UNIVERSITÉ DU QUÉBEC À MONTRÉAL

SPÉCIFICATION, VALIDATION ET SATISFIABILITÉ DE CONTRAINTES HYBRIDES PAR RÉDUCTION À LA LOGIQUE TEMPORELLE

THÈSE
PRÉSENTÉE
COMME EXIGENCE PARTIELLE
DU DOCTORAT EN INFORMATIQUE

PAR
SYLVAIN HALLÉ

NOVEMBRE 2008

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de cette thèse se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.01-2006). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

REMERCIEMENTS

La poursuite de ce doctorat fut une aventure qui m'a tout à la fois pris et donné plus que je ne m'y attendais. Je suis redevable de son aboutissement à la contribution particulière d'un petit nombre de personnes que je tiens ici à remercier expressément.

Il va sans dire que ce projet ne serait pas devenu ce qu'il est sans le concours de Roger Villemaire, professeur au Département d'informatique de l'UQAM, qui m'a accompagné dans mon travail durant plus de cinq ans, d'abord en tant que directeur de mon projet de maîtrise, puis de doctorat. Bien plus qu'un superviseur, j'ai trouvé en lui un mentor ; j'espère que la fin de ce doctorat ne sera pas la fin de notre enrichissante collaboration.

Merci à Omar Cherkaoui, également professeur au Département d'informatique de l'UQAM, dont la collaboration nous a ouvert des portes à de nombreuses reprises et qui a su enrichir nos travaux de motivations concrètes tirées de la pratique. Je salue aussi le travail de mes collègues et amis Rudy Deca, Éric Wenaas, Jérôme Tremblay, Boubker Ghandour, ainsi que de Daniel Puche, qui ont tous indirectement collaboré à la rédaction de cette thèse en participant à la préparation des articles qu'elle contient.

Je dois enfin souligner que la réalisation de ce projet fut rendue possible grâce à la contribution financière du Conseil de recherche en sciences naturelles et en génie du Canada, de l'Institut des sciences mathématiques, de Cisco Systems ainsi que du Laboratoire de téléinformatique et réseaux de l'UQAM.

Sylvain Hallé
Montréal, 14 février 2008

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

Depuis quelques années, de nombreux champs de l'informatique ont été transformés par l'introduction d'un nouvelle vision de la conception et de l'utilisation d'un système, appelée approche *déclarative*. Contrairement à l'approche dite *impérative*, qui consiste à décrire au moyen d'un langage formel les opérations à effectuer pour obtenir un résultat, l'approche déclarative suggère plutôt de décrire le résultat désiré, sans spécifier comment ce « but » doit être atteint. L'approche déclarative peut être vue comme le prolongement d'une tendance ayant cours depuis les débuts de l'informatique et visant à résoudre des problèmes en manipulant des concepts d'un niveau d'abstraction toujours plus élevé.

Le passage à un paradigme déclaratif pose cependant certains problèmes : les outils actuels sont peu appropriés à une utilisation déclarative. On identifie trois questions fondamentales qui doivent être résolues pour souscrire à ce nouveau paradigme : l'expression de contraintes dans un langage formel, la validation de ces contraintes sur une structure, et enfin la construction d'une structure satisfaisant une contrainte donnée.

Cette thèse étudie ces trois problèmes selon l'angle de la logique mathématique. On verra qu'en utilisant une logique comme fondement formel d'un langage de « buts », les questions de validation et de construction d'une structure se transposent en deux questions mathématiques, le *model checking* et la satisfiabilité, qui sont fondamentales et largement étudiées. En utilisant comme motivation deux contextes concrets, la gestion de réseaux et les architectures orientées services, le travail montrera qu'il est possible d'utiliser la logique mathématique pour décrire, vérifier et construire des configurations de réseaux ou des compositions de services web.

L'aboutissement de la recherche consiste en le développement de la logique CTL-FO$^+$, permettant d'exprimer des contraintes sur les données, sur la séquences des opérations d'un système, ainsi que des contraintes dites « hybrides ». Une réduction de CTL-FO$^+$ à la logique temporelle CTL permet de réutiliser de manière efficace des outils de vérification existants.

Mots-clés : méthodes formelles, services web, réseaux

ABSTRACT

In recent years, numerous fields of computer science have been transformed by the introduction of a new, *declarative* approach to the design and use of a system. While the *imperative* approach requires the description, by means of a formal language, of the operations to be executed in order to produce a desired result, the declarative approach rather suggests to describe the result without specifying how this "goal" should be reached. The declarative approach can be seen as the continuation of a movement towards higher levels of abstraction to solve problems.

Since most current tools were developed with an imperative view, the transition to the declarative paradigm requires that three fundamental problems be solved: expressing constraints in a formal language, validating these constraints on a suitable structure, and building a structure satisfying a given constraint.

This thesis studies these three problems using mathematical logic. By using logic as a formal foundation for a "goal" language, the questions of validating and building a structure according to a constraint can be transposed into two well-studied mathematical questions, the *model checking* and the *satisfiability* of a formula. The project is motivated by real-world examples taken from the fields of network management and service oriented architectures. We shall see that mathematical logic can be used in these contexts to describe, validate and build network configurations or web service compositions.

The conclusion of this research is the development of the logic CTL-FO$^+$, which allows to express constraints on data, the sequence of operations of a system, as well as hybrid, "data-aware" constraints. A reduction of CTL-FO$^+$ to the temporal logic CTL allows to efficiently leverage existing verification tools.

Keywords: formal methods, web services, networks

INTRODUCTION

Chacun des chapitres de cette thèse est intégralement tiré d'articles de conférences et de journaux soumis ou publiés au cours de ce projet de recherche :

Chapitre I : Hallé, S., Villemaire, R., Cherkaoui, O. (2006). CTL Model Checking for Labelled Tree Queries. In *Proceedings of the 13th International Symposium on Temporal Representation and Reasoning (TIME 2006)*, IEEE Computer Society Press, 27-35.

Chapitre II : Hallé, S., Wenaas, E., Villemaire, R., Cherkaoui, O. (2006). Self-configuration of Network Devices with Configuration Logic. In *Proceedings of Autonomic Networking, First International IFIP TC6 Conference (AN 2006)*, Springer Verlag : Lecture Notes in Computer Science 4195, 36-49.

Chapitre III : Hallé, S., Villemaire, R. (2008). Satisfying a Fragment of XQuery by Branching-Time Reduction. Soumis au *15th International Symposium on Temporal Representation and Reasoning (TIME 2008)*, janvier 2008.

Chapitre IV : Hallé, S., Deca, R., Cherkaoui, O., Villemaire, R., Puche , D. (2007). Modelling the Temporal Aspects of Network Configuration. In *Network Control and Engineering for QoS, Security and Mobility (NetCon 2005)*, Springer Verlag : IFIP International Federation for Information Processing, 229, 269-282.

Chapitre V : Hallé, S., Deca, R., Cherkaoui, O., Villemaire, R., Puche, D. (2006). Sequential Dependencies in Configuration Operations. In *Actes du 7^{e} Colloque francophone de Gestion de Réseaux et de Services (GRES 2006)*, 112-123.

Chapitre VI : Hallé, S., Villemaire, R., Cherkaoui, O. (2008). Specifying and Validating Data-Aware Temporal Web Service Properties Soumis aux *IEEE Transactions on Software Engineering*, février 2008.

An English translation of this introduction can be found on page 13.

Historiquement, les progrès de l'informatique ont été jalonnés par le développement successif de niveaux d'abstraction permettant de décrire les tâches à accomplir par un système. En programmation, par exemple, l'assembleur, puis les langages de programmation structurée, ont permis à l'utilisateur de s'éloigner progressivement des considérations techniques et matérielles pour donner plutôt des descriptions logiques du travail à effectuer.

Cette progression vers des descriptions de plus en plus abstraites poursuit présentement son cours. Jusqu'à tout récemment, le développement et l'utilisation d'un système utilisait une approche dite *impérative* : bien que les langages et les formalismes employés aient évolué vers un niveau croissant d'abstraction, le principe de base consistait toujours à décrire les tâches à effectuer dans le but d'obtenir un certain résultat. Or, de nombreux champs de l'informatique ont été marqués dans les dernières années par l'avènement d'une nouvelle approche dite *déclarative.*

Dans un paradigme déclaratif, ce sont les résultats souhaités qui sont décrits au moyen d'un langage. La manière dont ces résultats sont obtenus, les étapes concrètes effectivement suivies pour produire la réponse attendue ne sont pas spécifiées. Le travail de l'utilisateur ne consiste plus à concevoir et à exprimer une séquence de tâches que le système doit réaliser, mais bien à décrire de manière aussi précise que possible les données ou les fonctionnalités qu'il cherche à obtenir du système.[1] L'impératif « ordonne », le déclaratif « demande ».

Le langage de programmation Prolog (Bratko, 2000), datant des années 70, constitue l'un des premiers prototypes de l'approche déclarative. L'application de cette approche à d'autres domaines est cependant relativement récente et est appelée à transformer la manière dont on conçoit et réalise un système d'information, quel qu'il soit. Voyons

1. Le domaine où l'approche déclarative est présentement la plus aboutie est sans doute celui des bases de données. L'avènement de langages de requête comme SQL fut en son temps une révolution en permettant à l'utilisateur de décrire les données qu'il souhaite obtenir sans se soucier de la manière dont ces données sont stockées ou récupérées.

comment deux champs particuliers de l'informatique s'adaptent à l'approche déclarative.

La gestion des réseaux

Qu'ils soient publics, privés ou d'entreprise, les réseaux sont devenus la pierre angulaire de nombreuses applications informatiques. Au départ limités à quelques fonctionnalités de base (courriel, transfert de fichiers), on y voit maintenant foisonner un grand nombre de *services* : voix sur IP, réseaux privés virtuels (VPN), communautés *peer-to-peer*. L'apparition de telles fonctionnalités a considérablement accru la complexité de la gestion des équipements responsables de leur mise en œuvre, principalement les routeurs (Pujolle *et al.*, 2005).

En effet, le déploiement et la gestion d'un service réseau consiste en la modification de paramètres de configuration d'un ou plusieurs équipements dans le but d'implémenter la fonctionnalité souhaitée. Chaque service possédant ses propres demandes en termes de configuration, et l'interaction entre ceux-ci pouvant provoquer des effets secondaires multiples et parfois subtils, le travail de l'ingénieur réseaux devient rapidement un casse-tête de gestion permanent.

Dans ce contexte, l'application d'une approche déclarative s'avère hautement souhaitable. Elle consiste, pour un ingénieur ou un gestionnaire de réseaux, à décrire les services ou les fonctionnalités qu'il souhaite mettre en œuvre, ainsi que les modalités de consommation de ces mêmes services ou fonctionnalités par ses utilisateurs. On cherche à ce que ces descriptions soit d'abord formulées sous forme de *politiques*, utilisant un langage de haut niveau qui soit compréhensible par les gestionnaires d'une organisation. Ces politiques peuvent à leur tour se traduire en une suite de directives ou de contraintes plus concrètes que l'ingénieur réseaux peut appliquer. Ce principe est à la base du *policy-based management* (Sloman *et al.*, 2001).

L'architecture orientée services

Un second exemple nous est fourni par le principe de design général de l'architecture orientée services (*Service Oriented Architecture* ou SOA) (Alonso *et al.*, 2004). Dans ce contexte, les ressources et les fonctionnalités d'un système sont exposées aux consommateurs potentiels sous la forme d'entités appelées *services*. Un consommateur communique avec un service donné à travers son interface par l'échange d'éléments d'information (les *messages*), selon un patron d'interactions nommé « *workflow* ». Les fonctionnalités de plusieurs services peuvent être mises à profit en les faisant interagir, de manière à former un nouveau service « composé » dont le fonctionnement constitue une valeur ajoutée par rapport à chacun des services originaux.

Le développement SOA est devenu une tendance importante dans le développement de systèmes avec l'avènement des services web en raison de son caractère général et flexible. Cependant, l'approche modulaire et compositionnelle de l'architecture SOA ne se limite pas qu'aux services web : en 2005, on rapportait qu'au moins 70% des nouveaux logiciels, toutes catégories confondues, étaient conçus selon une infrastructure basée sur la composition de modules communicants (Wang *et al.*, 2006; Zhao et Shen, 2005).

Le domaine des architectures orientées services, bien que relativement récent, tombe également sous l'influence croissante d'une approche déclarative au développement. Les architectures SOA vivent présentement la révolution de la gestion des processus d'affaires (*business process management* ou BPM). Il s'agit d'une approche descendante (*top-down*), qui vise à décrire d'abord les objectifs de l'entreprise pour ensuite les concrétiser par le développement de services. On reconnaît ici les mêmes principes qui motivent le développement du *policy-based management* en gestion de réseaux ; on y retrouve jusque le concept de politique (Bajaj *et al.*, 2006). Gartner (Jim et Janelle, 2006) prédisait que l'année 2007 serait rétrospectivement vue comme celle où le développement SOA passerait majoritairement sous l'influence du BPM et que cette tendance atteindrait sa

maturité d'ici la fin de la décennie.

Trois problèmes fondamentaux

L'approche déclarative constitue cependant un changement de paradigme important par rapport à la philosophie impérative qui avait cours depuis près d'un demi-siècle. Nous identifions ici trois problèmes importants qui doivent être résolus pour souscrire à la philosophie déclarative.

Tout d'abord, les outils de développement, les langages de programmation, les protocoles de communication et les systèmes d'exploitation actuels sont tous orientés vers un mode d'utilisation impératif et sont peu appropriés à la déclaration. Intuitivement, on pourrait résumer le problème par le fait que le langage que l'on utilise pour décrire un but n'est pas celui par lequel on décrit le travail qui mène à ce but. Un premier problème d'importance est donc le suivant :

Problème 1. *Développer des langages permettant d'exprimer les fonctionnalités ou les résultats souhaités : buts, données, politiques.*

De la même manière que l'approche déclarative transforme la manière dont on exprime un problème, elle transforme également la manière dont on vérifie sa solution. Au débogage et aux tests classiques succède une nouvelle forme de vérification visant à s'assurer que la réponse ou le résultat obtenu satisfait les exigences précédemment déclarées, ce qui en fait notre deuxième problème fondamental :

Problème 2. *Développer des techniques automatiques permettant de vérifier qu'une élément donné (service, configuration) respecte bien les spécifications exprimées.*

Finalement, tel que mentionné plus tôt, l'approche déclarative ne spécifie pas la

manière dont une spécification, une politique ou une requête peut être satisfaite. Notre troisième problème devient ainsi :

Problème 3. *Développer des techniques automatiques permettant de produire un résultat (service, configuration) qui respecte les spécifications exprimées.*

La présente thèse s'attaque à ces trois problèmes en les étudiant selon l'angle de la logique mathématique.

Pour l'utilisation d'une logique

La prémisse de ce travail est la suivante : la logique mathématique constitue une fondation formelle appropriée à la résolution des trois problèmes précédents. En effet, dans son essence, son langage est déclaratif : par le biais de divers connecteurs logiques, on peut composer des énoncés qui deviennent soit vrais, soit faux en fonction du contexte auquel on se réfère pour les interpréter. Il paraît donc naturel de construire sur une base logique les langages de spécification mis en évidence par le problème 1. De plus, il existe un grand nombre de formalismes logiques adaptés à diverses situations : logiques classiques propositionnelle et du premier ordre, logique sur les structures arborescentes, logiques temporelles permettant d'exprimer des rapports de séquentialité entre des événements. L'arsenal logique semble donc suffisamment vaste pour « déclarer » un grand nombre d'exigences.

L'utilisation de la logique présente cependant un autre avantage non négligeable. En choisissant un formalisme logique comme langage de déclaration, les problèmes 2 et 3 ci-dessus se transposent en deux questions mathématiques importantes et largement étudiées. Le problème 2, qui consiste à vérifier qu'un objet satisfait une spécification, revient alors à vérifier qu'une certaine formule logique est satisfaite par un modèle donné. Ce problème, c'est celui du *model checking*, problème classique et pour lequel il existe

de nombreuses techniques et une foule de résultats théoriques. De la même manière, le problème 3, qui consiste à produire un objet satisfaisant une spécification, revient alors à construire un modèle d'une formule logique donnée, si un tel modèle existe. Il ne s'agit alors que d'une reformulation de la question de la *satisfiabilité* d'une logique, question elle aussi grandement étudiée sur des bases théoriques et pratiques.

Il semblerait dès lors le problème résolu. Pour chaque type de spécification, il suffit de choisir le type de logique se prêtant le mieux à sa description, et de traduire tout énoncé de haut niveau en une ou plusieurs formules de cette logique. Dès lors, il suffit d'utiliser les techniques de *model checking* ou de satisfiabilité développées pour cette logique pour résoudre les deux autres problèmes fondamentaux mentionnés précédemment. Le seul travail requis en est donc un d'adaptation. Cette situation idéale n'est malheureusement pas la réalité : il existe certains types de contraintes et d'exigences qui vont au-delà des formalismes logiques connus.

Contraintes statiques, dynamiques et hybrides

Des deux contextes précédents, il est possible de déduire un modèle formel commun, représenté à la figure 1. L'interaction avec un objet (rectangle du haut) s'effectue par une séquence d'opérations émises ou reçues (représentées par les documents au bas de la figure). Chacune de ces opérations est un couple (E, C), où E est une étiquette (représentée par les lettres A à D) et C est un contenu prenant la forme d'une structure arborescente.

Dans le contexte de la configuration réseau, l'étiquette de l'opération peut représenter le nom d'une commande ou d'un bloc de commandes à exécuter ; la structure arborescente représente les paramètres de l'opération, et particulièrement la configuration, ou la portion de configuration sur laquelle agit l'opération. De précédents travaux

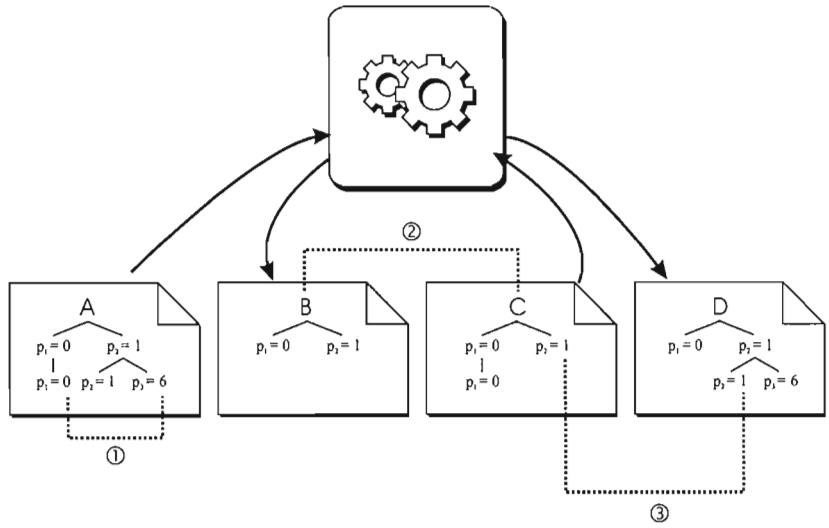


FIGURE 1: Un modèle formel pouvant représenter un système de configuration des équipements réseaux et une architecture orientée services. Les lignes numérotées 1, 2 et 3 représentent respectivement une contrainte statique, dynamique et hybride.

ont montré que de telles structures sont une abstraction adéquate des informations de configuration des équipements réseau et de leurs opérations (Deca *et al.*, 2004; Hallé *et al.*, 2005).

Dans le contexte de l'architecture orientée services, l'étiquette désigne le nom d'un message XML ou d'une opération et la structure arborescente renferme le contenu du message XML. À cet égard, les structures en arbres sont considérées comme une représentation formelle adéquate des documents XML (Cardelli et Ghelli, 2004) dont l'usage dans les architectures orientées services est largement répandu (Fu *et al.*, 2004c).

