTS (*Ranked Monotone Transition Systems*). Cette méthode est appliquée aux PASTD.

Mots-clés: vérification paramétrée ; *model checking* ; système d'information ; algèbre de processus ; automate ; système de transitions bien structuré ; beau préordre ; couverture ; sûreté ; accessibilité.

Remerciements

Je remercie mon directeur de thèse Marc Frappier pour son soutien à la fois scientifique, pédagogique et moral. Son ouverture d'esprit, ses encouragements et sa sympathie m'ont permis d'étudier dans un contexte très agréable. Je lui suis reconnaissant de m'avoir fait confiance en me laissant une grande liberté dans l'appréhension de mon sujet de thèse. De même, je remercie ma codirectrice de thèse Amel Mammar pour son encadrement malgré la distance. Sa grande disponibilité et ses conseils rédactionnels m'ont été d'une grande aide. Je tiens aussi à remercier notre coauteur Alain Finkel pour son expertise scientifique, mais aussi pour son encadrement complémentaire durant les derniers mois. Ses suggestions d'amélioration de présentation de mes travaux ont été essentielles. Par ailleurs, je souhaite remercier le comité d'encadrement de ma thèse composé de Richard St-Denis et de Froduald Kabanza, notre conseiller à la thèse Martin Beaudry, ainsi que mon jury de thèse pour leurs suggestions et commentaires avisés sur mon travail. Plus généralement, je suis reconnaissant envers l'ensemble de mes professeurs de m'avoir permis de développer mon intérêt pour la science, notamment envers Daniele Varacca et Eugene Asarin qui m'ont initié à la recherche scientifique.

J'aimerais remercier mes collègues et amis du GRIL d'avoir contribué à un environnement de travail agréable et sympathique durant tout mon séjour au laboratoire. En particulier, je remercie Benoît Fraikin de m'avoir mis en contact avec mon directeur de thèse et d'avoir accompagné mon intégration à l'Université de Sherbrooke. Je salue également l'efficacité et la disponibilité du personnel technique et administratif de l'université ; je pense notamment à Lise Charbonneau, Lynn Lebrun, Mario Paquet et Karine Bolduc qui m'ont été d'une aide précieuse. Enfin, je remercie mes proches, en particulier mes parents, pour leur soutien pendant toutes ces années.

Abréviations

ASTD *Algebraic State-Transition Diagram* / diagramme d'états-transitions algébrique

BDD *Binary Decision Diagram* / diagramme de décision binaire

CCS *Calculus of Communicating Systems*

CSP *Communicating Sequential Processes*

CTL *Computational Tree Logic* / logique temporelle arborescente

EB^3 *Entity-Based Black-Box specification*

IS *Information System* / système d'information

LTL *Linear Temporal Logic* / logique temporelle linéaire

LTS *Labeled Transition System* / système de transitions étiqueté

MTS *Monotone Transition System* / système de transitions monotone

NSW *Nicely Sliceable WSTS*

OTS *Ordered Transition System* / système de transitions ordonné

PASTD *Parameterized ASTD* / ASTD paramétré

PO Partial Order / ordre partiel

QBF *Quantified Boolean Formula* / formule booléenne quantifiée

Abréviations

QO *Quasi-Order* / préordre

RMTS *Ranked Monotone Transition System* / système de transitions monotone classé

SAT *Boolean SATisfiability problem* / problème de satisfaisabilité booléenne

STE Système de Transitions Étiqueté

TS Transition System / système de transitions

UML *Unified Modeling Language* / langage de modélisation unifié

VASS *Vector Addition Systems with States* / systèmes d'addition de vecteurs avec états

WPO *Well Partial Order* / bel ordre

WQO *Well-Quasi-Order* / beau préordre

WSTS *Well-Structured Transition System* / système de transitions bien structuré

Table des matières

Liste des figures

Introduction

Contexte et motivation

Les systèmes informatiques sont des systèmes complexes constitués de plusieurs structures différentes qui évoluent en interaction. Ils sont ainsi qualifiés de *systèmes dynamiques*. En particulier, les *systèmes d'information* sont des systèmes dynamiques composés de plusieurs entités en relation les unes avec les autres, et dont les données sont soumises à une évolution régulière. En effet, ils permettent de stocker et de traiter d'importantes quantités de données au fil du temps. Ils sont notamment utiles dans divers domaines comme l'entrepôt de données, la vente en ligne ou les services bancaires [10]. Par exemple, un système de gestion de bibliothèque, comportant une entité *membre*, une entité *livre* et une association *emprunt*, constitue un système d'information. Cependant, les opérations de manipulation des données mènent parfois à des situations imprévues par les concepteurs du système. Cela peut résulter en la perte ou la corruption de données importantes. Lors du développement d'un système d'information, la détection exhaustive des erreurs de conception reste difficile dans le cadre de simples tests fonctionnels ou de procédures manuelles de vérification. Ainsi, dès le début des années 80, des chercheurs ont commencé à rendre les méthodes de vérification de systèmes informatiques plus rigoureuses notamment en les automatisant [21, 80]. En effet, avec l'apparition de modèles mathématiques pour la spécification de systèmes dynamiques [59, 77], les premières approches de vérification formelle de systèmes ont vu le jour. On peut distinguer deux catégories de méthodes informatisées de vérification formelle : la *preuve automatique de théorèmes* (*automated theorem proving*) et l'*exploration de modèle* (*model checking*).

La preuve automatique de théorème découle de la théorie du *calcul des prédicats*

de Frege, qui consiste à démontrer des théorèmes, exprimés dans la logique du premier ordre, grâce à un certain nombre d'axiomes et de règles d'inférence. Du point de vue de la vérification de systèmes, il s'agit alors, à partir d'un ensemble de formules et d'une propriété, exprimée sous la forme d'un théorème, de prouver que le système respecte la propriété, en calculant un arbre de preuve pour le théorème. Cependant, bien que démontré comme complet par Gödel en 1929, le calcul des prédicats du premier ordre a aussi été démontré comme indécidable par Church en 1936 et par Turing en 1937. La preuve de théorème n'est donc, en réalité, pas totalement automatisable pour la logique du premier ordre (le *calcul propositionnel* étant, lui, décidable) et peut demander une interaction humaine. Il existe plusieurs *assistants de preuve*, comme *HOL* [54], *Isabelle* ou [74] *Coq* [11], qui permettent une vérification assistée par ordinateur pour des systèmes exprimés dans la logique des prédicats, mais d'autre méthodes sont plus adaptées au profil des systèmes dynamiques. Notamment, les *méthodes Z, B* [3] et *Event-B* [4], par exemple, permettent de modéliser les transitions d'un système plus facilement grâce à un modèle de machine abstraite.

La théorie du *model checking*, proposée indépendamment par Clarke et Emerson en 1981 [21] ainsi que par Queille et Sifakis en 1982 [80], permet, étant donné les spécifications formelles d'un système et d'une propriété, d'automatiser le processus de vérification sous certaines conditions. Le principe de la vérification par *model checking* est de parcourir algorithmiquement l'espace d'états complet d'un système afin de valider une certaine propriété. Lorsque l'espace d'états d'un système est isomorphe à un *automate fini*, il est aisé de déterminer un algorithme de parcours et ainsi de vérifier certaines propriétés. La procédure de vérification est alors entièrement automatisable. Cependant, le nombre d'états d'un système peut souvent être très grand, voire même infini dans certains cas, ce qui en fait le principal désavantage du *model checking*. En effet, la complexité algorithmique de la procédure de vérification dépend de la taille du système ainsi que de la forme de la propriété à vérifier. De plus, dans certains cas de systèmes infinis, le problème devient alors indécidable.

Nous nous intéressons dans cette thèse à la vérification par *model checking* de systèmes d'information.

Problématique

La validation de systèmes d'information est un problème qui a été abordé par Frappier *et al.* dans [40]. L'article présente une comparaison de six *model checkers* dans le cadre de l'étude d'un système simplifié de gestion de bibliothèque. Il est montré que les caractéristiques importantes pour la modélisation et la vérification de systèmes d'information sont :

- la capacité à représenter les différentes entités et associations qui composent un système d'information,
- la capacité à abstraire les instances d'entité, et
- la capacité à spécifier les propriétés usuelles pour la vérification de systèmes d'information comme les propriétés de sûreté et d'accessibilité.

La plupart des outils de *model checking* ne sont pas adaptés aux systèmes d'information. En effet, les langages de spécification permettent difficilement de spécifier la complexité des associations présentes dans de tels systèmes. De plus, lorsque l'abstraction sur les instances d'entité n'est pas possible, cela mène à une représentation souvent partielle du système d'information, car celui-ci doit être restreint à un nombre arbitraire d'instances. Dans ce cas, la taille du système augmente exponentiellement avec le nombre d'instances présentes. Cela a un impact direct sur la complexité algorithmique de la procédure de vérification. Ainsi, le problème du choix des méthodes de spécification et de vérification les plus adaptées aux systèmes d'information reste ouvert.

L'étude des méthodes de spécification formelle constitue la première étape de cette thèse. Les techniques de modélisation les plus simples existantes, telles que les structures de Kripke ou les automates [18], permettent de représenter des systèmes avec un bas niveau d'abstraction, et sont rarement utilisées pour la spécification de modèles complexes. Les modèles de machines abstraites des méthodes B et Event-B [3, 4] permettent une spécification formelle proche des langages de programmation, tandis que les algèbres de processus, comme CSP [51] ou CCS [70], permettent de décrire facilement les interactions entre divers processus. Les méthodes EB^3 [43] et

ASTD [42] sont inspirées des algèbres de processus et plus particulièrement adaptées aux systèmes d'information, car elles permettent de prendre en compte leurs structures spécifiques comportant des entités et des associations. Cependant, ces dernières méthodes ne sont pas dotées de procédures de vérification qui leur sont propres.

La deuxième étape de cette étude est l'analyse des différentes méthodes de vérification. Le principal problème du *model checking* est l'explosion combinatoire du nombre d'états. En effet, plus le système est de taille importante et la propriété à vérifier complexe, plus l'exploration de l'espace d'états est long. Ce problème peut être approché de plusieurs façons : la réduction de l'espace d'états, la compression des données, l'abstraction, etc. Cela est nécessaire en particulier pour un système dont le nombre d'états est infini, car il n'est pas possible de le parcourir entièrement. En contrepartie, ces méthodes imposent de fortes contraintes sur les systèmes et les propriétés à vérifier, et sont parfois incomplètes.

Outre sa structure basée sur des entités et des associations, un système d'information possède une autre caractéristique importante : il s'apparente à une classe particulière de systèmes non bornés qui est la classe des systèmes paramétrés. En effet, un système d'information possède des symétries intrinsèques dues à la réplication de structures identiques et à leurs compositions en parallèle. Intuitivement, le nombre de structures répliquées correspondant à une entité définit un paramètre du système. Par exemple, un système de gestion de bibliothèque aura pour paramètres le nombre de processus *membre* et le nombre de processus *livre*. Si l'on note $\mathcal{S}(n)$ un système dont le paramètre est n et ϕ une propriété à vérifier, le problème de la vérification paramétrée consiste alors à vérifier que pour tout $n \in \mathbb{N}$, $\mathcal{S}(n)$ vérifie la propriété ϕ. Ce problème est montré comme étant indécidable dans le cas général par Apt et Kozen [6]. Cependant, des méthodes de vérification paramétrée ont été proposées pour résoudre certains cas particuliers. Parmi elles, on retrouve des méthodes de vérification par abstraction [79, 20], des méthodes nécessitant de déterminer une limite appelée *cutoff* [34, 55], ou encore des méthodes nécessitant un raisonnement inductif sur la taille du système [95, 35, 63, 78]. De même, les méthodes de vérification de systèmes infinis [26, 1, 47, 57, 39] sont adaptables aux systèmes paramétrés, en considérant que l'union sur n de tous les $\mathcal{S}(n)$ est un système infini. Ainsi, la vérification paramétrée de systèmes d'information est l'un des problèmes principaux de cette thèse.

Méthodologie

La plupart des techniques existantes de vérification paramétrée sont associées à des méthodes particulières de spécification des systèmes. Ces méthodes sont limitées par des restrictions sur le type des systèmes pris en charge. Cela peut être trop contraignant si l'on souhaite modéliser des systèmes aussi complexes que les systèmes d'information. Par exemple, les méthodes décrites dans [34, 78, 79, 20] ne permettent pas de prendre en compte la multiplicité des entités et des associations. Ainsi, les techniques de vérification les plus générales restent les plus simples à adapter aux systèmes d'information. L'une des méthodes de vérification les plus abstraites s'appliquant aux systèmes infinis est la méthode des *systèmes de transitions bien structurés* (WSTS). Les WSTS, introduits par Finkel dans [37], sont des systèmes dont l'espace d'états est muni d'un *beau préordre* (WQO) et qui respectent une propriété de *monotonie*. Ces conditions étant relativement abstraites, la théorie des WSTS peut donc s'appliquer à un grand nombre de modèles de systèmes, et donc à certains systèmes paramétrés. Dans [1], Abdulla *et al.* proposent un algorithme pour les WSTS de résolution du problème de couverture qui se définit comme un problème d'accessibilité d'un ensemble d'états clos par le haut (par la relation de préordre). Cette thèse s'appuie sur cette méthode de *model checking* afin de déterminer un processus de vérification paramétrée pour les systèmes d'information. De plus, tout problème de sûreté étant équivalent à un problème d'accessibilité, il est possible d'identifier une classe de propriétés de sûreté pertinentes pour les systèmes d'information et qui peuvent être vérifiées par une analyse de couverture.

La solution choisie pour représenter les systèmes d'information se base sur la notation graphique des *diagrammes états-transitions algébriques* (ASTD) introduite par Frappier *et al.* dans [42]. Les ASTD sont une méthode de spécification formelle qui s'inspire des *statecharts* de Harel [49] et de l'algèbre de processus *Entity-based black-box specification* (EB^3) de Frappier et St-Denis [43]. Cette méthode a été proposée dans le but de faciliter la représentation de systèmes d'information tout en conservant la rigueur syntaxique et sémantique d'une algèbre de processus. Afin de permettre la représentation des systèmes paramétrés de façon explicite, on propose, à partir d'une sous-classe d'ASTD, une extension de la notation que l'on nomme *Parame-*

terized ASTD (PASTD). Les PASTD introduisent la notion de paramètre grâce à des variables représentant des ensembles non bornés d'identifiants et permettent de considérer l'ensemble de toutes les instances de PASTD comme un système infini. L'avantage de la méthode des PASTD réside à la fois dans sa notation graphique et dans sa capacité à décrire les systèmes paramétrés. Les modèles graphiques, comme Stateflow [65], UML [84] ou iUML-B [89], permettent de spécifier et de manipuler des machines à états hiérarchiques de façon plus simple et sont largement utilisés dans l'industrie, notamment l'aéronautique et l'industrie automobile [24]. D'autre part, la plupart des modèles formels ne permettent pas de spécifier aisément des systèmes paramétrés comme des systèmes d'information. Les réseaux de Petri, par exemple, sont insuffisants pour la modélisation des associations d'un système d'information paramétré.

L'étude de l'application de la théorie des WSTS aux PASTD met en évidence les contraintes nécessaires à la vérification des PASTD. Déterminer un préordre sur l'espace d'états des PASTD, qui satisfait à la fois les conditions de WQO et de monotonie, constitue l'une des difficultés majeures de cette thèse. D'une part, la structure en forme d'arbre des états de PASTD nous mène à explorer les différents WQO sur les graphes de la littérature tels que l'ordre de Kruskal sur les arbres [60], l'ordre mineur de Robertson et Seymour sur les graphes [82], ou encore l'ordre sous-graphe restreint à une classe particulière de graphes proposé par Ding [30]. Ce dernier étant le plus adapté à l'étude des PASTD, il impose néanmoins une contrainte importante sur le type de PASTD que l'on peut considérer. De ce fait, on définit une sous-classe de PASTD appelée *Bounded-PASTD* qui vérifie cette contrainte. D'autre part, la condition de monotonie, associée au préordre choisi, ne permet pas d'inclure certains opérateurs des ASTD dans la sémantique des PASTD. Par exemple, les synchronisations quantifiées sur des ensembles non bornés d'instances ne sont pas permises dans notre étude. Ainsi, il s'agit de déterminer une sous-classe de PASTD qui soit pertinente pour la modélisation de systèmes d'information tout en satisfaisant les propriétés des WSTS.

Dans la perspective de l'analyse de couverture de [1] qui correspond à un parcours en arrière des états, les PASTD doivent vérifier une condition supplémentaire importante qui est l'existence de *pred-base effective*, formulée par Finkel et Schnoebelen

dans [39]. Elle définit l'existence d'un algorithme de calcul d'une base finie de prédécesseurs pour chaque ensemble d'états clos par le haut. Bien que cette condition soit facile à montrer dans certains cas de modèles simples, dans le cadre des PASTD, cela demande un effort plus conséquent. On définit ainsi un nouveau cadre théorique appelé *Ranked monotone transition system* (RMTS) qui explicite un ensemble pratique de conditions permettant de prouver l'existence de pred-base effective. Les PASTD sont alors démontrés comme étant RMTS.

Contribution

Voici les contributions principales de cette thèse :

- *Une étude des méthodes de spécification et de vérification pour les systèmes d'information* : les caractéristiques importantes de comparaison des différentes méthodes sont la représentation des paramètres et la gestion de l'interaction des entités.
- *L'extension d'un langage de spécification formelle pour les systèmes paramétrés* : la classe des PASTD permet de représenter la plupart des particularités des systèmes d'information.
- *La proposition d'une sous-classe de PASTD qui est WSTS* : les *Bounded-PASTD* vérifient les propriétés de WQO et de monotonie pour un préordre pertinent pour la vérification.
- *La définition d'une nouvelle classe de systèmes monotones appelée RMTS* : elle permet de prouver l'existence de pred-base effective pour les systèmes monotones.
- *Une preuve formelle que les PASTD sont RMTS* : le problème de couverture des *Bounded-PASTD* est donc décidable.

Plan de la thèse

La thèse est organisée comme suit.

- Le chapitre 1 introduit les notions de base et présente brièvement différentes méthodes de spécification formelle pour les systèmes dynamiques.
- Le chapitre 2 fait l'état de l'art des techniques de vérification formelle, notamment des techniques de *model checking* pour les systèmes infinis et les systèmes paramétrés.
- Le chapitre 3 présente les problèmes soulevés par la spécification et la vérification de systèmes d'information. On y évalue les différentes méthodes existantes et on établie le lien entre le problème de la sûreté et celui de l'accessibilité.
- Le chapitre 4 étudie le problème de la couverture pour les PASTD à travers la théorie des WSTS. Il s'agit du chapitre central de cette thèse, il présente les principales contributions.
- Le chapitre 5 précise la méthodologie utilisée dans le chapitre 4 en explicitant les différents choix effectués.
- La conclusion résume et commente les résultats de la thèse. On y discute aussi les différentes perspectives de recherche.
- Les annexes A et B détaillent certaines des définitions et des preuves des théorèmes du chapitre 4 avec une syntaxe proche de la syntaxe originale des ASTD.

Chapitre 1

Méthodes de spécification formelle

Les *méthodes formelles* désignent un ensemble de techniques permettant de modéliser et manipuler rigoureusement des systèmes informatiques, notamment des systèmes dynamiques comme les systèmes d'information. L'avantage des méthodes formelles, par rapport aux méthodes de spécification plus classiques comme les modèles UML [84], est qu'elles permettent de produire des spécifications sans ambiguïté d'interprétation, de générer automatiquement dans certains cas du code vérifié, ou encore de valider mathématiquement certaines propriétés du système. Ce chapitre présente quelques méthodes importantes de spécification formelle qui sont adaptées aux systèmes dynamiques.

Parmi les outils mathématiques utiles à la représentation de systèmes dynamiques, on retrouve la logique des prédicats, les automates ou les langages formels. La plupart des systèmes simples peuvent être modélisés par des diagrammes d'états-transitions, dont il existe plusieurs variantes [18, 53, 14]. Les propriétés que l'on souhaiterait vérifier sur un système peuvent, quant à elles, être spécifiées par des formules de la logique des prédicats ou de la logique temporelle [77, 21, 33]. Pour des systèmes plus complexes, il est souvent plus pratique d'utiliser des modèles à base de machines abstraites, dont la syntaxe est proche des langages de programmation, comme les *méthodes Z, B* ou *Event-B* [3, 4], ou des algèbres de processus dotés d'opérateurs permettant de composer différents processus entre eux [70, 16, 51, 72].

La suite du chapitre est organisée de la façon suivante. La section 1.1 présente les systèmes de transition d'états, ses deux principales variantes les *automates* et les *structures de Kripke*. On y définit, par ailleurs, les *systèmes de transitions étiquetés* qui seront étudiés dans les chapitres suivants. La section 1.2 présente les langages formels et fait le lien avec les langages associés aux systèmes de transitions. La section 1.3 introduit la logique temporelle et la section 1.4 les différentes catégories de propriétés temporelles. On présente brièvement la *méthode B* dans la section 1.5, puis les algèbres de processus, notamment CSP [51] et EB^3 [43], dans la section 1.6. Enfin, la section 1.7 présente la méthode de spécification nommée ASTD que l'on étudiera dans les chapitres suivants.

1.1 Systèmes de transition d'états

Un système dynamique peut être décrit par l'ensemble des états dans lesquels il peut se trouver, ainsi que par une relation de transition reliant ces états. On nomme *système de transition d'états* (ou système de transitions) un tel système. Il existe plusieurs types de systèmes de transitions comme les structures de Kripke ou les automates, par exemple.

1.1.1 Structures de Kripke et automates

Définition 1.1 (Structure de Kripke)**.** *Une structure de Kripke est un 6-uplet* $K = (Q, T, I, AP, l)$ *tel que :*

- Q *est un ensemble d'états ;*
- $T \subseteq Q \times Q$ *est la relation de transition ;*
- $I \subseteq Q$ *est un ensemble d'états initiaux ;*
- AP *est un ensemble de propositions atomiques ;*
- $l : Q \to 2^{AP}$ *est une fonction d'étiquetage.*

Plus généralement, on définit un système de transition d'états par un triplet (Q, T, I) constitué d'un ensemble d'états Q, une relation de transition $T \subseteq Q \times Q$ et un ensemble d'états initiaux $I \subseteq Q$. La structure de Kripke ajoute à chaque état un étiquetage par un sous-ensemble de propositions atomiques. On appelle *exécution (ou chemin)* dans un système de transition d'états toute suite d'états, potentiellement infinie, de Q telle que le premier est un état initial de I et que toute paire d'états consécutifs appartient à la relation de transition T. Dans une structure de Kripke, l'ensemble des propositions atomiques permet de représenter les valeurs des variables du système à chaque étape et ainsi de modéliser les différents états du système. Une évolution du système est donc décrite par une suite de sous-ensembles de AP.

Une structure de Kripke modélise l'évolution de l'état courant d'un système. Cependant, dans certains cas, il peut être utile de connaitre l'évènement associé à chaque transition que le système effectue. Un *automate* permet de décrire cela en définissant un ensemble d'*évènements (ou d'actions)* et une relation de transition telle que chaque transition du système est étiquetée par un évènement.

Définition 1.2 (Automate [18])**.** *Un automate est un 5-uplet $A = (Q, \Sigma, T, I, F)$ où :*

- *Q est un ensemble d'états ;*
- *Σ est un ensemble d'évènements ;*
- *$T \subseteq Q \times \Sigma \times Q$ est la relation de transition ;*
- *$I \subseteq Q$ est un ensemble d'états initiaux ;*
- *$F \subseteq Q$ est un ensemble d'états finaux.*

Une exécution du système est définie par une suite finie d'états et d'évènements, alternativement, $(q_0, a_0, q_1, a_1, \ldots, q_n, a_n, q_{n+1})$ telle que $q_0 \in I$ et pour tout $i \in 1..n$, $q_i \in Q$, $a_i \in \Sigma$ et $(q_i, a_i, q_{i+1}) \in T$. On appelle *trace* la projection d'une exécution sur l'ensemble des évènements $(a_0, a_1, \ldots, a_n)$. On dit qu'une trace est acceptée par l'automate si le dernier état de l'exécution est final, *i.e.* $q_{n+1} \in F$. Un automate décrit ainsi un ensemble de traces acceptés, appelé *langage* de A, que l'on note $\mathcal{L}(A)$. On parle d'automate *déterministe* lorsque I est un singleton et que pour tout $q \in Q$ et $a \in \Sigma$, il existe au plus un $q' \in Q$ tel que $(q, a, q') \in T$.

Une variante d'automate appelée *automate de Büchi* permet de décrire des systèmes à comportement infini, en acceptant des traces d'exécution infinies. Un automate de Büchi est un automate dont la condition d'acceptation d'une trace est définie de façon différente. Une exécution infinie est acceptée par un automate de Büchi (Q, Σ, T, I, F) si elle passe "infiniment souvent" par un état final. Autrement dit, $(q_0, a_0, q_1, a_1, \ldots, q_n, a_n, q_{n+1} \ldots)$ est acceptée s'il existe une infinité d'indices $i_0, i_1, i_2 \ldots$ telle que $q_{i_0}, q_{i_1}, q_{i_2} \ldots \in F$.

Selon le système à modéliser, il peut être plus adapté d'utiliser une structure de Kripke ou un automate. Le modèle des structures de Kripke met l'accent sur la description des états d'un système tandis que le modèle des automates s'appuie sur la représentation des scénarios modélisés par des suites d'évènements. Toutefois, il est aussi possible de considérer des systèmes de transitions $S = (Q, \Sigma, T, I, AP, l)$, dont les états et les transitions sont étiquetés, afin de raisonner à la fois sur les évènements et sur les états. Les méthodes décrites dans les chapitres suivants sont souvent adaptées à une représentation particulière du système de transitions. Cependant, on pourra parfois passer d'une représentation à une autre.

1.1.2 Systèmes de transitions étiquetés

Dans la suite de ce manuscrit, on utilisera principalement la définition de *système de transitions étiqueté* (STE). Un tel système de transitions est décrit par un quadruplet $S = (Q, \Sigma, \rightarrow, I)$, *i.e.* un automate dont on ne considère pas d'états finaux. De même, on emploiera les notations suivantes. Soit $S = (Q, \Sigma, \rightarrow, I)$ un système de transitions étiqueté. L'ensemble des traces de S est noté $\mathrm{tr}(S)$, l'ensemble des traces finies $\mathrm{tr}_{\mathrm{f}}(S)$ et l'ensemble des traces infinies $\mathrm{tr}_{\omega}(S)$, où $\mathrm{tr}(S) = \mathrm{tr}_{\mathrm{f}}(S) \cup \mathrm{tr}_{\omega}(S)$. On note $q' \xrightarrow{a} q$ la transition (q', a, q), $q' \rightarrow q$ s'il existe une transition de q' vers q et $q' \xrightarrow{*} q$ s'il existe une exécution de q' vers q. On définit par $Pred(q) = \{q' \in Q \mid q' \rightarrow q\}$ l'ensemble des prédécesseurs immédiats de q et par $Pred^*(q) = \{q' \in Q \mid q' \xrightarrow{*} q\}$ l'ensemble de tous les prédécesseurs de q. On définit de façon similaire l'ensemble des successeurs $Succ(q)$ et $Succ^*(q)$. Le système est dit *sans blocage* si $Succ(q) \neq \emptyset$ pour tout $q \in Q$. Il est dit à *branchement fini* si $Succ(q)$ et $Pred(q)$ sont finis pour tout $q \in Q$.

Afin de pouvoir composer différents STE entre eux, on définit deux opérateurs : l'union et la composition parallèle. Pour $i \in \{1,2\}$, on pose $S_i = (Q_i, \Sigma_i, \rightarrow_i, I_i)$ un STE.

1. L'union de S_1 et S_2, notée $S_1 \cup S_2$, est le STE $(Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \rightarrow_1 \cup \rightarrow_2, I_1 \cup I_2)$.

2. Soit Δ un alphabet. La *composition parallèle* sur Δ de S_1 et S_2, notée $S_1 \parallel_\Delta S_2$, est le STE $(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \rightarrow_\parallel, I_1 \times I_2)$ où $\rightarrow_\parallel$ est un ensemble de transitions $((q_1, q_2), a, (q_1', q_2'))$ tel que

   - $a \in \Delta$ et $(q_i, a, q_i') \in \rightarrow_i$ pour $i \in \{1,2\}$, ou
   - $a \in \Sigma_i \backslash \Delta$ et $(q_i, a, q_i') \in \rightarrow_i$ et $q_j = q_j'$ pour $i, j \in \{1,2\}$ et $i \neq j$.

   Cas particuliers :

   (a) Si $\Delta = \Sigma_1 \cap \Sigma_2$, on note simplement $S_1 \parallel S_2$ la composition parallèle par défaut ;

   (b) Si $\Delta = \Sigma_1 \cup \Sigma_2$, on note $S_1 \otimes S_2$ que l'on appelle *produit synchrone* ;

   (c) Si $\Delta = \emptyset$, on note $S_1 \mid\mid\mid S_2$ et on parle d'*entrelacement* de S_1 et S_2.

   De plus, on note $\left|\left|\left|^i S\right.\right.\right.$ pour $i \in \mathbb{N}_1$, où $\mathbb{N}_1 = \mathbb{N} \backslash \{0\}$, le STE défini inductivement par $\left|\left|\left|^1 S = S\right.\right.\right.$ et $\left|\left|\left|^{n+1} S = (\left|\left|\left|^n S) \mid\mid\mid S\right.\right.\right.$. La *clôture positive de l'entrelacement* de S, notée $\left|\left|\left|^+ S\right.\right.\right.$, correspond au STE infini $\bigcup_{i \in \mathbb{N}_1} \left|\left|\left|^i S\right.\right.\right.$.

D'autres modèles de systèmes dynamiques basés sur les systèmes de transitions existent dans la littérature scientifique. Certains sont des variantes de systèmes de transitions couplées à des structures de données externes facilitant la représentation d'états plus complexes. Les *systèmes à addition de vecteurs avec états* [53] (*vector addition systems with states*, VASS), par exemple, sont dotés de vecteurs d'entiers mis à jour par des transitions incrémentant ou décrémentant une des composantes. Les VASS sont équivalents aux réseaux de Petri [53, 76]. Un autre exemple de modèle sont les *systèmes à canaux FIFO* [14]. Ces systèmes sont munis de canaux permettant la communication entre différents processus qui peuvent y lire et y écrire des messages. On apportera un intérêt plus particulier au cas des *systèmes à canaux non fiables* [38]

(*lossy channel systems*, LCS) dans les chapitres suivants. Ce modèle, contrairement aux systèmes à canaux normaux, prend en charge la perte aléatoire de messages dans les canaux.

1.2 Langages formels

En théorie des langages, on définit un *langage formel* comme un ensemble de mots. Du point de vue des systèmes informatiques, on peut considérer qu'une exécution ou une trace forment un mot. Par exemple, un langage peut être constitué par l'ensemble des traces qu'un STE peut exécuter, ou par l'ensemble des traces qu'un automate accepte.

Définition 1.3 (Langage)**.** *Formellement, on appelle alphabet un ensemble Σ dont les éléments sont appelés des lettres. Un mot sur Σ est une suite de lettres $(a_1, a_2 \ldots)$, aussi notée $a_1 a_2 \ldots$ telle que $a_i \in \Sigma$ pour tout $i \in \mathbb{N}_1$. Le mot vide est noté ε. On note Σ^* et Σ^ω l'ensemble des mots, respectivement, finis et infinis sur Σ et $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. On appelle alors langage sur Σ tout sous-ensemble $L \subseteq \Sigma^\infty$. Le langage L est dit finitaire si $L \subseteq \Sigma^*$ et infinitaire si $L \subseteq \Sigma^\omega$.*

L'alphabet d'un langage L, *i.e.* l'ensemble des lettres apparaissant dans L, est noté $\alpha(L)$. La concaténation de deux mots $u = a_0 a_1 \ldots a_n \in \Sigma^*$ et $v = b_0 b_1 \ldots \in \Sigma^\infty$ est définie par le mot noté $uv = a_0 a_1 \ldots a_n b_0 b_1 \ldots$ Un mot $u \in \Sigma^*$ est un préfixe de $w \in \Sigma^\infty$ s'il existe $v \in \Sigma^\infty$ tel que $w = uv$. On note $\mathrm{pref}(u)$ l'ensemble des préfixes de u.

Outre les opérateurs ensemblistes usuels d'union, d'intersection et de complément, on peut définir des opérateurs spécifiques aux langages comme la *concaténation*, la *fermeture (ou étoile) de Kleene* et le *mélange*.

Définition 1.4 (Concaténation)**.** *Soient Σ un alphabet, $L_1 \subseteq \Sigma^*$ et $L_2 \subseteq \Sigma^\infty$. On appelle concaténation de langages de L_1 et L_2, le langage noté $L_1 L_2$ et défini par :*

$$L_1 L_2 = \{u \in \Sigma^\infty \mid \exists v \in \Sigma^* \cdot \exists w \in \Sigma^\infty \cdot u = vw\}$$

Définition 1.5 (Itération)**.** *Soient Σ un alphabet et $L \subseteq \Sigma^*$. On définit la fermeture de Kleene par $L^* = \bigcup_{i \in \mathbb{N}} L^i$, où $L^0 = \{\varepsilon\}$ et $L^i = L^{i-1} L$ pour tout $i \in \mathbb{N}_1$. Par ailleurs,*

on note la fermeture positive $L^+ = \bigcup_{i\in\mathbb{N}_1} L^i$ *et la* ω*-itération la concaténation infinie* $L^\omega = LL\ldots$

Définition 1.6 (Mélange)**.** *Soient* Σ *un alphabet et* $u, v \in \Sigma^\infty$*. L'ensemble des mélanges de* u *et de* v*, noté* $u \sqcup\!\sqcup v$ *est défini par l'ensemble des mots* $w \in \Sigma^\infty$ *vérifiant l'une des deux conditions suivantes.*

1. $w = u_1v_1u_2v_2\ldots$*, où* $u_i, v_i \in \Sigma^*$*, pour tout* $i \in \mathbb{N}_1$*, sont tels que* $u = u_1u_2\ldots$ *et* $v = v_1v_2\ldots$*;*
2. $w = u_1v_1u_2v_2\ldots$*, avec* $u_i, v_i \in \Sigma^*$*, pour tout* $i \in \mathbb{N}_1$*, et soit* $u_1u_2\ldots \in \mathrm{pref}(u)$ *et* $v = v_1v_2\ldots \in \Sigma^\infty$*, soit* $u_1u_2\ldots \in \Sigma^\infty$ *et* $v = v_1v_2\ldots \in \mathrm{pref}(v)$*.*

Le mélange de deux langages $L_1, L_2 \subseteq \Sigma^\infty$ *est défini par* $L_1 \sqcup\!\sqcup L_2 = \{w \in u \sqcup\!\sqcup v \mid u \in L_1 \wedge v \in L_2\}$*.*

Définition 1.7 (Fermeture de mélange)**.** *Soient* Σ *un alphabet et* $L \subseteq \Sigma^\infty$*. La fermeture de mélange, ou le mélange itéré, de* L *est définie par* $\sqcup\!\sqcup^* L = \bigcup_{i\in\mathbb{N}} \sqcup\!\sqcup^i L$*, où* $\sqcup\!\sqcup^0 L = \{\varepsilon\}$ *et* $\sqcup\!\sqcup^i L = (\sqcup\!\sqcup^{i-1} L) \sqcup\!\sqcup L$ *pour tout* $i \in \mathbb{N}_1$*. La fermeture positive est définie par* $\sqcup\!\sqcup^+ L = \bigcup_{i\in\mathbb{N}_1} \sqcup\!\sqcup^i L$*.*

Si l'on considère le langage décrit par l'ensemble des traces d'un STE, il existe un lien entre les opérateurs des langages et ceux des STE.

Proposition 1.1. *Soit* S_1 *et* S_2 *deux STE.*

1. $\mathrm{tr}(S_1 \cup S_2) = \mathrm{tr}(S_1) \cup \mathrm{tr}(S_2)$
2. $\mathrm{tr}(S_1 \otimes S_2) = \mathrm{tr}(S_1) \cap \mathrm{tr}(S_2)$
3. $\mathrm{tr}(S_1 \,|||\, S_2) = \mathrm{tr}(S_1) \sqcup\!\sqcup \mathrm{tr}(S_2)$
4. $\mathrm{tr}(|||^+ S_1) = \sqcup\!\sqcup^+ \mathrm{tr}(S_1)$

Une classe notable de langages formels est celle des langages réguliers. Cette classe fait partie des *langages récursifs*, *i.e.* des langages Turing décidables. Plus précisément, d'après le théorème de Kleene, les langages réguliers sont équivalents aux langages reconnaissables par des automates. On définit cette classe de langages finitaires de la façon suivante.

Définition 1.8 (Langage régulier)**.** *Soit Σ un alphabet. Les langages réguliers (ou rationnels) sur Σ sont définis inductivement par :*

1. *$\{\varepsilon\}$ et $\emptyset$ sont des langages réguliers ;*
2. *Pour tout $a \in \Sigma$, $\{a\}$ est un langage régulier ;*
3. *Si $L_1, L_2 \subseteq \Sigma^*$ sont des langages réguliers, alors $L_1 \cup L_2$, $L_1 L_2$ et L_1^* sont des langages réguliers.*

De même, il existe une caractérisation similaire d'une classe de langages infinitaires, appelée *langages ω-réguliers*. Ces langages sont équivalents aux langages reconnus par les automates de Büchi.

Définition 1.9 (Langage ω-régulier)**.** *Soit Σ un alphabet. Les langages ω-réguliers sur Σ sont définis inductivement par :*

1. *Si $L \subseteq \Sigma^*$ est un langage régulier, $L \neq \emptyset$ et $\varepsilon \notin L$, alors L^ω est un langage ω-régulier ;*
2. *Si $L_1 \subseteq \Sigma^*$ est un langage régulier et $L_2 \subseteq \Sigma^\omega$ est un langage ω-régulier, alors $L_1 L_2$ est un langage ω-régulier ;*
3. *Si $L_1, L_2 \subseteq \Sigma^\omega$ sont des langages ω-réguliers, alors $L_1 \cup L_2$ est un langage ω-régulier.*

Par ailleurs, on notera que les langages réguliers et ω-réguliers sont clos par union, intersection et complémentation.

1.3 Logiques temporelles

Une logique temporelle est une logique modale servant à décrire en informatique l'évolution dans le temps d'un système. Elle permet de représenter de façon succincte le comportement particulier d'une partie d'un système sans pour autant décrire précisément le système dans son intégralité. Une formule de logique temporelle peut s'exprimer aussi bien sur les variables d'états que sur les évènements. Les deux logiques temporelles les plus utilisées en spécification sont la *logique temporelle linéaire*

(Linear Temporal Logic, LTL) de Pnueli [77] et la *logique temporelle arborescente* (Computational Tree Logic, CTL) de Clarke et Emerson [21]. Elles sont toutes deux des fragments de la logique temporelle CTL* définie plus tard par Emerson et Halpern [33].

1.3.1 Logique temporelle linéaire (LTL)

LTL permet de spécifier une propriété sur une exécution dans un système. Elle s'appuie sur la logique propositionnelle étendue d'opérateurs modaux temporels caractérisant le futur de l'état. Les opérateurs temporels sont les suivants.

- $\mathrm{X}\,\phi$ (*next*) : la proposition ϕ est vraie dans l'état suivant ;
- $\phi\,\mathrm{U}\,\psi$ (*until*) : ϕ est vraie jusqu'à ce que ψ soit vraie ;
- $\mathrm{F}\,\phi$ (*finally*) : ϕ est vérifiée dans un état futur ;
- $\mathrm{G}\,\phi$ (*globally*) : ϕ est vérifiée dans tous les états futurs.

Notons ϕ une formule LTL et p une proposition atomique. La syntaxe LTL est définie par la grammaire suivante.

$$\phi ::= \bot \mid p \mid \neg\phi \mid \phi \vee \phi \mid \mathrm{X}\,\phi \mid \phi\,\mathrm{U}\,\phi$$

$$\top := \neg\bot \qquad \phi \wedge \psi := \neg(\neg\phi \vee \neg\psi) \qquad \mathrm{F}\,\phi := \top\,\mathrm{U}\,\phi \qquad \mathrm{G}\,\phi := \neg\,\mathrm{F}\,\neg\phi$$

Soit un mot infini $w = a_0a_1a_2\ldots \in \Sigma^\omega$ sur un alphabet Σ. On note $w \models \phi$ lorsque le mot satisfait la formule ϕ et $w, i \models \phi$ lorsque le sous-mot $a_ia_{i+1}\ldots$ satisfait ϕ pour $i \in \mathbb{N}$. Considérons que les propositions atomiques sont les lettres de Σ. La sémantique formelle de LTL est donnée de la façon suivante.

$w, i \not\models \bot$

$w, i \models a$	**si**	$a_i = a$
$w, i \models \neg\phi$	**si**	$w, i \not\models \phi$
$w, i \models \phi \vee \psi$	**si**	$w, i \models \phi$ ou $w, i \models \psi$
$w, i \models \mathrm{X}\,\phi$	**si**	$w, i+1 \models \phi$
$w, i \models \phi\mathrm{U}\psi$	**si**	$\exists k \cdot i \leq k$ et $w, k \models \psi$ et $\forall j \cdot (i \leq j < k) \implies w, j \models \phi$

Notons qu'il s'agit d'une définition minimaliste et que les symboles $\bot$ (faux), $\wedge$ (et), F (*finally*) et G (*globally*) sont définis à partir des autres.

Une formule LTL Φ sur un alphabet Σ dénote ainsi un langage infinitaire $\Phi \subseteq \Sigma^\omega$. Par ailleurs, l'ensemble des langages LTL correspond à la classe des *langages infinitaires sans étoiles*, qui est une sous-classe de langages ω-réguliers. Toute formule LTL peut donc être représentée par un automate de Büchi. Par exemple, soient $\Sigma = \{a, b, c\}$ un alphabet et $\Phi = \mathrm{F}\, a$ une formule LTL sur Σ. Φ exprime le fait que tout mot du langage doit contenir la lettre a. Elle est équivalente au langage ω-régulier donné par l'expression $\Sigma^* a \Sigma^\omega = \{w \in \Sigma^\omega \mid \exists w_1 \in \Sigma^*, \exists w_2 \in \Sigma^\omega \cdot w = w_1 a w_2\}$.

Il est possible d'exprimer une formule LTL sur l'ensemble de ses évènements d'un STE. Soient Σ un alphabet et S un STE sans blocage, on définit une *propriété temporelle linéaire* Φ sur Σ comme une formule LTL correspondant à un langage infinitaire $\Phi \subseteq \Sigma^\omega$. On dit que S satisfait la propriété Φ, noté $S \models \Phi$, si et seulement si $\mathrm{tr}_\omega(S) \subseteq \Phi$.

1.3.2 Logique temporelle arborescente (CTL)

Contrairement à LTL, CTL est une logique arborescente. Elle permet de spécifier une propriété portant sur les différentes branches d'exécution possibles à partir d'un état donné plutôt que sur une unique trace d'exécution. En CTL, les opérateurs modaux temporels sont similaires à ceux de LTL, cependant chacun d'eux est couplé à un quantificateur sur les branches d'exécutions : soit A (*always*), un quantificateur universel, soit E (*eventually*), un quantificateur existentiel. Intuitivement, le quantificateur A stipule que toute branche d'exécution partant de l'état d'origine satisfait la formule temporelle qui suit. Tandis que la présence du quantificateur E signifie qu'il existe une branche qui satisfait la formule.
La syntaxe CTL est définie de la façon suivante.

$$\phi ::= \bot \mid p \mid \neg\phi \mid \phi \vee \phi \mid \mathrm{EX}\,\phi \mid \mathrm{AX}\,\phi \mid \mathrm{E}\,\phi\,\mathrm{U}\,\phi \mid \mathrm{A}\,\phi\,\mathrm{U}\,\phi$$

$\top := \neg\bot$ $\qquad$ $\phi \wedge \psi := \neg(\neg\phi \vee \neg\psi)$
$\mathrm{EF}\,\phi := \mathrm{E}\,\top\,\mathrm{U}\,\phi$ $\qquad$ $\mathrm{AF}\,\phi := \mathrm{A}\,\top\,\mathrm{U}\,\phi$
$\mathrm{EG}\,\phi := \neg\,\mathrm{AF}\,\neg\phi$ $\qquad$ $\mathrm{AG}\,\phi := \neg\,\mathrm{EF}\,\neg\phi$

Considérons une structure de Kripke $K = (Q, T, I, AP, l)$ et un état $s \in Q$. On note $s \models \phi$ lorsque l'état s vérifie la propriété ϕ. La sémantique formelle de CTL est la suivante.

$s \models p$	**si**	$p \in l(s)$
$s \models \mathrm{EX}\,\phi$	**si**	$\exists s' \cdot (s, s') \in T \wedge s' \models \phi$
$s \models \mathrm{AX}\,\phi$	**si**	$\forall s' \cdot (s, s') \in T \implies s' \models \phi$
$s \models \mathrm{E}\,\phi\,\mathrm{U}\,\psi$	**si**	$\exists (s, s_1, s_2, \ldots, s_j)$ un chemin fini tel que
		$s_j \models \psi$ et $\forall k \cdot 1 \leq k < j \implies s_k \models \phi$
$s \models \mathrm{A}\,\phi\,\mathrm{U}\,\psi$	**si**	$\forall (s, s_1, s_2 \ldots)$ chemin infini, $\exists j \in \mathbb{N}$ tel que
		$s_j \models \psi$ et $\forall k \cdot 1 \leq k < j \implies s_k \models \phi$

Remarquons que, contrairement à LTL, une formule CTL ne décrit pas un ensemble de traces d'exécution mais caractérise un sous-ensemble d'états du système. Une formule CTL n'est donc pas comparable à un langage formel.

1.3.3 CTL*

CTL* est une logique temporelle arborescente qui combine les opérateurs temporels et les quantificateurs de chemin de façon plus libre que CTL. Elle a été proposée par Emerson et Halpern en 1986 [33] comme extension à CTL et est strictement plus expressive que LTL et CTL. On distingue dans ce cas des *formules d'état* et des *formules de chemin*. Les formules d'état commencent toujours par un quantificateur de chemin lorsqu'elles contiennent un opérateur temporel et peuvent caractériser un état. Tandis que les formules de chemin caractérisent les traces.

Les formules d'état de CTL* sur un ensemble de propositions atomiques AP sont définies par la syntaxe.

$$\Phi ::= \bot \mid p \mid \neg\Phi \mid \Phi \vee \Phi \mid \mathrm{E}\,\phi$$

où $p \in AP$ et ϕ est une formule de chemin exprimée selon la syntaxe suivante.

$$\phi ::= \Phi \mid \neg\phi \mid \phi \vee \phi \mid \mathrm{X}\,\phi \mid \phi\,\mathrm{U}\,\phi$$

où Φ désigne une formule d'état et ϕ une formule de chemin.

On parle du *fragment universel* de CTL* noté ACTL* lorsque les formules sont écrites sous forme normale positive et ne contiennent pas de quantificateurs existentiels. De façon duale, on parle aussi de *fragment existentiel* noté ECTL*. On note $\text{CTL}^*_{\setminus X}$ l'ensemble des formules CTL* n'utilisant pas l'opérateur X.

1.4 Propriétés temporelles

Il est parfois utile de caractériser un système par un ensemble de propriétés qu'il vérifie, notamment pour le valider. Il existe plusieurs types de propriétés : les propriétés d'invariant, de sûreté, de vivacité, d'accessibilité, d'équité etc.

Une *propriété d'invariant* est une caractéristique du système qui restera toujours vraie tout au long de son exécution. C'est l'une des propriétés les plus simples à spécifier pour les systèmes. Par exemple, elle peut exprimer le fait qu'une variable du système ne dépassera jamais une certaine valeur, ce qui peut être représenté par une formule propositionnelle.

Alpern et Schneider décrivent en 1985 deux types de propriétés temporelles, qualifiées de propriétés temporelles linéaires, [5] définissant des ensembles de traces d'exécution : les *propriétés de sûreté et de vivacité*. Formellement, une propriété de sûreté est un sous-ensemble S des traces du système tel que : *pour toute trace u n'appartenant pas à S, il existe un préfixe fini $v \in \text{pref}(u)$ tel que toute trace w prolongeant ce préfixe ($v \in \text{pref}(w)$) n'appartient pas à S*. Une propriété de vivacité est un sous-ensemble de traces V tel que : *toute trace finie u peut-être prolongée en une trace v ($u \in \text{pref}(v)$) appartenant à V*. Intuitivement, la sûreté représente le fait qu'un mauvais comportement ne se produira jamais et la vivacité qu'un bon comportement se produira dans le futur. Les propriétés d'invariant font partie des propriétés de sûreté. Ces propriétés temporelles linéaires sont notamment représentables par des formules LTL. Notons aussi que toute propriété LTL est l'intersection d'une propriété de sûreté et d'une propriété de vivacité.

Cependant, ces types de propriétés ne caractérisent pas tous les comportements d'un système. En effet, les propriétés temporelles linéaires font référence à une unique trace à la fois et il est parfois nécessaire de raisonner sur l'arbre des exécutions possibles pour, notamment, spécifier des *propriétés d'accessibilité*. Celles-ci permettent

d'exprimer le fait qu'un système a la possibilité d'atteindre un état donné (de façon déterministe ou non) sans pour autant que cela soit obligatoire. La logique CTL, par exemple, permet de spécifier des propriétés d'accessibilité. En effet, l'opérateur E dans une formule CTL permet de quantifier existentiellement sur l'ensemble des branches d'exécution possibles. La formule $\mathrm{EG}\,\phi$, par exemple, requiert l'existence d'un chemin d'exécution où l'on finit par toujours satisfaire ϕ. On parle alors d'accessibilité ou d'atteignabilité.

1.5 Méthode B

La *méthode B* est une méthode de spécification formelle introduite dans les années 1980 par Abrial [3] et dont le but était de produire des outils sûrs de construction de logiciels. Elle se base sur les concepts de *machine abstraite* et de *raffinement*.

Une machine abstraite est caractérisée par une entête, la définition d'un état et la description des opérations pouvant transformer l'état. Ainsi, une machine abstraite peut être décrite par les clauses suivantes.

- MACHINE : le nom et paramètres de la machine ;
- CONSTRAINTS : la définition des propriétés des paramètres de la machine ;
- SETS : des ensembles abstraits et finis ;
- CONSTANTS : une liste des constantes de la machine ;
- PROPERTIES : les définitions des propriétés des constantes et des ensembles ;
- DEFINITIONS : une liste des abréviations pour les prédicats, les expressions ou les substitutions ;
- VARIABLES : des variables représentant l'état de la machine ;
- INVARIANT : une formule permettant de typer les variables et de définir les propriétés sur les variables ;
- INITIALISATION : les valeurs initiales des variables ;

- OPERATIONS : la définition des opérations.

Le raffinement en B permet de rendre la machine plus concrète en précisant les choix de réalisation tout en conservant la compatibilité avec la machine raffinée. La structure d'un raffinement est similaire à celle de la machine abstraite. La clause MACHINE est remplacée par la clause REFINEMENT suivie d'une clause REFINES. En outre, on ajoute à l'invariant de raffinement des prédicats de liaison entre les nouvelles variables et les variables de la machine raffinée.

Chaque machine abstraite B fournit des opérations agissant sur un état. La sémantique des opérations est définie par le *langage de substitutions généralisées*. En effet, des substitutions sont utilisées pour décrire les initialisations et le corps des opérations. Les principales substitutions sont les suivantes.

- SKIP : la substitution neutre, l'état reste inchangé ;
- $x := E$: la substitution simple, elle affecte la valeur E à la variable x ;
- $x, y := E, F$: la substitution multiple, elle décrit plusieurs substitutions simples en parallèle ;
- $S; T$: le séquencement, la substitution applique successivement S puis T ;
- PRE P THEN S END : la substitution préconditionnée, la substitution S s'exécute si la précondition P est satisfaite ; sinon rien ne peut être garanti ;
- SELECT P_1 THEN S_1 WHEN P_2 THEN S_2 ... ELSE S_d END : la substitution gardée, S_i ne peut être appliquée que dans le cas où l'état "pré" satisfait P_i ;
- VAR z IN S END : la substitution de choix non borné, elle applique la substitution S pour une valeur quelconque (non déterministe) de la variable z ;
- ANY a WHERE C THEN S END : la substitution la plus générale, si la variable a satisfait la condition C, alors la substitution S est exécutée.

1.6 Algèbres de processus

Les *algèbres de processus* sont une famille de langages formels permettant de modéliser des systèmes concurrents. Une algèbre de processus spécifie le comportement autorisé de chaque processus composant un système. Un processus est représenté par une *expression de processus* sous forme d'expression algébrique à l'aide d'opérateurs définis dans l'algèbre de processus qui s'appliquent à des actions. Cela permet de modéliser les interactions entre processus, de décrire les processus avec un ensemble restreint de primitives et de raisonner sur les expressions algébriques. Parmi les algèbres de processus les plus populaires, on notera *Communicating sequential processes* (CSP) [51], *Calculus of communicating systems* (CCS) [70] ou π-calcul [72].

1.6.1 *Communicating sequential processes* (CSP)

Le langage CSP a été initialement présenté par Hoare en 1978 afin de décrire un langage de programmation concurrente, puis a subi différentes améliorations lui procurant une syntaxe plus mathématique influencée par CCS. Une sémantique dénotationnelle de CSP, basée sur le concept de refus, a été présentée par Brookes, Hoare et Roscoe en 1984 [16]. Les principaux opérateurs de CSP sont définis par la syntaxe suivante :

$$
\begin{aligned}
P ::= \; & STOP \\
& \mid SKIP \\
& \mid a \rightarrow P \text{ (préfixe)} \\
& \mid (a \rightarrow P \mid a' \rightarrow P) \text{ (choix déterministe)} \\
& \mid P \,\Box\, P \text{ (choix externe)} \\
& \mid P \sqcap P \text{ (choix interne)} \\
& \mid P \,;\, P \text{ (séquence)} \\
& \mid P \parallel_X P \text{ (synchronisation)}
\end{aligned}
$$

où a et a' sont des actions.
$P \parallel Q := P \parallel_X Q$ (parallélisme), avec X l'ensemble des actions communes de P et Q
$P \,|||\, Q := P \parallel_\emptyset Q$ (entrelacement)

L'opérateur de *préfixe* permet de définir un processus $a \rightarrow P$ formé par une action a suivie d'une expression de processus P. Il explicite donc clairement le premier évènement d'une expression. Il existe plusieurs moyens de modéliser le choix en CSP : les opérateurs de *choix déterministe*, de *choix externe* et de *choix interne*. Le choix déterministe s'utilise lorsqu'il s'agit d'un choix du type $a \rightarrow P \mid b \rightarrow Q$. Le processus peut soit exécuter l'évènement a puis se comporter comme le processus P, soit exécuter l'évènement b et se comporter comme Q. Les choix interne et externe de type $P \sqcap Q$ et $P \,\Box\, Q$ signifient que le processus peut se comporter comme P ou comme Q. Le choix se détermine à la première action et ne change plus. On parle de choix externe lorsque celui-ci dépend d'un évènement extérieur tandis qu'il est dit interne s'il est non déterministe. La *séquence* $P\,;Q$ indique que le processus P est suivi du processus Q. L'opérateur de *synchronisation* permet la composition de deux processus $P \parallel_X Q$. Le processus résultant doit exécuter simultanément les évènements de P et de Q qui appartiennent à l'ensemble de synchronisation X. Les autres évènements peuvent être exécutés indépendamment dans l'ordre défini. On peut définir aussi à partir de cet opérateur les opérateurs de *parallélisme* et d'*entrelacement* où les ensembles de synchronisation sont respectivement les évènements communs des processus et l'ensemble vide. Enfin, le processus $STOP$ permet de mettre fin à un processus en terminant le programme, tandis que le processus $SKIP$ permet à un processus de terminer normalement. Une description plus complète des opérateurs CSP est donnée dans [51].

1.6.2 *Entity-based black-box specification* (EB^3)

La méthode *Entity-Based Black-Box Specification* (EB^3), proposée par Frappier et St-Denis [43], est une méthode de spécification formelle de systèmes d'information. Une telle spécification est composée de quatre parties :

- un *diagramme entité-association* : à l'instar des diagrammes de classes d'UML [84], le schéma représente les entités du système ainsi que les relations existant entre elles. On y indique les attributs, les méthodes et les cardinalités ;
- une *expression de processus* : elle s'inspire de CSP et décrit le cycle de vie d'une entité ou association ;

- un ensemble de *fonctions récursives* : elles permettent d'évaluer les attributs des entités et associations ;
- un ensemble de *règles d'entrée-sortie* : elles spécifient les sorties correspondant aux traces valides (entrées).

Comme pour CSP, une expression de processus en EB^3 décrit les contraintes d'ordonnancement des évènements. Les opérateurs de l'algèbre de processus sont donnés par la syntaxe :

$E ::= \boxdot$ (arrêt)
$\quad | \ \lambda$ (évènement interne)
$\quad | \ \mathsf{a}(\vec{t})$ (action)
$\quad | \ E \,.\, E$ (séquence)
$\quad | \ E \mid E$ (choix)
$\quad | \ E^*$ (fermeture de Kleene)
$\quad | \ E^+$ (fermeture positive)
$\quad | \ g \Longrightarrow E$ (garde)
$\quad | \ P(\vec{t})$ (appel de processus)
$\quad | \ E|[\Delta]| \, E$ (synchronisation)
$\quad | \ |\, x \in T \ : \ E$ (choix quantifié)
$\quad | \ ||[\Delta]|| \ x \in T \ : \ E$ (synchronisation quantifiée)

où a est une action, $\vec{t}$ est un vecteur de termes, g un prédicat, P un processus, Δ un ensemble d'actions. Une expression de processus élémentaire peut être le processus d'arrêt $\boxdot$, une action interne λ ou une action $\mathsf{a}(\vec{t})$ paramétrée par $\vec{t}$. Un processus peut alors être construit à partir des opérateurs usuels des langages réguliers (séquence, choix, fermetures itérative et positive) ainsi que par les opérateurs de garde, d'appel de processus et de synchronisation. Par ailleurs, EB^3 permet l'utilisation de quantificateurs sur les opérateurs de choix et de synchronisation. Il est ainsi possible, par exemple, d'effectuer un choix entre un ensemble d'actions déterminé par un ensemble de valeurs possibles de paramètres pour ces actions. La sémantique détaillée est décrite dans [43].

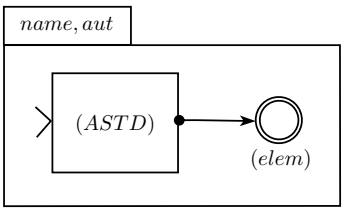

figure 1.1 – Exemple d'ASTD automate

1.7 Diagrammes états-transitions algébriques

Les *diagrammes états-transitions algébriques* (*Algebraic State-Transition Diagrams*, ASTD) [42] sont une notation graphique combinant des diagrammes états-transitions avec des opérateurs d'algèbres de processus tels que le choix ou la composition parallèle. Cette méthode s'inspire des automates, des *statecharts* [49] et de l'algèbre de processus EB^3. Un ASTD est similaire à une algèbre de processus dotée d'automates hiérarchiques. Sa structure s'exprime de façon inductive à l'aide d'un ensemble de types différents d'ASTD. Dans le cadre de cette thèse, nous nous restreignons à un fragment du modèle ASTD. Les types d'ASTD considérés sont les suivants.

1. L'*ASTD élémentaire* est un type de base qui sert à représenter les états de base d'un automate. Il ne peut contenir aucun autre type. Il est noté elem.

2. L'*ASTD automate*, représenté dans la figure 1.1, est un type d'ASTD qui est similaire aux automates classiques. Une structure de ce type est décrite par un automate dont les états peuvent être soit des ASTD élémentaires (états classiques), soit des états composites d'un autre type d'ASTD. Il est noté par un n-uplet $(\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)$, où $name$ est le nom de l'automate, Σ son alphabet, N l'ensemble des noms des états, ν une fonction qui associe à chaque $n \in N$ le sous-ASTD correspondant à l'état, δ une fonction de transition, $SF \subseteq N$ et $DF \subseteq N$ des sous-ensembles d'états finaux *shallow* et *deep* et $n_0 \in N$ un état initial. Un état final *shallow* indique que l'état est final quelle que soit la configuration du sous-état. Un état final *deep* est final seulement si son sous-état est aussi final.

3. L'*ASTD fermeture de Kleene* (figure 1.2) permet l'itération d'un ASTD un

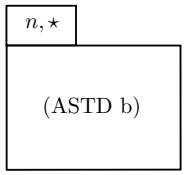

figure 1.2 – ASTD fermeture de Kleene

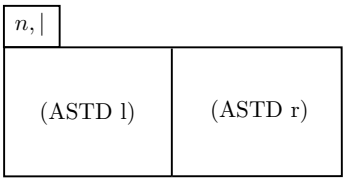

figure 1.3 – ASTD choix

nombre fini de fois (incluant zéro), une itération étant terminée lorsque le sous-ASTD atteint un état final. Il est noté $(\star, n, b)$, où n est le nom de l'ASTD et b le sous-ASTD.

4. L'*ASTD choix* $(|, n, l, r)$ permet de choisir entre deux sous-ASTD l et r. (figure 1.3)

5. L'*ASTD synchronisation* $(||, n, \Delta, l, r)$ décrit le comportement de deux sous-ASTD l et r s'exécutant en concurrence. Les actions de Δ s'exécutent simultanément tandis que les autres sont entrelacées. (figure 1.4)

6. L'*ASTD choix quantifié* $(|:, n, x, T, b)$ permet de choisir une valeur v dans l'ensemble T et d'exécuter le sous-ASTD b dans lequel la variable x est instanciée par la valeur v. (figure 1.5)

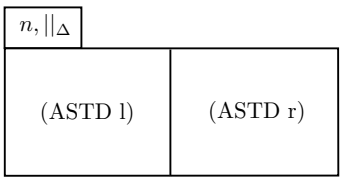

figure 1.4 – ASTD synchronisation

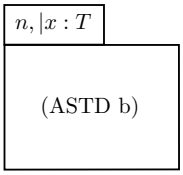

figure 1.5 – ASTD choix quantifié

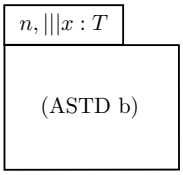

figure 1.6 – ASTD entrelacement quantifié

7. L'*ASTD entrelacement quantifié* $(|||:, n, x, T, b)$ permet de lancer autant d'instances de sous-ASTD b en entrelacement qu'il y a de valeurs dans T. Chaque instance s'exécute avec x instancié par la valeur correspondante. (figure 1.6)

La figure 1.7 montre un exemple d'ASTD. L'ASTD en haut et à gauche est un ASTD automate qui a pour nom *processA*. Il est constitué d'un état composé $S1$, qui est aussi un ASTD automate, et d'un état élémentaire $S2$. $S1$ est marqué comme initial grâce au symbole ">" et $S2$ comme final à l'aide du double cercle. Une transition étiquetée par une action b lie les deux états de l'automate. L'ASTD *processC* est un ASTD entrelacement quantifié qui instancie l'ASTD $processB(v)$ avec les valeurs 1 et 2. L'ASTD *main* synchronise *processA* et *processC*.

Chaque ASTD décrit un ensemble d'états. L'expression d'un état d'un ASTD est donnée suivant le type de l'ASTD correspondant. Il y a donc autant de types d'état que de types d'ASTD.

1. $(\mathsf{elem}_\circ)$, l'*état élémentaire*, qui est le seul état pour l'ASTD élémentaire ;

2. $(\mathsf{aut}_\circ, n, s)$, un *état automate*, où n est le nom de l'état et s un sous-état correspondant à l'état du sous-ASTD ;

3. $(\star_\circ, started?, [\bot \mid s])$, un *état fermeture de Kleene*, où *started*? est un booléen indiquant si la première itération a démarré, s est un sous-état et $\bot$ un symbole

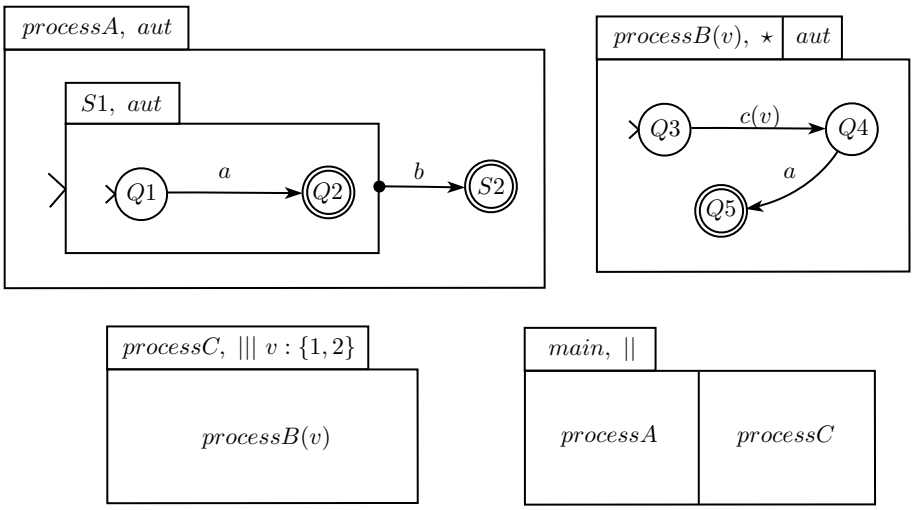

figure 1.7 – Exemple d'ASTD

signifiant l'absence de sous-état dans le cas où *started*? est faux ;

4. $(|_\circ, [\bot \mid \mathsf{left} \mid \mathsf{right}], [\bot \mid s])$, un *état choix*, avec une variable indiquant le choix (ASTD de droite ou de gauche) s'il a été fait et s un sous-état ;

5. $(||_\circ, s_l, s_r)$, un *état synchronisation*, où s_l et s_r sont des sous-états ;

6. $(|:_\circ, [\bot \mid v], [\bot \mid s])$, un *état choix quantifié*, où v est la valeur choisie et s un sous-état ;

7. $(|||:_\circ, f)$, un *état entrelacement quantifié*, où f est une fonction qui à chaque valeur du domaine associe un état d'ASTD.

Par exemple, un état de l'ASTD *processC* de la figure 1.7 est donné par l'expression $(|||:_\circ, \{1 \mapsto (\star_\circ, \text{true}, (\mathsf{aut}_\circ, Q4, (\mathsf{elem}_\circ))), 2 \mapsto (\star_\circ, \text{false}, \bot)\})$, il représente la configuration où $processB(1)$ est dans l'état $Q4$ et $processB(2)$ n'a pas démarré.

La sémantique opérationnelle pour chaque type d'ASTD est donnée en détail par des règles d'inférence dans [41] et dans l'annexe A (définition A.6).

Chapitre 2

Méthodes de vérification formelle

L'un des intérêts majeurs des spécifications formelles est de permettre de démontrer rigoureusement qu'un modèle respecte des propriétés données. Pour cela, il existe deux approches fondamentales : la *preuve de théorème* et l'*exploration de modèle* (ou *model checking*). D'une part, la première consiste, à partir d'un ensemble d'hypothèses issues du modèle, à déduire par inférence un théorème qui représente la propriété à démontrer. Cette procédure peut être effectuée manuellement ou être assistée par ordinateur. D'autre part, le *model checking* consiste en une exploration de l'espace d'états d'un système de transitions afin de s'assurer que la propriété donnée soit respectée à travers tout le système. Il s'agit donc de fournir un algorithme qui automatise complètement la tâche de vérification. Nous nous intéressons, dans ce chapitre, principalement au cas du *model checking*.

Comme montré dans le chapitre 1, un système dynamique peut être modélisé par un système de transitions, un modèle B, une algèbre de processus etc. En outre, les propriétés à vérifier sur le système sont modélisables par des formules de la logique temporelle, des invariants ou des sous-ensembles d'états interdits par exemple. Dans le cas de propriétés d'invariants ou d'accessibilité, un simple parcours de l'espace d'états peut être suffisant pour résoudre le problème de la vérification. Dans le cas de propriétés temporelles, il existe des procédures de *model checking* adaptées aux formules LTL et d'autres aux formules CTL, qui transforment le système avant de

le parcourir. Cependant, le principal problème du *model checking* est l'explosion de l'espace d'états. En effet, bien souvent les systèmes à vérifier comportent déjà un grand nombre d'états à parcourir, et la vérification de propriétés temporelles augmente d'autant plus la complexité de la vérification. Plusieurs méthodes existent pour réduire le temps de vérification comme des techniques de réduction de l'espace d'états, de représentation symbolique ou encore d'abstraction. Par ailleurs, certaines techniques permettent aussi de vérifier, sous certaines conditions, des systèmes dont l'espace d'états est infini. Notamment, les méthodes de *vérification paramétrée* prennent en charge une classe de systèmes infinis qui possèdent des caractéristiques permettant parfois une factorisation de l'espace d'états ou un raisonnement inductif.

Le chapitre est organisé comme suit. La section 2.1 présente les algorithmes de *model checking* pour LTL et CTL, ainsi que certaines techniques de base permettant d'aborder le problème de l'explosion de l'espace d'états. La section 2.2 se concentre sur le problème de la vérification de systèmes paramétrés et présente certaines des solutions existantes pour différents modèles. Finalement, la section 2.3 détaille la technique de vérification des *systèmes de transitions bien structurés* sur laquelle est basée l'étude du chapitre 4.

2.1 Techniques de *model checking*

2.1.1 *Model checking* LTL

Le problème du *model checking* de formule LTL peut être formulé de la façon suivante : "Soit ϕ une formule LTL sur un alphabet Σ et $M = (Q, \Sigma, T, I, F)$ un automate de Büchi. L'automate M satisfait-il la formule ϕ ? (noté $M \models \phi$) "

La procédure de vérification LTL, proposée par Vardi et Wolper en 1986 [92], est basée sur la théorie des langages et des automates. En effet, toute formule LTL ϕ peut être représentée par un automate de Büchi A_ϕ [93]. Ainsi, si l'on spécifie le modèle, de la même façon, par un automate M, il suffit de comparer les langages de chaque automate $\mathcal{L}(M)$ et $\mathcal{L}(A_\phi)$. On dit qu'une formule ϕ est satisfaite par un automate M si et seulement si le langage $\mathcal{L}(M)$ de l'automate est inclus dans le langage $\mathcal{L}(A_\phi)$. D'après la théorie des ensembles, on sait que cela est équivalent à vérifier si $\mathcal{L}(M) \cap$

$\overline{\mathcal{L}(A_\phi)}$ est vide. De plus, le complément du langage de ϕ peut être calculé par $\overline{\mathcal{L}(A_\phi)} = \mathcal{L}(A_{\neg\phi})$ et la théorie des automates permet de construire l'intersection de langages grâce au produit synchrone d'automates, noté $\otimes$. Le problème du *model checking* LTL consiste donc à vérifier si le langage d'un automate est vide, d'après les étapes suivantes :

$$\begin{aligned} M \models \phi \quad & \textbf{ssi} \quad \mathcal{L}(M) \subseteq \mathcal{L}(A_\phi) \\ & \textbf{ssi} \quad \mathcal{L}(M) \cap \mathcal{L}(A_{\neg\phi}) = \emptyset \\ & \textbf{ssi} \quad \mathcal{L}(M \otimes A_{\neg\phi}) = \emptyset \end{aligned}$$

L'algorithme de vérification consiste donc en les phases suivantes. À partir de ϕ, on construit un automate de Büchi $A_{\neg\phi}$ représentant le langage de la négation de la formule à vérifier, *i.e.* le complément du langage de ϕ. Ensuite, on effectue le produit synchrone du modèle M et de l'automate $A_{\neg\phi}$ noté $M \otimes A_{\neg\phi}$. Enfin, on teste la vacuité du langage de l'automate produit.

Il existe plusieurs techniques permettant de construire un automate de Büchi à partir d'une formule LTL [93, 91, 27]. Les algorithmes sont connus pour avoir une complexité exponentielle en temps et en espace dans la taille de la formule, soit $2^{\mathcal{O}(|\phi|)}$. Le produit synchrone d'automates et le test de vacuité sont effectués par les algorithmes bien connus sur les automates [92]. Il suffit de trouver une exécution qui visite infiniment souvent un état final dans le produit pour prouver que son langage n'est pas vide. Pour cela, on cherche à déterminer s'il existe un cycle acceptant, *i.e.* qui contient un état final, atteignable par une recherche en profondeur. Les algorithmes les plus efficaces sont linéaires. Le problème du *model checking* de formule LTL a ainsi été démontré comme étant PSPACE-complet [88]. Plus précisément, la complexité temporelle est en $\mathcal{O}(|M| \times 2^{|\phi|})$.

2.1.2 *Model checking* CTL

Le problème de la vérification CTL est le suivant : "Soit ϕ une formule CTL et $M = (S, T, I, AP, l)$ une structure de Kripke. Le modèle M satisfait-il la formule ϕ ?"

Les premiers algorithmes ont été présentés dans [21, 80]. Contrairement aux formules LTL, les formules CTL sont des formules d'état, *i.e.* évaluables sur un état. Il est

alors possible d'étiqueter chaque état par un ensemble de formules CTL qu'il vérifie. L'approche de vérification consiste à déterminer pour chaque état les sous-formules CTL de ϕ satisfaites à l'aide d'une procédure récursive. L'algorithme de vérification agit sur la structure de Kripke du système en ajoutant chaque sous-formule de ϕ à l'ensemble des propositions atomiques AP et en modifiant la fonction d'étiquetage des états l. Ainsi, $M \models \phi$ si et seulement si $\phi \in l(I)$. On note $Sat(\psi)$ l'ensemble des états de S tel que pour tout $s \in Sat(\psi)$, $s \models \psi$. On a alors $M \models \phi$ si et seulement si $I \subseteq Sat(\phi)$.

On raisonne par induction sur la structure de la formule ϕ. Pour chaque sous-formule de ϕ, l'algorithme de vérification recherche l'ensemble des états corrects par un calcul de plus grand (ν) ou plus petit (μ) point fixe caractérisé par des formules de μ-calcul telles que :

$$\mathrm{EF}\, p = \mu Y.(p \vee \mathrm{EX}\, Y)$$

$$\mathrm{EG}\, p = \nu Y.(p \wedge \mathrm{EX}\, Y)$$

$$\mathrm{E}(q\, \mathrm{U}\, p) = \mu Y.(p \vee (q \wedge \mathrm{EX}\, Y))$$

L'algorithme commence par les sous-formules terminales de ϕ puis remonte dans sa structure de façon à toujours pouvoir utiliser une caractérisation de point fixe sur une formule sans opérateurs temporels imbriqués. Finalement, le modèle vérifie la formule si et seulement si les états initiaux du modèle sont étiquetés par ϕ à la fin de l'algorithme. Les algorithmes connus de calcul de point fixe, pour chacun des opérateurs temporels, sont linéaires. Ainsi, la complexité de l'algorithme de calcul de $Sat(\phi)$ est en $\mathcal{O}(|M| \times |\phi|)$. Le problème du *model checking* de formules CTL est connu pour être P-complet [22]. Notons que cette différence avec la complexité du problème de vérification LTL s'explique en partie par le fait que la représentation d'une propriété en LTL est en général exponentiellement plus succincte que sa représentation en CTL, lorsque la propriété est spécifiable dans chacune de ces deux logiques.

2.1.3 Problème de l'explosion combinatoire d'états

Dans la pratique, le *model checking* est souvent appliqué à des modèles de très grande taille. La taille d'un système correspond au nombre d'états et de transitions

qu'il comporte, le nombre de transitions étant lui-même lié au nombre d'états. Par exemple, si l'on considère un programme comme un système dynamique, le nombre d'états du système augmente exponentiellement avec le nombre de variables du programme. On parle alors d'*explosion combinatoire d'états*. Représenter un programme par un système de transition d'états peut donc mener à construire des objets de très grande taille. Ce qui se révèle souvent être coûteux aussi bien en espace qu'en temps de calcul. En effet, les sections précédentes montrent que la complexité des algorithmes de vérification dépend des tailles respectives du système et de la propriété. Il est d'ailleurs parfois nécessaire de calculer le résultat de la synchronisation de structures (*e.g.* produit d'automates), ce qui augmente d'autant plus l'espace d'états final.

Plusieurs techniques ont été développées pour tenter de pallier ce problème. Certaines réduisent l'espace d'états en montrant qu'il n'est pas toujours nécessaire de considérer le système dans son intégralité. D'autres proposent différentes représentations des données afin de compresser l'espace d'états. Les sections suivantes décrivent les principales techniques abordant le problème de l'explosion d'états suivant différents angles d'attaque.

2.1.4 Techniques de réduction

La réduction de l'espace d'états est une approche qui consiste à ne construire ou ne considérer qu'une partie du système complet pour la vérification en factorisant l'espace d'états ou en optimisant les algorithmes de parcours. Cette section présente deux techniques de réduction : la vérification à la volée et la réduction d'ordre partiel.

Vérification à la volée

Les techniques de *vérification à la volée* [25] permettent de réduire le temps de vérification en pratique. En effet, les algorithmes à la volée ont pour principe de construire l'espace d'états tout en le parcourant. Cela permet de trouver une solution plus rapidement sans avoir à construire le système en entier.

L'outil SPIN [52] implémente par exemple un algorithme permettant de tester le vide de l'intersection de langages en alternant les phases de construction et de recherche. La procédure de vérification de formules LTL a été décrite précédemment :

un modèle M vérifie une formule ϕ si et seulement s'il existe un cycle acceptant dans le produit des automates de M et de $\neg\phi$. Dans ce cas, l'idée de la vérification à la volée consiste à générer l'automate pour $\neg\phi$ tout en parcourant en parallèle les états accessibles du système. On considère un successeur dans l'automate de $\neg\phi$ uniquement s'il satisfait le système M et l'algorithme s'arrête dès que le cycle acceptant est trouvé. L'algorithme peut alors trouver un cycle acceptant sans construire l'automate de $\neg\phi$ entièrement.

Réduction d'ordre partiel

La technique de *réduction d'ordre partiel* [90, 46, 75] permet d'exploiter la commutativité des transitions concurrentes d'un modèle afin de réduire l'espace d'états parcouru. Pour les systèmes dans lesquels l'ordre d'arrivée de certains évènements importe peu, on peut considérer des chemins équivalents.

L'approche de la réduction d'ordre partiel s'appuie sur la notion d'indépendance des actions [64, 66, 56] définie de la façon suivante. Soit $S = (Q, \Sigma, T, I, AP, l)$ un système de transition d'états déterministe. Notons $Act(q) = \{\alpha \in \Sigma \mid \exists q' \cdot (q, \alpha, q') \in T\}$ et pour tout $\alpha \in Act(q)$, $\alpha(q)$ le successeur de q. Considérons deux actions α, $\beta \in \Sigma$, α et β sont dits indépendants si et seulement si pour tout $q \in Q$ tel que $\alpha, \beta \in Act(q)$, on a $\beta \in Act(\alpha(q))$ et $\alpha \in Act(\beta(q))$ et $\alpha(\beta(q)) = \beta(\alpha(q))$. Intuitivement, deux actions sont indépendantes lorsqu'elles agissent sur des variables différentes. Ainsi, l'ordre dans lequel les actions sont exécutées n'a pas d'influence sur le résultat final.

Katz et Peled définissent dans [56] une approche à base d'*ample sets* pour vérifier des propriétés exprimées en $\mathrm{LTL}_{\setminus X}$ (LTL sans l'opérateur *Next*). Un *ample set* $ample(q)$ pour un état q est un sous-ensemble d'actions de $Act(q)$. L'idée est de ne prendre en compte qu'un certain nombre d'actions possibles en éliminant les autres. Cela permet ainsi de générer un système de transitions plus petit. Les actions de $ample(q)$ sont indépendantes de toutes les actions de $Act(q) \setminus ample(q)$ et dans tout fragment d'exécution partant de q, toute action de $ample(q)$ reste possible tant qu'aucune action de $ample(q)$ n'a été exécutée. De plus, si $ample(q)$ ne représente pas l'ensemble de toutes les actions d'origine, alors toute action $\alpha \in ample(q)$ est une action dite *invisible (stutter)*, *i.e.* telle que $l(q) = l(\alpha(q))$. Un *ample set* est défini

plus formellement par une série de conditions dans [56]. Cette approche permet de déduire une équivalence par *stutter trace* entre le système original et celui induit par la notion d'*ample set*. Ce qui mène à une équivalence en $\mathrm{LTL}_{\setminus X}$. La vérification de propriétés $\mathrm{LTL}_{\setminus X}$ peut ainsi s'effectuer sur le système réduit équivalent.

Cette approche a été adaptée à la logique $\mathrm{CTL}^*_{\setminus X}$ [44] en ajoutant à la définition des *ample sets* la condition que si $ample(q) \neq Act(q)$ alors $|ample(q)| = 1$. Les *ample sets* sont équivalents aux *persistent sets* [46] et aux *stubborn sets* [90] développés indépendamment dans le cadre de la réduction d'ordre partiel.

2.1.5 Techniques symboliques

La représentation des données joue un rôle important dans la vérification. En effet, outre la complexité temporelle, le problème du *model checking* doit aussi faire face à la gestion de la mémoire. Ainsi, les *méthodes symboliques* permettent de compresser l'espace d'états en adoptant des structures de données différentes. On parle de techniques *symboliques* lorsque les données sont représentées de façon implicite. Par exemple, une structure de graphe peut être encodée par des formules de la logique.

***Model checking* symbolique**

McMillan propose en 1987 [67] la première technique de *model checking* symbolique pour CTL. Il s'agit de représenter l'espace d'états ainsi que les transitions sous forme de *formules booléennes quantifiées* (*quantified Boolean formulas*, QBF) implémentées par des *diagrammes de décision binaires ordonnés et réduits* (*Reduced Ordered Binary Decision Diagram*, ROBDD) [62, 17]. En effet, une QBF peut par exemple représenter un ensemble d'états. Cela permet un gain d'espace très important et donc de manipuler des systèmes beaucoup plus grands. La technique peut aussi être appliquée aux formules LTL en passant par une transformation préalable du système et de la propriété [23].

La logique des formules booléennes quantifiées est une extension de la logique propositionnelle qui permet de quantifier sur des variables propositionnelles. Ainsi, si V est un ensemble de variables propositionnelles, $QBF(V)$ est le plus petit ensemble tel que :

- $\top$ (vrai) et $\bot$ (faux) sont des formules ;
- pour tout $v \in V$, v est une formule ;
- si p et q sont des formules alors $p \wedge q$ et $\neg p$ le sont aussi ;
- si p est une formule et $v \in V$, alors $\exists v.p$ est une formule.

Dans la suite, on note $z(x \leftarrow y)$ la substitution de x par y dans z. Une affectation de vérité a est une fonction $V \to \{\bot, \top\}$. On considère que chaque QBF correspond à un ensemble de valeurs d'affectation de vérité satisfaisant la formule. $\top$ représente l'ensemble de toutes les affectations de vérité, $\bot$ l'ensemble vide et une variable propositionnelle v l'ensemble des affectations de vérité a tel que $a(v) = \top$. De plus,

- $a \in (p \vee q)$ si et seulement si $a \in p$ ou $a \in q$;
- $a \in \neg p$ si et seulement si $a \notin p$;
- $a \in \exists v.p$ si et seulement si $a(v \leftarrow \top) \in p$ ou $a(v \leftarrow \bot) \in p$.

Il est possible de représenter une structure de Kripke par un ensemble de QBF. Un état du système peut être vu comme une affectation de vérité pour un ensemble de variables propositionnelles $V = \{v_1, \ldots, v_n\}$. Toute QBF représente donc un ensemble d'états. Afin de représenter une relation de transition entre les états, on introduit deux ensembles ordonnés de variables $V = \{v_1, \ldots, v_n\}$ et $V' = \{v'_1, \ldots, v'_n\}$. La relation de transition T est caractérisée par une formule construite sur les valeurs courantes des variables V et leurs prochaines valeurs V'. La fonction de valuation de la structure de Kripke donne la valeur de vérité de chaque variable v_i dans chaque état s.

Les propriétés exprimées en CTL peuvent de même être transformées dans la logique QBF de la façon suivante. Soit $V = \{v_1, \ldots, v_n\}$ et $V' = \{v'_1, \ldots, v'_n\}$ deux ensembles de variables disjoints, T la QBF de la relation de transition sur $V \cup V'$ et s un état alors pour tout p, q formules CTL :

- $s \models p$ *ssi* $s \in p$, où $p \in V$;
- $s \models p \vee q$ *ssi* $s \in (p \vee q)$;
- $s \models \neg p$ *ssi* $s \in (\neg p)$;

- $s \models \mathrm{EX}\, p$ *ssi* $s \in (\exists V'.(T \wedge p(V \leftarrow V')))$;
- $s \models \mathrm{E}\ (q\, \mathrm{U}\, p)$ *ssi* $s \in \mu Y.(p \vee (q \wedge \mathrm{EX}\, Y))$;
- $s \models \mathrm{EG}\, p$ *ssi* $s \in \nu Y.(p \wedge \mathrm{EX}\, Y)$.

Il est donc possible de résoudre le problème du model checking CTL en manipulant les QBF sans construire explicitement la structure de Kripke. En effet, pour déterminer si une formule CTL est satisfaite par un système, il suffit d'évaluer la formule QBF correspondant à la formule CTL.

Le *model checking* symbolique utilise par ailleurs la théorie des *diagrammes de décision binaires ordonnés (OBDD)* afin d'implémenter les formules QBF. En effet, les OBDD permettent de manipuler les formules booléennes et implémentent les opérations $p \wedge q$, $p \vee q$, $\neg p$, $\exists V'.p$ et $p(V \leftarrow V')$ de façon efficace. Malgré le gain en espace et les possibles performances du point de vue technique, la complexité théorique du problème est de la classe PSPACE-complet. Cela est dû notamment à la phase d'encodage des structures de données.

Model checking borné

L'autre approche symbolique majeure est le *model checking borné* [12]. Elle est inspirée de la technique de *model checking* symbolique précédente. Initialement conçue pour la vérification LTL, la méthode consiste à représenter de façon symbolique des traces bornées d'exécutions par des formules de la logique propositionnelle. Le problème de la vérification bornée se réduit alors au *problème de satisfaisabilité de formule propositionnelle (SAT)* qui consiste à chercher un modèle (ici une trace) satisfaisant une formule propositionnelle (la spécification).

Considérant ϕ une formule LTL, $M = (Q, T, I, AP, l)$ une structure de Kripke et k un entier, le problème du *model checking borné* est de déterminer si le modèle M satisfait la formule ϕ pour des traces de longueur bornée par k, noté $M \models_k \phi$. Pour cela, on construit une formule propositionnelle $[\![M, \phi]\!]_k$ modélisant une contrainte sur des variables $s_0, \ldots, s_k$, où pour tout $i \in 0..k$, s_i est une vecteur de variables d'états représentant un état du système et la suite $s_0, \ldots, s_k$ représente un chemin dans M. La formule propositionnelle est composée de la conjonction de deux formules : l'une

représentant le système et l'autre la propriété $[\![M, \phi]\!]_k := [\![M]\!]_k \wedge [\![\phi]\!]_k$. La première établit que $s_0, \ldots, s_k$ est une exécution de longueur k du système : $[\![M]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$, le prédicat I représente les états initiaux et le prédicat T la relation de transition. La formule propositionnelle de la propriété $[\![\phi]\!]_k$ est calculée de façon récursive sur la structure de la formule LTL. En effet, chaque opérateur LTL possède une transformation correspondante [12]. Il est à noter que cette transformation n'est pas la même si l'on considère que $[\![M]\!]_k$ est définie sur une boucle ou non. Il faut alors analyser tous les cas possibles de boucle. La formule pour ϕ s'écrit donc $[\![\phi]\!]_k := (\neg L_k \wedge [\![\phi]\!]_k^{sb}) \vee \bigvee_{l=0}^{k} (L_{k,l} \wedge [\![\phi]\!]_k^{b})$, où $L_{k,l}$ est le prédicat indiquant une transition entre k et l, $L_k := \bigvee_{l=0}^{k} L_{k,l}$ une boucle de k à l, $[\![\phi]\!]_k^{sb}$ le cas de l'exécution sans boucle et $[\![\phi]\!]_k^{b}$ celui avec boucle.

Afin de déterminer si $M \models_k \phi$ il convient de vérifier si $[\![M, \phi]\!]_k$ est satisfaisable. En effet, $M \models_k \neg\phi$ si et seulement si $[\![M, \phi]\!]_k$ est insatisfaisable. Ce problème peut alors être résolu par les algorithmes SAT décidant la satisfaisabilité d'une formule propositionnelle. En généralisant, $M \models \neg\phi$ si et seulement si $[\![M, \phi]\!]_k$ est satisfaisable pour tout $k \in \mathbb{N}$. A priori, la résolution de problème du *model checking* non borné, vu sous cet angle, nécessiterait de tester les traces de toutes les longueurs possibles. Cependant, ce problème est connu comme étant PSPACE-complet. En effet, il a été montré qu'il existe une borne $k \leq |M| \times 2^{|\phi|}$, appelée *diamètre de récurrence*, telle que $M \models_k \phi$ si et seulement si $M \models \phi$. Ce qui démontre la complétude de cette méthode.

2.1.6 Techniques d'abstraction

Le problème de la vérification de systèmes infinis est, dans le cas général, indécidable. Pour permettre le parcours de l'espace d'états de ce genre de systèmes, il est nécessaire de modifier la représentation du modèle et de le rendre fini grâce, entre autres, aux *méthodes d'abstraction*. Ces techniques peuvent aussi être utiles pour les systèmes de tailles arbitrairement grandes, notamment dans le cas de la *vérification paramétrée* que l'on présentera dans la section 2.2. L'abstraction a pour principe de modéliser un système en l'approximant. On fait ainsi abstraction de certaines données. Il s'agit d'éliminer les détails peu pertinents pour la propriété à vérifier. L'espace d'états est réduit vers un ensemble d'états plus petit. Pour cela, on considère des fonc-

tions d'abstraction qui font correspondre des ensembles d'états concrets à des états abstraits. Il existe deux façons d'abstraire un système : une *sur-approximation* ajoute des comportements au système alors qu'une *sous-approximation* en retire.

De même, on classe l'abstraction des transitions en deux catégories. L'*abstraction existentielle* introduit une transition à partir d'un état abstrait si au moins un état concret correspondant possède la transition. Il s'agit d'une sur-approximation. Au contraire, l'*abstraction universelle* est une sous-approximation. Elle crée une transition à partir d'un état abstrait si tous les états concrets correspondant possèdent la transition. Dans la suite, on considérera essentiellement le cas de l'abstraction existentielle.

Si l'on considère une formule ACTL* ϕ et un système M existentiellement abstrait par M', alors on obtient le résultat suivant : $M' \models \phi \implies M \models \phi$, *i.e.* si le système abstrait vérifie la propriété alors le système original la satisfait aussi. Pour montrer que le système satisfait une propriété, il suffit donc de montrer qu'une sur-approximation satisfait cette même propriété. Cependant, si $M' \not\models \phi$ alors on ne peut pas conclure que $M \not\models \phi$. En effet, il peut exister des faux contre-exemples lorsque l'on considère le système abstrait puisque celui-ci contient plus de comportements que le système d'origine. Il est alors nécessaire de modifier le modèle abstrait par raffinement et de ré-appliquer la procédure de vérification. Clarke *et al.* proposent dans [19] une méthode de raffinement appelée *Raffinement d'Abstraction Guidé par Contre-exemple* (*Counterexample guided abstraction refinement*, CEGAR) afin d'automatiser le processus de vérification par abstraction. La technique commence avec une première abstraction et la raffine au fur et à mesure. Lorsqu'un contre-exemple est trouvé, s'il correspond à un contre-exemple réalisable dans le système concret, alors le processus de vérification termine en retournant le contre-exemple. Dans le cas contraire, une preuve de la non réalisabilité est utilisée pour raffiner l'abstraction. Puis, un autre contre-exemple est recherché.

Interprétation abstraite

L'*interprétation abstraite* est une théorie d'approximation de la sémantique des langages de programmation ou de spécification. Elle a été formalisée par Cousot et Cousot dans [26] en 1974. Pour spécifier rigoureusement un programme ou un sys-

tème, on utilise une sémantique formelle. Suivant le problème à traiter, il est possible de considérer le système selon différents niveaux d'abstraction et donc différentes sémantiques. Lorsqu'une sémantique en approxime une autre, on dit alors qu'elle est un modèle abstrait de cette dernière. L'interprétation abstraite est une théorie qui formalise la relation entre une *sémantique concrète* et une *sémantique abstraite*.

Une interprétation abstraite définit un modèle abstrait sur un domaine partiellement ordonné $(A, \preceq)$ à partir d'un modèle concret lui aussi de domaine partiellement ordonné $(C, \sqsubseteq)$, d'une *fonction d'abstraction* $\alpha : C \to A$ et d'une *fonction de concrétisation* $\gamma : A \to C$. Elle doit satisfaire plusieurs conditions. Premièrement, les fonctions d'abstraction et de concrétisation doivent être monotones. Deuxièmement, la sémantique concrète que l'on peut reconstruire à partir de la sémantique abstraite doit être plus grande que la sémantique concrète initiale. Finalement, la sémantique concrète ne doit perdre aucune information sur la sémantique abstraite. Intuitivement, toute propriété vérifiée par le modèle abstrait doit l'être aussi par le modèle concret. Ces conditions sont caractéristiques des *correspondances de Galois* à la base de la théorie de l'interprétation abstraite. Formellement, $(\alpha : C \to A, \gamma : A \to C)$ forme une correspondance de Galois de $(C, \sqsubseteq)$ vers $(A, \preceq)$ si :

- α et γ sont totales et monotones, et
- pour tout $c \in C$, $c \sqsubseteq \gamma \circ \alpha(c)$, et
- pour tout $a \in A$, $\alpha \circ \gamma(a) \preceq a$.

Parmi les domaines d'application de la théorie de l'interprétation abstraite, on retrouve notamment le *model checking*. En effet, elle a inspiré bon nombre de techniques d'abstraction. Considérons un système concret $S_C = (Q_C, T_C, I_C, AP_C, l_C)$ pour lequel on voudrait construire une abstraction $S_A = (Q_A, T_A, I_A, AP_A, l_A)$. Dans ce cas, en reprenant les notations précédentes, $(\alpha : C \to A, \gamma : A \to C)$ formant une correspondance de Galois, le domaine de la fonction d'abstraction $C = \mathcal{P}(Q_C)$ peut représenter l'ensemble des parties des états du système concret et $A = Q_A$ un treillis d'états abstraits. Ainsi, α et γ permettent de passer des états concrets aux états abstraits et inversement. La fonction d'abstraction associe à chaque ensemble d'états concrets un état abstrait et la fonction de concrétisation associe à chaque état

abstrait l'ensemble des états concrets auxquels il correspond. On dit que S_A est une *abstraction conservatrice* de S_C si :

- $I_C \subseteq \gamma(I_A)$, et
- pour tout $q_A \in Q_A$, $T_C(\gamma(q_A)) \subseteq \gamma(T_A(q_A))$, où $T(Q)$ dénote les successeurs de Q par la relation de transition T.

Ces conditions expriment le fait que l'état initial abstrait représente au moins tous les états initiaux concrets et que le successeur de chaque état abstrait q_A par la relation de transition abstraite représente tous les successeurs de l'ensemble des états concrets représentés par q_A par la relation de transition concrète. Ainsi, chaque chemin d'exécution concret est représenté par au moins un chemin abstrait. On peut en conclure que pour toute propriété de la logique ACTL* vérifiée par le système abstrait, sa version concrète l'est aussi dans le système concret. Il s'agit d'une sur-approximation.

Abstraction par prédicats

L'*abstraction par prédicats* (*predicate abstraction*), proposée par Graf et Saïdi dans [47], est un schéma d'abstraction conservatrice basé sur l'interprétation abstraite. Le principe est de regrouper des états concrets satisfaisant les mêmes prédicats en état abstrait et d'abstraire existentiellement la relation de transition.

Considérons un système concret $S_C = (Q_C, T_C, I_C, AP_C, l_C)$ et une suite de prédicats $Pred = (\phi_1, \phi_2, \ldots, \phi_n)$ sur ce système concret. Un prédicat ϕ_i représente ici un ensemble d'états satisfaisant une formule propositionnelle f_i sur AP_C et pour tout $q_C \in Q_C$, $\phi_i(q_C)$ est vrai si et seulement si $q_C \models f_i$. Chaque état abstrait $q_A \in Q_A$ est défini par un vecteur de n booléens, chacun représentant une valeur de vérité d'un prédicat. Pour tout $q_A \in Q_A$, on note $\phi_i(q_A)$ la valeur de la i-ème variable booléenne de q_A. Par exemple, l'état abstrait $(\top, \top, \bot)$ représente l'ensemble des états concrets satisfaisant $\phi_1 \wedge \phi_2 \wedge \neg\phi_3$. Formellement, la fonction de concrétisation $\gamma : Q_A \to \mathcal{P}(Q_C)$ se définit par $\forall q_A \in Q_A,\ \gamma(q_A) = \{q_C \in Q_C \mid \forall i \in 1..n \cdot \phi_i(q_A) \iff \phi_i(q_C)\}$ et la fonction d'abstraction $\alpha : \mathcal{P}(Q_C) \to Q_A$ par la fonction inverse. L'ensemble des états abstraits initiaux I_A est défini grâce à la fonction d'abstraction $I_A = \alpha(I_C)$ et on a

$I_C \subseteq \gamma(I_A)$. Par ailleurs, la relation de transition du système abstrait T_A est définie à partir de celle du système concret T_C et de la fonction de concrétisation γ tel que $\forall s,t \in Q_A \cdot (s,t) \in T_A \iff \exists x,y \in Q_C \cdot x \in \gamma(s) \wedge y \in \gamma(t) \wedge (x,y) \in T_C$. Cette définition de T_A correspond à une abstraction existentielle et vérifie la condition $\forall q_A \in Q_A,\ T_C(\gamma(q_A)) \subseteq \gamma(T_A(q_A))$. L'abstraction par prédicat est donc une abstraction conservatrice.

Abstraction finitaire des données

L'*abstraction finitaire des données (finitary data abstraction)* a été proposée par Kesten *et al.* dans [57] et s'inspire de l'interprétation abstraite. Elle a pour principe d'abstraire un système concret $\mathcal{D}$ potentiellement infini ainsi qu'une formule ψ à vérifier sur ce système en un système abstrait fini $\mathcal{D}^\alpha$ et la formule abstraite adéquate ψ^α, considérant un schéma d'abstraction α. Un système concret ou abstrait est donné par un *système discret* $\mathcal{D} = (V, \Theta, \rho)$ où :

- V est un ensemble fini de variables typées ;
- Θ est la condition initiale, une formule d'état caractérisant l'ensemble des états initiaux ;
- ρ est la relation de transition, une assertion $\rho(V, V')$ liant les valeurs des états courants aux valeurs des prochains.

Un schéma d'abstraction α est donné par un ensemble de variables abstraites V_A et un ensemble d'expressions $\mathcal{E}^\alpha$ tels que $V_A = \mathcal{E}^\alpha(V)$. Posons $p(V)$ une assertion, on définit alors les opérateurs suivants :

- $\alpha^+(p) : \exists V(V_A = \mathcal{E}^\alpha(V) \wedge p(V))$ dénote l'ensemble des états abstraits dont il existe un état concret correspondant qui vérifie p ;
- $\alpha^-(p) : \exists V(V_A = \mathcal{E}^\alpha(V)) \wedge \forall V(V_A = \mathcal{E}^\alpha(V) \implies p(V))$ dénote l'ensemble des états abstraits dont tous les états concrets correspondant vérifient p.

Ces opérateurs peuvent être généralisés aux formules LTL. Pour abstraire une formule ψ, on applique l'opérateur α^- aux formules propositionnelles qu'elle contient comme

montré dans [57] et on note ψ^α la formule LTL résultante. Remarquons que si l'on peut vérifier ψ^α alors ψ est aussi vérifiée. Le système abstrait correspondant à $\mathcal{D}$ est $\mathcal{D}^\alpha = (V^\alpha, \Theta^\alpha, \rho^\alpha)$ où :

- $V^\alpha = V_A$;
- $\Theta^\alpha = \alpha^+(\Theta)$;
- $\rho^\alpha = \exists V, V'(V_A = \mathcal{E}^\alpha(V) \wedge V'_A = \mathcal{E}^\alpha(V') \wedge \rho(V, V'))$.

La méthode d'abstraction a été montrée comme étant conservatrice, *i.e.* si $\mathcal{D}^\alpha \models \psi^\alpha$ alors $\mathcal{D} \models \psi$. L'abstraction finitaire des données définit un cadre formel général d'abstraction conservatrice qui peut être utilisé pour certains schémas d'abstraction.

2.2 Techniques de vérification paramétrée

Les *systèmes paramétrés* sont une classe particulière de systèmes infinis. Cette classe de systèmes est très fréquente dans le domaine du logiciel, notamment celui des systèmes distribués. Intuitivement, un système paramétré est composé d'un nombre n de processus identiques évoluant en parallèle, où n est considéré comme le paramètre du système. Ainsi, il représente un ensemble (infini) de systèmes instanciés chacun par un entier différent. Considérant $S(n)$ un système paramétré et ϕ une propriété, le problème de la vérification paramétrée consiste à déterminer si pour tout $n \in \mathbb{N}_1$, $S(n) \models \phi$. Un système peut aussi posséder plusieurs paramètres lorsqu'il est constitué de plusieurs classes de processus. On parle parfois de système multiparamétré.

Les systèmes paramétrés pouvant être considérés comme des systèmes infinis, le problème de leur vérification est indécidable dans le cas général, comme montré par Apt et Kozen dans [6]. Cependant, leur structure particulière permet d'appliquer plus facilement certaines méthodes de vérification et dans certains cas, le problème devient décidable. Il existe plusieurs approches au problème de la vérification paramétrée. Le principe général est de limiter la taille du système en le réduisant, par abstraction notamment. Une classe de méthodes s'appuie sur la théorie de l'*interprétation abstraite* [26] en utilisant des fonctions d'abstraction sur les systèmes. Une deuxième catégorie

de méthodes se base sur la recherche de limites, appelées *cutoffs*. En effet, il est possible d'obtenir une limite pour la valeur de paramètre telle que le système vérifie la propriété pour toute valeur de paramètre si et seulement s'il vérifie la propriété pour toute valeur inférieure à la limite. Enfin, certaines techniques reposent sur le principe d'induction pour vérifier les systèmes paramétrés. Après avoir établi qu'un certain invariant est suffisant pour prouver une propriété donnée, on montre par induction que l'invariant est satisfait pour toute valeur de n.

2.2.1 Techniques d'abstraction

Abstraction par compteurs

Parmi les techniques d'abstraction connues pour le *model checking*, il existe des méthodes spécifiques aux systèmes paramétrés. Pnueli *et al.* proposent dans [79] une technique d'abstraction basée sur des compteurs (*counter abstraction*) grâce à laquelle il est possible de vérifier automatiquement des propriétés de vivacité. L'idée de cette méthode est de compter le nombre de processus se trouvant dans le même état local et ainsi d'abstraire les multiples processus par des compteurs.

Cette abstraction a été définie dans le cadre théorique de l'abstraction finitaire. Considérons un système paramétré décrit par un ensemble de n processus possédant le même code. Chaque processus possède un nombre fini L de places (états de contrôle) déterminé par les valeurs des variables locales. On note $\pi[i]$ la variable de contrôle indiquant la place du processus i et $y_1, \ldots, y_b$ les variables partagées. L'abstraction α est définie grâce aux variables abstraites $\kappa_1, \ldots, \kappa_L \in \{0, 1, 2\}$ et $Y_1, \ldots, Y_b$ et une fonction d'abstraction $\mathcal{E}^\alpha$ donnée par :

- pour $l \in 0..L, \kappa_l = \begin{cases} 0 & \text{si } \forall i \in 1..n \cdot \pi[i] \neq l, \\ 1 & \text{si } \exists i_1 \in 1..n \cdot \pi[i_1] = l \wedge \forall i_2 \neq i_1 \cdot \pi[i_2] \neq l, \\ 2 & \text{sinon}\,; \end{cases}$
- pour $j \in 1..b, Y_j = y_j$.

Autrement dit, κ_l est égal à 0 lorsqu'il n'y a aucun processus dans la place l, 1 s'il y en a exactement un et 2 s'il y en a deux ou plus. Les variables abstraites κ_l correspondent aux compteurs pour chaque place et les variables Y_j aux mêmes variables partagées.

La condition initiale abstraite et la relation de transition abstraite sont calculées grâce au schéma d'abstraction de la section 2.1.6. Ainsi, les identités des processus sont abstraites par des compteurs limités à 2.

Cependant, l'abstraction par compteurs est limitée par un problème majeur : chaque processus est considéré comme parfaitement identique aux autres. En effet, il n'est pas possible de les différencier même par un simple identifiant. Ce qui est contraignant pour exprimer des propriétés impliquant des processus particuliers.

Abstraction par environnement

Clarke *et al.* proposent dans [20] une technique nommée *abstraction par environnement* (*environment abstraction*), spécifique aux systèmes paramétrés. Elle combine les méthodes d'abstraction par compteurs et par prédicats afin de relâcher deux contraintes imposées par l'abstraction par compteur : les variables à domaine borné et la trop forte symétrie entre processus. L'idée générale est de considérer l'abstraction du point de vue d'un seul processus, que l'on nommera *processus de référence*, et d'abstraire les autres qui représenteront son *environnement*. L'environnement d'un processus x est décrit par les relations entre les variables du processus x et celles des autres processus ainsi que par les états internes des autres processus. On constitue ainsi des *prédicats d'environnement*.

Les auteurs décrivent un système paramétré $\mathcal{P}(k)$ comme un ensemble de k processus exécutant le même code. De plus, on note $\mathcal{P}(N)$ la collection infinie des systèmes $\mathcal{P}(2), \mathcal{P}(3) \ldots$ L'état local d'un processus i est caractérisé par un état interne défini par une variable de contrôle $pc[i]$ de domaine fini ainsi que par un ensemble fini de variables de données non bornées $\{u_1[i], \ldots, u_d[i]\}$. Un *état global* du système est ainsi représenté par un k-uplet $(\mathcal{L}_1, \ldots, \mathcal{L}_k)$, où $\mathcal{L}_i$ est l'état local du processus i. Les transitions du système sont décrites pour chaque processus soit par des *transitions gardées* soit par des *mises à jour*. Les transitions gardées sont définies par une précondition impliquant les variables de données des processus et modifient l'état interne du processus considéré, tandis que les mises à jour peuvent modifier les variables de données du processus courant.

Afin d'abstraire le système, on définit des prédicats d'environnement. Notons $\mathcal{EP}_1(x, y), \ldots, \mathcal{EP}_r(x, y)$ tous les prédicats présents dans les gardes du système concret,

où x et y sont des identifiants de processus. On appelle *condition de relation* entre x et y les prédicats $R_i(x,y)$ de la forme $\pm\mathcal{EP}_1(x,y) \wedge \cdots \wedge \pm\mathcal{EP}_r(x,y)$, où $\pm\mathcal{EP}_i$ dénote $\mathcal{EP}_i$ ou sa négation $\neg\mathcal{EP}_i$. Un prédicat d'environnement $\mathcal{E}(x)$ pour un processus x a alors la forme $\exists y \cdot y \neq x \wedge R_i(x,y) \wedge pc[y] = j$ et signifie qu'il existe un processus y différent de x dont la relation avec x est décrite par R_i et dont l'état interne est j. On obtient ainsi un ensemble de prédicats d'environnement $\{\mathcal{E}_1, \ldots, \mathcal{E}_T\}$. Une *description* $\Delta(x)$ du processus x représente l'état interne du processus x ainsi que ses relations avec les autres processus. Elle est donnée par un prédicat de la forme $pc[x] = i \wedge \pm\mathcal{E}_1(x,y) \wedge \cdots \wedge \pm\mathcal{E}_T(x,y)$ où i est un état interne.

On définit ainsi un modèle abstrait donné par le système de transitions (S^A, Θ^A, ρ^A). Le système abstrait est composé d'un ensemble d'états abstraits S^A constitués de n-uplets de la forme $(pc, e_1, \ldots, e_T)$ où pc est l'état interne du processus de référence et chaque e_j est un booléen représentant la valeur du prédicat $\mathcal{E}_j(x)$. Chacun de ces n-uplet correspond à une description. Pour un processus de référence p, on obtient alors une fonction d'abstraction α_p qui à chaque état global concret g du système $\mathcal{P}(k)$ associe un état abstrait $(pc, e_1, \ldots, e_T)$, telle que pc est l'état de contrôle de p dans l'état global g et pour tout e_j on a $e_j = \top \iff g \models \mathcal{E}_j(p)$. L'ensemble des états initiaux abstraits Θ^A est alors défini par l'ensemble des états abstraits $s^A \in S^A$ tel qu'il existe un état initial concret s et un processus p tels que $\alpha_p(s) = s^A$. Enfin, la relation de transition $\rho^A \subseteq S^A \times S^A$ du modèle abstrait est définie de la façon suivante : $(s_1^A, s_2^A) \in \rho^A$ s'il existe un système concret $\mathcal{P}(k)$ avec un processus p et une transition concrète (s_1, s_2) de $\mathcal{P}(k)$ tels que $\alpha_p(s_1) = s_1^A$ et $\alpha_p(s_2) = s_2^A$. Ce qui est une variante de l'abstraction existentielle.

L'article [20] montre, par ailleurs, comment abstraire une propriété de sûreté du type $\forall x, y \cdot \mathrm{AG}\,\phi(x,y)$, où $\phi(x,y)$ est une formule portant une condition sur les variables de contrôle de deux processus de la forme $pc[x] = L_1 \wedge pc[y] = L_2$. L'abstraction ϕ^A de $\phi(x,y)$ est un prédicat sur les états abstraits qui est satisfait exactement par les états abstraits s^A pour lesquels il existe un système concret $\mathcal{P}(k)$ avec un processus p et un état global s tels que $\alpha_p(s) = s^A$ et $s \models \phi(p)$. L'abstraction de $\forall x, y \cdot \mathrm{AG}\,\phi(x,y)$ est alors $\mathrm{AG}\,\phi^A$. Il a été montré que pour un modèle paramétré $\mathcal{P}(N)$ abstrait par $\mathcal{P}^A$, et une propriété de sûreté ψ, du type présenté précédemment, abstraite par ψ^A, si $\mathcal{P}^A \models \psi^A$ alors $\mathcal{P}(N) \models \psi$. De plus, un résultat similaire a été prouvé pour des

propriétés de vivacité portant sur la variable de contrôle d'un unique processus à la fois.

2.2.2 Techniques de *cutoff*

Une autre approche au problème de la vérification paramétrée consiste à déterminer une valeur de paramètre minimale à partir de laquelle le problème reste équivalent pour une certaine propriété à vérifier. Il s'agit, pour un système paramétré $S(n)$ et une propriété ϕ, de trouver une valeur $c \in \mathbb{N}_1$, nommée *limite* (ou *cutoff*), telle que $\forall n \cdot S(n) \models \phi \iff \forall k \leq c \cdot S(k) \models \phi$.

Calcul statique du *cutoff*

Emerson et Kahlon proposent dans [34] un moyen d'établir cette valeur de *cutoff* en considérant des systèmes paramétrés composés d'entrelacements de plusieurs instances de classes de processus.

On décrit un système paramétré grâce à des *gabarits de processus*. Pour chaque classe l de processus, on définit un gabarit $U_l = (Q_l, T_l, init_l)$, où Q_l est un ensemble d'*états locaux*, T_l un ensemble de *transitions* et $init_l$ un *état local initial*. Chaque transition de T_l est étiquetée par une garde décrite par une expression booléenne sur des propositions atomiques correspondant aux états locaux. On note $U_l^i = (Q_l^i, T_l^i, init_l^i)$ la i-ème instance de processus de classe l. Le système global $(U_1, \ldots, U_k)^{(n_1,\ldots,n_k)}$ résulte de la composition asynchrone de toutes les instances de toutes les classes de processus, où n_i dénote le nombre d'instances de processus de classe i. On cherche alors un k-uplet $(c_1, \ldots, c_k)$ que l'on nommera *cutoff* tel que pour une propriété f, $\forall (n_1, \ldots, n_k) \geq (1, \ldots, 1) \cdot (U_1, \ldots, U_k)^{(n_1,\ldots,n_k)} \models f$ si et seulement si $\forall (d_1, \ldots, d_k) \geq (c_1, \ldots, c_k) \cdot (U_1, \ldots, U_k)^{(d_1,\ldots,d_k)} \models f$.

Les gardes considérées par les auteurs dans leurs travaux ont deux formes particulières : les *gardes disjonctives* et les *gardes conjonctives (et initiales)*. Pour un processus U_l^i d'un système $(U_1, \ldots, U_k)^{(n_1,\ldots,n_k)}$, une transition (p_l^i, q_l^i) est étiquetée par une garde de la forme $\bigvee_{r\neq i}(a_l^r \vee \cdots \vee b_l^r) \vee \bigvee_{j\neq l}(\bigvee_{k\in[1..n_j]}(a_j^k \vee \cdots \vee b_j^k))$ lorsqu'il s'agit d'une garde disjonctive et par une garde de la forme $\bigwedge_{r\neq i}(init_l^r \vee a_l^r \vee \cdots \vee b_l^r) \wedge \bigwedge_{j\neq l}(\bigwedge_{k\in[1..n_j]}(init_j^k \vee a_j^k \vee \cdots \vee b_j^k))$ pour une garde conjonctive, où a_j^k dénote

la proposition atomique correspondant à l'état local a_j du processus k de classe j. On considère alors deux types de systèmes : les systèmes à gardes disjonctives et les systèmes à gardes conjonctives.

Pour toute propriété à vérifier du type $\bigwedge_{i_l} \mathrm{A}h(i_l)$ et $\bigwedge_{i_l} \mathrm{E}h(i_l)$ où $h(i_l)$ est une formule de $\mathrm{LTL}_{\setminus X}$, A et E les quantificateurs universel et existentiel de chemins et i_l un indice de processus de classe l, les auteurs montrent que la valeur de *cutoff* est donnée par :

- $(c_1, \ldots, c_k)$, où $c_l = |U_l| + 2$ et pour tout $i \neq l$, $c_i = |U_i| + 1$ dans le cas des systèmes à gardes disjonctives, ou
- $(c_1, \ldots, c_k)$, où $c_l = 2|U_l| + 1$ et pour tout $i \neq l$, $c_i = 2|U_i|$ dans le cas des systèmes à gardes conjonctives ;

où $|U_i|$ est le nombre d'états locaux du gabarit des processus de classe i. De plus, dans le cadre des systèmes à gardes disjonctives, un résultat similaire est donné pour des propriétés exprimées sur deux classes de processus différentes $\bigwedge_{i_l, j_m} \mathrm{A}h(i_l, j_m)$ ou $\bigwedge_{i_l, j_m} \mathrm{E}h(i_l, j_m)$. Le *cutoff* est donné par :

- $(c_1, \ldots, c_k)$, où $c_l = |U_l| + 2$, $c_m = |U_m| + 2$ et pour tout $i \neq l, m$, $c_i = |U_i| + 1$.

Détection dynamique du *cutoff*

Kaiser *et al.* proposent dans [55] une méthode pour vérifier des propriétés d'accessibilité sur des *systèmes à états finis répliqués* en calculant dynamiquement un *cutoff*. Cette méthode consiste à calculer l'ensemble des états accessibles du contrôleur d'un système paramétré itérativement en augmentant le nombre de processus le composant. Si l'on est capable de déterminer la limite à partir de laquelle l'ensemble des états accessibles n'augmente plus, alors on aura atteint un *cutoff* pour les propriétés d'accessibilité considérées. On parle aussi dans ce cas de plateau final.

La méthode s'applique à des systèmes à états finis répliqués représentés par des structures de Kripke. M_n est une structure de Kripke modélisant un programme concurrent à n processus. On nomme plusieurs types d'états : les états *partagés* représentant les variables partagées, les états *locaux* représentant les variables locales à chaque processus et les états *thread* qui consistent en un couple (s, l) d'état partagé s

et d'état local l. Un état de M_n quant à lui est donné sous la forme $(s, l_1, \ldots, l_n)$ où s est un état partagé et $l_1, \ldots, l_n$ sont des états locaux. Le système s'exécute de façon asynchrone, c'est-à-dire que toute transition dans M_n est motivée par une transition entre deux états *thread* $(s, l_i) \to (s', l'_i)$. Seuls l'état partagé et un état local sont modifiés, les autres états locaux restant inchangés. Finalement, on dit qu'un état *thread* (s, l) est *accessible dans* M_n s'il existe un un état global accessible de M_n de la forme $(s, l_1, \ldots, l, \ldots, l_n)$.

Ainsi, les auteurs proposent dans [55] un algorithme de détection du plateau final des états accessibles pour ce type de systèmes.

2.2.3 Techniques d'induction

Invariant de réseau

D'autres techniques sont basées sur le principe de l'induction. Il s'agit de montrer que si un système composé d'un processus vérifie une propriété donnée, alors on peut prouver que la propriété est vérifiée pour un système composé de n processus par un raisonnement inductif. Plusieurs travaux ont été publiés pour résoudre ce problème [61, 35, 63], notamment ceux de Wolper et Lovinfosse [95] qui utilisent la notion d'*invariant de réseau* (ou *network invariant*). Pour montrer que la composition de n processus P vérifie une propriété S, ils proposent de prouver qu'un processus satisfait une certaine propriété I plus forte que S. Puis, ils prouvent que la composition de tout processus satisfaisant I avec un processus P satisfait aussi I. On appelle ce I un *network invariant*.

On considère un ensemble $\mathcal{P}$ de processus et une relation d'ordre $\leq$ sur cet ensemble. De plus, on considère un opérateur de composition de processus $\odot$ monotone par rapport à $\leq$, *i.e.* si $P_1 \leq P_2$ alors pour tout $P \in \mathcal{P}$, $P \odot P_1 \leq P \odot P_2$. Dans le cadre de l'article [95], les auteurs proposent, par exemple, comme sémantique une variante de TCSP pour les processus et l'inclusion des ensembles d'échec et de divergence pour l'ordre sur les processus [51]. Un système paramétré composé de n processus identiques P est alors représenté par un processus N_n tel que $N_n = P_0 \odot P \odot \cdots \odot P$ où P est répété n fois et P_0 est un processus spécial. Ainsi pour prouver qu'une propriété S est satisfaite pour tout $n > 1$ par un système paramétré N_n, il suffit de montrer

$\forall n \cdot N_n \leq S$. Un *network invariant* est défini sous la forme d'un processus I tel que $I \leq S$, $P_0 \leq I$ et $I \odot P \leq I$. Il devient alors possible de prouver par induction que $N_n \leq S$. La vérification de l'ordre pouvant être effectuée algorithmiquement, la partie la plus difficile de la méthode consiste alors à déterminer l'invariant I pour une spécification donnée. Cependant, il a été montré qu'il n'est pas toujours possible de trouver un invariant permettant de faire la preuve.

Invariant invisible

Pnueli *et al.* présentent dans [78] la méthode des *invariants invisibles* (ou *invisible invariants*) aussi basée sur le principe de l'induction et du *cutoff*. Supposons que l'on souhaite vérifier une propriété d'invariant $\mathrm{G}\, p$ sur un système paramétré $S(N)$. Contrairement à la méthode précédente, l'induction s'effectue sur la relation de transition. On recherche un invariant auxiliaire ϕ tel que $\phi \implies p$ (I1). Afin de prouver $\mathrm{G}\, p$, il faut alors montrer les prémisses inductives suivantes : $\Theta \implies \phi$ (I2) et $\phi \wedge \rho \implies \phi'$ (I3), où Θ est la condition initiale et ρ la relation de transition telles que décrites pour les systèmes discrets $\mathcal{D} = (V, \Theta, \rho)$ présentés en section 2.1.6. Le processus de vérification se réduit donc à deviner une bonne propriété ϕ et à montrer la validité des prémisses I1 à I3.

Les auteurs de [78] proposent une méthode complètement automatique pour résoudre le problème. Afin d'établir les prémisses I1-I3, ils commencent par montrer un résultat de *cutoff* pour celles-ci. Pour toute propriété auxiliaire ϕ impliquant un ou deux processus différents, un *cutoff* c, linéaire dans le nombre de variables de V, est donné pour la propriété suivante : les prémisses I1-I3 sont valides sur $S(k)$ pour tout $k > 1$ si et seulement si elles sont valides sur $S(k)$ pour tout $1 < k \leq c$. Il est ainsi possible de prouver algorithmiquement les prémisses. Notamment, les auteurs suggèrent l'utilisation des BDD dans l'optique de cette preuve. Par ailleurs, Pnueli *et al.* donnent des algorithmes de calcul automatique d'invariants auxiliaires [78]. Le terme d'*invisible invariant* provient du fait que l'invariant est généré automatiquement, l'utilisateur n'étant pas sollicité.

La méthode de vérification proposée par Pnueli *et al.* est totalement automatisée et ne demande pas d'interaction avec l'utilisateur pour une preuve. Cependant, elle reste incomplète. Notons qu'une généralisation de cette méthode est présentée dans [7]. Elle

permet notamment de vérifier des systèmes avec plusieurs types de paramètres.

2.3 Vérification de systèmes de transitions bien structurés

Cette section introduit la théorie des *systèmes de transitions bien structurés* (*well structured transition systems*, WSTS) [37, 1, 39] et présente une méthode qui résout pour ces systèmes une variante du problème de l'accessibilité, appelé *couverture*. Cette méthode partage des similitudes avec les méthodes d'abstraction mais elle est complète pour le problème de couverture. Elle est utile pour la vérification de systèmes de transitions infinis munis d'un préordre respectant un certain nombre de propriétés. Cette méthode ayant un intérêt particulier pour les chapitres suivants, elle est décrite de façon plus détaillée que les précédentes.

2.3.1 Théorie des systèmes de transitions bien structurés

La théorie des WSTS a été développée dans le but d'unifier et de généraliser des résultats de décidabilité pour les problèmes de terminaison et de couverture pour une certaine classe de systèmes infinis. Cette classe requiert notamment que l'espace d'états soit muni d'un *beau préordre* et que le système soit monotone vis-à-vis de ce préordre.

Un *préordre* est une relation binaire réflexive et transitive $\leq$ sur un ensemble X ; on note aussi $(X, \leq)$ le préordre. Un *ordre (partiel)* est un préordre antisymétrique. Un préordre $(X, \leq)$ est *bien fondé* s'il n'existe pas de suite infinie $(x_n)_{n\in\mathbb{N}}$ sur X telle que $x_{i+1} \leq x_i$ pour tout $i \in \mathbb{N}$. Une *antichaine* est un ensemble dans lequel chaque couple d'éléments est incomparable.

Définition 2.1 (WQO)**.** *Un beau préordre (well-quasi-order, WQO) est un préordre qui est bien fondé et pour lequel il n'existe pas d'antichaine infinie. Lorsque le préordre est antisymétrique, et donc est un ordre, on parle alors de bel ordre (well partial order, WPO) et de bon ordre s'il est aussi total.*

On utilisera dans ce manuscrit l'acronyme WQO comme terme générique pour tous

les beaux préordres, lorsque l'antisymétrie ou la partialité n'ont pas de pertinence pour la compréhension du problème.

Considérant un préordre $\leq$ sur X, un ensemble *clos par le haut* est un ensemble $Y \subseteq X$ tel que si $x \leq y$ et $x \in Y$ alors $y \in Y$. À tout élément $x \in X$, on associe un ensemble clos par le haut $\uparrow x = \{y \in X \mid x \leq y\}$ et pour tout $Y \subseteq X$, $\uparrow Y = \bigcup_{x \in Y} \uparrow x$. Une *base* pour un ensemble clos par le haut $Y \subseteq X$ est un ensemble B tel que $Y = \uparrow B$. On dit qu'un ensemble clos par le haut Y possède une base finie s'il existe un ensemble fini $B \subseteq Y$ qui est une base de Y.

Dans la suite de cette section on considère les systèmes de transitions ordonnés de type $(Q, \rightarrow, \leq)$, où $(Q, \rightarrow)$ est un système de transitions et Q est muni d'un préordre $\leq$.

Définition 2.2 (Couverture)**.** *Soit $\mathcal{S} = (Q, \rightarrow, \leq)$ un système de transitions ordonné. Le problème de couverture consiste à déterminer si pour un ensemble d'états initiaux $I \subseteq Q$ et un état $s \in Q$, il existe une exécution $i \xrightarrow{*} s'$ telle que $s \leq s'$ et $i \in I$. On dit que s est couvrable s'il existe une telle exécution. On note $cov(\mathcal{S}, I, s)$ le problème de couverture.*

Notons que le problème de couverture de s est équivalent au problème d'accessibilité de $\uparrow s$.

Comme montré dans [1], le problème de couverture peut être résolu en calculant l'ensemble $Pred^*(\uparrow s)$ de tous les prédécesseurs de $\uparrow s$. Il suffit ensuite de déterminer si $I \cap Pred^*(\uparrow s)$ est vide. Pour calculer $Pred^*(\uparrow s)$, on calcule la suite des ensembles $Pred(\uparrow s), Pred^2(\uparrow s) = Pred(Pred(\uparrow s)), \ldots, Pred^n(\uparrow s) \ldots$ et on obtient $Pred^*(\uparrow s) = \bigcup_{n \in \mathbb{N}} Pred^n(\uparrow s)$. Cette suite se stabilise et $Pred^*(\uparrow s)$ est représentable par une base finie lorsque le système respecte les deux conditions de WQO et de monotonie.

Définition 2.3 (Système de transitions monotone [39])**.** *Un système de transitions monotone (monotone transition system, MTS) est un système de transitions ordonné $\mathcal{S} = (Q, \rightarrow, \leq)$ tel que pour tout $q_1 \leq q_1'$ et toute transition $q_1 \rightarrow q_2$, il existe une exécution $q_1' \xrightarrow{*} q_2'$ telle que $q_2 \leq q_2'$.*

Maintenant, lorsqu'un système de transitions monotone $\mathcal{S} = (Q, \rightarrow, \leq)$ est muni d'un WQO $\leq$, alors c'est un système de transitions bien structuré.

Définition 2.4 (Système de transitions bien structuré [39]). *Un système de transitions bien structuré (Well-Structured Transition System, WSTS) est un système de transitions monotone $\mathcal{S} = (Q, \rightarrow, \leq)$ tel que $(Q, \leq)$ est un WQO.*

Par exemple, les réseaux de Petri sont des WSTS si l'on considère l'inclusion des marquages comme ordre. Les *systèmes à canaux non fiables* (*lossy channel systems*) [38] sont un autre exemple de WSTS.

On suppose les MTS effectifs, *i.e.* $\leq$ et $\rightarrow$ sont décidables (il existe un algorithme qui détermine si un couple d'éléments appartient à $\leq$ ou $\rightarrow$ respectivement). De plus, afin d'appliquer l'algorithme de vérification, une condition d'effectivité supplémentaire est requise.

Définition 2.5 (Pred-base effective [39]). *Un système de transitions ordonné $\mathcal{S} = (Q, \rightarrow, \leq)$ possède une pred-base effective s'il existe un algorithme acceptant tout état $q \in Q$ et retournant $pb(q)$, une base finie de $\uparrow Pred(\uparrow q)$.*

Pour un système monotone $\mathcal{S} = (Q, \rightarrow, \leq)$ avec pred-base effective, et un sous-ensemble d'états $F \subseteq Q$, l'analyse de couverture en arrière consiste à calculer la suite des ensembles finis $K_0, K_1 \ldots$ tels que $K_0 = F$ et $K_{n+1} = K_n \cup pb(K_n)$. On a $Pred^*(\uparrow F) = \bigcup_{i \in \mathbb{N}} \uparrow K_i$. Si la suite converge, alors sa limite est calculable.

Lemme 2.1 ([39]). *Soit $\mathcal{S} = (Q, \rightarrow, \leq)$ un MTS avec pred-base effective et $s \in Q$. S'il existe $m \in \mathbb{N}$ tel que $\uparrow K_m = \uparrow K_{m+1}$, alors $\uparrow K_m = Pred^*(\uparrow s)$.*

La condition de pred-base effective est suffisante pour montrer que chaque $\uparrow K_i$ est calculable. Comme les MTS sont effectifs et que le préordre $\leq$ est décidable, alors l'inclusion $\uparrow K_n \supseteq \uparrow K_{n+1}$ et l'égalité $\uparrow K_n = \uparrow K_{n+1}$ peuvent être testés. En outre, si $\leq$ est un WQO, alors la suite $(\uparrow K_n)_{n \in \mathbb{N}}$ converge. Ainsi, le calcul itératif de K_n donne une procédure de décision au problème de couverture de s.

Théoreme 2.1 ([39]). *Pour un WSTS $\mathcal{S} = (Q, \rightarrow, \leq)$ avec pred-base effective et $s \in Q$, une base finie de $Pred^*(\uparrow s)$ est calculable. Donc, $cov(\mathcal{S}, I, s)$ est décidable.*

2.3.2 WQO et graphes

La théorie des WSTS repose principalement sur deux propriétés importantes : le WQO et la monotonie. Afin de mieux comprendre les problèmes soulevés dans

cette thèse, cette section détaille le concept de WQO. Notamment, on y présente les principaux WQO existants sur différentes structures intéressantes comme les arbres et les graphes.

Proposition 2.1. *Si $\leq$ est un préordre sur X, les propositions suivantes sont équivalentes :*

1. *$\leq$ est un WQO ;*
2. *Pour toute suite infinie $(x_n)_{n\in\mathbb{N}}$ de X, il existe $i < j$ tels que $x_i \leq x_j$;*
3. *Toute suite infinie $(x_n)_{n\in\mathbb{N}}$ de X contient une sous-suite infinie non décroissante $x_{n_0} \leq x_{n_1} \leq \cdots$ avec $n_0 < n_1 < \cdots$*

De l'équivalence 2, on définit par ailleurs le concept de *mauvaise suite* pour le préordre $(X, \leq)$ par une suite infinie $(x_n)_{n\in\mathbb{N}}$ de X telle que pour tout $i, j \in \mathbb{N}$ avec $i < j$, on a $x_i \not\leq x_j$. Ainsi, un WQO est un préordre ne contenant pas de mauvaise suite.

L'ordre sur les entiers naturels est par exemple un bel ordre tandis que l'ordre sur les entiers relatifs ne l'est pas, n'étant pas bien fondé. Un résultat sur les vecteurs d'entiers a été démontré par le lemme de Dickson suivant.

Proposition 2.2 (Lemme de Dickson [29])**.** *L'ordre produit $(\mathbb{N}^k, \leq)$ est un WQO.*

Le lemme de Dickson se généralise à tout produit cartésien de WQO de la façon suivante.

Proposition 2.3. *Soient $(X, \leq_X)$ et $(Y, \leq_Y)$ deux WQO. Alors, $(X \times Y, \leq)$, défini par $(x, y) \leq (x', y')$ ssi $x \leq_X x'$ et $y \leq_Y y'$, est un WQO.*

Les WQO possèdent aussi les deux autres propriétés intéressantes suivantes : réduire l'ensemble des éléments d'un WQO ou étendre la relation de préordre permet de conserver la propriété de WQO.

Proposition 2.4. *Soit $(X, \leq)$ un WQO.*

1. *Pour tout $Y \subseteq X$, $(Y, \leq)$ est un WQO.*
2. *Pour tout $\preceq \supseteq \leq$, $(X, \preceq)$ est un WQO.*

Démonstration. Soit $(X, \leq)$ un WQO.

1. Supposons par l'absurde que $(Y, \leq)$ possède une mauvaise suite $(x_n)_{n\in\mathbb{N}}$. Alors cette suite se retrouve aussi dans $X \supseteq Y$. Contradiction.

2. Si toute suite infinie $(x_n)_{n\in\mathbb{N}}$ de $(X, \leq)$ possède une paire croissante $x_i \leq x_j$ avec $i < j$, alors cette paire croissante se retrouve aussi dans $(X, \preceq)$.

□

Il existe plusieurs WQO notables dans la littérature qui sont associés à des structures de données plus complexes comme les mots, les arbres ou les graphes. Nous présentons dans la suite certains des plus importants. Introduisons tout d'abord la théorie de Nash-Williams [73] qui propose une façon de démontrer des WQO.

Définition 2.6 (Mauvaise suite minimale)**.** *Une mauvaise suite $(x_n)_{n\in\mathbb{N}}$ d'un préordre $(X, \leq)$ est dite minimale si pour tout $k \in \mathbb{N}$, il n'existe pas de mauvaise suite $x_0, \ldots, x_{k-1}, y_k, y_{k+1} \ldots$, où $y_k, y_{k+1} \ldots$ est une suite telle que $y_i < x_i$ pour tout $i \geq k$. De plus, notons $S((x_n)_{n\in\mathbb{N}}) = \{x \in X \mid \exists i \cdot x < x_i\}$.*

Plutôt que de montrer l'absence de mauvaise suite, lorsque le préordre est bien fondé, on peut se contenter de chercher les mauvaises suites minimales.

Lemme 2.2. *Tout préordre bien fondé qui n'est pas WQO possède une mauvaise suite minimale.*

L'argument de preuve principal de Nash-Williams pour les WQO se trouve dans le lemme suivant.

Lemme 2.3. *Soit $(X, \leq)$ un préordre et $(x_n)_{n\in\mathbb{N}}$ une mauvaise suite minimale. Alors $(S((x_n)_{n\in\mathbb{N}}), \leq)$ est un WQO.*

L'argument de Nash-Williams peut être utilisé dans la preuve de WQO du lemme de Higman, qui établit que l'ordre sous-mot est un WQO [50]. Définissons formellement cet ordre.

Définition 2.7 (Ordre sous-mot)**.** *Soit $(X, \leq)$ un préordre sur un alphabet X. On définit $\sqsubseteq$ sur X^* de la façon suivante. Si $u = a_0 a_1 \dots a_n$ et $v = b_0 b_1 \dots b_m$ sont deux mots de X^*, alors $u \sqsubseteq v$ ssi il existe $0 \leq k_0 < \dots < k_n$ tels que $a_i \leq b_{k_i}$ pour tout $i \leq n$.*

Par exemple, $abd \sqsubseteq abcde$. Remarquons que l'ordre sous-mot est différent de l'ordre préfixe dont il est un sur-ensemble.

Proposition 2.5 (Lemme de Higman [50])**.** *Si $(X, \leq)$ est un WQO, alors $(X^*, \sqsubseteq)$ est un WQO.*

Démonstration. Supposons par l'absurde que $(X^*, \sqsubseteq)$ n'est pas un WQO. Alors il existe une mauvaise suite minimale $(u_n)_{n\in\mathbb{N}}$. On sait que le mot vide ϵ n'appartient pas à cette suite car $\epsilon \sqsubseteq w$ pour tout $w \in X^*$. Donc chaque u_i peut s'écrire sous la forme $u_i = a_i.v_i$ où $a_i \in X$ et $v_i \in X^*$. Par le lemme 2.3, $(S((u_n)_{n\in\mathbb{N}}), \sqsubseteq)$ est un WQO. Donc $(\{v_i\}_{i\in\mathbb{N}}, \sqsubseteq)$ est un WQO car $\{v_i\}_{i\in\mathbb{N}} \subseteq S((u_n)_{n\in\mathbb{N}})$. Par ailleurs, $\{a_i\}_{i\in\mathbb{N}} \subseteq X$ donc $(\{a_i\}_{i\in\mathbb{N}}, \leq)$ est un WQO. Donc par composition, $\{(a_i, v_i)\}_{i\in\mathbb{N}}$ muni de l'ordre produit est WQO. Ce qui correspond à $(X^*, \sqsubseteq)$. On aboutit donc à une contradiction. □

Le lemme de Higman se généralise pour les structures d'arbres par le théorème de Kruskal [60]. Pour un alphabet X, on note $\mathcal{T}(X)$ l'ensemble des arbres étiquetés sur X et on représente un arbre étiqueté par une expression de la forme $Arbre = Noeud \mid Noeud[Arbre, \dots, Arbre]$, où $Noeud$ est un élément de X.

Définition 2.8. *Soit $(X, \leq)$ un préordre. On définit $\leq_T$ sur $\mathcal{T}(X)$ inductivement par $a[u_1, \dots, u_n] \leq_T b[t_1, \dots, t_m] \iff$*

- *$a \leq b$ et $(u_1, \dots, u_n) \leq_{T,*} (t_1, \dots, t_m)$,*
- *ou $\exists i \in \{1, \dots, m\}$ tel que $a[u_1, \dots, u_n] \leq_T t_i$.*

où $\leq_{T,*}$ est l'extension de $\leq_T$ aux séquences (définition similaire à l'ordre sous-mot).

Par exemple, $a[b, e] \leq_T a[b[d], c[e, f]]$.

Proposition 2.6 (Théorème de Kruskal [60])**.** *Si $(X, \leq)$ est un WQO, alors $(\mathcal{T}(X), \leq_T)$ est un WQO.*

Démonstration. $(\mathcal{T}(X), \leq_T)$ est clairement bien fondé. Supposons par l'absurde que $(\mathcal{T}(X), \leq_T)$ n'est pas un WQO. Donc il existe une mauvaise suite minimale $(t_n)_{n\in\mathbb{N}}$. Pour tout $i \in \mathbb{N}$, $t_i \neq \epsilon$ donc on peut écrire $t_i = a_i[s_{i,0}, \ldots, s_{i,n_i}]$. Par le lemme 2.3, $\{s_{i,j}\}_{i\in\mathbb{N};j\in[0,n_i]} \subseteq S((t_n)_{n\in\mathbb{N}})$ est un WQO. Donc $\{s_{i,j}\}^*_{i\in\mathbb{N};j\in[0,n_i]}$ est aussi un WQO par le lemme de Higman. Par ailleurs, $\{a_i\}_{i\in\mathbb{N}} \subseteq X$ est WQO. Donc par composition, $\{(a_i, (s_{i,0}, \ldots, s_{i,n_i})\}_{i\in\mathbb{N}}$ l'est aussi. Contradiction. □

De la même façon, on généralise le théorème de Kruskal aux graphes grâce au théorème de Robertson et Seymour [81, 82]. Introduisons tout d'abord quelques définitions préliminaires sur les graphes. On note (V, E) un graphe (non orienté), où V est un ensemble de sommets et E un ensemble d'arêtes, une arête étant définie par un paire de sommets. Dans un graphe (V, E), un *chemin* est une suite de sommets $v_1, v_2, \ldots, v_n \in V$ tels que $\{v_i, v_{i+1}\} \in E$ pour tout $1 \leq i \leq n-1$. Un *chemin simple* est un chemin $v_1, v_2, \ldots, v_n \in V$ tel que $v_i \neq v_j$ pour tout $i \neq j$. Un graphe (V, E) est dit *connexe* si tout couple de sommets $v, w \in V$ est relié par un chemin $v, \ldots, w$ dans E. (V, E) est un *sous-graphe* de (V', E') si $V \subseteq V'$ et $E \subseteq E'$. C'est un *sous-graphe induit* si $V \subseteq V'$ et $E = \{\{v_1, v_2\} \in E' \mid v_1, v_2 \in V\}$. Un graphe étiqueté est un triplet (V, E, λ), où (V, E) est un graphe et $\lambda : V \to X$ une fonction d'étiquetage avec X un alphabet. On note $\mathcal{G}(X)$ l'ensemble des graphes étiquetés dont la fonction d'étiquetage a pour codomaine X.

Définition 2.9. *Soit $(X, \leq)$ un préordre. Si $G_1 = (V_1, E_1, \lambda_1) \in \mathcal{G}(X)$ et $G_2 = (V_2, E_2, \lambda_2) \in \mathcal{G}(X)$ sont deux graphes étiquetés, on a $G_1 \leq_{GM} G_2$ ssi il existe une fonction η de domaine V_1 telle que :*

- *pour tout $v \in V_1$, $\eta(v)$ est un sous-graphe connexe de G_2 ;*
- *pour tout $v \in V_1$, il existe un sommet w de $\eta(v)$ tel que $\lambda_1(v) \leq \lambda_2(w)$;*
- *pour tout $v, v' \in V_1$ tels que $v \neq v'$, $\eta(v) \cap \eta(v') = \emptyset$.*

Si l'on excepte la condition d'ordonnancement des étiquettes, on utilise aussi le terme de *mineur* et on dit que le graphe (V_1, E_1) est un mineur de (V_2, E_2). Une autre façon de définir la relation de mineur est de dire que l'on obtient (V_1, E_1) à partir de (V_2, E_2) à la suite de suppressions de sommets ou d'arêtes et de contractions d'arêtes,

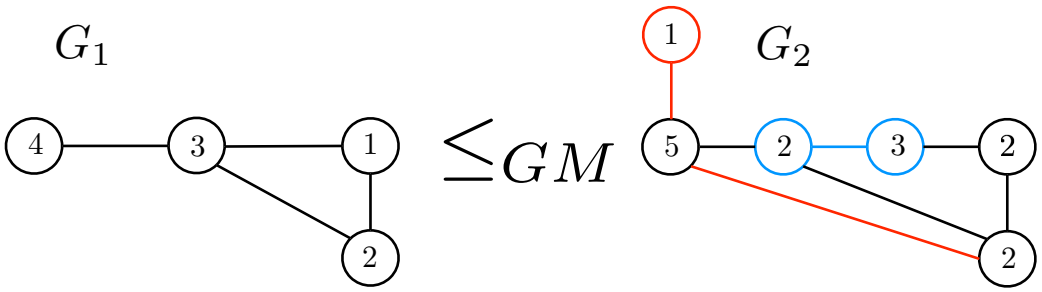

figure 2.1 – Exemple de mineur

une contraction d'arête correspondant à une fusion des deux sommets d'une arête. La figure 2.1 montre un exemple où $G_1 \leq_{GM} G_2$, avec $G_1, G_2 \in \mathcal{G}(\mathbb{N})$. Le sommet et les arêtes en rouge du graphe G_2 ont été supprimés de G_1 tandis que l'arête en bleu a été contractée.

Proposition 2.7 (Théorème de Robertson-Seymour [82])**.** *Si* $(X, \leq)$ *est un WQO, alors* $(\mathcal{G}(X), \leq_{GM})$ *est un WQO.*

Du fait de la forme de la structure d'un graphe, le schéma de preuve de Nash-Williams ne peut pas être réutilisé pour montrer le WQO. La preuve du théorème de Robertson-Seymour découle d'une série d'articles nommés *Graph Minors. I - XXIII* publiés de 1983 à 2010. On notera que le lemme de Higman est un cas particulier du théorème de Kruskal qui est lui-même un cas particulier du théorème de Robertson-Seymour.

Les relations de sous-graphe et de sous-graphe induit, quant à elles, bien qu'étant des préordres, ne sont pas des WQO comme le démontrent les antichaines infinies $(C_n)_{n\in\mathbb{N}}$ et $(H_n)_{n\in\mathbb{N}}$ représentées dans la figure 2.2. Plus précisément, ces deux antichaines sont caractéristiques des seuls types d'antichaines infinies possibles pour ces préordres dans le cadre des graphes non étiquetés. En effet, Ding prouve dans [30] qu'une sous-classe de graphes, ne contenant les graphes C_i et H_i qu'en nombre fini, est WQO pour la relation de sous-graphe induit (et donc de sous-graphe car sous-graphe induit implique sous-graphe). Cette propriété peut être aussi formulée en restreignant les chemins simples pour les graphes étiquetés. Notons $\mathcal{P}_k$ la classe des graphes qui ne contiennent pas de chemins simples de longueur k et $\mathcal{P}_k(X) \subseteq \mathcal{G}(X)$ celle des graphes $\mathcal{P}_k$ étiquetés par un alphabet X. On définit la relation de sous-graphe induit pour les graphes étiquetés formellement comme suit.

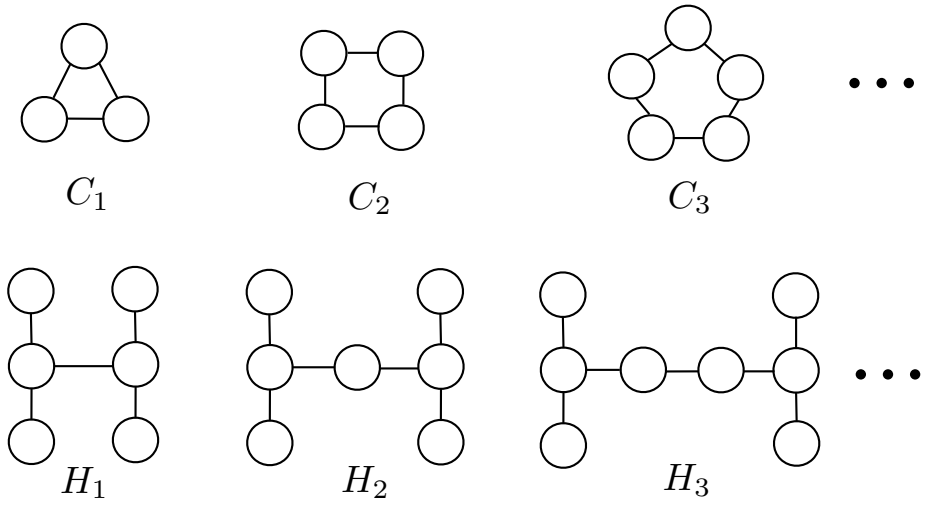

figure 2.2 – Antichaines infinies $(C_n)_{n\in\mathbb{N}}$ et $(H_n)_{n\in\mathbb{N}}$

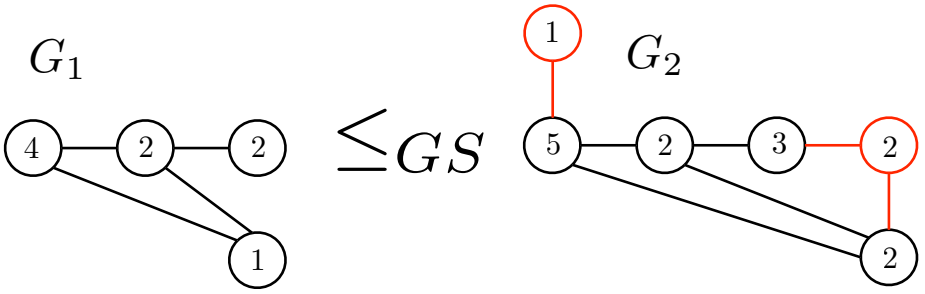

figure 2.3 – Exemple de sous-graphe induit

Définition 2.10. *Soit $(X, \leq)$ un préordre. Si $G_1 = (V_1, E_1, \lambda_1) \in \mathcal{G}(X)$ et $G_2 = (V_2, E_2, \lambda_2) \in \mathcal{G}(X)$ sont deux graphes étiquetés, on a $G_1 \leq_{GS} G_2$ ssi il existe une injection $f : V_1 \to V_2$ telle que :*

- *pour tout $v, w \in V_1$, $\{v, w\} \in E_1 \iff \{f(v), f(w)\} \in E_2$;*
- *pour tout $v \in V_1$, $\lambda_1(v) \leq \lambda_2(f(v))$.*

La figure 2.3 montre un exemple de sous-graphe induit étiqueté $G_1 \leq_{GS} G_2$ pour $G_1, G_2 \in \mathcal{G}(\mathbb{N})$. On donne un exemple d'injection symbolisée par la suppression des sommets et des arêtes en rouge.

Proposition 2.8 (Théorème de Ding [30])**.** *Soit $k \in \mathbb{N}$. Si $(X, \leq)$ est un WQO, alors*

$(\mathcal{P}_k(X), \leq_{GS})$ *est un WQO.*

Contrairement au théorème de Robertson-Seymour, le théorème de Ding se restreint à une sous-classe particulière de graphes mais dans la pratique le préordre de sous-graphe induit peut s'avérer plus utile que le préordre de mineur, comme nous le montrerons dans les chapitres suivants.

Chapitre 3

Étude des méthodes de spécification et de vérification pour les systèmes d'information

Ce chapitre s'intéresse à l'étude de plusieurs méthodes formelles de spécification et de vérification dans le cadre des systèmes d'information, dans le but de déterminer les techniques les plus adaptées. Le problème principal de la vérification des systèmes d'information réside dans le fait qu'il s'agit de systèmes paramétrés. La vérification de systèmes paramétrés, aussi appelée vérification paramétrée, est un problème connu comme étant indécidable dans le cas général [6]. Cependant, comme présenté dans le chapitre 2, il existe des méthodes de vérification pour des classes particulières de systèmes paramétrés. On compare, dans ce chapitre, plusieurs de ces méthodes de vérification que l'on applique aux systèmes d'information. Par ailleurs, on étudie plus particulièrement les problèmes de la sûreté et de l'accessibilité pour les systèmes paramétrés.

Ce chapitre est organisé de la façon suivante. La section 3.1 présente les systèmes d'information et les caractérise par rapport aux autres systèmes dynamiques. La section 3.2 explore les difficultés de spécification d'un système d'information et propose une méthode adaptée basée sur les ASTD. La section 3.3 analyse diverses techniques de vérification paramétrée du point de vue des caractéristiques des systèmes d'infor-

mation. Enfin, la section 3.4 présente une étude du problème de l'accessibilité pour un cas particulier de système paramétré.

3.1 Systèmes d'information

Un *système d'information* est un système dont le but est de stocker et gérer des données. Il comprend aussi bien les bases de données que les technologies permettant de les manipuler et d'interagir avec les utilisateurs. Dans la pratique, un tel système se caractérise par la quantité importante d'information à gérer et par la complexité des interactions entre les entités qui composent le système.

3.1.1 Entités et associations

Intuitivement, un système d'information est mû par un ensemble de processus s'exécutant en concurrence et regroupés en plusieurs classes. Ce genre de système est souvent représenté par un *modèle conceptuel de données* comme un *diagramme entité-association* ou un *diagramme de classes* [84]. Ces diagrammes sont composés d'*entités* qui représentent les classes de processus et d'*associations* qui modélisent les interactions entre elles. D'autre part, les données font partie de la structure des entités et des associations en tant qu'*attributs*. Chaque entité peut être instanciée plusieurs fois. Ce qui permet de créer des structures répliquées dans un système d'information. Un système d'information étant un système dynamique, le nombre d'instances de chaque entité peut varier. De même, les interactions entre instances sont multiples et évoluent au fil du temps. Tout cela résulte en une topologie des processus souvent complexe.

3.1.2 Exemple de système d'information

Un exemple de système d'information est le système de gestion de bibliothèque décrit par le diagramme de classes de la figure 3.1 [40]. Le système est composé de deux classes : *member* qui représente les utilisateurs inscrits de la bibliothèque, et *book* les livres qui y sont répertoriés. Un membre peut rejoindre et quitter le système. Un livre peut être acquis et retiré de la bibliothèque. On distingue deux associations entre

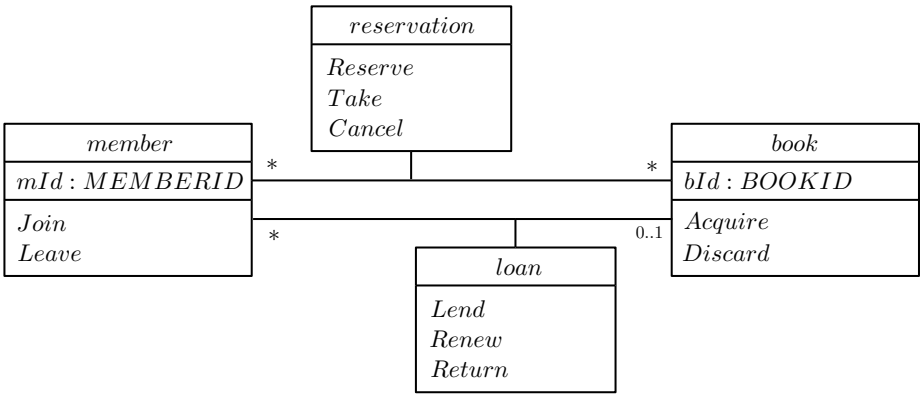

figure 3.1 – Diagramme de classes d'un système de gestion de bibliothèque

ces deux classes : une association *loan* et une association *reservation*. L'association *loan* correspond à des actions de prêt de livre à un membre ainsi qu'à des actions de renouvellement et de retour de prêt. Chaque livre peut être emprunté par un seul membre à la fois et chaque membre peut emprunter plusieurs livres. L'association *reservation* permet aux membres de réserver un livre qui n'est pas disponible et comporte les actions de réservation, de récupération et d'annulation. Il est possible pour un membre de réserver plusieurs livres et un livre peut être réservé par plusieurs membres.

Du point de vue des données, le système doit conserver la liste des membres inscrits, des livres acquis, les prêts et réservations en cours. Il est possible de modéliser ce système par un ensemble de processus de différentes classes. Par exemple, on peut distinguer une classe de processus pour les membres, une pour les livres, une pour les prêts et une dernière pour les réservations. Le système se définit alors par la composition en parallèle d'instances de chaque classe.

La modélisation des comportements possibles peut s'effectuer grâce à plusieurs méthodes de spécification. Quelques solutions de spécification de ce système sont proposées dans [40].

3.1.3 Caractérisation des systèmes d'information

L'article [40] présente une liste de plusieurs caractéristiques utiles pour un langage de spécification dans le cadre des systèmes d'information. Parmi celles-là, on retrouve

les suivantes :

- la représentation abstraite de plusieurs instances d'entités : le modèle doit permettre de paramétrer le nombre d'instances pour chaque entité ;
- la représentation des entités et des associations : le modèle doit permettre de représenter les actions qui impliquent plusieurs entités/associations ;
- la représentation de scénarios de systèmes d'information : le modèle doit permettre de spécifier l'ordonnancement de séries d'actions du système.

De même, les propriétés à vérifier pour ces systèmes ont souvent les caractéristiques suivantes :

- l'abstraction sur les instances d'entités : similairement à la caractéristique des systèmes, une propriété devrait pouvoir être quantifiée universellement sur l'ensemble des instances possibles ;
- les classes de propriétés les plus pertinentes sont les propriétés de sûreté, d'accessibilité et d'invariant ;
- les propriétés sont exprimables sur les états et sur les actions.

Systèmes d'information paramétrés

Un système d'information peut être considéré comme un système paramétré. En effet, il est composé de processus représentant différentes entités. Une entité génère une classe de processus qui partagent un code identique. Ainsi, un tel système comporte des structures identiques répliquées plusieurs fois. De plus, il possède, en général, plusieurs entités différentes. Les paramètres d'un système d'information correspondent alors aux nombres d'instances de processus de chaque entité qui compose le système. Par exemple, un système de gestion de bibliothèque, constitué de processus *book* et de processus *member*, est paramétré par des variables entières n et m correspondant aux nombres de livres et de membres. Il s'agit d'un système paramétré lorsque le modèle est défini pour toutes valeurs des paramètres. Le système paramétré représente alors l'ensemble infini de tous les systèmes possibles instanciant les valeurs n et m. Afin

de représenter un système d'information sous sa forme paramétrée, un langage de spécification doit donc permettre la modélisation des paramètres.

Propriétés

Dans le cadre des systèmes d'information, on s'intéresse plus particulièrement aux propriétés de sûreté et d'accessibilité, qui sont les propriétés les plus répandues pour la validation de ces systèmes. En effet, on souhaite généralement qu'un système d'information respecte certaines politiques de sécurité, que l'on peut modéliser par des propriétés de sûreté, et qu'il soit possible de contrôler le système afin de l'amener dans un état donné, ce qui est modélisable par des propriétés d'accessibilité. Dans l'exemple du système de gestion de bibliothèque de la figure 3.1, on peut définir les propriétés suivantes. *"Un livre ne peut pas être emprunté par deux membres à la fois"* est une propriété de sûreté (ainsi qu'une propriété d'invariant). *"Il existe toujours une procédure permettant à un membre de résilier son inscription"* est une propriété d'accessibilité. D'autre part, le problème de la vérification des propriétés de sûreté est similaire à celui de la vérification de propriété d'accessibilité (voir section 3.4).

États et actions

Les spécifications des systèmes peuvent être de deux types : *orientées états* (comme les structures de Kripke, la méthode B etc.) ou *orientées actions* (ou *évènements*) (comme les automates de Büchi, les algèbres de processus etc.). Le choix du modèle de spécification est important pour le type de propriétés que l'on souhaite vérifier sur le système. Par exemple, une propriété exprimée dans une logique temporelle portera soit sur des variables d'état soit sur les actions du système. Le cas d'étude de l'article [40] nous montre qu'il est possible de modéliser un système d'information des deux façons. Cependant, il peut être plus difficile de spécifier des propriétés ne portant que sur des actions. En effet, la propriété *"Un membre ne peut renouveler l'emprunt d'un livre s'il est réservé"* est facilement exprimable s'il existe une variable indiquant si un livre est emprunté. Mais dans le cas d'une spécification orientée actions, il est nécessaire de spécifier l'ordre des actions d'emprunt et de réservation pour modéliser la propriété. Toutefois, dans certains cas, les contraintes d'ordonnancement des actions

sont difficiles à spécifier. Il peut donc être utile, pour un langage de spécification, de permettre la modélisation simple des différents états du système.

3.2 Méthodes de modélisation pour les systèmes d'information

Cette section s'intéresse aux problèmes de modélisation soulevés par les systèmes d'information. Les modèles utilisés dans la suite permettent de spécifier directement les états d'un système ainsi que ses différents évènements. On commence par spécifier le système d'information de gestion d'une bibliothèque, présenté dans la section 3.1, grâce à la composition de plusieurs systèmes de transitions, auxquels on ajoute le concept de paramètre. Puis, on propose une méthode de spécification de systèmes paramétrés plus adaptée aux systèmes d'information et basée sur les ASTD présentés dans le chapitre 1. Cette méthode, nommée PASTD, permet de modéliser des systèmes paramétrés tout en prenant en compte la complexité des systèmes d'information.

3.2.1 Modélisation de systèmes d'information par des systèmes de transitions étiquetés

Une méthode élémentaire de spécification de système est le système de transitions étiqueté. Cependant, les systèmes d'information se comportant comme des systèmes paramétrés, un simple STE n'est pas suffisant pour représenter un système d'information. Ainsi, afin de permettre la spécification de systèmes paramétrés, on introduit la notion de paramètre pour les STE. Rappelons que les systèmes paramétrés sont une classe de systèmes généralement composés de plusieurs processus représentant une ou plusieurs entités. Un paramètre dans ce cas permet de définir le nombre de ces processus répliqués pour une entité donnée. Une valeur de paramètre donne la taille du système instancié.

STE à un paramètre entier

Considérons une composition de STE de la forme $\mathcal{S}(n) = \mathcal{A} \parallel \|\|^n \mathcal{B}$, où $\mathcal{A}$ et $\mathcal{B}$ sont deux STE finis et n est un entier correspondant au nombre de processus $\mathcal{B}$ composant le système. $\mathcal{S}(n)$ est un exemple de système paramétré qui représente un système composé d'un contrôleur $\mathcal{A}$ et d'un entrelacement de n processus répliqués $\mathcal{B}$. Notons que pour tout entier $n \in \mathbb{N}$, le système instancié $\mathcal{S}(n)$ peut aussi être représenté par un STE fini, dont la construction est trivialement donnée par la définition des opérateurs de composition parallèle et d'entrelacement. Cependant, si l'on souhaite représenter l'ensemble de tous les systèmes instanciables, alors on peut le noter par expression $\bigcup_{n\in\mathbb{N}} \mathcal{S}(n)$, *i.e.* l'union infinie de tous les STE. Cette union ne peut pas être représentée par un STE fini, ce qui justifie l'importance de la notation par une expression paramétrée.