On distingue trois types de contraintes susceptibles d'être exprimées par le langage formel visé ; la figure 1 nous fournit un exemple de chacune.

Contraintes statiques. Un premier type de contrainte consiste à énoncer une relation d'interdépendance entre différents éléments d'information à l'intérieur d'une même opération : il s'agit de dépendances dites *statiques*, qui se réfèrent à une opération ponc-

tuelle. Ce type de dépendance est désigné par l'arc pointillé « 1 » de la figure 1 : deux paramètres de la structure arborescente d'une opération sont contraints par une relation quelconque. Par exemple, le contexte pourrait exiger que les noeuds $p_1 = 0$ et $p_3 = 6$ possèdent des valeurs identiques, ou encore distinctes.

Pour gérer ce type de contrainte, la logique choisie doit pouvoir exprimer des relations sur des structures arborescentes. En cela, elle doit être similaire à des langages existants tels XPath (Clark et DeRose, 1999) ou XQuery (Boag *et al.*, 2005).

Contraintes dynamiques. Un second type de contrainte consiste à énoncer une relation d'interdépendance sur l'ordonnancement de deux opérations : on parle alors de dépendances *dynamiques* qui imposent une relation entre deux moments distincts dans le temps. Ce type de dépendance est désigné par l'arc pointillé « 2 » de la figure 1 ; par exemple, le contexte pourrait exiger que l'exécution de l'opération « C » doive toujours être précédé de la réception d'une réponse d'étiquette « B ».

Pour gérer ce type de contrainte, la logique choisie doit pouvoir exprimer des relations *temporelles*. Elle doit inclure des fonctions de séquence à l'image des opérateurs temporels des logiques CTL (Emerson et Halpern, 1985) et LTL (Gabbay *et al.*, 1980).

Contraintes hybrides. Les dépendances dites *hybrides* sont en quelque sorte la combinaison des deux types de contraintes précédents : une relation est imposée sur le contenu de deux opérations à deux moments distincts dans le temps, tel que le montre l'arc pointillé « 3 » de la figure 1. On ne peut considérer une telle contrainte comme une simple dépendance dynamique, puisqu'il est nécessaire d'accéder au contenu des opérations et pas seulement d'exprimer leur ordonnancement. De même, on ne peut non plus considérer cette dépendance comme statique, puisque deux opérations distinctes sont en jeu. Les contraintes hybrides représentent donc bel et bien plus que la somme de leurs parties.

Méthodologie

La spécification, la validation et la satisfiabilité efficace de contraintes hybrides est actuellement un problème ouvert. Bien qu'il existe un certain nombre de formalismes adéquats pour la spécification de contraintes statiques et dynamiques, par l'observation précédente, ceux-ci ne peuvent être tout bonnement utilisés « côte à côte » pour couvrir le cas des contraintes hybrides. Un nouveau langage doit être développé à cet effet, qui s'inspirera des formalismes statiques et dynamiques, mais leur permettra en plus d'interagir.

La présente thèse vise donc à répondre aux trois questions fondamentales mentionnées plus tôt, mais en utilisant un langage hybride. Cependant, il serait fâcheux et improductif de réinventer la roue sans pouvoir tirer parti de techniques et d'outils développés pour les logiques existantes et dont le rejeton hybride doit être somme toute assez proche. C'est pourquoi, dès le début du projet, une attention particulière fut portée à la possibilité de « traduire » un problème du nouveau formalisme en un problème de logique classique.

Deux choix s'offraient :

1. convertir la composante dynamique de la logique en un problème de logique sur les arbres et utiliser des outils de cette logique (engin XPath, TQL)
2. convertir la composante arborescente de la logique en un problème de logique temporelle et utiliser des outils de logique temporelle (*model checkers* NuSMV ou SPIN)

En raison du problème de l'explosion des états causé par la vérification de formules temporelles sur des systèmes, il a été jugé plus avantageux de tenter d'exprimer des formules arborescentes en logique temporelle que de chercher à représenter des systèmes de taille

exponentielle par des structures en arbres de type XML. Ce choix a déterminé l'orientation du travail. Par la suite, l'utilisation de la logique temporelle prit une place centrale en devenant le substrat dans lequel les contraintes devaient ultimement se réduire.

Organisation de la thèse

La vérification de contraintes statiques sur des configurations d'équipements réseau a été étudiée dans un travail précédent (Hallé, 2005). Comme l'illustre la figure 2, chaque chapitre de la présente thèse introduit une extension de cette question originale par rapport à au moins un des trois aspects suivants :

1. Le type de contraintes : les chapitres 1 à 3 s'intéressent aux contraintes statiques, les chapitres 4 et 5 traitent des contraintes dynamiques, et le chapitre 6 développe les contraintes hybrides.
2. Le domaine d'application : tandis que les chapitres 2, 4 et 5 sont spécifiques aux contraintes de configuration de réseaux, les chapitres 1, 3 et 6 concernent plus généralement les structures arborescentes de type XML.
3. Le problème étudié : les chapitres 2 et 3 s'intéressent à la satisfiabilité (décidabilité) de la logique, les autres se concentrent sur le problème du *model checking*.

Le chapitre 1 étudie la logique CL, un fragment du langage XQuery permettant d'exprimer des contraintes statiques sur des structures de données arborescentes, et montre comment le *model checking* de CL peut être réduit au *model checking* de la logique temporelle CTL.

Le chapitre 2 montre comment la satisfiabilité de CL peut être utilisée pour générer automatiquement des configurations d'équipements réseaux. Elle illustre cette approche par un exemple tiré des réseaux autonomique, en particulier l'auto-configuration d'un

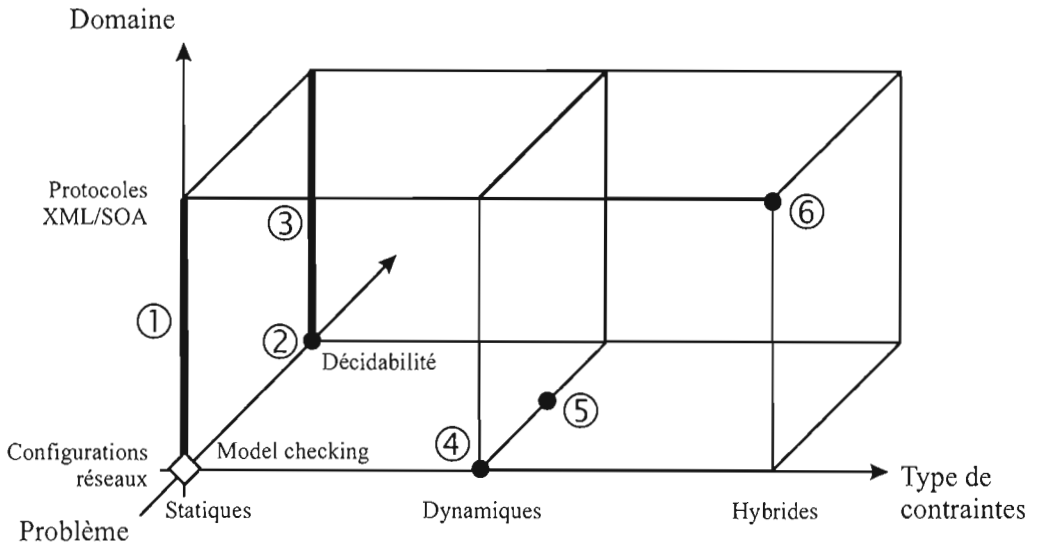


FIGURE 2: L'organisation des chapitres de cette thèse selon les trois composantes du travail.

service de type VLAN dans des commutateurs Cisco.

Le chapitre 3 montre que la satisfiabilité de CL étudiée au chapitre précédent peut à nouveau se réduire à un problème équivalent de la logique temporelle CTL. Pour ce faire, les constructions définies au chapitre 1 peuvent être réutilisées, moyennant certaines adaptations qui sont présentées en détail.

Le chapitre 4 suggère, au moyen d'exemples concrets tirés de la configuration d'équipements réseau, qu'il existe des dépendances séquentielles entre les opérations de configuration d'un service et que ces dépendances doivent être explicitées, modélisées, et vérifiées automatiquement. À cet effet, le concept de « borne » (*milestone*) est introduit et formellement défini.

Le chapitre 5 poursuit cette idée en suggérant que les bornes apparaissent naturellement dans les opérations de configuration lorsque les contraintes d'ordonnancement sont exprimées par des formules de logique temporelle LTL. Il est également démontré

comment la génération automatique de séquences d'opérations de configuration peut être automatisée en résolvant le problème de la satisfiabilité de LTL.

Enfin, le chapitre synthétise les résultats précédents en présentant CTL-FO$^+$, un formalisme logique permettant l'expression de contraintes statiques, dynamiques, de même que hybrides ; des contraintes tirées de l'utilisation de services web dans des contextes concrets ont illustré l'approche. En particulier, l'article se concentre sur une manière efficace de traduire le problème du *model checking* de CTL-FO$^+$ en CTL. La construction développée rend possible la vérification de contraintes inaccessibles par une traduction « naïve » en CTL et s'avère donc une approche prometteuse.

INTRODUCTION – ENGLISH SUMMARY

Historically, the advancement of computer science has been marked by the development of successive abstractions in the description of the tasks to be accomplished by a system. As an example, in the field of programming, the advent of the assembly language, followed by structured programming languages, allowed users to progressively distance themselves from technical and hardware issues to concentrate on a logical description of the work to be done.

This evolution towards more abstract descriptions continues to this day. Until recently, the development and use of a system followed an approach which could be called *imperative*: although the languages and the formalisms evolved towards more abstract concepts, the basic principle always consisted of describing the tasks to be carried out in order to produce a desired result. However, a number of fields in computer science have been transformed in recent years with the advent of a new approach called *declarative*.

In a declarative paradigm, the results, rather than the tasks, are described by means of a language. The actual way in which these results should be obtained are not specified. The new job for a user is no longer to design and express a sequence of steps that the system should follow, but rather to describe as precisely as possible the data or functionalities expected from the system [2]. While the imperative approach "commands", the declarative approach "demands".

The Prolog programming language (Bratko, 2000) dates from the 1970s and constitutes one of the first prototypes of the declarative approach in Artificial Intelligence. However, its application to other domains is relatively recent and is expected to transform the way in which information systems are designed and implemented. Let us

2. The declarative approach is probably most apparent is the domain of databases. The advent of query languages like SQL represented a real progress by allowing users to describe the data to be extracted, regardless of the physical implementation of a particular system.

examine two specific domains where the declarative approach has been used.

Network Management

Computer networks have become over the years the central element of numerous applications. While they were originally limited to a few basic functionalities (e-mail, file transfer), a plethora of new *services* are adding to the value of these networks: voice over IP, virtual private networks (VPNs), peer-to-peer communities. The growth of these functionalities over the years considerably increased the complexity of managing the devices responsible for their proper functioning, in particular network routers (Pujolle *et al.*, 2005).

The deployment and management of a network service calls for the modification of configuration parameters on one or more devices to implement the desired functionality. Each service has its own configuration requirements, and the interaction between services hosted on the same device can trigger numerous and subtle side effects; therefore the task of the network engineer rapidly becomes a constant management puzzle.

In this context, the application of a declarative approach becomes highly desirable. For a network manager, it consists of describing the services or the functionalities available on a network, in conjunction with the modalities for consuming these services or functionalities by the users (customers). Ideally, these descriptions should first be formulated using a high level language as *policies* that can be understood by the managers of an organization. These policies can then be translated into a set of more concrete directions or constraints that can be applied by the network engineer. This principle is called *policy-based management* (Sloman *et al.*, 2001).

Service Oriented Architectures

A second example is provided by the general design principle called Service Oriented Architecture (SOA) (Alonso *et al.*, 2004). In this context, the resources and functionalities of a system are exposed to potential consumers in the form of entities called *services*. A consumer communicates with the service through its interface by exchanging *messages*, according to an interaction pattern called "workflow". The functionalities of multiple services can interact to form a new, "composite" service which represents an added value with respect to each of the original services.

Because of its general and flexible nature, SOA has become an even more important concept in systems development with the advent of web services: in 2005, it was reported that at least 70% of all new software was based on an infrastructure of composite, communicating modules (Wang *et al.*, 2006; Zhao et Shen, 2005).

Although relatively recent, SOA is also under the increasing influence of the declarative approach. Business process management (BPM) is a top-down approach which aims at describing the goals of an enterprise to concretize them into communicating units (services). The same principles that were highlighted in policy-based management are at play, down to the concept of policy which also finds an equivalent here (Bajaj *et al.*, 2006). The Gartner Group (Jim et Janelle, 2006) forecasted that 2007 would be the year where SOA would fall decisively under the dominance of BPM, and that this tendency would reach its full maturity by the end of the decade.

Three Fundamental Problems

The declarative approach represents an important change of paradigm with respect to the imperative philosophy which was commonplace in the past fifty years. We can

identify three important problems that must be solved to fully subscribe to the declarative paradigm.

First, it should be noted that the existing development tools, programming languages, communication protocols, and operating systems are designed mostly in an imperative view, and are therefore only partially appropriate for declaration. Intuitively, we can summarize this problem by the fact that the language used to describe a goal is different from the language used to describe the tasks that lead to this goal. This leads to the first fundamental problem:

Problem 1. *Develop languages allowing for the expression of the desired functionalities or results: goals, data, policies.*

In the same way that a declarative approach transforms the way in which a problem is expressed, it transforms the way in which the solution is verified. Classical tests and debugging give way to a new form of verification whose goal is to ensure that a result satisfies the requirements previously declared. This makes our second fundamental problem:

Problem 2. *Develop automated techniques to verify that a given element (service, configuration) satisfies the specifications.*

Finally, as previously mentioned, the declarative approach does not specify the way in which a specification a policy or a query can be satisfied. The third fundamental problem is thus:

Problem 3. *Develop automated techniques to produce a result (service, configuration) that satisfies the specifications.*

This thesis addresses these three problems on the angle of mathematical logic.

Using a Logic

The premise of this work is the following: mathematical logic represents an appropriate formal foundation for the resolution of the three previously mentioned problems. Indeed, mathematical logic is essentially a declarative language: by means of logical connectives, it is possible to build statements that become true or false depending on the context on which they are interpreted. Hence, it seems natural to build on a logical basis the specification languages needed by Problem 1. Moreover, there exists a large number of logical formalisms adapted to diverse situations: classical propositional and first-order logic, logic on tree tree structures, temporal logics. The logical toolbox seems complete enough to "declare" a fair number of requirements.

The use of logic also brings an important consequence. By choosing a logical formalism as a declaration (or requirements) language, Problems 2 and 3 can be transposed into two important and widely studied mathematical questions. Problem 2, which consists of verifying that an object satisfies a specification, becomes the verification that some logical formula is satisfied by a given model. This is called *model checking* and is a classical problem for which numerous techniques and theoretical results exist. In the same way, Problem 3, which consists in producing an object satisfying a specification, becomes the problem of building a model for a given logical formula, if such a model exists. This is nothing but a reformulation of the question of *satisfiability* of a logic which has also been investigated (Grädel *et al.*, 2007).

It looks as if the problem is already solved. For every type of specification, it suffices to choose the type of logic most appropriate to describe it, and to translate each high-level statement into one or more logical formulæ, solving Problem 1. One simply has to use model checking or satisfiability techniques developed for that logic to solve Problems 2 and 3. The only work that really needs to be done is to devise adaptors between high-level statements and models, and lower-level logical constructions. However, reality is

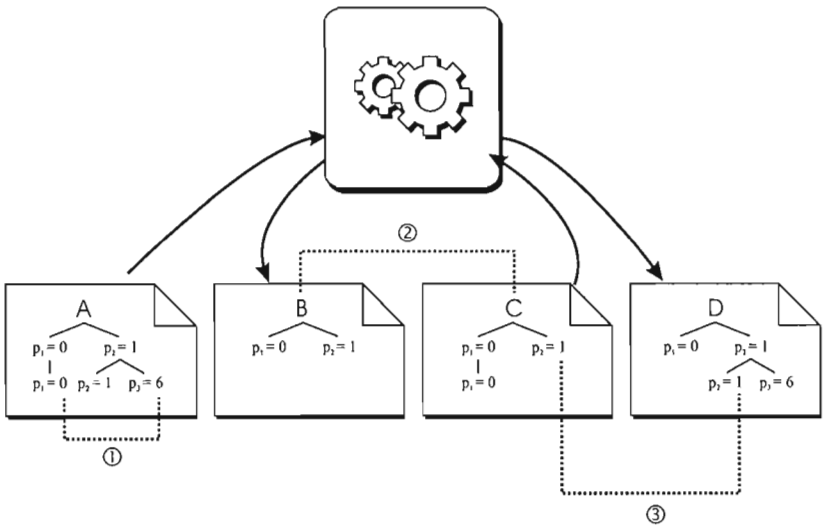


Figure 3: A formal model to represent a network device configuration system and a service oriented architecture. Arrows numbered 1, 2, 3 respectively represent static, dynamic and hybrid constraints.

more complex: there exists some constraints and requirements that go beyond existing logical formalisms.

Static, Dynamic and Hybrid Constraints

From the previous two contexts, it is possible to deduce a common formal model, represented in Figure 3. The interaction with an object (upper box) is carried by a sequence of operations sent or received (represented by the documents at the bottom). Each of these operations is a tuple (E, C), where E is a label (represented by letters A-D), and C is a "content" represented by a tree structure.

In the context of network configurations, the operation label can represent the name of a command or of a block of commands to execute; the tree structure represents the parameters of the operation, and in particular the configuration or the portion of configuration the operation acts upon. Previous works have shown that such structures

are an appropriate abstraction of configuration information and operations in network devices (Deca *et al.*, 2004; Hallé *et al.*, 2005).

In the context of service oriented architecture, the label represents the name of an XML message or of an operation, and the tree structure represents the content of the XML message. In this respect, tree structures are considered an adequate formal representation of XML documents (Cardelli et Ghelli, 2004) which are commonly used in SOA (Fu *et al.*, 2004c).

We can distinguish three types of constraints that can be potentially expressed in the target formal language; Figure 3 provides an example of each.

Static constraints. A first type of constraint consists of an interdependence relation between multiple data elements inside a single operation or state of the system: we call these dependencies *static*, which refer to one operation at one moment in time. This relation is described by the arrow labelled "1" in Figure 3: two parameters in the tree structure of an operation are constrained by an arbitrary relation. For example, the context could require that nodes $p_1 = 0$ and $p_3 = 6$ have identical (or distinct) values.

To handle this type of constraint, the chosen logic must express relations on tree structures. Therefore, it must be similar to existing languages such as XPath (Clark et DeRose, 1999) or XQuery (Boag *et al.*, 2005).

Dynamic constraints. A second type of constraint consists of an interdependence relation on the sequence of two operations: the constraint is hence called *dynamic*, since it imposes a relation between two different moments in time. This relation is described by the arrow labelled "2" in Figure 3; for example, the context could require that the execution of operation "C" always be preceded by the reception of a response labelled "B".

To handle this type of constraint, the chosen logic must express *temporal* relations.

It must include sequencing functions mirroring the temporal operators of logics like CTL (Emerson et Halpern, 1985) or LTL (Gabbay *et al.*, 1980).

Hybrid constraints. So-called *hybrid* constraints are the combination of the two previous types of constraints: a relation is imposed on the data content of two operations in two distinct moments in time, as shown by the arrow labelled "3" in Figure 3. It cannot be considered as a simple static constraint either, since two distinct operations are involved. In the same way, it cannot be considered as a simple dynamic constraint, since the content of operations is part of the constraint, and not only the sequencing of these operations. Hybrid constraints, also called "data-aware" temporal constraints for that reason, are therefore more than the sum of their parts.