Cependant, tous les systèmes paramétrés ne sont pas modélisables par un système du type $\mathcal{A} \parallel \|\|^n \mathcal{B}$. En effet, un système d'information peut comprendre plusieurs paramètres représentant chacun un type différent de processus. Par exemple, le système de gestion de bibliothèque a pour paramètres le nombre de livres et le nombre de membres qui le composent. De plus, les interactions entre les processus *book* et les processus *member* peuvent être gérées par une autre classe de processus, par exemple *loan*. Chacun des processus *loan* étant spécifique à une paire *book-member*, il devient alors important de conserver un identifiant pour chaque livre et chaque membre dans la spécification.

STE à plusieurs paramètres

Afin de prendre en compte les interactions entre processus, on peut étendre la définition des systèmes de transitions étiquetés.

- On considère que les transitions d'un STE $\mathcal{S} = (Q, \Sigma, \rightarrow, I)$ peuvent être étiquetées par des actions paramétrées par des variables, par exemple $a(x, y)$ où $a \in \Sigma$ et x et y représentent deux paramètres. De même, on note $\mathcal{S}(x, y)$ le système paramétré où x et y sont les variables des actions paramétrées. Afin d'obtenir un système de transition concret, chacun des paramètres doit être instancié par une valeur, par exemple $\mathcal{S}(i, j)$ est le STE résultant de l'instanciation de x par

i et de y par j. De même, $a(i,j)$ est une action concrète du système.

- Notons que dans le cadre d'une composition parallèle $\mathcal{S}_1(i,j) \parallel \mathcal{S}_2(i,j)$ on ne synchronise que les actions qui sont instanciées par les mêmes paramètres, donc $a(i,j)$ ne se synchronise pas avec $a(i,k)$ si j est différent de k. Par ailleurs, on ajoute à la syntaxe des actions un métacaractère "_". Ce métacaractère peut remplacer un paramètre dans la description d'une action paramétrée afin de représenter l'ensemble de toutes les valeurs d'instanciation possibles pour le paramètre. Par exemple $a(i,\_)$ représente l'ensemble de toutes les actions $a(i,j)$, $a(i,k)$, etc. Remarquons que $a(x,\_)$ se synchronise avec $a(x,y)$ pour toutes valeurs de x et de y.

- Enfin, les opérateurs de composition parallèle et d'entrelacement sont généralisés sur des ensembles de valeurs, par exemple $|||_{x\in V}\,\mathcal{S}(x)$, où V est un ensemble de valeurs et x un paramètre du système $\mathcal{S}$.

On peut représenter un système de gestion d'emprunts d'une bibliothèque (sans réservations) par l'expression Lib suivante :

$$Lib = (|||_{b\in B}\, Book(b)) \parallel (|||_{m\in M}\, Member(m)) \parallel (|||_{b\in B, m\in M}\, Loan(b,m))$$

où $Book(b)$, $Member(m)$ et $Loan(b,m)$ sont les STE représentant les processus pour les livres, les membres et les emprunts respectivement et décrits dans la figure 3.2. La bibliothèque est constituée de plusieurs livres et de plusieurs membres représentés par deux ensembles d'identifiants B et M. Chaque livre $b \in B$ est géré par un processus $Book(b)$. Il peut être acquis par la bibliothèque puis emprunté par un seul membre à la fois avant d'être retourné. De même chaque membre $m \in M$ est représenté par un processus $Member(m)$ mais peut effectuer plusieurs emprunts à la suite. Enfin, toute paire (livre b, membre m) est gérée par un processus $Loan(m,b)$.

Plus généralement, on peut construire un système paramétré de type

$$\mathcal{S}(X,Y) = (|||_{a\in X}\, \mathcal{A}(a)) \parallel (|||_{b\in Y}\, \mathcal{B}(b)) \parallel (|||_{a\in X, b\in Y}\, \mathcal{C}(a,b))$$

où X et Y sont des variables qui sont les paramètres du système $\mathcal{S}$. Intuitivement, contrairement au cas des STE à un paramètre dont la taille est donnée par un entier

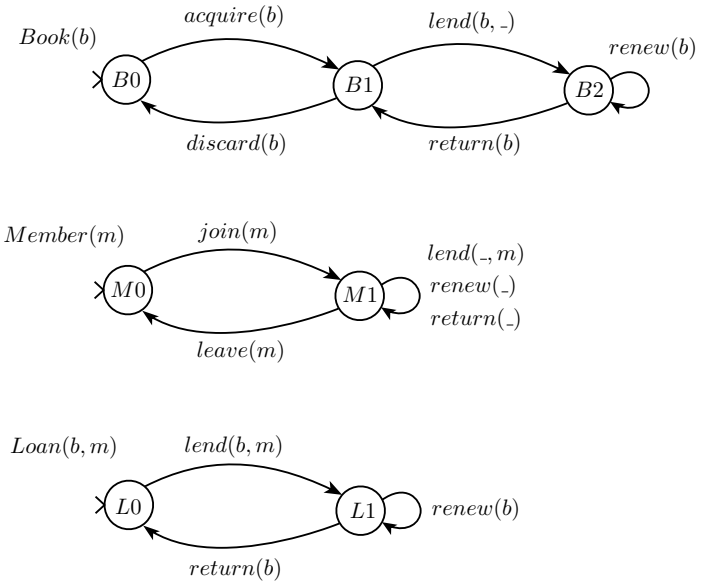

figure 3.2 – STE d'un système de gestion de bibliothèque

n, ici la taille du système est donnée par deux ensembles X et Y. Finalement, on peut représenter l'ensemble de tous les systèmes instanciés par l'expression :

$$\bigcup_{X,Y} \mathcal{S}(X, Y)$$

Un modèle similaire de STE à plusieurs paramètres est aussi utilisé dans [87, 86]. Dans cette thèse, les STE à plusieurs paramètres sont présentés essentiellement dans le but d'introduire une forme plus complexe de systèmes paramétrés. En effet, une approche similaire est développée dans la section suivante pour le modèle des ASTD.

3.2.2 Un modèle adapté aux systèmes d'information

Parmi les caractéristiques les plus intéressantes des ASTD pour la spécification de systèmes d'information, on retrouve les opérateurs de choix et d'entrelacement quantifiés. En effet, ils permettent de modéliser les processus répliqués d'un système paramétré et leurs interactions. Ainsi, si l'on considère le système d'information de gestion de bibliothèque gérant des membres, des livres et des emprunts, on utilisera l'entrelacement quantifié par exemple pour modéliser l'ensemble des processus de

chaque membre s'exécutant en concurrence. Cependant, dans la définition originale des ASTD, on doit spécifier un ensemble constant pour chaque opérateur quantifié. Cela signifie que le nombre de membres dans le système est fixé une fois pour toute. Afin de modéliser un système qui fonctionnerait quel que soit le nombre de membres, on doit considérer une spécification un peu plus abstraite et introduire des paramètres globaux aux ASTD. Dans la suite, on prendra des variables d'ensemble comme paramètres. Un exemple de paramètre est une variable dont l'ensemble des membres $\{m_1, m_2, \ldots, m_9\}$ est une instance. On utilisera ces variables dans la spécification des ASTD de choix et d'entrelacement quantifiés.

Définition 3.1. *Un ASTD paramétré (Parameterized ASTD, PASTD) $a(T_1, \ldots, T_k)$ est une expression de ASTD a munie des paramètres $T_1, \ldots, T_k$.*

- *Chaque paramètre T_i est associé à un ensemble de valeurs P_i appelé domaine de paramètre.*
- *Un paramètre T_i est une variable qui peut être instanciée par un sous-ensemble fini non vide de P_i. On note alors $a(V_1, \ldots, V_k)$ l'ASTD correspondant au PASTD instancié par les valeurs $V_1 \subseteq P_1, \ldots, V_k \subseteq P_k$.*
- *Les paramètres sont transférés aux sous-ASTD et utilisés comme ensembles de quantification pour les ASTD de choix et d'entrelacement quantifiés.*

Par exemple, la figure 3.3 présente un PASTD modélisant un système de gestion de bibliothèque. Le système gère les emprunts de livres par des membres. Il y a deux paramètres dans le système, soit T_m et T_b de domaines $P_m = \{m_1, m_2 \ldots\}$ et $P_b = \{b_1, b_2 \ldots\}$ respectivement, qui représentent l'ensemble des identifiants des membres et des livres.

Dans certains systèmes paramétrés comme les réseaux de Petri, les paramètres sont des entiers naturels et représentent le nombre de processus répliqués. Cependant, cette modélisation ne permet pas de rendre compte des interactions entre les différentes instances d'un système paramétré plus complexe. En effet, si l'on prend l'exemple de la bibliothèque, avec une variable d'entier comme paramètre, on pourrait spécifier le nombre de membres inscrits ou de livres acquis mais on ne pourrait pas représenter le fait que le membre m_1 ait emprunté le livre b_2 en particulier. C'est

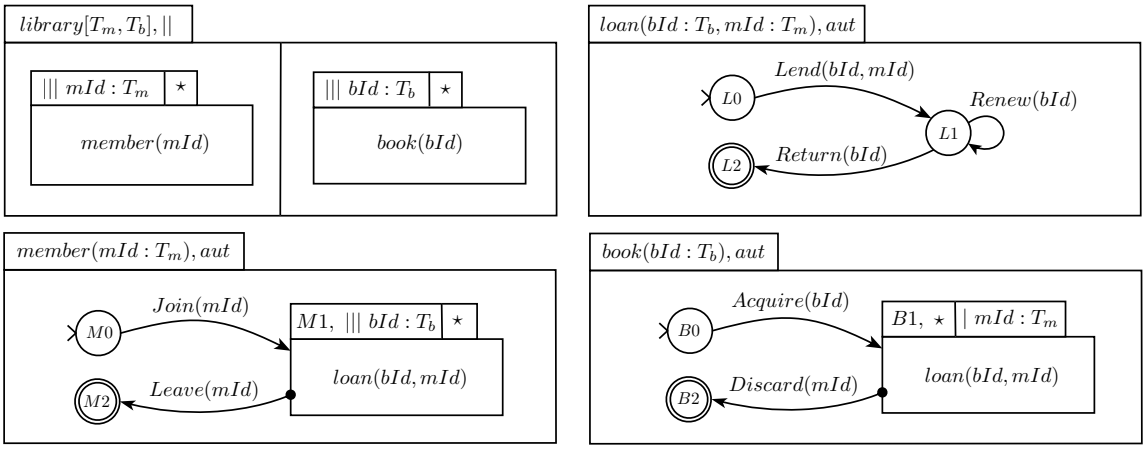

figure 3.3 – PASTD d'un système de gestion de bibliothèque

pourquoi dans nos PASTD, on utilise un ensemble d'identifiants comme une instance de paramètres plutôt qu'un entier. Ainsi, les PASTD sont utiles à la modélisation de systèmes d'information.

À chaque modèle d'ASTD ou de PASTD on associe, grâce à sa sémantique opérationnelle, un système de transitions. Dans le cas d'un PASTD, on obtient en général un système de transitions dont l'ensemble d'états est infini. En effet, si l'on note $\mathcal{S}_{a,\overrightarrow{V}}$ le système de transitions d'une instance de PASTD $a(\overrightarrow{T})$, où $\overrightarrow{V} \subseteq \overrightarrow{P}$ et $\overrightarrow{P}$ est un vecteur de domaines de paramètre, le système de transitions qui correspond au système paramétré global est donné par l'union de toutes les instances de système possibles $\mathcal{S}_a = \bigcup_{\overrightarrow{V}} \mathcal{S}_{a,\overrightarrow{V}}$. Cette union étant infinie si les domaines de paramètre sont infinis, le système résultant est alors aussi infini. On se référera à l'annexe A (définition A.13) pour plus de détails. Le chapitre suivant s'intéresse à la vérification de tels systèmes.

3.3 Évaluation des méthodes de vérification pour les systèmes d'information

Les systèmes d'information pouvant être assimilés à des systèmes paramétrés, on peut se demander si les techniques de *model checking* présentées précédemment peuvent être directement appliquées aux systèmes d'information. Cette section met en

évidence les particularités des systèmes d'information et les problèmes soulevés par les différentes techniques de vérification. On compare ainsi les avantages et inconvénients de chaque type de technique de vérification présentée dans le chapitre 2. On montre qu'aucune des techniques existantes pour le *model checking* de systèmes paramétrés ne s'applique facilement au cas des systèmes d'information, et on souligne les critères importants pour le choix d'une technique de vérification adéquate.

3.3.1 Techniques d'abstraction

L'abstraction par compteurs [79] est une abstraction trop forte dans le cadre des systèmes d'information. Le principal défaut de cette abstraction est la perte des identités des processus. En effet, il n'est plus possible de spécifier des propriétés qui concernent un processus en particulier et donc de suivre une évolution individuelle. De plus, les relations entres instances de classe sont difficilement représentables. Or, les systèmes d'information sont des systèmes dont les entités sont liées par des associations nombreuses et complexes. Ainsi, il n'est pas possible de vérifier une propriété du type *"un livre déjà emprunté ne peut l'être par un autre membre"*.

La méthode d'abstraction par environnement [20] semble plus adaptée à ce genre de systèmes que l'abstraction par compteurs. En effet, il est possible d'exprimer des propriétés portant sur un processus particulier dans ce cas. Cependant, cette méthode est conçue pour des systèmes composés d'une seule classe de processus et un seul paramètre. En outre, comme beaucoup d'autres méthodes d'abstraction, elle a pour défaut d'être incomplète : si le modèle abstrait ne vérifie pas la propriété alors on ne peut pas conclure que le modèle concret ne la vérifie pas non plus.

3.3.2 Techniques de *cutoff*

Les méthodes par *cutoff* sont en général des méthodes de vérification complètes. Cependant, en pratique la procédure peut devenir relativement longue puisqu'elle requiert la vérification de tous les systèmes instanciés par des valeurs inférieures au *cutoff*. La technique proposée par Emerson et Kahlon dans [34] permet de considérer plusieurs classes de processus. Elle s'applique à des systèmes possédant un certain type de gardes et à des propriétés $\mathrm{LTL}_{\setminus X}$ portant sur un unique processus ou une

paire de processus, éventuellement de classes différentes. Cette méthode permet de spécifier des propriétés de sûreté et d'accessibilité sur plusieurs classes de processus. La méthode de spécification des systèmes consiste en des sytèmes de transitions pourvus de gardes. Cependant, les types de gardes possibles sont trop restrictives pour représenter facilement les systèmes d'information. De plus, la méthode ne permet pas d'utiliser un même paramètre pour modéliser plusieurs processus différents. Ainsi les processus du type $loan(b, m)$, où $member(m)$ et $book(b)$ utilisent des paramètres liés m et b, ne peuvent être représentés correctement.

3.3.3 Techniques d'induction

Les techniques de vérification par induction imposent la construction d'invariants pour les preuves de correction. Elles sont donc limitées aux propriétés d'invariant. De plus, ces méthodes sont incomplètes, car il n'est pas toujours possible de trouver ces invariants. La méthode des *invisible invariants* [78] combine l'induction avec les méthodes de *cutoff* afin d'offrir une technique de preuve complètement automatisée. La méthode a été généralisée dans [7] et permet de gérer plusieurs types de données et plusieurs paramètres pour les systèmes. Cependant, la méthode de spécification des systèmes n'est pas pratique, notamment dans le cas des systèmes d'information.

3.3.4 Synthèse

Les techniques proposées dans la littérature pour vérifier les systèmes paramétrés sont nombreuses, mais aucune n'apporte une solution complète pour les systèmes d'information. Les différents travaux se restreignent à des cas particuliers de systèmes paramétrés qui ne correspondent pas aux systèmes d'information.

Du point de vue de la spécification de systèmes d'information, les méthodes présentées ne permettent pas de modéliser toute la complexité des systèmes d'information. Les techniques d'abstraction proposent des modèles trop simplifiés de spécification qui ne permettent pas la modélisation de plusieurs types d'entités ou leurs diverses interactions. D'une manière générale, l'abstraction est souvent trop forte. Par ailleurs, les méthodes de *cutoff* de [34] et d'induction de [78] imposent des restrictions trop fortes sur la forme des gardes à utiliser. Ainsi, il est préférable d'utiliser les modèles de

spécification avec paramètres présentés dans les sections 3.2.1 et 3.2.2 qui permettent de spécifier plus précisément le comportement des systèmes d'information.

Les méthodes de vérification basées sur des techniques de modélisation insuffisantes pour représenter les systèmes d'information ne sont pas pertinentes pour résoudre le problème soulevé dans cette thèse. De plus, les méthodes de vérification complètes et automatisables sont préférables aux autres. Comme aucune méthode de vérification n'existe pour la vérification d'ASTD, il devient intéressant de pouvoir en proposer une. Ainsi, nous proposerons dans le chapitre 4 une méthode de vérification d'ASTD paramétrés basée sur la méthode des WSTS étudiée dans la section 2.3.

3.4 Étude de l'accessibilité dans les systèmes de transitions paramétrés

On étudie dans cette section le problème de vérification d'une sous-classe particulière de systèmes paramétrés décrite dans la section 3.2.1 : les STE à un paramètre. Notamment, on s'intéresse aux problèmes de sûreté et d'accessibilité et on montre que la vérification d'une propriété de sûreté dans un STE est équivalente à un problème d'accessibilité dans un autre STE. On étudie par la suite le problème d'*accessibilité des états de contrôle* et celui de *couverture* pour les STE paramétrés. Ces problèmes sont résolus par la méthode des WSTS.

3.4.1 Problèmes de sûreté et d'accessibilité dans les STE

Rappelons qu'une propriété de sûreté Φ sur un alphabet Σ désigne un ensemble de traces d'exécution correctes $\Phi \subseteq \Sigma^\omega$ et peut s'exprimer par une formule de la logique temporelle linéaire. Elle se caractérise par le fait qu'elle soit falsifiable par l'existence d'une trace finie incorrecte. Formellement, on peut définir une propriété de sûreté de la façon suivante.

Définition 3.2 (Propriété de sûreté [8])**.** *Une propriété temporelle linéaire Φ sur Σ est appelée propriété de sûreté si pour tout mot $w \in \Sigma^\omega \setminus \Phi$, il existe un préfixe v de w tel que :*

$$\Phi \cap \{u \in \Sigma^\omega \mid v \in \mathrm{pref}(u)\} = \emptyset$$

L'ensemble de tels mots v constitue les mauvais préfixes pour Φ et est noté $\mathrm{BadPref}(\Phi)$.

$$\mathrm{BadPref}(\Phi) = \{v \in \Sigma^* \mid \forall u \in \Sigma^\omega \cdot v \in \mathrm{pref}(u) \implies u \notin \Phi\}$$

Prenons par exemple $\Sigma = \{a, b, c\}$ un alphabet et $\Phi = \{w \in \{a, b\}^\omega\}$ une propriété temporelle linéaire sur Σ. Cette propriété représente les mots ne contenant pas de lettre c. Cette propriété est une propriété de sûreté et son ensemble de mauvais préfixes est donné par $\mathrm{BadPref}(\Phi) = \{w \in \Sigma^* \mid \exists w' \in \Sigma^* \cdot w = w'c\}$.

Dans le cadre des STE, on peut définir une propriété de sûreté en se basant sur l'ordonnancement des actions d'une trace. Ainsi, de façon similaire à la vérification LTL d'automates, un STE $\mathcal{S}$ respecte une propriété de sûreté Φ, noté $\mathcal{S} \models \Phi$, si $tr(\mathcal{S}) \subseteq \Phi$. On peut montrer que le problème de la vérification de propriétés de sûreté se ramène à un problème d'accessibilité.

Définition 3.3 (Propriété d'accessibilité)**.** *Soit $\mathcal{S} = (Q, \Sigma, \to, I)$ un STE. Une propriété d'accessibilité se définit par un sous-ensemble d'états $F \subseteq Q$. On dit que F est accessible s'il existe $q_1 \in I$ et $q_2 \in F$ tels que $q_1 \xrightarrow{*} q_2$.*

STE simples

Soit $\mathcal{S} = (Q, \Sigma, \to, I)$ un STE et $\Phi \subseteq \Sigma^\omega$ une propriété de sûreté. On a $\mathcal{S} \models \Phi$ ssi $\forall w \in \mathrm{tr}(\mathcal{S}) \cdot w \in \Phi$ ssi $\forall w \in \Sigma^\omega \cdot w \in \mathrm{tr}(\mathcal{S}) \implies w \in \Phi$ ssi $\neg\exists w \in \Sigma^\omega \cdot w \in \mathrm{tr}(\mathcal{S}) \wedge w \notin \Phi$. Ainsi, pour falsifier $\mathcal{S} \models \Phi$, il suffit de trouver une trace $w \in \Sigma^\omega$ telle que : $w \in \mathrm{tr}(\mathcal{S})$ et $w \in \Sigma^\omega \setminus \Phi$. D'après la définition de propriété de sûreté, on peut aussi raisonner en terme de mauvais préfixes grâce au théorème suivant.

Théoreme 3.1 ([8])**.** *Soit $\mathcal{S}$ un STE sans blocage et Φ une propriété de sûreté. On a*

$$\mathcal{S} \models \Phi \iff \mathrm{tr_f}(\mathcal{S}) \cap \mathrm{BadPref}(\Phi) = \emptyset$$

Si Φ est une propriété de sûreté ω-régulière alors on peut représenter $\mathrm{BadPref}(\Phi)$ par un automate fini ainsi que $\mathrm{tr_f}(\mathcal{S}) \cap \mathrm{BadPref}(\Phi)$. Le problème de vérifier si $\mathcal{S} \models \Phi$ est donc équivalent à un problème d'accessibilité d'états finaux dans un automate. La propriété Φ est vérifiée si et seulement si l'accessibilité des états finaux échoue.

STE paramétrés

Soient $\mathcal{A}$ et $\mathcal{B}$ deux STE, $\mathcal{S}(n) = \mathcal{A} \parallel |||^n \mathcal{B}$ et Φ une propriété temporelle linéaire. Le problème de la vérification paramétrée consiste à déterminer si pour tout $n \geq 1$, $\mathcal{S}(n) \models \Phi$. Une façon simple d'aborder le problème de la vérification paramétrée est de vérifier graduellement les instances du système $\mathcal{S}(n)$ pour chaque entier n. Cependant, cette procédure de vérification pourrait ne jamais terminer dans le cas d'une propriété de sûreté vraie ou d'une propriété d'accessibilité fausse. Formulé autrement, le problème consiste à déterminer si $\bigcup_{n\in\mathbb{N}} \mathcal{S}(n) \models \Phi$. Le système $\bigcup_{n\in\mathbb{N}} \mathcal{S}(n)$ représente dans ce cas l'ensemble de tous les systèmes engendrés par toutes les valeurs de paramètre possibles. On peut, par ailleurs, vérifier que le système $\bigcup_{n\in\mathbb{N}} \mathcal{S}(n)$ est équivalent en termes de traces à $\mathcal{A} \parallel |||^+ \mathcal{B}$. En outre, il est aisé de voir que pour tout système $\mathcal{A} \parallel |||^+ \mathcal{B}$, on peut construire un système $\mathcal{A}' \otimes |||^+ \mathcal{B}'$ équivalent tel que $\alpha(\mathcal{A}') = \alpha(\mathcal{B}') = \alpha(\mathcal{A}) \cup \alpha(\mathcal{B})$. En effet, il suffit d'ajouter dans $\mathcal{A}$ des boucles de transition étiquetées par chaque action de $\alpha(\mathcal{B}) \setminus \alpha(\mathcal{A})$ sur chaque état et inversement. On peut donc considérer le problème $\mathcal{A}' \otimes |||^+ \mathcal{B}' \models \Phi$ de façon équivalente.

Montrons que l'on peut résoudre les problèmes de sûreté et d'accessibilité pour $\mathcal{A} \otimes |||^+ \mathcal{B}$. Soit $\mathcal{S} = \mathcal{A} \otimes |||^+ \mathcal{B}$ un STE, où $\mathcal{A}$ et $\mathcal{B}$ sont des STE finis et $\Phi \subseteq \Sigma^\omega$ une propriété de sûreté ω-régulière, tels que $\alpha(\Phi) \subseteq \alpha(\mathcal{A})$ avec $\alpha(\mathcal{A}) = \alpha(\mathcal{B})$. On cherche à déterminer si $\mathcal{S} \models \Phi$.

On a $\mathcal{S} \models \Phi$ ssi $\forall w \in \mathrm{tr}(\mathcal{S}) \cdot w \in \Phi$
ssi $\forall w \in \Sigma^\omega \cdot w \in \mathrm{tr}(\mathcal{A} \otimes |||^+ \mathcal{B}) \implies w \in \Phi$
ssi $\forall w \in \Sigma^\omega \cdot w \in \mathrm{tr}(\mathcal{A}) \cap \mathrm{tr}(|||^+ \mathcal{B}) \implies w \in \Phi$
ssi $\neg\exists w \in \Sigma^\omega \cdot w \in \mathrm{tr}(\mathcal{A}) \wedge w \in \mathrm{tr}(|||^+ \mathcal{B}) \wedge w \notin \Phi$

Pour falsifier $\mathcal{S} \models \Phi$, il suffit donc de trouver une trace $w \in \Sigma^\omega$ telle que :

$$w \in \mathrm{tr}(\mathcal{A}) \tag{3.1}$$

$$w \in \mathrm{tr}(|||^+ \mathcal{B}) \tag{3.2}$$

$$w \in \Sigma^\omega \setminus \Phi \tag{3.3}$$

Considérant que Φ est une propriété de sûreté, on peut utiliser le principe des mauvais

préfixes similairement au cas des STE simples. Pour falsifier Φ, il suffit alors de trouver une trace finie $w \in \Sigma^*$ telle que :

$$w \in \mathrm{tr_f}(\mathcal{A}) \tag{3.4}$$

$$w \in \mathrm{tr_f}(\left|\!\left|\!\right|\right|^+ \mathcal{B}) \tag{3.5}$$

$$w \in \mathrm{BadPref}(\Phi) \tag{3.6}$$

On sait que $\mathrm{tr}_f(\mathcal{A})$ et $\mathrm{BadPref}(\Phi)$ sont des langages réguliers. Le langage $\mathrm{tr_f}(\left|\!\left|\!\right|\right|^+ \mathcal{B})$, quant à lui, est un langage de *mélange simple* [45], de la forme $\sqcup\!\sqcup^+ L$ où L est le langage régulier de $\mathcal{B}$, et qui ont été démontrés comme étant inclus dans ceux des réseaux de Petri [45]. Le langage des réseaux de Petri comprenant les langages réguliers et étant clos par intersection, on peut en conclure que $\mathrm{tr_f}(\mathcal{A}) \cap \mathrm{tr_f}(\left|\!\left|\!\right|\right|^+ \mathcal{B}) \cap \mathrm{BadPref}(\Phi)$ est un langage des réseaux de Petri. Or, le problème du vide est décidable pour le langage des réseaux de Petri. On peut donc en déduire que les problèmes de vérification de propriétés de sûreté et d'accessibilité dans les systèmes paramétrés de type $\mathcal{A} \otimes \left|\!\left|\!\right|\right|^+ \mathcal{B}$ sont décidables.

Diverses formes d'accessibilité

On a montré, dans cette section, que dans les STE les problèmes de sûreté sont équivalents à des problèmes d'accessibilité et que ces problèmes sont décidables. Intuitivement, pour chaque propriété de sûreté, il suffit de construire l'automate de ses mauvais préfixes et en faire le produit synchrone avec le système à vérifier, puis de vérifier l'accessibilité des états interdits dans le système résultant. Cette méthode s'adapte facilement au cas des PASTD qui permettent à la fois la modélisation d'automates et la synchronisation. Ainsi, on considèrera, dans la section 3.4.2 et le chapitre 4, le problème de la vérification d'une propriété de sûreté dans les systèmes d'information comme un problème d'accessibilité d'un ensemble d'états donné. Plus précisément, dans le cadre de la méthode des WSTS, il s'agit de résoudre le problème de *couverture*.

Le *problème de couverture* est une variante du problème d'accessibilité. Afin de le définir, le système doit être muni d'une relation d'ordre ou de préordre $\leq$ sur l'espace d'états Q_S. Considérant un état $q \in Q_S$, le problème de couverture consiste alors

à déterminer s'il existe un état $q' \in Q_S$ qui soit accessible et tel que $q \leq q'$. On s'intéressera à ce problème dans le cadre de la méthode des WSTS.

Une autre variante du problème d'accessibilité est le problème de l'*accessibilité des états de contrôle*. Considérons les STE $\mathcal{A} = (Q_A, \Sigma_A, \rightarrow_A, I_A)$ et $\mathcal{B} = (Q_B, \Sigma_B, \rightarrow_B, I_B)$ et le STE $\mathcal{S} = (Q_S, \Sigma, \rightarrow, I_S)$ tels que $\mathcal{S} = \mathcal{A} \parallel \left|\!\left|\!\right|^{+} \mathcal{B}$. L'ensemble Q_S des états de $\mathcal{S}$ est formé du produit cartésien de l'ensemble fini Q_A des *états de contrôle* et de l'ensemble $\mathbb{N}^{Q_B}$ des multiensembles de support Q_B des *données*. On note $\{\!\{x_1, x_2 \ldots\}\!\}$ le multiensemble. Un état $s \in Q_A \times \mathbb{N}^{Q_B}$ est donc représenté par un couple de la forme $(q_A, \{\!\{q_{B_1}, \ldots, q_{B_n}\}\!\})$, où $q_A \in Q_A$ et pour tout $i \in 1..n$, $q_{B_i} \in Q_B$. Une *propriété d'accessibilité* dans $\mathcal{S}$ peut s'exprimer sous la forme d'un sous-ensemble $F \subseteq Q_S$ d'états de $\mathcal{S}$ mais aussi par un sous-ensemble $F_A \subseteq Q_A$ d'états de contrôle. On parle d'accessibilité des états de contrôle F_A lorsque l'on cherche à déterminer l'accessibilité de l'ensemble des états $F \subseteq Q_S$ tel pour tout $s = (q_A, d) \in F$, on a $q_A \in F_A$. Il s'agit en pratique de n'indiquer que les états à atteindre dans $\mathcal{A}$.

3.4.2 Application des WSTS aux STE paramétrés

Dans cette section, on applique la méthode des WSTS présentée dans la section 2.3 du chapitre 2 afin de résoudre le problème de l'accessibilité des états de contrôle pour un STE à un paramètre. Considérons deux STE $\mathcal{A} = (Q_A, \Sigma_A, \rightarrow_A, I_A)$ et $\mathcal{B} = (Q_B, \Sigma_B, \rightarrow_B, I_B)$ et $\Sigma = \Sigma_A \cup \Sigma_B$ et le STE paramétré $\mathcal{S} = (Q_S, \Sigma, \rightarrow, I_S)$ tels que $\mathcal{S} = \mathcal{A} \otimes \left|\!\left|\!\right|^{+} \mathcal{B}$. On définit un ordre sur $Q_A \times \mathbb{N}^{Q_B}$ notée $\preceq$ de la façon suivante.

Définition 3.4. *Soient $s, s' \in Q_S$, $q, q' \in Q_A$ et $d, d' \in \mathbb{N}^{Q_B}$ tels que $s = (q, d)$ et $s' = (q', d')$, on a $s \preceq s'$ si et seulement si $q = q'$ et $d \subseteq d'$.*

Intuitivement, pour deux éléments s et s' de $Q_A \times \mathbb{N}^{Q_B}$, si $s \preceq s'$ alors s' est composé d'au moins toutes les instances de processus présentes dans s et dans les mêmes états locaux.

Proposition 3.1. *$\preceq$ est un WQO.*

Démonstration. D'après le lemme de Dickson [29], l'inclusion multiensemble $\subseteq$, qui peut être vue comme l'ordre produit sur les vecteurs d'entiers, est un WQO donc $\preceq$ aussi. □

Notons que pour tout couple d'éléments $(s, s') \in Q_S$, on peut déterminer si $s \preceq s'$ ou non. On dit que $\preceq$ est décidable.

On peut ainsi considérer le problème de l'accessibilité des états de contrôle comme un problème de couverture. En effet, vérifier l'accessibilité d'un état de contrôle $q_A \in Q_A$ revient à vérifier la couverture de tout état $(q_A, \{\!\{q_B\}\!\})$, où $q_B \in Q_B$. On note $\uplus$ l'union additive de multiensembles.

Proposition 3.2 (Monotonie)**.** *Soient $s_1, s_2, s_3 \in Q_S$ et $a \in \Sigma$. Si $s_1 \preceq s_2$ et $s_1 \xrightarrow{a} s_3$, alors il existe $s_4 \in Q_S$ tel que $s_3 \preceq s_4$ et $s_2 \xrightarrow{a} s_4$.*

Démonstration. Soient $s_i = (q_i, d_i) \in Q_S$ pour tout $1 \leq i \leq 3$ tels que $s_1 \preceq s_2$ et $s_1 \xrightarrow{a} s_3$. Si on peut effectuer une transition sur a de s_1 vers s_3 alors on peut aussi en effectuer une depuis s_2 par la définition de la relation de transition. Plus particulièrement, il existe un état $s_4 = (q_4, d_4)$ tel que $s_3 \preceq s_4$ et $s_2 \xrightarrow{a} s_4$. On a $q_4 = q_3$ et $d_4 = (d_2 \setminus (d_1 \setminus d_3)) \uplus (d_3 \setminus d_1)$. Intuitivement, $d_1 \setminus d_3$ et $d_3 \setminus d_1$ représentent la composante de $\mathcal{B}$ qui se déplace. □

Des résultats précédents, on peut déduire que le système ordonné $\mathcal{S}$ est bien structuré.

Proposition 3.3. *$\mathcal{S} = \mathcal{A} \otimes \|\|^+ \mathcal{B}$ est un WSTS.*

Démonstration. Le système respecte les conditions (propositions 3.1 et 3.2). □

Le prédicat suivant permet de définir une base finie de prédécesseurs d'un ensemble d'états de $\uparrow s$.

Définition 3.5. *Soit $s \in Q_S$. $pb(s)$ est l'ensemble des s' respectant le prédicat suivant.*

$$
\begin{aligned}
predbase(s', s) \equiv \exists q_A, q'_A \in Q_A \cdot \exists q_B, q'_B \in Q_B \cdot \exists d, d' \in \mathbb{N}^{Q_B} \cdot \exists a \in \Sigma \cdot \\
s = (q_A, d) \wedge s' = (q'_A, d') \wedge q'_A \xrightarrow{a}_A q_A \wedge q'_B \xrightarrow{a}_A q_B \wedge q'_B \in d' \wedge \\
((q_B \in d \wedge d \setminus \{\!\{q_B\}\!\} = d' \setminus \{\!\{q'_B\}\!\}) \vee \qquad (3.7) \\
d' = d \uplus \{\!\{q'_B\}\!\}) \qquad (3.8)
\end{aligned}
$$

Pour chaque état $s \in Q_S$, $pb(s)$ peut générer deux types d'états : d'une part (3.7) les prédécesseurs directs de l'état s dans le système de transitions, d'autre part (3.8)

les prédécesseurs des états composés d'une instance de $\mathcal{B}$ supplémentaire (donc plus grands que s) et sur laquelle s'effectue la transition. Un état de $pb(s)$ correspond donc à un prédécesseur de $\uparrow s$. De plus, $pb(s)$ représente un ensemble minimal de prédécesseurs de s dans le sens où sa clôture par le haut donne l'ensemble de tous les prédécesseurs de s. Il s'agit maintenant de montrer que $pb(s)$ est une pred-base effective pour le système.

Lemme 3.1. *Soit E un ensemble d'états de $\mathcal{S}$. Si $s \in Pred(E)$, alors $\uparrow s \subseteq \uparrow Pred(E)$.*

Démonstration. Soient $s \in Pred(E)$ et $s' \in \uparrow s$. Alors il existe $t \in E$ tel que $s \to t$ et on a $s \preceq s'$. Par hypothèse de monotonie (proposition 3.2), il existe t' tel que $t \preceq t'$ et $s' \to t'$ donc $s' \in \uparrow Pred(t)$. Par conséquent, $s' \in \uparrow Pred(E)$. □

Lemme 3.2. *Pour tout $s \in Q_S$, $pb(s)$ est une base finie de $\uparrow Pred(\uparrow s)$.*

Démonstration. Le système étant à branchement fini, $pb(s)$ est clairement fini car il définit les prédécesseurs d'un ensemble fini d'état. Il reste à montrer que $\uparrow pb(s) = \uparrow Pred(\uparrow s)$.

- $\subseteq$: Soit $t \in pb(s)$. On a soit (3.7) $t \in Pred(s)$, soit (3.8) $t \in Pred(\uparrow s)$ par définition de $pb(s)$ donc $t \in Pred(\uparrow s)$. Et par le lemme 3.1, $\uparrow t \subseteq \uparrow Pred(\uparrow s)$.

- $\supseteq$: Soit $t \in Pred(\uparrow s)$. Il suffit de montrer qu'il existe $t' \preceq t$ tel que $t' \in pb(s)$. On sait qu'il existe u tel que $s \preceq u$ et $t \to u$ car $t \in Pred(\uparrow s)$. Posons $s = (q_s, d_s)$, $t = (q_t, d_t)$ et $u = (q_u, d_u)$. Alors il existe $a \in \Sigma$ et $q_1, q_2 \in Q_A$ tels que $q_1 \xrightarrow{a}_A q_2$, $q_t \xrightarrow{a}_A q_u$ et $d_t \setminus \{\!\{q_1\}\!\} = d_u \setminus \{\!\{q_2\}\!\}\}$. De plus, $d_s \subseteq d_u$. Par disjonction de cas :

  - Cas où $q_2 \in d_s$. Posons $t' = (q_{t'}, d_{t'})$ tel que $q_{t'} = q_t$ et $d_{t'} \setminus \{\!\{q_1\}\!\}\} = d_s \setminus \{\!\{q_2\}\!\}$. On a $d_{t'} \setminus \{\!\{q_1\}\!\} = d_s \setminus \{\!\{q_2\}\!\} \subseteq d_u \setminus \{\!\{q_2\}\!\} = d_t \setminus \{\!\{q_1\}\!\}$. Donc $d_{t'} \subseteq d_t$. De plus, $q_{t'} = q_t \xrightarrow{a}_A q_u = q_s$. Donc $q_{t'} \xrightarrow{a}_A q_s$. Par conséquent, $t' \preceq t$ et $t' \in pb(s)$ (de type (3.7)).
  - Cas où $q_2 \notin d_s$. Posons $t' = (q_{t'}, d_{t'})$ tel que $q_{t'} = q_t$ et $d_{t'} = d_s \uplus \{\!\{q_1\}\!\}$. On a $d_{t'} = d_s \uplus \{\!\{q_1\}\!\} \subseteq d_u \uplus \{\!\{q_1\}\!\}$ et $d_t = d_u \uplus \{\!\{q_1\}\!\} \setminus \{\!\{q_2\}\!\}$. De plus, $d_{t'} \setminus \{\!\{q_2\}\!\} = d_{t'}$ car $q_2 \notin d_s$. On a alors $d_{t'} = d_{t'} \setminus \{\!\{q_2\}\!\} \subseteq d_u \uplus \{\!\{q_1\}\!\} \setminus \{\!\{q_2\}\!\} = d_t$. Donc $d_{t'} \subseteq d_t$. De plus, $q_{t'} \xrightarrow{a}_A q_s$. Par conséquent, $t' \preceq t$ et $t' \in pb(s)$ (de type (3.8)).

□

Proposition 3.4. *$\mathcal{A} \otimes |||^+ \mathcal{B}$ possède une pred-base effective.*

Démonstration. $pb(s)$ est fini et clairement calculable d'après sa définition et définit une base finie de prédécesseurs d'après le lemme 3.2. □

Ainsi, il est possible de déterminer si un état s est couvrable dans $\mathcal{S}$ par la procédure de recherche en arrière décrite dans la section 2.3. On calcule $\uparrow Pred^*(\uparrow s)$ et on vérifie s'il contient un état initial.

Théoreme 3.2. *Le problème d'accessibilité des états de contrôle est décidable pour $\mathcal{A} \otimes |||^+ \mathcal{B}$.*

Démonstration. $\mathcal{A} \otimes |||^+ \mathcal{B}$ est un WSTS et possède une pred-base effective et $\preceq$ est décidable. Le problème d'accessibilité des états de contrôle est un problème de couverture. □

Dans le cadre des STE à plusieurs paramètres, le problème est plus complexe à résoudre par la méthode des WSTS. En effet, le même ordre ne peut pas être appliqué et le WQO est plus difficile à déterminer. On présentera dans le chapitre 4, le problème de l'accessibilité du point de vue des WSTS pour un modèle plus complexe que les STE à plusieurs paramètres et qui est celui des PASTD, présenté dans la section 3.2.2.

3.5 Conclusion

Ce chapitre a permis de mettre en évidence les caractéristiques des systèmes d'information importantes pour notre étude. L'appartenance à la classe des systèmes paramétrés et la présence des multiples interactions entre les différentes entités font des systèmes d'information des systèmes complexes à modéliser et à vérifier. Les modèles de systèmes paramétrés existants dans la littérature sont, pour la plupart, insuffisants pour représenter les systèmes d'information. Le modèle PASTD, proposé ici, permet de spécifier des systèmes à plusieurs paramètres, et donc de gérer les différentes entités d'un système d'information. Il permet aussi de représenter les associations du système.

3.5. Conclusion

Les méthodes de vérification paramétrée introduites dans le chapitre précédent ne sont pas adaptées aux systèmes d'information. En effet, elles imposent aux modèles des contraintes trop restrictives. Par ailleurs, certaines techniques sont incomplètes ou non automatisables. L'idéal serait donc de développer une méthode de vérification propre aux PASTD, et qui prendrait en compte leurs caractéristiques. L'analyse de couverture des WSTS est une méthode de vérification générale pour les systèmes infinis, et qui fonctionne pour la vérification de STE paramétrés. En outre, on constate que les propriétés usuelles que l'on souhaite vérifier pour les systèmes d'information sont des propriétés de sûreté et d'accessibilité. Le problème de sûreté pouvant être réduit à un problème de couverture, comme démontré ici dans le cas des STE paramétrés, cela justifie le choix de la méthode basée sur les WSTS pour vérifier des systèmes d'information modélisés par des PASTD. Ainsi, pour chaque propriété de sûreté Φ à vérifier sur un PASTD a, on peut construire un PASTD b résultant de la synchronisation de a et d'un automate pour BadPref(Φ). La propriété de sûreté est vérifiée si et seulement si l'accessibilité des états finaux de l'automate est falsifiée. De plus, ce problème d'accessibilité peut être formulé par un problème de couverture dans b. Le chapitre suivant s'intéressera donc à l'application de la méthode des WSTS pour la vérification de couverture aux PASTD.

Chapitre 4

Vérification paramétrée de systèmes d'information monotones

Résumé

Ce chapitre examine le problème de la vérification de systèmes d'information sous l'angle de la vérification paramétrée. On propose une méthode de modélisation des systèmes à multiples paramètres basée sur les ASTD et de vérification de ces systèmes basée sur la théorie des WSTS. On définit par ailleurs un nouveau cadre de travail permettant de prouver la condition de pred-base effective des WSTS.

Commentaires

Ce chapitre a été écrit sous la forme d'un article soumis au journal Formal Aspects of Computing. Il comporte les principales contributions de cette thèse. On y rappelle la syntaxe et la sémantique des ASTD et des PASTD ainsi que la théorie des WSTS déjà présentées dans les chapitres précédents. Cependant, on y fait le choix d'une syntaxe simplifiée pour des raisons de clarté et de concision. Un rapport technique en annexe restitue la syntaxe originale ainsi que les preuves manquantes. Le travail de recherche a été encadré par Marc Frappier et Amel Mammar et fait en collaboration avec Alain Finkel. J'ai rédigé et apporté l'essentiel des résultats de cet article.

Parameterized Verification of Monotone Information Systems

Raphaël Chane-Yack-Fa and Marc Frappier
GRIL, Université de Sherbrooke,
Sherbrooke, Québec, Canada

Amel Mammar
SAMOVAR, CNRS, Télécom SudParis,
Evry, France

Alain Finkel
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay,
Paris, France

Keywords: model checking; parameterized verification; information systems; process algebra; well-structured transition systems; well-quasi-ordering; coverability

Abstract

In this article, we study the information system verification problem as a parameterized verification one. Informations systems are modeled as multi-parameterized systems in a formal language based on the Algebraic State-Transition Diagrams (ASTD) notation. Then, we use the Well Structured Transition Systems (WSTS) theory to solve the coverability problem for an unbounded ASTD state space. Moreover, we define a new framework to prove the effective pred-basis condition of WSTSs, *i.e.* the computability of a base of predecessors for every states.

4.1 Introduction

Information Systems (IS) have been evolving for years now and the validation of such systems has become a prominent topic. An IS can be viewed as a complex system composed of a set of related entities and which can manage a substantial amount of data. They are used for various applications such as banking services, online sales or data warehouse [10]. In some cases, they are critical systems and one must ensure that the system is reliable. Therefore, the verification of IS specifications is essential. Furthermore, in this paper, we will see IS as parameterized systems which model systems with an unbounded number of components. For instance, the specification of a library management system, which is an IS, may abstract the number of books which interact in the library, and thus can be represented by a parameterized system. The aim of this paper is to find suitable techniques to specify and verify ISs from the point of view of parameterized systems.

Parameterized verification is a widely addressed topic in the literature [68, 34, 20, 69, 87, 86, 48, 55, 2, 28, 58] and we can explore the relation between parameterized systems and ISs. Among the characteristics of ISs, we can notice the multiplicity of entity types and the complex relationships between them. For example, in the library system we can model a book entity and a member entity together with a loan relationship. The number of instances of each entity is unbounded: that justifies the need for a parameterized specification. In general, an IS has more than one entity and thus multiple parameters. However, in most of the methods in the literature, parameterized verification is limited to some specific models like systems with only one parameter or without any relationship. We propose in this paper a method that handles systems with many parameters and relationships. To model ISs, we use an extension of the *Algebraic State-Transition Diagram* (ASTD) notation, proposed in [42], which we call *Parameterized ASTD* (PASTD) and which can model many related entities. As for the verification we based our method on the *Well-Structured Transition Systems* (WSTS) framework [39] which can verify monotone infinite systems equipped with a well-quasi-order.

In order to use the WSTS framework, we make some hypothesis on ISs that are essential to prove the monotony of the system and the termination of the verification

procedure. First, we assume that the specification of an IS cannot synchronize an arbitrary number of processes as it would break the monotony condition, which states that a similar configuration with more entity instances must be able to fire a similar transition. Moreover, to ensure termination, we devise some structural conditions on the states. However, unlike [39, 28, 69, 58], we deal with those conditions separately as we describe a general semi-decision procedure working for a large number of cases. This leads us to the definition of a new framework called *Ranked Monotone Transition System* (RMTS) which is useful to show the effectiveness condition of WSTS. As a result, we prove that PASTDs are RMTSs.

This paper is structured as follows. Section 4.2 presents the ASTD notation and its PASTD extension. In Section 4.3, we recall the WSTS theory and show how to apply it to PASTDs by determining adequate conditions. Section 4.4 presents the RMTS framework and proves that PASTDs are RMTSs. The related work is presented in Section 4.5 and Section 4.6 concludes.

4.2 A Visual Process Algebra

An *Algebraic State-Transition Diagram* (ASTD) [42] is a graphical notation combining automata, statecharts [49] and process algebras to describe complex dynamic systems like information systems. ASTDs are closely related to process algebras like CSP [51], CCS [71], ACP [9], LOTOS [15] and EB^3 [43]. Essentially, they are like a process algebra with hierarchical automata as elementary process expressions. Automata can be combined freely with process algebra operators. ASTDs have a structured operational semantics in the Plotkin style, which was first used by Milner for CCS and later on for LOTOS and CSP [83]. They are recursively defined structures and include many types like automaton, synchronization, quantified choice and quantified interleaving.

ASTDs are useful to model information systems as they provide a concise, visual and formal mechanism for specifying all the scenarios of an information system [42]. For example, they make explicit the handling of instances of information system entities by using quantifications. Furthermore, the model has been used in [32] to specify access control and security policies.

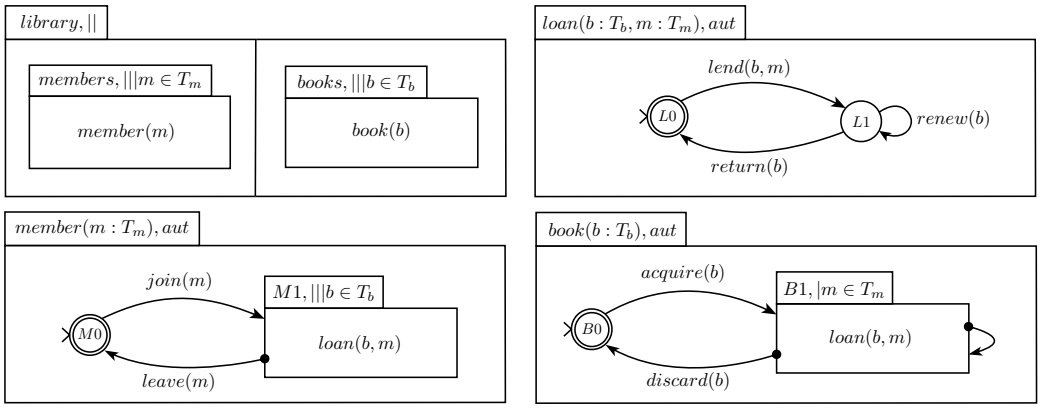

Figure 4.1 – Example of a library system

For instance, an ASTD model of a library system is given in Figure 4.1. The system manages loans of books by members. Books and members are the entities of the library system. The ASTD on the top left corner, whose name is *library*, is a synchronization between two other ASTDs, which consist in a quantified interleaving on the set T_m of members for the the process *member*, and a quantified interleaving on the set T_b of books for the process *book*. The process *book*(b) is described by the ASTD on the bottom right corner. A book is acquired by the library. It can be discarded if it is not lent. If acquired, a book b can "choose" a member m to run the process $loan(b, m)$. The member process is similar except that a member m runs the $loan(b, m)$ process for every book.

4.2.1 Parameterized ASTD

In this paper, we restrict ourselves to a fragment of the ASTD language. The graphical and textual notation of the original ASTDs are detailed in [41].

Let us denote a labeled tree by an expression thanks to the grammar $Tree ::= Node \mid Node[Tree, \ldots, Tree]$.

Definition 4.1 (ASTD expression)**.** *ASTD expressions are defined inductively by the following grammar:*

$\boldsymbol{F} ::= \mathcal{A}$ *(automaton)*
$\quad \mid \boldsymbol{F} \,||_{\Delta}\, \boldsymbol{F}$ *(synchronization)*

$|\ |_{x \in T} \boldsymbol{F}$ *(quantified choice)*
$|\ |||_{x \in T} \boldsymbol{F}$ *(quantified interleaving)*

where:

- $\mathcal{A}$ *is an Automaton ASTD* $(Q, \Sigma, \delta, q_0, Q_F)$ *such that* Q *is a finite set of states where for each* $q \in Q$, q *is either an elementary state or a composite state, i.e. another ASTD expression,* Σ *a set of labels,* δ *a labeled transition relation,* $q_0 \in Q$ *an initial state,* $Q_F \subseteq Q$ *a set of final states; note that a transition can be final (represented by a big dot at the origin of the arrow), i.e. it can be fired only from final states of the component ASTDs, or non-final, i.e. it can be fired at any time;*

- $\boldsymbol{F} ||_{\Delta} \boldsymbol{F}$ *denotes a Synchronization between two component ASTDs running concurrently by executing events, whose labels are in* Δ*, at the same time and interleaving the other events; if* Δ *is empty we denote the operator by* $|||$ *and if* Δ *is omitted, the processes synchronize on the set of shared labels;*

- *a Quantified Choice ASTD* $|_{x \in T} \boldsymbol{F}$ *allows us to pick a value* v *from a finite set* T *and execute its component ASTD* $\boldsymbol{F}$*, where every occurrence of the variable* x *is replaced by the value* v*;*

- *a Quantified Interleaving ASTD* $|||_{x \in T} \boldsymbol{F}$ *allows us to execute as many interleaving instances of* $\boldsymbol{F}$ *as the number of values in* T*, where each instance is executed such that* x *is replaced by the corresponding value.*

In the following, we introduce the following tree-like notation, which is easier to manipulate:

$\boldsymbol{F} ::= \mathcal{A}[q_1[\boldsymbol{F'}], \ldots, q_k[\boldsymbol{F'}]]$ *(one subtree for each state* $q_i \in Q$*)*
$\quad | \quad ||_{\Delta}[\boldsymbol{F}, \boldsymbol{F}]$
$\quad | \quad |_{x \in T}[\boldsymbol{F}]$
$\quad | \quad |||_{x \in T}[\boldsymbol{F}]$

$\boldsymbol{F'} ::= \boldsymbol{F}$
$\quad | \quad \epsilon$

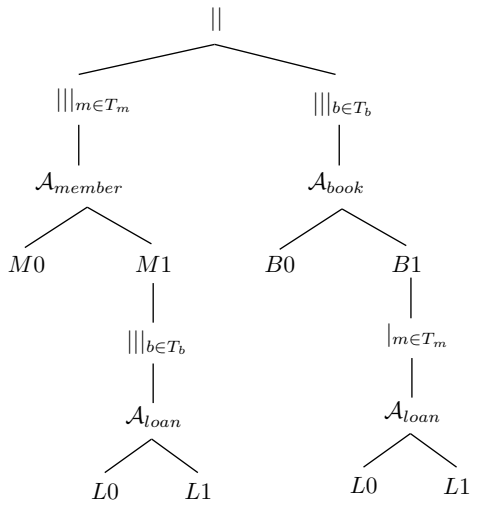

Figure 4.2 – The tree representation of an ASTD

Note that ϵ is used to create elementary automaton states. Figure 4.2 shows the tree notation of the ASTD of Figure 4.1, where $\mathcal{A}_{member}$, $\mathcal{A}_{book}$ and $\mathcal{A}_{loan}$ are the automata for member, book and loan processes respectively.

In this fragment of the ASTD language, we make two notable modifications from the original definition. First, we do not allow recursion, so that an ASTD specification is a tree-like structure. Moreover, we do not allow quantified synchronizations and we weaken the quantified interleaving ASTD by allowing the process to be in a final state at any time, *i.e.* there is no final synchronization between all interleaving processes. Indeed, the property of monotony, that we will explain in Section 4.3.2, is easier to obtain without quantified synchronizations.

The quantified operators (interleaving and choice) allow us to model replication of processes and complex interactions between them. For instance, in the example of the library system, we use the quantified interleaving to model many replicated processes running concurrently. To model a library that works with any number of members and books, we can consider a more abstract specification that takes the sets of members and books as parameters. Indeed, we allow quantification sets to be variables in quantified choice and quantified interleaving. We call those variables the parameters of the system.

Definition 4.2 (Parameterized ASTD)**.** *A Parameterized ASTD (PASTD) is a triple*

$(F, \overrightarrow{T}, \overrightarrow{P})$ where F is an ASTD expression, $\overrightarrow{T} = (T_1, \ldots, T_n)$ is a vector of n variables, called parameters, and $\overrightarrow{P} = (P_1, \ldots, P_n)$ is a vector of n sets of elements, called parameter domains. Each variable T_i represents a finite subset of the possibly infinite set P_i and appears in the expression of F as quantification set for quantified choice or quantified interleaving.

In a PASTD $(F, \overrightarrow{T}, \overrightarrow{P})$, remark that the ASTD F does not correspond to a concrete transition system as it contains the variables $\overrightarrow{T}$. However, if we choose a sequence of sets $R_1 \subseteq P_1, \ldots, R_n \subseteq P_n$, the substitution of $\overrightarrow{T}$ by $\overrightarrow{R}$ in F does represent a transition system. Intuitively, a PASTD represents the union of all possible instantiations of the expression F, which may correspond to an infinite system. For instance, in the library system, we use two parameters T_m and T_b, with parameter domains $P_m = \{m_1, m_2 \ldots\}$ and $P_b = \{b_1, b_2 \ldots\}$, respectively the sets of member and book identifiers.

As a PASTD describes a dynamical system, for each PASTD, we can define a set of PASTD states. A state is given by a tree structure.

Definition 4.3 (ASTD state)**.** *ASTD states are defined inductively by the following grammar:*

$\boldsymbol{S} ::= (\mathsf{aut}_\circ, q)$		*(automaton, elementary state q)*
$\mid$	$(\mathsf{aut}_\circ, q)[\boldsymbol{S}]$	*(automaton, composite state q)*
$\mid$	$\|\|_\circ[\boldsymbol{S}, \boldsymbol{S}]$	*(synchronization)*
$\mid$	$\|:_\circ$	*(quantified choice, value not chosen yet)*
$\mid$	$\|:_\circ[p[\boldsymbol{S}]]$	*(quantified choice, p is chosen)*
$\mid$	$\|\|\|:_\circ[p_1[\boldsymbol{S}], \ldots, p_k[\boldsymbol{S}]]$	*(quantified interleaving on $\{p_1, \ldots, p_k\}$, $k \in \mathbb{N}_1$)*

For a PASTD A, we denote by $\mathcal{T}_A$ the set of all well-formed states of A.

An ASTD state is represented by a tree where each node is labeled either by the type of state or by a value from quantification sets. If a type of state includes a sub-state in its definition, then it is represented by a child node. Furthermore, we split up the nodes for quantifications so that the values appear in child nodes.

Let us denote an undirected graph by a pair (V, E) such that V is a set of vertices and E a set of edges, where an edge is a pair of vertices. We call labeled graph a

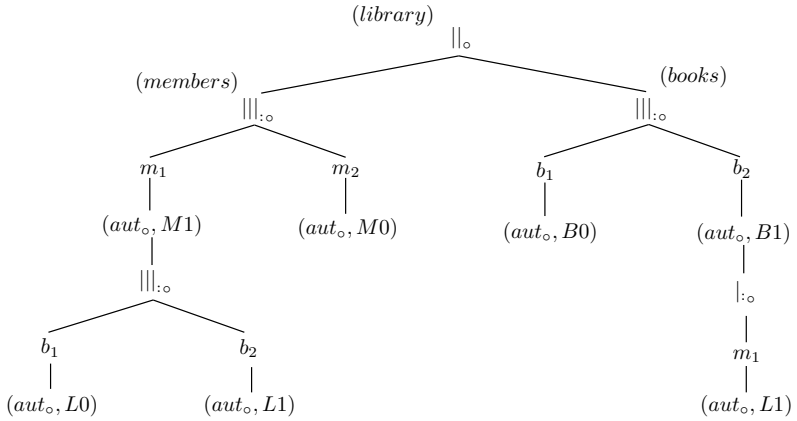

Figure 4.3 – A state of the library system

triple (V, E, λ) where (V, E) is a graph and $\lambda : V \to \Lambda$ a labeling function with Λ a set of labels. A labeled tree can be given by an expression $Tree ::= Node \mid Node[Tree, \ldots, Tree]$ or equivalently by a labeled acyclic connected graph (V, E, λ). In the following, we will sometimes represent an ASTD state by a graph (V, E, λ). We denote by $Im(f)$ the image of the function $f : X \to Y$, *i.e.* $Im(f) = \{y \in Y \mid \exists x \in X \cdot f(x) = y\}$.

Definition 4.4. *A PASTD state is always associated with an ASTD expression with which it must be consistent. For a PASTD* $A = (F, (T_1, \ldots, T_n), (P_1, \ldots, P_n))$ *and a state* $s = (V, E, \lambda) \in \mathcal{T}_A$*, we denote by* $val(s) = (R_1, \ldots, R_n)$ *the sets of elements appearing in the state* s*, i.e.* $R_i = P_i \cap Im(\lambda)$ *for all* $i \in 1..n$*.*

For instance, consider the library system $(F, (T_m, T_b), (P_m, P_b))$ of Figure 4.1. Let the state s consist of two members m_1 and m_2 and two books b_1 and b_2, where the book b_2 is borrowed by the member m_1 as depicted in Figure 4.3. We have $val(s) = (\{m_1, m_2\}, \{b_1, b_2\})$. In this state, the member m_1 has joined the library and has borrowed the book b_2. The member m_2 has not joined the library yet; similarly, the book b_1 has not been acquired yet; these two cases are represented by the initial state of the member and book automata, respectively.

Remark that we allow an ASTD state to represent a more abstract state by omitting some branches of the quantified interleaving and by considering that the local configuration of some entities is unknown. See for example Figure 4.4 where the book

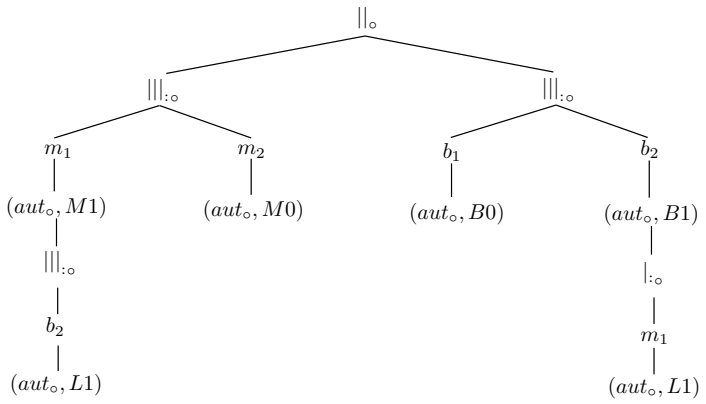

Figure 4.4 – An abstract state

b_1 is omitted in the quantified interleave of the state $M1$ in member process m_1.

The operational semantics of PASTD consists of a set of inference rules defined inductively on the ASTD states. For a PASTD A, we denote the corresponding transition relation on $\mathcal{T}_A$ by $\rightarrow$ and we obtain the transition system $\mathcal{S}_A = (\mathcal{T}_A, \rightarrow)$. Intuitively, $||_\circ$ makes a transition if both its sub-branches are able to do the transition on the same event. $|||_{:\circ}$ can do the transition if one of its branches can and $|_{:\circ}$ if its sub-branch can. Finally, an automaton state fires a transition either by a classical automaton transition or within a sub-state. Figure 4.5 shows an example of transition in the library system where the member m_1 returns the book b_2. The branches involved in the transition are depicted by bold strokes and the notable change in red. For a complete description of the operational semantics of ASTDs and PASTDs, see [41] and Appendix A (Definition A.6).

4.2.2 Expressiveness of PASTD

Considering the previous extension of ASTDs with parameters, we can show that PASTDs are more expressive than some types of infinite systems like *Petri Nets* [76] or *Vector Addition Systems with States* (VASS) [53]. More precisely, we remark that PASTDs can simulate VASSs with resets.

An n-dimensional VASS with resets (RVASS) is a finite directed graph (V, E) with arcs labeled by vectors of integers or a reset symbol r, *i.e.* $E \subseteq V \times (\mathbb{Z} \cup \{r\})^n \times V$ together with an initial vertex $p_i \in V$ and an initial natural vector $u_i \in \mathbb{N}^n$. A

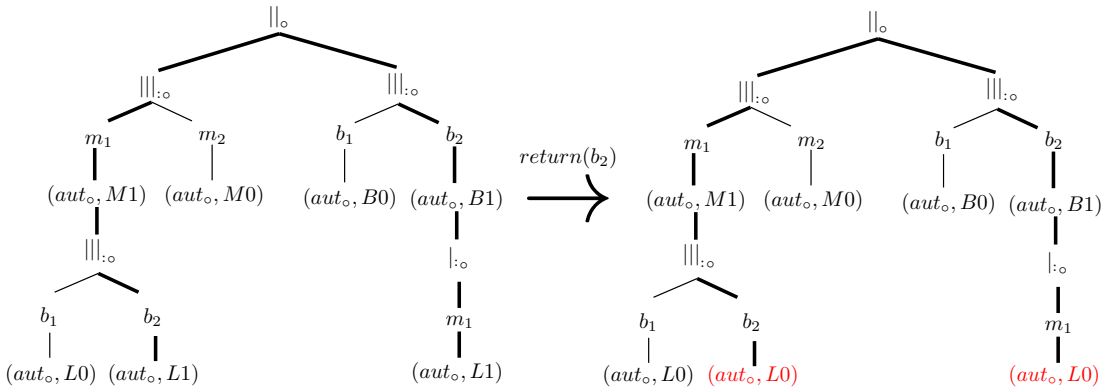

Figure 4.5 – A transition

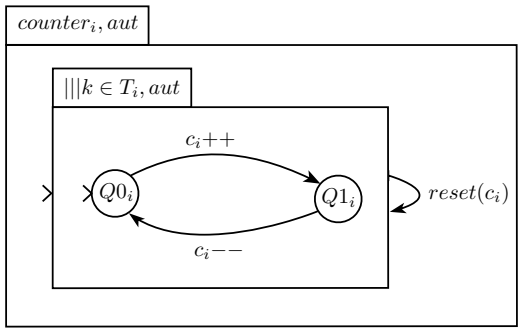

Figure 4.6 – ASTD model of a counter

configuration is a pair $(p, u) \in V \times \mathbb{N}^n$. There is a transition between two configurations $(p, (u_1, \ldots, u_n)) \to (p', (u'_1, \ldots, u'_n))$ if $(p, (a_1, \ldots, a_n), p') \in E$, where for all $i \in 1..n$, either $a_i = u'_i - u_i$ or $a_i = r \wedge u'_i = 0$.

We denote a transition system by a pair $S = (Q, \to)$, where Q is a set of states and $\to \subseteq Q \times Q$ is the set of transitions between states. We write $q \to q'$ for $(q, q') \in \to$, and $q \xrightarrow{*} q'$ if either $q = q'$ or there exists a finite sequence of transitions $q \to q_1 \to q_2 \to \cdots \to q'$, called a path. We say that a transition system $(Q, \to)$ simulates another transition system $(Q', \hookrightarrow)$ if there exists an injection $f : Q' \to Q$ such that for all $q'_1, q'_2 \in Q'$ such that $q'_1 \hookrightarrow q'_2$, we have $f(q'_1) \xrightarrow{*} f(q'_2)$.

Intuitively, an RVASS consists of a finite set of counters $\{c_1, \ldots, c_n\}$ and of a graph called control graph. Each arc of the graph is labeled by an action that can change the values of the counters.

Proposition 4.1. *PASTDs simulate VASSs with resets.*

Proof. Let (V, E) be an RVASS. Let us construct a PASTD that simulates (V, E).

- First, let us model the counters. Each counter c_i can be modeled by a two states automaton ASTD within a quantified interleaving ASTD as shown in Figure 4.6. There is a $c_i{+}{+}$ action which increments the counter i, a $c_i{-}{-}$ action which decrements it and a reset action $reset(c_i)$ which sets c_i to zero. To handle the reset action, we add a non-final loop on the interleaving operation. The value of c_i is given by the number of processes in the local state $Q1_i$. For an unbounded counter, the quantification set is simply represented by a variable T_i taking values in the subsets of $\mathbb{N}$.

- Second, the control graph can be represented by a simple automaton ASTD. According to the previous model of counters, we increase or decrease only one counter at the same time and only by one. Thus, we need to split some transitions into sequences of transitions by adding new control states.

- Finally, we put together all counters thanks to interleaving PASTDs. Then, the global PASTD modeling the RVASS is represented by a synchronization PASTD between the control graph and the counters.

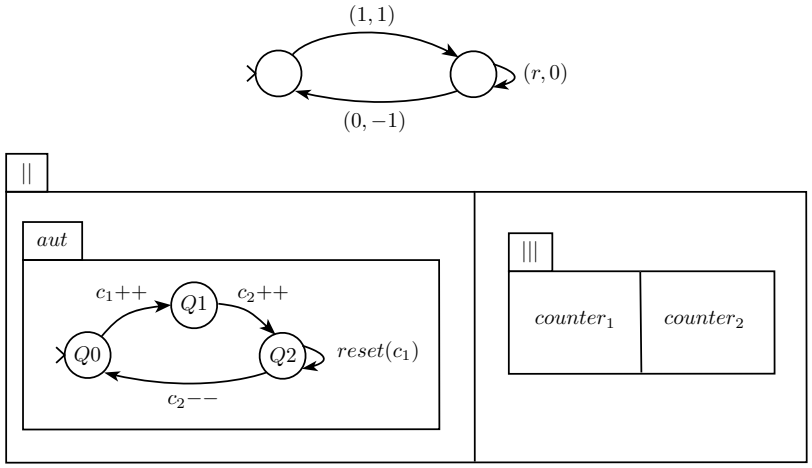

Figure 4.7 – An RVASS and its PASTD simulation

That way, each transition of configuration of the RVASS is mimicked by a transition or a sequence of transitions in the resulting PASTD. Thus, PASTDs can simulate RVASSs. See an example of simulation in Figure 4.7. □

This construction allows us to deduce some decidability results with regards to reachable configurations. For instance, the reachability problem is known to be undecidable for RVASSs [31]. Since simulation preserves reachability of states, we can conclude that the reachability problem is also undecidable for PASTDs.

Proposition 4.2. *Reachability is undecidable for PASTDs.*

Proof. PASTDs simulates RVASSs and reachability is undecidable for RVASSs. □

4.3 Well-Structured PASTD

4.3.1 Preliminaries

For a transition system $S = (Q, \to)$, we define $Pred(q) = \{q' \in Q \mid q' \to q\}$ as the set of immediate predecessors of q and $Pred^*(q) = \{q' \in Q \mid q' \xrightarrow{*} q\}$ as the set of all predecessors of q.

A *quasi-ordering (qo)* is a reflexive and transitive binary relation $\leq$ on a set X; we also say that $(X, \leq)$ is a qo. A *partial ordering (po)* is an antisymmetric qo. Note that

for natural numbers, we use the same symbol $\leq$ and denote by $(\mathbb{N}, \leq)$ the natural order and by $(\mathbb{N}^k, \leq)$ the product order.

Let $(X, \leq)$ be a po. For all $s_1, s_2 \in X$, we say that $s_3 \in X$ is the *supremum* (or sup) of s_1 and s_2 noted $sup(s_1, s_2)$ if s_3 is an upper bound of s_1 and s_2 (*i.e.* $s_1 \leq s_3$ and $s_2 \leq s_3$), and for all upper bounds $s_4 \in X$, we have $s_3 \leq s_4$.

Let $(X, \leq)$ be a qo. An *upward-closed* set is a subset $Y \subseteq X$ such that if $x \leq y$ and $x \in Y$ then $y \in Y$. For some $x \in X$, we write $\uparrow x = \{y \in X \mid x \leq y\}$ its upward-closure, and for some $Y \subseteq X$, $\uparrow Y = \bigcup_{x \in Y} \uparrow x$. A *basis* of an upward-closed subset $Y \subseteq X$ is any set B such that $Y = \uparrow B$. We say that an upward-closed set Y has a *finite basis* if there exists some finite basis B of Y.

A qo $(X, \leq)$ is *well-founded* if there is no infinite sequence $(x_n)_{n \in \mathbb{N}}$ over X such that $x_{i+1} \leq x_i$ for every $i \in \mathbb{N}$. It is a *well-quasi-ordering* (wqo) if every infinite sequence $(x_n)_{n \in \mathbb{N}}$ over X contains an increasing pair, *i.e.* $\exists i < j$ such that $x_i \leq x_j$. If the wqo is a po, then it is called a *well partial order* (wpo). An *antichain* is a set in which each pair of different elements is incomparable.

Proposition 4.3 ([50])**.** *If $\leq$ is a qo on X then the following are equivalent:*

1. *$\leq$ is a wqo;*
2. *Every infinite sequence $(x_n)_{n \in \mathbb{N}}$ in X contains an infinite nondecreasing subsequence $x_{n_0} \leq x_{n_1} \leq \cdots$ with $n_0 < n_1 < \cdots$;*
3. *$\leq$ is well-founded and has no infinite antichain;*
4. *Every upward-closed subset $U \subseteq X$ has a finite basis;*
5. *Every nondecreasing sequence $U_0 \subseteq U_1 \subseteq \cdots \subseteq U_i \subseteq \cdots$ of upward-closed subsets U_i of X eventually stabilizes.*

4.3.2 Well-Structured Transition Systems

The theory of *Well-Structured Transition Systems* [37, 1, 39] has been developed to unify and generalize some decidability results on termination, boundedness and coverability problems for infinite state systems equipped with a wqo on their states

and a monotone transition relation with regards to this wqo. However, because of the monotony condition, wqos are typically hard to find for complex systems. Sometimes, we will focus on transition systems with weaker constraints, that is, monotone transition systems.

We consider *Ordered Transition System* (OTS) $\mathcal{S} = (Q, \rightarrow, \leq)$ where $(Q, \rightarrow)$ is a transition system and Q is equipped with a quasi-ordering $\leq$. We are mainly interested here in the coverability problem for ordered transition systems, which consists, given an OTS $\mathcal{S} = (Q, \rightarrow, \leq)$, a set of initial states $I \subseteq Q$ and a state $s \in Q$, in deciding whether there exists a (possibly empty) run $i \xrightarrow{*} s'$ such that $s \leq s'$ and $i \in I$. We says that s is *coverable* if there is such a run. Let us denote the coverability problem by the predicate $cov(\mathcal{S}, I, s)$.

As shown in [1], the coverability problem can be solved by computing the set $Pred^*(\uparrow s)$ of all predecessors of $\uparrow s$ and then it suffices to check whether $I \cap Pred^*(\uparrow s)$ is empty to verify whether s is coverable. To compute $Pred^*(\uparrow s)$, we can iteratively compute the sequence of sets of predecessors $Pred(\uparrow s), Pred^2(\uparrow s) = Pred(Pred(\uparrow s))$, $\ldots, Pred^n(\uparrow s) \ldots$ and we have $Pred^*(\uparrow s) = \bigcup_{n \in \mathbb{N}} Pred^n(\uparrow s)$. But this sequence, in general, does not necessarily stabilize.

For monotone (ordered) transition systems, the set $Pred^*(\uparrow s)$ is upward-closed; and if moreover $\leq$ is a wqo, the set $Pred^*(\uparrow s)$ also have a finite basis. This opens the way to compute this finite basis which represents the set $Pred^*(\uparrow s)$.

Definition 4.5 (Monotone transition system [39])**.** *A monotone transition system (MTS) $\mathcal{S} = (Q, \rightarrow, \leq)$ is an ordered transition system such that for all $q_1 \leq q_1'$ and transition $q_1 \rightarrow q_2$, there exists a run $q_1' \xrightarrow{*} q_2'$ such that $q_2 \leq q_2'$*

In Proposition 3.1 of [39], the monotony of an ordered transition system is presented as the compatibility of $\leq$ with the transition relation $\rightarrow$. Now if a monotone transition system $\mathcal{S} = (Q, \rightarrow, \leq)$ has a wqo $\leq$, it is a WSTS.

Definition 4.6 (Well-Structured Transition System [39])**.** *A Well-Structured Transition System $\mathcal{S} = (Q, \rightarrow, \leq)$ is a monotone transition system such that $\leq$ is a wqo on Q.*

For example, Petri Nets are WSTSs if we take the inclusion of markings as the partial

order. Indeed, the multiset inclusion is wpo and the transition system is monotone. Lossy Channel Systems [38] are another example of WSTS.

Monotone transition systems and WSTSs are both supposed to be *effective*: $\leq$ is decidable (*i.e.* there exists an algorithm determining whether a pair of elements belongs to $\leq$) and $\rightarrow$ is decidable.

The following lemma relies on the monotony condition.

Lemma 4.1 ([39]). *For a monotone transition system $\mathcal{S} = (Q, \rightarrow, \leq)$ and $q \in Q$, we have $\uparrow Pred^*(\uparrow q) = Pred^*(\uparrow q)$.*

To make the iterative computation of the $Pred^n(\uparrow q)$, we need an OTS with *effective pred-basis*.

Definition 4.7 (Effective pred-basis [39]). *An OTS $\mathcal{S} = (Q, \rightarrow, \leq)$ has effective pred-basis if there exists an algorithm accepting any state $q \in Q$ and returning $pb(q)$, a finite basis of $\uparrow Pred(\uparrow q)$ (or pred-basis).*

For an OTS $\mathcal{S} = (Q, \rightarrow, \leq)$ with effective pred-basis, and a finite subset of states $F \subseteq Q$, the backward coverability analysis consists in computing a sequence of finite sets of states $K_0, K_1 \ldots$ such that $K_0 = F$ and $K_{n+1} = K_n \cup pb(K_n)$. We may verify that $Pred^*(\uparrow F) = \bigcup_{i \in \mathbb{N}} \uparrow K_i$.

Lemma 4.2 ([39]). *Let $\mathcal{S} = (Q, \rightarrow, \leq)$ be a monotone transition system and $s \in Q$. If there exists $m \in \mathbb{N}$ such that $\uparrow K_m = \uparrow K_{m+1}$, then $\uparrow K_m = Pred^*(\uparrow s)$.*

Effective pred-basis condition is sufficient to show that a finite basis for each $\uparrow K_i$ is computable according to the definition of the K_i. Since we suppose that MTSs and WSTSs are effective, the qo $\leq$ is decidable, and then inclusion $\uparrow K_n \supseteq \uparrow K_{n+1}$ and equality $\uparrow K_n = \uparrow K_{n+1}$ can be tested. Moreover, if $\leq$ is a wqo, then $(\uparrow K_i)_{i \in \mathbb{N}}$ converges. Hence, the iterative computation of K_n gives a procedure to the problem of reachability of $\uparrow s$ (*i.e.*, the coverability of s).

Theorem 4.1 ([39]). *For a WSTS $\mathcal{S} = (Q, \rightarrow, \leq)$ with effective pred-basis and $s \in Q$, a finite basis $B \subseteq Q$ of $Pred^*(\uparrow s)$ is computable. Hence, for any set of initial states $I \subseteq Q$, $cov(\mathcal{S}, I, s)$ is decidable, assuming the emptiness of $I \cap \uparrow B$ is decidable.*

Remark that if I is a finite set or an upward-closed set given by a finite basis, checking the emptiness of $I \cap \uparrow B$ is straightforward with a decidable $\leq$. But here we consider any I (because in PASTD the set of initial states is infinite and not upward-closed) and state the decidability of $I \cap \uparrow B = \emptyset$ as a condition compared to [39].

4.3.3 Monotone PASTD

In this section, we present a qo $\preceq$ based on the subgraph relation such that PASTDs are monotone.

To apply the theory of WSTSs to PASTDs, we need to define a qo on the set of states. But first, let us define an equivalence relation on states. We consider, from a reachability and coverability perspective, that the identifier of an entity instance has no importance and can always be replaced by another one; this has the advantage of reducing the computational complexity while allowing for a qo to be defined for a subset of PASTD. We consider that two states are equivalent if they are syntactically equivalent after renaming the entity identifiers using a permutation.

We call *permutation* any bijection $f : X \to X$. A *graph isomorphism* from (V, E) to (V', E') is a bijective function $f : V \to V'$ such that for all $v_1, v_2 \in V$, $\{v_1, v_2\} \in E$ iff $\{f(v_1), f(v_2)\} \in E'$.

Definition 4.8 (State equivalence)**.** *Let $A = (F, \overrightarrow{T}, \overrightarrow{P})$ be a PASTD and $s, s' \in \mathcal{T}_A$ such that $s = (V, E, \lambda)$ and $s' = (V', E', \lambda')$. We define $s \sim s'$ if there is a graph isomorphism ϕ from (V, E) to (V', E') such that for each P_i there is a permutation σ_i on P_i where for all $v \in V$, $\sigma_i(\lambda(v)) = \lambda'(\phi(v))$ if $\lambda(v) \in P_i$.*

Recall that for a PASTD $A = (F, \overrightarrow{T}, \overrightarrow{P})$, the set of labels for the nodes of the tree representation of a state is given by the set $\{(\mathsf{aut}_\circ, q_1), \ldots, (\mathsf{aut}_\circ, q_j), ||_\circ, |:_\circ, |||:_\circ\} \cup \bigcup_i P_i$. Definition 4.8 means that the entity instance identifiers are irrelevant for our purpose. Hence, we can always rename an identifier of P_i by another one of the same P_i if all occurrences are renamed similarly. For example, the state of Figure 4.3 is equivalent to the state of Figure 4.8 where the book b_1 is borrowed by the member m_2. Take the permutations σ_b and σ_m on P_b and P_m respectively such that $\sigma_b = id_{P_b} \oplus \{b_1 \mapsto b_2, b_2 \mapsto b_1\}$ and $\sigma_m = id_{P_m} \oplus \{m_1 \mapsto m_2, m_2 \mapsto m_1\}$, where id_X the identity on the

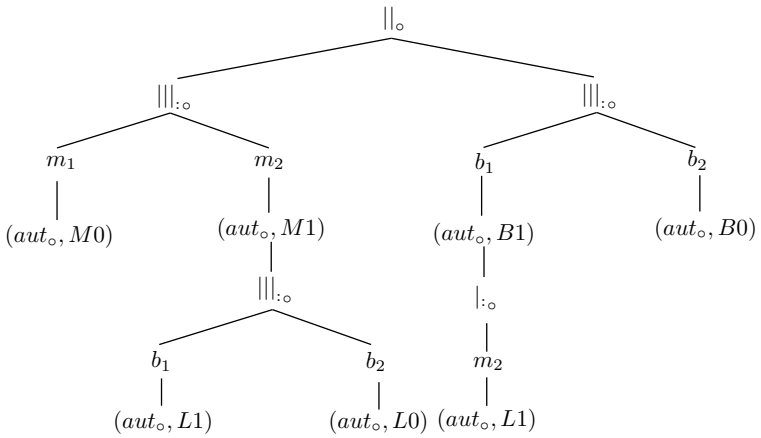

Figure 4.8 – A state of the library system

set X and $f \oplus g$ the overriding of $f : X \to Y$ by $g : W \to Y$, *i.e.* $(f \oplus g)(x) = g(x)$ if $x \in W$ and $(f \oplus g)(x) = f(x)$ otherwise.

To define a quasi-ordering on the set of states, we use the definition of induced subgraph. (V, E) is called an *induced subgraph* of (V', E') if $V \subseteq V'$ and $E = \{\{v_1, v_2\} \in E' \mid v_1, v_2 \in V\}$. Let $(\Lambda, \leq)$ be a qo set of labels. We define by $\sqsubseteq$ the extension of the induced subgraph relation (modulo isomorphism) to labeled graphs such that $(V, E, \lambda) \sqsubseteq (V', E', \lambda')$ if there exists f an isomorphism from (V, E) to an induced subgraph of (V', E') such that $\lambda(v) \leq \lambda'(f(v))$ for all $v \in V$. If no ordering on Λ is specified, we assume that $\sqsubseteq$ is defined according to the identity (Λ, id_Λ).

Definition 4.9 (State quasi-ordering)**.** *Let A be a PASTD and $s, s' \in \mathcal{T}_A$. We define $s \preceq s'$ if there exists $s'' \sim s$ such that $s'' \sqsubseteq s'$.*

Intuitively, we have $s \preceq s'$ if s is a subtree of s' modulo renaming. Furthermore, as we consider only well-formed states for a specific PASTD, subtree states can only be obtained by pruning branches from a quantified interleaving node.

Proposition 4.4. *PASTDs under $\preceq$ are monotone transition systems.*

Proof. The proof of monotony is similar to the one given in Appendix A (Theorem A.2) for a superset of PASTD. □

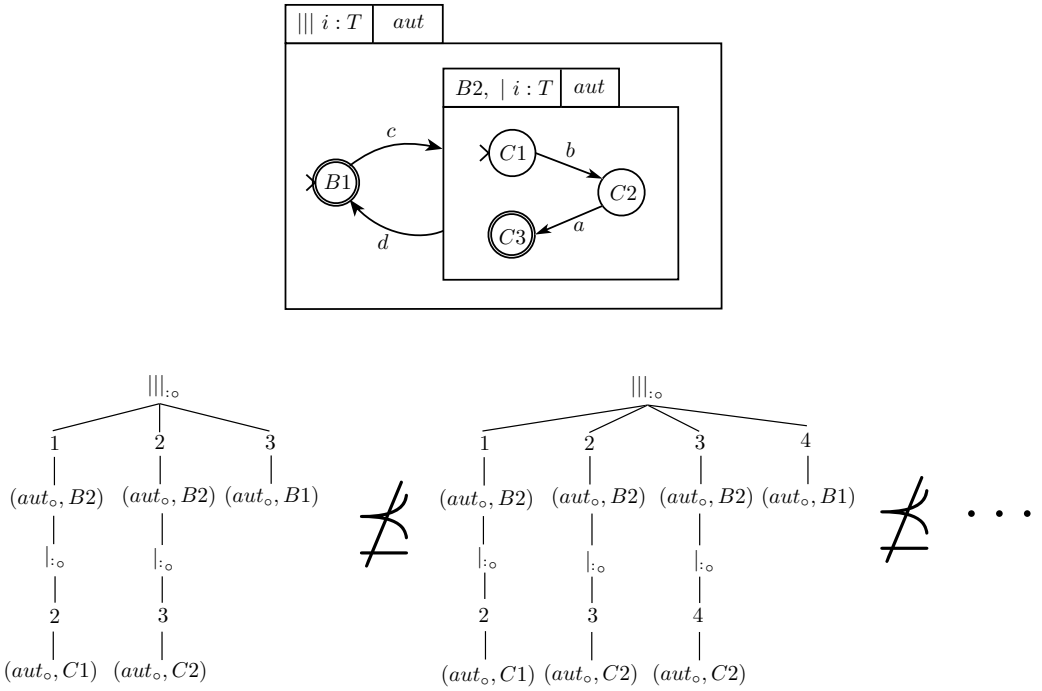

Figure 4.9 – An infinite antichain in $(\mathcal{T}_A, \preceq)$

4.3.4 Bounded-PASTD

PASTDs under $\preceq$ are not WSTSs because in general $\preceq$ is not wqo.

Proposition 4.5. *There is a PASTD such that $\preceq$ is not wqo.*

Proof. Figure 4.9 shows an example of infinite antichain for a PASTD. By Proposition 4.3, $\preceq$ is not wqo. □

We have not found yet a useful wqo for all PASTDs. Nonetheless, for some PASTDs, the set of states is wqo. For example, it is obviously the case for PASTDs without parameters, as the set of states is then finite. Some other weaker conditions can be used to get a wqo. We define in the following a class of PASTDs satisfying the wqo property. But first, we propose another representation of the state. Indeed, we aim at using a wqo theorem from [30] (see Theorem 4.2 in the sequel) and we need a graph structure with a wqo on the set of labels. However, we did not find any adequate wqo on the set of labels if we keep the same tree structure (that is also why

we cannot use Kruskal's wqo theorem on trees [60]). Hence, we use the following representation, where we replace labels denoting values e of P_i by new nodes which are labeled by i, thus abstracting from the particular value of e. This little trick removes infinite sequences of incomparable elements without losing critical information.

Definition 4.10 (Graph of s)**.** *Let $A = (F, \overrightarrow{T}, \overrightarrow{P})$ be a PASTD. For every $s = (V, E, \lambda) \in \mathcal{T}_A$ with $(R_1, \ldots, R_n) = val(s)$, let $g(s) = (V', E', \lambda')$ be defined as follows:*

- $V' = V \cup \{(i, p) \mid i \in 1..n \wedge p \in R_i\}$
- $E' = E \cup \{\{(i, p), w\} \mid i \in 1..n \wedge p \in R_i \wedge w \in V \wedge \lambda(w) = p\}$
- $\lambda' = \lambda \oplus \{(i, p) \mapsto i \mid i \in 1..n \wedge p \in R_i\} \oplus \{w \mapsto 0 \mid \exists i \in 1..n \cdot \lambda(w) \in R_i\}$

Intuitively, the function g consists in replacing labels p from parameter sets R_i by using new nodes of the form (i, p) and new edges between new nodes and nodes labeled from R_i. Each new node (i, p) is connected with each node that has the same label in P_i. Old labels p from R_i are replaced with a default value 0. New nodes (i, p) are labeled with i. In this way, we obtain another representation of the state, which is still a graph but not a tree. But the codomain of the labeling λ' for the set of all possible states $\mathcal{T}_A$ is now bounded as it does not include $\bigcup_i P_i$ anymore. Note that each equivalent state in the previous version has the same graph representation in the new version. An example is given in Figure 4.10 which shows the graph corresponding to the state of Figure 4.3. The red labels represent the new nodes and the blue ones are the old labels. The dotted lines are the new added edges.

We denote by $\mathcal{G}_A = g(\mathcal{T}_A)$ the set of new graph representations for states and by Λ_A the set of all labels in $\mathcal{G}_A$, *i.e.* the codomain of each labeling function.

$$\Lambda_A = \{(\mathsf{aut}_\circ, q_1), \ldots, (\mathsf{aut}_\circ, q_j), ||_\circ, |:_\circ, |||:_\circ\} \cup \bigcup_{i \in 1..n} \{i\} \cup \{0\}$$

We introduce a new class of PASTD called Bounded-PASTD which satisfies the wqo property. In a graph (V, E), a *path* is a sequence of vertices $v_1, v_2, \ldots, v_n \in V$ such that $\{v_i, v_{i+1}\} \in E$ for all $i \in 1..n-1$. A *simple path* is a path $v_1, v_2, \ldots, v_n \in V$ such that $v_i \neq v_j$ for all $i \neq j$. We denote by $\mathcal{P}_k(\Lambda)$ the set of graphs $\mathcal{P}_k$ whose vertices are labeled by the set Λ. Let $\mathcal{P}_k$ the class of graphs without simple paths of length k.

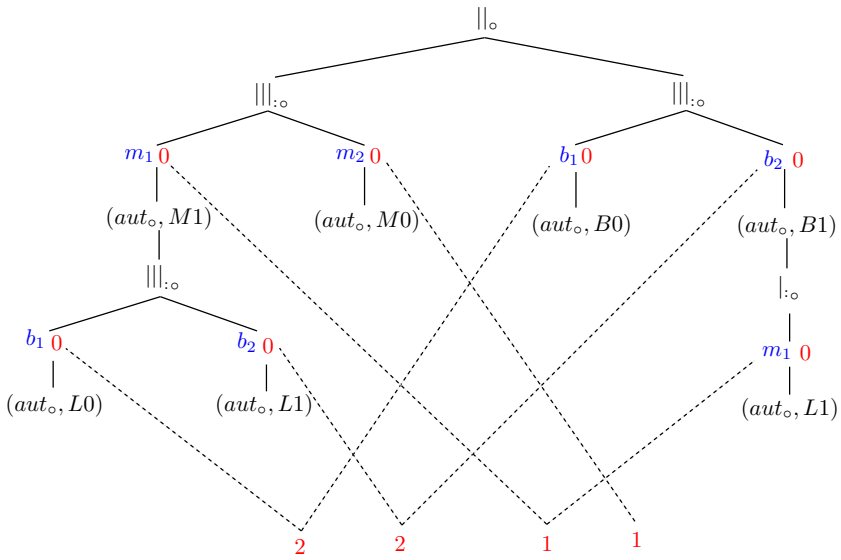

Figure 4.10 – The graph representation of the state from Figure 4.3

Definition 4.11 (Bounded-PASTD)**.** *A PASTD A is a Bounded-PASTD if there exists $k \in \mathbb{N}$ such that $\mathcal{G}_A \subseteq \mathcal{P}_k(\Lambda_A)$.*

Recall that $\sqsubseteq$ is the induced subgraph relation for labeled graphs with regards to a qo set of labels $(\Lambda, \leq)$. When $\mathcal{G}_A$ is only composed of graphs whose simple paths are bounded, we can refer to Ding's wqo theorem [30] to show that $(\mathcal{G}_A, \sqsubseteq)$ is a wqo, and thus to prove that Bounded-PASTDs are well-structured.

Theorem 4.2 (Ding's theorem [30])**.** *For $k \in \mathbb{N}$, $(\mathcal{P}_k(\Lambda), \sqsubseteq)$ is a wqo if $(\Lambda, \leq)$ is a wqo.*

We denote by $\mathcal{T}_A/\!\sim$ the quotient set of $\mathcal{T}_A$ with regards to the equivalence relation $\sim$ and we use the same symbol $\preceq$ to denote the corresponding partial order between the equivalence classes. Let $\tilde{g} : \mathcal{T}_A/\!\sim \to \mathcal{G}_A$ the function such that $\tilde{g}(\tilde{s}) = g(s)$ for all $s \in \mathcal{T}_A$, where $\tilde{s}$ denotes the equivalence class of s. Consider the qo $(\mathcal{G}_A, \sqsubseteq)$ using the identity $(\Lambda_A, id_{\Lambda_A})$ as the qo on the set of labels. Similarly to graph isomorphisms, we call isomorphism from a qo $(X, \leq_X)$ to a qo $(Y, \leq_Y)$ any bijective function $f : X \to Y$ such that for all $s_1, s_2 \in X$, $s_1 \leq_X s_2$ iff $f(s_1) \leq_Y f(s_2)$. Such isomorphisms preserve wqo.

Lemma 4.3. *For any PASTD A, $\tilde{g}$ is an isomorphism from $(\mathcal{T}_A/\!\sim, \preceq)$ to $(\mathcal{G}_A, \sqsubseteq)$.*

Proof. Let $A = (F, \{T_1, \ldots, T_n\}, \{P_1, \ldots, P_n\})$. Since $\mathcal{G}_A$ is the image set of g, *i.e.* $\mathcal{G}_A = g(\mathcal{T}_A)$, we have that g is surjective. Thus, $\tilde{g}$ is also surjective. Let us prove that there exists $f : \mathcal{G}_A \to \mathcal{T}_A/\sim$ such that $f \circ \tilde{g}$ is the identity on $\mathcal{T}_A/\sim$. Let $t = (V, E, \lambda) \in \mathcal{G}_A$. We define $f(t)$ as the equivalence class of the state $s = (V', E', \lambda')$ which is constructed as follows. For all $i \in 1..n$, let $V_i = \{v \in V \mid \lambda(v) = i\}$ and $\lambda_i : V_i \to P_i$ an arbitrary injection. Let $V' = V \setminus \bigcup_{i \in 1..n} V_i$ and $E' = E \setminus \{\{v, w\} \in E \mid \exists i \in 1..n \cdot w \in V_i\}$ and $\lambda' = (\lambda \oplus \bigcup_{i \in 1..n, w \in V_i} \{v \mapsto \lambda_i(w) \mid \{v, w\} \in E\})\restriction_{V'}$, where $f\restriction_X$ is the domain restriction of the function f to X. It is easy to see that $f(\tilde{g}(\tilde{s})) = \tilde{s}$ for $\tilde{s} \in \mathcal{T}_A/\sim$ by definition of $\tilde{g}$ and f. Hence, the function $\tilde{g}$ is also injective. Let $x, y \in \mathcal{T}_A/\sim$. If $x \preceq y$ then there are $s_1 \in x$, $s_2 \in y$ such that $s_1 \sqsubseteq s_2$. By definition of g, $g(s_1) \sqsubseteq g(s_2)$. By the definition of $\tilde{g}$, $\tilde{g}(x) = g(s_1)$ and $\tilde{g}(y) = g(s_2)$. Hence $\tilde{g}(x) \sqsubseteq \tilde{g}(y)$. Now suppose that $\tilde{g}(x) \sqsubseteq \tilde{g}(y)$. Then, for all $s_1 \in x$ and $s_2 \in y$, $g(s_1) \sqsubseteq g(s_2)$. Then, take $s_1 \in x$, $s_2 \in y$ such that $s_1 \preceq s_2$. Hence, $x \preceq y$. □

Theorem 4.3. *For any Bounded-PASTD A, $(\mathcal{T}_A, \preceq)$ is a wqo.*

Proof. Let $k \in \mathbb{N}$ satisfying Definition 4.11. By Ding's theorem $(\mathcal{P}_k(\Lambda_A), \sqsubseteq)$ is a wqo. Then, any subset $\mathcal{G}_A \subseteq \mathcal{P}_k(\Lambda_A)$ is a wqo and by Lemma 4.3 $(\mathcal{T}_A/\sim, \preceq)$ is a wqo, because isomorphisms preserve wqos. Thus, $(\mathcal{T}_A, \preceq)$ is a wqo. □

As Bounded-PASTDs are also monotone, we can conclude the following result.

Proposition 4.6. *Bounded-PASTDs are WSTSs.*

We will show in Section 4.4.2 that the coverability problem is decidable for Bounded-PASTDs.

4.3.5 Classes of Bounded-PASTD

Deciding if a PASTD is a Bounded-PASTD may not be easy, as it would require to check if all simple paths in states are bounded. Thus, in the following we propose two easily recognizable subclasses of Bounded-PASTD. For instance, a PASTD $A = (F, \overrightarrow{T}, \overrightarrow{P})$ where each parameter T_i appears only once in the expression F is called a 1-PASTD and a PASTD without nested quantifications is a Flat-PASTD. From an IS point of view, a 1-PASTD represents a system in which the entities can

only be associated by weak entity relationships and a Flat-PASTD a system without relationships. Let us define formally these subclasses of PASTD.

As illustrated in Figure 4.2, recall that for a PASTD $A = (F, \overrightarrow{T}, \overrightarrow{P})$, where F is given in its tree notation (V, E, λ), the codomain of λ is the set of labels $\Lambda = \bigcup_i(\{\mathcal{A}_i\} \cup \bigcup_j\{q_{i,j}\}) \cup \bigcup_i\{||_{\Delta_i}\} \cup \bigcup_{i\in 1..n}\{\,|_{x\in T_i}\} \cup \bigcup_{i\in 1..n}\{\,|||_{x\in T_i}\}$, where the $\mathcal{A}_i$ are the different automaton ASTDs, $q_{i,j}$ the states of $\mathcal{A}_i$ and $||_{\Delta_i}$ the synchronization ASTDs.

Definition 4.12 (1-PASTD)**.** *Let $A = (F, \{T_1, \ldots, T_n\}, \{P_1, \ldots, P_n\})$ be a PASTD with $F = (V, E, \lambda)$. We call A a 1-PASTD if for all $i \in 1..n$, the set of nodes $\{v \in V \mid \lambda(v) \in \{\,|_{x\in T_i}, |||_{x\in T_i}\}\}$ is a singleton set.*

In a (labeled acyclic connected) graph (V, E, λ) that is also a tree, we denote by $ch(v)$ the child nodes of v and by $des(v) = ch^+(v)$ its transitive closure, *i.e.* the sets of descendants of v. Similarly, we denote by $pa(v)$ and $anc(v) = pa^+(v)$ the parent and ancestors of v respectively.

Definition 4.13 (Flat-PASTD)**.** *Let $A = (F, \{T_1, \ldots, T_n\}, \{P_1, \ldots, P_n\})$ be a PASTD with $F = (V, E, \lambda)$. Let $V_q = \{v \in V \mid \lambda(v) \in \bigcup_{i\in 1..n}\{\,|_{x\in T_i}, |||_{x\in T_i}\}\}$ (the quantified operator nodes). We call A a Flat-PASTD if for all $v \in V_q$ and all $w \in des(v)$, $\lambda(w) \notin \bigcup_{i\in 1..n}\{\,|_{x\in T_i}, |||_{x\in T_i}\}$.*

For instance, the PASTD simulating a VASS in Figure 4.7 is a Flat-PASTD as there is no nested quantified interleaving operators. It is also a 1-PASTD because T_1 and T_2 appear only in one quantified interleaving operator each.

Proposition 4.7. *Any 1-PASTD is a Bounded-PASTD.*

Proof. Let $s \in \mathcal{T}_A$ such that $s = (V, E, \lambda)$ and $g(s) = (V', E', \lambda')$. It is easy to see that $g(s)$ is a tree because every new node $v \in V' \setminus V$, which represents a value of a quantified node, is connected to only one other node. Moreover, the depth of s is bounded by the depth of F. Thus, every simple path in $\mathcal{G}_A$ is bounded. □

Now, let us prove that Flat-PASTDs are Bounded-PASTDs. Let $A = (F, \overrightarrow{T}, \overrightarrow{P})$ a PASTD, $s = (V, E, \lambda) \in \mathcal{T}_A$ with $(R_1, \ldots, R_n) = val(s)$ and $g(s) = (V', E', \lambda')$. V can be decomposed into three disjoint sets $V = V_{s,1} \cup V_{s,2} \cup V_{s,3}$ as follows:

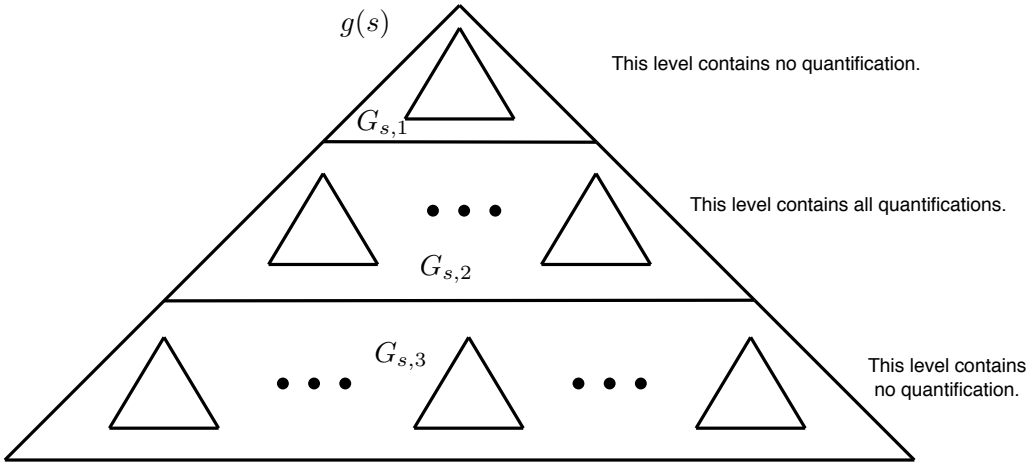

Figure 4.11 – Node partition of a state

- $V_{s,1} = \{v \in V \mid \lambda(v) \notin \{|:_\circ, |||:_\circ\} \wedge \lambda(anc(v)) \cap \{|:_\circ, |||:_\circ\} = \emptyset\}$, the nodes of the subtree containing the root of s and all descendants down to the first quantified operator;
- $V_{s,2} = \{v \in V \mid \lambda(v) \in \{|:_\circ, |||:_\circ\} \cup \bigcup_i P_i \vee$
  $(\lambda(anc(v)) \cap \{|:_\circ, |||:_\circ\} \neq \emptyset \wedge \lambda(des(v)) \cap \bigcup_i P_i \neq \emptyset)\}$,
  the nodes of subtrees whose root is a quantified operator and including all descendants down to the quantification values;
- $V_{s,3} = V \setminus (V_{s,1} \cup V_{s,2})$, the remaining nodes.

Intuitively, as $V_{s,2}$ is the only part containing values of quantification, then only the nodes of $V_{s,2}$ have a different label through g in V'. V' can be decomposed into three disjoint sets $V'_{s,1} = V_{s,1}$, $V'_{s,2} = V_{s,2} \cup \bigcup_{i \in 1..n, p \in R_i} \{(i,p)\}$ and $V'_{s,3} = V_{s,3}$. We denote by $G_{s,1}$, $G_{s,2}$ and $G_{s,3}$ the subgraphs of s induced by $V_{s,1}$, $V_{s,2}$ and $V_{s,3}$ respectively and $G'_{s,2}$ the subgraph of $g(s)$ induced by $V'_{s,2}$. See Figure 4.11.

Lemma 4.4. *Let A be a PASTD. There exists $k \in \mathbb{N}$ such that for all $s \in \mathcal{T}_A$, $G_{s,1}$ contains at most k nodes and $G_{s,3}$ is composed of trees whose depth is at most k.*

Proof. As there is no quantified operator in $V_{s,1}$, then by the definition of the state structure, the number of nodes is bounded and depends on F only. Same reasoning for each tree in $G_{s,3}$. □

Lemma 4.5. *Let A be a PASTD. A is a Bounded-PASTD iff there exists $k \in \mathbb{N}$ such that for all $s \in \mathcal{T}_A$, $G'_{s,2} \in \mathcal{P}_k(\Lambda_A)$.*

Proof. Direction $\implies$. By Definition 4.11, if A is a Bounded-PASTD, then there is $k \in \mathbb{N}$ such that for all $s \in \mathcal{T}_A$, we have $g(s) \in \mathcal{P}_k(\Lambda_A)$. Thus, $G'_{s,2} \in \mathcal{P}_k(\Lambda_A)$. Direction $\impliedby$. Let $k_1 \in \mathbb{N}$ such that for all $s \in \mathcal{T}_A$, we have $G'_{s,2} \in \mathcal{P}_{k_1}(\Lambda_A)$. Let $s = (V, E, \lambda) \in \mathcal{T}_A$. Let us prove that there is $k \in \mathbb{N}$ such that $g(s) \in \mathcal{P}_k(\Lambda_A)$. By Lemma 4.4, let $k_2 \in \mathbb{N}$ such that $G_{s,1}$ contains at most k_2 nodes and $G_{s,3}$ is composed of trees whose depth is at most k_2. Let $u = (v_1, \ldots, v_m)$ be a simple path in $g(s)$ with $v_i \in V$ for all $i \in 1..m$ and $m \in \mathbb{N}$ its size. If u crosses only $G_{s,1}$, then $m \leq k_2$. If u crosses only $G_{s,3}$, then $m \leq 2 \times k_2$, as there are only trees of depth $\leq k_2$ in $G_{s,3}$. If u crosses only $G'_{s,2}$, then $m \leq k_1$. A more general simple path u in $g(s)$ can be decomposed into several segments $u_1, \ldots, u_j$ such that each segment crosses only one of the subgraphs $G_{s,1}$, $G'_{s,2}$ or $G_{s,3}$. Typically, for a path having the most segments, remark that the first segment begins in $G_{s,3}$, then the next ones alternate between $G'_{s,2}$ and (possibly) $G_{s,1}$ and the final one ends back in $G_{s,3}$. The path can only alternate k_2 times between $G'_{s,2}$ and $G_{s,1}$ as $G_{s,1}$ contains at most k_2 nodes. Thus, $m \leq 2 \times k_2 + k_1 + k_2 \times (k_2 + k_1) + 2 \times k_2 = k_2(4 + k_1 + k_2) + k_1$. Let $k' = k_2(4 + k_1 + k_2) + k_1$. We have $g(s) \in \mathcal{P}_{k'}(\Lambda_A)$. As k' is independent from s, $g(s) \in \mathcal{P}_{k'}(\Lambda_A)$ for all $s \in \mathcal{T}_A$. □

Proposition 4.8. *Any Flat-PASTD is a Bounded-PASTD.*

Proof. Let $A = (F, \{T_1, \ldots, T_n\}, \{P_1, \ldots, P_n\})$ a PASTD with $F = (V_F, E_F, \lambda_F)$. Let us prove that there is $k \in \mathbb{N}$ such that for all $s \in \mathcal{T}_A$, we have $G'_{s,2} \in \mathcal{P}_k(\Lambda_A)$, which is equivalent to being a Bounded-PASTD thanks to Lemma 4.5. Let $s \in \mathcal{T}_A$ and $g(s) = (V, E, \lambda)$. Because A has no nested quantifications, there are only two types of nodes in $V'_{s,2} = W_1 \cup W_2$, where $W_1 = \{v \in V \mid \lambda(v) \in \{|:_\circ, |||:_\circ\}\}$ the set of "quantified operator" nodes, and $W_2 = \{v \in V \mid \lambda(v) \in \{0, 1, \ldots, n\}\}$ the set of "parameters" nodes. Clearly, the longest simple path including only nodes from W_2 is of length 3, by construction of $g(s)$. Moreover, remark that, without nested quantifications, the size of W_1 is bounded by a number k_1 that depends on the structure F only (W_1 contains at most one $v \in V$ with $\lambda(v) \in \{|:_\circ, |||:_\circ\}$ for each $v_F \in V_F$ such that $\lambda_F(v_F) \in \bigcup_{i \in 1..n} \{\,|_{x \in T_i}, |||_{x \in T_i}\}$). By a similar reasoning as in the previous lemma,

a "maximal" simple path alternates between W_1 and W_2 and its length is bounded by $k' = (1+3) \times k_1 = 4 \times k_1$. As k does not depend on s, $G'_{s,2} \in \mathcal{P}_{k'}(\Lambda_A)$ for all $s \in \mathcal{T}_A$. □

As 1-PASTDs and Flat-PASTDs are Bounded-PASTDs, we can conclude that we have two easily recognizable classes of PASTD that are WSTSs.

4.4 Computation of pred-basis for PASTDs

To apply the backward analysis algorithm presented in Section 4.3.2 to PASTDs, we need to prove a last condition that is the effective pred-basis. By definition, the existence of pred-basis condition can be related to the wqo condition, because with a wqo any upward-closed set has a finite basis. However, we think that these two conditions should be independent from each other to have a better understanding of the algorithm. Thus, we propose in Section 4.4.1 a new framework which identifies a set of conditions proving the effective pred-basis without wqo. Then, we show in Section 4.4.2 how to use the framework on PASTDs, hence to solve the coverability problem.

4.4.1 Ranked Monotone Transition Systems

In this section, we describe a general method to prove the *effective pred-basis* in some infinite systems without the wqo condition. We will show first that without wqo and under some other conditions, there exists a finite basis of $\uparrow Pred(\uparrow s)$. Then, we show that the finite pred-basis is computable with some effectivity hypothesis.

In order to compute a pred-basis for s, a naive approach would be to compute the upward-closure of s and then the set of all predecessors. However, this procedure does not terminate when $\uparrow s$ is infinite. In practice, for most interesting systems, there is no need to consider the entire set of upper states to compute a basis of predecessors. Thus, we propose a method that determines which finite subset of $\uparrow s$ is sufficient to compute a finite basis of $\uparrow Pred(\uparrow s)$. To this end, we define a new class of transition systems called *Ranked Monotone Transition Systems*. Note that, to keep

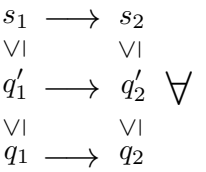

Figure 4.12 – Condition 4 of RMTS : the backward-downward monotony

the definitions simple, we consider OTSs with partial ordering in the following. We will see in the next section how to apply the method to OTSs with quasi-ordering.

Definition 4.14 (Ranked MTS)**.** *A Ranked Monotone Transition System* $\mathcal{S} = (Q, \rightarrow, \leq, \gamma, c, d)$ *is a monotone transition system* $(Q, \rightarrow, \leq)$*, where* $\leq$ *is a po, with a function* $\gamma : Q \rightarrow \mathbb{N}^n$ *and* $c, d \in \mathbb{N}^n$ *s.t. :*

1. γ *is a monotone function and* $\gamma^{-1}(r)$ *is finite for all* $r \in \mathbb{N}^n$*,*
2. *for all* $s_1, s_2, s_3 \in Q$ *such that* $s_1 \leq s_3$ *and* $s_2 \leq s_3$*, there exists* $s_4 \in Q$ *such that* $s_1 \leq s_4$*,* $s_2 \leq s_4$*,* $s_4 \leq s_3$ *and* $\gamma(s_4) \leq \gamma(s_1) + \gamma(s_2)$*,*
3. *for all* $s_1 \rightarrow s_2$*, there exists* $s'_1 \in Q$ *such that* $s'_1 \leq s_1$ *and* $s'_1 \rightarrow s_2$ *and* $\gamma(s'_1) \leq \gamma(s_2) + d$*,*
4. $\mathcal{S}$ *is backward-downward monotone (bdm): for each transition* $s_1 \rightarrow s_2$*, there is a transition* $q_1 \rightarrow q_2$ *s.t.:*
   (a) $q_2 \leq s_2$*,* $\gamma(q_2) \leq c$*, and*
   (b) $\forall q'_2 \in Q \cdot q_2 \leq q'_2 \leq s_2 \implies \exists q'_1 \in Q \cdot q'_1 \rightarrow q'_2 \wedge q_1 \leq q'_1 \leq s_1$*.*

Let us give the intuition for each condition of Definition 4.14:

1. The function γ associates a vector of natural numbers called *rank* to each state of the system. Intuitively, the rank of a state represents its size and this size can be multidimensional. We can see the rank function as a way to split the partial ordering into several finite layers of states, each layer corresponding to a rank. This principle applies well to some ordered transition systems. In particular, parameterized systems can be naturally equipped with rank functions. Indeed, a rank of a state may correspond to the number of instances of each entity in the state.

2. Condition 2 demands that if s_1 and s_2 have an upper bound s_3, then we can find a lesser one s_4 whose rank is bounded by the sum of the ranks of the two considered states.

3. To understand Condition 3, consider a parameterized system and suppose that the transition $s_1 \to s_2$ is "destructive", that is some instances of entities are lost during the transition and $\gamma(s_2) < \gamma(s_1)$. If it is a "reset transition" (*i.e.* the number of lost instances for similar transitions is unbounded), then there must exist a smaller configuration $s_1' \leq s_1$ whose rank is bounded by $\gamma(s_2) + d$ and that can fire the transition. If the transition relation never decreases the rank by more than d, then the property is always satisfied.

4. The central property is Condition 4. It states that the *kernel* of a transition can be identified and is small. This *kernel* is the set of entity instances that change state in a transition (*e.g.*, the book and member involved in a loan transition). For each transition we look for a smaller one that has the "same semantics" and such that there exist intermediary transitions. See Figure 4.12 for a graphical representation of the backward-downward monotony. The value c is an upper bound on the sizes of all minimal transitions in the system. Intuitively, for each transition we identify a smaller part that is essential to fire the transition while the rest remains stable. The definition ensures that the minimal transition is a good one by checking that every intermediary transition is also possible.

Definition 4.15 (Effective RMTS)**.** *An RMTS $\mathcal{S} = (Q, \to, \leq, \gamma, c, d)$ is effective if $\to$ and $\leq$ are decidable and $\gamma(q)$ and $\gamma^{-1}(r)$ are both computable for all $(q, r) \in Q \times \mathbb{N}^n$.*

Example 1 (Petri Nets)**.** *Let $N = (P, T, F)$ be a Petri Net where P is a set of places, T a set of transitions and $F : (P \times T \cup T \times P) \to \mathbb{N}$ a multiset of arcs. A marking is a multiset of places, i.e. a mapping $M : P \to \mathbb{N}$. We denote by $\mathcal{S}_N = (Q, \to, \subseteq)$ the corresponding transition system with Q the set of markings and $\subseteq$ the marking inclusion defined by $M_1 \subseteq M_2$ if $M_1(p) \leq M_2(p)$ for all place p. Let $\gamma : Q \to \mathbb{N}$ be a function such that $\gamma(M) = \sum_{p \in P} M(p)$.*

1. *Clearly, γ is monotone, i.e. $M_1 \subseteq M_2 \implies \gamma(M_1) \leq \gamma(M_2)$ and $\gamma^{-1}(r)$ is finite for all $r \in \mathbb{N}$ because P is finite.*

2. *For all* $M_1, M_2, M_3 \in Q$ *such that* M_3 *is an upper bound of* M_1 *and* M_2, $sup(M_1, M_2) \subseteq M_3$ *and* $\gamma(sup(M_1, M_2)) \leq \gamma(M_1)+\gamma(M_2)$, *because* $sup(M_1, M_2)$ *denotes* $max(M_1(p), M_2(p))$ *for each place* p.

3. *Let* $d \in \mathbb{N}$ *be the maximum number of tokens needed to fire a transition, then for all transitions* $M_1 \to M_2$, *we have* $\gamma(M_1) \leq \gamma(M_2) + d$.

4. *Finally, take* $c = max(F)$, *i.e. the maximum token consumed or created by a transition. Then, the bdm condition is verified. Indeed, for all transition* $M_1 \to M_2$ *fired by* $t \in T$, *we can always find a smaller one* $M_1' \to M_2'$ *where for all* $p \in P$, $M_1'(p) = F(p, t)$ *and* $M_2'(p) = F(t, p)$ *such that Condition 4 is verified.*

Thus, $(Q, \to, \subseteq, \gamma, c, d)$ *is an RMTS.*

Example 2 (Petri Net Extensions)**.** *Petri Nets with transfer arcs are RMTSs, considering the same function* γ. *Condition 3 is satisfied because the number of tokens lost during the transition is still bounded. The backward-downward monotony holds because a transfer arc is always possible for any smaller marking. The same holds for Petri Nets with reset arcs except that the number of tokens lost during a reset transition is not bounded but for any reset transition there is always a lesser transition that is bounded.*

Example 3 (Lossy Channel Systems)**.** *Consider a Lossy Channel System* $\mathcal{S}_C$ *with* Q *a set of control states,* Σ *an alphabet and* $c_1, \ldots, c_n$ *a set of FIFO channels. The system can send a message along a channel, receive one from a channel or lose one in a channel. Each message is represented by one letter of* Σ. *Hence, a transition can add or remove one letter in a channel at a time. We denote by* $\leq$ *the ordering between configurations from* $Q \times \Sigma^{*n}$ *such that* $(q, w_1, \ldots, w_n) \leq (q', w_1', \ldots, w_n')$ *if* $q = q'$ *and* $w_i \sqsubseteq w_i'$ *for all* $i \leq n$, *where* $\sqsubseteq$ *is the sub-word ordering. Take the function* $\gamma : Q \times \Sigma^{*n} \to \mathbb{N}^n$ *such that* $\gamma(q, w_1, \ldots, w_n) = (|w_1|, \ldots, |w_n|)$.

1. γ *is monotone and* $\gamma^{-1}(r)$ *is finite for all* $r \in \mathbb{N}^n$ *as* Q *and* Σ *are finite.*

2. *Since* $|sup(w, w')| \leq |w| + |w'|$ *for all* $w, w' \in \Sigma^*$, *we have* $\gamma(sup(s, s')) \leq \gamma(s) + \gamma(s')$ *for all* $s, s' \in Q \times \Sigma^{*n}$. *Thus, the supremum satisfies Condition 2.*

3. *The size of a word in a channel can only increase or decrease by one, hence for every transition* $s \to s'$ *we have* $\gamma(s) \leq \gamma(s') + (1, \ldots, 1)$.

4. *For each type of transition we have a minimal one satisfying the bdm condition. If it is a send action* $(q, w_1, \ldots, w_i, \ldots, w_n) \to (q', w_1, \ldots, w_i.a, \ldots, w_n)$, *then take the smaller transition* $(q, \epsilon, \ldots, \epsilon) \to (q', \epsilon, \ldots, a, \ldots, \epsilon)$. *If it is a receive action* $(q, w_1, \ldots, a.w_i, \ldots, w_n) \to (q', w_1, \ldots, w_i, \ldots, w_n)$, *then take the transition* $(q, \epsilon, \ldots, a, \ldots, \epsilon) \to (q', \epsilon, \ldots, \epsilon, \ldots, \epsilon)$. *Finally, if it is a loss of message* $(q, w_1, \ldots, a.w_i, \ldots, w_n) \to (q, w_1, \ldots, w_i, \ldots, w_n)$, *then take the transition* $(q, \epsilon, \ldots, a, \ldots, \epsilon) \to (q, \epsilon, \ldots, \epsilon, \ldots, \epsilon)$. *Condition 4 is then verified for* $c = (1, \ldots, 1)$.

Thus, $(Q \times \Sigma^{*n}, \to, \leq, \gamma, c, c)$ *is an RMTS.*

Nicely Sliceable WSTS (NSW)[13] share some similarities with RMTSs. They both use the same notion of partitioning of the state space and downward monotony. However, they rely on different assumptions and some of their requirements are unnecessary for our purpose. In [13], the authors propose an algorithm to compute the set of all predecessors $Pred^*(\uparrow s)$ and use the NSW framework in order to prove its correctness. That differs from our goal which is to give a complete method to prove the existence of pred-basis, that is not guaranteed without the wqo condition, and to compute it. Petri Nets, Broadcast Protocols, Lossy Channel Systems and Context-free grammars are examples of NSWs [13].

Definition 4.16 ([13])**.** *A δ-NSW (Nicely Sliceable WSTS) is a WSTS* $\mathcal{S} = (Q, \to, \leq)$ *such that*

1. $\leq$ *is discrete over* Q, *i.e.*

   (a) $\forall q \in Q, \exists k \in \mathbb{N}$ *s.t. for any sequence* $q_0 \leq \cdots \leq q_l = q$ *we have* $l \leq k$ *and there is a weight function* $w : Q \to \mathbb{N}$ *that maps each* $q \in Q$ *to the minimum such* k.

   (b) $\{q \in Q \mid w(q) = i\}$ *is finite for each* $i \in \mathbb{N}$

2. $\mathcal{S}$ *is weight respecting, i.e.*

(a) *for all* $x, x', y \in Q$ *such that* $x \to x'$ *and* $x \leq y$, *there exists* $y' \in Q$ *such that* $y \to y'$ *and* $w(x') - w(x) = w(y') - w(y)$.

3. $\mathcal{S}$ *is* δ*-deflatable, i.e.*

(a) *for* $\delta \in \mathbb{N}$, *if* $x \to x'$ *and* $z \leq x'$, *then there exist* y, y' *such that*

- $y \leq x$ *and* $y \to y'$ *and* $z \leq y'$,
- $w(y) \leq w(z) + \delta$ *and* $w(y') \leq w(z) + \delta$.

More precisely, the main differences between NSWs and RMTSs are that NSWs are based on a wqo and require the "weight respecting" condition whereas RMTSs need a po. "Weight respecting" is necessary in [13] to prove their convergence theorem (Theorem 1). Nonetheless, the RMTS condition 1 is similar to the NSW definition of discrete system (without wqo) and the RMTS conditions 2, 3 and 4 to the NSW definition of deflatability (where $\delta = max(c, d)$). Even if the RMTS conditions may look more complex, we think that stating these conditions instead of the deflatability one is more useful for the following reasons.

- The backward-downward monotony gives a better intuition, because the search for the right c is guided by the notion of the "least transition" whose states have their rank bounded by c. Thus, the RMTS definition targets the biggest "least transition" of the system.

- Proving the existential condition of the deflatability, may require an explicit construction of the states y and y', which is not trivial. We will see in the proof of Lemma 4.6 that the conditions 2 and 4 give a direct construction of the state y' (the state y is then chosen among the predecessors of y').

- RMTSs distinguish between the two values c and d (instead of δ). That allows for a better precision at the computation of the pred-basis, as shown in the following.

The objective is now to show that RMTSs allow for the determination of effective pred-basis easily without wqo. In order to find a finite basis of $\uparrow Pred(\uparrow s)$, the idea is to compute a finite subset $S \subset \uparrow s$, then a finite subset $P \subseteq Pred(S)$.

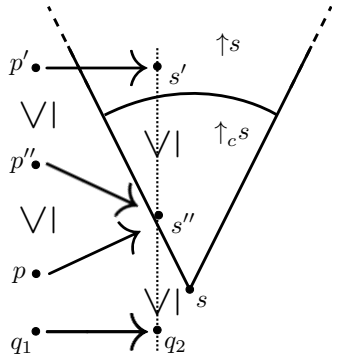

Figure 4.13 – Illustration of Lemma 4.6

Definition 4.17. *Let $\mathcal{S} = (Q, \to, \leq, \gamma, c, d)$ be an RMTS. For all $e \in \mathbb{N}^n$, let us denote $Pred_e(s) = \{q \in Pred(s) \mid \gamma(q) \leq \gamma(s) + e\}$ and $\uparrow_e s = \{q \in \uparrow s \mid \gamma(q) \leq \gamma(s) + e\}$.*

$Pred_c(s)$ and $\uparrow_c s$ restrict the set of predecessors and the upward-closure of s to the states that are interesting to compute the pred-basis.

Lemma 4.6. *Let $\mathcal{S} = (Q, \to, \leq, \gamma, c, d)$ be an RMTS and $s \in Q$. We have $\uparrow Pred(\uparrow s) = \uparrow Pred_d(\uparrow_c s)$.*

Proof. $\supseteq$ is obvious. To prove $\subseteq$, let us show that $Pred(\uparrow s) \subseteq \uparrow Pred_d(\uparrow_c s)$ (then $\uparrow Pred(\uparrow s) \subseteq \uparrow Pred_d(\uparrow_c s)$ by upward-closure). Let $p' \in Pred(\uparrow s)$ and $s' \in Q$ such that $p \to s'$ and $s \leq s'$. We want to show that there exists $p \in Q$ such that $p \leq p' \wedge \exists s'' \in Q \cdot s \leq s'' \wedge p \to s'' \wedge \gamma(s'') \leq \gamma(s) + c \wedge \gamma(p) \leq \gamma(s'') + d$. By the bdm, take $q_1 \to q_2$ such that $q_2 \leq s'$, $\gamma(q_2) \leq c$, and $\forall q_2' \in Q \cdot q_2 \leq q_2' \leq s' \implies \exists q_1' \in Q \cdot q_1' \to q_2' \wedge q_1 \leq q_1' \leq p'$. Take s'' an upper bound of s and q_2 such that $s'' \leq s'$, satisfying Condition 2. We have $q_2 \leq s'' \leq s'$. Thus, by the bdm, there is $p'' \in Q$ such that $q_1 \leq p'' \leq p'$ and $p'' \to s''$. By Condition 3, take $p \in Q$ such that $p \leq p''$, $p \to s''$ and $\gamma(p) \leq \gamma(s'') + d$. Finally, by Condition 2, $\gamma(s'') \leq \gamma(s) + \gamma(q_2) \leq \gamma(s) + c$. A graphical representation of the proof is given in Figure 4.13. □

We construct a basis of $\uparrow Pred(\uparrow s)$ by computing the set $Pred_d(\uparrow_c s)$. Let us show that this set is finite.

Lemma 4.7. *Let $\mathcal{S} = (Q, \to, \leq, \gamma, c, d)$ an RMTS. For all $s \in Q$, $\uparrow Pred(\uparrow s)$ has a finite basis.*

Proof. $Pred_d(\uparrow_c s)$ is a basis of $\uparrow Pred(\uparrow s)$ by Lemma 4.6. The set $\{r \in \mathbb{N}^n \mid r \leq c\}$ is finite, thus $\uparrow_c s$ is finite by definition of a rank function. Similarly, for all $q \in Q$, $Pred_d(q)$ is finite. Consequently, $Pred_d(\uparrow_c s)$ is finite. □

Proposition 4.9. *Effective RMTSs have effective pred-basis.*

Proof. Let $\mathcal{S} = (Q, \to, \leq, \gamma, c, d)$ be an effective RMTS. Let us show that for all $s \in Q$, $Pred_d(\uparrow_c s)$ is computable. As γ is effective, then for all $r \in \mathbb{N}^n$, $\gamma^{-1}(r)$ is computable. To compute $\uparrow_c s = \{q \in \uparrow s \mid \gamma(q) \leq \gamma(s) + c\}$, compute the finite set $X = \bigcup_{r \leq \gamma(s)+c} \gamma^{-1}(r)$ and select the elements $q \in X$ such that $s \leq q$. Then, for all $q \in \uparrow_c s$, compute $Pred_d(q) = \{q' \in Pred(q) \mid \gamma(q') \leq \gamma(q) + d\} = \{q' \in \bigcup_{r \leq \gamma(q)+d} \gamma^{-1}(r) \mid q' \to q\}$. □

This proof gives a naive algorithm to compute $Pred_d(\uparrow_c s)$, but in practice enumerating an entire set $\gamma^{-1}(r)$ is not always necessary. Especially, if $Pred(q)$ is computable, then computing $Pred_d(q)$ is more straightforward.

Example 4 (Petri Nets)**.** *Let $N = (P, T, F)$ be a Petri Net. We denote by $\mathcal{S}_N = (Q, \to, \subseteq, \gamma, c, d)$ the corresponding RMTS defined in Example 1 with $c, d \in \mathbb{N}$. Let M be a marking and let us compute a pred-basis of M. The set of markings $\uparrow_c M$ is clearly finite and can be enumerated. For each element M' of $\uparrow_c M$, $Pred(M')$ is also finite and computable, hence so is $Pred_d(M')$.*

Even if computing a pred-basis is easy for systems like Petri Nets or Lossy Channel Systems, finding one for some more complex systems [58, 69] is more difficult. The RMTS framework gives a systematic way to prove the existence of pred-basis without the wqo hypothesis and gives a generic algorithm.

Now, if $\uparrow Pred(\uparrow s)$ is computable, then as shown in Section 4.3.2, we can compute $Pred^*(\uparrow s)$ by iteration assuming the number of iterations is finite. This is guaranteed by the fact that the quasi-ordering is a wqo.

Proposition 4.10. *For an effective RMTS $\mathcal{S} = (Q, \to, \leq, \gamma, c, d)$ with $I \subseteq Q$ and $s \in Q$, if $\leq$ is a wqo, then a finite basis $B \subseteq Q$ of $Pred^*(\uparrow s)$ is computable. In addition, if $I \cap \uparrow B = \emptyset$ is decidable, then $cov(\mathcal{S}, I, s)$ is decidable.*

Proof. $\mathcal{S}$ has effective pred-basis thanks to Proposition 4.9. Since $\leq$ is a wqo, then $\mathcal{S}$ is a WSTS. Thus, by Theorem 4.1, the coverability problem is decidable. □

Remark that if we do not have a wqo, we can still use the procedure computing $Pred^*(\uparrow s)$ of Section 4.3.2 as a semi-algorithm. Furthermore, like in [13], we have intermediate results by restricting ourselves to the verification for some specific ranks. Let $\mathcal{S} = (Q, \rightarrow, \leq, \gamma, c, d)$ be an effective RMTS. Then, we can always consider a cut-off value $r \in \mathbb{N}^n$ and a subsystem $\mathcal{S}_r = (Q_r, \rightarrow, \leq)$ whose set of states is reduced to $Q_r = \{q \in Q \mid \gamma(q) \leq r\}$. During the computing of a basis of $Pred^*(\uparrow s)$, we can conclude on the coverability of s in $\mathcal{S}_r$ as we go. Indeed, if all backward paths contain a state of rank r and no initial state is currently reached, then s is not coverable in $\mathcal{S}_r$. That is more efficient than choosing a value r and checking each $\mathcal{S}_r$ one by one. We just need to adapt the backward reachability procedure of Section 4.3.2 either by prioritizing lower ranks or by keeping track of computed paths.

4.4.2 Ranked PASTD

In this section we use RMTSs to prove that PASTDs have effective pred-basis. Because we use a quasi-ordering on the state space of PASTDs, we cannot say that PASTDs are RMTSs as they require a partial ordering. However, it is natural to consider the quotient set of the state space instead of the entire space when studying systems like PASTDs. Exploiting the symmetries in systems in order to simplify a verification process is usual in the context of formal verification [36]. In our case the symmetries entailed by the equivalence relation allow us to consider the coverability problem on the quotient space.

For a PASTD A, $\mathcal{T}_A/\sim$ is the quotient set of $\mathcal{T}_A$ and for any relation (transition relation or ordering in our case) on $\mathcal{T}_A$, we use the same notation for the corresponding one on $\mathcal{T}_A/\sim$. More formally, for all $x_1, x_2 \in \mathcal{T}_A/\sim$, $x_1 \preceq x_2$ iff $\exists s_1 \in x_1, \exists s_2 \in x_2 \cdot s_1 \preceq s_2$, and $x_1 \rightarrow x_2$ iff $\exists s_1 \in x_1, \exists s_2 \in x_2 \cdot s_1 \rightarrow s_2$. We denote by $\tilde{\mathcal{S}}_A = (\mathcal{T}_A/\sim, \rightarrow, \preceq)$ the quotient system.

Proposition 4.11. *Let A be a PASTD. Then, $(\mathcal{T}_A, \rightarrow, \sim)$ and $\tilde{\mathcal{S}}_A = (\mathcal{T}_A/\sim, \rightarrow, \preceq)$ are both monotone.*

Proof. The proof of monotony of $(\mathcal{T}_A, \rightarrow, \sim)$ is similar to the one for $(\mathcal{T}_A, \rightarrow, \preceq)$ (Proposition 4.4). Let us prove that $(\mathcal{T}_A/\sim, \rightarrow, \preceq)$ is monotone. Let $x_1, y_1, x_2 \in \mathcal{T}_A/\sim$ such that $x_1 \preceq y_1$ and $x_1 \rightarrow x_2$. Then, there exist $s_1, q_1 \in x_1$, $s_2 \in x_2$ and $s_1' \in y_1$ such that $q_1 \preceq s_1'$ and $s_1 \rightarrow s_2$. By monotony of $(\mathcal{T}_A, \rightarrow, \sim)$, there exists $q_2 \in x_2$ such that $q_1 \rightarrow q_2$. Thus, by monotony of $(\mathcal{T}_A, \rightarrow, \preceq)$, there exists $s_2' \in \mathcal{T}_A$ such that $q_2 \preceq s_2'$ and $s_1' \rightarrow s_2'$. Let y_2 be the equivalence class of s_2'. Then, we have $x_2 \preceq y_2$ and $y_1 \rightarrow y_2$. □

The monotony of $(\mathcal{T}_A, \rightarrow, \sim)$ can be used to prove that the coverability problem in $\mathcal{S}_A$ can be reduced to the coverability problem in $\tilde{\mathcal{S}}_A$.

Theorem 4.4. *Let A be a PASTD, $\mathcal{S}_A = (\mathcal{T}_A, \rightarrow, \preceq)$ its corresponding OTS and $\tilde{\mathcal{S}}_A = (\mathcal{T}_A/\sim, \rightarrow, \preceq)$ its quotient OTS. Let $I \subseteq \mathcal{T}_A$, $s \in \mathcal{T}_A$, $\tilde{I} = \{\tilde{i} \mid i \in I\}$ and $\tilde{s}$ the equivalence class of s. Then, $cov(\mathcal{S}_A, I, s) \iff cov(\tilde{\mathcal{S}}_A, \tilde{I}, \tilde{s})$.*

Proof. Let us prove that $i \xrightarrow{*} s' \cdot s \preceq s' \wedge i \in I \iff \tilde{i} \xrightarrow{*} \tilde{s'} \cdot \tilde{s} \preceq \tilde{s'} \wedge \tilde{i} \in \tilde{I}$. $\implies$ is trivial. $\impliedby$ is true because $(\mathcal{T}_A, \rightarrow, \sim)$ is monotone (see Proposition 4.11). □

From now on, for a PASTD A we study the coverability problem on the quotient MTS $\tilde{\mathcal{S}}_A = (\mathcal{T}_A/\sim, \rightarrow, \preceq)$. The objective is to find an RMTS for A. Let us define an adequate rank function.

Definition 4.18. *For a PASTD $A = (F, (T_1, \ldots, T_n), (P_1, \ldots, P_n))$, let $\gamma : \mathcal{T}_A \rightarrow \mathbb{N}^n$ be the function defined by $\gamma(s) = (|R_1|, \ldots, |R_n|)$ where $\overrightarrow{R} = val(s)$ for all $s \in \mathcal{T}_A$.*

The function γ gives for each state s the number of instances of each type that appear within the description of s. For instance, if s is the state of Figure 4.3 from the library example, then $\gamma(s) = (2, 2)$ as $val(s) = (\{m_1, m_2\}, \{b_1, b_2\})$. The state describes a configuration with two members and two books.

Clearly, for all $s_1, s_2 \in x \in \mathcal{T}_A/\sim$, $\gamma(s_1) = \gamma(s_2)$. Thus, we can extend the rank function to the equivalence classes $\gamma : \mathcal{T}_A/\sim \rightarrow \mathbb{N}^n$ as $\gamma(\tilde{s}) = \gamma(s)$ for any $s \in \mathcal{T}_A$. The function has the following property with regards to transitions.

Proposition 4.12. *Let A be a PASTD. For all $x, y \in \mathcal{T}_A/\sim$, such that $x \rightarrow y$, there exists $z \preceq x$ such that $z \rightarrow y$ and $\gamma(z) = \gamma(y)$.*

Proof. By definition of the transition relation. See Appendix A (Proposition A.5). □

Now, let us prove that Condition 2 of Definition 4.14 is satisfied, *i.e.* that if two PASTD states have an upper bound, they have a "bounded" one according to the rank function γ.

Lemma 4.8. *Let A be a PASTD. For all $x_1, x_2, x_3 \in \mathcal{T}_A/\sim$ such that $x_1 \preceq x_3$ and $x_2 \preceq x_3$, there exists $x_4 \in \mathcal{T}_A/\sim$ such that $x_1 \preceq x_4$, $x_2 \preceq x_4$, $x_4 \preceq x_3$ and $\gamma(x_4) \leq \gamma(x_1) + \gamma(x_2)$.*

Proof. Let $A = (F, (T_1, \ldots, T_n), (P_1, \ldots, P_n))$. Let $x_1, x_2, x_3 \in \mathcal{T}_A/\sim$ such that $x_1 \preceq x_3$ and $x_2 \preceq x_3$. Then, there are $s_1 \in x_1, s_2 \in x_2, s_3 \in x_3$ such that $s_1 \preceq s_3$ and $s_2 \preceq s_3$. Let $s_1 = (V_1, E_1, \lambda_1)$, $s_2 = (V_2, E_2, \lambda_2)$ and $s_3 = (V_3, E_3, \lambda_3)$. By Definitions 4.8 and 4.9, as $s_1 \preceq s_3$ there exist an isomorphism f_{13} from (V_1, E_1) to an induced subgraph of (V_3, E_3) and permutations $\sigma_{13,i}$ on P_i for each $i \in 1..n$. Consider the same notations f_{23} and $\sigma_{23,i}$ for $s_2 \preceq s_3$. We construct $s_4 = (V_4, E_4, \lambda_4)$ as follows: $V_4 = \{v \in V_3 \mid v \in Im(f_{13}) \cup Im(f_{23})\}$, $E_4 = \{\{v, w\} \in E_3 \mid v, w \in V_4\}$ and $\lambda_4 = \lambda_3{\restriction_{V_4}}$. s_4 is a well-formed state because s_1, s_2 and s_3 are from the same PASTD and only branches from quantified interleaving nodes are pruned from s_3 to obtain s_4. Hence, $s_4 \in \mathcal{T}_A$. Clearly, $s_4 \preceq s_3$ by construction. Using the same injection f_{13} and permutations $\sigma_{13,i}$, we have that $s_1 \preceq s_4$. Same for $s_2 \preceq s_4$. By definition of γ, we have that $\gamma(s_4) \leq \gamma(s_1) + \gamma(s_2)$ because for each $v \in V_4$ with $\lambda_4(v) \in P_i$ either there is $w \in V_1$ such that $\sigma_{13,i}(\lambda_1(w)) = \lambda_4(v)$ or there is $w \in V_2$ such that $\sigma_{23,i}(\lambda_2(w)) = \lambda_4(v)$. Take $x_4 \in \mathcal{T}_A/\sim$ the equivalence class of s_4. We have $x_1 \preceq x_4$, $x_2 \preceq x_4$, $x_4 \preceq x_3$ and $\gamma(x_4) \leq \gamma(x_1) + \gamma(x_2)$. □

Intuitively, take the example of the library and consider a state s_1 such that $\gamma(s_1) = (1, 1)$ and where the member m_1 borrowed the book b_1 and a state s_2 such that $\gamma(s_2) = (1, 1)$ and where the member m_2 has returned the book b_2, then clearly s_1 and s_2 have upper bounds. We can construct a "little" state s_3 such that $\gamma(s_3) = (2, 2)$ including the two different cases.

Now, for each PASTD, we can determine a $c \in \mathbb{N}^n$ which is a bound satisfying Condition 4 of Definition 4.14, using the function μ defined as follows.

Definition 4.19. *Let $A = (F, \overrightarrow{T}, \overrightarrow{P})$ be a PASTD. The function μ, which gives for each PASTD expression F a vector of natural $c \in \mathbb{N}^n$, is defined recursively as follows.*

1. $\mu(\epsilon) = (0, \ldots, 0)$
2. $\mu(\mathcal{A}[q_1[F_1], \ldots, q_j[F_j]]) = sup(\{\mu(F_i) \mid i \in 1..j\})$
3. $\mu(||_\Delta[F_1, F_2]) = \mu(F_1) + \mu(F_2)$
4. $$\mu(\,|_{x \in X}[F]) = \begin{cases} \mu(F) & \text{if } X \notin \{T_1, \ldots, T_n\} \\ \mu(F) + (u_1, \ldots, u_n) & \text{if } X = T_i \\ & \quad \text{where } u_i = 1 \text{ and} \\ & \quad u_j = 0 \text{ for all } i \neq j \end{cases}$$
5. $$\mu(\,|||_{x \in X}[F]) = \begin{cases} |X| \cdot \mu(F) & \text{if } X \notin \{T_1, \ldots, T_n\} \\ \mu(F) + (u_1, \ldots, u_n) & \text{if } X = T_i \\ & \quad \text{where } u_i = 1 \text{ and} \\ & \quad u_j = 0 \text{ for all } i \neq j \end{cases}$$

Note that for all $C \subset \mathbb{N}^n$, we denote by $sup(C)$ the supremum of the set C in $\mathbb{N}^n$.

The function μ determines the maximum number of instances of each type that are involved in a transition. Intuitively, we count one instance for each quantified operator associated to the corresponding parameter. Remark that there is at least one instance for each parameter that is used in a quantified operator. For a PASTD $A = (F, \overrightarrow{T}, \overrightarrow{P})$, $\mu(F)$ allows us to prove the backward-downward monotony.

Lemma 4.9. *Let $A = (F, \overrightarrow{T}, \overrightarrow{P})$ be a PASTD and $c = \mu(F)$. For each transition $x_1 \to x_2$, there is $y_1 \to y_2$ such that $y_2 \preceq x_2$, $\gamma(y_2) \leq c$, and $\forall z_2 \in \mathcal{T}_A/\!\sim \cdot\, y_2 \preceq z_2 \preceq x_2 \implies \exists z_1 \in \mathcal{T}_A/\!\sim \cdot\, z_1 \to z_2 \wedge y_1 \preceq z_1 \preceq x_1$.*

Proof. By induction on the structure F of the PASTD. See the detailed proof in Appendix A (Lemma A.14). □

For instance, consider the PASTD of the library $(F, (T_m, T_b), (P_m, P_b))$ whose expression F is illustrated in Figure 4.2. By Definition 4.19, we have $\mu(F) = (2, 2)$ (remark that each parameter appears twice in the tree). It means that for each transition in

the system, a maximum of 2 members and 2 books will be actually involved. The property is verified in the transition depicted in Figure 4.5: the lesser transition is represented by bold strokes and includes only one member and one book.

We can now state that PASTDs are effective RMTSs.

Theorem 4.5. *Let $A = (F, \overrightarrow{T}, \overrightarrow{P})$ be a PASTD and $\tilde{\mathcal{S}}_A = (\mathcal{T}_A/\sim, \rightarrow, \preceq)$ the quotient MTS, then $(\mathcal{T}_A/\sim, \rightarrow, \preceq, \gamma, \mu(F), \overrightarrow{0})$ is an effective RMTS.*

Proof. Let $\mathcal{S} = (\mathcal{T}_A/\sim, \rightarrow, \preceq, \gamma, \mu(F), \overrightarrow{0})$. $(\mathcal{T}_A/\sim, \rightarrow, \preceq)$ is a MTS by Proposition 4.11 and $\preceq$ a po on $\mathcal{T}_A/\sim$.

1. Let $x, y \in \mathcal{T}_A/\sim$ such that $x \preceq y$. We clearly have $\gamma(x) \leq \gamma(y)$ by definition of γ. Thus, γ is monotone. Let $r \in \mathbb{N}^n$ be a specific rank. Remark that there exists a $k \in \mathbb{N}$ such that for all $x \in \gamma^{-1}(r)$ with $\tilde{g}(x) = (V, E, \lambda)$, $|V| \leq k$. Then, for all $x \in \gamma^{-1}(r)$ with $\tilde{g}(x) = (V, E, \lambda)$, $|V|$ and $|E|$ are bounded and $Im(\lambda)$ is always finite. By a combinatorial argument, there is a finite number of possible graphs whose rank is r. And because $\tilde{g}$ is a bijection, $\gamma^{-1}(r)$ is then finite.

2. Condition 2 is proved by Lemma 4.8.

3. For all $x \rightarrow y$, there exists $z \preceq x$ such that $z \rightarrow y$ and $\gamma(z) \leq \gamma(y) + \overrightarrow{0}$ because $\gamma(z) = \gamma(y)$ by Proposition 4.12.

4. Condition 4 is proved by Lemma 4.9.

As a consequence, $\mathcal{S}$ is an RMTS. The transition relation is decidable as it is defined by a set of rules in [41]. Also by definition, $\preceq$ is decidable and $\gamma(\tilde{s})$ computable for $s \in \mathcal{T}_A$. For $r \in \mathbb{N}^n$, $\gamma^{-1}(r)$ is computable by enumerating all well-formed states $\tilde{s}$ such that $\gamma(\tilde{s}) = r$. Thus, $\mathcal{S}$ is an effective RMTS. □

For any PASTD, we conclude that we can compute a finite pred-basis of any state s. However, if we want to compute a finite basis of $Pred^*(\uparrow s)$ and decide coverability, we need the wqo condition.

Theorem 4.6. *Bounded-PASTDs are WSTSs with effective pred-basis. Hence, the coverability problem is decidable.*

Proof. Let $A = (F, \overrightarrow{T}, \overrightarrow{P})$ be a Bounded-PASTD, $(\mathcal{T}_A/\sim, \rightarrow, \preceq, \mu(F), \overrightarrow{0})$ the corresponding RMTS, $\tilde{I} \subseteq \mathcal{T}_A/\sim$ and $\tilde{s} \in \mathcal{T}_A/\sim$. We have that $\preceq$ is a wpo as A is a Bounded-PASTD, thus we can compute a basis of $Pred^*(\uparrow\tilde{s})$. We can check whether $Pred^*(\uparrow\tilde{s}) \cap \tilde{I}$ is empty or not because $\tilde{I}$ is easily recognizable (initial states are explicited in [41]). Hence, by Proposition 4.10, $cov(\tilde{\mathcal{S}_A}, \tilde{I}, \tilde{s})$ is decidable. □

4.5 Related Work

Several models in the literature allow for the specification of parameterized systems [76, 48, 87, 28, 69, 58], but most of them are too restrictive for some complex systems like information systems because they do not allow us to model relationships between entities or parameters. In general, process algebras [9, 15, 43] are suitable to specify information systems but it is not always clear how to handle parameterized systems. In particular, [40] shows that specifying and verifying information systems can be done for finite state systems. However, most of the existing models are not adapted to parameterized verification. The verification of EB^3 specifications, from which the ASTD notation derives, has been studied in [94] but the paper does not consider parameters. In [87], the authors present a model close to ours called *LTS schema* that improves labeled transition systems with process algebra operators and allows parameterization.

Along with abstraction [68, 20, 2] and cut-off techniques [34, 48, 87, 86, 55, 2], well-structured transition systems were widely used to verify parameterized systems [69, 58, 28, 2] or more generally infinite systems. Finding the right wqo on the set of states is a central issue. In [69], [28] and [58], the authors used a graph representation of their states and the adequate subgraph wqo that is similar to our case: the set of states is restricted to those of bounded simple paths. Alternate wqos on graphs, like the *graph minor* relation [82], can also be used [58]. But in our case it does not satisfies the monotony condition.

In [58], the authors present a framework for viewing *Graph Transformation Systems* as WSTSs under certain conditions. They give as well an algorithm computing pred-basis in this particular context. The framework of *Nicely Sliceable WSTS* [13], which is closer to ours in term of requirements, gives also a subclass of WSTS. How-

ever, they are both tied to a wqo condition (by definition of WSTS), unlike the RMTS framework.

4.6 Conclusion

Results and discussion. In this article, we presented a model called PASTD, a variant of ASTD, which allows for the specification of complex parameterized systems like information systems. A PASTD specification is a formal model with a graphical notation which is easy to read. PASTDs are more expressive than VASSs with resets, hence the reachability problem is undecidable for PASTDs. Furthermore, we pointed out some subclasses of PASTD (Bounded-PASTD, 1-PASTD and Flat-PASTD) which are WSTSs. Then, we introduced a new general framework called RMTS to prove the existence and computability of pred-basis in any monotone transition system. RMTSs give a set of properties that should be satisfied by a system to compute a pred-basis but do not require a wqo as in WSTSs. Finally, we showed that PASTDs are RMTSs and thus that the coverability problem is decidable for Bounded-PASTDs. The relation between the different classes of model presented in this paper is summarized in Figure 4.14. Remark that to match with the definition of RMTS, we assume that all quasi-orderings are partial orderings (by considering quotient space for instance).

For Bounded-PASTDs, we have a complete procedure to decide coverability and thus to verify safety properties. We think that Bounded-PASTDs can model a larger class of parameterized systems than classical models like RVASSs. Unfortunately, many parameterized systems are not Bounded-PASTDs. Indeed, the library example presented in this paper is not well-structured as we can find an example of antichain similar to the one of Figure 4.9. However, we were able to prove that PASTDs are effective RMTSs.

In fact, the RMTS framework allowed us to find the adequate conditions to construct the PASTD class of systems that is RMTS. For instance, considering abstract states in PASTDs was necessary to prove the backward-downward monotony, thus to prove the effective pred-basis without wqo and can make the computation easier in the same time. Intuitively, an abstract state is a state where the local configurations of some instances are unknown. It is useful to model a quantified interleaving

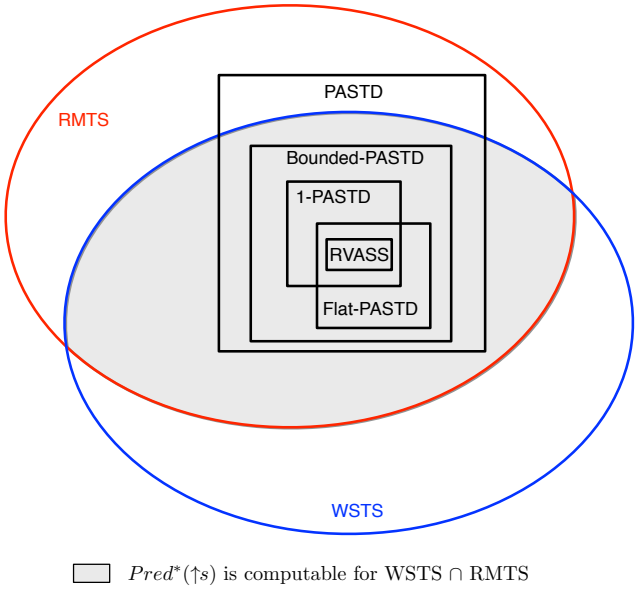

Figure 4.14 – Comparison of the models

state with partial information. Hence, an abstract state represents a set of concrete states whose quantified interleaving are completely instantiated. This allows for the factorization of the pred-basis computation (see Appendix A, Section A.5 for more details).

Considering an effective RMTS $\mathcal{S} = (Q, \rightarrow, \leq, \gamma, c, d)$ and a state $s \in Q$, a semi-decision procedure to determine if an initial state belongs to $Pred^*(\uparrow s)$ is possible. More than a semi-algorithm, sometimes the procedure may terminate without wqo even if s is not coverable. This is the case when a basis of $Pred^*(\uparrow s)$ is computed. Indeed, even if $(Q, \leq)$ is not a wqo, $(Pred^*(\uparrow s), \leq)$, which depends on the state s, may be wqo. That justifies the practical importance of this semi-decidability result. Moreover, determining the complexity of a wqo-based algorithm is not easy [85]. In practice, if we are not able to get a reasonable complexity, the wqo argument may not be sufficient to guarantee that the implemented verification procedure will end.

Future work. Experimenting the backward algorithm on different RMTSs like PASTDs would be the next step in order to determine for which kind of systems and coverability properties the procedure is likely to stop within a reasonable amount of time. Besides,

concerning PASTDs, we have given two easily recognizable subclasses of Bounded-PASTDs but we should be able to determine a more general algorithm deciding if a PASTD is well-structured, which is hinted by Lemma 4.5. Finally, as computing $Pred^*(\uparrow s)$ should not be necessary to solve the coverability problem, it would be interesting to find other termination conditions than wqo for a backward algorithm that checks emptiness of the intersection of $Pred^*(\uparrow s)$ and the set of initial states; some conditions that would be more dependent on the transition system.

Chapitre 5

Appréciation des choix et limites de l'approche

Dans le chapitre 4, nous avons proposé une solution pour le problème de la vérification d'ASTD paramétrés en nous appuyant sur les travaux existants sur les WSTS. Cependant, la vérification de systèmes paramétrés étant limitée, il nous a fallu imposer plusieurs contraintes aux systèmes étudiés afin d'appliquer notre méthode. En effet, pour considérer nos systèmes comme des WSTS, par exemple, il a été nécessaire d'identifier une sous-classe de PASTD vérifiant la propriété de beau préodre (WQO) comme présenté dans la section 4.3.4. Ainsi, plusieurs hypothèses importantes ont été faites vis-à-vis des ASTD originaux et de leur sémantique.

Dans ce chapitre, nous revenons sur certains des choix les plus importants qui ont été effectués dans le chapitre précédent, aussi bien ceux des contraintes imposées sur les ASTD que ceux de modélisation de systèmes monotones, et analysons leurs portées. La section 5.1 justifie les choix de sémantique faits pour les PASTD. On revient notamment sur l'importance des symétries, sur le problème des synchronisations quantifiées et sur l'ajout des états abstraits à l'espace d'états. La section 5.2 présente différents préordres alternatifs qui ont été étudiés pour cette thèse et compare leurs difficultés.

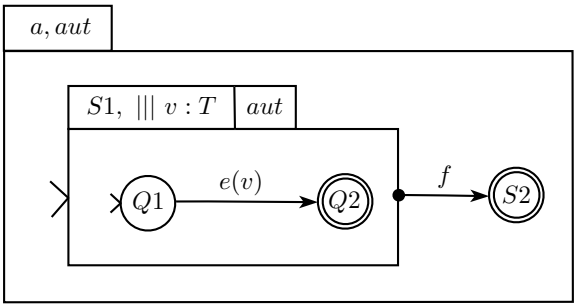

figure 5.1 – Exemple de PASTD avec entrelacement quantifié

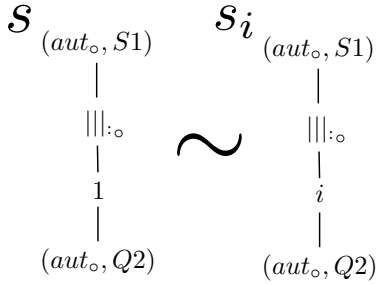

figure 5.2 – Exemple d'états équivalents

5.1 Particularités de la sémantique des PASTD

5.1.1 Symétries

Dans la section 4.3.3 du chapitre précédent, on définit un préordre sur l'espace d'états en introduisant tout d'abord une relation d'équivalence. Cette équivalence est basée sur l'idée d'une symétrie, entre les états du système, utile à la factorisation de l'espace d'états, mais aussi nécessaire pour l'obtention d'un WQO. Considérons l'exemple de PASTD de la figure 5.1, où T est un paramètre de domaine $\mathbb{N}$. Un prédécesseur de l'état $s = (\mathsf{aut}_\circ, S2)$ est l'état $s_1 = (\mathsf{aut}_\circ, S1)[|||:_\circ[1[(\mathsf{aut}_\circ, Q2)]]]$. De même pour tout $i \in \mathbb{N}$, $s_i = (\mathsf{aut}_\circ, S1)[|||:_\circ[i[(\mathsf{aut}_\circ, Q2)]]]$ est un prédécesseur de s (cf. figure 5.2). Si l'on ne considère pas la relation d'équivalence 4.8, on obtient alors une infinité d'états incomparables et sans borne inférieure. Il est alors impossible d'avoir une base finie de prédécesseurs. Ce qui n'est pas le cas si l'on considère que les états sont équivalents en renommant les identifiants.

La relation d'équivalence est pertinente, car elle préserve la monotonie. En effet,

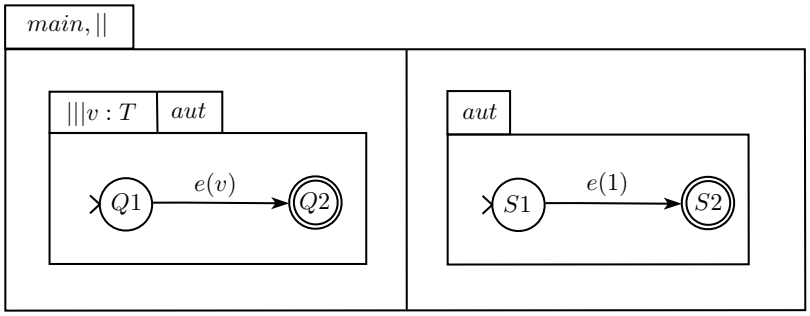

figure 5.3 – Exemple de spécification invalide

intuitivement, les identifiants d'un même domaine ont des rôles symétriques dans le système comme prouvé en annexe A par le lemme A.8. Cependant, afin de conserver cette propriété, il est important de restreindre les spécifications PASTD à celles où un élément d'un domaine de paramètre ne peut pas être directement utilisé dans le modèle. Par exemple, la spécification de la figure 5.3 est invalide, car elle utilise une action $e(1)$ paramétrée par une valeur du domaine directement sur une transition. Ainsi, les systèmes où T est instancié par $\{1\}$ et $\{2\}$, respectivement, possèdent des comportements différents qui ne respectent pas la monotonie. Dans le premier cas, la transition $e(1)$ est possible tandis qu'aucune n'est possible dans le second. Notons que l'action $e(1)$ de la partie droite de l'ASTD n'est pas affectée par le renommage d'identifiants lors de la recherche d'une permutation pour satisfaire la définition 4.8.

Par ailleurs, notons que cette contrainte de symétrie affecte aussi le type de propriété d'accessibilité que l'on peut vérifier sur le système. En effet, une propriété d'accessibilité ne peut être représentée que par un ensemble d'états clos par le haut et donc incluant toutes les permutations possibles sur les identifiants. Cependant, cela ne pose pas de problème en pratique, car les propriétés à vérifier possèdent souvent cette symétrie intrinsèquement (exemple : un livre, quel qu'il soit, ne peut être emprunté par plusieurs membres à la fois). Ainsi, pour l'exemple de la figure 5.1, déterminer si l'état s_1 est accessible est équivalent à vérifier si s_i est accessible pour tout $i \in \mathbb{N}$.

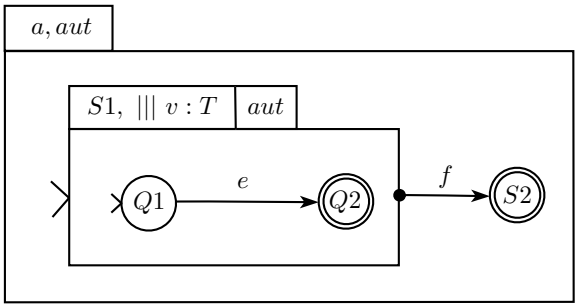

figure 5.4 – Exemple de PASTD non monotone

5.1.2 Problème des synchronisations quantifiées

L'une des contraintes les plus fortes imposées aux PASTD dans le chapitre précédent est l'absence de synchronisation quantifiée. Nous montrons dans cette section que cet opérateur ne permet pas d'obtenir la propriété de monotonie tout en conservant l'ordre partiel de la définition 4.9. De plus, l'opérateur d'entrelacement quantifié a été affaibli pour une raison similaire. En effet, la définition originale de l'entrelacement quantifié [41] impose une synchronisation de tous les sous-processus sur l'état final afin de pouvoir atteindre l'état final global.