Methodology

The specification, efficient validation and satisfiability of hybrid constraints is currently an open problem. Although there exists a number of adequate formalisms to specify static and dynamic constraints, by the previous observation, they cannot simply be used "side by side" to cover the case of hybrid constraints. A new language must be developed to this end; this language should be inspired from both static and dynamic formalisms, but should also allow them to interact.

The goal of this thesis is therefore to address the previous three fundamental problems using a hybrid language. However, it would be counter-productive to reinvent the wheel without leveraging existing techniques and tools developed for existing logics. This is even more true that the hybrid "offspring" probably bears a large number of similarities with its "parents". Therefore, from the beginning of the project, a special attention has been taken to translate a problem of the new formalism into a classical logical problem.

Two choices were possible:

1. convert the dynamic component of the logic into a tree logic problem and use tools for these logics (e.g. XPath engine, TQL)
2. convert the tree component of the logic into a temporal logic problem and use temporal logic tools (e.g. NuSMV or SPIN model checkers)

Because of the state explosion problem caused by the verification of temporal formulæ on systems, it was judged more efficient to try to express tree formulæ in terms of temporal logics, rather than to try to represent systems of exponential size in XML tree structures. This choice determined the orientation of the work: the use of temporal logic took a central place by becoming the basis in which the constraints were ultimately to be translated.

Organization Of This Thesis

A French survey of relevant literature follows this introduction. This survey is simply a collection of related work mentioned in each chapter; for this reason, it has not been translated.

The verification of static constraints on network device configurations has been studied in a previous work (Hallé, 2005). As Figure 4 shows, each chapter of this thesis presents an extension of this initial question with respect to at least one of the following three aspects:

1. Type of constraints: Chapters 1 to 3 concentrate on static constraints, Chapters 4 and 5 describe dynamic constraints, and Chapter 6 develops hybrid constraints.
2. Domain of application: while Chapters 2, 4 and 5 are specific to network device

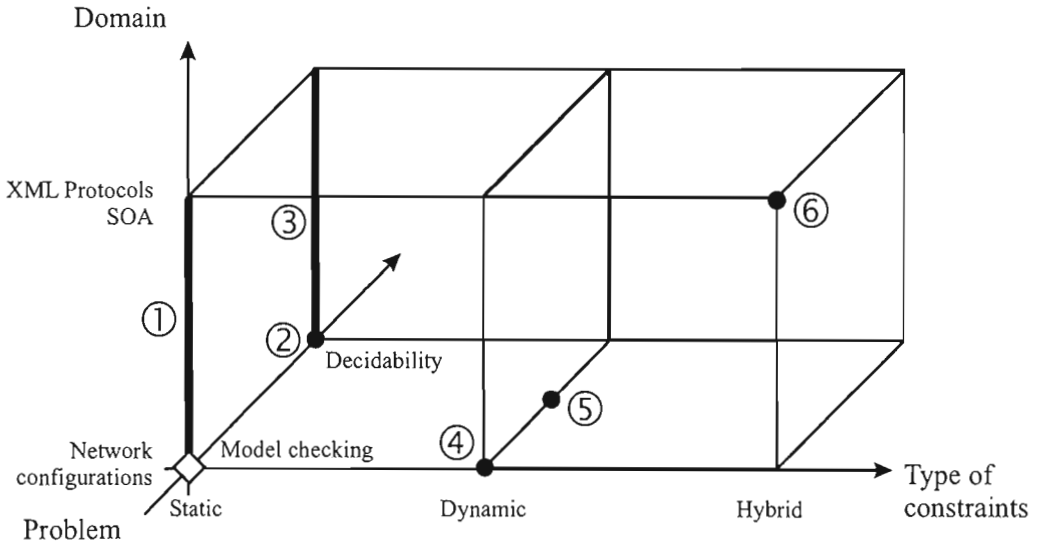


Figure 4: The organization of the chapters of this thesis along the three components of the project.

configurations, Chapters 1, 3 and 6 are concerned with the more general model of XML tree structures common to network management and SOA.

3. Type of problem: Chapters 2 and 3 develop on the satisfiability of the logic and its applications, while the other chapters concentrate on the model checking problem.

The text of each chapter is taken directly from conference and journal papers published during the three years of this research project. The paper from Chapter 3 was submitted in January 2008 and is awaiting acceptance at the time of this writing. The results of Chapter 6 were published in two distinct papers in 2007; the text presented here is the augmented journal version that was submitted in February 2008. At the beginning of each chapter, the exact reference of each original paper, followed by a short presentation in French, is given before the original English text begins.

Finally, a short English summary of the conclusion can be found at the end of the thesis.

REVUE DE LA LITTÉRATURE

Les prochains chapitres possédant déjà un revue de littérature détaillée spécifique à chaque question traitée, nous nous contenterons de mentionner ici les références les plus pertinentes à la compréhension générale de cette thèse, regroupées en grandes catégories. Certaines d'entre elles, qui présentent des idées fondamentales, sont décrites en détail ; d'autres ne sont que mentionnées à des fins de référence pour le lecteur intéressé.

Données semi-structurées

La vocable *données semi-structurées* regroupe tous les langages formels permettant de décrire et d'extraire des données d'une structure arborescente.

Ces structures apparaissent elles-mêmes dans une grande variété de situations. Un premier usage remonte aux années 70 par des chercheurs issus du domaine de la linguistique, lesquels introduisirent le concept des *attribute value structures*, représentées par des *attribute value matrices*. Ces structures sont utilisées dans l'organisation et la représentation de concepts linguistiques tels les noms, les nombres, les genres et les fonctions des mots ou des groupes de mots. L'imbrication successive de ces matrices en tant que cellules de matrices d'ordre supérieur donne naturellement lieu à une hiérarchisation de l'information qui peut alternativement se représenter sous forme d'un arbre dont chaque nœud est une étiquette. Ce dernier constat fut établi par Blackburn (1993).

Plus récemment, l'étude des données semi-structurées prit un nouvel essor en raison de l'avènement du langage XML. Encore une fois, le concept-clé est celui de l'imbrication : dans ce cas-ci, il s'agit d'une imbrication successive de structures composées de balises (*tags*) à l'intérieur d'une nouvelle paire de balises ouvrante et fermante qui crée l'arborescence. Les nœuds de l'arbre sont les étiquettes formées des noms de chacune des

balises ; les enfants de chaque nœud sont les balises immédiatement incluses à l'intérieur du parent.

Étant donné la grande polyvalence de la notation XML pour représenter des données en tous genres, un grand nombre de travaux se sont intéressés à la manière « d'interroger » un document XML pour en extraire des portions répondant à certains critères. À cet égard, plusieurs langages d'extraction ou de requête ont été développés et étudiés autant selon un angle pratique et applicatif que selon un angle théorique et mathématique.

Le principal langage à cet effet est sans conteste XPath, normalisé par l'organisme W3C (Clark et DeRose, 1999). Ainsi que son nom l'indique, XPath est un langage permettant d'exprimer des chemins dans une arborescence ; il s'agit d'un langage de requête au moyen duquel il est possible de spécifier une succession de balises, chacune étant imbriquée dans la précédente. La sous-structure (c.-à-d. le sous-arbre) se trouvant au bout du chemin devient la portion de document XML qui sera extraite par la requête lorsque exécutée. La norme W3C prévoit un certain nombre de raffinements à ce principe de base : il est ainsi possible de faire appel à des caractères génériques ne donnant aucune contrainte sur le nom d'une balise, d'exécuter une requête « déracinée » qui sera évaluée en chaque nœud de la structure, de faire appel à la fermeture transitive de la relation hiérarchique ou encore de composer de manière booléenne des conditions sur plusieurs balises. XPath n'est pas un langage déclaratif ; c'est un langage de requête qui permet d'extraire des informations. On verra plus loin comment plusieurs travaux transforment XPath pour le rendre approprié dans des situations déclaratives.

XQuery (Boag *et al.*, 2005) est une extension de XPath qui enrichit l'expressivité du langage original. Dans les expressions XQuery, il est possible d'utiliser des variables, de quantifier ces variables universellement ou existentiellement, de même que de faire appel à un certain nombre de fonctions (addition, transformation de chaînes de caractères).

De par son extrême richesse (XQuery est un langage Turing-complet), il est possible d'utiliser XQuery pour simuler l'exécution de n'importe quel programme, et par le fait même exprimer virtuellement n'importe quelle requête raisonnablement concevable.

Déterminer de manière formelle et rigoureuse la classe de complexité de l'évaluation d'une requête dans ces différents langages sur une structure XML donnée est un problème mathématique à part entière qui a fait l'objet de nombreuses études sur une foule de variantes du problème original. En effet, selon que l'on retire un ou plusieurs éléments du langage complet, on obtient des « fragments » de XPath ou de XQuery pour lesquels la complexité varie. C'est ainsi que XQuery est rarement utilisé tel quel et qu'à l'instar de XPath, la plupart des applications se restreignent à certains de ses fragments.

Langage complémentaire à XPath et à XQuery, TQL (Cardelli et Ghelli, 2001; Cardelli et Ghelli, 2004). a été développé par Cardelli (2001) pour décrire des structures arborescentes; il a ensuite été adapté (Cardelli *et al.*, 2002) pour devenir un langage de requête permettant l'extraction de données dans ces structures, à la manière de XPath ou XQuery. Basé sur la logique des ambients (Cardelli et Gordon, 1998), TQL est un formalisme d'une expressivité équivalente à celle de XQuery. Pour cette raison, dans la suite de cette section, nous considérerons les différents fragments de XPath, XQuery ou TQL étudiés dans les travaux cités comme différents fragments et différentes syntaxes d'une même « logique sur les arbres ». Enfin, par souci de complétude, mentionnons Schematron (Coen *et al.*, 2004) qui se veut une autre alternative aux langages XPath et XQuery (Boag *et al.*, 2005). Bien qu'il s'agisse d'une norme ISO, Schematron a été peu étudié et n'est pratiquement pas mentionné dans la littérature.

L'évaluation d'expressions XPath (en quelque sorte le « *model checking* », bien qu'il ne s'agisse pas d'un langage déclaratif) s'avère relativement simple et peut être considérée comme un problème résolu. Des algorithmes efficaces de traitement des requêtes XPath ont été donnés par Gottlob *et al.* (2002). Les auteurs y déplorent entre autres

que les implémentations classiques des moteurs de requête XPath, tels XALAN, XT et l'engin inclus dans Internet Explorer, utilisent des algorithmes de traitement des requêtes XPath hautement inefficaces et exponentielles en pire cas ; ces affirmations sont motivées par une évaluation empirique de ces outils sur des contraintes synthétiques de taille paramétrable. Ce principe d'évaluation expérimentale d'un algorithme sera repris dans la présente thèse au terme de pratiquement tous les chapitres. L'article poursuit en fournissant de nouveaux algorithmes d'évaluation des requêtes XPath dont la complexité est polynomiale en termes de la taille du document XML et de la longueur de la requête à traiter. La structure de cet article ressemble en particulier au chapitre 6 de cette thèse, où deux méthodes de *model checking* d'expressions de la logique CTL-FO$^+$ seront départagées par leur temps d'exécution réel pour différents exemples de contraintes.

Par contre, le même problème posé pour un fragment de logique sur les arbres plus riche que XPath donne des résultats très différents. Charatonik et Talbot (2001) étudient la question du *model checking* d'expressions en logique des ambients (un sur-ensemble de TQL) et démontre que le problème général est indécidable. L'article cite un résultat dû à Trakhtenbrot qui sera également utilisé au chapitre 3.

L'indécidabilité des plus riches fragments de la logique sur les arbres n'est en soi pas très surprenante. Les plus récents efforts ont plutôt consisté à circonscrire l'indécidabilité en retrouvant les fragments maximaux qui préservent la décidabilité du *model checking* ou de la satisfiabilité. C'est en particulier l'objectif du chapitre 3 (dans le cas de la satisfiabilité).

Par exemple, Deutsch et Tannen (2001) démontrent que la fermeture transitive de XPath ne peut être exprimée au moyen d'un langage du premier ordre, ce qui complique d'autant l'évaluation de telles requêtes. On s'intéresse surtout dans cet article à la complexité de certaines contraintes d'intégrité : il s'agit de formules de logique du premier ordre dont les atomes sont des expressions XPath. On ne cherche donc plus à extraire des

informations comme avec XPath, mais bien d'exprimer des relations sur des éléments de document extraits au moyen de XPath. Ce faisant, les contraintes d'intégrité deviennent des énoncés déclaratifs, qui peuvent être soit vrais, soit faux pour un document donné.

L'article de Deutsch et Tannen s'intéresse particulièrement à la question de l'inclusion (*containment*) : étant donné deux contraintes d'intégrité XPath φ et ψ, est-il vrai que pour toute structure T qui vérifie φ, T vérifie également ψ ? Cette question revêt une importance particulière dans le domaine des bases de données, dont le problème de la satisfiabilité, exposé dans l'introduction de cette thèse, peut être vu comme un cas particulier. Bien que la satisfiabilité de la logique du premier ordre en général soit indécidable, l'article montre que la question de l'inclusion de contraintes d'intégrité de Simple XPath est décidable et se situe dans la classe EXPTIME. La question de la décidabilité pour une logique arborescente similaire en plusieurs aspects aux contraintes d'intégrité présentées dans cet article sera étudiée en détail au chapitre 3 et aboutira à une classe de complexité proche de EXPTIME.

La notion d'*intégrité* provient du fait qu'il est possible d'exprimer des relations qui imposent une structure générale à un document, telles que « chaque balise `person` possède au plus un seul enfant de nom `address` ». En cela, l'article amène une idée importante qui sera abordée aux chapitres 1 et 2 : une contrainte d'intégrité est une restriction sur la forme et le contenu d'un document qui prend ses origines dans la sémantique des informations qu'il contient. Cette sémantique peut être, par exemple, tirée des conditions de configuration pour qu'un service réseau soit fonctionnel, une idée que nous avons déjà exploitée (Hallé *et al.*, 2004). À cet égard, les contraintes d'intégrité étudiées par Deutsch et Tannen s'approchent davantage de formules expressibles dans la logique XQuery.

Le problème de la satisfiabilité pour une logique sans variables a d'abord été soulevé par Blackburn *et al.* (2003) et par Calcagno *et al.* (2003), où le fragment sans quantificateurs de la logique des ambients est ainsi démontré décidable. Des fragments de XPath

ont été analysés de la même manière (Hidders, 2003; Marx, 2004; Marx, 2003; Lakshmanan *et al.*, 2004). En introduisant des variables, le langage devient XQuery, dont un fragment a été analysé par Koch (2006). Plus récemment, Filiot *et al.* (2007) considèrent un fragment *gardé* de TQL avec quantification sur les sous-arbres (cf. section 3.2.3 pour une description de la logique gardée). Enfin, le problème général de la satisfiabilité de TQL a été étudié par Calcagno *et al.* (2003) et Conforti et Ghelli (2004). Un grand nombre de résultats de décidabilité pour divers fragments est présenté à la section 3.2.3.

Le lien entre les structures en arbres et la logique modale décrit aux chapitres 1 et 3 remonte à Blackburn (1993) ; d'autres liens entre XPath et les logiques modales ont été suggérés (Alechina *et al.*, 2003) et comparés expérimentalement (Franceschet et Zimuel, 2005). La réduction du *model checking* de formules logiques arborescentes au model checking de logiques temporelles a été d'abord étudié en utilisant la logique CTL (Miklau et Suciu, 2002; Afanasiev, 2004; Gottlob *et al.*, 2002; Afanasiev *et al.*, 2004) avec l'outil de *model checking* NuSMV (Cimatti *et al.*, 2002), ainsi qu'en utilisant la logique PDL (Afanasiev *et al.*, 2005).

Finalement, la logique CL présentée au chapitre 1 entretient des liens étroits avec la logique hybride (Areces *et al.*, 1999). Ce lien avec la logique hybride a d'ailleurs été reconnu et énoncé par d'autres auteurs (Bidoit *et al.*, 2004; Franceschet et de Rijke, 2006).

Des résultats de décidabilité ont été démontrés par ten Cate et Franceschet (2005). Leur article traite de la logique hybride $HL(@, \downarrow)$. D'emblée, le résumé annonce que cette logique est indécidable et que son model checking est PSPACE-complet. On cherche ensuite des restrictions syntaxiques qui permettent de retrouver la décidabilité. Le principal résultat de cette recherche est que la satisfiabilité de la logique hybride *sans* le patron d'opérateurs $\Box \downarrow \Box$ est décidable et dans 2EXPTIME ; il en va de même pour la validité des formules sans le patron $\Diamond \downarrow \Diamond$. En conséquence, une formule de logique hybride qui

ne possède ni l'un ni l'autre est automatiquement décidable à la fois pour les problèmes de satisfiabilité et de validité. Ce résultat particulier s'avère intéressant puisque, comme on le verra au chapitre 3, la logique CL peut être plongée dans un fragment de $HL(@, \downarrow)$ qui contient le patron $\Box \downarrow \Box$.

La quantification de type *freeze*, utilisée avec quelques variantes dans les chapitres 1, 3 et 6, a d'abord été suggérée par Alur et Henzinger (1994) et reprise par Demri *et al.* (2005). Une idée similaire a également été utilisée par Rensink (2006) pour sa logique QCTL ; le texte explique que le système construit possède deux types de transitions : des *regular* et des *assignment next-steps*, et que ces dernières sont des transitions où le système essaie de deviner la valeur d'une variable quantifiée de la formule.

Une autre approche a été abordée par Kupferman (1995) : son article étend les logiques temporelles CTL et CTL* avec une quantification existentielle sur les propositions atomiques, ce qui donne les langages EQCTL et EQCTL* : si φ est une formule CTL(*) et p, q, r sont des propositions atomiques, alors $\langle p, q, r \rangle \varphi$ est une formule EQCTL(*). Une structure de Kripke K satisfait cette formule si et seulement si il existe une structure K' telle que K' satisfait φ et que K et K' sont les mêmes à un changement de nom près de p, q et r. Seule la quantification existentielle est considérée ; l'ensemble des formules EQCTL(*) n'est pas fermé par négation et il n'y a donc pas de quantification universelle. Tel qu'attendu, on démontre que EQCTL(*) est strictement plus expressive que CTL(*), et on établit la classe de complexité du problème du *model checking* pour EQCTL, lesquels sont, aux dires de l'auteure, peu encourageants (« *cheerless* »). Le chapitre 6 mentionne les différences entre EQCTL et la logique CTL-FO$^+$ développée dans la présente thèse.

Vérification de configurations

L'utilisation d'une logique a été proposée entre autres par Feamster et Balakrishnan (2003) pour vérifier qu'un protocole de routage satisfait certaines propriétés ; l'auteur suggère également l'usage de cette logique comme d'un langage de spécification de politiques de haut niveau. Ponder (Damianou *et al.*, 2001) et OCL (Warmer et Kleppe, 2003) sont d'autres tels langages pouvant exprimer des politiques. Parmi les autres langages de haut niveau, notons les approches basées sur les ontologies (de Vergara *et al.*, 2002; Noy *et al.*, 2001; Crubézy, 2002).

Le concept de *dépendances* dans un système informatique, central à la présente thèse, a été exploré de manière formelle par Sun et Couch (2006). Plus spécifiquement, Benedikt *et al.* (2002) introduisent la notion de contraintes d'intégrité dans les structures XML et les étendent à la génération automatique de code de vérification (gardes) au moment de l'exécution (Benedikt et Bruns, 2004).

La vérification de problèmes de configurations dans les routeurs a été soulevée par Le *et al.* (2006) qui utilisent une approche de *data mining* pour résoudre le problème.

Une approche intéressante à la configuration d'équipement réseaux se trouve dans l'article de Cacciagrano *et al.* (2006). Cet article s'oriente en fait vers les architectures orientées services, mais les principes qui y sont décrits peuvent s'appliquer directement au domaine des réseaux. En cela, il établit un excellent pont entre ces deux domaines, comme le soutiennent d'ailleurs les travaux de cette thèse.