Supposons que l'on prenne en compte la synchronisation finale de l'opérateur d'entrelacement quantifié. Prenons alors l'exemple de PASTD a paramétré par T dont le domaine est $\mathbb{N}$ représenté sur la figure 5.4. La condition finale de l'entrelacement quantifié est représentée par une flèche commençant par un point noir sur la figure. On peut alors donner une paire d'états qui brise la propriété de monotonie. En effet, prenons s l'état du PASTD a où T est instancié par $\{1, 2\}$ et chacune des deux instances de l'automate dans $S1$ se trouve dans l'état local $Q2$ (voir figure 5.5). Ainsi, comme chaque instance de l'entrelacement quantifié est dans un état final, la transition f peut se déclencher. Considérons maintenant un état s' où T est instancié par $\{1, 2, 3\}$, les instances 1 et 2 sont dans l'état $Q2$ et l'instance 3 est dans l'état $Q1$. Clairement, d'après la définition 4.9, on a $s \preceq s'$ car on ajoute simplement une instance 3 dans un état arbitraire. Cependant, la transition f n'est pas possible à partir de l'état s', car l'instance 3 n'est pas dans un état final. Ce qui falsifie la propriété de forte monotonie du système. Le même raisonnement peut être effectué dans le cas plus général des opérateurs de synchronisation quantifiée. Ce qui invalide

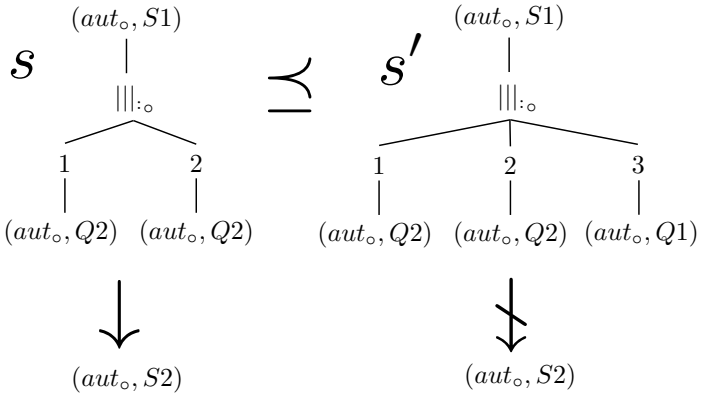

figure 5.5 – Exemple de non monotonie

l'utilisation de tels opérateurs dans le cadre de systèmes monotones.

Notons que le contre-exemple précédent ne permet en réalité que d'invalider la monotonie forte (voir la définition de *compatibilité forte* dans [39]), où l'état plus grand ne peut qu'effectuer qu'une unique transition avant d'atteindre une destination plus grande. Il se trouve que dans l'exemple donné, s' peut effectuer la transition f juste après une transition e et la paire s, s' ne falsifie pas la monotonie faible. Cependant, il est toujours possible de trouver un exemple falsifiant la monotonie faible avec le même raisonnement en limitant par exemple le nombre de transitions e possibles. Ainsi, l'exemple de la figure 5.6 contredit la monotonie faible aussi. En effet, à cause de l'automate b qui est en synchronisation avec l'automate a, seules deux instances de $S1$ peuvent effectuer la transition e, une troisième instance ajoutée dans l'état $Q1$ ne peut atteindre l'état $Q2$ et cela bloque la transition finale f.

Par ailleurs, on remarque que ce problème est spécifique aux opérateurs de quantification (synchronisation quantifiée et entrelacement quantifié), car ceux-ci ne sont pas bornés en nombre de processus qu'ils peuvent gérer. Dans le cadre d'une synchronisation entre un nombre fixe de processus, on utilisera l'opérateur de synchronisation binaire autant de fois que nécessaire.

En conséquence, on a fait le choix de restreindre les PASTD aux spécifications ne comportant pas de synchronisation quantifiée et on a modifié la sémantique de l'état final d'un entrelacement quantifié. En effet, afin de forcer la monotonie, on souhaite que pour un état s de type $|||:_\circ$, si s est final alors pour tout état s' tel que $s \preceq s'$,

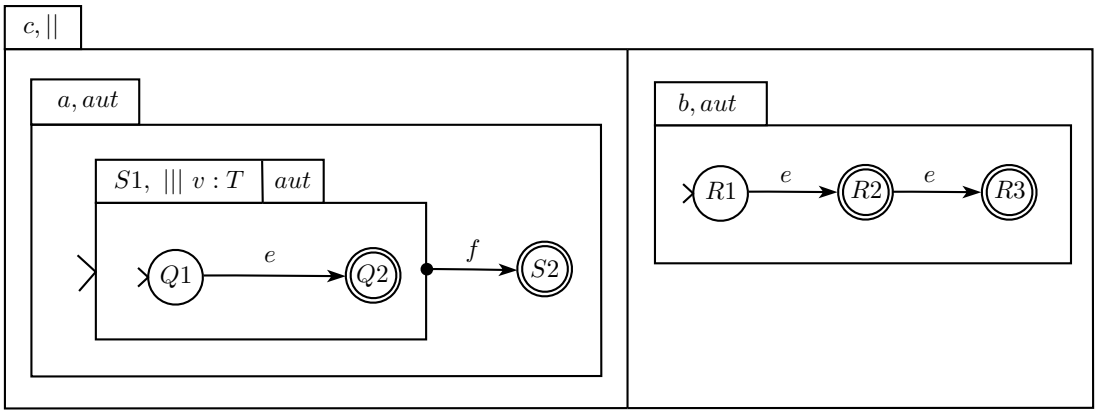

figure 5.6 – Exemple de PASTD ne vérifiant pas la compatibilité faible

on a s' qui est final aussi. La sémantique originale ([41]) impose que chacun des états composites soit final. Le choix qui a été retenu, pour la rédaction du chapitre 4 et de l'annexe A (voir définition A.15), consiste à considérer que l'état d'entrelacement quantifié est final si un des ses états composites l'est aussi. Par exemple, dans le PASTD de la figure 5.4, on autorise la transition f à être possible à partir du moment où l'une des instances se trouve dans l'état $Q2$ sans se soucier de savoir si les états des autres instances sont finaux ou pas. On notera que d'autres solutions sont possibles comme rendre tous les états d'entrelacement quantifié finaux ou n'en permettre aucun de l'être.

Changer la sémantique de l'opérateur d'entrelacement quantifié comme préconisé par la définition A.15 est l'une des modifications les plus simples à réaliser afin d'obtenir la monotonie des PASTD. Cependant, cette définition peut se révéler problématique dans certains cas de modélisation. Prenons par exemple le système de gestion de la bibliothèque de la figure 3.3 présenté dans le chapitre 3. On souhaite spécifier le processus membre afin que chaque membre retourne tous ses emprunts avant de pourvoir se désinscrire du système. Si l'on considère la modification précédente dans la sémantique des PASTD, il suffirait à un membre de retourner un seul de ses livres empruntés afin de pouvoir se désinscrire. Ce scénario pose clairement un problème pour cette sémantique de l'opérateur d'entrelacement quantifié.

Nous proposons une solution permettant de résoudre le problème précédent en changeant la façon de modéliser le système. Supposons que le nombre d'emprunts

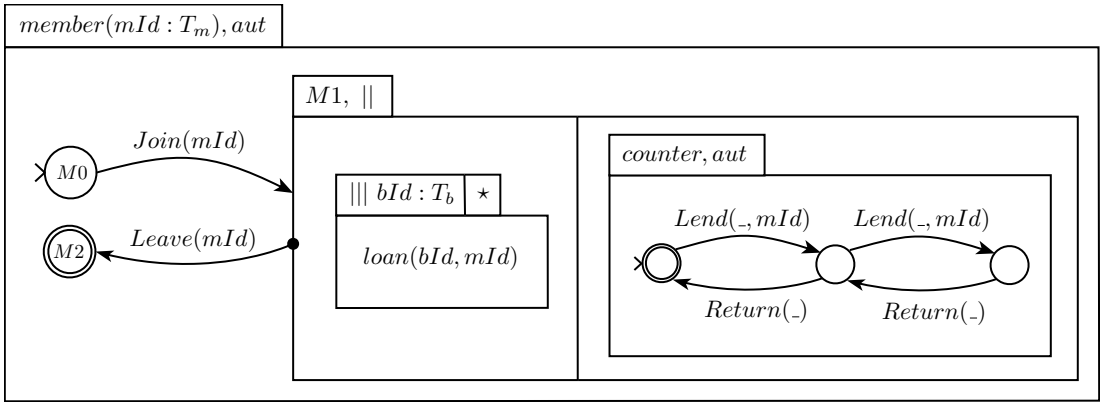

figure 5.7 – Modification du processus “member”

est limité à deux livres au total par membre. On peut alors modifier la spécification PASTD du processus membre pour obtenir celle de la figure 5.7. Notons que le symbole “_” est un métacaractère similaire à celui présenté dans la section 3.2.1 du chapitre 3. Il peut remplacer n'importe quelle valeur du type adéquat. On ajoute à la spécification du processus membre un nouvel ASTD automate *counter* en synchronisation avec l'entrelacement quantifié. Ainsi, on peut forcer l'état final à être un état où aucun emprunt n'est en cours, ce qui sera un prérequis à la transition *Leave*. Intuitivement, l'automate compte le nombre d'emprunts et permet de terminer le processus $M1$ seulement si le nombre d'emprunts devient nul. Dans ce cas, un membre ne sera pas autorisé à se désinscrire de la bibliothèque s'il n'a pas rendu tous ses livres. Cependant, ce principe de spécification ne peut être généralisé qu'aux PASTD dont le nombre de processus réellement actifs dans un entrelacement quantifié peut être limité. En effet, l'automate compteur ne peut contenir qu'un nombre fini d'états.

5.1.3 Extension de l'espace d'états

Afin de satisfaire la propriété de *backward-downward monotony* (cf. définition 4.14) et de pred-base finie du chapitre 4, une modification de l'espace d'états des ASTD a été nécessaire. En effet, la définition originale des ASTD impose que l'état d'un entrelacement quantifié comprenne un sous-état pour chaque valeur existante de l'ensemble de quantification. La définition d'état abstrait présentée dans le chapitre précédent permet, quant à elle, de former un état sans spécifier systématiquement tous ses sous-

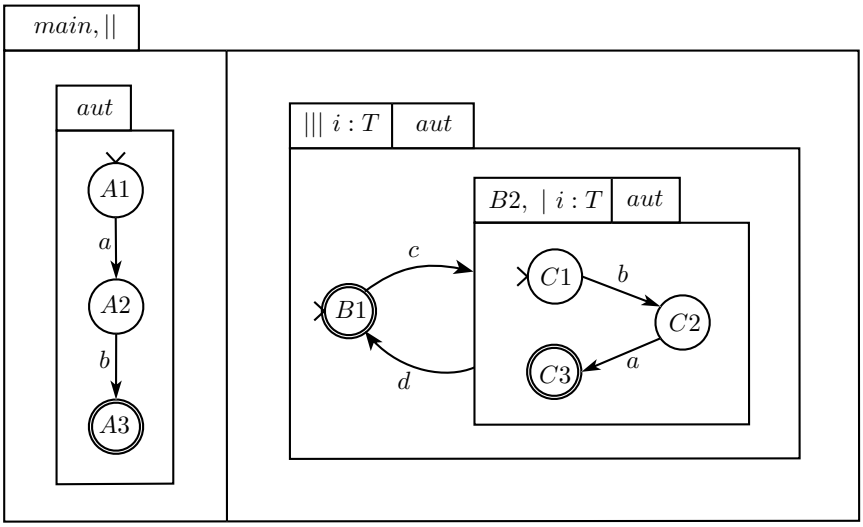

figure 5.8 – Exemple de PASTD

états. Nous montrons dans cette section l'importance de la prise en compte de tels états dans la procédure d'exploration de l'espace d'états.

Si l'on considère le cadre de la définition originale des états donnée dans [41], construire une base finie de prédécesseurs pour un état donné n'est pas toujours possible, notamment si l'espace d'états n'est pas doté d'un beau préordre. Considérons, par exemple, le PASTD *main* de la figure 5.8, où $\mathbb{N}$ est le domaine du paramètre T. Prenons l'état $s = ||_\circ[(\mathsf{aut}_\circ, A3), |||:_\circ[1[(\mathsf{aut}_\circ, B2)[|:_\circ[1[(\mathsf{aut}_\circ, C3)]]]]]$. On peut alors construire une antichaine d'états de $\uparrow s$ de taille infinie comme montré dans la figure 5.9. De même, pour chaque élément de l'antichaine, on détermine un prédécesseur à partir de la transition sur l'action b. Cet ensemble de prédécesseurs constitue lui aussi une antichaine infinie. On constate donc qu'il n'existe pas de base finie pour $\uparrow Pred(\uparrow s)$.

Pourtant, on remarque que les états de l'antichaine de $\uparrow s$ possèdent un sous-arbre en commun. De la même façon, l'antichaine des prédécesseurs a aussi une partie commune. Ceux deux parties ne forment pas des états complets, mais elles correspondent à la définition des états abstraits donnée dans le chapitre précédent. La figure 5.10 montre cette paire d'états abstraits que l'on note s_1 et s_2 représentant l'antichaine de $\uparrow s$ et celle des prédécesseurs respectivement. Intuitivement, un état abstrait est un état

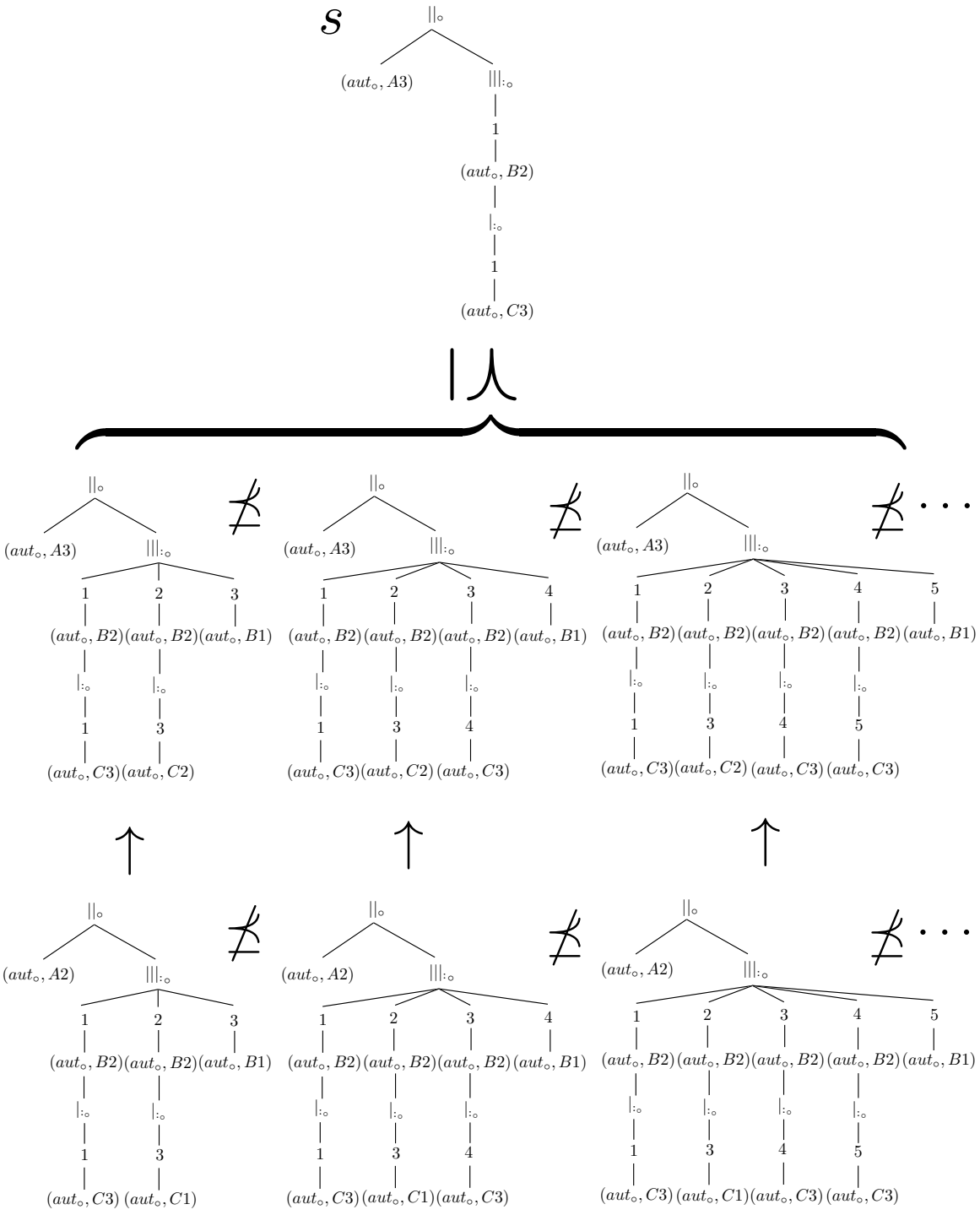

figure 5.9 – Deux antichaines infinies de $\uparrow s$ et de $Pred(\uparrow s)$

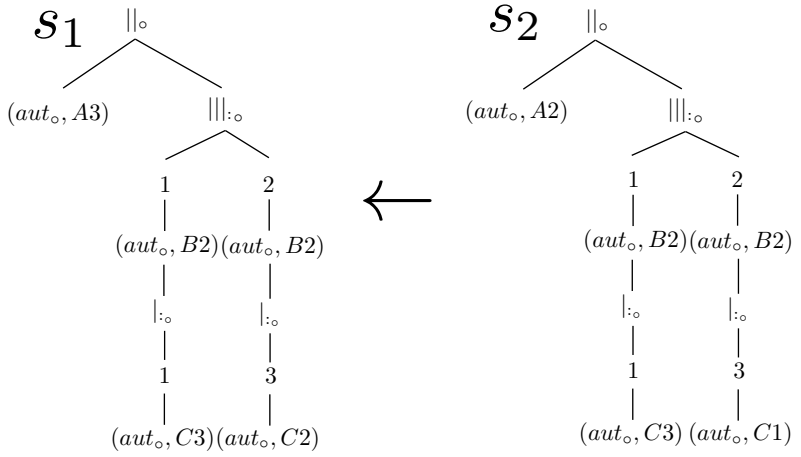

figure 5.10 – Une transition d'états abstraits

dont la configuration locale de certaines instances de sous-processus est inconnue. Un tel état représente l'ensemble des états concrets (bien formés selon la définition originale) plus grands et dont les entrelacements quantifiés sont complètement instanciés. Ainsi, pour déterminer une base de prédécesseurs pour s, on factorise les transitions à parcourir en passant par les états abstrait s_1 et s_2. Ces états contiennent en réalité l'information nécessaire à l'ensemble des transitions entre les antichaines respectives. Les états abstraits permettent donc de représenter les prédécesseurs de $\uparrow s$ par une base finie.

L'autre avantage d'avoir étendu l'espace d'états est la facilité de spécifier un problème de couverture. En effet, il est plus simple et concis de spécifier un ensemble d'états à atteindre en modélisant les configurations locales d'un sous-ensemble d'instances importantes et en omettant de préciser les configurations des autres. En particulier, il est parfois impossible de spécifier une propriété d'accessibilité par une propriété de couverture d'un unique état. Par exemple, reprenons le PASTD de la figure 5.8. Si l'on souhaite déterminer si l'ensemble R, des états plus grand que $||_\circ[(\mathsf{aut}_\circ, A3), |||_{:\circ}[i[(\mathsf{aut}_\circ, B2)[|_{:\circ}[j[(\mathsf{aut}_\circ, C2)]]]]$ pour tout $i \neq j$, est accessible, alors on ne peut représenter R par une base finie dans l'espace des états concrets. Mais cela devient possible dans l'espace d'états étendu.

L'espace d'états ayant été modifié par l'ajout des états abstraits, il reste à s'assurer que le système de transitions associé conserve toutes ses propriétés essentielles

à la vérification. La définition formelle d'état abstrait est donnée en annexe A (définition A.11). On y adapte aussi la sémantique opérationnelle aux nouveaux états (définition A.20). Notons $\mathcal{S}_A$ le système de transition d'un PASTD A selon la sémantique sans états abstraits et $\mathcal{S}_A^\sharp$ selon celle avec les états abstraits. On constate alors que $\mathcal{S}_A^\sharp$ a un espace d'états plus grand et plus de transitions. Tout d'abord, la monotonie est préservée comme montré par le théorème A.3. Ensuite, il est important de s'assurer que la vérification est équivalente dans $\mathcal{S}_A$ et dans $\mathcal{S}_A^\sharp$. Soit $\mathcal{S}_A = (Q, \rightarrow)$ et $\mathcal{S}_a^\sharp = (Q', \rightarrow')$ avec des ensembles d'états initiaux $I \subseteq Q$ et $I' \subseteq Q'$ respectivement. Pour le problème de couverture d'un état $s \in Q$, on peut effectuer la vérification dans $\mathcal{S}_A^\sharp$ si $cov(\mathcal{S}_A, I, s) \iff cov(\mathcal{S}_A^\sharp, I', s)$. Le théorème A.4 prouve qu'il est, en effet, équivalent d'utiliser l'un ou l'autre des systèmes de transitions pour le problème de couverture.

5.2 Étude du choix du préordre

Dans la section 4.3.3 du chapitre précédent, on a proposé un préordre préservant la monotonie du système de transitions et pouvant être un WQO sous certaines restrictions. Cependant, comme présenté dans la section 5.1.2, la propriété de monotonie repose sur l'absence de synchronisations quantifiées. Déterminer un préordre sur les PASTD qui soit à la fois un WQO et préserve la monotonie est l'une des questions centrales de cette thèse. Plusieurs préordres sur l'espace d'états des PASTD ont été envisagés durant cette étude. Dans cette section, nous passons en revue différents préordres et analysons leurs principales difficultés.

Rappelons tout d'abord que la théorie des WSTS autorisant des classes d'équivalences, on parle dans cette section de préordre plutôt que d'ordre. La condition d'antisymétrie de la relation est donc facultative. La difficulté du choix d'un préordre réside dans le fait que satisfaire à la fois les conditions de monotonie et de WQO est souvent impossible sans modifier la sémantique du système considéré. La monotonie donne l'idée intuitive que si s est un état, alors un état s' plus grand peut simuler tout scénario possible à partir de s. La condition de WQO, quant à elle, a pour fonction de rendre finis les calculs itératifs d'ensembles croissants et clos par le haut. Un exemple simple de préordre rendant un système monotone est la relation où tous les états

sont deux à deux incomparables. On respecte ainsi la monotonie de façon triviale car pour un état s donné, on ne peut trouver d'état plus grand. Cependant, lorsque l'espace d'états est infini, la condition de WQO n'est pas respectée, car tout ensemble infini d'états forme une antichaine. À l'inverse, en prenant un WQO simple comme un ordre total bien fondé, par exemple $(\mathbb{N}, \leq)$, la monotonie est difficile à satisfaire dans la plupart des systèmes complexes. Si l'on considère les PASTD, par exemple, il n'existe pas d'ordre ou de préordre total avec une sémantique pertinente sur l'espace d'états. En effet, bien souvent deux états pris au hasard ont une forte probabilité d'être incomparables. Ainsi, construire un préordre intéressant pour les PASTD peut demander de prendre des hypothèses supplémentaires sur les systèmes. Pour trouver le préordre adéquat, il y a donc deux approches possibles : partir d'une idée de préordre satisfaisant clairement la monotonie et trouver les conditions qui permettent de le rendre WQO, ou considérer un WQO connu et l'appliquer à la structure des données du système en essayant de satisfaire la monotonie.

5.2.1 Approche par la monotonie

Le choix du préordre pour les PASTD proposé dans cette thèse a été guidé par l'idée intuitive suivante. Pour un état s, un état plus grand s' est un état qui contient les mêmes instances dans les mêmes configurations ainsi que des instances supplémentaires dans des configurations arbitraires. Par exemple, si s est un état de la bibliothèque contenant un membre inscrit et un livre disponible, alors un état plus grand s' contiendrait par exemple deux membres inscrits, un livre disponible et un livre emprunté par l'un des membres. Intuitivement, le fait de rajouter des instances de membres ou de livres en laissant les autres dans les mêmes configurations ne bloque pas les actions possibles pour ces dernières. La monotonie est donc garantie si l'on se passe de synchronisations quantifiées. D'autre part, la section 4.3.4 propose des conditions sur les PASTD permettant de rendre le préordre WQO. Il s'agit de conditions limitant la forme des états et donc celle des PASTD. La monotonie étant ainsi obtenue, d'autres préordres similaires auraient pu être envisagés.

Une alternative au préordre proposé dans le chapitre précédent est un sous-ensemble de ce préordre qui permet de conserver la monotonie tout en gardant la

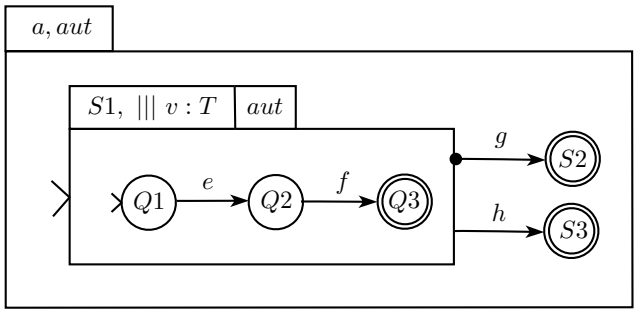

figure 5.11 – Exemple de PASTD monotone pour $\leq_F$

définition originale de l'opérateur d'entrelacement quantifié, c'est-à-dire celle permettant la synchronisation finale (cf. section 5.1.2). Ce préordre consiste à ne considérer comme plus grands que l'état s que les états s' tels que $s \preceq s'$ mais auxquels seuls des instances dans leurs configurations finales sont "ajoutées". Intuitivement, les branches que s' possède en plus par rapport à s doivent être finales au sens de la définition A.4 du prédicat $final$ de l'annexe A. Prenons, par exemple, le PASTD de la figure 5.11 et notons $\leq_F$ le préordre en question. Notons que l'état d'automate $S2$ est accessible depuis le PASTD composite $S1$ par une transition finale tandis que l'état $S3$ est lui accessible par une transition non finale. Sur la figure 5.12, on a $s_1 \leq_F s_2$ et $s_1 \not\leq_F s_3$ car la branche de l'instance 3 est finale dans s_2 mais pas dans s_3 ($Q3$ étant le seul état final du sous-automate local). Clairement, le choix de ce préordre permet de résoudre le problème de monotonie soulevé dans la section 5.1.2 car les branches finales d'un état ne sont pas bloquantes pour les autres instances. On a bien $s_1 \rightarrow (\mathsf{aut}_\circ, S2)$ et $s'_1 \rightarrow (\mathsf{aut}_\circ, S2)$ pour tout s'_1 tel que $s_1 \leq_F s'_1$. Il est, en outre, possible de résoudre le problème de couverture pour l'état $(\mathsf{aut}_\circ, S2)$ par exemple, en appliquant la procédure de vérification décrite dans le chapitre précédent. Cependant, le préordre $\leq_F$ possède plusieurs inconvénients majeurs.

1. Le problème de couverture d'un état s s'exprimant en fonction du préordre choisi, l'accessibilité de $\uparrow s$ n'aura pas le même sens selon que l'on considère $\preceq$ ou $\leq_F$. Dans le cadre du système de la bibliothèque, par exemple, il est plus pertinent de vérifier si la limite d'emprunts est toujours respectée pour un membre quels que soient les états locaux des autres membres, plutôt que de vérifier si elle est respectée seulement lorsque tous les autres sont dans leurs

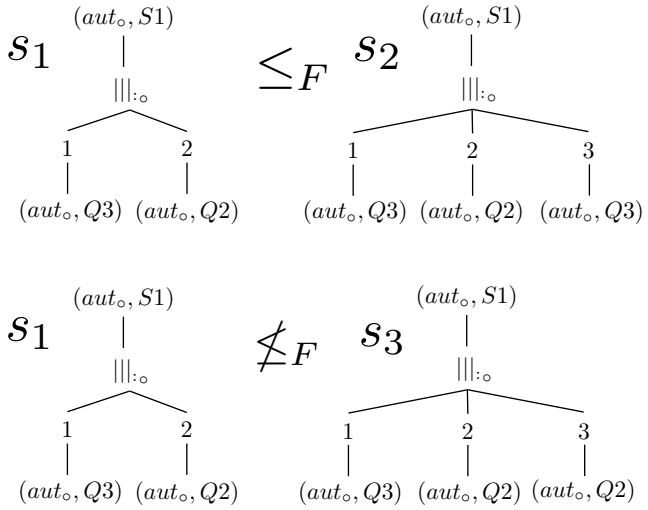

figure 5.12 – Exemples d'états comparables et incomparables pour $\leq_F$

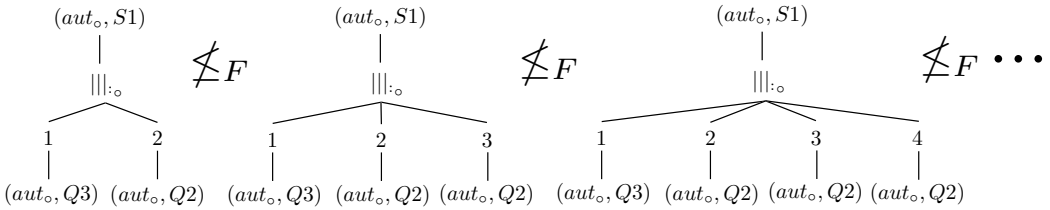

figure 5.13 – Exemple d'antichaine infinie pour $\leq_F$

états finaux. Le préordre $\leq_F$ n'est donc pas adapté dans ce cas.

2. Le préordre $\leq_F$ étant un sous-ensemble de $\preceq$, cela signifie qu'il est plus difficile de satisfaire la condition de WQO. En effet, la figure 5.13 montre un exemple d'antichaine infinie pour $\leq_F$ mais qui ne l'est pas pour $\preceq$.

3. Si l'on considère la même fonction de rang γ définie dans la section 4.4.2, la condition de *backward-downward monotony* (Définition 4.14) n'est pas satisfaite pour $\leq_F$. Ce qui a pour conséquence de ne pas garantir l'existence d'une pred-base finie dans le cas sans WQO. En effet, supposons qu'il existe $c \in \mathbb{N}$ satisfaisant la condition de *backward-downward monotony* et considérons la transition $s' \to s$ de la figure 5.14, où s' est l'un des états de l'antichaine de la figure 5.13 avec $c+2$ instances et s est l'état $(\mathsf{aut}_\circ, S3)$. Alors le plus petit état $s'' \leq_F s'$ satisfaisant $s'' \to s$ est celui donné dans la figure 5.14 et comporte

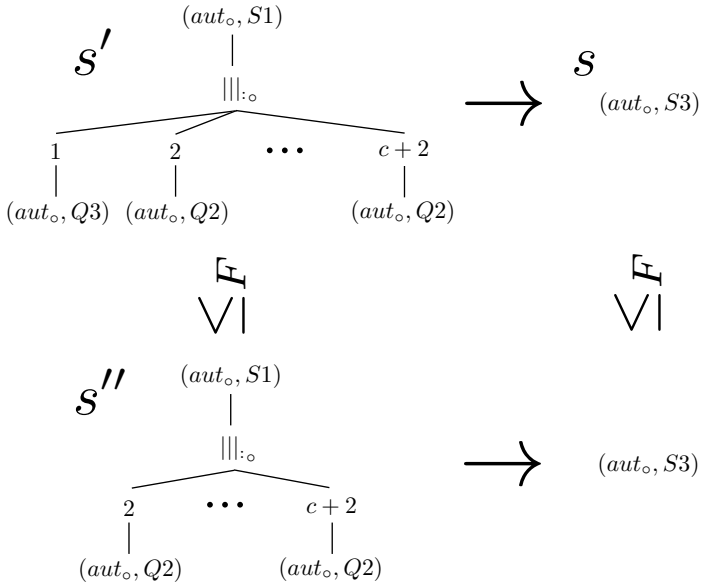

figure 5.14 – Contre-exemple pour la bdm

$c+1$ instances, *i.e.* $\gamma(s'') = c+1$, ce qui contredit la condition. En outre, dans le cas précis de l'état s, il est clair que $Pred(\uparrow s)$ ne possède pas de base finie. L'utilisation de transitions non finales est donc problématique pour $\leq_F$.

5.2.2 Approche par les WQO

Rappelons que les WQO sont une classe particulière de préordres qui sont à la fois bien fondés et qui ne contiennent pas d'antichaines infinies. Ils sont utiles dans la théorie des WSTS car ils permettent notamment la terminaison de la procédure de vérification de propriétés de couverture. Lorsque l'espace d'états d'un système de transitions est infini, il n'est souvent pas trivial de lui associer un WQO adéquat afin de former un WSTS. Parmi les WQO les plus connus, on retrouve par exemple l'ordre produit sur les vecteurs d'entiers $(\mathbb{N}^k, \leq)$, prouvé WQO par Dickson [29] et utilisé pour les réseaux de Petri, ou bien l'ordre sous-mot $(X^*, \sqsubseteq)$ de Higman [50] pour les *lossy channel systems*. Dans le cadre des PASTD, la structure arborescente des états ne nous permet pas d'utiliser de tels préordres. Il existe néanmoins plusieurs WQO intéressants sur ces structures de graphe, notamment celui sur les arbres proposé par Kruskal dans [60] ou l'ordre mineur de graphe par Robertson et Seymour dans [81]

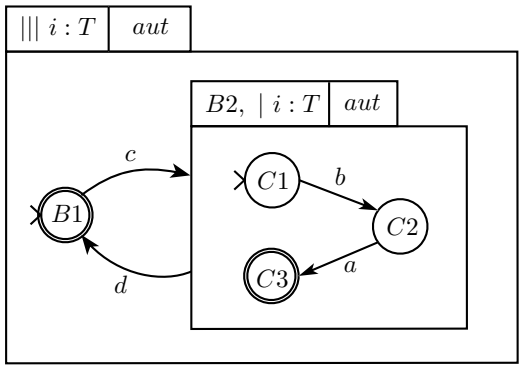

figure 5.15 – Exemple de PASTD

qui généralisent le lemme de Higman au cas des arbres et des graphes respectivement. Nous montrons dans cette section pourquoi ces préordres, bien que WQO, ne sont pas adaptés à notre étude de cas contrairement à l'ordre de sous-graphe induit présenté dans la section 4.3.3.

Comme présenté dans la section 4.3.4, on peut voir un état d'ASTD (ou de PASTD) comme un arbre dont les noeuds sont étiquetés par le type d'état ou une valeur de quantification. Parmi les WQO sur les arbres étiquetés, il y a celui présenté par Kruskal ou ceux qui utilisent l'ordre sous-graphe tout en restreignant l'ensemble ordonné à considérer comme celui proposé par Meyer dans [69]. Cependant, dans tous les cas, il est nécessaire de construire le WQO sur l'ensemble des arbres à partir d'un WQO sur l'ensemble des étiquettes. Considérons un PASTD $A = (F, \overrightarrow{T}, \overrightarrow{P})$. L'ensemble des étiquettes est donné par $X = \{(\mathsf{aut}_\circ, q_1), \ldots, (\mathsf{aut}_\circ, q_j), ||_\circ, |:_\circ, |||:_\circ\} \cup \bigcup_i P_i$. Si l'on prend le théorème de Kruskal, $(\mathcal{T}(X), \leq_T)$ est un WQO si $(X, \leq)$ est un WQO (cf. proposition 2.6). Si tous les domaines de paramètres P_i sont finis, un préordre trivialement WQO est l'identité sur X. Dans le cas contraire, on doit associer un WQO à chacun des P_i infini. Or, aucun ne respecte la sémantique intuitive des états et donc la propriété de monotonie. Prenons le PASTD de la figure 5.15, où le paramètre T a pour domaine $\mathbb{N}$. Si l'on considère l'ordre naturel sur les entiers comme WQO sur le domaine de paramètre, complété par l'identité sur les autres éléments de X, alors la figure 5.16 montre deux exemples problématiques. Le premier représente deux états s_1 et s_2 que l'on souhaite équivalents, au sens de la définition 4.8, mais qui ne le sont pas en prenant l'ordre de Kruskal. Le deuxième montre deux états $s_3 \leq_T s_4$

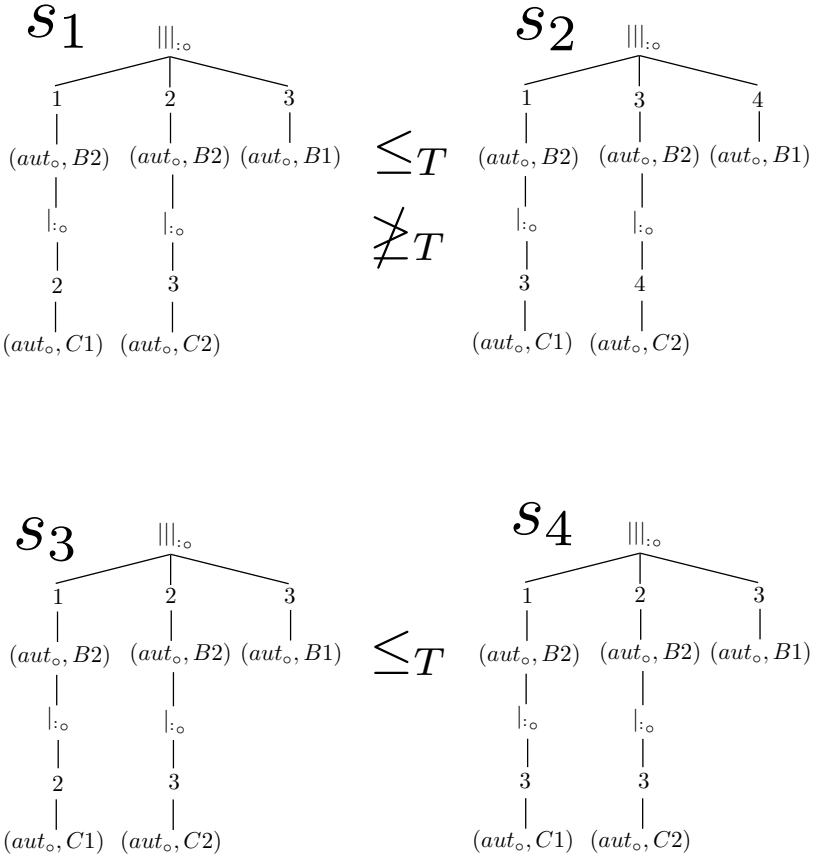

figure 5.16 – Exemples d'états comparables pour l'ordre naturel sur $\mathbb{N}$

mais que l'on souhaite incomparables. De même, si l'on considère le produit cartésien $\mathbb{N} \times \mathbb{N}$ comme WQO, on a un exemple sur la figure 5.17 de paire d'états équivalents selon l'ordre mais qui ne respectent pas la définition 4.8. Ainsi, la forme arborescente des états n'est pas la plus adaptée pour définir un WQO sur l'espace d'états des PASTD. Ce qui justifie la transformation de la structure d'arbre en graphe proposée par la définition 4.10 du chapitre 4. En effet, la transformation des arbres en graphes permet de remplacer l'ensemble infini d'étiquettes par un ensemble fini en reportant l'encodage des identités des instances par des arêtes et des sommets supplémentaires.

Les WQO intéressants sur les graphes sont le mineur proposé par Robertson et Seymour et l'ordre sous-graphe étudié par Ding (cf. section 2.3.2). Cependant, contrairement à l'ordre sous-graphe, le mineur de graphe permet la contraction d'arêtes, ce qui pose problème dans notre cas. Rappelons que pour tout état de PASTD s, $gr(s)$

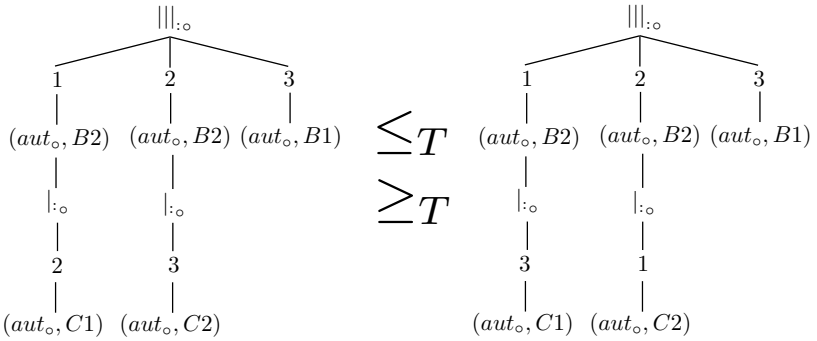

figure 5.17 – Exemple d'états comparables pour le produit cartésien $\mathbb{N}^2$

dénote la transformation en graphe de l'arbre syntaxique de s. Si l'on reprend le PASTD de la figure 5.15, la figure 5.18 montre un exemple de couple d'états s et s' tels que $gr(s)$ peut être obtenu à partir de $gr(s')$ par contraction d'arêtes. Les arêtes contractées sont représentées en rouge. Or, la sémantique des états s et s' ne permet pas, intuitivement, de les comparer. Ces contractions permettraient de fusionner deux instances différentes, ici les instances 2 et 3. L'ordre mineur contient plus de paires comparables que l'ordre sous-graphe sur l'espace d'états des PASTD et celles-ci ne sont pas souhaitables dans notre modélisation. Ainsi, à moins de pouvoir considérer une abstraction permettant de confondre plusieurs instances, l'ordre de mineur ne convient pas à notre étude.

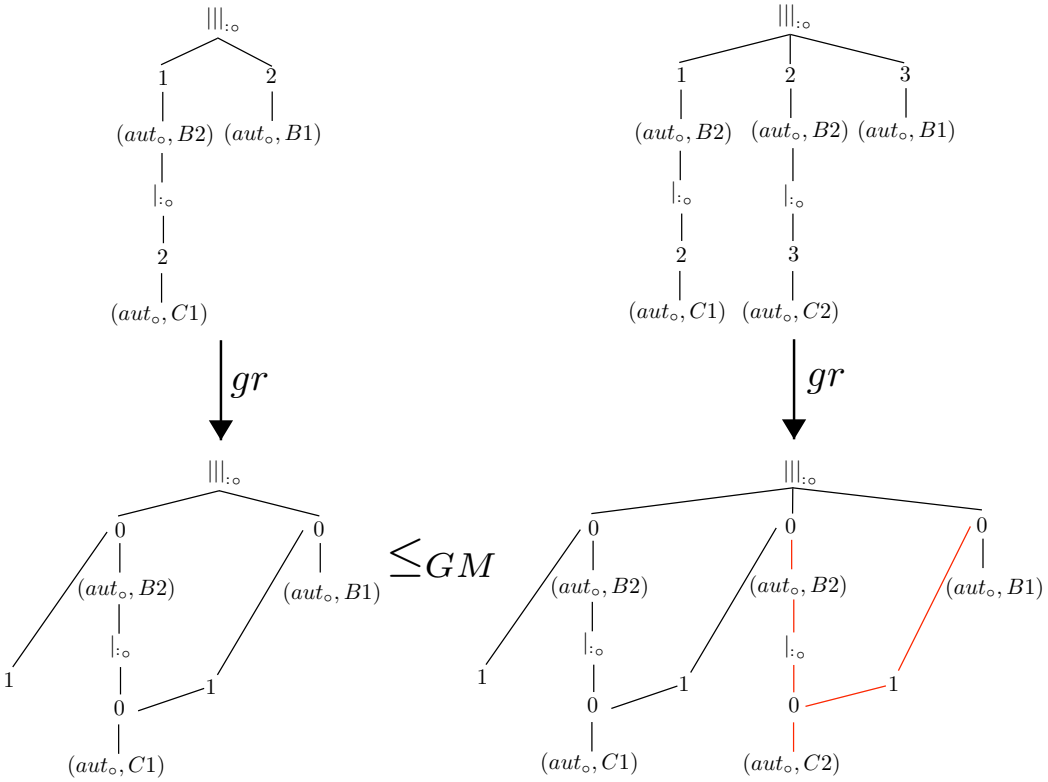

figure 5.18 – Exemple d'états comparables pour le préordre mineur

Conclusion

Synthèse et analyse des résultats

La spécification et la vérification de systèmes d'information sont des problèmes qui peuvent être abordés par le biais des différentes techniques de méthode formelle développées dans la littérature scientifique. Cependant, ces techniques sont souvent peu adaptées aux systèmes d'information dû aux diverses contraintes imposées aux modèles. Cette thèse exploite les caractéristiques principales des systèmes d'information afin de proposer des méthodes de spécification et de vérification plus appropriées.

Une étude préalable des systèmes d'information a mis en évidence leurs caractéristiques importantes pour leurs spécifications. La première caractéristique est l'appartenance à la classe des systèmes paramétrés. En effet, un système d'information peut être constitué de plusieurs processus partageant la même structure en nombre non borné. De plus, il existe en général plusieurs classes de processus, chaque entité du système en générant une. La deuxième est l'interaction entre ces processus qui en fait un système paramétré plus complexe que les systèmes à compteurs usuels. Ces interactions représentent les différentes associations du système. Une autre caractéristique est la forme des propriétés que l'on souhaite vérifier sur ces systèmes : elles sont, pour la plupart, des propriétés de sûreté ou d'accessibilité. Le problème de sûreté pouvant facilement être réduit à un problème d'accessibilité, cela justifie le choix d'une méthode de vérification basée sur l'analyse d'accessibilité ou de couverture.

L'approche qui a été proposée pour la modélisation de systèmes d'information est celle des PASTD, une variante des ASTD qui permet de modéliser la complexité des systèmes d'information en tant que systèmes paramétrés. Une spécification PASTD est un modèle formel doté d'une notation graphique facile à lire. De plus, il a été

montré que les PASTD constituent un langage plus expressif que celui des VASS avec reset. Ce qui permet de conclure l'indécidabilité de l'accessibilité en général. Néanmoins, le problème de couverture a été résolu pour des sous-classes de PASTD : les Bounded-PASTD, les 1-PASTD et les Flat-PASTD, qui sont des WSTS. D'un point de vue des systèmes d'information, les 1-PASTD représentent les systèmes dont les entités ne sont liées que par des associations d'entité faible et les Flat-PASTD des systèmes sans associations.

Afin de démontrer l'existence de pred-base effective pour les PASTD, le concept de RMTS a été introduit. Les RMTS permettent de déterminer un ensemble suffisant de conditions à satisfaire pour démontrer la pred-base effective pour tout système monotone sans se soucier de la condition de WQO. Finalement, les PASTD ont été prouvés comme satisfaisant les conditions des RMTS et donc de pred-base effective. Ainsi, le problème de couverture est décidable pour les Bounded-PASTD, car ils possèdent un WQO. En pratique, les RMTS servent de guide à la démonstration de pred-base effective. En effet, les conditions des RMTS mettent en avant l'idée intuitive de "noyau de transition" dont la taille maximale sur l'ensemble des transitions est utile à l'algorithme de calcul de la base de prédécesseurs. De plus, afin de satisfaire ces conditions, il a été nécessaire, par exemple, de considérer un espace d'états pour les PASTD qui diffère de celui des ASTD par l'ajout d'états abstraits. Cet ensemble d'états permet non seulement de satisfaire la pred-base effective en cas d'absence de WQO, mais aussi d'améliorer la procédure de calcul des prédécesseurs en factorisant des ensembles d'états concrets par un unique état abstrait.

Sans WQO, le modèle des RMTS n'est pas suffisant pour obtenir un algorithme de décision pour le problème de couverture. Cependant, la procédure de calcul des prédécesseurs reste correcte en tant que semi-algorithme. En effet, l'arbre d'exploration des états du parcours en arrière est un arbre à branchement fini grâce à la propriété de pred-base effective. Une recherche en largeur d'un état initial donne une procédure de semi-décision. Considérons un RMTS effectif $\mathcal{S} = (Q, \rightarrow, \leq, \gamma, c, d)$ et un état $s \in Q$ dont on cherche à déterminer s'il est couvrable. Si la couverture de s est vérifiable, alors la procédure de calcul de $Pred^*(\uparrow s)$ découvrira un état initial et s'arrêtera. Plus qu'un semi-algorithme, parfois la procédure pourra terminer, même si s n'est pas couvrable. C'est le cas lorsqu'une base pour $Pred^*(\uparrow s)$ a été complètement

calculée. En effet, même si $(Q, \leq)$ n'est pas un WQO, $(Pred^*(\uparrow s), \leq)$, qui dépend de l'état s, peut quand même être un WQO. Cela justifie l'importance pratique du résultat de semi-décidabilité. De plus, déterminer la complexité d'un algorithme basé sur les WQO n'est pas aisé. En pratique, s'il n'est pas possible d'obtenir une complexité raisonnable, l'argument du WQO peut ne pas être suffisant pour garantir la terminaison de l'algorithme implémenté. D'autre part, si l'on restreint la recherche des états initiaux par une valeur de *cutoff* sur la taille des états, et donc des paramètres dans le cas des PASTD, on obtient un algorithme de recherche plus efficace que la simple procédure qui consiste à vérifier l'accessibilité pour chaque instance de PASTD jusqu'à la valeur de *cutoff*.

Perspectives

L'expérimentation de la procédure de vérification de couverture pour les PASTD est l'une des perspectives manifestes de cette thèse. Pour cela, une implémentation complète du modèle des PASTD est nécessaire. Afin d'obtenir une implémentation efficace, il pourrait être important d'étudier la meilleure façon de représenter les états d'un PASTD et de les parcourir, la forme arborescente proposée en annexe n'étant pas la plus optimale en termes de gestion de l'espace. Puis, une étude empirique de différents modèles de systèmes d'information modélisés par des PASTD permettrait de déterminer les types de systèmes et de propriétés dont la vérification n'aboutirait pas en un temps raisonnable via la méthode des RMTS.

Deuxièmement, déterminer d'autres sous-classes de Bounded-PASTD peut être un problème intéressant. Contrairement aux Bounded-PASTD, les 1-PASTD et les Flat-PASTD possèdent des critères qui permettent de les reconnaître aisément. En outre, un algorithme de reconnaissance de Bounded-PASTD serait utile. En effet, le lemme 4.5 donne l'idée intuitive qu'il est possible de déterminer s'il existe des chemins (de sommets) non bornés dans les états en analysant la forme du PASTD. Ainsi, un algorithme préalable permettrait de confirmer si la procédure de vérification d'un PASTD terminera à coup sûr.

Finalement, l'incomplétude de la méthode des RMTS sans WQO nous oriente vers la question de l'existence de conditions alternatives de terminaison pour la procédure

de recherche en arrière. Le WQO étant une condition purement statique et ne dépendant que de la forme des états, on peut se demander s'il existe des conditions qui dépendraient de la relation de transition. En particulier, la notion de "noyau de transition" des RMTS pourrait être utile à déterminer ces conditions. De même, le calcul complet de $Pred^*(\uparrow s)$ ne devrait pas être nécessaire pour résoudre le problème de couverture de s. Vérifier la vacuité de l'intersection de l'ensemble des états initiaux avec $Pred^*(\uparrow s)$ pourrait être effectué avec un calcul partiel de $Pred^*(\uparrow s)$.

Bibliographie

[1] Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson et Yih-Kuen Tsay.
« General decidability theorems for infinite-state systems ».
Dans *Logic in Computer Science*, pages 313–321. IEEE, 1996.

[2] Parosh Aziz Abdulla, Frédéric Haziza et Lukáš Holík.
« All for the price of few ».
Dans *Verification, Model Checking, and Abstract Interpretation*, volume 7737 de *LNCS*, pages 476–495. Springer, 2013.

[3] Jean-Raymond Abrial.
The B-book : Assigning Programs to Meanings.
Cambridge University Press, 1996.

[4] Jean-Raymond Abrial.
Modeling in Event-B : System and Software Engineering.
Cambridge University Press, 2010.

[5] Bowen Alpern et Fred B. Schneider.
« Defining liveness ».
Information Processing Letters, 21:181–185, 1985.

[6] Krzysztof R. Apt et Dexter C. Kozen.
« Limits for automatic verification of finite-state concurrent systems ».
Information Processing Letters, 22(6):307–309, 1986.

[7] Tamarah Arons, Amir Pnueli, Sitvanit Ruah, Ying Xu et Lenore Zuck.
« Parameterized verification with automatically computed inductive assertions ».

Dans *Computer Aided Verification*, volume 2102 de *LNCS*, pages 221–234. Springer, 2001.

[8] Christel Baier et Joost-Pieter Katoen.
Principles of Model Checking.
The MIT Press, 2008.

[9] J. A. Bergstra et J. W. Klop.
« Process algebra for synchronous communication ».
Information and Control, 60(1):109–137, 1984.

[10] P. Bernus, G. Schmidt et K. Mertins, éditeurs.
Handbook on Architectures of Information Systems.
Springer, 1999.

[11] Yves Bertot et Pierre Castéran.
Interactive Theorem Proving and Program Development : Coq'Art The Calculus of Inductive Constructions.
Springer, 2010.

[12] Armin Biere, Alessandro Cimatti, Edmund Clarke et Yunshan Zhu.
« Symbolic model checking without BDDs ».
Dans *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 de *LNCS*, pages 193–207. Springer, 1999.

[13] Jesse D. Bingham et Alan J. Hu.
« Empirically efficient verification for a class of infinite-state systems ».
Dans *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 de *LNCS*, pages 77–92. Springer, 2005.

[14] Gregor V. Bochmann.
« Finite state description of communication protocols ».
Computer Networks, 2(4):361–372, 1978.

[15] Tommaso Bolognesi et Ed Brinksma.
« Introduction to the ISO specification language LOTOS ».
Computer Networks and ISDN Systems, 14(1):25–59, 1987.

[16] Stephen D. Brookes, Charles A. R. Hoare et Andrew W. Roscoe.
« A theory of communicating sequential processes ».
Journal of the ACM, 31(3):560–599, 1984.

[17] Randal E. Bryant.
« Graph-based algorithms for Boolean function manipulation ».
IEEE Transactions on Computers, C-35:677–691, 1986.

[18] J. Richard Büchi.
« Symposium on Decision Problems : On a decision method in a restricted second order arithmetic ».
Dans *Logic, Methodology and Philosophy of Science*, volume 44 de *Studies in Logic and the Foundations of Mathematics*, pages 1–11. Elsevier, 1966.

[19] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu et Helmut Veith.
« Counterexample-guided abstraction refinement ».
Dans *Computer Aided Verification*, volume 1855 de *LNCS*, pages 154–169. Springer, 2000.

[20] Edmund Clarke, Muralidhar Talupur et Helmut Veith.
« Environment abstraction for parameterized verification ».
Dans *Verification, Model Checking, and Abstract Interpretation*, volume 3855 de *LNCS*, pages 126–141. Springer, 2006.

[21] Edmund M. Clarke et E. Allen Emerson.
« Design and synthesis of synchronization skeletons using branching-time temporal logic ».
Dans *Logic of Programs*, volume 131 de *LNCS*, pages 52–71. Springer, 1982.

[22] Edmund M. Clarke, E. Allen Emerson et A. Prasad Sistla.
« Automatic verification of finite-state concurrent systems using temporal logic specifications ».
Programming Languages and Systems, 8(2):244–263, 1986.

[23] Edmund M. Clarke, Orna Grumberg et Kiyoharu Hamaguchi.
« Another look at LTL model checking ».
Formal Methods in System Design, 10(1):47–71, 1997.

[24] National Research Council.
Industrial Methods for the Effective Development and Testing of Defense Systems.
The National Academies Press, 2012.

[25] Costas Courcoubetis, Moshe Y. Vardi, Piere Wolper et Mihalis Yannakakis.
« Memory-efficient algorithms for the verification of temporal properties ».
Formal Methods in System Design, 1(2):275–288, 1992.

[26] Patrick Cousot et Radhia Cousot.
« Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints ».
Dans *Principles of Programming Languages*, pages 238–252. ACM, 1977.

[27] Marco Daniele, Fausto Giunchiglia et Moshe Y. Vardi.
« Improved automata generation for linear temporal logic ».
Dans *Computer Aided Verification*, volume 1633 de *LNCS*, pages 249–260. Springer, 1999.

[28] Giorgio Delzanno, Arnaud Sangnier et Gianluigi Zavattaro.
« Parameterized verification of ad hoc networks ».
Dans *Concurrency Theory*, volume 6269 de *LNCS*, pages 313–327. Springer, 2010.

[29] Leonard Eugene Dickson.
« Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors ».
American Journal of Mathematics, 35(4):413–422, 1913.

[30] Guoli Ding.
« Subgraphs and well-quasi-ordering ».
Journal of Graph Theory, 16(5):489–502, 1992.

[31] Catherine Dufourd, Alain Finkel et Philippe Schnoebelen.
« Reset nets between decidability and undecidability ».
Dans *Automata, Languages and Programming*, volume 1443 de *LNCS*, pages 103–115. Springer, 1998.

[32] Michel Embe-Jiague, Marc Frappier, Frederic Gervais, Pierre Konopacki, Regine Laleau, Jeremy Milhau et Richard St-Denis.
« Model-Driven Engineering of Functional Security Policies ».
Dans *International Conference on Enterprise Information Systems*, pages 374–379. SciTePress, 2010.

[33] E. Allen Emerson et Joseph Y. Halpern.
« "Sometimes" and "not never" revisited : on branching versus linear time temporal logic ».
Journal of the ACM, 33:151–178, 1986.

[34] E. Allen Emerson et Vineet Kahlon.
« Reducing model checking of the many to the few ».
Dans *Automated Deduction*, volume 1831 de *LNCS*, pages 236–254. Springer, 2000.

[35] E. Allen Emerson et Kedar S. Namjoshi.
« Reasoning about rings ».
Dans *Principles of Programming Languages*, pages 85–94. ACM, 1995.

[36] E. Allen Emerson et A. Prasad Sistla.
« Symmetry and model checking ».
Formal Methods in System Design, 9(1-2):105–131, 1996.

[37] Alain Finkel.
« A generalization of the procedure of Karp and Miller to well structured transition systems ».
Dans *Automata, Languages and Programming*, volume 267 de *LNCS*, pages 499–508. Springer, 1987.

[38] Alain Finkel.
« Decidability of the termination problem for completely specified protocols ».
Distributed Computing, 7(3):129–135, 1994.

[39] Alain Finkel et Philippe Schnoebelen.
« Well-structured transition systems everywhere ! ».
Theoretical Computer Science, 256(1):63–92, 2001.

[40] Marc Frappier, Benoît Fraikin, Romain Chossart, Raphaël Chane-Yack-Fa et Mohammed Ouenzar.
« Comparison of model checking tools for information systems ».
Dans *Formal Methods and Software Engineering*, volume 6447 de *LNCS*, pages 581–596. Springer, 2010.

[41] Marc Frappier, Frédéric Gervais, Régine Laleau et Benoît Fraikin.
« Algebraic state transition diagrams ».
Rapport Technique, Université de Sherbrooke, 2008, `http://www.dmi.usherb.ca/~frappier/Papers/astd.pdf`.

[42] Marc Frappier, Frédéric Gervais, Régine Laleau, Benoît Fraikin et Richard St-Denis.
« Extending statecharts with process algebra operators ».
Innovations in Systems and Software Engineering, 4(3):285–292, 2008.

[43] Marc Frappier et Richard St-Denis.
« EB^3 : an entity-based black-box specification method for information systems ».
Software and Systems Modeling, 2(2):134–149, 2003.

[44] Rob Gerth, Ruurd Kuiper, Doron Peled et Wojciech Penczek.
« A partial order approach to branching time logic model checking ».
Information and Computation, 150(2):132–152, 1999.

[45] Jay Gischer.
« Shuffle languages, Petri nets, and context-sensitive grammars ».
Communications of the ACM, 24(9):597–605, 1981.

[46] Patrice Godefroid.
« Using partial orders to improve automatic verification methods ».
Dans *Computer Aided Verification*, volume 531 de *LNCS*, pages 176–185. Springer, 1991.

[47] Susanne Graf et Hassen Saïdi.
« Construction of abstract state graphs with PVS ».
Dans *Computer Aided Verification*, volume 1254 de *LNCS*, pages 72–83. Springer, 1997.

[48] Youssef Hanna, David Samuelson, Samik Basu et Hridesh Rajan.
« Automating cut-off for multi-parameterized systems ».
Dans *Formal Methods and Software Engineering*, volume 6447 de *LNCS*, pages 338–354. Springer, 2010.

[49] David Harel.
« Statecharts : a visual formalism for complex systems ».
Science of Computer Programming, 8(3):231–274, 1987.

[50] Graham Higman.
« Ordering by divisibility in abstract algebras ».
Dans *Proceedings of the London Mathematical Society*, volume s3-2, pages 326–336, 1952.

[51] Charles A. R. Hoare.
« Communicating sequential processes ».
Communications of the ACM, 21(8):666–677, 1978.

[52] Gerard Holzmann.
Spin Model Checker, the : Primer and Reference Manual.
Addison-Wesley, 2003.

[53] John Hopcroft et Jean-Jacques Pansiot.
« On the reachability problem for 5-dimensional vector addition systems ».
Theoretical Computer Science, 8(2):135–159, 1979.

[54] Graham Hutton.
Introduction to HOL : A Theorem Proving Environment for Higher Order Logic.
Cambridge University Press, 1993.

[55] Alexander Kaiser, Daniel Kroening et Thomas Wahl.
« Dynamic cutoff detection in parameterized concurrent programs ».
Dans *Computer Aided Verification*, volume 6174 de *LNCS*, pages 645–659. Springer, 2010.

[56] Shmuel Katz et Doron Peled.
« Defining conditional independence using collapses ».
Dans *Semantics for Concurrency*, WORKSHOPS COMP., pages 262–280. Springer, 1990.

[57] Yonit Kesten et Amir Pnueli.
« Control and data abstraction : the cornerstones of practical formal verification ».
International Journal on Software Tools for Technology Transfer, 2(4):328–342, 2000.

[58] Barbara König et Jan Stückrath.
« A general framework for well-structured graph transformation systems ».
Dans *Concurrency Theory*, volume 8704 de *LNCS*, pages 467–481. Springer, 2014.

[59] Saul Kripke.
« Semantical considerations on modal logic ».
Acta Philosophical Fennica, 16:83–94, 1963.

[60] Joseph B. Kruskal.
« Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture ».
Transactions of the American Mathematical Society, pages 210–225, 1960.

[61] Robert P. Kurshan et Kenneth L. McMillan.
« A structural induction theorem for processes ».
Information and Computation, 117(1):1–11, 1995.

[62] C. Y. Lee.
« Representation of switching circuits by binary decision programs ».
The Bell Systems Technical Journal, 38(4):985–999, 1959.

[63] David Lesens, Nicolas Halbwachs et Pascal Raymond.
« Automatic verification of parameterized networks of processes ».
Theoretical Computer Science, 256(1):113–144, 2001.

[64] Richard J. Lipton.
« Reduction : A method of proving properties of parallel programs ».
Communications of the ACM, 18(12):717–721, 1975.

[65] The MathWorks.
Stateflow User's Guide.
The MathWorks, 2017.

[66] Antoni W. Mazurkiewicz.
« Trace Theory ».
Dans *Advances in Petri Nets*, volume 255 de *LNCS*, pages 279–324. Springer, 1986.

[67] Kenneth L. McMillan.
« *Symbolic model checking : an approach to the state explosion problem* ».
Thèse de doctorat, Carnegie Mellon University, 1992.

[68] Kenneth L. McMillan.
« Verification of infinite state systems by compositional Mmodel checking ».
Dans *Correct Hardware Design and Verification Methods*, volume 1703 de *LNCS*, pages 219–234. Springer, 1999.

[69] Roland Meyer.
« On boundedness in depth in the π-calculus ».
Dans *Ifip International Conference On Theoretical Computer Science*, volume 273 de *IFIPAICT*, pages 477–489. Springer, 2008.

[70] Robin Milner.
A Calculus of Communicating Systems.
volume 92 de *LNCS*. Springer, 1980.

[71] Robin Milner.
Communication and Concurrency.
Prentice Hall, 1989.

[72] Robin Milner, Joachim Parrow et David Walker.
« A calculus of mobile processes, I ».
Information and Computation, 100(1):1–40, 1992.

[73] C. St. J. A. Nash-Williams.
« On well-quasi-ordering finite trees ».
Mathematical Proceedings of the Cambridge Philosophical Society, 59(4):833–835, 1963.

[74] Tobias Nipkow, Markus Wenzel et Lawrence C. Paulson.
Isabelle/HOL : A Proof Assistant for Higher-order Logic.
Springer, 2002.

[75] Doron Peled.
« All from one, one for all : on model checking using representatives ».
Dans *Computer Aided Verification*, volume 697 de *LNCS*, pages 409–423. Springer, 1993.

[76] James Lyle Peterson.
Petri Net Theory and the Modeling of Systems.
Prentice Hall, 1981.

[77] Amir Pnueli.
« The temporal logic of programs ».
Dans *Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[78] Amir Pnueli, Sitvanit Ruah et Lenore Zuck.
« Automatic deductive verification with invisible invariants ».

Dans *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 de *LNCS*, pages 82–97. Springer, 2001.

[79] Amir Pnueli, Jessie Xu et Lenore Zuck.
« Liveness with (0,1,∞)- counter abstraction ».
Dans *Computer Aided Verification*, volume 2404 de *LNCS*, pages 107–122. Springer, 2002.

[80] Jean-Pierre Queille et Joseph Sifakis.
« Specification and verification of concurrent systems in CESAR ».
Dans *International Symposium on Programming*, volume 137 de *LNCS*, pages 337–351. Springer, 1982.

[81] Neil Robertson et Paul D. Seymour.
« Graph minors. XX. Wagner's conjecture ».
Journal of Combinatorial Theory, 92(2):325–357, 2004.

[82] Neil Robertson et Paul D. Seymour.
« Graph minors XXIII. Nash-Williams' immersion conjecture ».
Journal of Combinatorial Theory, 100(2):181–205, 2010.

[83] A. W. Roscoe, C. A. R. Hoare et Richard Bird.
The Theory and Practice of Concurrency.
Prentice Hall, 1997.

[84] James E. Rumbaugh, Ivar Jacobson et Grady Booch.
The Unified Modeling Language Reference Manual.
Addison-Wesley, 1999.

[85] Sylvain Schmitz et Philippe Schnoebelen.
« Algorithmic aspects of WQO theory ».
Lecture Notes, 2012.

[86] Antti Siirtola et Juha Kortelainen.
« Algorithmic verification with multiple and nested parameters ».
Dans *Formal Methods and Software Engineering*, volume 5885 de *LNCS*, pages 561–580. Springer, 2009.

[87] Antti Siirtola et Juha Kortelainen.
« Parameterised process algebraic verification by precongruence reduction ».
Dans *Application of Concurrency to System Design*, pages 158–167. IEEE, 2009.

[88] A. Prasad Sistla et Edmund M. Clarke.
« The complexity of propositional linear temporal logics ».
Journal of the ACM, 32(3):733–749, 1985.

[89] Colin Snook.
« iUML-B statemachines ».
Dans *Proceedings of the Rodin User and Developer Workshop*, pages 29–30. University of Southampton, 2014.

[90] Antti Valmari.
« Stubborn sets for reduced state space generation ».
Dans *Advances in Petri Nets*, volume 483 de *LNCS*, pages 491–515. Springer, 1991.

[91] Moshe Y. Vardi.
« Alternating automata and program verification ».
Dans *Computer Science Today*, volume 1000 de *LNCS*, pages 471–485. Springer, 1995.

[92] Moshe Y. Vardi et Pierre Wolper.
« An automata-theoretic approach to automatic program verification ».
Dans *Logic in Computer Science*, pages 332–344. IEEE, 1986.

[93] Moshe Y. Vardi et Pierre Wolper.
« Reasoning about infinite computations ».
Information and Computation, 115(1):1–37, 1994.

[94] Dimitris Vekris, Frédéric Lang, Catalin Dima et Radu Mateescu.
« Verification of EB^3 specifications using CADP ».
Dans *Integrated Formal Methods*, volume 7940 de *LNCS*, pages 61–76. Springer, 2013.

[95] Pierre Wolper et Vinciane Lovinfosse.
« Verifying properties of large sets of processes with network invariants ».
Dans *Automatic Verification Methods for Finite State Systems*, volume 407 de *LNCS*, pages 68–80. Springer, 1990.

Annexe A

Parameterized ASTD

A.1 Preliminaries

A *Transition System* (TS) is a pair $S = (Q, \rightarrow)$, where Q is a set (of states) and $\rightarrow \subseteq Q \times Q$ is the set of transitions between states. We write $q \rightarrow q'$ for $(q, q') \in \rightarrow$, and $q \xrightarrow{*} q'$ if either $q = q'$ or there exists a finite sequence of transitions $q \rightarrow q_1 \rightarrow q_2 \rightarrow \cdots \rightarrow q'$, called a path. We define $Pred(q) = \{q' \in Q \mid q' \rightarrow q\}$ as the set of immediate predecessors of q and $Pred^*(q) = \{q' \in Q \mid q' \xrightarrow{*} q\}$ as the set of all predecessors of q.

A *quasi-ordering* (qo) is a reflexive and transitive binary relation $\leq$ on a set X; we also say that $(X, \leq)$ is a qo. A *partial ordering* (po) is an antisymmetric qo.

Let $(X, \leq)$ be a po. For all $s_1, s_2 \in X$, we say that $s_3 \in X$ is the *supremum* (or sup) of s_1 and s_2 noted $sup(s_1, s_2)$ if s_3 is an upper bound of s_1 and s_2 (*i.e.* $s_1 \leq s_3$ and $s_2 \leq s_3$), and for all upper bounds $s_4 \in X$, we have $s_3 \leq s_4$.

Let $\leq$ be a qo on a set X. An *upward-closed* set is a subset $Y \subseteq X$ such that if $x \leq y$ and $x \in Y$ then $y \in Y$. For some $x \in X$, we write $\uparrow x = \{y \in X \mid x \leq y\}$ its upward-closure, and for some $Y \subseteq X$, $\uparrow Y = \bigcup_{x \in Y} \uparrow x$. A *basis* of an upward-closed subset $Y \subseteq X$ is any set B such that $Y = \uparrow B$. We say that an upward closed set Y has a *finite basis* if there exists some finite basis B of Y.

A qo $(X, \leq)$ is *well-founded* if there is no infinite sequence $(x_n)_{n \in \mathbb{N}}$ over X such that $x_{i+1} \leq x_i$ for every $i \in \mathbb{N}$. It is a *well-quasi-ordering* (wqo) if every infinite sequence $(x_n)_{n \in \mathbb{N}}$ over X contains an increasing pair, i.e. $\exists i < j$ such that $x_i \leq x_j$. If

the wqo is a po, then it is called a *well partial order* (wpo). An *antichain* is a set in which each pair of different elements is incomparable.

We call *permutation* any bijection $f : X \to X$. We denote by id_X the identity on the set X. We denote by $dom(f)$ the domain of the function $f : X \to Y$, $ran(f)$ its image , *i.e.* $ran(f) = \{y \in Y \mid \exists x \in X \cdot f(x) = y\}$, $X \lhd f$ the domain restriction of the function to X and by $f \lhd\!\!\!- g$ the overriding of $f : X \to Y$ by $g : W \to Y$, *i.e.* $(f \lhd\!\!\!- g)(x) = g(x)$ if $x \in W$ and $(f \lhd\!\!\!- g)(x) = f(x)$ otherwise.

A.2 A Fragment of the ASTD Language

Algebraic State-Transition Diagram (ASTD) [42] is a graphical notation combining automata, statecharts and process algebra to describe complex dynamic systems like information systems. They are closely related to process algebras like CSP [51], CCS [71], ACP [9], LOTOS [15] and EB^3 [43]. Essentially, ASTDs are like a process algebra with hierarchical automata as elementary process expressions. Automata can be combined freely with process algebra operators. ASTDs have a structured operational semantics in the Plotkin style, which was first used by Milner for CCS and later on for LOTOS and CSP [83]. They are recursively defined structures and includes many types like automaton, synchronization, quantified choice and quantified interleaving.

ASTDs are useful to model information systems as they provide a concise, visual and formal mechanism for specifying all the scenarios of an information system [42]. For example, they make explicit the handling of entity instances by using quantifications. Furthermore, the model has been used in [32] to specify access control and security policies.

For instance, an ASTD model of a library system is given in Figure A.1. The system manage loans of books by members. The ASTD on the top left hand corner, whose name is *library*, is a synchronization between two other ASTDs which consist in a quantified interleaving on the set T_m of members for the the process *member* and a quantified interleaving on the set T_b of books for the process *book*. The process $book(bId)$ is described by the ASTD on the bottom right corner. A book is acquired by the library. It can be discarded if it is not lent. If acquired, a book bId can "choose"

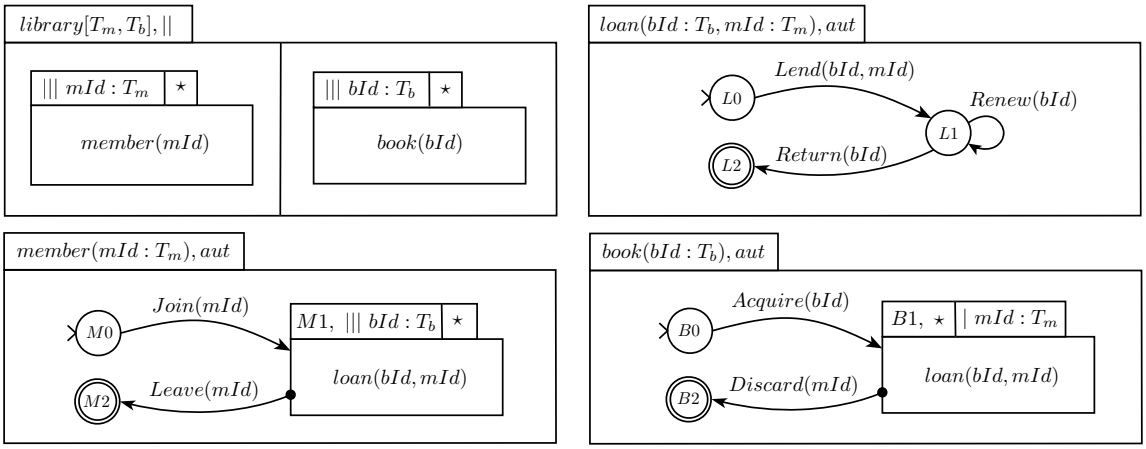

Figure A.1 – Example of a library system

a member m to run the process $loan(bId, mId)$. The member process is similar except that a member m runs the $loan(bId, mId)$ process for every book.

Definition A.1. *ASTDs are defined inductively and include the following types: Automaton, Kleene Closure, Choice, Synchronization, Quantified Choice, Quantified Interleaving, ASTD Call and Elementary ASTD. The signature of an ASTD is given by a tuple whose first element is a tag corresponding to the ASTD type.*

1. *The Elementary ASTD* $\langle \mathsf{elem} \rangle$ *is the simplest ASTD that can be defined. It is used to represent some state of the Automaton ASTD.*

2. *An Automaton ASTD is similar to a traditional automaton, except that its states can be of any ASTD type. It is denoted by* $\langle \mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0 \rangle$*, where name is the name of the ASTD structure,* Σ *a set of events,* N *a set of state names,* ν *a function that maps each state name to an ASTD,* δ *a transition relation,* $SF \subseteq N$ *a set of shallow final states,* $DF \subseteq N$ *a set of deep final states and* $n_0 \in N$ *an initial state. The transition relation* δ *is given by a set of tuples* $\langle n_1, n_2, \sigma, final? \rangle$*, where* n_1, n_2 *are two state names,* $\sigma \in \Sigma$ *is an event and* $final?$ *is a boolean denoting a transition that can be fired only from a final state (the definition of a final ASTD state is given in the sequel). An event is noted* $l(v_1, \ldots, v_n)$ *where* l *is called the event label, and* v_i *are event parameters.*

Function α extracts the label of an event: $\alpha(l(v_1, \ldots, v_n)) = l$. By extension, $\alpha(a)$ returns the set of labels occurring in ASTD a.

3. *The Kleene Closure is given by a tuple $\langle \star, n, b \rangle$, where n is the ASTD name and b an ASTD. It allows for iteration on ASTD b a finite number of time (including zero). An iteration is completed when the component ASTD has reached a final state.*

4. *A Choice ASTD $\langle |, n, l, r \rangle$, where n is the ASTD name, allows a choice between two component ASTDs l and r like in a process algebra.*

5. *A Synchronization ASTD $\langle ||, n, \Delta, l, r \rangle$ between two component ASTDs l and r describes a behavior where l and r run concurrently by executing events, whose labels are in Δ, at the same time and interleaving the other events. We denote by $||$ the Synchronization ASTD where $\Delta = \alpha(l) \cap \alpha(r)$.*

6. *A Quantified Choice ASTD $\langle |:, n, x, T, b \rangle$ allows to pick a value v from a finite set T and execute component ASTD b where every occurrence of the variable x is replaced by the value v.*

7. *A Quantified Interleaving ASTD $\langle |||:, n, x, T, b \rangle$ allows for executing as many interleaving instances of ASTD b as the number of values in a finite set T. Each instance is executed such that x is replaced by the corresponding value.*

8. *Finally, it is possible to call an ASTD defined in another diagram together with some parameters, but not recursively in our case. According to this restriction, we consider ASTD Calls as a syntactic sugar, that is we can always replace a call by the called ASTD. In the following, we do not consider ASTD Calls for the sake of simplicity.*

Besides, in order to represent a concrete system, an ASTD must not contain any free variable. In such ASTDs, we consider that every variable used in a parameterized event is bound either by a Quantified Choice or a Quantified Interleaving.

Figure A.2 shows some examples of ASTDs in their graphical notations. For example, the ASTD on the top left hand corner is an Automaton ASTD, whose name is

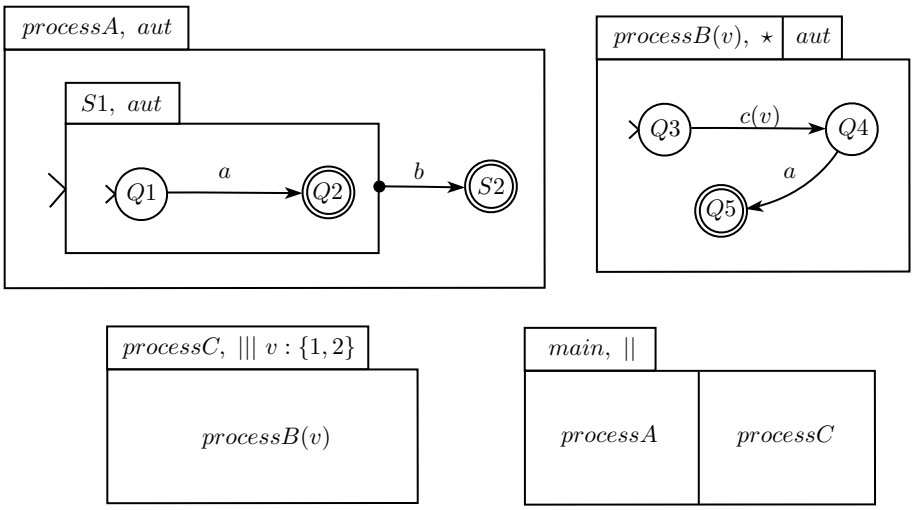

Figure A.2 – Example of ASTD

processA. It is composed of two states and one transition labeled by the event b. The first state $S1$ is itself an Automaton ASTD and the second one $S2$ is an Elementary ASTD. The symbol > denotes the initial automaton state and the double circle a final automaton state. The transition that is decorated by a bullet at its source means that its boolean $final?$ is true, *i.e.* it can be fired only if $S1$ is in its final state $Q2$. An example of an ASTD with free variable is the ASTD $processB(v)$ that contains the variable v. As $c(v)$ is a parameterized event, we must instantiate $processB(v)$ with a value to obtain a concrete event. The ASTD *processC* is a Quantified Interleaving ASTD that calls the $processB(v)$ ASTD and binds the variable v to its quantified operator. The ASTD *main*, that synchronizes two sub-ASTDs, is another example of an ASTD Call. Note that naming the sub-ASTDs is not always necessary and that we can coalesce ASTD boxes when the outer ASTD is a unary operator. The coalescing is indicated by adding the tab of the inner ASTD to the outer unary one, like in $processB(v)$.