L'article avance que la norme WSDL (Christensen *et al.*, 2001) est trop limitée pour exprimer certaines contraintes sur la structure des messages. En particulier, les valeurs contenues dans le message sont importantes : la valeur d'un paramètre dans un message peut dépendre de la valeur d'un autre paramètre dans le même message. Cette constatation n'est autre que la définition des contraintes dites statiques exposées dans

l'introduction de cet ouvrage. Par ailleurs, les auteurs soutiennent que les ontologies et les langages comme RuleML sont trop lourds pour exprimer ce genre de contraintes. OCL serait intéressant, mais n'est pas basé sur XML et n'est donc pas approprié non plus. Parmi les langages intéressants, il y reste donc Schematron (qu'on a mentionné plus tôt) et CLiX (*Constraint Language in XML*). L'objectif de l'article consiste à générer automatiquement des morceaux de code Java appelés *stubs* qui, en plus, vérifient des contraintes CLiX. Les contraintes CLiX sont rendues disponibles par le service web au même titre que son WSDL et peuvent donc être connues à l'avance.

La vérification est effectuée au moment de l'exécution. Le stub Java transforme en structure XML les données qu'il veut envoyer au service web, et avant d'être envoyé, le message ainsi formé est passé dans un validateur CLiX qui s'assure que les contraintes sur le message sont respectées. Si c'est le cas, le message est envoyé. Les auteurs décrivent ensuite comment ce principe est réalisé dans la pratique en utilisant un traducteur en stub (WSDL2Java) et un validateur CLiX open source (OpenCLiXML). Les auteurs affirment que la charge supplémentaire requise par la validation en temps réel de chaque message par le validateur CLiX est compensée par l'économie de messages d'erreur et de compensation qui seraient autrement échangés advenant une invocation incorrecte du service. Cet argument est fondamental. Il s'applique également au travail de la présente thèse et en constitue une justification implicite : l'effort d'une validation formelle *a priori* est compensé par une gestion d'erreurs réduite au moment de l'exécution.

L'article de Cacciagrano *et al.* (2006) est, à notre connaissance, l'un des rares articles soulevant le fait que les valeurs à l'intérieur d'un même message XML peuvent être interdépendantes. Cependant, les contraintes inter-messages (c.-à-d. hybrides) ne sont pas considérées : toutes les contraintes se réfèrent au contenu d'un seul message. L'architecture proposée l'interdit, puisque chaque message est évalué indépendamment de tout ce qui a été envoyé avant ou sera envoyé après.

Génération automatique de configurations

La génération automatique d'une structure satisfaisant un ensemble de propriétés participe d'un paradigme plus général appelé l'informatique autonome (*autonomic computing*) (Hinnelund, 2004; Parashar et Hariri, 2004) ; appliquée particulièrement aux réseaux informatiques, cette notion devint celle de réseaux autonomes (ou *autonomiques*) (Konstantinou, 2003; Gaïti *et al.*, 2005). Narain *et al.* (2003) proposent le concept de grammaire de services (*service grammar*) pour définir les propriétés désirables du système.

Dans la même lignée, Keller (2004) propose des services dits auto-configurables ; le terme même de *self-configuration* est mentionné par Melcher et Mitchell (2004) et rejoint le concept de réseaux programmables mis de l'avant par Cisco dès 2002 (Cisco Systems, 2002). Mentionnons également le projet SELFCON (Boutaba *et al.*, 2001), l'environnement AUTONOMIA (Hariri *et al.*, 2003), les *smart routers* (Pujolle et Gaïti, 2004; Gaïti *et al.*, 2005) et le *General Switch Management Protocol* (GSMP) (Cha *et al.*, 2004) qui tend également vers l'auto-configuration.

La plupart des systèmes concrets de gestion des configurations, tels Cfengine (Couch et Gilfix, 1999) et Bcfg2 (Desai *et al.*, 2006) utilisent une approche impérative (décrivant en détail les actions à effectuer) plutôt que déclarative (décrivant les buts à atteindre sans préciser de plan) et sont seulement partiellement appropriés à ce travail. Nous avons déjà mentionné le protocole Netconf (Enns, 2005), à partir duquel une suite de gestion de réseaux, appelée ENSUITE, fut développée (Cridlig *et al.*, 2005).

Une dernière approche a été proposée par Narain (2005) qui utilise un engin de satisfiabilité de formules booléennes (SAT) pour produire une configuration d'un ensemble de routeurs satisfaisant un certain nombre de contraintes réseau exprimées dans le langage d'Alloy (Jackson, 2000). Cette approche est la plus proche de la satisfiabilité telle

qu'entendue dans ce travail, par exemple aux chapitres 2 et 5.

L'utilisation du *model checking* pour la spécification de séquences d'opérations est évoquée par Giunchiglia et Traverso (1999). Le titre de leur article (*Planning as Model Checking*) annonce dès le départ le principe suivi par les auteurs : la recherche d'une solution à un problème de planification peut se résumer à la recherche d'une trace satisfaisant un ensemble de formules temporelles ; ces formules sont une représentation mathématique des contraintes d'ordonnancement qui doivent être suivies. L'idée de base est la suivante : si K est une structure de Kripke et φ une formule qui est vraie seulement pour les états-buts, alors le problème de trouver un plan est réduit (grossièrement) au *model checking* de la formule CTL $\mathbf{EF}\varphi$. Pistore *et al.* (2004a) définissent EaGLe, un langage d'expression des buts qui étend CTL. Ce principe est similaire à celui utilisé au chapitre 5 pour produire une séquence d'opérations de configuration.

Un problème semblable est abordé, dans le domaine des architectures orientées services, par Duan *et al.* (2004) qui proposent une sémantique restreinte du langage de composition BPEL pour laquelle il est possible de générer automatiquement la composition (c.-à-d. l'interaction) de deux services, ainsi qu'une logique (logique du premier ordre additionnée d'opérateurs XPath) pour exprimer des pré/postconditions sur les tâches. Ils suggèrent un algorithme qui calcule automatiquement cette composition à partir de règles d'inférence données. On ne tente pas ici de résoudre de façon générale le problème de la composition, mais on se limite à un cadre supposé réaliste avec des hypothèses plus fortes ; sous ces hypothèses, le calcul des invariants devient automatisable.

Dépendances temporelles et model checking

Dans le domaine des services réseaux, la notion d'ordonnancement des opérations de configuration a été relativement peu étudiée. On peut cependant mentionner le concept

de *convergence* introduit dans Couch et Gilfix (1999) et raffiné dans Couch et Sun (2003) : on considère un ensemble d'actions primitives P dont l'exécution modifie l'état d'un système. Ces actions prises une à une sont dites convergentes si elles sont idempotentes, c.-à-d. l'application répétée de la même action $p \in P$ à une configuration X n'a aucun effet ($p(X) = p(p(X))$). Une certaine forme de séquentialité émerge de cette étude algébrique de la configuration des systèmes informatiques en raison de l'observation suivante : la composition de deux actions convergentes p et q peut ne pas être commutative (c.-à-d. $p(q(X)) \neq q(p(X))$). L'ordre dans lequel p et q sont appliquées devient important et ouvre ainsi la porte à la notion de dépendances séquentielles dans les opérations de configuration.

La notion de commutativité revêt une grande signification dans l'article précédent. Pour P un ensemble d'actions convergentes et commutatives deux à deux, l'exécution successive de toutes les actions $p \in P$ aboutira toujours à la même configuration finale, quel que soit l'ordre ou le nombre d'occurrences (supérieur à 0) de chaque action dans la séquence. Les auteurs avancent donc l'idée que P représente une « intention » visant un « but » particulier (l'unique configuration finale), et que chaque action collabore à l'atteinte de ce but en n'interférant pas avec les autres (en raison de sa commutativité). Cette notion se rapproche du concept de composante et de *milestone* tel que présenté aux chapitres 4 et 5. Il est intéressant de souligner que dans ces deux chapitres, ces concepts émergent aussi naturellement des dépendances séquentielles (c.-à-d. de la non-commutativité) entre les opérations de configuration.

C'est cependant dans le domaine des architectures orientées services que l'on retrouve la littérature la plus foisonnante sur la vérification de systèmes à base de messages. En effet, comme on l'a vu plus tôt, un grand nombre d'applications peuvent être exposées à un utilisateur au moyen d'une interface d'échange de messages, ce qui explique le déplacement des intérêts de la communauté de la recherche vers ce domaine particulier. Mentionnons d'abord que la norme WSDL (Christensen *et al.*, 2001) et le langage d'or-

chestration BPEL (Andrews *et al.*, 2003) sont les deux pierres angulaires du domaine : le premier permet de décrire la syntaxe des messages utilisés pour converser avec un service, tandis que le second agit comme un langage de scriptage permettant l'interaction entre plusieurs services.

La notion de séquence de messages dans un service web est explicitement mentionnée dans le livre de Alonso *et al.* (2004), qui se veut une introduction générale aux technologies relatives aux services web. En particulier, on y présente la notion de spécification de transactions entre services au moyen de la norme WS-Transaction définie par le W3C. Cependant, cette spécification est relativement éloignée d'une approche basée sur la logique temporelle, et les questions de validation automatique de ces transactions au moyen de méthodes formelles ne sont pas abordées. Par contre, on y avance l'idée d'un *transaction monitor*, entité se plaçant entre deux services web qui communiquent, et dont la tâche consiste à intercepter les messages envoyés et reçus aux deux extrémités. Par un mécanisme qui n'est pas spécifié avec précision, le moniteur de transaction suit le déroulement de l'interaction entre les deux services et détecte en temps réel la violation d'une contrainte spécifiée au moyen d'une expression de la norme WS Transaction. Cette idée, transposée aux formules CTL-FO$^+$, fait partie des extensions futures de cette thèse telles que mentionnées à la fin du chapitre 6.

La notion de dépendances séquentielles est également mentionnée par Greenfield *et al.* (2005), où ces dépendances s'appellent des contraintes de cohérence (*consistency*). Cependant, la séquentialité y est étudiée avec une préoccupation d'intégrité des données proche de problématiques de bases de données (propriétés ACID, transactions atomiques).

Plus près de la méthodologie de cette thèse, de nombreux travaux sur la validation de séquences de messages dits « propositionnels » (avec abstraction de leur contenu) ont été réalisés dans les dernières années.

L'article de Bultan *et al.* (2003) fait date dans le domaine de l'étude formelle des propriétés déclaratives en introduisant un certain nombre de notions et de problèmes fondamentaux dont la plupart sont encore ouverts à ce jour. L'idée principale repose sur la notion de *conversation specification*. Une conversation est un échange de messages bilatéraux entre plusieurs services. À l'instar du moniteur de transactions mentionné précédemment, un observateur externe (*watcher*) capture les messages selon leur ordre d'envoi. Chaque message est atomique (sans contenu) et peut être représenté par un symbole ; une spécification de conversation devient une ensemble de mots (au sens combinatoire du terme) correspondant aux interactions permises entre les pairs. Sur un plan plus technologique, le *Web Service Choreography Description Language* (WS-CDL) normalisé par le W3C vise un objectif équivalent.

Dès lors, il est possible de spécifier une conversation de deux façons : la manière descendante (*top down*), où on donne le patron général d'échange qu'un observateur externe pourrait capturer (à la manière de WS-CDL), et la manière ascendante (*bottom-up*) qui spécifie donne l'automate (ou plus précisément, la machine de Mealy) que chaque service individuel e. Lorsque l'observateur externe est abstrait par un automate, les spécifications de conversations deviennent des langages réguliers classiques qu'il devient possible de reconnaître. En cela, l'article présente une implémentation possible d'une version simplifiée de l'idée mentionnée par Alonso *et al.* (2004).

L'article soulève cependant deux questions fondamentales à la théorie de la validation de systèmes communicants. La première est la suivante : étant donné une spécification globale de conversation (un langage), existe-t-il des automates (ou plus précisément, des machines de Mealy) représentant chacun des « interlocuteurs » tels que leur composition produit ce langage ? La réponse n'est pas toujours positive. De la même manière que, pour les données semi-structurées, on s'est intéressé aux fragments des logiques sur les arbres qui préservaient la décidabilité des problèmes, ici on cherche des conditions sur le modèle des services qui garantissent la *réalisabilité* d'une conversation.

Pour ce faire, deux modèles de communication sont présentés. Dans le modèle synchrone, lorsqu'un service A envoie un message à un partenaire B, les deux doivent être disponibles en même temps. Les auteurs comparent cette situation à une conversation téléphonique lorsque les interlocuteurs n'ont pas de répondeur automatique : ils doivent être en même temps au bout du fil pour entrer en communication. Dans le modèle asynchrone, lorsqu'un service A envoie un message à B, ce message est placé dans une file de réception chez B, qui peut en prendre connaissance (c.-à-d. « consommer » le message) à n'importe quel moment ultérieur. Cette situation correspond à une conversation téléphonique avec un répondeur.

Bien évidemment, la réalisabilité d'une conversation se démontre de manière beaucoup moins évidente dans un modèle asynchrone et plusieurs propriétés (dont le *model checking* de propriétés LTL classiques) deviennent indécidables, même si les services sont en soi représentés par des machines à états finis. Il est également possible (et les auteurs le mentionnent) d'étendre le concept à des grammaires et d'en tirer les conséquences théoriques appropriées.

La seconde question fondamentale abordée par les auteurs est celle de la *synchronisabilité* : étant donné un modèle asynchrone représentant un ensemble de pairs, existe-t-il un modèle synchrone qui produise la même conversation (c.-à-d. le même langage) ? L'intérêt, si tel est le cas, consiste alors à vérifier des propriétés sur le modèle synchrone, plus facile à étudier et démontrablement équivalent à l'original du point de vue des séquences de messages échangés.

Ces deux questions ont fait l'objet d'une série d'articles des mêmes auteurs (Fu *et al.*, 2003; Fu *et al.*, 2004a; Fu *et al.*, 2004b), y compris une synthèse abrégée (Bultan *et al.*, 2006). Ces auteurs sont d'ailleurs les seuls à avoir étudié en détail l'impact de communications asynchrones sur la complexité théorique de ces questions en rapport aux services web. À titre d'exemple, les communications dans les travaux de la présente

thèse sont toujours supposées synchrones.

La question de la réalisabilité a fait l'objet de travaux indépendants relativement similaires aux précédents. Ainsi, Decker et Weske (2007) modélisent des patrons de processus d'affaires sous la forme de réseaux de Petri, et s'intéressent ensuite à la question de l'application locale (*local enforceability*) qui n'est qu'un terme alternatif pour désigner la réalisabilité telle que définie précédemment.

Pour ce faire, le langage *Let's Dance* est un langage de description des chorégraphies entre services web. Il permet de décrire les patrons d'échanges de messages entre différents services. Trois opérateurs sont permis :

1. L'envoi du message M_1 de A vers B doit précéder l'envoi du message M_2 de C vers D
2. L'envoi du message M_1 de A vers B empêche par la suite l'envoi du message M_2 de C vers D
3. L'envoi du message M_2 de C vers D est interdit jusqu'à ce que A ait envoyé un message M_1 vers B ou jusqu'à ce que cette possibilité soit inhibée

On peut ensuite composer ces constructions entre elles pour créer des patrons complexes d'envoi de messages. La dichotomie entre les approches ascendante et descendante à la spécification des conversations est reprise par Zaha *et al.* (2006) sous les termes de « vue locale » et « vue globale » ; mises à part ces différences terminologiques, les mêmes questions générales sont abordées.

Un certain nombre de travaux se sont intéressés à la correspondance entre des langages de spécification de conversations et d'interactions de services web de haut niveau et des formalismes mathématiques existants. C'est le cas du travail de Decker *et al.* (2006) qui fournit une correspondance entre le langage de chorégraphies *Let's Dance* dont on a parlé plus tôt (inspiré à certains égards de WS-CDL) et le π-calcul —une notation visuelle similaire à celle de *Let's Dance* a également été proposée par Brambilla *et al.* (2005)

pour la représentation de formules LTL.

Le π-calcul est également proposé par Woodman *et al.* (2004) pour modéliser la communication entre les services web. L'objectif consiste à donner des formules en π-calcul qui décrivent l'ordre dans lequel les opérations doivent être appelées pour utiliser un service web correctement. Les auteurs suggèrent que ces formules soient transformées en XML et ajoutées aux documents WSDL des services web. Cette idée a également été formulée, pour des contraintes statiques, par Cacciagrano *et al.* (2006).

La seule validation effectuée consiste à vérifier l'absence de *deadlocks* et de *livelocks* de la composition. Cette vérification est effectuée à la main ; les auteurs expliquent en fin de document qu'il n'existe pas d'outil pour valider un modèle en π-calcul et qu'ils n'ont pas essayé d'autres avenues. L'article mérite surtout une mention pour avoir suggéré de publiciser le « protocole » (c.-à-d. les contraintes) qu'on doit suivre pour dialoguer avec un service donné, idée centrale au paradigme déclaratif.

Schmidt et Stahl (2004) transforment un fragment du langage d'exécution de services web BPEL en structures de réseaux de Petri. Cette approche est favorite dans la communauté, puisqu'une pléthore de (subtiles) variantes de BPEL ont été interprétées en réseaux de Petri depuis ce temps (Hinz *et al.*, 2005; Lohmann *et al.*, 2006; Schlingloff *et al.*, 2005).

L'article de Schlingloff *et al.* mérite une mention spéciale. Un script BPEL y est modélisé en un *open workflow net*, une variante « ouverte » des réseaux de Petri où un réseau peut avoir des emplacements d'entrée (qui reçoivent un jeton de l'extérieur) et de sortie (qui « envoient » un jeton). On peut ensuite composer deux réseaux de Petri en joignant leurs emplacements d'entrée-sortie correspondants et en faire un seul grand réseau de Petri. Une stratégie pour un réseau de Petri N est un réseau S tel que la composition de N et S donne un réseau qui termine toujours dans une configuration "finale". Un réseau N est ensuite dit *contrôlable* s'il existe une stratégie pour N.

La question qui intéresse les auteurs est la suivante : étant donné un processus BPEL, celui-ci est-il contrôlable ? Le cas échéant, on cherche à obtenir :

1. Une stratégie pour le script, qu'ils appellent un graphe d'interaction (ce qui se résume, *grosso modo*, à un automate) ;
2. ou encore mieux, les directives d'opération (*operating guidelines*) du processus ; il s'agit du graphe dont toutes les stratégies possibles sont des sous-graphes (c'est le graphe qui modélise tous les contrôleurs possibles et donc toutes les interactions légales possibles).

L'article présente un outil appelé Fiona qui permet de calculer les directives d'opération d'un réseau de Petri donné. À cet égard, le travail s'avère un peu différent de l'objectif recherché par cette thèse : les auteurs cherchent à déterminer si un service est contrôlable, et non à publier (et valider) un ensemble de contraintes que d'éventuels utilisateurs doivent respecter. Jusqu'à quel point ces concepts n'ont pas déjà été bien étudiés en théorie du contrôle demeure également nébuleux.

L'idée des directives d'opération est également présentée dans l'article de Li et Jagadish (2003), où deux services web sont modélisés par un graphe d'états dont les arêtes étiquetées représentent des actions. L'article donne une méthode permettant de vérifier si un utilisateur potentiel possède un graphe d'états compatible avec celui du service web qu'il pense utiliser (répondant ainsi à la question « ce service me permet-il d'exécuter tous les scénarios que je prévois ? ») en fournissant la meilleure correspondance des états et actions de l'un vers les états et les actions de l'autre.

Prenant une distance d'avec les réseaux de Petri, Foster *et al.* (2003; 2006) préconisent plutôt une traduction d'interactions exprimées selon la notation UML, plus précisément en *Message Sequence Charts* (MSC), sous la forme de processus à états finis. L'idée en soi n'était pas nouvelle, la notation UML ayant déjà été proposée pour la modélisation de processus dans une thèse de doctorat (Eshuis, 2002). Le rapport technique de van Breugel

et Koshkina (2006) donne un excellent aperçu des différentes approches d'interprétation du langage BPEL en structures formelles.

L'objectif ultime de ces travaux consiste à utiliser ces formalismes pour vérifier des propriétés séquentielles (c.-à-d. de logique temporelle classique). Par exemple, Ferrara (2004) propose l'utilisation des algèbres de processus pour modéliser, puis vérifier des contraintes d'échanges de messages entre des services web. L'architecture de vérification proposée en début d'article, laquelle met en jeu la traduction d'un processus BPEL en modèle formel, puis la vérification de ce modèle au moyen d'outils de validation existants, est la même que celle utilisée dans cette thèse. Plutôt qu'une correspondance entre BPEL et une structure de Kripke soumise à l'outil NuSMV, l'article décrit cependant une correspondance entre BPEL et l'algèbre des processus LOTOS.

Une idée similaire est proposée par Koshkina et van Breugel (2003) qui introduisent une algèbre de processus appelée BPE-calcul pour décrire des propriétés simples de services web. Les processus BPEL sont modélisés dans ce langage, puis vérifiés au moyen du *Process Algebra Compiler* (PAC), qui n'est autre qu'une interface se superposant au classique *Concurrency Workbench* (CWB), qui sera reprise ultérieurement (Koshkina et van Breugel, 2004). La formalisation des architectures orientées services au moyen des algèbres de processus a également fait l'objet d'une thèse de doctorat (Guidi, 2007).

Fondamental pour l'approche déclarative, l'article de Pistore *et al.* (2004b) ramène à l'avant-plan la notion d'exigences (*requirements*). L'article propose un langage appelé Tropos agissant comme langage de spécification et comme représentation graphique des exigences imposées à un service web. Les processus BPEL peuvent ensuite être vérifiés au moyen du T-Tool, programme basé sur l'outil de *model checking* NuSMV. Les propriétés Tropos deviennent alors des formules LTL classiques.

La notion d'exigences est également à l'ordre du jour dans l'article de Governatori *et al.* (2006), qui s'intéresse à la conformité d'un processus d'affaires à un contrat. Le

« contrat » dont on parle ici est un vrai contrat légal, qui est ensuite formalisé en utilisant la logique déontique. Par exemple, on cherche à exprimer le fait que « lorsque le vendeur prend connaissance d'une panne, il doit la réparer dans un délai de 24 heures ou payer une prime au client ».

À cet égard, la logique déontique est une logique propositionnelle à laquelle on ajoute deux opérateurs déontiques : $O\varphi$ signifie qu'il est obligatoire que φ se produise, et $P\varphi$ signifie qu'il est permis que φ se produise. Un troisième opérateur, $\times$, indique la réparation d'une violation. Ainsi, $O\varphi \times O\psi$ indique que φ doit se produire, mais que si ce n'est pas le cas, alors ψ doit se produire. La formule ψ est vue comme la réparation de la violation de φ.

L'article indique comment transformer un ensemble de règles de logique déontique en forme normale en combinant des réparations compatibles et en éliminant des règles qui sont incluses dans d'autres. Un exemple d'un contrat réel est formalisé et analysé du début à la fin sur un processus d'affaires modélisé en BMNN (un langage de spécification de processus d'affaires). Cependant, l'analyse est effectuée manuellement. Il n'est fait mention d'aucun outil de modélisation ou de validation automatique. De plus, la notion de contrat est prise ici sous son angle légal, ce qui diffère des autres travaux sur les services web. Finalement, la logique déontique est propositionnelle ; il n'y a pas de quantificateurs possibles sur le « contenu », comme le demande l'expression de contraintes dynamiques présentée dans l'introduction de cette thèse.

Par contre, Pesic et van der Aalst (2006) s'intéressent aux processus d'affaires sous l'angle déclaratif qui nous intéresse, mais cherche plutôt à construire des processus dits *flexibles* qui peuvent s'adapter à des changements d'exigences sans devoir être « reprogrammés ». Pour ce faire, le langage *ConDec* est introduit ; ConDec est basé sur la logique temporelle plutôt que sur un langage de type impératif. Un autre langage déclaratif appelé *DecSerFlow* est présenté dans un autre article (van der Aalst et Pesic, 2006). Une

idée similaire portant le nom de processus d'affaires *agiles* a également été développée par IBM (Graml *et al.*, 2007).

Un raffinement de ces travaux consiste à modéliser le contenu des messages. L'outil VERBUS est un prototype expérimental permettant de traduire un processus BPEL en un modèle formel, cette fois-ci une structure de Kripke. Ce projet a fait l'objet d'une thèse de doctorat (Arias-Fisteus, 2005). Ces travaux présentent l'intérêt de reproduire de manière complète la sémantique du langage BPEL en structure de Kripke ; cette sémantique inclut même les mécanismes de gestion d'erreurs (*compensation handlers*) définis par BPEL. Un certain nombre d'adaptateurs permettent de convertir la structure en un fichier reconnu par l'outil de *model checking* SPIN (Arias-Fisteus *et al.*, 2004a). Le projet fut ensuite étendu pour fournir un adaptateur permettant de remplacer SPIN par NuSMV (Arias-Fisteus *et al.*, 2005; Arias-Fisteus *et al.*, 2004b). Ce travail revêt une importance particulière, VERBUS étant l'outil utilisé au chapitre 6 pour traduire un processus BPEL en une structure de Kripke au format de l'outil NuSMV.

Turner propose un autre outil de vérification (Turner, 2005). Le langage MUSTARD permet de spécifier les traces dans une syntaxe proche de celle de BPEL (incluant des primitives telles *interleave* et *send*). Fait intéressant, dans les travaux qui ont mené à la publication de l'article constituant le chapitre 6 de cette thèse, MUSTARD a d'abord été considéré comme outil pour traduire un processus BPEL en une structure de Kripke avant que le choix ne s'arrête sur VERBUS.

Il existe un grand nombre d'autres travaux plus ou moins similaires, utilisant diverses techniques de *model checking*. Aucune d'entre elles ne tient compte du contenu des messages dans l'expression des propriétés : elles ne permettent donc pas la validation de contraintes hybrides. Nous les citons en rafale par souci de complétude (Kazhamiakin *et al.*, 2006; Berardi *et al.*, 2005; Duan *et al.*, 2004; Lohmann *et al.*, 2006; Nakajima, 2004). Des outils de validation expérimentaux du même ordre ont également été propo-

sés (Walton, 2004; Brogi *et al.*, 2004; Pistore *et al.*, 2004b; Johnson *et al.*, 2004; Decker *et al.*, 2006; van der Aalst et Pesic, 2006).

Dans l'article de Deutsch *et al.* (2004), on vérifie des propriétés temporelles sur des services web en faisant référence au contenu dans les « messages ». L'exemple employé est celui d'un site web d'achat d'ordinateurs qui renvoie des pages web à l'utilisateur. Le modèle de service web est un n-uplet décrivant les inputs d'une page et les transitions possibles étant donné ces inputs au moyen de prédicats ; cette approche s'avère plus près des bases de données que des autres travaux sur la validation des services web. L'article poursuit en présentant un grand nombre de résultats de décidabilité pour des propriétés utilisant LTL, CTL, avec ou sans restrictions sur les entrées des pages. Il convient de remarquer que les résultats de complexité se situent tous dans des classes relativement élevées (PSPACE, EXPTIME, 2EXPTIME).

Un outil de validation appelé WAVE a été développé à partir de ces théories (Deutsch *et al.*, 2005). WAVE permet de vérifier des traces d'exécution de sites web transactionnels en tenant compte des données saisies par l'utilisateur dans le cours de l'interaction. Une idée similaire est reprise par le langage WebML [3], qui permet de générer automatiquement le code d'un portail web en ligne satisfaisant un ensemble de propriétés exprimées en WebML. L'outil peut, paraît-il, produire 100% du code nécessaire sans intervention humaine. L'approche employée, cependant, possède peu de fondements mathématiques.

Par contre, l'article de Constant *et al.* (2007) fournit une approche basée sur les automates et souligne qu'il existe deux points de vue d'un système :

1. La vérification formelle : elle vérifie que la spécification formelle d'un système e un certain nombre de propriétés. C'est l'idée généralement acceptée du *model checking* : vérifier des formules sur une structure de Kripke représentant un système.

3. `www.webml.org`

2. La vérification de conformité (*conformance testing*) : elle compare le comportement observable d'un système avec un comportement observable décrit par une spécification formelle. Pour reprendre l'exemple précédent, ceci correspond à vérifier qu'une implémentation concrète d'un système fait la même chose que la structure de Kripke qui est supposée le modéliser. Pour ce faire, on doit produire des tests (c.-à-d. des séquences d'opérations) et suivre les mêmes actions en parallèle sur l'implémentation concrète et sur la spécification. À tout moment, la spécification et le système réel doivent « dire la même chose ». Le problème consiste à trouver de bonnes séquences d'opérations qui pourraient exhiber une différence entre les deux.

Une spécification peut donc respecter des propriétés (démontrées par *model checking*), mais si le système concret n'est pas conforme à la spécification, les propriétés peuvent quand même être violées dans la réalité. À cet effet, l'article propose une approche uniforme qui permet de produire un automate dans lequel on peut à la fois identifier les traces permettant de vérifier formellement qu'une spécification e une propriété, ainsi qu'identifier les traces permettant de tester qu'une implémentation e une spécification. Le modèle en question s'appelle un *input-output state transition system* (IOSTS), extension d'un automate classique auquel on ajoute des variables internes et des paramètres d'entrée-sortie.

Fait intéressant, pour décrire les propriétés que le système doit respecter, on utilise le concept d'observateur. Il s'agit d'un IOSTS avec un état cul-de-sac particulier étiqueté *Violate* (observateur négatif) ou *Satisfy* (observateur positif). Intuitivement, les observateurs négatifs servent à représenter des situations qu'on ne veut pas voir se produire et les positifs, des situations qu'on veut qui soient possibles. Les observateurs représentent une alternative aux logiques temporelles pour exprimer des propriétés.

À l'extérieur du domaine des architectures orientées services, mais pertinents à la

problématique étudiée dans cette thèse sont les travaux menés par une équipe de Microsoft Research. Le projet SLAM (Ball *et al.*, 2004) et deux autres projets appelés Vault et ESP sont des outils pour vérifier de manière rigoureuse qu'un programme (écrit dans le langage C) obéit à des « *interface usage rules* ». Moyennant quelques traductions de terminologie, ce problème est similaire à la validation de contraintes statiques, dynamiques, voire hybrides.

Dans le rapport technique de Ball et Rajamani (2000), on explique le principe de base : un programme est écrit en C et contient des variables (entières et booléennes) qui sont manipulées dans des structures de contrôle classiques (*if*, boucles, etc.). Disséminés à travers le code, des blocs conditionnels vérifient des assertions (ou plutôt des contre-assertions). Si la condition du bloc est vraie, la ligne suivante est une assertion de la constante *faux*, ce qui signifie qu'une situation indésirable vient de se produire. Soit alors k une de ces lignes où se trouve une telle contre-assertion. La ligne k est-elle accessible par une exécution du système ? Si la réponse est oui, c'est qu'il existe un moyen pour le programme de violer la contre-assertion de la ligne $k-1$. Il s'agit d'une forme assez simple de *model checking*. Cependant, calculer si un programme vérifie une telle contrainte est indécidable en général, et trop compliqué en particulier : il y a trop d'états possibles. Les auteurs s'intéressent alors aux programmes ne contenant que des variables booléennes, qui ne servent qu'à représenter le résultat de tests booléens apparaissant quelque part dans le programme et qui influencent son exécution, plutôt que de conserver les « vraies » valeurs des variables originales du script. Ce principe n'est rien d'autre que le concept d'abstraction des prédicats bien connu en *model checking* (Clarke *et al.*, 1994).

La question qu'on se pose est toujours la même : la ligne k est-elle accessible ? Il n'y a pas de logique temporelle associée à l'outil créé par les auteurs : on lui donne seulement le programme et le numéro de la ligne. Le seul travail effectué par l'outil consiste à vérifier l'accessibilité d'une ligne d'un certain programme, et à retourner la trace le cas échéant. À cet égard, même si certaines idées de ce projet peuvent être récupérées pour

la validation des contraintes hybrides, on doit rappeler que ces dernières sont *a priori* des formules quelconques de logique temporelle.

CHAPITRE I

CTL MODEL CHECKING FOR LABELLED TREE QUERIES

Ce chapitre est tiré de l'article suivant :

> Hallé, S., Villemaire, R., Cherkaoui, O. (2006). CTL Model Checking for Labelled Tree Queries. In *Proceedings of the 13th International Symposium on Temporal Representation and Reasoning (TIME 2006)*, IEEE Computer Society Press, 27-35.

Ce chapitre pose la première pierre du travail. Il décrit comment la validation de propriétés exprimées dans la logique sur les arbres CL (Villemaire *et al.*, 2005a) peut être traduite en un problème de *model checking* de logique temporelle classique. On verra dans les prochains chapitres en quoi l'utilisation du *model checking* est nécessaire pour vérifier des propriétés de nature dynamique dans des protocoles d'échanges de messages.

La construction présentée dans ce chapitre sera reprise et étendue au chapitre 3 pour réduire de manière similaire la question de la satisfiabilité de formules CL à un problème de satisfiabilité de logique temporelle classique.

L'approche de quantification « freeze » développée ici est une idée maîtresse du travail ; elle sera reprise de manière plus élaborée au chapitre 6 lors de la présentation de la logique temporelle CTL-FO^{+}. De plus, la section 1.3.3 donne une première critique de

l'approche « freeze » par rapport à une quantification explicite de la formule originale ; l'analyse détaillée de la question fera l'objet d'expériences plus poussées au chapitre 6.

Abstract

Querying and efficiently validating properties on labelled tree structures has become an important part of research in numerous domains. In this paper, we show how a fragment of XPath called Configuration Logic (CL) can be embedded into Computation Tree Logic. This framework embeds into CTL a larger subset of XPath than previous work and in particular allows universally and existentially quantified variables in formulæ. Finally, we show how the variable binding mechanism of CL can be seen as a branching-time equivalent of the "freeze" quantifier.

1.1 Introduction and Related Work

The representation of data in the form of tree-like structures is a natural choice for numerous applications. Arborescent, or so-called *semi-structured* data has been used in areas as diverse as mobile ambients (Cardelli et Gordon, 1998), database systems (Abiteboul, 1997) and network equipment configuration (Hallé *et al.*, 2004). The widespread use of the Extensible Markup Language (XML) (Bray *et al.*, 2004) has further confirmed the importance of labelled trees as a mean of representing information, forming among other things the basis for web services standards.

Complementary to the representation of data into structures is the need to query those structures, either to validate constraints on them, to express patterns or to extract parts respecting some selection criterion. To this end, various languages have been proposed with different levels of expressiveness and fields of application, such as the Tree Query Logic (TQL) (Cardelli et Ghelli, 2001), XML Schema (Fallside et Walmsley, 2004), Schematron/Schemapath (Coen *et al.*, 2004), XPath (Clark et DeRose, 1999), XQuery (Boag *et al.*, 2005), and Configuration Logic (Villemaire *et al.*, 2005a).

In Afanasiev's works (Afanasiev *et al.*, 2004; Afanasiev, 2004), a method for validating formulæ on tree structures by means of model checking has been presented. More precisely, the authors show how the formulæ of the logical fragment of Core XPath (Gottlob *et al.*, 2002), called Simple XPath, can be translated into Computation Tree Logic (CTL) (Clarke *et al.*, 2000). By this framework, the problem of checking a Simple XPath formula on a given XML tree amounts to checking a CTL formula on a suitably constructed Kripke structure whose states and transitions reflect the nodes and links of the original tree, a result already hinted in several previous papers (Miklau et Suciu, 2002; Gottlob *et al.*, 2002).

In this paper, we demonstrate that a similar reduction can be done for a larger subset of XPath called Configuration Logic (CL). CL has been presented as a formalism tailored for validating logical properties on the configuration of network routers expressed as XML parameter-value hierarchies (Villemaire *et al.*, 2005a). The major difference between CL and Simple XPath is in the use of variables: in Simple XPath, all references to the node labels of the parsed tree are constant, whereas CL allows the use of existentially and universally quantified variables to stand for node labels and to be tested for equality. The construction we describe is inspired in part from the trace semantics of Core XPath described in Hartel's paper (2005), but adapted to a branching-logic setting.

The main contribution of our work is twofold. First, we show in section 1.2 that the use of variables in CL is closely related, and actually extends, the use of the "freeze" quantifier already developed for linear time logics (Alur et Henzinger, 1994; Demri *et al.*, 2005) to branching time logics. Second, in section 1.3, we use this approach to embed CL formulæ into CTL. Section 1.4 concludes and indicates further avenues of investigation.

1.2 Configuration Logic

In this section, we briefly recall the basic structure of Configuration Logic (CL) and its intended purpose. We further show how the variable quantification mechanism of CL extends the notion of a "freeze" quantifier.

1.2.1 Overview of CL

Simply put, CL is a logic over tree structures whose nodes are name-value pairs. It has been introduced in the context of configuration management of network routers (Villemaire *et al.*, 2005a).

In the CL framework, a *configuration* is a forest such as the one shown in Figure 1.1. This particular kind of structure has been introduced as a way of representing the configuration parameters of network routers in a hierarchical fashion that mirrors the organization of their command line interface into modes, submodes, interfaces, and so on. Clearly, a configuration can be encoded as a particular kind of XML document, and this document can be readily exchanged by standard configuration protocols such as Netconf (Enns, 2005; Hallé *et al.*, 2005).

Each node is formed of two parts: a *name* and a *value*, represented in the form "name = value". In typical network configurations, examples of names are `interface`, `router`, `ip-address`. To simplify things, we will top each forest with an additional source node connected to the roots of all trees and consider from now on that a configuration is a tree. For example, Figure 1.1 shows a configuration formed of two trees whose respective roots, a = 1 (node 1) and a = 6 (node 2), are linked to a source node.

Definition 1.1. *A* configuration *is a structure of the form* $\langle V, N, \tilde{R}_1, ..., \tilde{R}_n \rangle$ *where:*

- V *is a set, whose elements are called* values.

- *N is a set of words closed under prefix, on the alphabet formed of $(p = v)$, with p a name and $v \in V$. The elements of N are called* nodes.
- *$\tilde{R}_1, \ldots, \tilde{R}_n$ are relations on V (i.e. subsets of $V^{arity(R_1)}, \ldots, V^{arity(R_n)}$ respectively).*

In this paper, the only relation we consider is equality.

Named paths The succession of name-value pairs from the source node to an arbitrary node is called a *named path*; two nodes are considered identical if they have the same named path. For example, the following expression, when referring to Figure 1.1, corresponds to a traversal of the configuration that starts at the source node and goes all the way down to node 10.

$$\mathrm{a} = 6\,,\ \mathrm{b} = 6\,,\ \mathrm{e} = 7 \tag{1.1}$$

A named path can have one or more variables in place of the value part of some nodes. In this case, depending on the values taken by those variables, the named path will designate different nodes. For example, the named path

$$\mathrm{a} = 6\,,\ \mathrm{d} = x \tag{1.2}$$

designates node 6 if $x = 7$, node 7 if $x = 3$, and no node of the configuration otherwise. Variables standing for the name of nodes are not authorized.