The main differences with the ASTD described in [41] are that Automata are simplified, that there is no Sequence ASTD, no Guard ASTD and that Quantified Synchronizations are replaced by Quantified Interleavings. All definitions in this section are slightly modified from [41] according to the previous considerations on the ASTD language.

As an ASTD describes a dynamical system, for each ASTD, we can define a set of ASTD states. ASTD states are given by the following definition.

Definition A.2. *A state expression of an ASTD is given inductively by an expression according to the following types, where the first component is a label identifying the state type (since an ASTD state is a sum type):*

1. $\langle \mathsf{elem}_\circ \rangle$*, the elementary state, which is the only type of state for an Elementary ASTD;*

2. $\langle \mathsf{aut}_\circ, n, s \rangle$*, an automaton state, where* n *is the name of a state of the automaton and* s *a sub-state;*

3. $\langle \star_\circ, \mathit{started?}, [\bot \mid s] \rangle$*, a Kleene closure state, with a boolean started? indicating whether the first iteration has been started, and where* s *is a sub-state, and* $\bot$ *is used instead of* s *when started? is* false*;*

4. $\langle |_\circ, [\bot \mid \mathsf{left} \mid \mathsf{right}], [\bot \mid s] \rangle$*, a choice state, with a variable indicating which choice has been made if it has been and where* s *is a sub-state;*

5. $\langle ||_\circ, s_l, s_r \rangle$*, a synchronization state, where* s_l *and* s_r *are sub-states;*

6. $\langle |:_\circ, [\bot \mid v], [\bot \mid s] \rangle$*, a quantified choice state, where* v *is the value of the quantified choice (if it has been made) and* s *a sub-state;*

7. $\langle |||:_\circ, f \rangle$*, a quantified interleaving state, where* f *is a function, whose domain of definition is a non-empty and finite set of values and whose codomain is the set of ASTD states.*

For example, a state of the ASTD *processC* from Figure A.2 can be given by the expression $(|||:_\circ, \{1 \mapsto (\star_\circ, \text{true}, (\mathsf{aut}_\circ, Q4, (\mathsf{elem}_\circ))), 2 \mapsto (\star_\circ, \text{false}, \bot)\})$, which means that $processB(1)$ is in state $Q4$ and $processB(2)$ has not yet started.

Some states of an ASTD may be initial or final. This is specified by a function *init* that returns the initial state of an ASTD and a predicate *final* that determines whether a state is considered as final.

Definition A.3. *Let a be an ASTD. The function init, which returns the initial state of a, is defined as follows:*

1. $init((\mathit{elem})) = (\mathit{elem}_\circ)$, *the elementary state;*

2. $init((\mathit{aut}, n, \Sigma, N, \nu, \delta, SF, DF, n_0)) = (\mathit{aut}_\circ, n_0, init(\nu(n_0)))$, *the automaton state, whose name* n_0 *is the initial automaton state's one, and whose sub-state is define recursively by the initial state of* n_0*;*

3. $init((\star, n, b)) = (\star_\circ, \text{false}, \bot)$, *the Kleene closure state, where the first iteration has not been started;*

4. $init((|, n, l, r)) = (|_\circ, \bot, \bot)$, *the choice state, where the choice has not been made;*

5. $init((||, n, \Delta, l, r)) = (||_\circ, init(l), init(r))$, *the synchronization state, where the two component states are in their initial states;*

6. $init((|:, n, x, T, b)) = (|:_\circ, \bot, \bot)$, *the quantified choice state, where the choice has not been made;*

7. $init((|||:, n, x, T, b)) = (|||:_\circ, T \times \{init(b)\})$, *the quantified interleaving state, where each component state is set to its initial state.*

Definition A.4. *Let* s *be an ASTD state and* a *an ASTD of the same type. The predicate* $final(s)$*, which determines if* s *is final in* a*, is defined as follows:*

1. *If* $s = (\mathit{elem}_\circ)$*, then* $final(s) \equiv \text{true}$*, the elementary state is final in the Elementary ASTD;*

2. *If* $s = (\mathit{aut}_\circ, n, ss)$ *and* $a = (\mathit{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)$*, then* $final(s) \equiv (n \in DF \wedge final(ss)) \vee n \in SF$*, i.e. the automaton state* s *is final in the Automaton ASTD* a *either if the state is marked as deep final and the corresponding sub-state* ss *is final or if the state is marked as shallow final;*

3. *If* $s = (\star_\circ, started?, ss)$*, then* $final(s) \equiv final(ss) \vee \neg started?$*, i.e.* s *is final either if the sub-state* ss *is final or if the first iteration has not been started;*

4. *If* $s = (|_\circ, \bot, \bot)$ *and* $a = (|, n, l, r)$*, then* $final(s) \equiv final(init(l)) \vee final(init(r))$*, i.e.* s *is final if either the initial state of the left component ASTD or the initial state of the right component is final;*

5. *If* $s = (|_\circ, \_, ss)$, *then* $final(s) \equiv final(ss)$, *i.e. s is final if the sub-state ss is final;*

6. *If* $s = (||_\circ, s_l, s_r)$, *then* $final(s) \equiv final(s_l) \wedge final(s_r)$, *i.e. s is final if both sub-states are final;*

7. *If* $s = (|:_\circ, \bot, \bot)$ *and* $a = (|:, n, x, T, b)$, *then* $final(s) \equiv final(init(b))$, *i.e. s is final if the initial state of the component ASTD is final;*

8. *If* $s = (|:_\circ, v, ss)$ *with* $v \neq \bot$, *then* $final(s) \equiv final(ss)$, *i.e. s is final if the sub-state ss is final;*

9. *If* $s = (|||:_\circ, f)$ *and* $a = (|||:, n, x, T, b)$, *then* $final(s) \equiv \forall v \in T \cdot final(f(v))$, *i.e. s is final if each sub-state is final.*

Consider the example of the ASTD *processA* of Figure A.2. The initial state of the ASTD is $(\mathsf{aut}_\circ, S1, (\mathsf{aut}_\circ, Q1, (\mathsf{elem}_\circ)))$ and the only final state is $(\mathsf{aut}_\circ, S2, (\mathsf{elem}_\circ))$.

As seen previously, Automaton ASTDs can be defined with free variables, which are used in parameterized events. Those variables can be bound by some quantified operator ASTDs. In order to keep track of the bound variables in a sub-state, when computing a transition, we need an execution environment that contains a valuation for each variable.

Definition A.5. *An environment is a function which maps a variable to a value. An environment is noted* $(\![x_1, \ldots, x_n := v_1, \ldots, v_n]\!)$, *or* $(\![\vec{x} := \vec{v}]\!)$. *An empty environment is noted* $(\![]\!)$. *An environment* Γ *can be used in a substitution. The symbol* $\lhd$ *is a composition operator on environments such that if* Γ_1, Γ_2 *are environments and u an expression,* $u[\Gamma_1 \lhd \Gamma_2] = (u[\Gamma_1])[\Gamma_2]$. *Note that* Γ_1 *has precedence over* Γ_2 *when* $\Gamma_1 \lhd \Gamma_2$ *is used in a substitution.*

For example, if we consider the ASTD *processC* from Figure A.2, and when looking locally at the sub-state $(\star_\circ, \text{true}, (\mathsf{aut}_\circ, Q3, (\mathsf{elem}_\circ)))$ in $processB(1)$, we need the environment $(\![v := 1]\!)$ to compute the transition $c(1)$, because the valuation does not appear in the expression of the sub-state.

Remark that the notion of environment does not affect the definitions of the function *init* and predicate *final*. However, it is important in the definition of the

transition relation of ASTDs. The operational semantics of ASTDs consists of a set of inference rules defined inductively. We write transitions with respect to an environment Γ, noted as $s \xrightarrow{\sigma,\Gamma}_a s'$, where s and s' are two states of an ASTD a and σ an event. Subscript a can be omitted when it is clear from the context which ASTD is being referred to.

Definition A.6. *Let a be an ASTD without free variables and s and s' two states of a. We compute a transition starting from an empty environment, using the following inference rule.*

$$\textsf{env}\ \frac{s \xrightarrow{\sigma,[]} s'}{s \xrightarrow{\sigma} s'}$$

The rule env *allows to introduce the environment in transitions, that is necessary to use the other inference rules described below. Let a be an ASTD, Γ an environment and σ an event. The operational semantics is inductively defined for each ASTD subtype.*

1. *If $a = (\textsf{aut}, n, \Sigma, N, \nu, \delta, SF, DF, n_0)$, each transition in a is given by one of the two following inference rules:*

$$\textsf{aut}_1\ \frac{\delta(n_1, n_2, \sigma', final?) \qquad final? \Rightarrow final(s) \wedge \sigma'[\Gamma] = \sigma}{(\textsf{aut}_\circ, n_1, s) \xrightarrow{\sigma,\Gamma} (\textsf{aut}_\circ, n_2, init(\nu(n_2)))}$$

$$\textsf{aut}_2\ \frac{s \xrightarrow{\sigma,\Gamma}_{\nu(n)} s'}{(\textsf{aut}_\circ, n, s) \xrightarrow{\sigma,\Gamma} (\textsf{aut}_\circ, n, s')}$$

   Rule $\textsf{aut}_1$ *describes a transition between local states. There is a transition, labeled by σ and with environment Γ, between some state of n_1 and the initial state of n_2 if there is an arrow in the automaton between n_1 and n_2, labeled by σ' and such that the environment applied as a substitution on σ' gives σ, and if the transition marked as final, then the source state must be a final state. Rule* $\textsf{aut}_2$ *handles transition within a sub-state.*

2. *If $a = (\star, n, b)$, the inference rules are:*

$$\star_1 \frac{(\mathit{final}(s) \vee \neg \mathit{started}?) \qquad \mathit{init}(b) \xrightarrow{\sigma,\Gamma}_b s'}{(\star_\circ, \mathit{started}?, s) \xrightarrow{\sigma,\Gamma} (\star_\circ, \text{true}, s')}$$

$$\star_2 \frac{s \xrightarrow{\sigma,\Gamma}_b s'}{(\star_\circ, \text{true}, s) \xrightarrow{\sigma,\Gamma} (\star_\circ, \text{true}, s')}$$

Rule $\star_1$ allows for (re-)starting from the initial state of the component ASTD when a final state has been reached or for the first iteration. Rule $\star_2$ allows for execution on the component ASTD when an iteration has been started.

3. *If $a = (|, n, l, r)$,*

$$|_1 \frac{\mathit{init}(l) \xrightarrow{\sigma,\Gamma}_l s'}{(|_\circ, \bot, \bot) \xrightarrow{\sigma,\Gamma} (|_\circ, \mathit{left}, s')} \quad |_2 \frac{\mathit{init}(r) \xrightarrow{\sigma,\Gamma}_r s'}{(|_\circ, \bot, \bot) \xrightarrow{\sigma,\Gamma} (|_\circ, \mathit{right}, s')}$$

$$|_3 \frac{s \xrightarrow{\sigma,\Gamma}_l s'}{(|_\circ, \mathit{left}, s) \xrightarrow{\sigma,\Gamma} (|_\circ, \mathit{left}, s')} \quad |_4 \frac{s \xrightarrow{\sigma,\Gamma}_r s'}{(|_\circ, \mathit{right}, s) \xrightarrow{\sigma,\Gamma} (|_\circ, \mathit{right}, s')}$$

Rules $|_1$ and $|_2$ deal with the execution of the first event from the initial state. Rules $|_3$ and $|_4$ deal with the execution of the subsequent events from the chosen component.

4. *If $a = (||, n, \Delta, l, r)$,*

$$||_1 \frac{\alpha(\sigma) \notin \Delta \qquad s_l \xrightarrow{\sigma,\Gamma}_l s'_l}{(||_\circ, s_l, s_r) \xrightarrow{\sigma,\Gamma} (||_\circ, s'_l, s_r)} \quad ||_2 \frac{\alpha(\sigma) \notin \Delta \qquad s_r \xrightarrow{\sigma,\Gamma}_r s'_r}{(||_\circ, s_l, s_r) \xrightarrow{\sigma,\Gamma} (||_\circ, s_l, s'_r)}$$

$$||_3 \frac{\alpha(\sigma) \in \Delta \qquad s_l \xrightarrow{\sigma,\Gamma}_l s'_l \qquad s_r \xrightarrow{\sigma,\Gamma}_r s'_r}{(||_\circ, s_l, s_r) \xrightarrow{\sigma,\Gamma} (||_\circ, s'_l, s'_r)}$$

Rules $||_1$ and $||_2$ respectively describe execution of events with no synchronization required on the left-hand side and the right hand side of the synchronization ASTD. Rule $||_3$ describes the synchronization between the left hand side and the right hand side.

5. *If* $a = (|:, n, x, T, b)$,

$$|:_1 \frac{init(b) \xrightarrow{\sigma,([x:=v])\triangleleft\Gamma}_b s' \qquad v \in T}{(|:_\circ, \bot, \bot) \xrightarrow{\sigma,\Gamma} (|:_\circ, v, s')} \quad |:_2 \frac{s \xrightarrow{\sigma,([x:=v])\triangleleft\Gamma}_b s' \qquad v \neq \bot}{(|:_\circ, v, s) \xrightarrow{\sigma,\Gamma} (|:_\circ, v, s')}$$

Rule $|:_1$ *describes the execution of the first event from the initial state, whereas Rule* $|:_2$ *deal with the execution of the subsequent events. In both cases, the value bound to the quantification variable is added to the execution environment and can be used to make the proof after the environment has been applied as a substitution.*

6. *If* $a = (|||:, n, x, T, b)$,

$$|||:_1 \frac{f(v) \xrightarrow{\sigma,([x:=v])\triangleleft\Gamma}_b s'}{(|||:_\circ, f) \xrightarrow{\sigma,\Gamma} (|||:_\circ, f \Leftarrow \{v \mapsto s'\})}$$

Rule $|||:_1$ *describes the execution of an event from the component ASTD instantiated by value* v*. The value* v *bound to the quantification variable* x *is added to the execution environment.*

For example, consider the ASTD *processC* from Figure A.2 and let $s = (|||:_\circ, \{1 \mapsto (\star_\circ, \text{true}, (\mathsf{aut}_\circ, Q4, (\mathsf{elem}_\circ))), 2 \mapsto (\star_\circ, \text{false}, \bot)\})$ be a state of *processC*. Let us prove that there is a transition labeled by $c(1)$ between the initial state of *processC* and s. We have $init(processC) = (|||:_\circ, \{1 \mapsto (\star_\circ, \text{false}, \bot), 2 \mapsto (\star_\circ, \text{false}, \bot)\})$. We can derive the transition as follows:

$$\mathsf{env} \frac{\star_1 \dfrac{\neg started? \qquad \mathsf{aut}_1 \dfrac{\delta(Q3, Q4, c(v), \text{true}) \qquad final((\mathsf{elem}_\circ)) \wedge c(v)[([v := 1])] = c(1)}{(\mathsf{aut}_\circ, Q3, (\mathsf{elem}_\circ)) \xrightarrow{c(1),([v:=1])} (\mathsf{aut}_\circ, Q4, (\mathsf{elem}_\circ))}}{|||:_1 \dfrac{(\star_\circ, \text{false}, \bot) \xrightarrow{c(1),([v:=1])} (\star_\circ, \text{true}, (\mathsf{aut}_\circ, Q4, (\mathsf{elem}_\circ)))}{init(processC) \xrightarrow{c(1),([])} s}}}{init(processC) \xrightarrow{c(1)} s}$$

To make the state expressions easier to read, we introduce a graphical representation for the ASTD states. An ASTD state can be represented by a tree where each

node is labeled by a tuple giving the type of the state. If a type of state includes a sub-state in its definition, then it is represented by a child node. Furthermore, we split up the nodes for quantifications so that the values appear in child nodes. A labeled tree is given by an expression thanks to the grammar $Tree ::= Node \mid Node\langle Tree, \ldots, Tree\rangle$.

Definition A.7. *The tree representation of an ASTD state is given by the tree function, which is defined inductively as follows:*

1. $tree((\mathit{aut}_\circ, n, (\mathit{elem}_\circ))) = (\mathit{aut}_\circ, n)$
2. $tree((\mathit{aut}_\circ, n, s)) = (\mathit{aut}_\circ, n)\langle tree(s)\rangle \qquad \text{if } s \neq (\mathit{elem}_\circ)$
3. $tree((\star_\circ, started?, s)) = \begin{cases} \star_\circ & \text{if } started? = \text{false} \\ \star_\circ\langle tree(s)\rangle & \text{else} \end{cases}$
4. $tree((|_\circ, side, x)) = \begin{cases} |_\circ & \text{if } side = \bot \\ (|_\circ, side)\langle tree(x)\rangle & \text{else} \end{cases}$
5. $tree((||_\circ, s_l, s_r)) = ||_\circ\langle tree(s_l), tree(s_r)\rangle$
6. $tree((|:_\circ, v, x)) = \begin{cases} |:_\circ & \text{if } v = \bot \\ |:_\circ\langle v\langle tree(x)\rangle\rangle & \text{else} \end{cases}$
7. $tree((|||:_\circ, f)) = |||:_\circ\langle v_1\langle tree(f(v_1))\rangle, \ldots, v_k\langle tree(f(v_k))\rangle\rangle$ *where* $\bigcup_i\{v_i\} = dom(f)$

The set of labels consists of a finite set of tuples for each type of ASTD state and many (possibly infinite) sets of values (used in the case of quantified operators).

For instance, the state

$$s = (|||:_\circ, \{1 \mapsto (\star_\circ, \text{true}, (\mathsf{aut}_\circ, Q4, (\mathsf{elem}_\circ))), 2 \mapsto (\star_\circ, \text{false}, \bot)\})$$

of *processC* from Figure A.2 is represented by a tree depicted in Figure A.3.

Remark that the tree branches are ordered in that representation, even for the quantified interleaving operator. We simply assume that the ordering of the branches of $dom(f)$ is arbitrary. Except from that, the function *tree* is a simple rewriting and it is obvious that for all ASTD state s, the tree representation $tree(s)$ is semantically

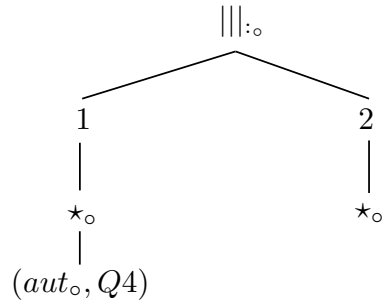

Figure A.3 – Example of tree representation of a state

equivalent to s. In the following, we may refer indifferently to s or $tree(s)$ as the state expression s.

A.3 Parameterized ASTD

Some of the most interesting features of the ASTDs are the quantified operators (interleaving and choice) that allow to model replication of processes and complex interactions between them. Consider a simple library system handling members, books and loans. If we want to specify an ASTD for this system, we can use the Quantified Interleaving to model many replicated processes running concurrently, each process representing a specific member for example. In the original definition of quantified operator ASTDs, we have to specify a constant set of quantification. This means that the number of members in the library, for example, is fixed. To model a library that works with any number of members, we need to consider a more abstract specification that takes the set of members as a parameter. The system can intuitively be parameterized by a number of members and of books. Therefore, we allow quantification sets to be variables in Quantified Choice and Quantified Interleaving. We call those new variables the parameters of the system. To this end, we introduce the new definition of Parameterized ASTD.

Definition A.8 (PASTD)**.** *A Parameterized ASTD (PASTD) $a[T_1, \ldots, T_k]$ is a model of an ASTD equipped with some parameters $T_1, \ldots, T_k$. Each parameter T_i has a type P_i, called a parameter domain. A parameter can only be instantiated with a finite subset of its parameter domain. Parameter domains are disjoint. Moreover, no subset*

of a parameter domain can be used as a constant in the specification. We denote by $a[T_1 := V_1, \ldots, T_k := V_k]$ *the ASTD corresponding to a PASTD instantiated by values* $V_1 \subset P_1, \ldots, V_k \subset P_k$*. Parameters are transferred to sub-PASTDs and are used as sets of quantification for quantified choices or quantified interleavings.*

For instance, a PASTD model of a library system is given by the ASTD of Figure A.1, if we consider the parameters T_m and T_b. The system manage loans of books by members. The two parameters T_m and T_b have domains $P_m = \{m_1, m_2 \ldots\}$ and $P_b = \{b_1, b_2 \ldots\}$, respectively, representing the sets of member and book identifiers. A book is acquired by the library. It can be discarded if it is not lent. A member must join the library in order to borrow a book. She can leave the library membership when all her loans are returned.

In some parameterized systems, like Petri Nets, parameters are natural numbers and denote the number of replicated processes. However, this abstraction cannot be made in our case. This is due to the interactions between the various instances in some complex parameterized systems. Indeed, consider the previous example of the library system. With natural numbers as parameters, we could specify the number of registered members or acquired books in the library but not the fact that a specific member m_1 has borrowed the book b_2. That is why, in PASTDs, we use a finite set of identifiers instead of a natural number to instantiate a parameter. This said, we will see later how to take into account the symmetries induced by this modeling.

As we introduce free variables in the specification of PASTDs, we need a valuation of those variables for the model to represent a concrete system. Consider an "additional environment" consisting of a vector of parameters values. Besides, parameters of PASTDs are noted between brackets "[...]" and parameters of events and calls between parenthesis "(...)".

Definition A.9 (IPASTD)**.** *If* $a[\overrightarrow{T}]$ *is a PASTD, we call* $a[\overrightarrow{T} := \overrightarrow{V}]$ *an Instantiated PASTD (IPASTD). PASTDs with parameters* $\overrightarrow{T} = (T_1, \ldots, T_k)$ *are instantiated with values* $\overrightarrow{V} = (V_1, \ldots, V_k)$ *as follows:*

1. $(\mathit{elem})[\overrightarrow{T} := \overrightarrow{V}] = (\mathit{elem})$
2. $(\mathit{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T} := \overrightarrow{V}] =$
   $(\mathit{aut}, name, \Sigma, N, \nu', \delta, SF, DF, n_0)$*, where* $\nu'(n) = \nu(n)[\overrightarrow{T} := \overrightarrow{V}]$

3. $(\star, n, b)[\overrightarrow{T} := \overrightarrow{V}] = (\star, n, b[\overrightarrow{T} := \overrightarrow{V}])$
4. $(|, n, l, r)[\overrightarrow{T} := \overrightarrow{V}] = (|, n, l[\overrightarrow{T} := \overrightarrow{V}], r[\overrightarrow{T} := \overrightarrow{V}])$
5. $(||, n, \Delta, l, r)[\overrightarrow{T} := \overrightarrow{V}] = (||, n, \Delta, l[\overrightarrow{T} := \overrightarrow{V}], r[\overrightarrow{T} := \overrightarrow{V}])$
6. $(|:, n, x, T_i, b)[\overrightarrow{T} := \overrightarrow{V}] = (|:, n, x, V_i, b[\overrightarrow{T} := \overrightarrow{V}])$, *where* $T_i \in \overrightarrow{T}$ *is a parameter and* $V_i \in \overrightarrow{V}$ *a value*
7. $(|:, n, x, W, b)[\overrightarrow{T} := \overrightarrow{V}] = (|:, n, x, W, b[\overrightarrow{T} := \overrightarrow{V}])$, *if* W *is not a parameter*
8. $(|||:, n, x, T_i, b)[\overrightarrow{T} := \overrightarrow{V}] = (|||:, n, x, V_i, b[\overrightarrow{T} := \overrightarrow{V}])$, *where* $T_i \in \overrightarrow{T}$ *is a parameter and* $V_i \in \overrightarrow{V}$ *a value*
9. $(|||:, n, x, W, b)[\overrightarrow{T} := \overrightarrow{V}] = (|||:, n, x, W, b[\overrightarrow{T} := \overrightarrow{V}])$, *if* W *is not a parameter*

If there is no ambiguity, the IPASTD is denoted by $a[\overrightarrow{V}]$ *instead of* $a[\overrightarrow{T} := \overrightarrow{V}]$.

For instance, in the library system of Figure A.1, let us instantiate T_m and T_b by $V_m = \{m_1, m_2\}$ and $V_b = \{b_1, b_2, b_3\}$ respectively. We have that $library[V_m, V_b]$ is an IPASTD and an example of a state expression is depicted on Figure A.4. The state consists of two members m_1 and m_2 and three books b_1, b_2 and b_3 where the book b_2 is borrowed by the member m_1. Remark that parameter values should not be empty if we want to be able to instantiate some states.

If the model contains no free variables (in Automaton ASTDs), then an IPASTD $a[\overrightarrow{V}]$ can be seen as a TS. Each state of $a[\overrightarrow{V}]$ is given by an ASTD state expression s and the vector of parameter values $\overrightarrow{V} = (V_1, \ldots, V_k)$, such that s is a well-formed state expression consistent with the instantiated ASTD $a[V_1, \ldots, V_k]$. To be more precise, a state of the TS must also contain the ASTD that the expression s refers to, *i.e.* the PASTD $a[\overrightarrow{T}]$ and the valuation $\overrightarrow{V}$. In [41], states are defined without mentioning any corresponding ASTD as they are implicit. In our case, this is necessary to compare states from different IPASTDs.

Definition A.10. *In order to handle PASTDs, we make the following modifications to some previous definitions.*

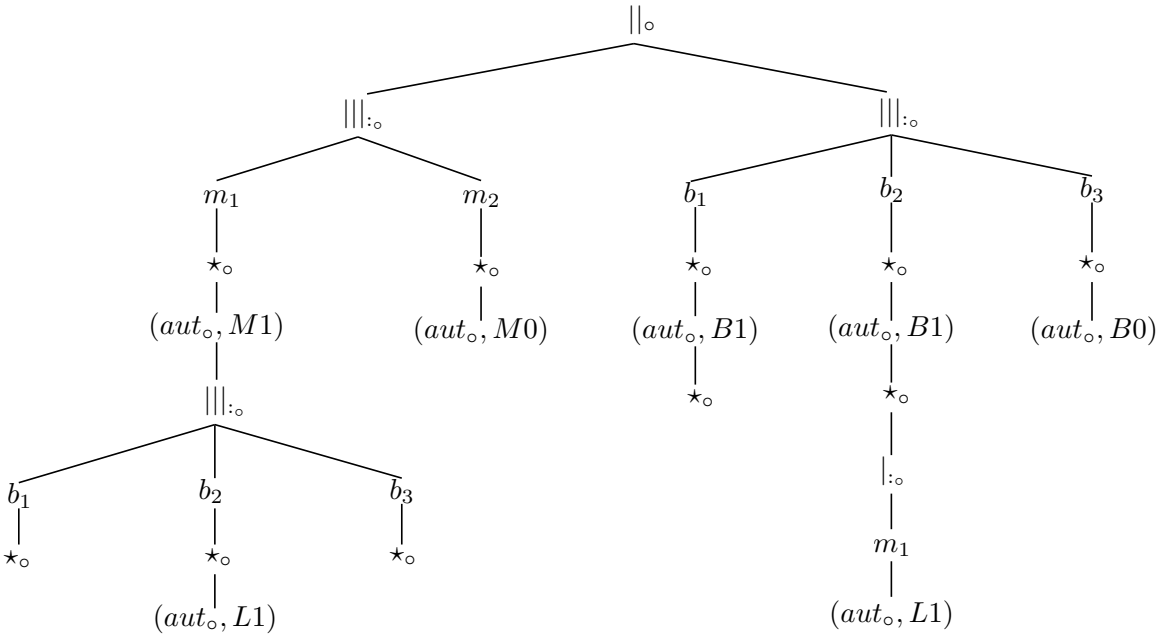

Figure A.4 – A state of the library system

- *In Definition A.2: each tuple describing a state is augmented by a PASTD and a valuation. However, for the sake of concision and to keep the same notation, we will denote a tuple representing a state as usual. We denote by s.pa the PASTD associated to a state s and s.val the parameter values.*
- *In Definition A.3: if* $a[\overrightarrow{V}]$ *is an IPASTD, then we define* $init(a[\overrightarrow{V}]).pa = a[\overrightarrow{T}]$ *and* $init(a[\overrightarrow{V}]).val = \overrightarrow{V}$.
- *In Definition A.4: the adaptation for the predicate final is straightforward.*
- *In Definition A.6: the transition relation preserves the PASTD and the valuation. If* $s \rightarrow s'$, *then* $s.pa = s'.pa$ *and* $s.val = s'.val$.

For example, a state s of an Automaton ASTD $a[\overrightarrow{V}]$ is written as an extended tuple $(\mathsf{aut}_\circ, n, ss, a, \overrightarrow{V})$, or by a simple tuple $(\mathsf{aut}_\circ, n, ss)$ assuming $s.pa = a$ and $s.val = \overrightarrow{V}$.

Note that if a model contains free variables in its Automaton ASTD components, we still need the environment to define a state of a concrete system.

A.3. Parameterized ASTD

Definition A.11. *We denote by $\mathcal{Q}$ the set of IPASTD states s such that s is a well-formed expression with regards to its PASTD $s.pa$ and the valuation $s.val$. In addition to PASTD parameters, $s.pa$ may contain free variables in Automaton ASTDs. Formally, $s \in \mathcal{Q}$ iff*

1. $s = (\mathit{elem}_\circ)$ *and* $s.pa = (\mathit{elem})$;
2. $s = (\mathit{aut}_\circ, n, ss)$ *and* $s.pa = (\mathit{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)$ *with* $n \in N$ *and* $ss.pa = \nu(n)$ *and* $ss.val = s.val$ *and* $ss \in \mathcal{Q}$;
3. $s = (\star_\circ, started?, x)$ *and* $s.pa = (\star, n, b)$ *and*
   - $x \neq \bot$ *and* $x.pa = b$ *and* $x.val = s.val$ *and* $x \in \mathcal{Q}$, *or*
   - $x = \bot$ *and* $started? = \text{false}$;
4. $s = (|_\circ, side, x)$ *and* $s.pa = (|, n, l, r)$ *and*
   - $side = \mathit{left}$ *and* $x.pa = l$ *and* $x.val = s.val$ *and* $x \in \mathcal{Q}$, *or*
   - $side = \mathit{right}$ *and* $x.pa = r$ *and* $x.val = s.val$ *and* $x \in \mathcal{Q}$, *or*
   - $side = \bot$ *and* $x = \bot$;
5. $s = (||_\circ, s_l, s_r)$ *and* $s.pa = (||, n, \Delta, l, r)$ *and*
   - $s_l.pa = l$ *and* $s_l.val = s.val$ *and* $s_l \in \mathcal{Q}$, *and*
   - $s_r.pa = r$ *and* $s_r.val = s.val$ *and* $s_r \in \mathcal{Q}$;
6. $s = (|:_\circ, v, x)$ *and*
   - $s.pa = (|:, n, y, T_i, b)$ *and* $v \in V_i$ *and* $s.val = \overrightarrow{V}$ *and* $x.pa = b$ *and* $x.val = \overrightarrow{V}$ *and* $x \in \mathcal{Q}$, *where* $T_i \in \overrightarrow{T}$ *is a parameter and* $V_i \in \overrightarrow{V}$ *a value, or*
   - $s.pa = (|:, n, y, W, b)$ *and* $v \in W$ *and* $x.pa = b$ *and* $x.val = s.val$ *and* $x \in \mathcal{Q}$, *where* W *is not a parameter, or*
   - $s.pa = (|:, n, y, T, b)$ *and* $v = \bot$ *and* $x = \bot$;
7. $s = (|||:_\circ, f)$ *and*

- $s.pa = (|||:, n, y, T_i, b)$ *and* $dom(f) = V_i$ *and* $s.val = \overrightarrow{V}$ *and for all* $ss \in ran(f)$ *we have* $ss.pa = b$, $ss.val = \overrightarrow{V}$ *and* $ss \in \mathcal{Q}$, *where* $T_i \in \overrightarrow{T}$ *is a parameter and* $V_i \in \overrightarrow{V}$ *a value, or*
- $s.pa = (|||:, n, y, W, b)$ *and* $dom(f) = W$ *and for all* $ss \in ran(f)$ *we have* $ss.pa = b$, $ss.val = s.val$ *and* $ss \in \mathcal{Q}$, *where* W *is not a parameter;*

We are now ready to define the transition system corresponding to an IPASTD. Indeed, as we have seen previously, a state expression of an ASTD is slightly different from a state of a transition system. In the following definition we give a formal definition of IPASTDs from a TS viewpoint.

Definition A.12. *Let* $a[\overrightarrow{T}]$ *be a PASTD with parameter domains* $\overrightarrow{P}$ *and* $\overrightarrow{V} \subset \overrightarrow{P}$ *a vector of values. If the IPASTD* $a[\overrightarrow{V}]$ *contains no free variables, then we define the corresponding TS* $\mathcal{S}_{a,\overrightarrow{V}} = (Q, \rightarrow)$ *with initial states* $I \subseteq Q$ *as follows:*

- Q *consists of the IPASTD states* $s \in \mathcal{Q}$ *such that* $s.pa = a$ *and* $s.val = \overrightarrow{V}$;
- $\rightarrow$ *is the set of all transitions* $s \xrightarrow{\sigma} s'$ *that can be derived in* $a[\overrightarrow{V}]$, *where* s *and* s' *are in* Q;
- $I \subseteq Q$, *the set of initial states, is the singleton* $\{init(a[\overrightarrow{V}])\}$.

Remark that the function *init* always returns a state of $\mathcal{Q}$ and, for all state $s \in \mathcal{Q}$, if $s \rightarrow s'$ then $s' \in \mathcal{Q}$. Thus, every path starting from the initial state possible in the ASTD is modeled in the associated TS.

Lemma A.1. *Let* $a[\overrightarrow{T}]$ *be a PASTD. For any parameter value* $\overrightarrow{V}$, $init(a[\overrightarrow{V}]) \in \mathcal{Q}$.

Lemma A.2. *Let* $a[\overrightarrow{T}]$ *be a PASTD and* $\overrightarrow{V}$ *a parameter value. Let* s_1, s_2 *two states such that* $s_1.pa = s_2.pa = a$ *and* $s_1.val = s_2.val = \overrightarrow{V}$. *If* $s_1 \in \mathcal{Q}$ *and* $s_1 \xrightarrow{\sigma,\Gamma} s_2$, *then* $s_2 \in \mathcal{Q}$.

Let us now transform PASTDs into TSs. Intuitively, a parameterized system represents a family of systems, that is a set of systems. In our case, we consider that a parameter of a PASTD can be instantiated by any non-empty finite subset of a (possibly infinite) adequate set of identifiers. We define the global system that includes all possible instantiated systems as follows.

Definition A.13. *Let $a[\overrightarrow{T}]$ be a PASTD with parameter domains $\overrightarrow{P}$. If $a[\overrightarrow{T}]$ contains no free variables other than $\overrightarrow{T}$, then we define the corresponding TS $\mathcal{S}_a$ by the union of all possible instantiations of system:*

$$\mathcal{S}_a = \bigcup_{\overrightarrow{V}} \mathcal{S}_{a,\overrightarrow{V}}$$

where $\overrightarrow{V} = (V_1, \ldots, V_k)$, $\overrightarrow{P} = (P_1, \ldots, P_k)$ and V_i takes every values in the set of non-empty finite subset of P_i. The union of systems is equivalent to a choice between systems. Formally, $\mathcal{S}_a = (Q, \rightarrow)$ with initial states $I \subseteq Q$ is such that:

- *Q is the union of the sets of states of each TS;*
- *$\rightarrow$ is the union of the sets of transitions of each TS;*
- *I is the union of the sets of initial states of each TS.*

Remark that $\mathcal{S}_a$ may be an infinite state system, if it is formed by an infinite union of systems. Indeed, if a parameter domain P_i is infinite, then the union over V_i is infinite.
In practice, such global TS that represents parameterized systems are infinite systems. In the example of the library system, the corresponding TS is a infinite system consisting of the union of all possible finite systems instantiated with a natural number of members and of books.

A.4 PASTDs are MTSs

A.4.1 Defining a quasi-ordering

The theory of Well-Structured Transition Systems is used to check coverability properties in infinite systems like Petri Nets and should be useful to verify other parameterized systems like PASTDs. To apply this method to PASTDs, we need to define a qo on its set of states satisfying the monotony condition. Besides, the qo has to be chosen according to the coverability property we want to verify, that is an upward-closed set of states.

A.4. PASTDs are MTSs

Let us consider a PASTD $a[\overrightarrow{T}]$. Intuitively, for a state s of an IPASTD $a[\overrightarrow{V}]$, there is a larger state s' of $a[\overrightarrow{V'}]$ with $\overrightarrow{V} \subseteq \overrightarrow{V'}$, where $\subseteq$ is the point-wise extension of the inclusion relation to vectors, and such that s' can simulate in $a[\overrightarrow{V'}]$ the behavior of s in $a[\overrightarrow{V}]$, *i.e.* every possible execution trace from s is also possible from s'. If the elements of $\overrightarrow{V}$ occur in quantified interleavings, an example of a state s' which is larger than a state s is a state which differs from s only for values in $V_i' \setminus V_i$.

Definition A.14. *We define the relation $\sqsubseteq$ on the set $\mathcal{Q}$ of IPASTD states such that $s \sqsubseteq s'$ iff $s.pa = s'.pa$ and $s.val \subseteq s'.val$ and*

1. $s = (\mathit{elem}_\circ)$ *and* $s' = (\mathit{elem}_\circ)$
2. $s = (\mathit{aut}_\circ, n, ss)$ *and* $s' = (\mathit{aut}_\circ, n, ss')$ *and* $ss \sqsubseteq ss'$
3. $s = (\star_\circ, \mathit{started?}, x)$ *and* $s' = (\star_\circ, \mathit{started?}, x')$ *and* $(x = x' = \bot$ *or* $x \sqsubseteq x')$
4. $s = (|_\circ, \mathit{side}, x)$ *and* $s' = (|_\circ, \mathit{side}, x')$ *and* $(x = x' = \bot$ *or* $x \sqsubseteq x')$
5. $s = (||_\circ, s_l, s_r)$ *and* $s' = (||_\circ, s_l', s_r')$ *and* $s_l \sqsubseteq s_l'$ *and* $s_r \sqsubseteq s_r'$
6. $s = (|:_\circ, v, x)$ *and* $s' = (|:_\circ, v, x')$ *and* $(x = x' = \bot$ *or* $x \sqsubseteq x')$
7. $s = (|||:_\circ, f)$ *and* $s' = (|||:_\circ, f')$ *and* $dom(f) \subseteq dom(f')$ *and* $\forall v \in dom(f) \cdot f(v) \sqsubseteq f'(v)$

Note that for the last case, $dom(f) \subseteq dom(f')$ is only possible if their respective domain comes from the instantiation of a parameter. Indeed, we require that $s.pa = s'.pa$. Thus, s and s' are incomparable if they do not have the same domain and the quantification set is not a parameter.

Proposition A.1. *The relation $\sqsubseteq$ on $\mathcal{Q}$ is a partial ordering.*

Proof. By Definition A.14, $\sqsubseteq$ is clearly reflexive, transitive and antisymmetric, as $\subseteq$ is so. □

Unfortunately, monotony does not hold for this partial ordering considering the current transition relation in PASTDs. Quantified Interleaving operators do not preserve monotony as every sub-state of quantified interleaving state must synchronize on the final state. Figure A.5 shows an example of PASTD which does not

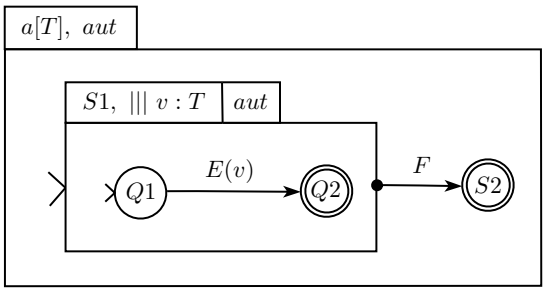

Figure A.5 – Example of PASTD

satisfy the monotony. Suppose that the domain of the parameter T is $\mathbb{N}$ and let $V = \{1, 2\} \subseteq V' = \{1, 2, 3\} \subseteq \mathbb{N}$. Let s be a state of $a[V]$ such that both instances of the automaton in $S1$ are in the local state $Q2$. Then, as every instance of the Quantified Interleaving are in final state, the transition F can fire. Consider now a state s' whose ASTD is $a[V']$ and such that the instances 1 and 2 are in $Q2$ but the instance 3 is in $Q1$. According to Definition A.14, we have $s \sqsubseteq s'$, as we just add a new instance in an arbitrary local state. However, the transition F cannot be executed from the state s', as the instance 3 is not in a final configuration.

In order to tackle this problem, we change the definition of the predicate *final* such that the following property holds: for a state $s = (|||:_\circ, f)$, if s is final then for each state s' such that $s \sqsubseteq s'$, s' is final too. This allows the transition F, from the previous example, to be able to fire at any moment, thus to preserve the monotony.

Definition A.15. *Let $s \in \mathcal{Q}$ be an IPASTD state. The predicate final(s), which determines if s is final in s.pa, is identical to the one in Definition A.4 (adapted by Definition A.12), except that:*

9. *If $s = (|||:_\circ, f)$ and $s.pa = (|||:, n, x, X, b)[\overrightarrow{T}]$, then*
   - *if X is a parameter, final$(s) \equiv \exists v \in X \cdot final(f(v))$,*
   - *else, final$(s) \equiv \forall v \in X \cdot final(f(v))$.*

Definition A.15 is one of the several modifications that can be done to obtain monotony. It means that if the interleaving operator is used with a parameter, then we change the semantics of the quantified interleaving such that we only need one instance to

be final for the whole interleaving to be final too. As this can be sometimes inconvenient, we keep the old semantics when X is not a parameter. We could have used a simpler definition where $\mathit{final}(s) \equiv \text{false}$ or where $\mathit{final}(s) \equiv \text{true}$ for example. In any way, that modification allows us to prove Lemma A.4, which is necessary to show Lemma A.5; they are introduced in the sequel.

Lemma A.3. *Let $a[\overrightarrow{T}]$ be a PASTD. For any parameter values $\overrightarrow{V}$ and $\overrightarrow{V'}$ such that $\overrightarrow{V} \subseteq \overrightarrow{V'}$, $\mathit{init}(a[\overrightarrow{V}]) \sqsubseteq \mathit{init}(a[\overrightarrow{V'}])$.*

Lemma A.4. *For any IPASTD states $s, s' \in \mathcal{Q}$ such that $s \sqsubseteq s'$, if $\mathit{final}(s)$ then $\mathit{final}(s')$.*

Lemma A.5. *For any IPASTD states $s_1, s'_1, s_2 \in \mathcal{Q}$, any event σ and any environment Γ, if $s_1 \sqsubseteq s'_1$ and $s_1 \xrightarrow{\sigma,\Gamma} s_2$, then there exists s'_2 such that $s_2 \sqsubseteq s'_2$ and $s'_1 \xrightarrow{\sigma,\Gamma} s'_2$.*

Proofs of Lemmas A.3, A.4 and A.5 are done inductively on the structure of PASTDs. Lemmas A.3 and A.4 give some properties on the function *init* and the predicate *final*. Lemma A.5 states a monotony theorem considering an event σ and an environment Γ. The monotony for $\sqsubseteq$ is a simple corollary of this lemma. Proofs are provided in Appendix B.2.

Theorem A.1. *Let $a[\overrightarrow{T}]$ be a PASTD without free variables in events, with $\mathcal{S}_a$ the corresponding TS. $\mathcal{S}_a$ is monotone wrt $\sqsubseteq$.*

Proof. Let s_1, s'_1, s_2 states of $\mathcal{S}_a$ such that $s_1 \sqsubseteq s'_1$ and $s_1 \to s_2$. There is an event σ such that $s_1 \xrightarrow{\sigma} s_2$. By the only inference rule env, we have $s_1 \xrightarrow{\sigma,(\!|\,|\!)} s_2$. Then, by Lemma A.5 there exists $s'_2 \in \mathcal{Q}$ such that $s_2 \sqsubseteq s'_2$ and $s'_1 \xrightarrow{\sigma,(\!|\,|\!)} s'_2$. Thus, by the rule env, we have $s'_1 \xrightarrow{\sigma} s'_2$. As s'_2 is a state of $\mathcal{S}_a$ too, we can conclude. □

Nevertheless, Definition A.15 could be problematic in some cases. For example, recall the example of the library system of Figure A.1, where a member must complete, *i.e.* return, all her loans in order to leave the library system. If we consider the previous definition, she would be able to leave whenever she has returned one of her loans without completing the other loan processes. This scenario could be an issue of that new definition, but we propose a solution to solve the problem in that case. Suppose

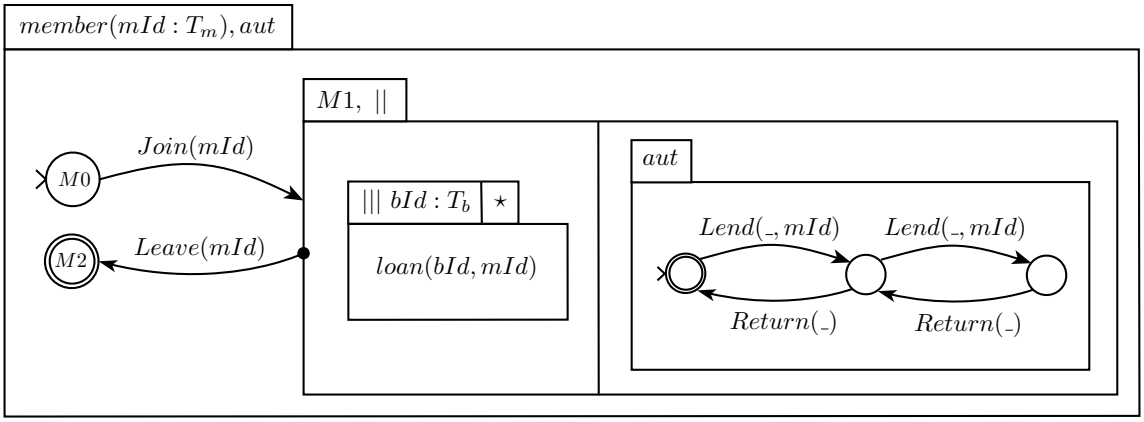

Figure A.6 – Modification of the member process

that the number of loans is limited to two books per member. Then, we can modify the ASTD of the member process to obtain the one of Figure A.6, where symbol "_" in events means that the symbol can be replaced by any value. Indeed, we add a new Automaton ASTD in synchronization with the Quantified Interleaving in order to be able to specify a final state for the ASTD $M1$. Intuitively, this automaton counts the number of loans and allows to terminate only if this number is equal to zero. In our case, the member will not be allowed to leave if she has not returned all her loans. This principle can be generalized to every ASTD specification where we can limit the number of processes in the Quantified Interleaving, because the counting automaton must be finite state.

A.4.2 Symmetries

The notion of *symmetry* is an important topic in the context of model checking. In [36], the authors show several way to exploit isomorphisms in systems in order to simplify the verification process. We use here a similar approach in PASTDs and we will see that the detection of symmetries is essential in our case.

In PASTDs, we are interested in the symmetries appearing in parameter domains. Indeed, the name of an instance has no importance in the system and can always be renamed by another element of the parameter domain. We consider that two states are equivalent if they are syntactically equivalent under some specific rename function

that renames all occurrences of a parameter element by another one. Let us define formally those rename functions.

Definition A.16 (Rename Function)**.** *Let $P_1, P_2, \ldots, P_k$ a set of parameter domains. A rename function ξ for the set of IPASTD states $\mathcal{Q}$ is defined by a set of permutations $\rho_1, \rho_2, \ldots, \rho_k$ on $P_1, P_2, \ldots, P_k$ respectively such that:*

1. $\xi((\mathsf{elem}_\circ)) = (\mathsf{elem}_\circ)$
2. $\xi((\mathsf{aut}_\circ, n, s)) = (\mathsf{aut}_\circ, n, \xi(s))$
3. $\xi((\star_\circ, started?, s)) = \begin{cases} (\star_\circ, \text{false}, \bot) & \text{if } started? = \text{false} \\ (\star_\circ, \text{true}, \xi(s)) & \text{else} \end{cases}$
4. $\xi((|_\circ, side, x)) = \begin{cases} (|_\circ, \bot, \bot) & \text{if } side = \bot \\ (|_\circ, side, \xi(x)) & \text{else} \end{cases}$
5. $\xi((||_\circ, s_l, s_r)) = (||_\circ, \xi(s_l), \xi(s_r))$
6. $\xi((|:_\circ, v, x)) = \begin{cases} (|:_\circ, \bot, \bot) & \text{if } v = \bot \\ (|:_\circ, \rho_i(v), \xi(x)) & \text{if } \exists i \cdot v \in P_i \\ (|:_\circ, v, \xi(x)) & \text{else} \end{cases}$
7. $\xi((|||:_\circ, f)) = \begin{cases} (|||:_\circ, f') & \text{if } \exists i \cdot dom(f) \subseteq P_i \\ (|||:_\circ, f'') & \text{else} \end{cases}$
   where $dom(f') = \rho_i(dom(f))$ and for all $v \in dom(f)$ we have $f'(\rho_i(v)) = \xi(f(v))$, and $dom(f'') = dom(f)$ and for all $v \in dom(f)$ we have $f''(v) = \xi(f(v))$

and $\xi(s).pa = s.pa$ and $\xi(s).val = (\rho_1(V_1), \ldots, \rho_k(V_k))$, where $s.val = (V_1, \ldots, V_k)$.

Now we can define a quasi-ordering on $\mathcal{Q}$ regarding the equivalence relation induced by the rename functions.

Definition A.17. *We define the relation $\preceq$ on $\mathcal{Q}$ by:*

$$s \preceq s' \iff \exists \xi \text{ a rename function s.t. } s \sqsubseteq \xi(s')$$

A.4. PASTDs are MTSs

Proposition A.2. *The relation $\preceq$ on $\mathcal{Q}$ is a quasi-ordering.*

Proof. The relation $\preceq$ is reflexive because the identity is a rename function and $\sqsubseteq$ is reflexive. The relation $\preceq$ is transitive because the composition of rename functions is a rename function and $\sqsubseteq$ is transitive. □

The notion of renaming is useful to factorize the state space of a PASTD. In fact, it is necessary in order to satisfy the finite pred-basis property needed in WSTSs. Consider the PASTD of Figure A.5. Let $s = (\mathsf{aut}_\circ, S2, (\mathsf{elem}_\circ))$ be the state where we are at the local state $S2$. A predecessor of s is a state $s' = (\mathsf{aut}_\circ, S1, (|||:_\circ, \{1 \mapsto (\mathsf{aut}_\circ, Q2, (\mathsf{elem}_\circ))\}))$ where the instance 1 of the Quantified Interleaving in the local state $S1$. But $s'' = (\mathsf{aut}_\circ, S1, (|||:_\circ, \{2 \mapsto (\mathsf{aut}_\circ, Q2, (\mathsf{elem}_\circ))\}))$ is also a predecessor of s. However, s' and s'' are incomparable and have no lower bound. Thus the pred-basis of s includes s' and s''. As the parameter domain may be infinite, we can conclude that s has no finite pred-basis if we consider the qo $\sqsubseteq$. This is not the case with $\preceq$, because $s' \preceq s''$ (take a rename function defined by a permutation ρ on $\mathbb{N}$ such that $\rho(1) = 2$ and $\rho(2) = 1$).

The following three lemmas are very similar to those from the previous section and allow to do the proof of monotony. Their proofs have the same structure and are detailed in Appendix B.3.

Lemma A.6. *Let $a[\overrightarrow{T}]$ be a PASTD. For any parameter value $\overrightarrow{V}$ and any rename function ξ defined by permutations $\rho_1, \ldots, \rho_k$, we have*

$$\xi(init(a[\overrightarrow{V}])) = init(a[\rho_1(V_1), \ldots, \rho_k(V_k)])$$

Lemma A.7. *For any $s \in \mathcal{Q}$ and for any rename function ξ, if final(s), then final$(\xi(s))$.*

Lemma A.8. *Let $P_1, \ldots, P_k$ a set of parameter domains. For any states $s_1, s_2 \in \mathcal{Q}$, any rename function ξ defined by permutations $\rho_1, \ldots, \rho_k$ on $P_1, \ldots, P_k$ respectively, any event σ and any environment $\Gamma = (\!|x_1, \ldots, x_n := v_1, \ldots, v_n|\!)$, if $s_1 \xrightarrow{\sigma,\Gamma} s_2$, then $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$, where $\sigma' = \sigma[v_1, \ldots, v_n := v'_1, \ldots, v'_n]$ and $\Gamma' = (\!|x_1, \ldots, x_n := v'_1, \ldots, v'_n|\!)$ with $v'_i = \rho_j(v_i)$ if $\exists j \cdot v_i \in P_j$ or $v'_i = v_i$ else.*

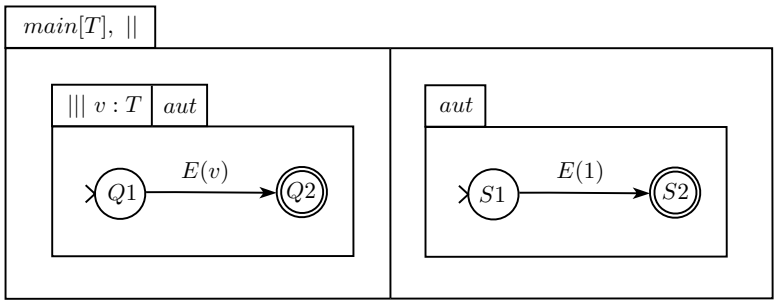

Figure A.7 – Example of invalid PASTD

We can now state the final monotony theorem for the qo $\preceq$. Its proof results from Lemma A.8 and the monotony for $\sqsubseteq$.

Theorem A.2. *Let $a[\overrightarrow{T}]$ be a PASTD without free variables in events, with $\mathcal{S}_a$ the corresponding TS. $\mathcal{S}_a$ is monotone wrt $\preceq$.*

Proof. Let s_1, s_2, s'_1 three states of $\mathcal{S}_a$ such that $s_1 \preceq s'_1$ and $s_1 \to s_2$. There is an event σ such that $s_1 \xrightarrow{\sigma} s_2$ and by the rule env we have $s_1 \xrightarrow{\sigma,(\!|\,|\!)} s_2$. By Definition A.17, there exists a rename function ξ such that $s_1 \sqsubseteq s''_1$ with $s''_1 = \xi(s'_1)$. And by Lemma A.5, let s''_2 such that $s_2 \sqsubseteq s''_2$ and $s''_1 \xrightarrow{\sigma,(\!|\,|\!)} s''_2$. Let $s'_2 = \xi^{-1}(s''_2)$, then we have $s'_1 = \xi^{-1}(s''_1) \xrightarrow{\sigma',(\!|\,|\!)} \xi^{-1}(s''_2) = s'_2$ by Lemma A.8, with σ' the renaming of σ. As $s_2 \sqsubseteq s''_2$, then $s_2 \preceq \xi^{-1}(s''_2) = s'_2$. The inference rule env gives $s'_1 \xrightarrow{\sigma'} s'_2$, thus $s'_1 \to s'_2$. We have s'_2 state of $\mathcal{S}_a$, thus $\mathcal{S}_a$ is monotone. □

Intuitively, renaming preserves monotony because the roles of identifiers from parameter domains are symmetrical in the system, as proved by Lemma A.8. This is the reason why we do not allow the use of an element or a subset of a parameter domain as a constant in a PASTD specification. For example, the specification in Figure A.7 is invalid because it uses the event $E(1)$ in a transition. It is easy to see that ASTD $main[\{1\}]$ and $main[\{2\}]$ have different behaviors. Note that the event $E(1)$ from the right-hand side sub-ASTD is not affected by any rename function, as it is not part of any state expression.

Furthermore, the reachability property must satisfy the same constraint of symmetry, that is the set R must be upward-closed with regards to the qo $\preceq$. This should not be an issue, since, in most cases, properties do not deal with specific instances of a

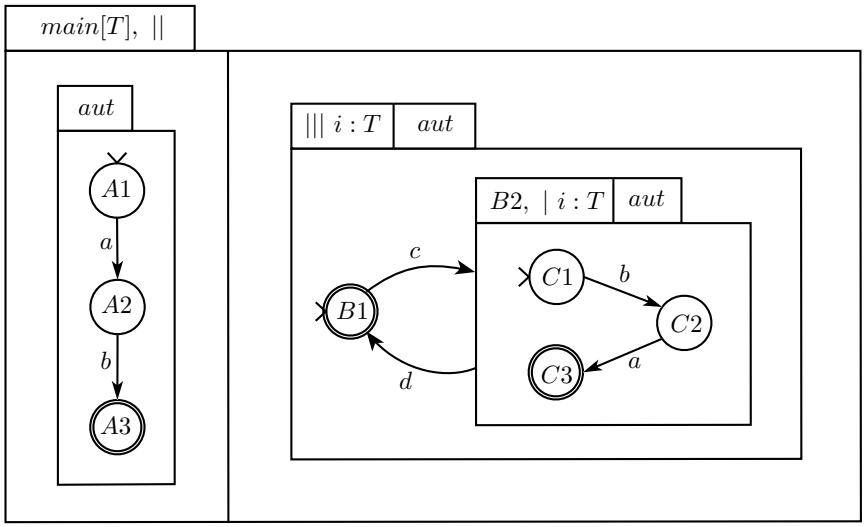

Figure A.8 – Example of PASTD

set of elements. Take for example the PASTD of Figure A.5, where parameter T has domain $\mathbb{N}$. Determining if the state $s = (\mathsf{aut}_\circ, S1, (|||:_\circ, \{1 \mapsto (\mathsf{aut}_\circ, Q2, (\mathsf{elem}_\circ))\}))$ is reachable is equivalent to determining if any state of kind $(\mathsf{aut}_\circ, S1, (|||:_\circ, \{id \mapsto (\mathsf{aut}_\circ, Q2, (\mathsf{elem}_\circ))\}))$, where id is an element of $\mathbb{N}$, is reachable.

A.5 Extending the State Space

In order to satisfy the *backward-downward monotony*, we need to make some modifications to the state space of PASTDs. Indeed, with the current definition of IPASTD states (Definition A.11), the bdm property does not hold and thus a state may not have a finite pred-basis.

Consider for example the PASTD of Figure A.8, with $\mathbb{N}$ as the parameter domain of T, and a state $s = (||_\circ, (\mathsf{aut}_\circ, A3, (\mathsf{elem}_\circ)), (|||:_\circ, f))$, where the function for the quantified interleaving is defined by $f = \{1 \mapsto (\mathsf{aut}_\circ, B2, (|:_\circ, 1, (\mathsf{aut}_\circ, C3, (\mathsf{elem}_\circ))))\}$, and with $s.pa = main$ and $s.val = (\{1\})$. Figure A.9 shows some elements of the set $\uparrow Pred(\uparrow s)$, which are not comparable. And as we can find an infinite number of such states, we can deduce that there is no finite basis for that set. The transition relation does not satisfy the bdm. This is because each instance appearing in the quantified

choice has to appear in the quantified interleaving too. (Note that it is not the case if the set of states is well-quasi-ordered.)

However, we can notice that the central part of the transition can be localized in the state structure and is the same one. Thus, we should be able to construct a lower bound for that set of predecessors, which would correspond to some state $s' = (||_\circ, (\mathsf{aut}_\circ, A2, (\mathsf{elem}_\circ)), (|||:_\circ, f'))$, where the function f' is defined by $f' = \{1 \mapsto (\mathsf{aut}_\circ, B2, (|:_\circ, 1, (\mathsf{aut}_\circ, C3, (\mathsf{elem}_\circ)))), 2 \mapsto (\mathsf{aut}_\circ, B2, (|:_\circ, 3, (\mathsf{aut}_\circ, C1, (\mathsf{elem}_\circ))))\}$, and with $s.pa = main$ and $s.val = (\{1, 2, 3\})$. But, as $dom(f') \neq \{1, 2, 3\}$, s' is not a well-formed state according to Definition A.11.

To overcome this issue, we augment the set of states to consider in PASTD systems by adding some specific abstract states. Intuitively, an abstract state is a state where the local configurations of some instances are unknown. Such an abstract state s represents the set of all concrete states where the quantified interleavings are completely instantiated but include the instances appearing in s within their configurations. Formally, the new set of IPASTD states is defined by similarly as Definition A.11 except for the last case.

Definition A.18. *We denote by $\mathcal{Q}^\sharp$ the set of IPASTD states such that $s \in \mathcal{Q}^\sharp$ iff either it verifies one of the cases from 1 to 6 of Definition A.11 or,*

7. *$s = (|||:_\circ, f)$ and*
   - *$s.pa = (|||:, n, y, T_i, b)$ and $dom(f) \subseteq V_i$ and $dom(f) \neq \emptyset$ and $s.val = \overrightarrow{V}$ and for all $ss \in ran(f)$ we have $ss.pa = b$ and $ss.val = \overrightarrow{V}$ and $ss \in \mathcal{Q}^\sharp$, where $T_i \in \overrightarrow{T}$ is a parameter and $V_i \in \overrightarrow{V}$ a value, or*
   - *$s.pa = (|||:, n, y, W, b)$ and $dom(f) = W$ and for all $ss \in ran(f)$ we have $ss.pa = b$ and $ss.val = s.val$ and $ss \in \mathcal{Q}^\sharp$, where W is not a parameter;*

The difference with Definition A.11 is that the quantified interleaving case $s = (|||:_\circ, f)$ with $s.pa = (|||:, n, y, T_i, b)$, does not require the domain of f to match the entire set of values V_i but only a non-empty subset, where T_i is a parameter. Clearly, $\mathcal{Q}^\sharp$ is a superset of $\mathcal{Q}$.

We can extend the quasi-orderings $\sqsubseteq$ and $\preceq$ of Definition A.14 and A.17 to the set $\mathcal{Q}^\sharp$. Indeed, Definitions A.14 and A.16 can be applied to the set $\mathcal{Q}^\sharp$, as the quantified

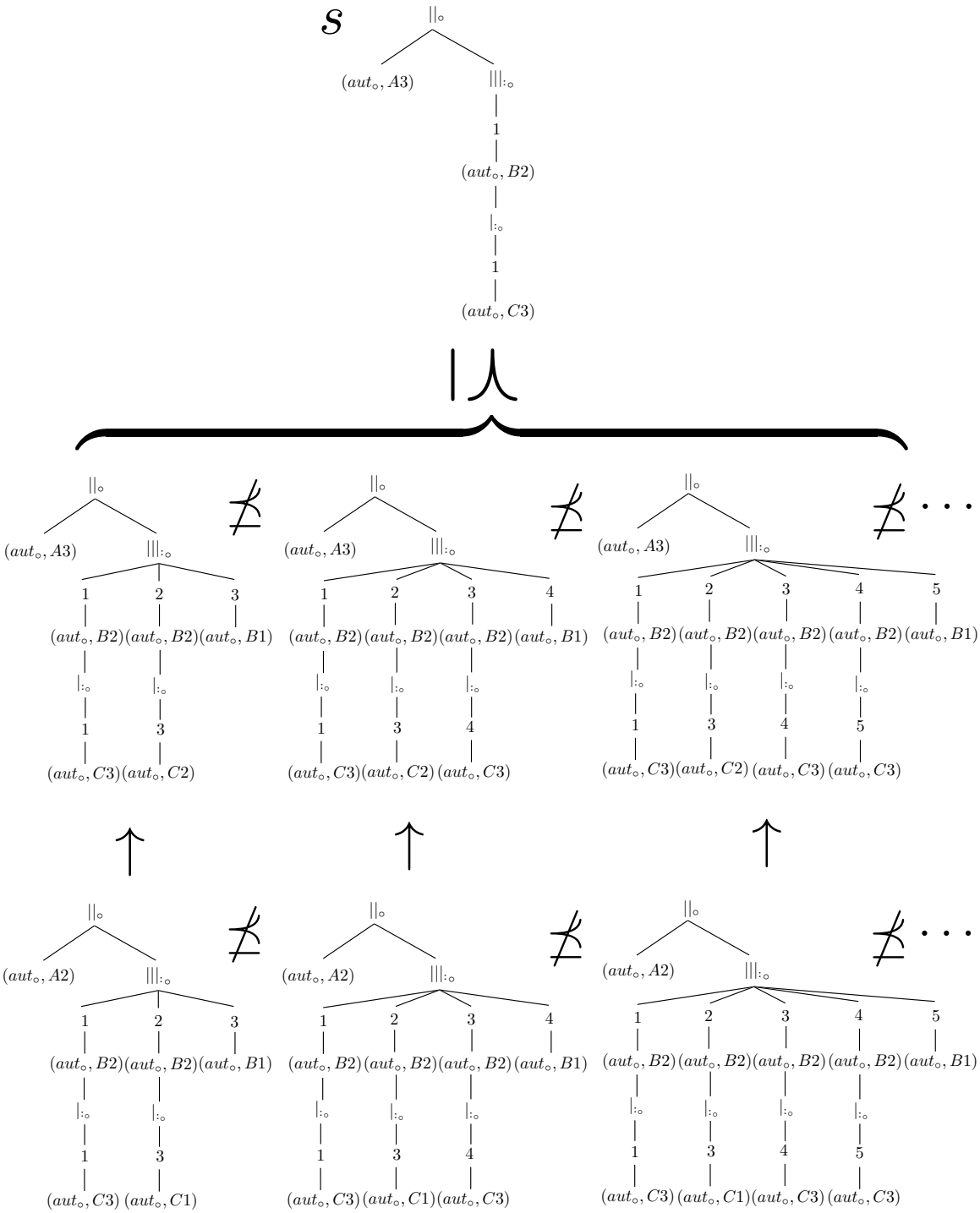

Figure A.9 – Example of a infinite antichain of $Pred(\uparrow s)$

interleaving cases $(|||:_\circ, f)$ of both definitions refer to the domain of the function f. We keep the same notation for the quasi-ordering extended to $\mathcal{Q}^\sharp$.

Proposition A.3. *The relations $\sqsubseteq$ and $\preceq$ are quasi-ordering on $\mathcal{Q}^\sharp$.*

Considering the new set of states $\mathcal{Q}^\sharp$, the TS associated to an IPASTD has its set of states augmented as well as the transition relation. First, let us introduce a function *Abinit* which characterizes the set of abstract initial states.

Definition A.19. *Let $a[\overrightarrow{T}]$ be a PASTD and $\overrightarrow{V}$ a parameter value. We define the set $Abinit(a[\overrightarrow{V}])$ by:*

$$Abinit(a[\overrightarrow{V}]) = \{s \in \mathcal{Q}^\sharp \mid s \sqsubseteq init(a[\overrightarrow{V}])\}$$

For any concrete initial state $init(a[\overrightarrow{V}])$, we define the set of its corresponding abstract states $Abinit(a[\overrightarrow{V}])$ by the abstract states from $\mathcal{Q}^\sharp$ whose branches can be completed to obtain $init(a[\overrightarrow{V}])$.

Furthermore, we replace some transition rules from Definition A.6 to deal with abstract initial states. Each condition involving an initial state $init(a)$ is simply replaced by a condition that uses the function *Abinit*.

Definition A.20. *The operational semantics of PASTDs is given by Definition A.6 except for the following cases.*

- *If $a[\overrightarrow{V}] = (\mathsf{aut}, n, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{V}]$, the rule aut_1 is replaced by the following rule aut_{1a}:*

$$\mathsf{aut}_{1a}\ \frac{\delta(n_1, n_2, \sigma', final?) \qquad final? \Rightarrow final(s) \wedge \sigma'[\Gamma] = \sigma \qquad \chi}{(\mathsf{aut}_\circ, n_1, s) \xrightarrow{\sigma,\Gamma} (\mathsf{aut}_\circ, n_2, s')}$$

  where $\chi = s' \in Abinit(\nu(n_2)[\overrightarrow{V}])$

- *If $a[\overrightarrow{V}] = (\star, n, b[\overrightarrow{V}])$, the rule $\star_1$ is replaced by the following rule $\star_{1a}$:*

$$\star_{1a}\ \frac{(final(s) \vee \neg started?) \qquad q \xrightarrow{\sigma,\Gamma}_{b[\overrightarrow{V}]} s' \qquad q \in Abinit(b[\overrightarrow{V}])}{(\star_\circ, started?, s) \xrightarrow{\sigma,\Gamma} (\star_\circ, \text{true}, s')}$$

- *If* $a[\overrightarrow{V}] = (|, n, l[\overrightarrow{V}], r[\overrightarrow{V}])$*, rules* $|_1$ *and* $|_2$ *are respectively replaced by rules* $|_{1a}$ *and* $|_{2a}$*:*

$$|_{1a}\ \frac{q \xrightarrow{\sigma,\Gamma}_{l[\overrightarrow{V}]} s' \qquad q \in Abinit(l[\overrightarrow{V}])}{(|_\circ, \bot, \bot) \xrightarrow{\sigma,\Gamma} (|_\circ, \mathit{left}, s')}$$

$$|_{2a}\ \frac{q \xrightarrow{\sigma,\Gamma}_{r[\overrightarrow{V}]} s' \qquad q \in Abinit(r[\overrightarrow{V}])}{(|_\circ, \bot, \bot) \xrightarrow{\sigma,\Gamma} (|_\circ, \mathit{right}, s')}$$

- *If* $a[\overrightarrow{V}] = (|:, n, x, T, b[\overrightarrow{V}])$*, the rule* $|:_1$ *is replaced by the following rule:*

$$|:_{1a}\ \frac{q \xrightarrow{\sigma,([x:=v])\triangleleft\Gamma}_{b[\overrightarrow{V}]} s' \qquad q \in Abinit(b[\overrightarrow{V}]) \qquad v \in T}{(|:_\circ, \bot, \bot) \xrightarrow{\sigma,\Gamma} (|:_\circ, v, s')}$$

Note that for each transition $s \xrightarrow{\sigma,\Gamma} s'$*, we still have implicitly that* $s.pa = s'.pa$ *and* $s.val = s'.val$*. Moreover, these rules are specific to PASTDs because of the definition of the function Abinit.*

The other rules from Definition A.6 take into account those abstract states as the rule for the quantified interleaving operator applies to a function f with any domain of definition.

Similarly, we redefine the TS associated to an IPASTD by adding the abstract initial states.

Definition A.21. *Let* $a[\overrightarrow{T}]$ *be a PASTD with parameter domains* $\overrightarrow{P}$ *and* $\overrightarrow{V} \subset \overrightarrow{P}$ *a vector of values. If the IPASTD* $a[\overrightarrow{V}]$ *contains no free variables, then we define the corresponding TS* $\mathcal{S}^\sharp_{a,\overrightarrow{V}} = (Q, \rightarrow)$ *with abstract states as follows:*

- *Q consists of the IPASTD states* $s \in \mathcal{Q}^\sharp$ *such that* $s.pa = a$ *and* $s.val = \overrightarrow{V}$*;*
- $\rightarrow$ *is the set of all transitions* $s \xrightarrow{\sigma} s'$ *that can be derived in* $a[\overrightarrow{V}]$ *following Definition A.20, where* s *and* s' *are in* Q*;*
- $I \subseteq Q$*, the set of initial states, is the set* $\{q \mid q \in Abinit(a[\overrightarrow{V}])\}$*.*

By convention, if $a[\overrightarrow{V}]$ is an IPASTD and $\mathcal{S}_{a,\overrightarrow{V}}$ the TS derived from Definition A.11, then we denote by $\mathcal{S}^\sharp_{a,\overrightarrow{V}}$ the TS derived from the previous definition. For a PASTD $a[\overrightarrow{T}]$ the TS corresponding to the definition with abstract states is given as follows (similar to Definition A.13).

Definition A.22. *Let $a[\overrightarrow{T}]$ be a PASTD with parameter domains $\overrightarrow{P}$. If $a[\overrightarrow{T}]$ contains no free variables other than $\overrightarrow{T}$, then we define the corresponding TS $\mathcal{S}^\sharp_a$ by the union of all possible instantiations of system:*

$$\mathcal{S}^\sharp_a = \bigcup_{\overrightarrow{V}} \mathcal{S}^\sharp_{a,\overrightarrow{V}}$$

where $\overrightarrow{V} = (V_1, \ldots, V_k)$, $\overrightarrow{P} = (P_1, \ldots, P_k)$ and V_i takes every values in the set of non-empty finite subset of P_i.

Compared to the TS $\mathcal{S}_a$, the new system $\mathcal{S}^\sharp_a$ has a larger set of states and of transitions.

Now, we must ensure that the extended quasi-ordering preserves the monotony of $\mathcal{S}^\sharp_a$. The proof is similar to the one for Theorem A.1 and is given in Appendix B.4.

Theorem A.3. *Let $a[\overrightarrow{T}]$ be a PASTD without free variables in events, with $\mathcal{S}^\sharp_a$ the corresponding TS from Definition A.22. $\mathcal{S}^\sharp_a$ is monotone wrt $\sqsubseteq$ and $\preceq$.*

If we change the TS to consider in the procedure, we need to show that the verification is still correct. Consider a PASTD $a[\overrightarrow{T}]$, the TS $\mathcal{S}_a = (Q, \rightarrow)$ with initial states $I \subseteq Q$, which is derived from Definition A.13, and the TS $\mathcal{S}^\sharp_a = (Q', \rightarrow')$ with initial states $I' \subseteq Q'$ from Definition A.22. Suppose that we want to determine if $s \in Q$ is coverable in $a[\overrightarrow{T}]$. We can replace $\mathcal{S}_a$ by $\mathcal{S}^\sharp_a$ in the verification procedure only if they are equivalent with regards to the coverability problem, *i.e.* $cov(\mathcal{S}_a, I, s) \iff cov(\mathcal{S}^\sharp_a, I', s)$.

Lemma A.9. *Let $a[\overrightarrow{T}]$ be a PASTD, $\mathcal{S}_a = (Q, \rightarrow)$ the TS derived from Definition A.13 with initial states I, and $\mathcal{S}^\sharp_a = (Q', \rightarrow')$ the TS from Definition A.22 with initial states I'. Then, for all path $s'_0 \rightarrow' \cdots \rightarrow' s'_n$ in $\mathcal{S}^\sharp_a$ with $s'_0 \in I'$, there exists a path $s_0 \rightarrow \cdots \rightarrow s_n$ in $\mathcal{S}_a$ with $s_0 \in I$ such that $s'_i \leq s_i$ for all $i \leq n$.*

We can now state the theorem that supports our extension of state space.

Theorem A.4. *Let $a[\overrightarrow{T}]$ be a PASTD, $\mathcal{S}_a = (Q, \rightarrow)$ the TS derived from Definition A.13 with initial states I, and $\mathcal{S}_a^\sharp = (Q', \rightarrow')$ the TS from Definition A.22 with initial states I'. Let $s \in Q$ a state. Then, $cov(\mathcal{S}_a, I, s) \iff cov(\mathcal{S}_a^\sharp, I', s)$.*

Proof. $cov(\mathcal{S}_a, I, s) \implies cov(\mathcal{S}_a^\sharp, I', s)$ is trivial. By Lemma A.9, $cov(\mathcal{S}_a, I, s) \impliedby cov(\mathcal{S}_a^\sharp, I', s)$. □

Remark that some interesting reachability properties in $\mathcal{Q}^\sharp$ cannot be represented in $\mathcal{Q}$. Take for instance the example of Figure A.8. Let R be the set of states greater than the state $(||_\circ, (\mathsf{aut}_\circ, A3, (\mathsf{elem}_\circ)), (|||:_\circ, f))$, such that $f = \{i \mapsto (\mathsf{aut}_\circ, B2, (|:_\circ, j, (\mathsf{aut}_\circ, C2, (\mathsf{elem}_\circ))))\}$, for all $i \neq j$. We have seen in Figure A.9 that R has no finite basis in $\mathcal{Q}$ but can be represented by a finite basis in $\mathcal{Q}^\sharp$. We can then verify the coverabilty of a finite set of states in $\mathcal{Q}^\sharp$. From now, we will consider that coverability problems are directly specified in $\mathcal{Q}^\sharp$.

A.6 PASTDs are RMTSs

In this section we use RMTSs to prove that PASTDs have effective pred-basis. Because we use a quasi-ordering on the state space of PASTDs, we cannot say that PASTDs are RMTSs as they require a partial ordering. However, it is natural to consider the quotient set of the state space instead of the entire space when studying systems like PASTDs. Exploiting the symmetries in systems in order to simplify a verification process is usual in the context of formal verification [36]. In our case the symmetries entailed by the equivalence relation allow us to consider the coverability problem on the quotient space.

For a PASTD $a[\overrightarrow{T}]$ with $\mathcal{S}_a^\sharp = (Q, \rightarrow)$, $\mathcal{T}_a/\!\sim$ is the quotient set of $\mathcal{T}_a$ and for any relation (transition relation or ordering in our case) on $\mathcal{T}_a$, we use the same notation for the corresponding one on $\mathcal{T}_a/\!\sim$. More formally, $\tilde{s_1} \preceq \tilde{s_2}$ if there is $s_1 \in \tilde{s_1}$ and $s_2 \in \tilde{s_2}$ such that $s_1 \preceq s_2$, and $\tilde{s_1} \rightarrow \tilde{s_2}$ if there is $s_1 \in \tilde{s_1}$ and $s_2 \in \tilde{s_2}$ such that $s_1 \rightarrow s_2$. We denote by $\tilde{\mathcal{S}}_a = (\mathcal{T}_a/\!\sim, \rightarrow)$ the quotient system.

Proposition A.4. *Let $a[\overrightarrow{T}]$ be a PASTD and $\mathcal{S}_a^\sharp = (Q, \rightarrow)$. Then, $(\mathcal{T}_a, \rightarrow, \sim)$ and $(\mathcal{T}_a/\!\sim, \rightarrow, \preceq)$ are both monotone.*

Proof. The proof of monotony of $(\mathcal{T}_a, \rightarrow, \sim)$ is similar to the one for $(\mathcal{T}_a, \rightarrow, \preceq)$. Let us prove that $(\mathcal{T}_a/\sim, \rightarrow, \preceq)$ is monotone. Let $\tilde{s_1}, \tilde{s_1'}, \tilde{s_2} \in \mathcal{T}_a/\sim$ such that $\tilde{s_1} \preceq \tilde{s_1'}$ and $\tilde{s_1} \rightarrow \tilde{s_2}$. Then, there exist $s_1, q_1 \in \tilde{s_1}$ and $s_2 \in \tilde{s_2}$ and $s_1' \in \tilde{s_1'}$ such that $q_1 \preceq s_1'$ and $s_1 \rightarrow s_2$. By monotony of $(\mathcal{T}_a, \rightarrow, \sim)$, there exists $q_2 \in \tilde{s_2}$ such that $q_1 \rightarrow q_2$. Thus, by monotony of $(\mathcal{T}_a, \rightarrow, \preceq)$, there exists $s_2' \in \mathcal{T}_a$ such that $q_2 \preceq s_2'$ and $s_1' \rightarrow s_2'$. Let $\tilde{s_2'}$ be the equivalence class of s_2'. Then, we have $\tilde{s_2} \preceq \tilde{s_2'}$ and $\tilde{s_1'} \rightarrow \tilde{s_2'}$. □

The monotony of $(\mathcal{T}_a, \rightarrow, \sim)$ allows to prove that the coverability problem in $\mathcal{S}_a^\sharp$ can be reduced to the coverability problem in $\tilde{\mathcal{S}}_a$.