Finally, the $*$ symbol is a shortcut that can be replaced by any number of nodes in a named path. For example, the named path

$$*\,,\ \mathrm{d} = 3 \tag{1.3}$$

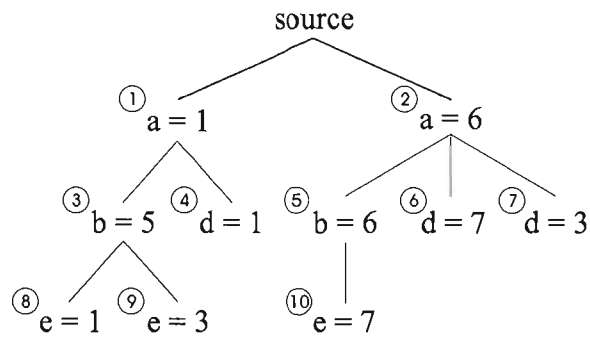


Figure 1.1: A sample configuration composed of two trees linked by a source node. Names and values are abstract.

designates a node "d = 3" at any depth in the tree; in the case of Figure 1.1, this expression corresponds to two named paths for the nodes 4 and 7. This shortcut symbol has a tricky semantics and is not part of the original definitions (Villemaire *et al.*, 2005a), as no real-world network property uses it. It is only included here for the sake of later comparison with Simple XPath.

Quantifiers CL is a formalism tailored for expressing logical properties on the values occurring in the nodes of configurations. It is composed of the traditional Boolean connectives and allows existential ($\langle\rangle$) and universal ($[]$) quantification on the "value" part of "name = value" pairs. Quantification in CL resembles in some way to the restricted "next" operator in sub-LTL (Muscholl et Walukiewicz, 2004) which does not consider all possible future states, but only a subset of them that has a specific label.

$$\langle \overline{p} = \overline{x}\,;\, n = x \rangle\, \varphi \tag{1.4}$$

Equation 1.4 shows the form of an existential quantifier where $\overline{p} = \overline{x}$ is a (possibly empty) named path, n is a name and x is a variable free in φ. Note that in this quantification, only x is considered bound; the other variables possibly occurring in $\overline{p}$ are considered free. Therefore, as stated above, the actual admissible values for x

depend on how the named path that precedes it is valuated. Semantically, the previous quantification means that there exists a value c of x for which the node designated by the named path $\overline{p} = \overline{x}\,,\ n = c$ exists, such that $\varphi[x/c]$ is true.

$$[\overline{p} = \overline{x}\,;\ n = x]\ \varphi \tag{1.5}$$

Universal quantification, shown in equation 1.5, likewise asserts that all values c for which the node designated by the named path $\overline{p} = \overline{x}\,,\ n = c$ exists, are such that $\varphi[x/c]$ is true.

Formally, the semantics of CL is the following (Villemaire *et al.*, 2005a):

Definition 1.2. *Let $\mathcal{C} = \langle V, N, \tilde{R}_1, ..., \tilde{R}_n \rangle$ be a configuration and ρ be a valuation for this configuration. We say that $\mathcal{C}, \rho$ satisfies a configuration logic formula φ (in notation $\mathcal{C}, \rho \models \varphi$), if recursively:*

- $\mathcal{C}, \rho \models R_i(\bar{x})$ *if* $\tilde{R}_i(\rho(\bar{x}))$ *holds*
- $\mathcal{C}, \rho \models \varphi \wedge \psi$ *if* $\mathcal{C}, \rho \models \varphi$ *and* $\mathcal{C}, \rho \models \psi$
- $\mathcal{C}, \rho \models \varphi \vee \psi$ *if* $\mathcal{C}, \rho \models \varphi$ *or* $\mathcal{C}, \rho \models \psi$
- $\mathcal{C}, \rho \models \neg\varphi$ *if* $\mathcal{C}, \rho \not\models \varphi$
- $\mathcal{C}, \rho \models \langle \bar{p} = \bar{x}; p = x \rangle \varphi$ *if there exists a* $v \in V$ *such that* $(\bar{p} = \rho(\bar{x}))(p = v) \in N$ *and* $\mathcal{C}, \rho[x/v] \models \varphi$
- $\mathcal{C}, \rho \models [\bar{p} = \bar{x}; p = x] \varphi$ *if for all* $v \in V$ *such that* $(\bar{p} = \rho(\bar{x}))(p = v) \in N$ *it holds that* $\mathcal{C}, \rho[x/v] \models \varphi$

We suppose that formulæ are *well-named* —that is, each variable is quantified only once. It can be easily shown that every CL formula can be transformed to an equivalent well-named formula. As previously explained, the initial semantics of CL can be extended by allowing the condition $(\bar{p} = \rho(\bar{x}))(p = v) \in N$ to include $*$ symbols that

stand for any number of nodes.

Sentences A sentence is a closed CL formula. For example, the formula

$$\begin{aligned} &\langle\, ;\, \mathrm{a} = x_1 \rangle \\ &\quad \langle \mathrm{a} = x_1\, ;\, \mathrm{b} = x_2 \rangle \\ &\quad\quad x_1 \neq x_2 \end{aligned} \tag{1.6}$$

states that there exists a child of the source node with name "a", which itself has a child with name "b" and a different value. One can see that this property is true for the tree of Figure 1.1 by considering nodes 1 and 3. The named path figuring in a quantifier is important, as it constrains the admissible values of a variable. For example, the following formula differs only from equation 1.6 in the named path of the second quantifier.

$$\begin{aligned} &\langle\, ;\, \mathrm{a} = x_1 \rangle \\ &\quad \langle\, ;\, \mathrm{b} = x_2 \rangle \\ &\quad\quad x_1 \neq x_2 \end{aligned} \tag{1.7}$$

It states that there exists a child of the source node with name "a", and another child of the source node with name "b", and that both have different values; this formula is false on the tree of Figure 1.1.

CL also allows to express constraints on values in more than one branch. As a more complex example, consider the following formula:

$$\begin{aligned} &[\, ;\, \mathrm{a} = x_1] \\ &\quad \langle \mathrm{a} = x_1\, ;\, \mathrm{b} = x_2 \rangle \\ &\quad\quad \langle \mathrm{a} = x_1\, ;\, \mathrm{d} = x_3 \rangle \\ &\quad\quad\quad x_2 \neq x_3 \end{aligned} \tag{1.8}$$

This formula states that every child of the source with name "a" has at least two children: one of name "b", and one of name "d" that have different values. For example, in Figure 1.1, the first node of name "a" under the source is node 1; it has a "d" child in node 4, whose value is different from the "b" node 3. One can find a similar pairing (nodes 5 and 6) under node 2, and therefore equation 1.8 is true on the configuration of Figure 1.1. Remark that in this case, the values that are ultimately compared (x_2 and x_3) are taken along two different branches: this is what gained CL the appellation of a *multi-site* modal logic.

The reader is directed to (Villemaire *et al.*, 2005a) for a more detailed presentation of CL, and to (Hallé *et al.*, 2004) for further examples of applications to network constraints.

1.2.2 CL versus Simple XPath

We now proceed to sketch the differences between CL and Simple XPath (Afanasiev, 2004; Gottlob *et al.*, 2002).

First, while CL is a *logic* whose sentences are either true or false for a given configuration, Simple XPath is a language aimed at extracting parts of an XML document according to some criteria. Therefore, XPath expressions are not sentences that are true or false, but queries that return a set of nodes called an *answer set*. However, a Simple XPath query φ can be converted to a sentence by interpreting it as the assertion "the answer set of φ contains the root of the tree", which is either true or false (Afanasiev *et al.*, 2004).

Second, one must remark that CL separates a node between name and value; Simple XPath does not make this distinction. However, generic XML trees can be transformed into configurations by taking each label as the value of the node, and by appending the same dummy name, say "n", to each node.

Apart from that, one can see that the semantics of Simple XPath (Afanasiev *et al.*, 2004) is very similar to CL. Actually, the interpretations of all predicates and Boolean connectives coincide with CL's. A Simple XPath "sentence" is a CL formula where equality testing only occurs between a variable and a constant, but never between two variables. For example, consider the following XPath query called Q1 in Afanasiev's paper (2004):

$$\begin{aligned} &/[\text{child::site/child::regions/child::africa/} \\ &\quad \text{child::item/child::description/} \\ &\quad \text{child::parlist/child::listitem/child::text}] \end{aligned} \tag{1.9}$$

This formula can be translated to the following CL formula:

$$\begin{aligned} &\langle ;\, \mathrm{n} = x_1 \rangle \\ &\quad \langle \mathrm{n} = x_1 \,;\, \mathrm{n} = x_2 \rangle \\ &\quad \ldots \\ &\quad \langle \mathrm{n} = x_1 \,,\, \mathrm{n} = x_2 \,,\, \mathrm{n} = x_3 \,,\, \mathrm{n} = x_4 \,, \\ &\qquad \mathrm{n} = x_5 \,,\, \mathrm{n} = x_6 \,,\, \mathrm{n} = x_7 \,,\, \mathrm{n} = x_8 \rangle \\ &\qquad\quad x_1 = \text{site} \wedge x_2 = \text{regions} \wedge \ldots \\ &\qquad\qquad \wedge\, x_7 = \text{listitem} \wedge x_8 = \text{text} \end{aligned} \tag{1.10}$$

As another example, query Q3, stated in formula 1.11, becomes formula 1.12 in CL.

$$/\,[\text{descendant::item/descendant::text}] \tag{1.11}$$

$$\begin{aligned} &\langle *\,;\, \mathrm{n} = x_1 \rangle \\ &\quad \langle *\,,\, \mathrm{n} = x_1 \,;\, *\,;\, \mathrm{n} = x_2 \rangle \\ &\qquad x_1 = \text{item} \wedge x_2 = \text{text} \end{aligned} \tag{1.12}$$

The first quantifier fetches a node n = x_1 at an arbitrary depth under the root, while the second quantifier fetches a node n = x_2 at an arbitrary depth under the first one. The end of the formula then constrains the first node to be named "item" and the second node "text". The resulting formula is true if and only if there exists a path from the root encountering "item" at some depth, followed by "text" later on.

Conversely though, there exist CL formulæ that cannot be expressed in Simple XPath. Formulæ 1.6 and 1.8 are two examples of properties that cannot be expressed in Simple XPath, as they compare two quantified node labels, feature that Simple XPath does not allow.

There is one feature of Simple XPath that is not included into CL: the support for *relative* queries, i.e. queries φ that are evaluated at every node of a tree and whose answer set contains the nodes for which the query evaluates to true; these queries are identified in XPath by the // symbol. However, we can use the same procedure as in Afanasiev *et al.* (2004) to handle them —that is, to root the formula successively in every node of the tree, and to return the nodes for which it is true. This fact allows us to claim the translation presented here actually extends previous work.

1.2.3 CL as a Freeze Logic

The quantification process in CL bears some resemblance with the Timed Propositional Temporal Logic (TPTL) (Alur et Henzinger, 1994), and in particular with its concept of "freeze" quantifier. TPTL can be summarized as a Linear Temporal Logic (LTL) with an additional quantifier, $x.$, that allows to retain a value in a variable x at some specific instant in the evolution of a system to be compared at a later time with another value. In the particular case of TPTL, one is interested in freezing the value of a global clock. For example, the formula

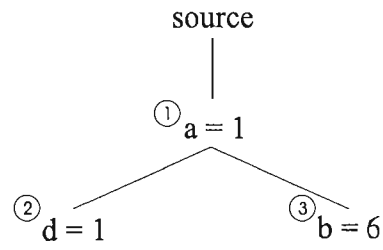


Figure 1.2: A small configuration forest

$$\mathbf{G}\, x.(p \rightarrow \mathbf{F}\, y.(q \wedge y \leq x + 10)) \tag{1.13}$$

states that at every state where p holds, there exists a future state where q holds within 10 clock ticks. When the **G** operator is evaluated, the value of the global clock where p holds is stored in x; when the **F** operator is evaluated, the value of the global clock where q holds is stored in y.

The quantification process in CL can be interpreted in a similar way. Instead of a global clock ticking at each state change, we consider the *value* part of the name-value pair encountered at each node along a name path. The named path inside a quantifier starts at the root of the tree and descends down a specific branch; at the end of the path, the value in that particular node is fetched and stored into the quantified variable. It can then be recalled further in the formula and be used either to orient the descent down a branch, or be compared for equality with another value.

For example, consider the CL formula 1.6 validated against the tree of Figure 1.2. The first quantifier, $\langle \mathrm{a} = x_1 \rangle$, starts from the source node of the configuration and descends into a node with name "a"; this is represented by arrow 1 in Figure 1.3. It freezes the value of that node into variable x_1. The second quantifier, $\langle \mathrm{a} = x_1 \,;\, \mathrm{b} = x_2 \rangle$, then restarts from the source node (arrow 2), goes down the node "a = x_1" (arrow 3), and then down a node of name "b" (arrow 4); it freezes the value of that node in x_2.

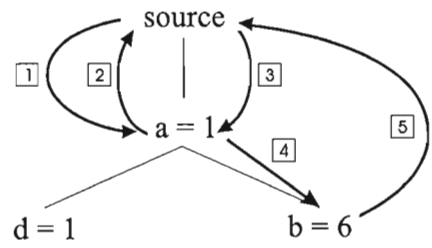


Figure 1.3: Quantification in CL is a traversal of the tree ended by the freezing of a node's value into a variable.

Finally, the last part of the formula returns back to the source node of the configuration (arrow 5) and checks that $x_1 = x_2$.

Therefore, the quantifiers in a CL sentence describe multiple passes through the tree that freeze the value of one more variable on each pass. It is therefore more than a mere CTL or LTL with freeze, which would only be able to compare values that have been frozen along the same path. Figure 1.3 shows only one possible traversal of the configuration depicted in Figure 1.2.

In the previous example, there was only one way of traversing the tree on each pass; however, in most trees, there are multiple possible passes. This would be the case in Figure 1.1. Depending on whether the quantifiers in a formula are universal or existential, the CL formula asserts something about all possible traversals or only some of them.

1.3 Embedding CL into CTL

The observation that variable quantification in CL can be seen as a special type of freeze quantification will be used to perform our embedding of CL into CTL. We proceed in two steps: first, we show how to convert a configuration T and a CL formula φ to a Kripke structure $K_{T,\varphi}$; then, we show how a CL formula φ can be translated to a CTL

formula φ' and show that $T \models_{\mathrm{CL}} \varphi$ if and only if $K_{T,\varphi} \models_{\mathrm{CTL}} \varphi'$, thereby reducing the problem of CL model checking to CTL model checking.

The resulting Kripke structure depends on φ only for the number of distinct quantified variables that appear in it. Therefore, for a fixed number of quantified variables, the translation of T is the same for all formulæ. In the following, we will simply denote it by K_T by assuming all considered formulæ have less than k variables for some $k > 0$.

1.3.1 From a Configuration to a Kripke Structure

The Kripke structure K_T is built in such a way that each state of the system is the image of some node of T.

The conversion procedure we show takes a configuration T and a CL formula φ and generates a Kripke structure $K_T = (S, I, R, L)$, where S is the set of states, I is the set of initial states, $R \subseteq S \times S$ is a total transition relation and L is a labelling function $S \to 2^{AP}$, for AP a set of atomic propositions. For the sake of clarity, we will abuse notation and use state variables that take their values in arbitrary discrete sets instead of Boolean; the set AP is therefore the set of atomic propositions that encode into Boolean variables the discrete values occurring in the structure. This operation is identical as in Afanasiev's works (Afanasiev, 2004; Afanasiev *et al.*, 2004).

Actually, each node of T will have many images in K_T; there are as many copies as there are possible combinations of values for the quantified variables appearing in the formula.

Set of states Each state of K_T (with the exception of a source and a sink node) is intended to mimic one, and only one, node of the original tree T. Let N be the set of all names appearing in T, and V be the set of all values appearing in T. For the tree of Figure 1.2, we have $N = \{a, b, d\}$ and $V = \{1, 6\}$. From these sets, create the sets N'

and V' by adding a special, unused symbol $\#$ that will stand for "undefined". A state of S is uniquely identified by $k+2$ state variables:

- $\alpha \in N'$ contains the name of the node in the original tree
- $\beta \in V'$ contains the value of the node in the original tree
- $x_1, \ldots, x_k \in V'$ contain the values of the quantified variables that are frozen when a formula is evaluated; they can hold either a node value, or the "undefined" symbol $\#$.

The set S of states of K_T is the subset of $N' \times V' \times V'^k$ such that $s = (\alpha, \beta, x_1, \ldots x_k) \in S$ if and only if there exists a node in T labelled $\alpha = \beta$. By convention, the special source node of every configuration has both name and value equal to $\#$. For example, if T is the tree of Figure 1.2 and φ is formula 1.6, then the state where $\alpha = a$, $\beta = 1$, $x_1 = x_2 = \#$ is an element of S that is associated with node 1.

One can remark that for each node in T, there are multiple states in S that are associated to it, namely one for each possible valuation of the $x_1, \ldots, x_k$. We will call a *copy* of T the subset of S in which the x_i are valuated in the same way. There are $|V'|^k$ copies of T in S. The set I contains only one initial state: it is the source state of the copy of T where all x_i are undefined, i.e. $(\#, \#, \#, \ldots, \#)$.

Transition relation Next, we define the transition relation R of that Kripke structure. There are two kinds of transitions in R. Let $s_1 = (\alpha_1, \beta_1, x_{1,1}, x_{1,2}, \ldots, x_{1,n})$ and $(\alpha_2, \beta_2, x_{2,1}, x_{2,2}, \ldots, x_{2,n})$ be two states in S.

The first set of transitions are *tree* transitions: they correspond to a descent into the original tree. The tree transitions in R link all the states inside each copy of T according to the original tree structure. In such transitions, no quantified variable changes its value. Formally, this represents the transitions (s_1, s_2) in R such that s_2 is a child of s_1 in the original tree and $x_{1,i} = x_{2,i}$ for all $1 \leq i \leq k$. In Figure 1.3, the arrows 1, 3 and 4 are

represented in K_T by tree transitions.

The second set of transitions are the *freeze* transitions exposed in section 1.2.3. They correspond to the arrows 2 and 5 of Figure 1.3, when the value of a node is frozen into a variable and the pass into the tree is ended by a return to the source node. In K_T, these transitions always lead to the source state of a copy of the original tree where all variables keep their values, except for one x_i that switches from # to some defined value. The value that this x_i takes is the value (β) of the node from which the switch has been made, therefore depicting the freezing of that value into x_i. Formally, this represents the transitions (s_1, s_2) in T such that there exists a $1 \leq j \leq n$ such that $x_{1,i} = x_{2,i}$ for all $i \neq j$, $\beta_1 = v$ for some $v \in V$, $\alpha_2 = \beta_2 = \#$, $x_{1,j} = \#$ and $x_{2,j} = v$.

As has been explained in section 1.2.3, a quantified CL formula describes multiple passes down T that each freeze a node value into some variable. A trace in K_T represents just that: a sequence of descents down T along some named paths; at the end of each named path, a variable x_i is quantified, and the value v of the node at this point is frozen in x_i. This freezing is done by switching to the copy of T where x_i has value v. The traversal of T caused by the next quantifier then restarts from its source node, which is the image of T's source state in the copy where x_i has been frozen by the previous quantifier.

Optimizations It is possible to further optimize the structure. First, one can assume that the variables are always quantified in the order $x_1, x_2, \ldots, x_n$. Moreover, every formula that does not respect this condition can be transformed by a simple renaming of its variables to a formula that complies with this restriction. Doing so reduces the size of the structure and the number of transitions, as only one variable at a time can change from # to some value.

Second, the copies of T where all variables are defined can be trimmed of everything but their source state. This is possible because at that point in the evaluation of a

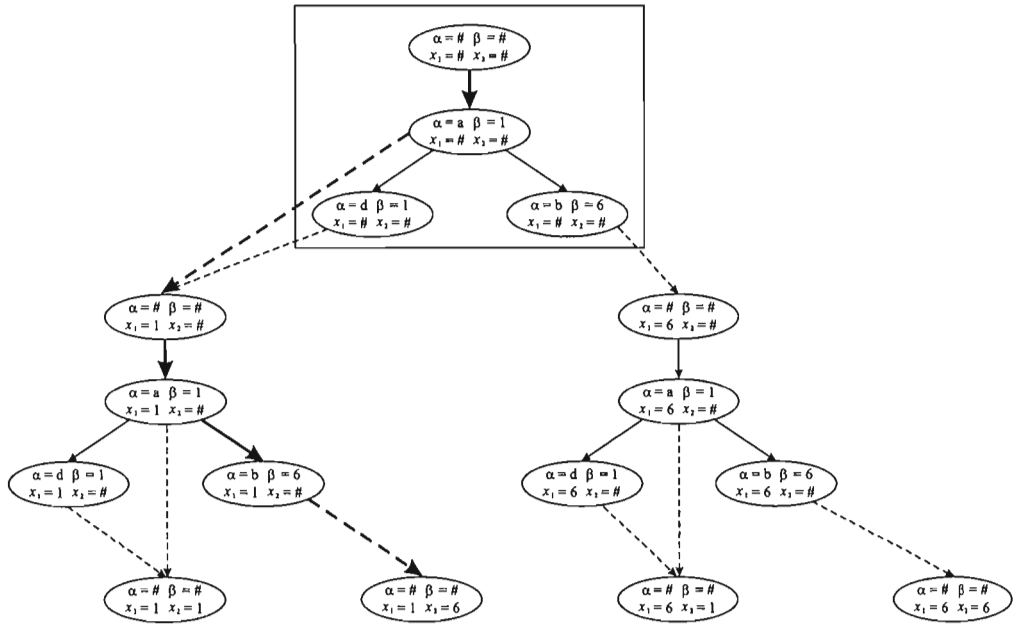


Figure 1.4: The resulting transition system K_T obtained from Figure 1.2 and formula 1.6.

formula, no new variable can be quantified, and therefore no further pass down the tree is possible. Hence, the nodes of these copies of T are useless, since they can never be accessed.

The resulting Kripke structure obtained from Figure 1.2 and formula 1.6 is shown in Figure 1.4. Solid arrows represent tree transitions; dotted arrows are the freeze transitions. The optimizations discussed above have been applied to the graph for the sake of clarity; however, the remaining of the discussion will suppose a complete structure as defined previously.

1.3.2 From CL to CTL

From the Kripke structure K_T constructed in the previous section, it becomes straightforward to translate CL's semantics into CTL. We proceed define a linear em-

bedding ω of CL into CTL formulæ, with ϵ as the empty string. Let φ and ψ be CL formulæ, c be a constant in V, m and n be names in N, and the x_i be quantified variables. Translating the Boolean connectives and the ground equality testings is direct:

$$\omega(x_i = c) = x_i = c \tag{1.14}$$

$$\omega(x_i = x_j) = x_i = x_j \tag{1.15}$$

$$\omega(\varphi \wedge \psi) = \omega(\varphi) \wedge \omega(\psi) \tag{1.16}$$

$$\omega(\varphi \vee \psi) = \omega(\varphi) \vee \omega(\psi) \tag{1.17}$$

$$\omega(\neg\varphi) = \neg\omega(\varphi) \tag{1.18}$$

The translation of named paths and quantifiers is slightly more complicated. It proceeds by nesting one element of the named path at a time until reaching the semicolon; as a consequence of the semantics of CL, this process is identical in both existential and universal quantifiers. Then, the quantified part is translated. Here as before, we designate by $\overline{p}$ a (possibly empty) named path as defined in section 1.2.1.

$$\begin{aligned} &\omega(\langle n = x_i\,,\, \overline{p}\,;\, m = x_i\rangle\varphi) = \\ &\quad \mathbf{EX}\,(\alpha = n \wedge \beta = x_i \\ &\qquad \wedge\, \omega(\langle \overline{p}\,;\, m = x_i\rangle\varphi)) \end{aligned} \tag{1.19}$$

$$\begin{aligned} &\omega([n = x_i\,,\, \overline{p}\,;\, m = x_i]\,\varphi) = \\ &\quad \mathbf{EX}\,(\alpha = n \wedge \beta = x_i \\ &\qquad \wedge\, \omega([\overline{p}\,;\, m = x_i]\,\varphi)) \end{aligned} \tag{1.20}$$

$$\begin{aligned}\omega(\langle *\,,\,\overline{p}\,;\,m = x_i\rangle\varphi) = \\ \mathbf{EF}\,\omega(\langle\overline{p}\,;\,m = x_i\rangle\varphi)\end{aligned} \tag{1.21}$$

$$\begin{aligned}\omega([*\,,\,\overline{p}\,;\,m = x_i]\,\varphi) = \\ \mathbf{EF}\,\omega([\overline{p}\,;\,m = x_i]\,\varphi)\end{aligned} \tag{1.22}$$

$$\begin{aligned}\omega(\langle\,;\,n = x_i\rangle\varphi) = \\ \mathbf{EX}\,((\alpha = n \wedge x_i = \#) \\ \wedge\,\mathbf{EX}\,(x_i \neq \# \wedge \omega(\varphi)))\end{aligned} \tag{1.23}$$

$$\begin{aligned}\omega([\,;\,n = x_i]\,\varphi) = \\ \mathbf{AX}\,((\alpha = n \wedge x_i = \#) \\ \rightarrow \mathbf{EX}\,(x_i \neq \# \wedge \omega(\varphi)))\end{aligned} \tag{1.24}$$

For example, formula 1.6 is translated to the following CTL expression:

$$\begin{aligned}&\mathbf{EX}\,((\alpha = \mathrm{a} \wedge x_1 = \#) \wedge \\ &\quad\mathbf{EX}\,(x_1 \neq \# \wedge \\ &\qquad\mathbf{EX}\,(\alpha = \mathrm{a} \wedge \beta = x_1 \wedge \\ &\qquad\quad\mathbf{EX}\,((\alpha = \mathrm{b} \wedge x_2 = \#) \wedge \\ &\qquad\qquad\mathbf{EX}\,(x_2 \neq \# \wedge \\ &\qquad\qquad\quad x_1 \neq x_2)))))\end{aligned} \tag{1.25}$$

The trace that makes this formula true on K_T is shown in bold arrows in Figure 1.4. Note that this trace represents two passes in the tree where appropriate node values are frozen into variables x_1 and x_2, as explained in section 1.2.3. The trace in K_T is nothing but the succession of states, including the values of frozen variables, that occur when

one traverses the original tree as shown in Figure 1.3

Theorem 1.1. *Let T be a configuration, φ be a CL formula and ω be the embedding defined previously. Let K_T be the Kripke structure built as shown in the previous section. Then $T \models_{CL} \varphi$ if and only if $K_T \models_{CTL} \omega(\varphi)$.*

Proof. To demonstrate the equivalence of this embedding, we need to introduce the notion of *restriction* of a Kripke structure K_T to some valuation ρ. This restriction, noted $K|_\rho$, is the Kripke structure obtained from K by keeping only the states where the values of the x_i agree with ρ. More precisely, we have that $\rho(x_i) = v \leftrightarrow K|_\rho \models_{CTL} x_i = v$ for $v \in V$. Constants are mapped identically. The initial state of $K|_\rho$ is the state $\alpha = \#$, $\beta = \#$ with the smallest set of defined variables that agree with ρ. One can easily show that, from the construction of K, this state is unique.

The proof is then done by structural induction. The base case is ground equality testing. $T, \rho \models_{CL} x_i = x_j$ if and only if $\rho(x_i) = \rho(x_j)$, by Definition 1.2. But this is true in turn if and only if both $K|_\rho \models_{CTL} x_i = \rho(x_i)$ and $K|_\rho \models_{CTL} x_j = \rho(x_j)$, which is equivalent to $K|_\rho \models_{CTL} x_i = x_j$. A similar reasoning can be made in the case of the comparison of a variable with a constant.

For the induction step, we check the cases one by one.

$T, \rho \models_{CL} \varphi \wedge \psi$ if and only if $T, \rho \models_{CL} \varphi$ and $T, \rho \models_{\mathrm{CL}} \psi$, by Definition 1.2. By the induction hypothesis, these two assertions are equivalent to $K|_\rho \models_{\mathrm{CTL}} \omega(\varphi)$ and $K|_\rho \models_{\mathrm{CTL}} \omega(\psi)$, which by equation 1.16 is equal to $K|_\rho \models_{\mathrm{CTL}} \omega(\varphi \wedge \psi)$. A similar reasoning can be done for disjunction and negation.

The case of quantification is more complex. As stated in Definition 1.2, $T, \rho \models_{\mathrm{CL}} \langle \overline{p}\,;\, n = x_i \rangle \varphi$ if and only if

1. there exists a $v \in V$ such that $\rho(\overline{p}\,,\, n = v)$ is a node of T and

2. $T, \rho[x_i/v] \models_{\mathrm{CL}} \varphi$.

The first part is true if and only if there exists a path from the initial state of $K|_\rho$ that matches $\rho(\overline{p}, n = v)$; but this is direct from the translation of named paths in equations 1.19–1.22 and the way $K|_\rho$ has been constructed in Section 1.3.1.

The second part is true if and only if there exists a way to extend ρ to ρ' in such a way that $\rho = \rho'$ except for $\rho'(x_i) = v$. Because of the manner in which K is built, this happens if there exists a way to transit from the current copy of T to a copy where x_i is no longer undefined (#), but rather takes value v (the same v as the value of the node at the end of the named path); in such a transition, all x_j $(j \neq i)$ keep their already defined values, except for x_i which switches from # to v. Moreover, such a transition leads to a copy of the source node of T; what is accessible from this state is nothing but $K|_{\rho'}$. Then, by induction hypothesis, we have that $T, \rho' \models_{\mathrm{CL}} \varphi$ if and only if $K|_{\rho'} \models_{\mathrm{CTL}} \omega(\varphi)$. A similar reasoning can be made for the case of the universal quantifier. □

1.3.3 Theoretical Consequences

The embedding described in Section 1.3.2 is linear; that is, if we denote by $|\varphi|$ the length of a CL formula φ, then $|\omega(\varphi)| \in O(|\varphi|)$. It suffices to remark that each translation rule consumes at least one symbol of the original CL formula and contributes a fixed number of symbols in the resulting CTL formula. Moreover, as explained in section 1.3.1, K_T has $|V'|^k$ copies of T, with k the maximum number of quantified variables occurring in φ; but clearly, $|V| \leq |T|$, as there cannot be more different values than there are nodes in T, and therefore $|K_T| \in O(|T|^k)$.

Since the problem of validating a CL formula can be reduced to the problem of model checking a particular CTL formula, all theoretical results proved for CTL also apply to

this particular context of CL.

Complexity of model checking Model checking CTL formulæ is decidable and has time complexity of $O(|\varphi| \times |K|)$ (Clarke *et al.*, 2000), where $|K|$ is the size (vertices + edges) of the Kripke structure. This gives us that the model checking of CL is decidable and has time complexity of $O(|\varphi| \times |T|^k)$. This means that model checking is polynomial in the size of the tree and the size of the formula, but exponential in the number of different quantified variables appearing in φ. The optimizations suggested in section 1.3.1 do not change this complexity.

This result places CL in comparison with other tree languages and logics mentioned in Section 1.1. It can be shown that CL is a fragment of TQL. Calcagno *et al.* (2003) have shown that the model checking, validity and satisfiability problems for closed formulæ in TQL with no quantifiers is decidable. Moreover, Boneva and Talbot (2004) demonstrated that model checking in TQL is PSPACE-hard; this result also applies to CL by using the same argument. This polynomial space bound is effectively achieved when using on-the-fly model checking methods.

In turn, as section 1.2.2 showed it, Simple XPath is a fragment of CL. It has been demonstrated that Simple XPath model checking is equivalent, in terms of time complexity, to CTL model checking ($O(|\varphi| \times |T|)$) (Afanasiev, 2004), a result that can be proved in a different ways (Gottlob *et al.*, 2002). Our result is accordingly more complex, as the logic presented here is richer; however, it is interesting to remark that it preserves polynomiality in terms of formula length and tree size.

Choice of the approach These complexity results also validate the choice of approach used for embedding CL into CTL. In the method presented here, the Kripke structure is responsible for handling the values of the quantified variables by containing one copy of the original configuration for each possible assignment of the x_i and by linking these copies by freeze transitions. A different approach would have been to create a

Kripke structure K_T containing a single copy of T without taking into account the x_i as state variables. The handling of the quantified variables would be transferred instead to the formula by translating the quantifiers in the classical fashion:

$$\omega(\langle \overline{p}\,;\, n = x_i \rangle\, \varphi) = \bigvee_{v \in V(\overline{p})} \varphi\,[x_i/v] \tag{1.26}$$

$$\omega([\,\overline{p}\,;\, n = x_i]\, \varphi) = \bigwedge_{v \in V(\overline{p})} \varphi\,[x_i/v] \tag{1.27}$$

However, because of the semantics of CL, special care should be taken to consider for v only values that are at the end of the named path $\overline{p}$, and not all values of V; this is represented in the previous formulæ by $V(\overline{p})$. Thus, instead of making multiple copies of the tree in K_T according to each possible value of the x_i, one makes multiple copies of the original CL formula φ. This would lead to a Kripke structure of size $O(|T|)$, and to a formula of length $O(|\varphi|^k)$. The complexity of model checking CL formulæ in such a framework would therefore be of $O(|\varphi|^k \times |T|)$ instead of $O(|\varphi| \times |T|^k)$ as is shown here.

This alternate approach has several drawbacks. First, although the translation of a tree into a Kripke structure is straightforward, the translation of a CL formula into CTL becomes tightly coupled with the tree on which it has to be checked. This is because the translation of the quantifiers shown in equations 1.26–1.27 depends on the values occurring in the tree at the end of some specific named paths —therefore, restricting possible values to some predetermined set does not solve the problem, since the structure of each tree still has to be taken into account.

In practice, however, it is far more frequent to check the same formula on many configurations than the opposite. Therefore, the approach presented in this paper, where the translation of formulæ does not depend in any way on the tree, is more efficient. More-

over, we have shown in section 1.3.1 that given a fixed maximum number of quantified variables, the translation of a configuration into a Kripke structure is also independent of any formula.

Second, on a more technical aspect, standard model checkers such as NuSMV (Cimatti *et al.*, 2002) can easily handle systems with very large state spaces and reasonably short temporal formulæ, but are far less efficient for checking exponentially long formulæ on relatively small systems. It is therefore natural to choose an approach where the exponential is placed on system size rather than formula length.

1.4 Conclusion

In this paper, we have briefly recalled the origins of Configuration Logic (CL) and stated its differences with respect to Simple XPath. We have shown how a fragment of XPath called Configuration Logic (CL) can be embedded into Computation Tree Logic. We have also shown that this framework embeds into CTL a larger subset of XPath than a previous work and in particular allows universally and existentially quantified variables in formulæ. Finally, we have also demonstrated how the variable binding mechanism of CL can be seen as a branching-time equivalent of the "freeze" quantifier. This embedding of CL into CTL opens the way to the reverse process of using decidability and model existence of CTL to prove decidability and model existence of CL, a property that is highly desirable in the context of network configurations and that would make CL one of the few decidable tree logics around.

CHAPITRE II

SELF-CONFIGURATION OF NETWORK DEVICES WITH CONFIGURATION LOGIC

Ce chapitre est tiré de l'article suivant :

> Hallé, S., Wenaas, E., Villemaire, R., Cherkaoui, O. (2006). Self-configuration of Network Devices with Configuration Logic. In *Proceedings of Autonomic Networking, First International IFIP TC6 Conference (AN 2006)*, Springer Verlag : Lecture Notes in Computer Science 4195, 36-49.

Dans ce chapitre, on explique comment la configuration d'équipements réseau peut être générée automatiquement à partir d'un ensemble de contraintes exprimées dans le langage CL présenté au chapitre précédent. Il s'agit d'une application du problème de la satisfiabilité de CL aux réseaux autonomiques ; un premier algorithme de satisfiabilité partielle est présenté dans cet article.

Pour résoudre un tel problème, la logique dans laquelle les contraintes sont exprimées doit être *décidable*. Cependant, au moment de la publication, la logique CL n'était pas encore démontrée décidable en général. Cette question sera étudiée dans le chapitre suivant ; on montrera que la satisfiabilité de CL est décidable pour les relations unaires.

Abstract

Autonomic networking is an emerging approach to the management of computer networks that aims at developing self-governed devices. Among the main issues of autonomic systems is the question of self-configuration. In this paper, we describe a method for discovering and self-generating the configuration of a network device in order to dynamically push a new service into a network. On each configuration, several rules representing the semantics of the services are expressed in a logical formalism called Configuration Logic. From these rules, we show how to use traditional satisfiability methods to automatically generate or modify the configuration of a device with respect to the configuration of its neighbours. We illustrate our case with an example of a switch that automatically discovers its VLAN configuration when connected to an existing network. The results presented here have been implemented into the configuration management tool ValidMaker.

2.1 Introduction

Despite the tremendous development of network services and functionalities over the years, the configuration and deployment of network elements such as routers and switches remains a mostly manual task. An intricate knowledge of each devices' and services' inner workings and dependencies between configuration parameters is required from the network engineer in order to successfully run even basic use cases. The addition of a new device or the deployment of a new service to an existing infrastructure requires repetitive but careful manipulation of multiple configuration parameters on many elements, and even such a cautious work can spawn unpredicted side effects that are discovered by trial and error.

The application of the autonomic systems paradigm (Parashar et Hariri, 2004) to

computer networks offers a promising means to release the burden of knowledge and tedious manipulations currently needed from engineers. By definition, self-governed devices can automatically modulate their behaviour in reaction to configuration inconsistencies or changes in the topology or management policies of the network.

In this paper, we describe a method for automatically discovering and generating the configuration of a network device. Each configuration is represented in the form of a hierarchy of parameter-value pairs that is assimilated to a labelled tree. The dependencies between these parameters are then expressed as self-rules in a special formalism called Configuration Logic (CL) (Villemaire *et al.*, 2005a).

From the CL formula defining each rule, we show how an action can be automatically associated; this action consists in a number of configuration operations whose end result ensures that the rule is fulfilled. A CL validation engine can automatically check whether a given set of self-rules are respected in a network and trigger the appropriate actions associated with the rules that are violated.

The system then uses a standard Boolean satisfiability test to generate a plan that properly instantiates parameter values and chooses between conflicting actions; in the case where more than a single plan is possible, the final choice can then be passed on to a higher-level policy manager. This method can be used for integrating self-configuring and self-healing functionalities in any autonomic network where devices' configurations are represented by trees. We illustrate it by a simple example on Virtual LANs in which a new switch is connected and discovers its configuration in a plug-and-play manner based on data obtained from other devices in the network.

The rest of the paper is structured as follows. Section 2.2 presents related work. Section 2.3 presents the VLAN use case that illustrates our method and elicits a number of VLAN configuration self-rules. Section 2.4 presents the tree structure used to describe configurations and formalizes the configuration rules using Configuration Logic.

Section 2.5 extends the original VLAN scenario to allow for self-configuration and shows how violated VLAN rules trigger configuration operations. Finally, section 2.6 concludes and indicates possible future work.

2.2 Related Work

The autonomic approach to network management has been the source of numerous related work in recent years. Companies like Cisco and Motorola are developing similar concepts respectively called *programmable networks* (Cisco Systems, 2002) and *cognitive networks*; the idea has also been developed by Keller (2004).