Theorem A.5. *Let $a[\overrightarrow{T}]$ be a PASTD, $\mathcal{S}_a^\sharp = (\mathcal{T}_a, \rightarrow, \preceq)$ its corresponding OTS and $\tilde{\mathcal{S}}_a = (\mathcal{T}_a/\sim, \rightarrow, \preceq)$ its quotient OTS. Let $I \subseteq \mathcal{T}_a$, $s \in \mathcal{T}_a$, $\tilde{I} = \{\tilde{i} \in \mathcal{T}_a/\sim \mid \exists i \in I \cdot i \in \tilde{i}\}$ and $\tilde{s}$ the equivalence class of s. Then, $cov(\mathcal{S}_a^\sharp, I, s) \iff cov(\tilde{\mathcal{S}}_a, \tilde{I}, \tilde{s})$.*

Proof. Let us prove that $\exists i \xrightarrow{*} s' \cdot s \preceq s' \wedge i \in I \Leftrightarrow \exists \tilde{i} \xrightarrow{*} \tilde{s'} \cdot \tilde{s} \preceq \tilde{s'} \wedge \tilde{i} \in \tilde{I}$. $\Rightarrow$ is trivial. $\Leftarrow$ is true because $(\mathcal{T}_a, \rightarrow, \sim)$ is monotone (see Proposition A.4). □

From now on, for a PASTD $a[\overrightarrow{T}]$ we will study the coverability problem on the quotient MTS $\tilde{\mathcal{S}}_a^\sharp = (\mathcal{T}_a/\sim, \rightarrow, \preceq)$. The objective is to find an RMTS for $a[\overrightarrow{T}]$. Let us define an adequate rank function.

Definition A.23. *We define a function $\gamma : \mathcal{Q}^\sharp \rightarrow \mathbb{N}^k$ on the set of IPASTD states by $\gamma(s) = (|V_1|, \ldots, |V_k|)$ for all $s \in \mathcal{Q}^\sharp$ with $s.val = (V_1, \ldots, V_k)$.*

The function γ gives for each state s the number of instances of each type that appear within the description of s. For instance, if s is the state of Figure A.4 from the library example, then $\gamma(s) = (2, 3)$ as $s.val = (\{m_1, m_2\}, \{b_1, b_2, b_3\})$. The state describes a configuration with two members and three books.

Clearly, for all $s_1, s_2 \in \tilde{s} \in \mathcal{Q}^\sharp/\sim$, $\gamma(s_1) = \gamma(s_2)$. Thus, we can extend the rank function to the equivalence classes $\gamma : \mathcal{Q}^\sharp/\sim \rightarrow \mathbb{N}^k$ as $\gamma(\tilde{s}) = \gamma(s)$ for any $s \in \tilde{s}$. The function has the following property with regards to transitions.

Proposition A.5. *For all $\tilde{s_1}, \tilde{s_2} \in \mathcal{Q}^\sharp/\sim$, such that $\tilde{s_1} \rightarrow \tilde{s_2}$, we have $\gamma(\tilde{s_1}) = \gamma(\tilde{s_2})$.*

Proof. By Definitions A.20 and A.23. □

Now, for each PASTD, we can determine a $c \in \mathbb{N}^n$ which is a bound satisfying Condition 4 of Definition 4.14.

Definition A.24 (Maximum transition size)**.** *Let $A = (F, \overrightarrow{T}, \overrightarrow{P})$ be a PASTD. The function μ, which gives for each PASTD expression F a vector of natural $c \in \mathbb{N}^n$ called maximum transition size, is defined recursively as follows.*

1. $\mu((\mathit{elem})[\overrightarrow{T}]) = (0, \ldots, 0)$
2. $\mu((\mathit{aut}, n, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]) = sup(\{\mu(\nu(n')) \mid n' \in N\})$
3. $\mu((\star, n, b)[\overrightarrow{T}]) = \mu(b[\overrightarrow{T}])$
4. $\mu((|, n, l, r)[\overrightarrow{T}]) = sup(\{\mu(l[\overrightarrow{T}]), \mu(r[\overrightarrow{T}])\})$
5. $\mu((||, n, \Delta, l, r)[\overrightarrow{T}]) = \mu(l[\overrightarrow{T}]) + \mu(r[\overrightarrow{T}])$
6. $\mu((|:, n, x, X, b)[\overrightarrow{T}]) = \begin{cases} \mu(b[\overrightarrow{T}]) & \text{if } X \text{ is not a parameter} \\ \mu(b[\overrightarrow{T}]) + (u_1, \ldots, u_k) & \text{if } X = T_i \\ & \quad \text{where } u_i = 1 \text{ and} \\ & \quad u_j = 0 \text{ for all } i \neq j \end{cases}$
7. $\mu((|||:, n, x, X, b)[\overrightarrow{T}]) = \begin{cases} |X| \cdot \mu(b[\overrightarrow{T}]) & \text{if } X \text{ is not a parameter} \\ \mu(b[\overrightarrow{T}]) + (u_1, \ldots, u_k) & \text{if } X = T_i \\ & \quad \text{where } u_i = 1 \text{ and} \\ & \quad u_j = 0 \text{ for all } i \neq j \end{cases}$

Note that for all $C \subset \mathbb{N}^k$, we denote by $sup(C)$ the supremum of the set C in $\mathbb{N}^k$.

For instance, consider the PASTD of the library illustrated in Figure A.1. By Definition A.24, we have $\mu(F) = (2, 2)$ (remark that each parameter appears twice in the tree). It means that for each transition in the system, a maximum of two members and two books will be actually involved.

The function μ determines the maximum number of instances of each type that are involved in a transition. Intuitively, we count one instance for each quantified operator associated to the corresponding parameter. Remark that there will be at least one instance for each parameter that is used in a quantified operator. As the function μ

is recursive, $\mu(a[\overrightarrow{T}])$ may return 0 in one or more components when $a[\overrightarrow{T}]$ is a sub-PASTD. In that case, it means that the associated parameter T_i is not used in $a[\overrightarrow{T}]$. Thus, it should be possible to find a sub-state of rank $\mu(a[\overrightarrow{T}])$ because the empty parameter value will have no consequence in the construction of the sub-state. For instance, the PASTD (elem) with parameters $(T_1, \ldots, T_k)$ has maximum transition size $(0, \ldots, 0)$ and we can find a state $s = (\mathsf{elem}_\circ)$ such that $s.val = (\emptyset, \ldots, \emptyset)$.

The function μ will allow us to prove the backward-downward monotony. But first, let us examine some properties of the set of states $\mathcal{Q}^\sharp$. Thanks to the extension of the state space, we are able to construct abstract states from a concrete one by either extending the parameter value without changing the state expression or by pruning, in some cases, some branches of a state expression. Those two operations are essential to the proof of the bdm and are represented by the following lemmas.

Lemma A.10 (Lifting)**.** *Let $a[\overrightarrow{T}]$ be a PASTD and $\overrightarrow{V}$ a parameter value. Let $q_1, q_2 \in \mathcal{Q}^\sharp$ such that $q_1.pa = q_2.pa = a[\overrightarrow{T}]$ and $q_1 \sqsubseteq q_2$. For all value $\overrightarrow{V}$ such that $q_1.val \subseteq \overrightarrow{V} \subseteq q_2.val$, there exists $q \in \mathcal{Q}^\sharp$ such that $q_1 \sqsubseteq q \sqsubseteq q_2$ and $q.val = \overrightarrow{V}$.*

Proof. Take $q \in \mathcal{Q}^\sharp$ such that q has the same state expression and same PASTD as q_1 but with $q.val = \overrightarrow{V}$. This is possible by definition of $\mathcal{Q}^\sharp$, which allows the domain of the function of the quantified interleaving operator to be partial. We still have that $q \sqsubseteq q_2$. □

Lemma A.10 allows to easily pick an intermediary state for a given parameter value between two ordered states. It does not hold within $\mathcal{Q}$ as pruning some branches (of the quantified interleaving operator) from the greater state q_2 may results into a partially defined state. On the other hand, it is not always possible to scale a state down to any parameter value, but we can find a smaller state whose rank is bounded by a specific value. Indeed, for all state, there is a smaller one whose size is bounded by the value given by the function of Definition A.24.

Lemma A.11 (Pruning)**.** *Let $a[\overrightarrow{T}]$ be a PASTD. Let $c \in \mathbb{N}^k$ such that $c = \mu(a[\overrightarrow{T}])$. For all $s \in \mathcal{Q}^\sharp$ such that $s.pa = a[\overrightarrow{T}]$, there exists $q \in \mathcal{Q}^\sharp$ such that $q \sqsubseteq s$ and $\gamma(q) \leq_k c$.*

Remark that the maximum transition size coincides with the upper bound for the smallest valid part of a state, that we will prove further. First, let us show that pruning also preserves final states.

Lemma A.12 (Pruning final)**.** *Let $a[\overrightarrow{T}]$ be a PASTD. Let $c \in \mathbb{N}^k$ such that $c = \mu(a[\overrightarrow{T}])$. For all $s \in \mathcal{Q}^\sharp$ such that $s.pa = a[\overrightarrow{T}]$, if final$(s)$, then there exists $q \in \mathcal{Q}^\sharp$ such that $q \sqsubseteq s$, $\gamma(q) \leq_k c$ and final(q).*

Considering the previous lemmas, we are now able to show the bdm for $\mathcal{S}_a^\sharp$. Detailed proofs are given in Appendix B.5.

Lemma A.13. *Let $a[\overrightarrow{T}]$ be a PASTD. Let $c \in \mathbb{N}^k$ such that $c = \mu(a[\overrightarrow{T}])$. For all $s_1, s_2 \in \mathcal{Q}^\sharp$ such that $s_1.pa = s_2.pa = a[\overrightarrow{T}]$ and $s_1 \xrightarrow{\sigma,\Gamma} s_2$, there exist $q_1, q_2 \in \mathcal{Q}^\sharp$ such that $q_1.pa = q_2.pa = a[\overrightarrow{T}]$ and $q_1 \xrightarrow{\sigma,\Gamma} q_2$ and:*

1. *$\gamma(q_1) = \gamma(q_2) \leq_k c$, and*
2. *$q_1 \preceq s_1$ and $q_2 \preceq s_2$, and*
3. *for all $q_2' \in \mathcal{Q}^\sharp$ such that $q_2 \preceq q_2' \preceq s_2$, there exists $q_1' \in \mathcal{Q}^\sharp$ such that $q_1 \preceq q_1' \preceq s_1$ and $q_1' \xrightarrow{\sigma',\Gamma'} q_2'$, where σ' and Γ' are the renamed event and environment with regards to the rename function ξ such that $q_2 \sqsubseteq \xi(q_2')$.*

Lemma A.14. *Let $a[\overrightarrow{T}]$ be a PASTD, $\tilde{\mathcal{S}}_a = (\mathcal{T}_a/\!\sim, \rightarrow, \preceq)$ and $c = \mu(a[\overrightarrow{T}])$. For each transition $\tilde{s_1} \rightarrow \tilde{s_2}$ in $\tilde{\mathcal{S}}_a$, there is $\tilde{q_1} \rightarrow \tilde{q_2}$ in $\tilde{\mathcal{S}}_a$ such that $\tilde{q_1} \preceq \tilde{s_1}$, $\tilde{q_2} \preceq \tilde{s_2}$, $\gamma(\tilde{q_1}) \leq c$, $\gamma(\tilde{q_2}) \leq c$, and $\forall q_2' \in \mathcal{T}_a/\!\sim \cdot\, \tilde{q_2} \preceq \tilde{q_2'} \preceq \tilde{s_2} \implies \exists \tilde{q_1'} \in \mathcal{T}_a/\!\sim \cdot\, \tilde{q_1'} \rightarrow \tilde{q_2'} \wedge \tilde{q_1} \preceq \tilde{q_1'} \preceq \tilde{s_1}$.*

Proof. By application of the env rule and Lemma A.13. □

Now, let us prove that Condition 2 of Definition 4.14 is satisfied, *i.e.* if two PASTD states have an upper bound, they have a "bounded" one according to the rank function γ.

Lemma A.15. *Let $a[\overrightarrow{T}]$ be a PASTD and $\tilde{\mathcal{S}}_a = (\mathcal{T}_a/\!\sim, \rightarrow, \preceq)$. For all $\tilde{s_1}, \tilde{s_2}, \tilde{s_3} \in \mathcal{T}_a/\!\sim$ such that $\tilde{s_1} \preceq \tilde{s_3}$ and $\tilde{s_2} \preceq \tilde{s_3}$, there exists $\tilde{s_4} \in \mathcal{T}_a/\!\sim$ such that $\tilde{s_1} \preceq \tilde{s_4}$, $\tilde{s_2} \preceq \tilde{s_4}$, $\tilde{s_4} \preceq \tilde{s_3}$ and $\gamma(\tilde{s_4}) \leq \gamma(\tilde{s_1}) + \gamma(\tilde{s_2})$.*

Proof. Let $\tilde{s_1}, \tilde{s_2}, \tilde{s_3} \in \mathcal{T}_a/\sim$ such that $\tilde{s_1} \preceq \tilde{s_3}$ and $\tilde{s_2} \preceq \tilde{s_3}$. Then, there is $s_1 \in \tilde{s_1}, s_2 \in \tilde{s_2}, s_3 \in \tilde{s_3}$ such that $s_1 \preceq s_3$ and $s_2 \preceq s_3$. We construct s_4 as the least state $s_4 \sqsubseteq s_3$ such that $s_1 \preceq s_4$ and $s_2 \preceq s_4$. s_4 is a well-formed state because s_1, s_2 and s_3 are from the same PASTD and only branches from quantified interleaving nodes are pruned from s_3 to obtain s_4. Hence, $s_4 \in \mathcal{T}_a$. By definition of γ, we have that $\gamma(s_4) \leq \gamma(s_1) + \gamma(s_2)$. Take $\tilde{s_4} \in \mathcal{T}_a/\sim$ the equivalence class of s_4. We have $\tilde{s_1} \preceq \tilde{s_4}$, $\tilde{s_2} \preceq \tilde{s_4}$, $\tilde{s_4} \preceq \tilde{s_3}$ and $\gamma(\tilde{s_4}) \leq \gamma(\tilde{s_1}) + \gamma(\tilde{s_2})$. □

Intuitively, take the example of the library and consider a state s_1 such that $\gamma(s_1) = (1, 1)$ and where the member m_1 borrowed the book b_1 and a state s_2 such that $\gamma(s_2) = (1, 1)$ and where the member m_2 borrowed the book b_2, then clearly s_1 and s_2 have upper bounds. Besides, we can construct a "little" state s_3 such that $\gamma(s_3) = (2, 2)$ including the two different loans.

We can now state that PASTDs are effective RMTSs.

Theorem A.6. *Let $a[\overrightarrow{T}]$ be a PASTD and $\tilde{\mathcal{S}}_a = (\mathcal{T}_a/\sim, \rightarrow, \preceq)$ the quotient MTS, then $(\mathcal{T}_a/\sim, \rightarrow, \preceq, \gamma, \mu(a[\overrightarrow{T}]), \overrightarrow{0})$ is an effective RMTS.*

Proof. Let $\mathcal{S} = (\mathcal{T}_a/\sim, \rightarrow, \preceq, \gamma, \mu(a[\overrightarrow{T}]), \overrightarrow{0})$. $(\mathcal{T}_a/\sim, \rightarrow, \preceq)$ is an MTS by Proposition A.4 and $\preceq$ a po on $\mathcal{T}_a/\sim$.

1. Let $\tilde{s}, \tilde{s'} \in \mathcal{T}_a/\sim$ such that $\tilde{s} \preceq \tilde{s'}$. We clearly have $\gamma(\tilde{s}) \leq \gamma(\tilde{s'})$ by definition of γ. Thus, γ is monotonic. Let $r \in \mathbb{N}^k$. $\gamma^{-1}(r)$ is finite by a combinatorial argument.

2. By Lemma A.15, for all $\tilde{s_1}, \tilde{s_2}, \tilde{s_3} \in \mathcal{T}_a/\sim$ such that $\tilde{s_1} \preceq \tilde{s_3}$ and $\tilde{s_2} \preceq \tilde{s_3}$, there exists $\tilde{s_4} \in \mathcal{T}_a/\sim$ such that $\tilde{s_1} \preceq \tilde{s_4}$, $\tilde{s_2} \preceq \tilde{s_4}$, $\tilde{s_4} \preceq \tilde{s_3}$ and $\gamma(\tilde{s_4}) \leq \gamma(\tilde{s_1}) + \gamma(\tilde{s_2})$.

3. For all $\tilde{s_1} \rightarrow \tilde{s_2}$, $\gamma(\tilde{s_1}) \leq \gamma(\tilde{s_2}) + \overrightarrow{0}$ because $\gamma(\tilde{s_1}) = \gamma(\tilde{s_2})$ by Proposition A.5.

4. By Lemma A.14, for all $\tilde{s_1} \rightarrow \tilde{s_2}$, there is $\tilde{q_1} \rightarrow \tilde{q_2}$ such that $\tilde{q_1} \preceq \tilde{s_1}$, $\tilde{q_2} \preceq \tilde{s_2}$, $\gamma(\tilde{q_1}) \leq \mu(a[\overrightarrow{T}])$, $\gamma(\tilde{q_2}) \leq \mu(a[\overrightarrow{T}])$, and $\forall \tilde{q'_2} \in \mathcal{T}_a/\sim \cdot \tilde{q_2} \preceq \tilde{q'_2} \preceq \tilde{s_2} \implies \exists \tilde{q'_1} \in \mathcal{T}_a/\sim \cdot \tilde{q'_1} \rightarrow \tilde{q'_2} \wedge \tilde{q_1} \preceq \tilde{q'_1} \preceq \tilde{s_1}$.

As a consequence, $\mathcal{S}$ is an RMTS. The transition relation is decidable as it is defined by a set of rules in [41]. Also by definition, $\preceq$ is decidable and $\gamma(\tilde{s})$ computable for

$\tilde{s} \in \mathcal{T}_a/\sim$. For $r \in \mathbb{N}^k$, $\gamma^{-1}(r)$ is computable by enumerating all well-formed states $\tilde{s}$ such that $\gamma(\tilde{s}) = r$. Thus, $\mathcal{S}$ is an effective RMTS. □

For any PASTD, we can conclude that we can compute a finite pred-basis of any state s. However, if we want to compute a finite basis for $Pred^*(\uparrow s)$ and decide coverability, we need the wqo condition.

Annexe B

Proofs

B.1 Proofs of Section A.3

Lemma A.1. Let $a[\overrightarrow{T}]$ be a PASTD. For any parameter value $\overrightarrow{V}$, $init(a[\overrightarrow{V}]) \in \mathcal{Q}$.

Proof. By structural induction on $a[\overrightarrow{T}]$, we show that each case match Definition A.11. Note that $init(a[\overrightarrow{V}]).pa = a[\overrightarrow{T}]$ and $init(a[\overrightarrow{V}]).val = \overrightarrow{V}$.

1. Base case: $a[\overrightarrow{T}] = (\mathsf{elem})$. We have $init(a[\overrightarrow{V}]) = (\mathsf{elem}_\circ) \in \mathcal{Q}$.
2. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$. By definition of *init*, we have $init(a[\overrightarrow{V}]) = (\mathsf{aut}_\circ, n_0, init(\nu(n_0)[\overrightarrow{V}]))$. By induction hypothesis, we have $init(\nu(n_0)[\overrightarrow{V}]) \in \mathcal{Q}$. Thus, $init(a[\overrightarrow{V}]) \in \mathcal{Q}$.
3. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. We have $init(a[\overrightarrow{V}]) = (\star_\circ, \text{false}, \bot) \in \mathcal{Q}$.
4. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. We have $init(a[\overrightarrow{V}]) = (|_\circ, \bot, \bot) \in \mathcal{Q}$.
5. Inductive case: $a[\overrightarrow{T}] = (||, n, \Delta, l, r)[\overrightarrow{T}]$. We have $init(a[\overrightarrow{V}]) = (||_\circ, init(l[\overrightarrow{V}]), init(r[\overrightarrow{V}]))$. By induction hypothesis, $init(l[\overrightarrow{V}]) \in \mathcal{Q}$ and $init(r[\overrightarrow{V}]) \in \mathcal{Q}$. Thus, $init(a[\overrightarrow{V}]) \in \mathcal{Q}$.
6. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b)[\overrightarrow{T}]$. We have $init(a[\overrightarrow{V}]) = (|:_\circ, \bot, \bot)$. Thus, $init(a[\overrightarrow{V}]) \in \mathcal{Q}$.
7. Inductive case: $a[\overrightarrow{T}] = (|||:, n, x, X, b)[\overrightarrow{T}]$. By cases on X.

- If $X = T_i \in \overrightarrow{T}$. We have $init(a[\overrightarrow{V}]) = (|||:_\circ, V_i \times \{init(b[\overrightarrow{V}])\})$. By induction hypothesis, $init(b[\overrightarrow{V}]) \in \mathcal{Q}$. Thus, $init(a[\overrightarrow{V}]) \in \mathcal{Q}$.
- If $X = W$ is not a parameter, we can conclude as well by induction hypothesis.

□

Lemma A.2. Let $a[\overrightarrow{T}]$ be a PASTD and $\overrightarrow{V}$ a parameter value. Let s_1, s_2 two states such that $s_1.pa = s_2.pa = a$ and $s_1.val = s_2.val = \overrightarrow{V}$. If $s_1 \in \mathcal{Q}$ and $s_1 \xrightarrow{\sigma,\Gamma} s_2$, then $s_2 \in \mathcal{Q}$.

Proof. Let s_1, s_2 two states, σ an event and Γ an environment such that $s_1 \xrightarrow{\sigma,\Gamma} s_2$ and $s_1 \in \mathcal{Q}$. By structural induction on $s_1.pa = s_2.pa = a[\overrightarrow{T}]$.

1. Base case: $a[\overrightarrow{T}] = (\mathsf{elem})$. There is no transition from $(\mathsf{elem}_\circ)$.
2. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$. We have $s_1 = (\mathsf{aut}_\circ, n_1, ss_1)$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:
   - Case aut_1: $s_2 = (\mathsf{aut}_\circ, n_2, init(\nu(n_2)[s_1.val]))$. By Lemma A.1, we have $init(\nu(n_2)[s_1.val]) \in \mathcal{Q}$. Thus, $s_2 \in \mathcal{Q}$.
   - Case aut_2: $s_2 = (\mathsf{aut}_\circ, n_1, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. By induction hypothesis and as $ss_1 \in \mathcal{Q}$, we have $ss_2 \in \mathcal{Q}$. Thus, $s_2 \in \mathcal{Q}$.
3. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:
   - Case $\star_1$: $s_1 = (\star_\circ, started?, ss_1)$, $s_2 = (\star_\circ, \text{true}, ss_2)$ and $init(b[s_1.val]) \xrightarrow{\sigma,\Gamma} ss_2$. By Lemma A.1, $init(b[s_1.val]) \in \mathcal{Q}$, and by induction hypothesis, $ss_2 \in \mathcal{Q}$. Thus, $s_2 \in \mathcal{Q}$.
   - Case $\star_2$: $s_1 = (\star_\circ, \text{true}, ss_1)$ and $s_2 = (\star_\circ, \text{true}, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. As $ss_1 \in \mathcal{Q}$ and by induction hypothesis, $ss_2 \in \mathcal{Q}$. Thus, $s_2 \in \mathcal{Q}$.
4. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:
   - Case $|_1$: $s_1 = (|_\circ, \bot, \bot)$ and $s_2 = (|_\circ, \mathsf{left}, ss_2)$ with $init(l[s_1.val]) \xrightarrow{\sigma,\Gamma} ss_2$. By Lemma A.1, $init(l[s_1.val]) \in \mathcal{Q}$. By induction hypothesis, $ss_2 \in \mathcal{Q}$. Thus, $s_2 \in \mathcal{Q}$.

- Case $|_2$: same as previous.
- Case $|_3$: $s_1 = (|_\circ, \mathsf{left}, ss_1)$ and $s_2 = (|_\circ, \mathsf{left}, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. We have $ss_1 \in \mathcal{Q}$. By induction hypothesis, $ss_2 \in \mathcal{Q}$. Thus, $s_2 \in \mathcal{Q}$.
- Case $|_4$: same as previous.

5. Inductive case: $a[\overrightarrow{T}] = (||, n, \Delta, l, r)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:
   - Case $||_1$: $s_1 = (||_\circ, s_{l1}, s_{r1})$ and $s_2 = (||_\circ, s_{l2}, s_{r1})$ with $\alpha(\sigma) \notin \Delta$ and $s_{l1} \xrightarrow{\sigma,\Gamma} s_{l2}$. We have $s_{l1} \in \mathcal{Q}$ and $s_{r1} \in \mathcal{Q}$. By induction hypothesis, $s_{l2} \in \mathcal{Q}$. Thus, $s_2 \in \mathcal{Q}$.
   - Case $||_2$: same.
   - Case $||_3$: similar proof. This time $\alpha(\sigma) \in \Delta$ and the induction hypothesis is used twice (for l and r).
6. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:
   - Case $|:_1$: $s_1 = (|:_\circ, \bot, \bot)$ and $s_2 = (|:_\circ, v, ss_2)$ with $init(b[s_1.val]) \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} ss_2$ and $v \in X$. By Lemma A.1, $init(b[s_1.val]) \in \mathcal{Q}$. By induction hypothesis, $ss_2 \in \mathcal{Q}$. Thus, $s_2 \in \mathcal{Q}$.
   - Case $|:_2$: $s_1 = (|:_\circ, v, ss_1)$ and $s_2 = (|:_\circ, v, ss_2)$ with $ss_1 \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} ss_2$ and $v \neq \bot$. We have $ss_1 \in \mathcal{Q}$. By induction hypothesis, $ss_2 \in \mathcal{Q}$. Thus, $s_2 \in \mathcal{Q}$.
7. Inductive case: $a[\overrightarrow{T}] = (|||:_:, n, x, X, b)[\overrightarrow{T}]$. By $|||:_1$, the only rule for $s_1 \xrightarrow{\sigma,\Gamma} s_2$, $s_1 = (|||:_\circ, f)$ and $s_2 = (|||:_\circ, f \lhd \{v \mapsto ss_2\})$ with $f(v) \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} ss_2$. We have $f(v) \in \mathcal{Q}$. By induction hypothesis, $ss_2 \in \mathcal{Q}$. Thus, $s_2 \in \mathcal{Q}$.

□

B.2 Proofs of Section A.4.1

Lemma A.3. Let $a[\overrightarrow{T}]$ be a PASTD. For any parameter values $\overrightarrow{V}$ and $\overrightarrow{V'}$ such that $\overrightarrow{V} \subseteq \overrightarrow{V'}$, $init(a[\overrightarrow{V}]) \sqsubseteq init(a[\overrightarrow{V'}])$.

B.2. Proofs of Section A.4.1

Proof. Let $a[\overrightarrow{T}]$ be a PASTD, $\overrightarrow{V}$ and $\overrightarrow{V'}$ such that $\overrightarrow{V} \subseteq \overrightarrow{V'}$. By structural induction on $a[\overrightarrow{T}]$.

1. Base case: $a[\overrightarrow{T}] = (\mathsf{elem})$. We have $init(a[\overrightarrow{V}]) = (\mathsf{elem}_\circ)$ and $init(a[\overrightarrow{V'}]) = (\mathsf{elem}_\circ)$. Thus, $init(a[\overrightarrow{V}]) \sqsubseteq init(a[\overrightarrow{V'}])$.

2. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$. By definition, $init(a[\overrightarrow{V}]) = (\mathsf{aut}_\circ, n_0, init(\nu(n_0)[\overrightarrow{V}]))$, $init(a[\overrightarrow{V'}]) = (\mathsf{aut}_\circ, n_0, init(\nu(n_0)[\overrightarrow{V'}]))$. By induction hypothesis, $init(\nu(n_0)[\overrightarrow{V}]) \sqsubseteq init(\nu(n_0)[\overrightarrow{V'}])$. Thus, $init(a[\overrightarrow{V}]) \sqsubseteq init(a[\overrightarrow{V'}])$, by definition of $\sqsubseteq$.

3. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. We have $init(a[\overrightarrow{V}]) = (\star_\circ, \text{false}, \bot)$. Likewise, $init(a[\overrightarrow{V'}]) = (\star_\circ, \text{false}, \bot)$. Thus, $init(a[\overrightarrow{V}]) \sqsubseteq init(a[\overrightarrow{V'}])$.

4. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. We have $init(a[\overrightarrow{V}]) = (|_\circ, \bot, \bot)$ and $init(a[\overrightarrow{V'}]) = (|_\circ, \bot, \bot)$.Thus, $init(a[\overrightarrow{V}]) \sqsubseteq init(a[\overrightarrow{V'}])$.

5. Inductive case: $a[\overrightarrow{T}] = (||, n, \Delta, l, r)[\overrightarrow{T}]$. We have $init(a[\overrightarrow{V}]) = (||_\circ, init(l[\overrightarrow{V}]), init(r[\overrightarrow{V}]))$ and $init(a[\overrightarrow{V'}]) = (||_\circ, init(l[\overrightarrow{V'}]), init(r[\overrightarrow{V'}]))$. By induction hypothesis, $init(l[\overrightarrow{V}]) \sqsubseteq init(l[\overrightarrow{V'}])$ and $init(r[\overrightarrow{V}]) \sqsubseteq init(r[\overrightarrow{V'}])$. Thus, $init(a[\overrightarrow{V}]) \sqsubseteq init(a[\overrightarrow{V'}])$.

6. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b)[\overrightarrow{T}]$. We have $init(a[\overrightarrow{V}]) = (|:_\circ, \bot, \bot)$. Likewise, $init(a[\overrightarrow{V'}]) = (|:_\circ, \bot, \bot)$. Thus, $init(a[\overrightarrow{V}]) \sqsubseteq init(a[\overrightarrow{V'}])$.

7. Inductive case: $a[\overrightarrow{T}] = (|||:, n, x, X, b)[\overrightarrow{T}]$. By cases on X.

   - If $X = T_i \in \overrightarrow{T}$. We have $init(a[\overrightarrow{V}]) = (|||:_\circ, V_i \times \{init(b[\overrightarrow{V}])\})$ and $init(a[\overrightarrow{V'}]) = (|||:_\circ, V'_i \times \{init(b[\overrightarrow{V'}])\})$. By induction hypothesis, we have $init(b[\overrightarrow{V}]) \sqsubseteq init(b[\overrightarrow{V'}])$. As $V_i \subseteq V'_i$, we conclude that $init(a[\overrightarrow{V}]) \sqsubseteq init(a[\overrightarrow{V'}])$.
   - If $X = W$ is not a parameter, we can conclude as well by induction hypothesis.

□

B.2. Proofs of Section A.4.1

Lemma A.4. For any IPASTD states $s, s' \in \mathcal{Q}$ such that $s \sqsubseteq s'$, if $\mathit{final}(s)$ then $\mathit{final}(s')$.

Proof. Let s, s' such that $s \sqsubseteq s'$. By structural induction on $s.pa$.

1. Base case: $s.pa = (\mathsf{elem})$. We have $s'.pa = (\mathsf{elem})$ and $s' = (\mathsf{elem}_\circ)$. Thus $\mathit{final}(s')$.

2. Inductive case: $s.pa = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$.
   We have $s = (\mathsf{aut}_\circ, n, ss)$, $s' = (\mathsf{aut}_\circ, n, ss')$ with $ss \sqsubseteq ss'$. Assume $\mathit{final}(s)$, *i.e.* by definition $(n \in DF \wedge \mathit{final}(ss)) \vee n \in SF$. By induction hypothesis, if $\mathit{final}(ss)$ then $\mathit{final}(ss')$. We can conclude that $(n \in DF \wedge \mathit{final}(ss')) \vee n \in SF$, *i.e.* $\mathit{final}(s')$ is true.

3. Inductive case: $s.pa = (\star, n, b)[\overrightarrow{T}]$. We have $s = (\star_\circ, \mathit{started}?, ss)$ and $s' = (\star_\circ, \mathit{started}?, ss')$. Assume $\mathit{final}(s)$, *i.e.* $\mathit{final}(ss) \vee \neg \mathit{started}?$. By induction hypothesis, we have $\mathit{final}(ss')$, as $ss \sqsubseteq ss'$. Thus, $\mathit{final}(s')$.

4. Inductive case: $s.pa = (|, n, l, r)[\overrightarrow{T}]$. By cases on s:

   - If $s = (|_\circ, \bot, \bot)$, then $s' = (|_\circ, \bot, \bot)$. Suppose that $\mathit{final}(s)$ is true, that is $\mathit{final}(\mathit{init}(l[s.val])) \vee \mathit{final}(\mathit{init}(r[s.val]))$.
     - Case $\mathit{final}(\mathit{init}(l[s.val]))$. By Lemma A.3, $\mathit{init}(l[s.val]) \sqsubseteq \mathit{init}(l[s'.val])$, because $s.val \subseteq s'.val$. So, by induction hypothesis, $\mathit{final}(\mathit{init}(l[s'.val]))$ is true. Thus, $\mathit{final}(s')$ is true by definition.
     - Same reasoning for $\mathit{final}(\mathit{init}(r[s.val]))$.
   - If $s = (|_\circ, \mathsf{left}, ss)$, then $s' = (|_\circ, \mathsf{left}, ss')$ with $ss \sqsubseteq ss'$. Assume $\mathit{final}(s)$, *i.e.* $\mathit{final}(ss)$. By induction hypothesis, $\mathit{final}(ss')$ is true, and is equivalent to $\mathit{final}(s')$ by definition.
   - Same for $s = (|_\circ, \mathsf{right}, ss)$.

5. Inductive case: $s.pa = (||, n, \Delta, l, r)[\overrightarrow{T}]$. We have $s = (||_\circ, s_l, s_r)$ and $s' = (||_\circ, s'_l, s'_r)$ with $s_l \sqsubseteq s'_l$ and $s_r \sqsubseteq s'_r$. Assume $\mathit{final}(s)$, *i.e.* $\mathit{final}(s_l) \wedge \mathit{final}(s_r)$. By induction hypothesis, $\mathit{final}(s'_l)$ and $\mathit{final}(s'_r)$. Thus, $\mathit{final}(s')$.

6. Inductive case: $s.pa = (|:, n, x, X, b)[\overrightarrow{T}]$. By cases on s:

   - If $s = (|:_\circ, \bot, \bot)$, then $s' = (|:_\circ, \bot, \bot)$.
     Assume *final*(s), *i.e.* *final*$(init(b[s.val]))$. By Lemma A.3 and induction hypothesis, *final*$(init(b[s'.val]))$. Thus, *final*(s').
   - If $s = (|:_\circ, v, ss)$, $s' = (|:_\circ, v, ss')$ with $ss \sqsubseteq ss'$. Assume *final*(s), *i.e.* *final*(ss). By induction hypothesis, *final*(ss'), *i.e.* *final*(s').

7. Inductive case: $s.pa = (|||:, n, x, X, b)[\overrightarrow{T}]$. We have $s = (|||:_\circ, f)$ and $s' = (|||:_\circ, f')$ with $dom(f) \subseteq dom(f')$ and for all $v \in dom(f)$, $f(v) \sqsubseteq f'(v)$. Suppose *final*(s). By Definition A.15 and by cases on X:

   - If X is a parameter, then there exists $v \in dom(f)$ such that *final*$(f(v))$. By induction hypothesis, *final*$(f'(v))$. Thus, *final*(s').
   - Else, $dom(f) = dom(f')$ and for all $v \in dom(f)$ we have *final*$(f(v))$. By induction hypothesis, for all $v \in dom(f)$, *final*$(f'(v))$. Thus, *final*(s').

□

Lemma A.5. For any IPASTD states $s_1, s'_1, s_2 \in \mathcal{Q}$, any event σ and any environment Γ, if $s_1 \sqsubseteq s'_1$ and $s_1 \xrightarrow{\sigma,\Gamma} s_2$, then there exists s'_2 such that $s_2 \sqsubseteq s'_2$ and $s'_1 \xrightarrow{\sigma,\Gamma} s'_2$.

Proof. Let s_1, s_2, s'_1 three states, σ an event and Γ an environment such that $s_1 \sqsubseteq s'_1$ and $s_1 \xrightarrow{\sigma,\Gamma} s_2$. By structural induction on $s_1.pa = s_2.pa = s'_1.pa = a[\overrightarrow{T}]$.

1. Base case: $a[\overrightarrow{T}] = (\mathsf{elem})$. There is no transition from $(\mathsf{elem}_\circ)$.
2. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$. We have $s_1 = (\mathsf{aut}_\circ, n_1, ss_1)$, $s'_1 = (\mathsf{aut}_\circ, n_1, ss'_1)$ with $ss_1 \sqsubseteq ss'_1$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case aut_1: $s_2 = (\mathsf{aut}_\circ, n_2, init(\nu(n_2)[s_1.val]))$ with $\delta(n_1, n_2, \sigma', final?)$ and $final? \Rightarrow$ *final*(ss_1) and $\sigma'[\Gamma] = \sigma$. By Lemma A.4, *final*$(ss_1) \Rightarrow$ *final*(ss'_1). And, by the same inference rule, we have $(\mathsf{aut}_\circ, n_1, ss'_1) \xrightarrow{\sigma,\Gamma} s'_2$ such that $s'_2 = (\mathsf{aut}_\circ, n_2, init(\nu(n_2)[s'_1.val]))$. Thus, $s_2 \sqsubseteq s'_2$ by Lemma A.3 and $s'_1 \xrightarrow{\sigma,\Gamma} s'_2$.

- Case aut_2: $s_2 = (\mathsf{aut}_\circ, n_1, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. By induction hypothesis, there exists ss_2' such that $ss_2 \sqsubseteq ss_2'$ and $ss_1' \xrightarrow{\sigma,\Gamma} ss_2'$. Let $s_2' = (\mathsf{aut}_\circ, n_1, ss_2')$. We have $s_2 \sqsubseteq s_2'$ by definition of $\sqsubseteq$ and $s_1' \xrightarrow{\sigma,\Gamma} s_2'$ by the rule aut_2.

3. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $\star_1$: $s_1 = (\star_\circ, \mathit{started}?, ss_1)$ and $s_2 = (\star_\circ, \mathrm{true}, ss_2)$ with $\mathit{final}(ss_1) \vee \neg\mathit{started}?$ and $\mathit{init}(b[s_1.\mathit{val}]) \xrightarrow{\sigma,\Gamma} ss_2$. We have $s_1' = (\star_\circ, \mathit{started}?, ss_1')$ with $ss_1 \sqsubseteq ss_1'$. If $\mathit{final}(ss_1)$ then $\mathit{final}(ss_1')$ by Lemma A.4. Thus, $\mathit{final}(ss_1') \vee \neg\mathit{started}?$. By Lemma A.3 and induction hypothesis, there exists ss_2' such that $ss_2 \sqsubseteq ss_2'$ and $\mathit{init}(b[s_1'.\mathit{val}]) \xrightarrow{\sigma,\Gamma} ss_2'$. Let $s_2' = (\star_\circ, \mathrm{true}, ss_2')$. We have $s_2 \sqsubseteq s_2'$ and $s_1' \xrightarrow{\sigma,\Gamma} s_2'$ by the rule $\star_1$.
   - Case $\star_2$: $s_1 = (\star_\circ, \mathrm{true}, ss_1)$ and $s_2 = (\star_\circ, \mathrm{true}, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. We have $s_1' = (\star_\circ, \mathrm{true}, ss_1')$ with $ss_1 \sqsubseteq ss_1'$. By induction hypothesis, there exists ss_2' such that $ss_2 \sqsubseteq ss_2'$ and $ss_1' \xrightarrow{\sigma,\Gamma} ss_2'$. Let $s_2' = (\star_\circ, \mathrm{true}, ss_2')$. We have $s_2 \sqsubseteq s_2'$ and $s_1' \xrightarrow{\sigma,\Gamma} s_2'$.

4. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $|_1$: $s_1 = (|_\circ, \bot, \bot)$ and $s_2 = (|_\circ, \mathsf{left}, ss_2)$ with $\mathit{init}(l[s_1.\mathit{val}]) \xrightarrow{\sigma,\Gamma} ss_2$. We have $s_1' = (|_\circ, \bot, \bot)$. And by Lemma A.3, $\mathit{init}(l[s_1.\mathit{val}]) \sqsubseteq \mathit{init}(l[s_1'.\mathit{val}])$ because $s_1 \sqsubseteq s_1'$. Thus, by induction hypothesis, there exists ss_2' such that $ss_2 \sqsubseteq ss_2'$ and $\mathit{init}(l[s_1'.\mathit{val}]) \xrightarrow{\sigma,\Gamma} ss_2'$. Let $s_2' = (|_\circ, \mathsf{left}, ss_2')$. We can conclude that $s_2 \sqsubseteq s_2'$ and $s_1' \xrightarrow{\sigma,\Gamma} s_2'$.
   - Case $|_2$: same as previous.
   - Case $|_3$: $s_1 = (|_\circ, \mathsf{left}, ss_1)$ and $s_2 = (|_\circ, \mathsf{left}, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. We have $s_1' = (|_\circ, \mathsf{left}, ss_1')$ with $ss_1 \sqsubseteq ss_1'$. By induction hypothesis, there exists ss_2' such that $ss_2 \sqsubseteq ss_2'$ and $ss_1' \xrightarrow{\sigma,\Gamma} ss_2'$. Let $s_2' = (|_\circ, \mathsf{left}, ss_2')$. Thus, $s_2 \sqsubseteq s_2'$ and $s_1' \xrightarrow{\sigma,\Gamma} s_2'$.
   - Case $|_4$: same as previous.

5. Inductive case: $a[\overrightarrow{T}] = (||, n, \Delta, l, r)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

- Case $||_1$: $s_1 = (||_\circ, s_{l1}, s_{r1})$ and $s_2 = (||_\circ, s_{l2}, s_{r1})$ with $\alpha(\sigma) \notin \Delta$ and $s_{l1} \xrightarrow{\sigma,\Gamma} s_{l2}$. We have $s'_1 = (||_\circ, s'_{l1}, s'_{r1})$ with $s_{l1} \sqsubseteq s'_{l1}$ and $s_{r1} \sqsubseteq s'_{r1}$. By induction hypothesis, there exists s'_{l2} such that $s_{l2} \sqsubseteq s'_{l2}$ and $s'_{l1} \xrightarrow{\sigma,\Gamma} s'_{l2}$. Let $s'_2 = (||_\circ, s'_{l2}, s'_{r1})$. We have $s'_1 \xrightarrow{\sigma,\Gamma} s'_2$ by the rule $||_1$ and $s_2 \sqsubseteq s'_2$.
- Case $||_2$: same.
- Case $||_3$: similar proof. This time $\alpha(\sigma) \in \Delta$ and the induction hypothesis is used twice (for l and r).

6. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:
   - Case $|:_1$: $s_1 = (|:_\circ, \bot, \bot)$ and $s_2 = (|:_\circ, v, ss_2)$ with $init(b[s_1.val]) \xrightarrow{\sigma,([x:=v])\triangleleft\Gamma} ss_2$ and $v \in X$. We have $s'_1 = (|:_\circ, \bot, \bot)$ with $s_1.val \subseteq s'_1.val$. And by Lemma A.3, $init(b[s_1.val]) \sqsubseteq init(b[s'_1.val])$. Then, by induction hypothesis, there exists ss'_2 such that $ss_2 \sqsubseteq ss'_2$ and $init(b[s'_1.val]) \xrightarrow{\sigma,([x:=v])\triangleleft\Gamma} ss'_2$. Let $s'_2 = (|:_\circ, v, ss'_2)$. We have $s_2 \sqsubseteq s'_2$. By cases on X:
     - If $X = V_i \in s_1.val$, then $v \in V'_i \supseteq V_i$ with $V_i \in s_1.val$ and $V'_i \in s'_1.val$. Thus by the rule $|:_1$, $s'_1 \xrightarrow{\sigma,\Gamma} s'_2$.
     - If $X = W \notin s_1.val$, then $v \in W$ holds and $s'_1 \xrightarrow{\sigma,\Gamma} s'_2$.
   - Case $|:_2$: $s_1 = (|:_\circ, v, ss_1)$ and $s_2 = (|:_\circ, v, ss_2)$ with $ss_1 \xrightarrow{\sigma,([x:=v])\triangleleft\Gamma} ss_2$ and $v \neq \bot$. We have $s'_1 = (|:_\circ, v, ss'_1)$ with $ss_1 \sqsubseteq ss'_1$. By induction hypothesis, there is ss'_2 such that $ss_2 \sqsubseteq ss'_2$ and $ss'_1 \xrightarrow{\sigma,([x:=v])\triangleleft\Gamma} ss'_2$. Let $s'_2 = (|:_\circ, v, ss'_2)$. We have $s_2 \sqsubseteq s'_2$ and $s'_1 \xrightarrow{\sigma,\Gamma} s'_2$.

7. Inductive case: $a[\overrightarrow{T}] = (|||:_\cdot, n, x, X, b)[\overrightarrow{T}]$. By $|||:_1$, the only rule for $s_1 \xrightarrow{\sigma,\Gamma} s_2$, $s_1 = (|||:_\circ, f)$ and $s_2 = (|||:_\circ, f \lhd \{v \mapsto ss_2\})$ with $f(v) \xrightarrow{\sigma,([x:=v])\triangleleft\Gamma} ss_2$. We have $s'_1 = (|||:_\circ, f')$ with $dom(f) \subseteq dom(f')$ and for all $w \in dom(f)$, $f(w) \sqsubseteq f'(w)$. As $f(v) \sqsubseteq f'(v)$, we use induction hypothesis to conclude that there exists ss'_2 such that $ss_2 \sqsubseteq ss'_2$ and $f'(v) \xrightarrow{\sigma,([x:=v])\triangleleft\Gamma} ss'_2$. By the rule $|||:_1$, we have $(|||:_\circ, f') \xrightarrow{\sigma,\Gamma} (|||:_\circ, f' \lhd \{v \mapsto ss'_2\})$. And by definition of $\sqsubseteq$, $(|||:_\circ, f \lhd \{v \mapsto ss_2\}) \sqsubseteq (|||:_\circ, f' \lhd \{v \mapsto ss'_2\})$.

□

B.3 Proofs of Section A.4.2

Lemma A.6. Let $a[\overrightarrow{T}]$ be a PASTD. For any parameter value $\overrightarrow{V}$ and any rename function ξ defined by permutations $\rho_1, \ldots, \rho_k$, we have

$$\xi(\mathit{init}(a[\overrightarrow{V}])) = \mathit{init}(a[\rho_1(V_1), \ldots, \rho_k(V_k)])$$

Proof. Let $a[\overrightarrow{T}]$ be a PASTD, $\overrightarrow{V}$ a parameter value and ξ a rename function given by permutations $\rho_1, \ldots, \rho_k$. For $\overrightarrow{V} = (V_1, \ldots, V_k)$, let $\rho(\overrightarrow{V}) = (\rho_1(V_1), \ldots, \rho_k(V_k))$. By structural induction on $a[\overrightarrow{T}]$.

1. Base case: $a[\overrightarrow{T}] = (\mathsf{elem})$. We have $\mathit{init}(a[\overrightarrow{V}]) = (\mathsf{elem}_\circ)$. Thus, $\xi(\mathit{init}(a[\overrightarrow{V}])) = (\mathsf{elem}_\circ)$ such that $\xi(\mathit{init}(a[\overrightarrow{V}])).\mathit{val} = \rho(\overrightarrow{V})$ and $\mathit{init}(a[\rho(\overrightarrow{V})]) = (\mathsf{elem}_\circ)$ such that $\mathit{init}(a[\rho(\overrightarrow{V})]).\mathit{val} = \rho(\overrightarrow{V})$.

2. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, \mathit{name}, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$.
   By definition, w have $\mathit{init}(a[\overrightarrow{V}]) = (\mathsf{aut}_\circ, n_0, \mathit{init}(\nu(n_0)[\overrightarrow{V}]))$ and $\xi(\mathit{init}(a[\overrightarrow{V}])) = (\mathsf{aut}_\circ, n_0, \xi(\mathit{init}(\nu(n_0)[\overrightarrow{V}])))$. Moreover, $\mathit{init}(a[\rho(\overrightarrow{V})]) = (\mathsf{aut}_\circ, n_0, \mathit{init}(\nu(n_0)[\rho(\overrightarrow{V})]))$. By the induction hypothesis, $(\mathsf{aut}_\circ, n_0, \xi(\mathit{init}(\nu(n_0)[\overrightarrow{V}]))) = (\mathsf{aut}_\circ, n_0, \mathit{init}(\nu(n_0)[\rho(\overrightarrow{V})]))$. Finally, we have $\xi(\mathit{init}(a[\overrightarrow{V}])) = \mathit{init}(a[\rho(\overrightarrow{V})])$.

3. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. We have $\mathit{init}(a[\overrightarrow{V}]) = (\star_\circ, \mathrm{false}, \bot)$. Moreover, $\xi(\mathit{init}(a[\overrightarrow{V}])) = (\star_\circ, \mathrm{false}, \bot)$ with $\xi(\mathit{init}(a[\overrightarrow{V}])).\mathit{val} = \rho(\overrightarrow{V})$. Thus, $\xi(\mathit{init}(a[\overrightarrow{V}])) = \mathit{init}(a[\rho(\overrightarrow{V})])$.

4. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. We have $\mathit{init}(a[\overrightarrow{V}]) = (|_\circ, \bot, \bot)$. Moreover, $\xi(\mathit{init}(a[\overrightarrow{V}])) = (|_\circ, \bot, \bot)$ with $\xi(\mathit{init}(a[\overrightarrow{V}])).\mathit{val} = \rho(\overrightarrow{V})$. Thus, $\xi(\mathit{init}(a[\overrightarrow{V}])) = \mathit{init}(a[\rho(\overrightarrow{V})])$.

5. Inductive case: $a[\overrightarrow{T}] = (||, n, \Delta, l, r)[\overrightarrow{T}]$. We have $\mathit{init}(a[\overrightarrow{V}]) = (||_\circ, \mathit{init}(l[\overrightarrow{V}]), \mathit{init}(r[\overrightarrow{V}]))$ and $\xi(\mathit{init}(a[\overrightarrow{V}])) = (||_\circ, \xi(\mathit{init}(l[\overrightarrow{V}])), \xi(\mathit{init}(r[\overrightarrow{V}])))$. By induction hypothesis, we have that $\xi(\mathit{init}(a[\overrightarrow{V}])) = (||_\circ, \mathit{init}(l[\rho(\overrightarrow{V})]), \mathit{init}(r[\rho(\overrightarrow{V})]))$.

And as $init(a[\rho(\overrightarrow{V})]) = (||_\circ, init(l[\rho(\overrightarrow{V})]), init(r[\rho(\overrightarrow{V})]))$, then $\xi(init(a[\overrightarrow{V}])) = init(a[\rho(\overrightarrow{V})])$.

6. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b)[\overrightarrow{T}]$. We have $init(a[\overrightarrow{V}]) = (|:_\circ, \bot, \bot)$. Moreover, $\xi(init(a[\overrightarrow{V}])) = (|:_\circ, \bot, \bot)$ with $\xi(init(a[\overrightarrow{V}])).val = \rho(\overrightarrow{V})$. Thus, $\xi(init(a[\overrightarrow{V}])) = init(a[\rho(\overrightarrow{V})])$.

7. Inductive case: $a[\overrightarrow{T}] = (|||:, n, x, X, b)[\overrightarrow{T}]$. By cases on X.

   - If $X = T_i \in \overrightarrow{T}$. We have $init(a[\overrightarrow{V}]) = (|||:_\circ, V_i \times \{init(b[\overrightarrow{V}])\})$ and $\xi(init(a[\overrightarrow{V}])) = (|||:_\circ, \rho_i(V_i) \times \{\xi(init(b[\overrightarrow{V}]))\})$. By induction hypothesis, we have $\xi(init(b[\overrightarrow{V}])) = init(b[\rho(\overrightarrow{V})])$. Thus, $\xi(init(a[\overrightarrow{V}])) = init(a[\rho(\overrightarrow{V})])$.
   - If $X = W$ is not a parameter, we can conclude as well by induction hypothesis.

□

Lemma A.7. For any $s \in \mathcal{Q}$ and for any rename function ξ, if $final(s)$, then $final(\xi(s))$.

Proof. Let s be an IPASTD state and ξ a rename function given by permutations $\rho_1, \ldots, \rho_k$. For a parameter value $\overrightarrow{V} = (V_1, \ldots, V_k)$, let $\rho(\overrightarrow{V}) = (\rho_1(V_1), \ldots, \rho_k(V_k))$. By structural induction on $s.pa$.

1. Base case: $s.pa = (\mathsf{elem})$. We have $\xi(s) = (\mathsf{elem}_\circ)$ and $final(\xi(s))$.

2. Inductive case: $s.pa = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$.
   We have $s = (\mathsf{aut}_\circ, n, ss)$. Assume $final(s)$, *i.e.* $(n \in DF \wedge final(ss)) \vee n \in SF$. By induction hypothesis, $final(\xi(ss))$. We have $\xi(s) = (\mathsf{aut}_\circ, n, \xi(ss))$. Thus, we conclude $final(\xi(s))$ by definition of final.

3. Inductive case: $s.pa = (\star, n, b)[\overrightarrow{T}]$. We have $s = (\star_\circ, started?, ss)$ and $\xi(s) = (\star_\circ, started?, \xi(ss))$. Assume $final(s)$, *i.e.* $final(ss) \vee \neg started?$. By induction hypothesis, we have $final(\xi(ss))$. Thus, $final(\xi(s))$.

4. Inductive case: $s.pa = (|, n, l, r)[\overrightarrow{T}]$. By cases on s:

- If $s = (|_\circ, \bot, \bot)$, then $\xi(s) = (|_\circ, \bot, \bot)$ with $\xi(s).val = \rho(s.val)$. Suppose *final*(s), *i.e.* *final*$(init(l[s.val])) \vee$ *final*$(init(r[s.val]))$.
  - Case *final*$(init(l[s.val]))$.
    By induction hypothesis, *final*$(\xi(init(l[s.val])))$. So by Lemma A.6, *final*$(init(l[\rho(s.val)]))$. Thus, *final*$(\xi(s))$ is true by definition.
  - Same reasoning for *final*$(init(r[s.val]))$.
- If $s = (|_\circ, \mathsf{left}, ss)$, then $\xi(s) = (|_\circ, \mathsf{left}, \xi(ss))$. Assume *final*$(s)$, *i.e.* *final*$(ss)$. By induction hypothesis, *final*$(\xi(ss))$ is true, and is equivalent to *final*$(\xi(s))$ by definition.
- Same for $s = (|_\circ, \mathsf{right}, ss)$.

5. Inductive case: $s.pa = (||, n, \Delta, l, r)[\overrightarrow{T}]$. We have $s = (||_\circ, s_l, s_r)$ and $\xi(s) = (||_\circ, \xi(s_l), \xi(s_r))$. Assume *final*$(s)$, *i.e.* *final*$(s_l) \wedge$ *final*(s_r). By induction hypothesis, *final*$(\xi(s_l))$ and *final*$(\xi(s_r))$. Thus, *final*$(\xi(s))$.

6. Inductive case: $s.pa = (|:, n, x, X, b)[\overrightarrow{T}]$. By cases on s:

   - If $s = (|:_\circ, \bot, \bot)$, then $\xi(s) = (|:_\circ, \bot, \bot)$ with $\xi(s).val = \rho(s.val)$. Assume *final*(s), *i.e.* *final*$(init(b[s.val]))$. Thus, *final*$(init(b[s.val]))$. By Lemma A.6 and induction hypothesis, we conclude *final*$(init(b[\rho(s.val)]))$.
     Thus, *final*$(\xi(s))$.
   - If $s = (|:_\circ, v, ss)$, then $\xi(s) = (|:_\circ, \_, \xi(ss))$. Assume *final*$(s)$, *i.e.* *final*$(ss)$. By induction hypothesis, *final*$(\xi(ss))$, *i.e.* *final*$(\xi(s))$.

7. Inductive case: $s.pa = (|||:, n, x, X, b)[\overrightarrow{T}]$. We have $s = (|||:_\circ, f)$ and $\xi(s) = (|||:_\circ, f')$. By cases on X:

   - If $X = T_i$ is a parameter, then $dom(f') = \rho_i(dom(f))$ and for all $v \in dom(f)$ we have $f'(\rho_i(v)) = \xi(f(v))$. Suppose *final*$(s)$. By Definition A.15, there exists $v \in dom(f)$ such that *final*$(f(v))$. By induction hypothesis, *final*$(f'(\rho_i(v)))$. Thus, *final*$(\xi(s))$.
   - Else, $dom(f) = dom(f')$ and for all $v \in dom(f)$ we have $f'(v) = \xi(f(v))$. Suppose *final*(s). By Definition A.15, for all $v \in dom(f)$ we have *final*$(f(v))$. By induction hypothesis, *final*$(f'(v))$. Thus, *final*$(\xi(s))$.

□

Lemma A.8. Let $P_1, \ldots, P_k$ a set of parameter domains. For any states $s_1, s_2 \in \mathcal{Q}$, any rename function ξ defined by permutations $\rho_1, \ldots, \rho_k$ on $P_1, \ldots, P_k$ respectively, any event σ and any environment $\Gamma = (\!|x_1, \ldots, x_n := v_1, \ldots, v_n|\!)$, if $s_1 \xrightarrow{\sigma,\Gamma} s_2$, then $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$, where $\sigma' = \sigma[v_1, \ldots, v_n := v'_1, \ldots, v'_n]$ and $\Gamma' = (\!|x_1, \ldots, x_n := v'_1, \ldots, v'_n|\!)$ with $v'_i = \rho_j(v_i)$ if $\exists j \cdot v_i \in P_j$ or $v'_i = v_i$ else.

Proof. Let $P_1, \ldots, P_k$ a set of parameter domains. Let $s_1, s_2 \in \mathcal{Q}$, ξ defined by permutations $\rho_1, \ldots, \rho_k$ on $P_1, \ldots, P_k$, σ an event and $\Gamma = (\!|x_1, \ldots, x_n := v_1, \ldots, v_n|\!)$ an environment, such that $s_1 \xrightarrow{\sigma,\Gamma} s_2$. Let $\sigma' = \sigma[v_1, \ldots, v_n := v'_1, \ldots, v'_n]$ and $\Gamma' = (\!|x_1, \ldots, x_n := v'_1, \ldots, v'_n|\!)$ with $v'_i = \rho_j(v_i)$ if $\exists j \cdot v_i \in P_j$ or $v'_i = v_i$ else. Let $\overrightarrow{V} = s_1.val = s_2.val$ and $\overrightarrow{V'} = \rho(\overrightarrow{V})$. By structural induction on $s_1.pa = s_2.pa = a[\overrightarrow{T}]$.

1. Base case: $a[\overrightarrow{T}] = (\mathsf{elem})$. There is no transition from $(\mathsf{elem}_\circ)$.
2. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$, $s_1 = (\mathsf{aut}_\circ, n_1, ss_1)$ and $\xi(s_1) = (\mathsf{aut}_\circ, n_1, \xi(ss_1))$ with $\xi(s_1).val = \overrightarrow{V'}$. By cases on inference rule for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:
   - Case aut_1: $s_2 = (\mathsf{aut}_\circ, n_2, \mathit{init}(\nu(n_2)[\overrightarrow{V}]))$ with $\delta(n_1, n_2, \sigma'', \mathit{final}?)$ and $\mathit{final}? \Rightarrow \mathit{final}(ss_1)$ and $\sigma''[\Gamma] = \sigma$.
     We have $\xi(s_2) = (\mathsf{aut}_\circ, n_2, \xi(\mathit{init}(\nu(n_2)[\overrightarrow{V}])))$ and
     $\xi(s_2) = (\mathsf{aut}_\circ, n_2, \mathit{init}(\nu(n_2)[\overrightarrow{V'}]))$ by Lemma A.6, with $\xi(s_2).val = \overrightarrow{V'}$. Besides, by Lemma A.7, $\mathit{final}(ss_1) \Rightarrow \mathit{final}(\xi(ss_1))$. Moreover, $\sigma''[\Gamma'] = \sigma[v_1, \ldots, v_n := v'_1, \ldots, v'_n] = \sigma'$. Thus, by the same inference rule, we have $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$.
   - Case aut_2: $s_2 = (\mathsf{aut}_\circ, n_1, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. By induction hypothesis, we have $\xi(ss_1) \xrightarrow{\sigma',\Gamma'} \xi(ss_2)$. Thus, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$ by the rule aut_2.
3. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. By cases on inference rule for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:
   - Case $\star_1$: $s_1 = (\star_\circ, \mathit{started}?, ss_1)$ and $s_2 = (\star_\circ, \mathrm{true}, ss_2)$ with $\mathit{final}(ss_1) \vee \neg \mathit{started}?$ and $\mathit{init}(b[\overrightarrow{V}]) \xrightarrow{\sigma,\Gamma} ss_2$. We have $\xi(s_1) = (\star_\circ, \mathit{started}?, \xi(ss_1))$. If

final(ss_1) then *final*$(\xi(ss_1))$ by Lemma A.7. Thus, *final*$(\xi(ss_1))\vee\neg$*started*?. By Lemma A.6 and induction hypothesis, $\mathit{init}(b[\overrightarrow{V'}]) \xrightarrow{\sigma',\Gamma'} \xi(ss_2)$. We have $\xi(s_2) = (\star_\circ, \text{true}, \xi(ss_2))$. Thus, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$ by the rule $\star_1$.

- Case $\star_2$: $s_1 = (\star_\circ, \text{true}, ss_1)$ and $s_2 = (\star_\circ, \text{true}, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. We have $\xi(s_1) = (\star_\circ, \text{true}, \xi(ss_1))$. By induction hypothesis, $\xi(ss_1) \xrightarrow{\sigma',\Gamma'} \xi(ss_2)$. We have $\xi(s_2) = (\star_\circ, \text{true}, \xi(ss_2))$. Thus, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$.

4. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $|_1$: $s_1 = (|_\circ, \bot, \bot)$ and $s_2 = (|_\circ, \mathsf{left}, ss_2)$ with $\mathit{init}(l[\overrightarrow{V}]) \xrightarrow{\sigma,\Gamma} ss_2$. We have $\xi(s_1) = (|_\circ, \bot, \bot)$. By Lemma A.6, $\xi(\mathit{init}(l[\overrightarrow{V}])) = \mathit{init}(l[\overrightarrow{V'}])$. Thus, by induction hypothesis, $\mathit{init}(l[\overrightarrow{V'}]) \xrightarrow{\sigma',\Gamma'} \xi(ss_2)$. We have $\xi(s_2) = (|_\circ, \mathsf{left}, \xi(ss_2))$. We can conclude that $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$.
   - Case $|_2$: same as previous.
   - Case $|_3$: $s_1 = (|_\circ, \mathsf{left}, ss_1)$ and $s_2 = (|_\circ, \mathsf{left}, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. We have $\xi(s_1) = (|_\circ, \mathsf{left}, \xi(ss_1))$ and $\xi(s_2) = (|_\circ, \mathsf{left}, \xi(ss_2))$. By induction hypothesis, $\xi(ss_1) \xrightarrow{\sigma',\Gamma'} \xi(ss_2)$. Thus, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$.
   - Case $|_4$: same as previous.

5. Inductive case: $a[\overrightarrow{T}] = (||, n, \Delta, l, r)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $||_1$: $s_1 = (||_\circ, s_{l1}, s_{r1})$ and $s_2 = (||_\circ, s_{l2}, s_{r1})$ with $\alpha(\sigma) \notin \Delta$ and $s_{l1} \xrightarrow{\sigma,\Gamma} s_{l2}$. We have $\xi(s_1) = (||_\circ, \xi(s_{l1}), \xi(s_{r1}))$ and $\xi(s_2) = (||_\circ, \xi(s_{l2}), \xi(s_{r1}))$. By induction hypothesis, $\xi(s_{l1}) \xrightarrow{\sigma',\Gamma'} \xi(s_{l2})$. Thus, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$ by the rule $||_1$.
   - Case $||_2$: same.
   - Case $||_3$: similar proof. This time $\alpha(\sigma) \in \Delta$ and the induction hypothesis is used twice (for l and r).

6. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

- Case $|:_1$: $s_1 = (|:_\circ, \bot, \bot)$ and $s_2 = (|:_\circ, v, ss_2)$ with $init(b[\overrightarrow{V}]) \xrightarrow{\sigma,(\![x:=v]\!)\lhd\Gamma} ss_2$ and $v \in X$. We have $\xi(s_1) = (|:_\circ, \bot, \bot)$ with $\xi(s_1).val = \overrightarrow{V'}$. By Lemma A.6, $\xi(init(b[\overrightarrow{V}])) = init(b[\overrightarrow{V'}])$. By cases on X:
  - If $X = V_i \in \overrightarrow{V}$. Let $v' = \rho_i(v)$. Then $v' \in V'_i = \rho_i(V_i)$ with $V'_i \in \overrightarrow{V'}$. By induction hypothesis, $init(b[\overrightarrow{V'}]) \xrightarrow{\sigma',(\![x:=v']\!)\lhd\Gamma'} \xi(ss_2)$. We have $\xi(s_2) = (|:_\circ, v', \xi(ss_2))$. Thus by the rule $|:_1$, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$.
  - If $X = W \notin \overrightarrow{V}$, then we still have $v \in W$. By induction hypothesis, $init(b[\overrightarrow{V'}]) \xrightarrow{\sigma',(\![x:=v]\!)\lhd\Gamma'} \xi(ss_2)$. We have $\xi(s_2) = (|:_\circ, v, \xi(ss_2))$. Thus by the rule $|:_1$, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$.
- Case $|:_2$: $s_1 = (|:_\circ, v, ss_1)$ and $s_2 = (|:_\circ, v, ss_2)$ with $ss_1 \xrightarrow{\sigma,(\![x:=v]\!)\lhd\Gamma} ss_2$ and $v \neq \bot$. By cases on$X$:
  - If $X = V_i \in \overrightarrow{V}$. Let $v' = \rho_i(v)$. We have $\xi(s_1) = (|:_\circ, v', \xi(ss_1))$ and $\xi(s_2) = (|:_\circ, v', \xi(ss_2))$. By induction hypothesis, $\xi(ss_1) \xrightarrow{\sigma',(\![x:=v']\!)\lhd\Gamma'} \xi(ss_2)$. Thus, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$.
  - If $X = W \notin \overrightarrow{V}$.
    We have $\xi(s_1) = (|:_\circ, v, \xi(ss_1))$ and $\xi(s_2) = (|:_\circ, v, \xi(ss_2))$. By induction hypothesis, $\xi(ss_1) \xrightarrow{\sigma',(\![x:=v]\!)\lhd\Gamma'} \xi(ss_2)$. Thus, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$.

7. Inductive case: $a[\overrightarrow{T}] = (|||:, n, x, X, b)[\overrightarrow{T}]$. By $|||:_1$, the only rule for $s_1 \xrightarrow{\sigma,\Gamma} s_2$, $s_1 = (|||:_\circ, f)$ and $s_2 = (|||:_\circ, f \lhd \{v \mapsto ss_2\})$ with $f(v) \xrightarrow{\sigma,(\![x:=v]\!)\lhd\Gamma} ss_2$. By cases on X:
   - If $X = V_k \in \overrightarrow{V}$. Let $v' = \rho_k(v)$. We have $\xi(s_1) = (|||:_\circ, f')$ with $dom(f') = \rho_k(dom(f))$ and for all $w \in dom(f)$, $f'(\rho_k(w)) = \xi(f(w))$. By induction hypothesis, $\xi(f(v)) = f'(v') \xrightarrow{\sigma',(\![x:=v']\!)\lhd\Gamma'} \xi(ss_2)$. By the rule $|||:_1$, we have $(|||:_\circ, f') \xrightarrow{\sigma',\Gamma'} (|||:_\circ, f' \lhd \{v' \mapsto \xi(ss_2)\})$. We have $\xi(s_2) = (|||:_\circ, f' \lhd \{v' \mapsto \xi(ss_2)\})$. Thus, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$.
   - If $X = W \notin \overrightarrow{V}$. We have $\xi(s_1) = (|||:_\circ, f')$ with $dom(f') = dom(f)$ and for all $w \in dom(f)$, $f'(w) = \xi(f(w))$. By induction hypothesis, $\xi(f(v)) = f'(v) \xrightarrow{\sigma',(\![x:=v]\!)\lhd\Gamma'} \xi(ss_2)$. By the rule $|||:_1$, we have $(|||:_\circ, f') \xrightarrow{\sigma',\Gamma'} (|||:_\circ, f' \lhd \{v \mapsto \xi(ss_2)\})$. We have $\xi(s_2) = (|||:_\circ, f' \lhd \{v \mapsto \xi(ss_2)\})$. Thus, $\xi(s_1) \xrightarrow{\sigma',\Gamma'} \xi(s_2)$.

□

B.4 Proofs of Section A.5

Lemma B.1. *Let $a[\overrightarrow{T}]$ be a PASTD. For any parameter values $\overrightarrow{V}$ and $\overrightarrow{V'}$ such that $\gamma(\overrightarrow{V}) \leq_k \gamma(\overrightarrow{V'})$, $init(a[\overrightarrow{V}]) \preceq init(a[\overrightarrow{V'}])$.*

Proof. By Lemmas A.3 and A.6. □

Lemma B.2. *For any IPASTD states $s, s' \in \mathcal{Q}^\sharp$ such that $s \sqsubseteq s'$, if $final(s)$ then $final(s')$.*

Proof. Same proof as for Lemma A.4. □

Lemma B.3. *For any IPASTD states $s, s' \in \mathcal{Q}^\sharp$ such that $s \preceq s'$, if $final(s)$ then $final(s')$.*

Proof. By Lemmas B.2 and A.7 (which are easily generalized to $\mathcal{Q}^\sharp$). □

Lemma B.4. *Let $a[\overrightarrow{T}]$ be a PASTD and $\overrightarrow{V}$ a parameter value. Let $q \in Abinit(a[\overrightarrow{V}])$. For all value $\overrightarrow{V'}$ such that $|\overrightarrow{V}| \leq_k |\overrightarrow{V'}|$, there exists $q' \in \mathcal{Q}^\sharp$ such that $q \preceq q'$ and $q' \in Abinit(a[\overrightarrow{V'}])$.*

Proof. Take $q' \in \mathcal{Q}^\sharp$ such that q' has the same state expression (modulo renaming) and same PASTD as q but with $q'.val = \overrightarrow{V'}$. This is possible by definition of $\mathcal{Q}^\sharp$, which allows the domain of the function of the quantified interleaving operator to be partial. □

Lemma B.5. *For any $s_1, s_1', s_2 \in \mathcal{Q}^\sharp$, any event σ and any environment $\Gamma = (\![x_1, \ldots, x_n := v_1, \ldots, v_n]\!)$, if $s_1 \preceq s_1'$ and $s_1 \xrightarrow{\sigma,\Gamma} s_2$, then there exists s_2' such that $s_2 \preceq s_2'$ and $s_1' \xrightarrow{\sigma',\Gamma'} s_2'$ such that ξ is the rename function defined by $\rho_1, \ldots, \rho_k$ and $s_1 \sqsubseteq \xi(s_1')$ and $s_2 \sqsubseteq \xi(s_2')$, where $\sigma' = \sigma[v_1, \ldots, v_n := v_1', \ldots, v_n']$ and $\Gamma' = (\![x_1, \ldots, x_n := v_1', \ldots, v_n']\!)$ with $v_i' = \rho_j(v_i)$ if $\exists j \cdot v_i \in P_j$ or $v_i' = v_i$ else.*

Proof. The proof is done by structural induction on $s_1.pa = s_2.pa = s_1'.pa = a[\overrightarrow{T}]$ and is similar to the one for Lemma A.8 (modulo renaming). We detail only the new cases defined in Definition A.20.

B.4. Proofs of Section A.5

1. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case aut_{1a}: $s_1 = (\mathsf{aut}_\circ, n_1, ss_1)$ and $s_2 = (\mathsf{aut}_\circ, n_2, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $\delta(n_1, n_2, \sigma'', final?)$, $final? \Rightarrow final(ss_1)$, $\sigma''[\Gamma] = \sigma$ and $ss_2 \in Abinit(\nu(n_2)[\overrightarrow{V}])$. Let $s_1'.val = \overrightarrow{V'}$. We have $s_1' = (\mathsf{aut}_\circ, n_1, ss_1')$ with $ss_1 \sqsubseteq \xi(ss_1')$. Take $s_2' = (\mathsf{aut}_\circ, n_2, init(\nu(n_2)[\overrightarrow{V'}]))$. By Definition A.19, we have $ss_2 \sqsubseteq init(\nu(n_2)[\overrightarrow{V}])$ and, by Lemma B.1, $init(\nu(n_2)[\overrightarrow{V}]) \preceq init(\nu(n_2)[\overrightarrow{V'}])$. Thus, $ss_2 \preceq ss_2'$ and $s_2 \preceq s_2'$. By Lemma B.3, $final? \Rightarrow final(ss_1')$. Moreover, $\sigma''[\Gamma'] = \sigma[v_1, \ldots, v_n := v_1', \ldots, v_n'] = \sigma'$. Consequently, by the inference rule aut_{1a}, $s_1' \xrightarrow{\sigma',\Gamma'} s_2'$.

2. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $\star_{1a}$: $s_1 = (\star_\circ, started?, ss_1)$ and $s_2 = (\star_\circ, \mathrm{true}, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $final(ss_1) \vee \neg started?$, $q \xrightarrow{\sigma,\Gamma} ss_2$ and $q \in Abinit(b[\overrightarrow{V}])$. Let $s_1' \in \mathcal{Q}^\sharp$ such that $s_1 \preceq s_1'$ with $s_1'.val = \overrightarrow{V'}$. We have $s_1' = (\star_\circ, started?, ss_1')$ with $ss_1 \preceq ss_1'$. Take $q' \in \mathcal{Q}^\sharp$ such that $q \preceq q'$ and $q' \in Abinit(b[\overrightarrow{V'}])$ thanks to Lemma B.4. By induction hypothesis, there exists $ss_2' \in \mathcal{Q}^\sharp$ such that $ss_2 \preceq ss_2'$ and $q' \xrightarrow{\sigma',\Gamma'} ss_2'$. Let $s_2' = (\star_\circ, true, ss_2')$. We have $s_2 \preceq s_2'$. By Lemma B.3, $final(ss_1') \vee \neg started?$. Thus, by the inference rule $\star_{1a}$, $s_1' \xrightarrow{\sigma',\Gamma'} s_2'$.

3. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $|_{1a}$: $s_1 = (|_\circ, \bot, \bot)$ and $s_2 = (|_\circ, \mathsf{left}, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $q \xrightarrow{\sigma,\Gamma} ss_2$ and $q \in Abinit(l[\overrightarrow{V}])$. Let $s_1' \in \mathcal{Q}^\sharp$ such that $s_1 \preceq s_1'$ with $s_1'.val = \overrightarrow{V'}$. We have $s_1' = (|_\circ, \bot, \bot)$. Take $q' \in \mathcal{Q}^\sharp$ such that $q \preceq q'$ and $q' \in Abinit(l[\overrightarrow{V'}])$ thanks to Lemma B.4. By induction hypothesis, there exists $ss_2' \in \mathcal{Q}^\sharp$ such that $ss_2 \preceq ss_2'$ and $q' \xrightarrow{\sigma',\Gamma'} ss_2'$. Let $s_2' = (|_\circ, \mathsf{left}, ss_2')$. We have $s_2 \preceq s_2'$. Finally, by the inference rule $\star_{1a}$, $s_1' \xrightarrow{\sigma',\Gamma'} s_2'$.
   - Case $|_{2a}$: same as previous.

4. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

- Case $|:_{1a}$: $s_1 = (|:_\circ, \bot, \bot)$ and $s_2 = (|:_\circ, v, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $q \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} ss_2$, $q \in \mathit{Abinit}(b[\overrightarrow{V}])$ and $v \in X$. Let $s'_1 \in \mathcal{Q}^\sharp$ such that $s_1 \preceq s'_1$ with $s'_1.val = \overrightarrow{V'}$. We have $s'_1 = (|:_\circ, \bot, \bot)$. Take $q' \in \mathcal{Q}^\sharp$ such that $q \preceq q'$ and $q' \in \mathit{Abinit}(b[\overrightarrow{V'}])$ thanks to Lemma B.4. By cases on X:
  - If $X = T_i$, then $v \in V_i$. Let $v' = \rho_i^{-1}(v)$. By induction hypothesis, there exists $ss'_2 \in \mathcal{Q}^\sharp$ such that $ss_2 \preceq ss'_2$ and $q' \xrightarrow{\sigma',(\![x:=v']\!)\triangleleft\Gamma'} ss'_2$. Let $s'_2 = (|:_\circ, v', ss'_2)$. Then, $s_2 \preceq s'_2$. Moreover, $v' \in \rho_i^{-1}(V_i) \subseteq V'_i$. Thus, by the rule $|:_{1a}$, $s'_1 \xrightarrow{\sigma',\Gamma'} s'_2$.
  - If $X = W \notin \overrightarrow{T}$, then $v \in W$. By induction hypothesis, there exists $ss'_2 \in \mathcal{Q}^\sharp$ such that $ss_2 \preceq ss'_2$ and $q' \xrightarrow{\sigma',(\![x:=v]\!)\triangleleft\Gamma'} ss'_2$. Let $s'_2 = (|:_\circ, v, ss'_2)$. We have $s_2 \preceq s'_2$ and $s'_1 \xrightarrow{\sigma',\Gamma'} s'_2$.

□

Theorem A.3. Let $a[\overrightarrow{T}]$ be a PASTD without free variables in events, with $\mathcal{S}_a^\sharp$ the corresponding TS from Definition A.22. $\mathcal{S}_a^\sharp$ is monotone wrt $\sqsubseteq$ and $\preceq$.

Proof. Let s_1, s'_1, s_2 states of $\mathcal{S}_a^\sharp$ such that $s_1 \preceq s'_1$ and $s_1 \to s_2$. There is an event σ such that $s_1 \xrightarrow{\sigma} s_2$. By the only inference rule env, we have $s_1 \xrightarrow{\sigma,(\!|\!|)} s_2$. Then, by Lemma B.5 there exists $s'_2 \in \mathcal{Q}^\sharp$ such that $s_2 \preceq s'_2$ and $s'_1 \xrightarrow{\sigma',(\!|\!|)} s'_2$, with σ' a renaming of σ. Thus, by the rule env, we have $s'_1 \xrightarrow{\sigma'} s'_2$. As s'_2 is a state of $\mathcal{S}_a^\sharp$ too, we can conclude. Similar proof for $\preceq$. □

Lemma B.6. *Let $s_1, s_2 \in \mathcal{Q}^\sharp$. Consider the transition rules from Definition A.20. If $s_1 \xrightarrow{\sigma,\Gamma} s_2$ and $s_1 \in \mathcal{Q}$, then there exists $s_3 \in \mathcal{Q}$ such that $s_1 \xrightarrow{\sigma,\Gamma} s_3$ and $s_2 \sqsubseteq s_3$.*

Proof. Let $s_1, s_2 \in \mathcal{Q}^\sharp$, σ an event and Γ an environment such that $s_1 \xrightarrow{\sigma,\Gamma} s_2$ and $s_1 \in \mathcal{Q}$. By structural induction on $s_1.pa = s_2.pa = a[\overrightarrow{T}]$.

1. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case aut_{1a}: $s_1 = (\mathsf{aut}_\circ, n_1, ss_1)$ and $s_2 = (\mathsf{aut}_\circ, n_2, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $\delta(n_1, n_2, \sigma', final?)$, $final? \Rightarrow \mathit{final}(ss_1)$, $\sigma'[\Gamma] = \sigma$ and $ss_2 \in \mathit{Abinit}(\nu(n_2)[\overrightarrow{V}])$. Let $ss_3 = \mathit{init}(\nu(n_2)[\overrightarrow{V}])$ and $s_3 = (\mathsf{aut}_\circ, n_2, ss_3)$.

We have $ss_2 \sqsubseteq ss_3$ and $s_2.val = s_3.val$. Thus, $s_2 \sqsubseteq s_3$. By Lemma A.1, $ss_3 \in \mathcal{Q}$. Thus, $s_3 \in \mathcal{Q}$. By inference rule aut_{1a}, $s_1 \xrightarrow{\sigma,\Gamma} s_3$.

- Case aut_2: $s_1 = (\mathsf{aut}_\circ, n_1, ss_1)$ and $s_2 = (\mathsf{aut}_\circ, n_1, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$ and $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. If $s_1 \in \mathcal{Q}$, then $ss_1 \in \mathcal{Q}$. By induction hypothesis, there exists $ss_3 \in \mathcal{Q}$ such that $ss_1 \xrightarrow{\sigma,\Gamma} ss_3$ and $ss_2 \sqsubseteq ss_3$. Let $s_3 = (\mathsf{aut}_\circ, n_1, ss_3)$. We have $s_3 \in \mathcal{Q}$ and $s_2 \sqsubseteq s_3$. By inference rule aut_2, $s_1 \xrightarrow{\sigma,\Gamma} s_3$.

2. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $\star_{1a}$: $s_1 = (\star_\circ, \mathit{started?}, ss_1)$ and $s_2 = (\star_\circ, \mathrm{true}, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $\mathit{final}(ss_1) \vee \neg \mathit{started?}$, $q \xrightarrow{\sigma,\Gamma} ss_2$ and $q \in \mathit{Abinit}(b[\overrightarrow{V}])$. Let $q' = \mathit{init}(b[\overrightarrow{V}])$. We have $q' \in \mathcal{Q}$ and $q \sqsubseteq q'$. By compatibility, there exists $ss_2' \in \mathcal{Q}^\sharp$ such that $q' \xrightarrow{\sigma,\Gamma} ss_2'$ and $ss_2 \sqsubseteq ss_2'$. By induction hypothesis, there exists $ss_3 \in \mathcal{Q}$ such that $q' \xrightarrow{\sigma,\Gamma} ss_3$ and $ss_2' \sqsubseteq ss_3$. Let $s_3 = (\star_\circ, \mathrm{true}, ss_3)$. We have $s_3 \in \mathcal{Q}$ and $s_2 \sqsubseteq s_2' \sqsubseteq s_3$. By inference rule $\star_{1a}$, $s_1 \xrightarrow{\sigma,\Gamma} s_3$.
   - Case $\star_2$: similar to case aut_2.

3. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Cases $|_{1a}$ and $|_{2a}$: similar to case $\star_{1a}$.
   - Cases $|_3$ and $|_4$: similar to case aut_2.

4. Inductive case: $a[\overrightarrow{T}] = (||, n, \Delta, l, r)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Cases $||_1$, $||_2$ and $||_3$: similar to case aut_2.

5. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $|:_{1a}$: similar to case $\star_{1a}$.
   - Case $|:_2$: similar to case aut_2.

6. Inductive case: $a[\overrightarrow{T}] = (|||:, n, x, X, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

- Case $|||:_1$: $s_1 = (|||:_\circ, f))$ and $s_2 = (|||:_\circ, f \Leftarrow \{v \mapsto ss_2)$. Similar to case aut_2, with induction hypothesis on ss_2. Note that if $s_1 \in \mathcal{Q}$, then for all $v' \in dom(f)$, $f(v') \in \mathcal{Q}$.

□

Lemma B.7. *Let us denote by $\rightarrow_1$ the transition relation from Definition A.6 and by $\rightarrow_2$ the transition relation from Definition A.20. Let $s_1, s_2 \in \mathcal{Q}$. If $s_1 \xrightarrow{\sigma,\Gamma}_2 s_2$, then $s_1 \xrightarrow{\sigma,\Gamma}_1 s_2$.*

Proof. Let $s_1, s_2 \in \mathcal{Q}$, σ an event and Γ an environment such that $s_1 \xrightarrow{\sigma,\Gamma}_2 s_2$. By structural induction on $s_1.pa = s_2.pa = a[\overrightarrow{T}]$, let us show that $s_1 \xrightarrow{\sigma,\Gamma}_1 s_2$. We detail only the new rules, as the other cases are obvious.

1. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma}_2 s_2$:

   - Case aut_{1a}: $s_1 = (\mathsf{aut}_\circ, n_1, ss_1)$ and $s_2 = (\mathsf{aut}_\circ, n_2, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $\delta(n_1, n_2, \sigma', final?)$, $final? \Rightarrow final(ss_1)$, $\sigma'[\Gamma] = \sigma$ and $ss_2 \in Abinit(\nu(n_2)[\overrightarrow{V}])$. As $s_2 \in \mathcal{Q}$, then $ss_2 \in \mathcal{Q}$. Thus, $ss_2 = init(\nu(n_2)[\overrightarrow{V}])$. By the inference rule aut_1, $s_1 \xrightarrow{\sigma,\Gamma}_1 s_2$.

2. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $\star_{1a}$: $s_1 = (\star_\circ, started?, ss_1)$ and $s_2 = (\star_\circ, \text{true}, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $final(ss_1) \vee \neg started?$, $q \xrightarrow{\sigma,\Gamma}_2 ss_2$ and $q \in Abinit(b[\overrightarrow{V}])$. Let $q' = init(b[\overrightarrow{V}])$. We have $q' \in \mathcal{Q}$ and $q \sqsubseteq q'$. By compatibility, there exists $ss_2' \in \mathcal{Q}^\sharp$ such that $q' \xrightarrow{\sigma,\Gamma}_2 ss_2'$ and $ss_2 \sqsubseteq ss_2'$. As $q.val = q'.val = \overrightarrow{V}$, then $ss_2'.val = \overrightarrow{V}$. We have $ss_2.val = \overrightarrow{V} = ss_2'.val$, $ss_2 \in \mathcal{Q}$ and $ss_2 \sqsubseteq ss_2'$. Thus, $ss_2 = ss_2'$. As a consequence, $init(b[\overrightarrow{V}]) \xrightarrow{\sigma,\Gamma}_2 ss_2$ and by induction hypothesis, $init(b[\overrightarrow{V}]) \xrightarrow{\sigma,\Gamma}_1 ss_2$. Thus, by the inference rule $\star_1$, $s_1 \xrightarrow{\sigma,\Gamma}_1 s_2$.

3. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Cases $|_{1a}$ and $|_{2a}$: similar to case $\star_{1a}$.

4. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b)[\overrightarrow{T}]$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $|:_{1a}$: similar to case $\star_{1a}$.

□

Lemma A.9. Let $a[\overrightarrow{T}]$ be a PASTD, $\mathcal{S}_a = (Q, \rightarrow)$ the TS derived from Definition A.13 with initial states I, and $\mathcal{S}_a^\sharp = (Q', \rightarrow')$ the TS from Definition A.22 with initial states I'. Then, for all path $s'_0 \rightarrow' \cdots \rightarrow' s'_n$ in $\mathcal{S}_a^\sharp$ with $s'_0 \in I'$, there exists a path $s_0 \rightarrow \cdots \rightarrow s_n$ in $\mathcal{S}_a$ with $s_0 \in I$ such that $s'_i \leq s_i$ for all $i \leq n$.

Proof. We have $Q \subseteq Q' \subseteq \mathcal{Q}^\sharp$ and $\rightarrow \subseteq \rightarrow'$. Let $s'_0 \rightarrow' \cdots \rightarrow' s'_n$ be a path in $\mathcal{S}_a^\sharp$ with $s'_0 \in I'$. Let $\overrightarrow{V} = s'_0.val$. By induction on the length n of the path $s'_0 \rightarrow' \cdots \rightarrow' s'_n$, let us show that there exists a path $s_0 \rightarrow \cdots \rightarrow s_n$ in $\mathcal{S}_a$ with $s_0 \in I$ such that $s'_i \sqsubseteq s_i$ for all $i \leq n$.

- Base case: $n = 1$, $s'_0 \rightarrow' s'_1$. Let $s_0 = \mathit{init}(a[\overrightarrow{V}])$. We have $s_0 \in I$ and $s'_0 \sqsubseteq s_0$. By monotony (Theorem A.3), there exists $q \in Q'$ such that $s_0 \rightarrow' q$ and $s'_1 \sqsubseteq q$. As $s_0 \in \mathcal{Q}$, take $s_1 \in \mathcal{Q}$ such that $s_0 \rightarrow' s_1$ and $q \sqsubseteq s_1$ thanks to Lemma B.6. We have $s_1 \in Q$ and $s'_1 \sqsubseteq s_1$. As $s_0, s_1 \in Q$ and $s_0 \rightarrow' s_1$, then $s_0 \rightarrow s_1$ by Lemma B.7.

- Inductive case: let $s'_0 \rightarrow' \cdots \rightarrow' s'_{n+1}$ be a path in $\mathcal{S}_a^\sharp$ with $s'_0 \in I'$ and suppose that there exists a path $s_0 \rightarrow \cdots \rightarrow s_n$ in $\mathcal{S}_a$ with $s_0 \in I$ such that $s'_i \sqsubseteq s_i$ for all $i \leq n$. Let us show that there is a path of length $n+1$ in $\mathcal{S}_a$. By monotony (Theorem A.3), there exists $q \in Q'$ such that $s_n \rightarrow' q$ and $s'_{n+1} \sqsubseteq q$. As $s_n \in \mathcal{Q}$, take $s_{n+1} \in \mathcal{Q}$ such that $s_n \rightarrow' s_{n+1}$ and $q \sqsubseteq s_{n+1}$ thanks to Lemma B.6. We have $s_{n+1} \in Q$ and $s'_{n+1} \sqsubseteq s_{n+1}$. As $s_n, s_{n+1} \in Q$ and $s_n \rightarrow' s_{n+1}$, then $s_n \rightarrow s_{n+1}$ by Lemma B.7.

□

B.5 Proofs of Section A.6

Lemma B.8. *Let $a[\overrightarrow{T}]$ be a PASTD and $\overrightarrow{V}$ a parameter value. Let $s \in \mathcal{Q}^\sharp$ such that $s \in Abinit(a[\overrightarrow{V}])$. Then, for all $q \in \mathcal{Q}^\sharp$ such that $q \sqsubseteq s$ and with $q.val = \overrightarrow{W}$, we have $q \in Abinit(a[\overrightarrow{W}])$.*

Proof. Let $q \in \mathcal{Q}^\sharp$ such that $q \sqsubseteq s$ with $q.val = \overrightarrow{W}$. By Definition A.19, $s \sqsubseteq init(a[\overrightarrow{V}])$. By transitivity, $q \sqsubseteq init(a[\overrightarrow{V}])$. By definition of *init*, $q \sqsubseteq init(a[\overrightarrow{W}])$ (easy proof by induction on $a[\overrightarrow{T}]$). □

Lemma A.11. Let $a[\overrightarrow{T}]$ be a PASTD. Let $c \in \mathbb{N}^k$ such that $c = \mu(a[\overrightarrow{T}])$. For all $s \in \mathcal{Q}^\sharp$ such that $s.pa = a[\overrightarrow{T}]$, there exists $q \in \mathcal{Q}^\sharp$ such that $q \sqsubseteq s$ and $\gamma(q) \leq_k c$.

Proof. Let $s \in \mathcal{Q}^\sharp$ such that $s.pa = a[\overrightarrow{V}]$ with $\overrightarrow{V} = s.val$. Let $c = \mu(a[\overrightarrow{T}])$. By structural induction on $s.pa = a[\overrightarrow{T}]$.

1. Base case: $a[\overrightarrow{T}] = (\mathsf{elem})$. We have $s = (\mathsf{elem}_\circ)$. Take $q = (\mathsf{elem}_\circ)$ with $\gamma(q) = (0, \ldots, 0)$. Thus $\gamma(q) = c$.

2. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$. We have $s = (\mathsf{aut}_\circ, n, ss)$ with $ss.pa = \nu(n)[\overrightarrow{T}]$. We have $c = sup(\{\mu(\nu(n')) \mid n' \in N\})$. By induction hypothesis, there exists $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$ and $\gamma(qq) \leq_k \mu(\nu(n)[\overrightarrow{T}])$. Take $q = (\mathsf{aut}_\circ, n, qq) \in \mathcal{Q}^\sharp$ such that $q.pa = s.pa$ and $q.val = qq.val$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq) \leq_k \mu(\nu(n)[\overrightarrow{T}]) \leq_k c$.

3. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. We have $c = \mu(a[\overrightarrow{T}]) = \mu(b[\overrightarrow{T}])$. If $s = (\star_\circ, \text{false}, \bot)$, then take $q = (\star_\circ, \text{false}, \bot)$ with $\gamma(q) = (0, \ldots, 0) \leq_k c$. Else, we have $s = (\star_\circ, \text{true}, ss)$ with $ss.pa = b[\overrightarrow{T}]$. By induction hypothesis, there exists $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$ and $\gamma(qq) \leq_k \mu(b[\overrightarrow{T}])$. Take $q = (\star_\circ, \text{true}, qq) \in \mathcal{Q}^\sharp$ such that $q.pa = s.pa$ and $q.val = qq.val$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq) \leq_k \mu(b[\overrightarrow{T}]) = c$.

4. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. We have $c = sup(\mu(l[\overrightarrow{T}]), \mu(r[\overrightarrow{T}]))$. If $s = (|_\circ, \bot, \bot)$, then take $q = (|_\circ, \bot, \bot)$ with $\gamma(q) = (0, \ldots, 0) \leq_k c$. Else, we have $s = (|_\circ, side, ss)$ with $ss.pa = l[\overrightarrow{T}]$ (or $ss.pa = r[\overrightarrow{T}]$). By induction hypothesis, there exists $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$ and $\gamma(qq) \leq_k \mu(l[\overrightarrow{T}])$ (or

$\gamma(qq) \leq_k \mu(r[\overrightarrow{T}]))$. Take $q = (|_\circ, side, qq) \in \mathcal{Q}^\sharp$ such that $q.pa = s.pa$ and $q.val = qq.val$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq) \leq_k c$.

5. Inductive case: $a[\overrightarrow{T}] = (||, n, \Delta, l, r)[\overrightarrow{T}]$. We have $c = \mu(l[\overrightarrow{T}]) + \mu(r[\overrightarrow{T}])$. We have $s = (||_\circ, s_l, s_r)$ with $s_l.pa = l[\overrightarrow{T}]$ and $s_r.pa = r[\overrightarrow{T}]$. By induction hypothesis, there exist $q_l, q_r \in \mathcal{Q}^\sharp$ such that $q_l \sqsubseteq s_l$, $q_r \sqsubseteq s_r$, $\gamma(q_l) \leq_k \mu(l[\overrightarrow{T}])$ and $\gamma(q_r) \leq_k \mu(r[\overrightarrow{T}])$. Let $\overrightarrow{W} = q_l.val \cup q_r.val$. By Lemma A.10, take $q'_l, q'_r \in \mathcal{Q}^\sharp$ such that $q_l \sqsubseteq q'_l \sqsubseteq s_l$, $q_r \sqsubseteq q'_r \sqsubseteq s_r$ and $q'_l.val = q'_r.val = \overrightarrow{W}$. Take $q = (||_\circ, q'_l, q'_r) \in \mathcal{Q}^\sharp$ such that $q.pa = s.pa$ and $q.val = \overrightarrow{W}$. We have $q \sqsubseteq s$ and $\gamma(q) \leq_k \gamma(q_l) + \gamma(q_r) \leq_k c$.

6. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b[\overrightarrow{T}])$. If $s = (|:_\circ, \bot, \bot)$, then take $q = (|:_\circ, \bot, \bot)$ with $\gamma(q) = (0, \ldots, 0) \leq_k c$. Else, we have $s = (|:_\circ, v, ss)$ with $ss.pa = b[\overrightarrow{T}]$. By induction hypothesis, there exists $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$ and $\gamma(qq) \leq_k \mu(b[\overrightarrow{T}])$. By cases on X:

   - If X is not a parameter, then take $q = (|:_\circ, v, qq) \in \mathcal{Q}^\sharp$ such that $q.pa = s.pa$ and $q.val = qq.val$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq) \leq_k c = \mu(b[\overrightarrow{T}])$.
   - If $X = T_i$ is the i-th parameter, then let $\overrightarrow{W} = qq.val \cup (\emptyset, \ldots, \{v\}, \ldots, \emptyset)$. By Lemma A.10, take $qq' \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq qq' \sqsubseteq ss$ and $qq'.val = \overrightarrow{W}$. Let $q = (|:, v, qq')$ with $q.pa = s.pa$ and $q.val = \overrightarrow{W}$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq') \leq_k \gamma(qq) + (0, \ldots, 1, \ldots, 0) \leq_k \mu(b[\overrightarrow{T}]) + (0, \ldots, 1, \ldots, 0) = c$.

7. Inductive case: $a[\overrightarrow{T}] = (|||:, n, x, X, b[\overrightarrow{T}])$. We have $s = (|||:_\circ, f)$ with for all $ss \in ran(f)$, $ss.pa = b[\overrightarrow{T}]$. By induction hypothesis, for all $ss_i \in ran(f)$, let $qq_i \in \mathcal{Q}^\sharp$ such that $qq_i \sqsubseteq ss_i$ and $\gamma(qq_i) \leq_k \mu(b[\overrightarrow{T}])$. By cases on X:

   - If X is not a parameter, then let $\overrightarrow{W} = \bigcup_i qq_i.val \subseteq \overrightarrow{V}$. We have $|\overrightarrow{W}| \leq_k \sum_i \gamma(qq_i) = |X| \cdot \gamma(qq_1)$. By Lemma A.10, for each qq_i take $qq'_i \in \mathcal{Q}^\sharp$ such that $qq_i \sqsubseteq qq'_i \sqsubseteq ss_i$ and $qq'_i.val = \overrightarrow{W}$. Take $q = (|||:_\circ, f') \in \mathcal{Q}^\sharp$ such that $dom(f') = dom(f)$, $f'(f^{-1}(ss_i)) = qq'_i$, $q.pa = s.pa$ and $q.val = \overrightarrow{W}$. We have $q \sqsubseteq s$ and $\gamma(q) = |X| \cdot \gamma(qq_1) \leq_k |X| \cdot \mu(b[\overrightarrow{T}]) = c$.
   - If $X = T_i$ is the i-th parameter, then choose a value $v \in dom(f)$ and let $ss = f(v)$. By induction hypothesis, let $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$ and

$\gamma(qq) \leq_k \mu(b[\overrightarrow{T}])$. Let $\overrightarrow{W} = qq.val \cup (\emptyset, \ldots, \{v\}, \ldots, \emptyset)$. By Lemma A.10, take $qq' \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq qq' \sqsubseteq ss$ and $qq'.val = \overrightarrow{W}$. Let $q = (|||:, \{v \mapsto qq'\})$ with $q.pa = s.pa$ and $q.val = \overrightarrow{W}$. We have $q \sqsubseteq s$ and $\gamma(q) = |\overrightarrow{W}| \leq_k \gamma(qq) + (0, \ldots, 1, \ldots, 0) \leq_k \mu(b[\overrightarrow{T}]) + (0, \ldots, 1, \ldots, 0) = c$.

□

Lemma A.12. Let $a[\overrightarrow{T}]$ be a PASTD. Let $c \in \mathbb{N}^k$ such that $c = \mu(a[\overrightarrow{T}])$. For all $s \in \mathcal{Q}^\sharp$ such that $s.pa = a[\overrightarrow{T}]$, if $\mathit{final}(s)$, then there exists $q \in \mathcal{Q}^\sharp$ such that $q \sqsubseteq s$, $\gamma(q) \leq_k c$ and $\mathit{final}(q)$.

Proof. Let $s \in \mathcal{Q}^\sharp$ such that $s.pa = a[\overrightarrow{V}]$ with $\overrightarrow{V} = s.val$ and $\mathit{final}(s)$. Let $c = \mu(a[\overrightarrow{T}])$. By structural induction on $s.pa = a[\overrightarrow{T}]$.

1. Base case: $a[\overrightarrow{T}] = (\mathsf{elem})$. We have $s = (\mathsf{elem}_\circ)$. Take $q = (\mathsf{elem}_\circ)$ with $\gamma(q) = (0, \ldots, 0)$. Thus $\gamma(q) = c$. Moreover, we have $\mathit{final}(q)$.

2. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$. We have $s = (\mathsf{aut}_\circ, n, ss)$ with $ss.pa = \nu(n)[\overrightarrow{T}]$. We have $c = sup(\{\mu(\nu(n')) \mid n' \in N\})$. By induction hypothesis, there exists $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$, $\gamma(qq) \leq_k \mu(\nu(n)[\overrightarrow{T}])$ and $\mathit{final}(ss) \implies \mathit{final}(qq)$. Take $q = (\mathsf{aut}_\circ, n, qq) \in \mathcal{Q}^\sharp$ such that $q.pa = s.pa$ and $q.val = qq.val$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq) \leq_k \mu(\nu(n)[\overrightarrow{T}]) \leq_k c$. As $\mathit{final}(s)$, either $n \in DF$ and $\mathit{final}(q)$, or $\mathit{final}(ss)$ and $\mathit{final}(qq)$, thus $\mathit{final}(q)$.

3. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. We have $c = \mu(a[\overrightarrow{T}]) = \mu(b[\overrightarrow{T}])$. If $s = (\star_\circ, \text{false}, \bot)$, then take $q = (\star_\circ, \text{false}, \bot)$ with $\gamma(q) = (0, \ldots, 0) \leq_k c$. Moreover, we have $\mathit{final}(q)$. Else, we have $s = (\star_\circ, \text{true}, ss)$ with $ss.pa = b[\overrightarrow{T}]$. By induction hypothesis, there exists $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$, $\gamma(qq) \leq_k \mu(b[\overrightarrow{T}])$ and $\mathit{final}(ss) \implies \mathit{final}(qq)$. Take $q = (\star_\circ, \text{true}, qq) \in \mathcal{Q}^\sharp$ such that $q.pa = s.pa$ and $q.val = qq.val$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq) \leq_k \mu(b[\overrightarrow{T}]) = c$. As $\mathit{final}(ss)$, we have $\mathit{final}(qq)$, thus $\mathit{final}(q)$.

4. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. We have $c = sup(\mu(l[\overrightarrow{T}]), \mu(r[\overrightarrow{T}]))$.

   - If $s = (|_\circ, \bot, \bot)$, then $\mathit{final}(\mathit{init}(l[\overrightarrow{V}])) \vee \mathit{final}(\mathit{init}(r[\overrightarrow{V}]))$.

  - Case $\mathit{final}(\mathit{init}(l[\overrightarrow{V}]))$: let $ss = \mathit{init}(l[\overrightarrow{V}])$. By induction hypothesis, there exists $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$, $\gamma(qq) \leq_k \mu(l[\overrightarrow{T}])$ and $\mathit{final}(qq)$. Let $\overrightarrow{W} = qq.val$. By Lemma B.8, $qq \in \mathit{Abinit}(l[\overrightarrow{W}])$. Thus, by Lemma B.2, $\mathit{final}(\mathit{init}(l[\overrightarrow{W}]))$. Take $q = (|_\circ, \bot, \bot)$ with $q.pa = s.pa$ and $q.val = \overrightarrow{W}$. We have $\mathit{final}(q)$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq) \leq_k c$.
  - Case $\mathit{final}(\mathit{init}(r[\overrightarrow{V}]))$: similar.

- Else, we have $s = (|_\circ, \mathit{side}, ss)$ with $ss.pa = l[\overrightarrow{T}]$ (or $ss.pa = r[\overrightarrow{T}]$). As $\mathit{final}(s)$, then $\mathit{final}(ss)$. By induction hypothesis, there exists $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$ and $\gamma(qq) \leq_k \mu(l[\overrightarrow{T}])$ (or $\gamma(qq) \leq_k \mu(r[\overrightarrow{T}])$) and $\mathit{final}(qq)$. Take $q = (|_\circ, \mathit{side}, qq) \in \mathcal{Q}^\sharp$ such that $q.pa = s.pa$ and $q.val = qq.val$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq) \leq_k c$. And as $\mathit{final}(qq)$, we have $\mathit{final}(q)$.

5. Inductive case: $a[\overrightarrow{T}] = (||, n, \Delta, l, r)[\overrightarrow{T}]$. We have $c = \mu(l[\overrightarrow{T}]) + \mu(r[\overrightarrow{T}])$. We have $s = (||_\circ, s_l, s_r)$ with $s_l.pa = l[\overrightarrow{T}]$ and $s_r.pa = r[\overrightarrow{T}]$. Moreover, as $\mathit{final}(s)$, then $\mathit{final}(s_l) \wedge \mathit{final}(s_r)$. By induction hypothesis, there exist $q_l, q_r \in \mathcal{Q}^\sharp$ such that $q_l \sqsubseteq s_l$, $q_r \sqsubseteq s_r$, $\gamma(q_l) \leq_k \mu(l[\overrightarrow{T}])$, $\gamma(q_r) \leq_k \mu(r[\overrightarrow{T}])$, $\mathit{final}(q_l)$ and $\mathit{final}(q_r)$. Let $\overrightarrow{W} = q_l.val \cup q_r.val$. By Lemma A.10, take $q'_l, q'_r \in \mathcal{Q}^\sharp$ such that $q_l \sqsubseteq q'_l \sqsubseteq s_l$, $q_r \sqsubseteq q'_r \sqsubseteq s_r$ and $q'_l.val = q'_r.val = \overrightarrow{W}$. Take $q = (||_\circ, q'_l, q'_r) \in \mathcal{Q}^\sharp$ such that $q.pa = s.pa$ and $q.val = \overrightarrow{W}$. We have $q \sqsubseteq s$ and $\gamma(q) \leq_k \gamma(q_l) + \gamma(q_r) \leq_k c$. By Lemma B.2, $\mathit{final}(q'_l)$ and $\mathit{final}(q'_r)$. Thus, $\mathit{final}(q)$.

6. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b[\overrightarrow{T}])$.

   - If $s = (|:_\circ, \bot, \bot)$, then $\mathit{final}(\mathit{init}(b[\overrightarrow{V}]))$. Let $ss = \mathit{init}(b[\overrightarrow{V}])$. By induction hypothesis, there exists $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$, $\gamma(qq) \leq_k \mu(b[\overrightarrow{T}])$ and $\mathit{final}(qq)$. Let $\overrightarrow{W} = qq.val$. By Lemma B.8, $qq \in \mathit{Abinit}(b[\overrightarrow{W}])$. Thus, we have $\mathit{final}(\mathit{init}(b[\overrightarrow{W}]))$ by Lemma B.2. Take $q = (|:_\circ, \bot, \bot)$ with $q.pa = s.pa$ and $q.val = \overrightarrow{W}$. We have $\mathit{final}(q)$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq) \leq_k c$ (if X is a parameter or not).
   - Else, we have $s = (|:_\circ, v, ss)$ with $ss.pa = b[\overrightarrow{T}]$ and $\mathit{final}(ss)$. By induction hypothesis, there exists $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$, $\gamma(qq) \leq_k \mu(b[\overrightarrow{T}])$ and $\mathit{final}(qq)$. By cases on X:

- If X is not a parameter, then take $q = (|:_\circ, v, qq) \in \mathcal{Q}^\sharp$ such that $q.pa = s.pa$ and $q.val = qq.val$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq) \leq_k c = \mu(b[\overrightarrow{T}])$. And as *final*$(qq)$, we have *final*$(q)$.
- If $X = T_i$ is the i-th parameter, then let $\overrightarrow{W} = qq.val \cup (\emptyset, \ldots, \{v\}, \ldots, \emptyset)$. By Lemma A.10, take $qq' \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq qq' \sqsubseteq ss$ and $qq'.val = \overrightarrow{W}$. Let $q = (|:, v, qq')$ with $q.pa = s.pa$ and $q.val = \overrightarrow{W}$. We have $q \sqsubseteq s$ and $\gamma(q) = \gamma(qq') \leq_k \gamma(qq) + (0, \ldots, 1, \ldots, 0) \leq_k \mu(b[\overrightarrow{T}]) + (0, \ldots, 1, \ldots, 0) = c$. And as *final*$(qq)$, then *final*$(qq')$ by Lemma B.2. Thus, *final*(q).

7. Inductive case: $a[\overrightarrow{T}] = (|||:, n, x, X, b[\overrightarrow{T}])$. We have $s = (|||:_\circ, f)$ where for all $ss \in ran(f)$, $ss.pa = b[\overrightarrow{T}]$. By induction hypothesis, for all $ss_i \in ran(f)$, let $qq_i \in \mathcal{Q}^\sharp$ such that $qq_i \sqsubseteq ss_i$, $\gamma(qq_i) \leq_k \mu(b[\overrightarrow{T}])$ and *final*$(ss_i) \implies$ *final*(qq_i). By cases on X:

   - If X is not a parameter, then for all $v \in X$, *final*$(f(v))$. Let $\overrightarrow{W} = \bigcup_i qq_i.val \subseteq \overrightarrow{V}$. We have $|\overrightarrow{W}| \leq_k \sum_i \gamma(qq_i) = |X| \cdot \gamma(qq_1)$. By Lemma A.10, for each qq_i take $qq_i' \in \mathcal{Q}^\sharp$ such that $qq_i \sqsubseteq qq_i' \sqsubseteq ss_i$ and $qq_i'.val = \overrightarrow{W}$. Take $q = (|||:_\circ, f') \in \mathcal{Q}^\sharp$ such that $dom(f') = dom(f)$, $f'(f^{-1}(ss_i)) = qq_i'$, $q.pa = s.pa$ and $q.val = \overrightarrow{W}$. We have $q \sqsubseteq s$ and $\gamma(q) = |X| \cdot \gamma(qq_1) \leq_k |X| \cdot \mu(b[\overrightarrow{T}]) = c$. By Lemma B.2, for all $qq_i' \in dom(f')$, *final*(qq_i'). Thus, *final*(q).
   - If $X = T_i$ is the i-th parameter, take $v \in dom(f)$ such that *final*$(f(v))$ and let $ss = f(v)$. By induction hypothesis, let $qq \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq ss$, $\gamma(qq) \leq_k \mu(b[\overrightarrow{T}])$ and *final*(qq). Let $\overrightarrow{W} = qq.val \cup (\emptyset, \ldots, \{v\}, \ldots, \emptyset)$. By Lemma A.10, take $qq' \in \mathcal{Q}^\sharp$ such that $qq \sqsubseteq qq' \sqsubseteq ss$ and $qq'.val = \overrightarrow{W}$. Let $q = (|||:, \{v \mapsto qq'\})$ with $q.pa = s.pa$ and $q.val = \overrightarrow{W}$. We have $q \sqsubseteq s$ and $\gamma(q) = |\overrightarrow{W}| \leq_k \gamma(qq) + (0, \ldots, 1, \ldots, 0) \leq_k \mu(b[\overrightarrow{T}]) + (0, \ldots, 1, \ldots, 0) = c$. By Lemma B.2, *final*$(qq')$. Thus, *final*$(q)$.

□

Lemma B.9. *Let $a[\overrightarrow{T}]$ be a PASTD. Let $c \in \mathbb{N}^k$ such that $c = \mu(a[\overrightarrow{T}])$. For all $s_1, s_2 \in \mathcal{Q}^\sharp$ such that $s_1.pa = s_2.pa = a[\overrightarrow{T}]$ and $s_1 \xrightarrow{\sigma,\Gamma} s_2$, there exist $q_1, q_2 \in \mathcal{Q}^\sharp$ such that $q_1.pa = q_2.pa = a[\overrightarrow{T}]$ and $q_1 \xrightarrow{\sigma,\Gamma} q_2$ and:*

B.5. Proofs of Section A.6

1. $\gamma(q_1) = \gamma(q_2) \leq_k c$, *and*
2. $q_1 \sqsubseteq s_1$ *and* $q_2 \sqsubseteq s_2$, *and*
3. *for all* $q_2' \in \mathcal{Q}^\sharp$ *such that* $q_2 \sqsubseteq q_2' \sqsubseteq s_2$, *there exists* $q_1' \in \mathcal{Q}^\sharp$ *such that* $q_1 \sqsubseteq q_1' \sqsubseteq s_1$ *and* $q_1' \xrightarrow{\sigma,\Gamma} q_2'$.

Proof. Let $s_1, s_2 \in \mathcal{Q}^\sharp$ two states, σ an event and Γ an environment such that $s_1 \xrightarrow{\sigma,\Gamma} s_2$. Let $\overrightarrow{V} = s_1.val = s_2.val$. By structural induction on $s_1.pa = s_2.pa = a[\overrightarrow{T}]$.

1. Base case: $a[\overrightarrow{T}] = (\mathsf{elem})$. There is no transition from $(\mathsf{elem}_\circ)$.
2. Inductive case: $a[\overrightarrow{T}] = (\mathsf{aut}, name, \Sigma, N, \nu, \delta, SF, DF, n_0)[\overrightarrow{T}]$. Let $c = \mu(a[\overrightarrow{T}]) = sup(\{\mu(\nu(n')) \mid n' \in N\})$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:
   - Case aut_{1a}: $s_1 = (\mathsf{aut}_\circ, n_1, ss_1)$ and $s_2 = (\mathsf{aut}_\circ, n_2, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $\delta(n_1, n_2, \sigma', final?)$, $final? \Rightarrow final(ss_1)$, $\sigma'[\Gamma] = \sigma$ and $ss_2 \in Abinit(\nu(n_2)[\overrightarrow{V}])$. Let $cc_1 = \mu(ss_1.pa) = \mu(\nu(n_1))$ and $cc_2 = \mu(ss_2.pa) = \mu(\nu(n_2))$.
     - If $final?$, then $final(ss_1)$. By Lemma A.12, take $qq_1 \in \mathcal{Q}^\sharp$ such that $final(qq_1)$, $\gamma(qq_1) \leq_k cc_1$ and $qq_1 \sqsubseteq ss_1$. We also have that for all $qq_1' \in \mathcal{Q}^\sharp$ such that $qq_1 \sqsubseteq qq_1' \sqsubseteq ss_1$, $final(qq_1')$ by Lemma A.4. By Lemma A.11, take $qq_2 \in \mathcal{Q}^\sharp$ such that $qq_2 \sqsubseteq ss_2$ and $\gamma(qq_2) \leq_k cc_2$. We have $qq_2 \in Abinit(\nu(n_2)[qq_2.val])$ by Lemma B.8. Let $\overrightarrow{W} = sup(qq_1.val, qq_2.val) \subseteq \overrightarrow{V}$. We have $|\overrightarrow{W}| \leq_k sup(cc_1, cc_2)$ by definition of sup. Take $qq_1' \in \mathcal{Q}^\sharp$ such that $qq_1 \sqsubseteq qq_1' \sqsubseteq ss_1$ and $qq_1'.val = \overrightarrow{W}$, and $qq_2' \in \mathcal{Q}^\sharp$ such that $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$ and $qq_2'.val = \overrightarrow{W}$, which both exist by Lemma A.10. We still have $final(qq_1')$ and $qq_2' \in Abinit(\nu(n_2)[\overrightarrow{W}])$. Let $q_1 = (\mathsf{aut}_\circ, n_1, qq_1')$ and $q_2 = (\mathsf{aut}_\circ, n_2, qq_2')$. Thus, by the inference rule aut_{1a}, we have $q_1 \xrightarrow{\sigma,\Gamma} q_2$.
       (a) We have $\gamma(q_1) = \gamma(q_2) = |\overrightarrow{W}| \leq_k sup(cc_1, cc_2) \leq_k c$.
       (b) As $qq_1' \sqsubseteq ss_1$ and $qq_2' \sqsubseteq ss_2$, then $q_1 \sqsubseteq s_1$ and $q_2 \sqsubseteq s_2$.
       (c) Let $q_2'' \in \mathcal{Q}^\sharp$ such that $q_2 \sqsubseteq q_2'' \sqsubseteq s_2$ with $q_2''.val = \overrightarrow{V'}$. We have $q_2'' = (\mathsf{aut}_\circ, n_2, qq_2'')$ with $qq_2' \sqsubseteq qq_2'' \sqsubseteq ss_2$ and $qq_2'' \in Abinit(\nu(n_2)[\overrightarrow{V'}])$

by Lemma B.8. Take $qq_1'' \in \mathcal{Q}^\sharp$ such that $qq_1''.val = \overrightarrow{V'}$ and $qq_1' \sqsubseteq qq_1'' \sqsubseteq ss_1$, and which exists by Lemma A.10. We have $\mathit{final}(qq_1'')$ by Lemma A.4. Let $q_1'' = (\mathsf{aut}_\circ, n_1, qq_1'')$. Thus, $q_1 \sqsubseteq q_1'' \sqsubseteq s_1$. And by the rule aut_{1a}, $q_1'' \xrightarrow{\sigma,\Gamma} q_2''$.

- If $\neg final?$, then take $qq_1 \in \mathcal{Q}^\sharp$ such that $qq_1 \sqsubseteq ss_1$ and $\gamma(qq_1) \leq_k cc_1$, thanks to Lemma A.11 instead of Lemma A.12. The rest is similar to the case $final?$, without using the "final" conditions.

- Case aut_2: $s_1 = (\mathsf{aut}_\circ, n_1, ss_1)$ and $s_2 = (\mathsf{aut}_\circ, n_1, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$ and $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. Let $cc = \mu(ss_1.pa) = \mu(\nu(n_1)[\overrightarrow{T}])$. By induction hypothesis, take $qq_1, qq_2 \in \mathcal{Q}^\sharp$ such that $qq_1 \xrightarrow{\sigma,\Gamma} qq_2$ with $\gamma(qq_1) = \gamma(qq_2) \leq_k cc$, $qq_1 \sqsubseteq ss_1$, $qq_2 \sqsubseteq ss_2$ and for all $qq_2' \in \mathcal{Q}^\sharp$ with $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$ there exists $qq_1' \in \mathcal{Q}^\sharp$ such that $qq_1 \sqsubseteq qq_1' \sqsubseteq ss_1$ and $qq_1' \xrightarrow{\sigma,\Gamma} qq_2'$. Let $\overrightarrow{W} = qq_1.val$, $q_1 = (\mathsf{aut}_\circ, n_1, qq_1)$ with $q_1.pa = a$ and $q_1.val = \overrightarrow{W}$ and $q_2 = (\mathsf{aut}_\circ, n_1, qq_2)$ with $q_2.pa = a$ and $q_2.val = \overrightarrow{W}$. By the rule aut_2, we have $q_1 \xrightarrow{\sigma,\Gamma} q_2$.
  (a) We have $\gamma(q_1) = \gamma(q_2) = |\overrightarrow{W}| \leq_k cc \leq_k c$.
  (b) As $qq_1 \sqsubseteq ss_1$ and $qq_2 \sqsubseteq ss_2$, then $q_1 \sqsubseteq s_1$ and $q_2 \sqsubseteq s_2$.
  (c) Let $q_2' \in \mathcal{Q}^\sharp$ such that $q_2 \sqsubseteq q_2' \sqsubseteq s_2$ with $q_2'.val = \overrightarrow{V'}$. We have $q_2' = (\mathsf{aut}_\circ, n_1, qq_2')$ with $q_2'.pa = a$ and $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$. Take $qq_1' \in \mathcal{Q}^\sharp$ satisfying the induction hypothesis, *i.e.* $qq_1 \sqsubseteq qq_1' \sqsubseteq ss_1$ and $qq_1' \xrightarrow{\sigma,\Gamma} qq_2'$. Let $q_1' = (\mathsf{aut}_\circ, n_1, qq_1')$ with $q_1'.pa = a$ and $q_1'.val = \overrightarrow{V'}$. We have $q_1 \sqsubseteq q_1' \sqsubseteq s_1$ because $qq_1 \sqsubseteq qq_1' \sqsubseteq ss_1$. Also, $q_1' \xrightarrow{\sigma,\Gamma} q_2'$ by the rule aut_2.

3. Inductive case: $a[\overrightarrow{T}] = (\star, n, b)[\overrightarrow{T}]$. Let $c = \mu(a[\overrightarrow{T}]) = \mu(b[\overrightarrow{T}])$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $\star_{1a}$: $s_1 = (\star_\circ, started?, ss_1)$ and $s_2 = (\star_\circ, \mathrm{true}, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $\mathit{final}(ss_1) \vee \neg started?$, $q \xrightarrow{\sigma,\Gamma} ss_2$ and $q \in \mathit{Abinit}(b[\overrightarrow{V}])$. Let $cc = \mu(b[\overrightarrow{T}]) = c$.
     - If $started?$, then we have $\mathit{final}(ss_1)$. By Lemma A.12, take $qq_1 \in \mathcal{Q}^\sharp$ such that $\mathit{final}(qq_1)$, $\gamma(qq_1) \leq_k cc = c$ and $qq_1 \sqsubseteq ss_1$. By Lemma A.4, for all $qq_1' \in \mathcal{Q}^\sharp$ such that $qq_1 \sqsubseteq qq_1' \sqsubseteq ss_1$, $\mathit{final}(qq_1')$. By induction hypothesis, take $q', qq_2 \in \mathcal{Q}^\sharp$ such that $q' \xrightarrow{\sigma,\Gamma} qq_2$ with $\gamma(q') = \gamma(qq_2) \leq_k$

$cc = c$, $q' \sqsubseteq q$, $qq_2 \sqsubseteq ss_2$ and for all $qq_2' \in \mathcal{Q}^\sharp$ with $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$, there exists $q'' \in \mathcal{Q}^\sharp$ such that $q' \sqsubseteq q'' \sqsubseteq q$ and $q'' \xrightarrow{\sigma,\Gamma} qq_2'$. Let $\overrightarrow{W} = sup(qq_1.val, qq_2.val) \subseteq \overrightarrow{V}$. We have $|\overrightarrow{W}| \leq_k cc$. Take $qq_1' \in \mathcal{Q}^\sharp$ such that $qq_1 \sqsubseteq qq_1' \sqsubseteq ss_1$ and $qq_1'.val = \overrightarrow{W}$, and $qq_2' \in \mathcal{Q}^\sharp$ such that $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$ and $qq_2'.val = \overrightarrow{W}$, which both exist by Lemma A.10. We have $final(qq_1')$. And by induction hypothesis, take $q'' \in \mathcal{Q}^\sharp$ such that $q' \sqsubseteq q'' \sqsubseteq q$ and $q'' \xrightarrow{\sigma,\Gamma} qq_2'$. We have $q'' \in Abinit(b[\overrightarrow{W}])$ by Lemma B.8. Let $q_1 = (\star_\circ, started?, qq_1')$ and $q_2 = (\star_\circ, \text{true}, qq_2')$. Thus, by the inference rule $\star_{1a}$, we have $q_1 \xrightarrow{\sigma,\Gamma} q_2$.

(a) We have $\gamma(q_1) = \gamma(q_2) = |\overrightarrow{W}| \leq_k cc = c$.

(b) As $qq_1' \sqsubseteq ss_1$ and $qq_2' \sqsubseteq ss_2$, then $q_1 \sqsubseteq s_1$ and $q_2 \sqsubseteq s_2$.

(c) Let $q_2'' \in \mathcal{Q}^\sharp$ such that $q_2 \sqsubseteq q_2'' \sqsubseteq s_2$ with $q_2''.val = \overrightarrow{V'}$. We have $q_2'' = (\star_\circ, \text{true}, qq_2'')$ with $qq_2' \sqsubseteq qq_2'' \sqsubseteq ss_2$. Take $q'' \in \mathcal{Q}^\sharp$ satisfying the induction hypothesis, *i.e.* $q' \sqsubseteq q'' \sqsubseteq q$ and $q'' \xrightarrow{\sigma,\Gamma} qq_2'$. We have $q'' \in Abinit(b[\overrightarrow{V'}])$ by Lemma B.8. Take $qq_1'' \in \mathcal{Q}^\sharp$ such that $qq_1''.val = q_2''.val = \overrightarrow{V'}$ and $qq_1' \sqsubseteq qq_1'' \sqsubseteq ss_1$, and which exists by Lemma A.10. We have $final(qq_1'')$ by Lemma A.4. Let $q_1'' = (\star_\circ, started?, qq_1'')$. We have $q_1 \sqsubseteq q_1'' \sqsubseteq s_1$. And by the rule $\star_{1a}$, $q_1'' \xrightarrow{\sigma,\Gamma} q_2''$.

– If $\neg started?$, then take $qq_1 \in \mathcal{Q}^\sharp$ such that $qq_1 \sqsubseteq ss_1$ and $\gamma(qq_1) \leq_k cc$, thanks to Lemma A.11 instead of Lemma A.12. The rest is similar to the case *started*?, without using the "final" conditions.

- Case $\star_2$: $s_1 = (\star_\circ, \text{true}, ss_1)$ and $s_2 = (\star_\circ, \text{true}, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. Let $cc = \mu(b[\overrightarrow{T}])$. Similar to case aut_2.

4. Inductive case: $a[\overrightarrow{T}] = (|, n, l, r)[\overrightarrow{T}]$. Let $c = \mu(a[\overrightarrow{T}]) = sup(\mu(l[\overrightarrow{T}]), \mu(r[\overrightarrow{T}]))$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

- Case $|_{1a}$: $s_1 = (|_\circ, \bot, \bot)$ and $s_2 = (|_\circ, \mathsf{left}, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $q \xrightarrow{\sigma,\Gamma} ss_2$ and $q \in Abinit(l[\overrightarrow{V}])$. Let $cc = \mu(ss_2.pa) = \mu(l[\overrightarrow{T}])$. By induction hypothesis, take $q', qq_2 \in \mathcal{Q}^\sharp$ such that $q' \xrightarrow{\sigma,\Gamma} qq_2$ with $\gamma(q') = \gamma(qq_2) \leq_k cc$, $q' \sqsubseteq q$, $qq_2 \sqsubseteq ss_2$ and for all $qq_2' \in \mathcal{Q}^\sharp$ with $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$,

there exists $q'' \in \mathcal{Q}^\sharp$ such that $q' \sqsubseteq q'' \sqsubseteq q$ and $q'' \xrightarrow{\sigma,\Gamma} qq_2'$. Let $\overrightarrow{W} = qq_2.val$. Let $q_1 = (|_\circ, \bot, \bot)$ and $q_2 = (|_\circ, \mathsf{left}, qq_2')$, with $q_1.val = q_2.val = \overrightarrow{W}$ and $q_1.pa = q_2.pa = a[\overrightarrow{T}]$. We have $q' \in \mathit{Abinit}(l[\overrightarrow{W}])$ by Lemma B.8. Thus, by the inference rule $|_{1a}$, we have $q_1 \xrightarrow{\sigma,\Gamma} q_2$.

(a) We have $\gamma(q_1) = \gamma(q_2) = |\overrightarrow{W}| \leq_k cc \leq_k c$.

(b) As $\overrightarrow{W} \subseteq \overrightarrow{V}$ and $qq_2 \sqsubseteq ss_2$, then $q_1 \sqsubseteq s_1$ and $q_2 \sqsubseteq s_2$.

(c) Let $q_2' \in \mathcal{Q}^\sharp$ such that $q_2 \sqsubseteq q_2' \sqsubseteq s_2$ with $q_2'.val = \overrightarrow{V'}$. We have $q_2' = (|_\circ, \mathsf{left}, qq_2')$ with $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$. Take $q'' \in \mathcal{Q}^\sharp$ satisfying the induction hypothesis, *i.e.* $q' \sqsubseteq q'' \sqsubseteq q$ and $q'' \xrightarrow{\sigma,\Gamma} qq_2'$. We have $q'' \in \mathit{Abinit}(l[\overrightarrow{V'}])$ by Lemma B.8. Let $q_1' = (|_\circ, \bot, \bot)$ with $q_1'.val = \overrightarrow{V'}$ and $q_1'.pa = a[\overrightarrow{T}]$. We have $q_1 \sqsubseteq q_1' \sqsubseteq s_1$. And by the rule $|_{1a}$, $q_1' \xrightarrow{\sigma,\Gamma} q_2'$.

- Case $|_{2a}$: same as previous with $r[\overrightarrow{T}]$.
- Case $|_3$: $s_1 = (|_\circ, \mathsf{left}, ss_1)$ and $s_2 = (|_\circ, \mathsf{left}, ss_2)$ with $ss_1 \xrightarrow{\sigma,\Gamma} ss_2$. Let $cc = \mu(ss_1.pa) = \mu(l[\overrightarrow{T}])$. Similar to the case aut_2.
- Case $|_4$: same as previous.

5. Inductive case: $a[\overrightarrow{T}] = (||, n, \Delta, l, r)[\overrightarrow{T}]$. Let $c = \mu(a[\overrightarrow{T}]) = \mu(l[\overrightarrow{T}]) + \mu(r[\overrightarrow{T}])$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $||_1$: $s_1 = (||_\circ, s_{l1}, s_{r1})$ and $s_2 = (||_\circ, s_{l2}, s_{r1})$ with $s_1.val = s_2.val = \overrightarrow{V}$, $\alpha(\sigma) \notin \Delta$ and $s_{l1} \xrightarrow{\sigma,\Gamma} s_{l2}$. Let $cc_l = \mu(s_{l1}.pa) = \mu(l[\overrightarrow{T}])$ and $cc_r = \mu(s_{r1}.pa) = \mu(r[\overrightarrow{T}])$. By induction hypothesis, take $q_{l1}, q_{l2} \in \mathcal{Q}^\sharp$ such that $q_{l1} \xrightarrow{\sigma,\Gamma} q_{l2}$ with $\gamma(q_{l1}) = \gamma(q_{l2}) \leq_k cc_l$, $q_{l1} \sqsubseteq s_{l1}$, $q_{l2} \sqsubseteq s_{l2}$ and for all $q_{l2}' \in \mathcal{Q}^\sharp$ with $q_{l2} \sqsubseteq q_{l2}' \sqsubseteq s_{l2}$ there exists $q_{l1}' \in \mathcal{Q}^\sharp$ such that $q_{l1} \sqsubseteq q_{l1}' \sqsubseteq s_{l1}$ and $q_{l1}' \xrightarrow{\sigma,\Gamma} q_{l2}'$. Besides, by Lemma A.11, take $q_{r1} \in \mathcal{Q}^\sharp$ such that $\gamma(q_{r1}) \leq_k cc_r$ and $q_{r1} \sqsubseteq s_{r1}$. Let $\overrightarrow{W} = q_{l1}.val \cup q_{r1}.val \subseteq \overrightarrow{V}$. We have $|\overrightarrow{W}| \leq_k \gamma(q_{l1}) + \gamma(q_{r1}) \leq_k cc_l + cc_r$. Take $q_{l2}', q_{r1}' \in \mathcal{Q}^\sharp$ such that $q_{l2} \sqsubseteq q_{l2}' \sqsubseteq s_{l2}$, $q_{r1} \sqsubseteq q_{r1}' \sqsubseteq s_{r1}$ and $q_{l2}'.val = q_{r1}'.val = \overrightarrow{W}$ by Lemma A.10. By the induction hypothesis, take $q_{l1}' \in \mathcal{Q}^\sharp$ such that $q_{l1} \sqsubseteq q_{l1}' \sqsubseteq s_{l1}$ and $q_{l1}' \xrightarrow{\sigma,\Gamma} q_{l2}'$. Let $q_1 = (||_\circ, q_{l1}', q_{r1}')$ with $q_1.pa = a$ and $q_1.val = \overrightarrow{W}$ and $q_2 = (||_\circ, q_{l2}', q_{r1}')$ with $q_2.pa = a$ and $q_2.val = \overrightarrow{W}$. By the rule $||_1$, we have $q_1 \xrightarrow{\sigma,\Gamma} q_2$.

(a) We have $\gamma(q_1) = \gamma(q_2) = |\overrightarrow{W}| \leq_k cc_l + cc_r = c$.

(b) As $q_{l1}' \sqsubseteq s_{l1}$, $q_{l2}' \sqsubseteq s_{l2}$ and $q_{r1}' \sqsubseteq s_{r1}$ then $q_1 \sqsubseteq s_1$ and $q_2 \sqsubseteq s_2$.

(c) Let $q_2' \in \mathcal{Q}^\sharp$ such that $q_2 \sqsubseteq q_2' \sqsubseteq s_2$ with $q_2'.val = \overrightarrow{V'}$. We have $q_2' = (||_\circ, q_{l2}'', q_{r1}'')$ with $q_2'.pa = a$ and $q_{l2}' \sqsubseteq q_{l2}'' \sqsubseteq s_{l2}$ and $q_{r1}' \sqsubseteq q_{r1}'' \sqsubseteq s_{r1}$ and $q_{l2}''.val = q_{r2}'' = \overrightarrow{V'}$. Take $q_{l1}'' \in \mathcal{Q}^\sharp$ satisfying the induction hypothesis, *i.e.* $q_{l1}' \sqsubseteq q_{l1}'' \sqsubseteq s_{l1}$ and $q_{l1}'' \xrightarrow{\sigma,\Gamma} q_{l2}''$. Let $q_1' = (||_\circ, q_{l1}'', q_{r1}'')$ with $q_1'.pa = a$ and $q_1'.val = q_{l1}''.val = q_{r1}''.val = \overrightarrow{V'}$. We have $q_1 \sqsubseteq q_1' \sqsubseteq s_1$ because $q_{l1}' \sqsubseteq q_{l1}'' \sqsubseteq s_{l1}$ and $q_{r1}' \sqsubseteq q_{r1}'' \sqsubseteq s_{r1}$. Finally, $q_1' \xrightarrow{\sigma,\Gamma} q_2'$ by the rule $||_1$.

- Case $||_2$: similar.
- Case $||_3$: $s_1 = (||_\circ, s_{l1}, s_{r1})$ and $s_2 = (||_\circ, s_{l2}, s_{r1})$ with $s_1.val = s_2.val = \overrightarrow{V}$, $\alpha(\sigma) \in \Delta$, $s_{l1} \xrightarrow{\sigma,\Gamma} s_{l2}$ and $s_{r1} \xrightarrow{\sigma,\Gamma} s_{r2}$. Let $cc_l = \mu(s_{l1}.pa) = \mu(l[\overrightarrow{T}])$ and $cc_r = \mu(s_{r1}.pa) = \mu(r[\overrightarrow{T}])$. By induction hypothesis, take $q_{l1}, q_{l2} \in \mathcal{Q}^\sharp$ such that $q_{l1} \xrightarrow{\sigma,\Gamma} q_{l2}$ with $\gamma(q_{l1}) = \gamma(q_{l2}) \leq_k cc_l$, $q_{l1} \sqsubseteq s_{l1}$, $q_{l2} \sqsubseteq s_{l2}$ and for all $q_{l2}' \in \mathcal{Q}^\sharp$ with $q_{l2} \sqsubseteq q_{l2}' \sqsubseteq s_{l2}$ there exists $q_{l1}' \in \mathcal{Q}^\sharp$ such that $q_{l1} \sqsubseteq q_{l1}' \sqsubseteq s_{l1}$ and $q_{l1}' \xrightarrow{\sigma,\Gamma} q_{l2}'$. Similarly, take $q_{r1}, q_{r2} \in \mathcal{Q}^\sharp$ with $q_{r1} \xrightarrow{\sigma,\Gamma} q_{r2}$, $\gamma(q_{r1}) = \gamma(q_{r2}) \leq_k cc_r$, $q_{r1} \sqsubseteq s_{r1}$, $q_{r2} \sqsubseteq s_{r2}$ ... Let $\overrightarrow{W} = q_{l1}.val \cup q_{r1}.val \subseteq \overrightarrow{V}$. We have $|\overrightarrow{W}| \leq_k cc_l + cc_r$. Take $q_{l2}', q_{r2}' \in \mathcal{Q}^\sharp$ such that $q_{l2} \sqsubseteq q_{l2}' \sqsubseteq s_{l2}$, $q_{r2} \sqsubseteq q_{r2}' \sqsubseteq s_{r2}$ and $q_{l2}'.val = q_{r2}'.val = \overrightarrow{W}$ by Lemma A.10. By the induction hypothesis, take $q_{l1}', q_{r1}' \in \mathcal{Q}^\sharp$ such that $q_{l1} \sqsubseteq q_{l1}' \sqsubseteq s_{l1}$, $q_{r1} \sqsubseteq q_{r1}' \sqsubseteq s_{r1}$, $q_{l1}' \xrightarrow{\sigma,\Gamma} q_{l2}'$ and $q_{r1}' \xrightarrow{\sigma,\Gamma} q_{r2}'$. Let $q_1 = (||_\circ, q_{l1}', q_{r1}')$ with $q_1.pa = a$ and $q_1.val = \overrightarrow{W}$ and $q_2 = (||_\circ, q_{l2}', q_{r2}')$ with $q_2.pa = a$ and $q_2.val = \overrightarrow{W}$. By the rule $||_3$, we have $q_1 \xrightarrow{\sigma,\Gamma} q_2$.

  (a) We have $\gamma(q_1) = \gamma(q_2) = |\overrightarrow{W}| \leq_k cc_l + cc_r = c$.

  (b) As $q_{l1}' \sqsubseteq s_{l1}$, $q_{l2}' \sqsubseteq s_{l2}$, $q_{r1}' \sqsubseteq s_{r1}$ and $q_{r2}' \sqsubseteq s_{r2}$ then $q_1 \sqsubseteq s_1$ and $q_2 \sqsubseteq s_2$.

  (c) Let $q_2' \in \mathcal{Q}^\sharp$ such that $q_2 \sqsubseteq q_2' \sqsubseteq s_2$ with $q_2'.val = \overrightarrow{V'}$. We have $q_2' = (||_\circ, q_{l2}'', q_{r2}'')$ with $q_2'.pa = a$ and $q_{l2}' \sqsubseteq q_{l2}'' \sqsubseteq s_{l2}$ and $q_{r2}' \sqsubseteq q_{r2}'' \sqsubseteq s_{r2}$ and $q_{l2}''.val = q_{r2}'' = \overrightarrow{V'}$. Take $q_{l1}'', q_{r1}'' \in \mathcal{Q}^\sharp$ satisfying the induction hypothesis, *i.e.* $q_{l1}' \sqsubseteq q_{l1}'' \sqsubseteq s_{l1}$, $q_{r1}' \sqsubseteq q_{r1}'' \sqsubseteq s_{r1}$, $q_{l1}'' \xrightarrow{\sigma,\Gamma} q_{l2}''$ and $q_{r1}'' \xrightarrow{\sigma,\Gamma} q_{r2}''$. Let $q_1' = (||_\circ, q_{l1}'', q_{r1}'')$ with $q_1'.pa = a$ and $q_1'.val = q_{l1}''.val = q_{r1}''.val = \overrightarrow{V'}$. We have $q_1 \sqsubseteq q_1' \sqsubseteq s_1$ because $q_{l1}' \sqsubseteq q_{l1}'' \sqsubseteq s_{l1}$

and $q'_{r1} \sqsubseteq q''_{r1} \sqsubseteq s_{r1}$. Finally, $q'_1 \xrightarrow{\sigma,\Gamma} q'_2$ by the rule $||_3$.

6. Inductive case: $a[\overrightarrow{T}] = (|:, n, x, X, b)[\overrightarrow{T}]$. Let $c = \mu(a[\overrightarrow{T}])$. By cases on inference rules for $s_1 \xrightarrow{\sigma,\Gamma} s_2$:

   - Case $|:_1$: $s_1 = (|:_\circ, \bot, \bot)$ and $s_2 = (|:_\circ, v, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $q \xrightarrow{\sigma,([x:=v])\lhd\Gamma} ss_2$, $q \in \mathit{Abinit}(b[\overrightarrow{V}])$ and $v \in X$.
     - If X is the i-th parameter, then $c = \mu(b[\overrightarrow{T}]) + (0, \ldots, 1, \ldots, 0)$. Let $cc = \mu(ss_2.pa) = \mu(b[\overrightarrow{T}])$. By induction hypothesis, take $q', qq_2 \in \mathcal{Q}^\sharp$ such that $q' \xrightarrow{\sigma,([x:=v])\lhd\Gamma} qq_2$ with $\gamma(q') = \gamma(qq_2) \leq_k cc$, $q' \sqsubseteq q$, $qq_2 \sqsubseteq ss_2$ and for all $qq'_2 \in \mathcal{Q}^\sharp$ with $qq_2 \sqsubseteq qq'_2 \sqsubseteq ss_2$, there exists $q'' \in \mathcal{Q}^\sharp$ such that $q' \sqsubseteq q'' \sqsubseteq q$ and $q'' \xrightarrow{\sigma,([x:=v])\lhd\Gamma} qq'_2$. Let $\overrightarrow{W} = qq_2.val$ and $\overrightarrow{W'} = \overrightarrow{W} \cup (\emptyset, \ldots, \{v\}, \ldots, \emptyset)$ (the i-th component of $\overrightarrow{W}$ is augmented by v). We have $|\overrightarrow{W'}| \leq_k cc + (0, \ldots, 1, \ldots, 0)$. Let $qq'_2 \in \mathcal{Q}^\sharp$ such that $qq_2 \sqsubseteq qq'_2 \sqsubseteq ss_2$ and $qq'_2.val = \overrightarrow{W'}$, which exists by Lemma A.10. By induction hypothesis, take $q'' \in \mathcal{Q}^\sharp$ such that $q' \sqsubseteq q'' \sqsubseteq q$ and $q'' \xrightarrow{\sigma,([x:=v])\lhd\Gamma} qq'_2$. Let $q_1 = (|:_\circ, \bot, \bot)$ and $q_2 = (|:_\circ, v, qq'_2)$, with $q_1.val = q_2.val = \overrightarrow{W'}$ and $q_1.pa = q_2.pa = a[\overrightarrow{T}]$ (q_2 is valid as $v \in W'_i$). We have $q'' \in \mathit{Abinit}(b[\overrightarrow{W'}])$ by Lemma B.8. Thus, by the inference rule $|:_1$, we have $q_1 \xrightarrow{\sigma,\Gamma} q_2$.
       (a) We have $\gamma(q_1) = \gamma(q_2) = |\overrightarrow{W'}| \leq_k cc + (0, \ldots, 1, \ldots, 0) = c$.
       (b) As $\overrightarrow{W'} \subseteq \overrightarrow{V}$ and $qq'_2 \sqsubseteq ss_2$, then $q_1 \sqsubseteq s_1$ and $q_2 \sqsubseteq s_2$.
       (c) Let $q'_2 \in \mathcal{Q}^\sharp$ such that $q_2 \sqsubseteq q'_2 \sqsubseteq s_2$ with $q'_2.val = \overrightarrow{V'}$. We have $q'_2 = (|:_\circ, v, qq''_2)$ with $qq'_2 \sqsubseteq qq''_2 \sqsubseteq ss_2$. Take $q''' \in \mathcal{Q}^\sharp$ satisfying the induction hypothesis, *i.e.* $q'' \sqsubseteq q''' \sqsubseteq q$ and $q''' \xrightarrow{\sigma,([x:=v])\lhd\Gamma} qq''_2$. We have $q''' \in \mathit{Abinit}(b[\overrightarrow{V'}])$ by Lemma B.8. Let $q'_1 = (|:_\circ, \bot, \bot)$ with $q'_1.val = \overrightarrow{V'}$ and $q'_1.pa = a[\overrightarrow{T}]$. We have $q_1 \sqsubseteq q'_1 \sqsubseteq s_1$. And by the rule $|:_1$, $q'_1 \xrightarrow{\sigma,\Gamma} q'_2$.
     - If X is not a parameter, then the proof is similar except that $c = \mu(b[\overrightarrow{T}])$ and we do not augment $\overrightarrow{W}$ with v.
   - Case $|:_2$: $s_1 = (|:_\circ, v, ss_1)$ and $s_2 = (|:_\circ, v, ss_2)$ with $s_1.val = s_2.val = \overrightarrow{V}$, $ss_1 \xrightarrow{\sigma,([x:=v])\lhd\Gamma} ss_2$ and $v \neq \bot$.

- If X is the i-th parameter, then $c = \mu(b[\overrightarrow{T}]) + (0, \ldots, 1, \ldots, 0)$. Let $cc = \mu(ss_2.pa) = \mu(b[\overrightarrow{T}])$. By induction hypothesis, take $qq_1, qq_2 \in \mathcal{Q}^\sharp$ such that $qq_1 \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} qq_2$ with $\gamma(qq_1) = \gamma(qq_2) \leq_k cc$, $qq_1 \sqsubseteq ss_1$, $qq_2 \sqsubseteq ss_2$ and for all $qq_2' \in \mathcal{Q}^\sharp$ with $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$ there exists $qq_1' \in \mathcal{Q}^\sharp$ such that $qq_1 \sqsubseteq qq_1' \sqsubseteq ss_1$ and $qq_1' \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} qq_2'$. Let $\overrightarrow{W} = qq_1.val$ and $\overrightarrow{W'} = \overrightarrow{W} \cup (\emptyset, \ldots, \{v\}, \ldots, \emptyset)$. We have $|\overrightarrow{W'}| \leq_k cc + (0, \ldots, 1, \ldots, 0)$. Let $qq_2' \in \mathcal{Q}^\sharp$ such that $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$ and $qq_2'.val = \overrightarrow{W'}$, which exists by Lemma A.10. By induction hypothesis, take $qq_1' \in \mathcal{Q}^\sharp$ such that $qq_1 \sqsubseteq qq_1' \sqsubseteq ss_1$ and $qq_1' \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} qq_2'$. Let $q_1 = (|:_\circ, v, qq_1')$ with $q_1.pa = a$ and $q_1.val = \overrightarrow{W'}$ and $q_2 = (|:_\circ, v, qq_2')$ with $q_2.pa = a$ and $q_2.val = \overrightarrow{W'}$. By the rule $|:_2$, we have $q_1 \xrightarrow{\sigma,\Gamma} q_2$.
  (a) We have $\gamma(q_1) = \gamma(q_2) = |\overrightarrow{W'}| \leq_k cc + (0, \ldots, 1, \ldots, 0) = c$.
  (b) As $qq_1' \sqsubseteq ss_1$ and $qq_2' \sqsubseteq ss_2$, then $q_1 \sqsubseteq s_1$ and $q_2 \sqsubseteq s_2$.
  (c) Let $q_2' \in \mathcal{Q}^\sharp$ such that $q_2 \sqsubseteq q_2' \sqsubseteq s_2$ with $q_2'.val = \overrightarrow{V'}$. We have $q_2' = (|:_\circ, v, qq_2'')$ with $q_2'.pa = a$ and $qq_2' \sqsubseteq qq_2'' \sqsubseteq ss_2$. Take $qq_1'' \in \mathcal{Q}^\sharp$ satisfying the induction hypothesis, *i.e.* $qq_1' \sqsubseteq qq_1'' \sqsubseteq ss_1$ and $qq_1'' \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} qq_2''$. Let $q_1' = (|:_\circ, v, qq_1'')$ with $q_1'.pa = a$ and $q_1'.val = \overrightarrow{V'}$. We have $q_1 \sqsubseteq q_1' \sqsubseteq s_1$ because $qq_1' \sqsubseteq qq_1'' \sqsubseteq ss_1$. Finally, $q_1' \xrightarrow{\sigma,\Gamma} q_2'$ by the rule $|:_2$.
- If X is not a parameter, then the proof is similar except that $c = \mu(b[\overrightarrow{T}])$ and we do not augment $\overrightarrow{W}$ with v.

7. Inductive case: $a[\overrightarrow{T}] = (|||:_:, n, x, X, b)[\overrightarrow{T}]$. Let $c = \mu(a[\overrightarrow{T}])$. By $|||:_1$, the only rule for $s_1 \xrightarrow{\sigma,\Gamma} s_2$, $s_1 = (|||:_\circ, f)$ and $s_2 = (|||:_\circ, f \Lsh \{v \mapsto ss_2\})$ with $s_1.val = s_2.val = \overrightarrow{V}$ and $f(v) \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} ss_2$.

   - If X is the i-th parameter, then $c = \mu(b[\overrightarrow{T}]) + (0, \ldots, 1, \ldots, 0)$. Let $cc = \mu(ss_2.pa) = \mu(b[\overrightarrow{T}])$. By induction hypothesis, take $q, qq_2 \in \mathcal{Q}^\sharp$ such that $q \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} qq_2$ with $\gamma(q') = \gamma(qq_2) \leq_k cc$, $q \sqsubseteq f(v)$, $qq_2 \sqsubseteq ss_2$ and for all $qq_2' \in \mathcal{Q}^\sharp$ with $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$, there exists $q' \in \mathcal{Q}^\sharp$ such that $q \sqsubseteq q' \sqsubseteq f(v)$ and $q' \xrightarrow{\sigma,(\![x:=v]\!)\triangleleft\Gamma} qq_2'$. Let $\overrightarrow{W} = qq_2.val$ and $\overrightarrow{W'} = \overrightarrow{W} \cup (\emptyset, \ldots, \{v\}, \ldots, \emptyset)$. We have $|\overrightarrow{W'}| \leq_k cc + (0, \ldots, 1, \ldots, 0)$. Let $qq_2' \in \mathcal{Q}^\sharp$ such that $qq_2 \sqsubseteq qq_2' \sqsubseteq ss_2$ and $qq_2'.val = \overrightarrow{W'}$, which exists by Lemma A.10. By induction

hypothesis, take $q' \in \mathcal{Q}^\sharp$ such that $q \sqsubseteq q' \sqsubseteq f(v)$ and $q' \xrightarrow{\sigma,([x:=v])\lhd\Gamma} qq'_2$. Let $q_1 = (|||:_\circ, \{v \mapsto q'\})$ and $q_2 = (|||:_\circ, \{v \mapsto qq'_2\})$, with $q_1.val = q_2.val = \overrightarrow{W'}$ and $q_1.pa = q_2.pa = a[\overrightarrow{T}]$. Thus, by the inference rule $|||:_1$, we have $q_1 \xrightarrow{\sigma,\Gamma} q_2$.

(a) We have $\gamma(q_1) = \gamma(q_2) = |\overrightarrow{W'}| \leq_k cc + (0, \ldots, 1, \ldots, 0) = c$.

(b) As $q' \sqsubseteq f(v)$ and $qq'_2 \sqsubseteq ss_2$, then $q_1 \sqsubseteq s_1$ and $q_2 \sqsubseteq s_2$.

(c) Let $q'_2 \in \mathcal{Q}^\sharp$ such that $q_2 \sqsubseteq q'_2 \sqsubseteq s_2$ with $q'_2.val = \overrightarrow{V'}$. We have $q'_2 = (|||:_\circ, f_2)$ with $f_2 = \{v \mapsto qq''_2 \ldots\}$ and $qq'_2 \sqsubseteq qq''_2 \sqsubseteq ss_2$. Take $q'' \in \mathcal{Q}^\sharp$ satisfying the induction hypothesis, *i.e.* $q' \sqsubseteq q'' \sqsubseteq f(v)$ and $q'' \xrightarrow{\sigma,([x:=v])\lhd\Gamma} qq''_2$. Let $q'_1 = (|||:_\circ, f_1)$ with $f_1 = f_2 \lhd \{v \mapsto q''\}$, $q'_1.val = \overrightarrow{V'}$ and $q'_1.pa = a[\overrightarrow{T}]$. Trivially, $q_1 \sqsubseteq q'_1$. For all $v' \in dom(f_2)$ such that $v' \neq v$, $f_2(v') \sqsubseteq f(v')$ and $q'' \sqsubseteq f(v)$. Thus, for all $v' \in dom(f_1)$, $f_1(v') \sqsubseteq f(v')$. Consequently, $q'_1 \sqsubseteq s_1$. Finally, by the rule $|||:_1$, $q'_1 \xrightarrow{\sigma,\Gamma} q'_2$.

- If X is not a parameter, then $c = |X| \cdot \mu(b[\overrightarrow{T}])$. Let $cc = \mu(ss_2.pa) = \mu(b[\overrightarrow{T}])$. By induction hypothesis, take $q, qq_2 \in \mathcal{Q}^\sharp$ such that $q \xrightarrow{\sigma,([x:=v])\lhd\Gamma} qq_2$ with $\gamma(q') = \gamma(qq_2) \leq_k cc$, $q \sqsubseteq f(v)$, $qq_2 \sqsubseteq ss_2$ and for all $qq'_2 \in \mathcal{Q}^\sharp$ with $qq_2 \sqsubseteq qq'_2 \sqsubseteq ss_2$, there exists $q' \in \mathcal{Q}^\sharp$ such that $q \sqsubseteq q' \sqsubseteq f(v)$ and $q' \xrightarrow{\sigma,([x:=v])\lhd\Gamma} qq'_2$. Besides, by Lemma A.11, for all $ss_{1,i} \in ran(f)$ such that $ss_{1,i} \neq ss_2$, take $qq_{1,i} \in \mathcal{Q}^\sharp$ such that $\gamma(qq_{1,i}) \leq_k cc$ and $qq_{1,i} \sqsubseteq ss_{1,i}$. Let $\overrightarrow{W} = \bigcup_i qq_{1,i}.val \cup qq_2.val \subseteq \overrightarrow{V}$. We have $|\overrightarrow{W}| \leq_k \sum_i \gamma(qq_{1,i}) + \gamma(qq_2) \leq_k |X| \cdot cc$. Take $qq'_2 \in \mathcal{Q}^\sharp$ such that $qq_2 \sqsubseteq qq'_2 \sqsubseteq ss_2$ and $qq'_2.val = \overrightarrow{W}$ by Lemma A.10. Likewise, for all $qq_{1,i}$, take $qq'_{1,i} \in \mathcal{Q}^\sharp$ such that $qq_{1,i} \sqsubseteq qq'_{1,i} \sqsubseteq ss_{1,i}$ and $qq'_{1,i}.val = \overrightarrow{W}$. By the induction hypothesis, take $q' \in \mathcal{Q}^\sharp$ such that $q \sqsubseteq q' \sqsubseteq f(v)$ and $q' \xrightarrow{\sigma,([x:=v])\lhd\Gamma} qq'_2$. Let $q_1 = (|||:_\circ, f_1)$ and $q_2 = (|||:_\circ, f_2)$, with $q_1.val = q_2.val = \overrightarrow{W}$, $q_1.pa = q_2.pa = a[\overrightarrow{T}]$ and where f_1, f_2 are such that $dom(f_1) = dom(f_2) = dom(f)$, $f_1(f^{-1}(ss_{1,i})) = f_2(f^{-1}(ss_{1,i})) = qq'_{1,i}$, $f_1(v) = q'$ and $f_2(v) = qq'_2$. Thus, by the inference rule $|||:_1$, we have $q_1 \xrightarrow{\sigma,\Gamma} q_2$.

  (a) We have $\gamma(q_1) = \gamma(q_2) = |\overrightarrow{W}| \leq_k |X| \cdot cc = c$.

  (b) As $qq'_1 \sqsubseteq ss_1$, $qq'_2 \sqsubseteq ss_2$ and for all i $qq'_{1,i} \sqsubseteq ss_{1,i}$ then $q_1 \sqsubseteq s_1$ and

$q_2 \sqsubseteq s_2$.

(c) Let $q'_2 \in \mathcal{Q}^\sharp$ such that $q_2 \sqsubseteq q'_2 \sqsubseteq s_2$ with $q'_2.val = \overrightarrow{V'}$. We have $q'_2 = (|||:_\circ, f'_2)$ with $f'_2 = \{v \mapsto qq''_2 \ldots\}$ and $qq'_2 \sqsubseteq qq''_2 \sqsubseteq ss_2$. Take $q'' \in \mathcal{Q}^\sharp$ satisfying the induction hypothesis, *i.e.* $q' \sqsubseteq q'' \sqsubseteq f(v)$ and $q'' \xrightarrow{\sigma,([x:=v])\lhd\Gamma} qq''_2$. Let $q'_1 = (|||:_\circ, f'_1)$ with $f'_1 = f'_2 \lhd \{v \mapsto q''\}$, $q'_1.val = \overrightarrow{V'}$ and $q'_1.pa = a[\overrightarrow{T}]$. For all $v' \in dom(f)$ such that $v' \neq v$, $f_1(v') = f_2(v') \sqsubseteq f'_2(v') = f'_1(v') \sqsubseteq f(v')$ and $q' \sqsubseteq q'' \sqsubseteq f(v)$. Thus, for all $v' \in dom(f)$, $f_1(v') \sqsubseteq f'_1(v') \sqsubseteq f(v')$. Consequently, $q_1 \sqsubseteq q'_1 \sqsubseteq s_1$. Finally, by the rule $|||:_1$, $q'_1 \xrightarrow{\sigma,\Gamma} q'_2$.

□

Lemma A.13. Let $a[\overrightarrow{T}]$ be a PASTD. Let $c \in \mathbb{N}^k$ such that $c = \mu(a[\overrightarrow{T}])$. For all $s_1, s_2 \in \mathcal{Q}^\sharp$ such that $s_1.pa = s_2.pa = a[\overrightarrow{T}]$ and $s_1 \xrightarrow{\sigma,\Gamma} s_2$, there exist $q_1, q_2 \in \mathcal{Q}^\sharp$ such that $q_1.pa = q_2.pa = a[\overrightarrow{T}]$ and $q_1 \xrightarrow{\sigma,\Gamma} q_2$ and:

1. $\gamma(q_1) = \gamma(q_2) \leq_k c$, and
2. $q_1 \preceq s_1$ and $q_2 \preceq s_2$, and
3. for all $q'_2 \in \mathcal{Q}^\sharp$ such that $q_2 \preceq q'_2 \preceq s_2$, there exists $q'_1 \in \mathcal{Q}^\sharp$ such that $q_1 \preceq q'_1 \preceq s_1$ and $q'_1 \xrightarrow{\sigma',\Gamma'} q'_2$, where σ' and Γ' are the renamed event and environment with regards to the rename function ξ such that $q_2 \sqsubseteq \xi(q'_2)$.

Proof. Let $a[\overrightarrow{T}]$ be a PASTD. Let $c \in \mathbb{N}^k$ such that $c = \mu(a[\overrightarrow{T}])$. Let $s_1, s_2 \in \mathcal{Q}^\sharp$ such that $s_1.pa = s_2.pa = a[\overrightarrow{T}]$ and $s_1 \xrightarrow{\sigma,\Gamma} s_2$. Take $q_1, q_2 \in \mathcal{Q}^\sharp$ verifying Lemma B.9. We have $q_1.pa = q_2.pa = a[\overrightarrow{T}]$ and $q_1 \xrightarrow{\sigma,\Gamma} q_2$.

1. We have $\gamma(q_1) = \gamma(q_2) \leq_k c$.
2. We have $q_1 \sqsubseteq s_1 \implies q_1 \preceq s_1$ and $q_2 \sqsubseteq s_2 \implies q_2 \preceq s_2$.
3. Let $q'_2 \in \mathcal{Q}^\sharp$ such that $q_2 \preceq q'_2 \preceq s_2$. Let $q''_2 = \xi(q'_2)$ such that $q_2 \sqsubseteq q''_2 \sqsubseteq s_2$ by Definition A.16, with ξ an adequate rename function (it exists because $q_2.val \subseteq s_2.val$). Then, by Lemma B.9, take $q''_1 \in \mathcal{Q}^\sharp$ such that $q_1 \sqsubseteq q''_1 \sqsubseteq s_1$ and $q''_1 \xrightarrow{\sigma,\Gamma} q''_2$. Let $q'_1 \in \mathcal{Q}^\sharp$ such that $q''_1 = \xi(q'_1)$. We have $q_1 \sqsubseteq q''_1 \sqsubseteq s_1$. And by

B.5. Proofs of Section A.6

Lemma A.8, $q_1' \xrightarrow{\sigma',\Gamma'} q_2'$, where σ' and Γ' are the renamed event and environment with regards to the rename function ξ^{-1}.

□