The SELFCON (Boutaba *et al.*, 2001) project developed a self-configuring environment based on the Directory-enabled Networking (DEN) (Strassner, 1999) principles, where devices register at a centralized directory server that notifies them of changes in configuration policies or network state. Similarly, the AUTONOMIA (Hariri *et al.*, 2003) environment provides an autonomic architecture for automated control and management of networked applications. In a work by Pujolle and Gaïti (2004), a suite of protocols compatible with TCP/IP is described that could be implemented into autonomic, agent-based "smart routers" that take decisions about which protocol in the suite should be used to optimize some user-defined goals. All these projects describe an infrastructure in terms of high level concepts and do not concentrate on the representation, validation and actual generation of configurations and rules, but rather provide an environment in which these operations can take place. The agent approach is extended by Gaïti *et al.* (2005) from a quality of service perspective and is currently under development.

In another work (Golab et Boutaba, 2004), the parameters of an optical network are automatically reconfigured based on link traffic using regression techniques. However, the range of legal values of these parameters is fixed and known in advance and

the reconfiguration only aims at finding an optimal adjustment of existing values: the network itself is supposed to be properly working at any moment. Our work rather attempts to structurally modify a configuration by adding and removing parameters. Moreover, in our situation, the legal range of values changes from time to time and our method attempts to discover that range from the configuration rules.

The GulfStream software system (Fakhouri *et al.*, 2001) provides a dynamic topology discovery and failure detection for clusters of switches in multiple VLANs; the approach is not based on the examination of the configuration of other switches, but rather on the broadcasting of BEACON and heartbeat messages between VLAN peers and is somewhat restrained to this particular situation.

Our approach also differs from well-established configuration systems like Cfengine (Couch et Gilfix, 1999) in that only the desired properties of the configuration are expressed in a declarative way, but no action or script must be specified by the user. The method we present automatically determines the proper actions to take on the configuration in order to fulfil the desired rules.

Finally, a lot of work has been published about self-configuration applied to wireless sensor networks. In this context, the word "configuration" generally refers to the topological arrangement of the different elements forming the sensor mesh, and not the logical parameters that regulate the behaviour of a device in itself. It is therefore only faintly related to the present work.

2.3 Constraints and Rules in Network Services

In this section, we develop a configuration example based on Virtual Local Area Networks and the Virtual Trunking Protocol. We express configuration self-rules for a configuration of this type to be functional. This example will later be used to show how

we can find the appropriate configuration of a device in a self-configuring and self-healing context.

2.3.1 Virtual Local Area Networks and the Virtual Trunking Protocol

Switches allow a network to be partitioned into logical segments through the use of Virtual Local Area Networks (VLAN). This segmentation is independent of the physical location of the users in the network. The ports of a switch that are assigned to the same VLAN can communicate at Layer 2 while ports not assigned to the same VLAN require Layer 3 communication. All the switches that need to share Layer 2 intra-VLAN communication need to be connected by a *trunk*. IEEE 802.1Q (IEEE, 2003) and VTP (Cisco Systems, 2006) are two popular protocols for VLAN trunks.

In principle, for a VLAN to exist on a switch, it has to be manually created by the network engineer on the said switch. The Virtual Trunking Protocol (VTP) (Cisco Systems, 2006) has been developed on Cisco devices to centralize the creation and deletion of VLANs in a network into a VTP *server*. This server takes care of creating, deleting, and updating the status of existing VLANs to the other switches sharing the same VTP *domain*.

2.3.2 Constraints on VLAN Configurations

Our configuration example involves VTP. Consider a network of switches such as the one shown in Figure 2.1 where several VLANs are available. We express a number of VTP self-rules that must be true across this network.

First, in order to have a working VTP configuration, the network needs a unique VTP server; all other switches must be VTP clients. This calls for a first set of two self-rules:

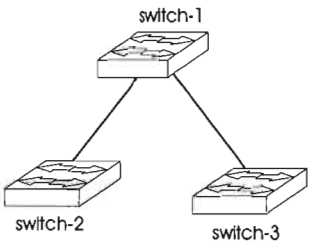


Figure 2.1: A simple cluster of switches in the same VLAN. The links are VLAN trunks.

VTP Self-Rule 1. *The VTP must be activated on all switches.*

VTP Self-Rule 2. *There is a unique VTP server.*

We impose that all switches be in the same VTP domain, and that if two switches are connected by a trunk, then this trunk must be encapsulated in the same mode on both interfaces. This gives us two more constraints that should be true at all times:

VTP Self-Rule 3. *All switches must be in the same VTP domain.*

VTP Self-Rule 4. *The interfaces at both ends of a trunk should be defined as such and encapsulated in the same mode.*

2.4 Configurations and Rules

In this section, we give a brief overview of Configuration Logic (CL), a logical formalism over tree structures developed for expressing properties of configurations. We show how the VTP Self-Rules described in the previous section can be rewritten in CL to become Formal VTP Self-Rules.

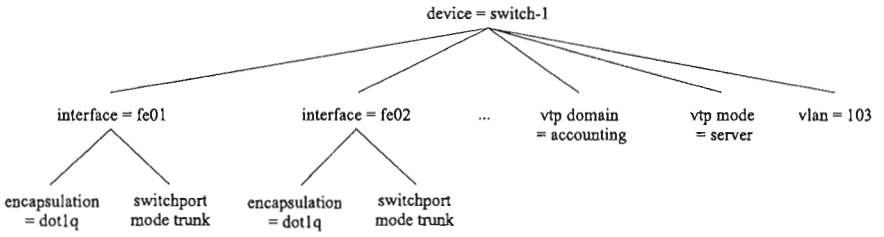


Figure 2.2: A portion of the configuration of the `switch-1` in the network of Figure 2.1. The configuration of `switch-2` and `switch-3` differs in the VTP mode and trunk information.

2.4.1 Representing Configurations

As has already been explained (Hallé *et al.*, 2004), the configuration of network devices such as routers and switches can be represented as a tree where each node is a pair composed of a name and a value. This tree represents the hierarchy of parameters inherent to the configuration of such devices. As an example, Figure 2.2 shows a tree representation of the configuration of `switch-1` in the network of Figure 2.1.

Building a tree from an XML document is trivial; therefore, the representation of configurations as trees closely matches the XML nature of a protocol such as Netconf (Enns, 2005) that uses this format to fetch and modify the configuration of a device.

2.4.2 A Logic for Configurations

Configuration Logic (Villemaire *et al.*, 2005a) was developed to express properties on configuration trees. CL formulæ use the traditional Boolean connectives of predicate logic: $\wedge$ ("and"), $\vee$ ("or"), $\neg$ ("not"), $\rightarrow$ ("implies"), to which two special quantifiers are added. The universal quantifier, identified by [ ], indicates a path in the tree and imposes that a formula be true for all nodes at the end of that path. Likewise, the existential quantifier, identified by $\langle\ \rangle$, indicates a path in the tree and imposes that a formula

be true for some node at the end of that path. Quantifiers have the lowest operator precedence.

To improve readability of CL rules, we can use predicates. Predicates are expressed in the same way as rules, with the exception that arguments which correspond to already quantified nodes can be specified. For example, to create a predicate that returns true if a switch is a VTP Client, one can do so by writing:

$$\begin{aligned}&\text{IsVTPClient}(S) \ :- \\ &\quad \langle S \,;\, \text{vtp mode} = x \rangle \, x = \text{client}\end{aligned}$$

This predicate states that under the node S passed as an argument, there exists a node whose name is "vtp mode" and whose value is x, and where x is equal to "client". In other words, this predicate is true whenever S has a child labelled "vtp mode = client", which means that the device is indeed a VTP client.

Similarly, the predicates IsVTPServer and IsTrunk respectively indicate that the device is a VTP server and that a specific interface is connected to a trunk. They are defined as the following:

$$\begin{aligned}&\text{IsVTPServer}(S) \ :- \\ &\quad \langle S \,;\, \text{vtp mode} = x \rangle \, x = \text{server}\end{aligned}$$

$$\begin{aligned}&\text{IsTrunk}(I) \ :- \\ &\quad \langle I \,;\, \text{switchport mode} = x \rangle \, x = \text{trunk}\end{aligned}$$

The next predicate asserts that two switches are in the same VTP domain. This is done by checking that for two nodes S and T representing the root of the configuration tree of two devices, every VTP domain listed under S also appears under T.

$$\begin{aligned}&\text{SwitchesInSameVTPDomain}(S,T)\ :-\\&\quad[S\,;\ \text{vtp domain} = x]\\&\qquad\langle T\,;\ \text{vtp domain} = y\rangle\ x = y\end{aligned}$$

Finally, this last predicate verifies that the encapsulation on a VLAN trunk is either IEEE 802.11Q or ISL, and that both ends use matching protocols:

$$\begin{aligned}&\text{SameEncapsulation}(I_1,I_2)\ :-\\&\quad[I_1\,;\ \text{switchport encapsulation} = x_1]\\&\qquad\langle I_2\,;\ \text{switchport encapsulation} = x_2\rangle\\&\qquad\quad(x_1 = \text{dot1q} \wedge x_2 = \text{dot1q}) \vee (x_1 = \text{isl} \wedge x_2 = \text{isl})\end{aligned}$$

For more details about CL, the reader is directed to a previous paper (Villemaire *et al.*, 2005a; Villemaire *et al.*, 2005b; Hallé *et al.*, 2006b).

2.4.3 Rules Expressed in Configuration Logic

Equipped with the previous predicates, the VTP self-rules mentioned in section 2.3 can now be expressed in Configuration Logic. The first rule checks whether VTP is activated on all switches.

Formal VTP Self-Rule 1 (VTPActivated).

$$\begin{aligned}&[\text{device} = s_1]\\&\quad\text{IsVTPServer}(s_1) \vee \text{ISVTPClient}(s_1)\end{aligned}$$

This is done by stating that for every device s_1, either s_1 is a VTP server, or s_1 is a VTP client. Note that this rule uses the predicates defined above to simplify its

expression; however, the predicates are not necessary, and their definition could have simply been copied in the rule instead. To simplify the notation in the formula, we write the argument of the predicate as s_1, although we actually pass the *node* "device = s_1".

The second rule makes sure that there is one, and only one server in the network. It first states that there exists a device s_1 which is a VTP server, and then that every device s_2 different from s_1 is a VTP client. Remark that this rule subsumes the previous one, which has still been included to illustrate later concepts.

Formal VTP Self-Rule 2 (UniqueServer).

$$\begin{array}{l}\langle \text{device} = s_1 \rangle \\ \quad (\text{IsVTPServer}(s_1) \wedge \\ \quad\quad [\text{device} = s_2] \\ \quad\quad\quad s_1 \neq s_2 \rightarrow \text{IsVTPClient}(s_2))\end{array}$$

The third rule checks that all switches in the network have the same VTP Domain.

Formal VTP Self-Rule 3 (SameVTPDomain).

$$\begin{array}{l}[\text{device} = s_1] \\ \quad [\text{device} = s_2] \\ \quad\quad \text{SwitchesInSameVTPDomain}(s_1, s_2)\end{array}$$

It does so by stating that for every pair of devices s_1 and s_2, the predicate "SwitchesInSameVTPDomain" defined previously is true.

Finally, the fourth rule checks that if two interfaces are connected, then these two interfaces must be defined as trunks and have the same encapsulation protocol.

Formal VTP Self-Rule 4 (TrunkActive).

$$
\begin{aligned}
&[\text{device} = s_1] \\
&\quad [\text{device} = s_2] \\
&\quad\quad [s_1 \,;\, \text{interface} = i_1] \\
&\quad\quad\quad \langle s_2 \,;\, \text{interface} = i_2 \rangle \\
&\quad\quad\quad\quad (\text{InterfacesConnected}(i_1, i_2) \rightarrow (\text{IsTrunk}(i_1) \\
&\quad\quad\quad\quad\quad \wedge \text{IsTrunk}(i_2) \wedge \text{SameEncapsulation}(i_1, i_2)))
\end{aligned}
$$

The relation "InterfacesConnected" is not a predicate defined in CL, but rather a system primitive that the network can tell us about. It returns true if the two interfaces are connected by a link. This requires knowledge of the topology of the network and depends on the architecture of the autonomic component, as will be discussed in the next section.

2.5 A Self-Configuration Scenario

In this section, we show how to use Configuration Logic to trigger configuration changes in devices. Figure 2.3 presents a possible architecture using this method. It shows how an existing tool called ValidMaker (Deca *et al.*, 2004) that downloads and uploads configuration trees from the configuration file stored on routers and switches, can be used off-the-shelf to provide autonomic behaviour to the devices. A CL validation engine is implemented within ValidMaker. Therefore, one can load configurations, define self-rules on these configurations and automatically verify them. If one or more rules happen to be false, an additional configuration generator can adjust the configuration trees and generate the new configurations which can then be put back on the devices.

To this end, we revisit the original VLAN scenario described in section 2.3 from a

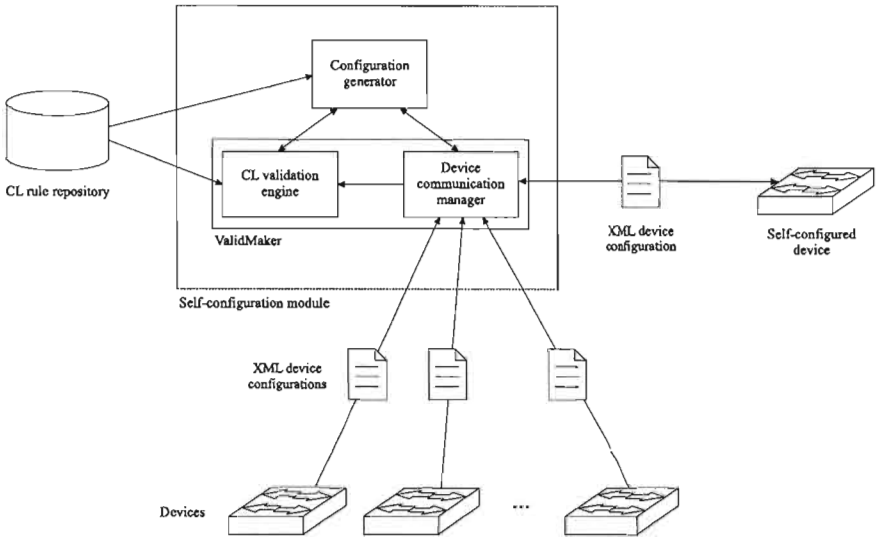


Figure 2.3: A possible self-configuring architecture using off-the-shelf components.

network autonomic viewpoint. Instead of validating configuration rules on a static, fully formed cluster of switches, we suppose one switch is connected to the cluster with a blank configuration, as shown in Figure 2.4. More precisely, the interface fe04 of `switch-4` is connected to the interface fe06 of `switch-3`. The initial configuration tree of `switch-4` is is a tree with only one root node labelled "device=switch-4".

Based on this use case, we describe a method that enables the switch to automatically discover its configuration.

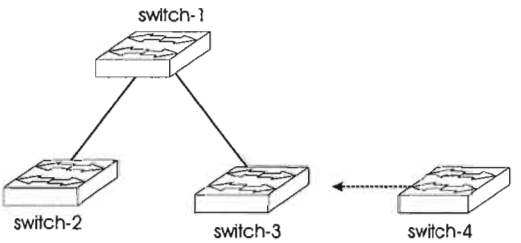


Figure 2.4: Autonomic use case. Switch 4 is connected to Switch 3 and attempts to discover its configuration.

2.5.1 Active Constraints

The network rules shown in section 2.3.2 have been up to now interpreted in a static way: their validation allowed us to determine whether or not a configuration satisfied them. However, it is possible to go further and automatically associate with each rule an action, thereby making it a "self-rule". This action, in line with the semantics of the operators used in the rule, is generated such that executing it changes the configuration in a way that fulfils the rule that was originally proved false.

As an example, consider a rule of the form $\langle \mathrm{p} = x \rangle \varphi(x)$. Such a rule imposes the existence of a node $\mathrm{p} = x$ in the configuration of the device. If it is violated, this means that no such tuple $\mathrm{p} = x$ exists; to make this rule true, the node must be added to the configuration. This addition of a new node corresponds to a configuration operation equivalent to typing a command (or a sequence of commands) to the CLI of the device. Moreover, the value x of this parameter must be such that $\varphi(x)$ is true.

For such a procedure to be possible, the formulæ it attempts to fulfil must be expressed in a logic where the existence of a model is a *decidable* problem. Otherwise, it is not always possible to find and construct a configuration satisfying a given formula. Therefore, the choice of CL as the logical formalism of formal rules is not arbitrary: it has been shown that under reasonable conditions, CL is a decidable logic (Villemaire *et al.*, 2005b). Using a richer logic such as XPath or TQL would lead to undecidability (Calcagno *et al.*, 2003).

Hence, by recursively descending through the nested CL subformulæ, it is possible to come up with a series of configuration operations and constraints on their parameters that make every self-rule true. To this end, we define a procedure called $\textsc{Plan}_T$ that recursively creates the plans for each rule given a global configuration tree T. The special notation $\oplus(\overline{p} = \overline{x}\,;\, p = x)$ indicates that a node of parameter p and value x must be

$$
\begin{aligned}
\textsc{Plan}_T(\varphi) &= 1 \text{ if } T \models \varphi \\
\textsc{Plan}_T(\neg\varphi) &= \neg\textsc{Plan}_T(\varphi) \\
\textsc{Plan}_T(\varphi \wedge \psi) &= \textsc{Plan}_T(\varphi) \wedge \textsc{Plan}_T(\psi) \\
\textsc{Plan}_T(\varphi \vee \psi) &= \textsc{Plan}_T(\varphi) \vee \textsc{Plan}_T(\psi) \\
\textsc{Plan}_T(\langle \overline{p} = \overline{x}\,;\, p = x\rangle\, \varphi(x)) &= \oplus(\overline{p} = \overline{x}\,;\, p = x) \wedge \textsc{Plan}_T(\varphi) \\
\textsc{Plan}_T([\overline{p} = \overline{x}\,;\, p = x]\, \varphi(x)) &= \bigwedge_{x \in D} \textsc{Plan}_T(\varphi)
\end{aligned}
$$

Table 2.1: The recursive plan generation procedure.

added to the tree at the tip of the branch $\overline{p} = \overline{x}$. This procedure is defined in Table 2.1. The domain D in the last line of the table is defined as $D = \{c : \overline{p} = \overline{x}, p = c \in T\}$.

As an example, consider the initial configuration of `switch-4` when added to the configuration of the original cluster of switches shown in Figure 2.2. It is clear that Formal VTP Self-Rule 1, which imposes the presence of either server or client definition on every switch, is violated for `switch-4`. The recursive application of PLAN on this formula produces the following action:

$$
\begin{aligned}
&\oplus(\text{device} = \text{switch-4}\,;\, \text{vtp mode} = \text{server}) \vee \\
&\oplus\,(\text{device} = \text{switch-4}\,;\, \text{vtp mode} = \text{client})
\end{aligned}
\tag{2.1}
$$

which commands that the VTP role of the switch be determined. This role is specified by the addition of either node vtp mode = server or vtp mode = client under the root of `switch-4`'s configuration tree.

Similarly, Formal VTP Self-Rule 2 is violated. It imposes that there is only one server in the network and that all other be clients, and produces the following action:

$$
\oplus(\text{device} = \text{switch-4}\,;\, \text{vtp mode} = \text{client}) \tag{2.2}
$$

In other words, the only possible action is the definition of `switch-4` as a client.

Formal VTP Self-Rule 3 is also violated. This rule imposes that all switches be in the same domain. The application of PLAN on this rule produces this action:

$$\oplus(\text{device} = \text{switch-4}\,;\ \text{vtp domain} = \text{accounting}) \tag{2.3}$$

Finally, the actions produced by Formal VTP Self-Rule 4 are the following:

$$\begin{aligned}
&\oplus(\text{device} = \text{switch-3}\,;\ \text{interface} = \text{fe06})\\
&\ \wedge \oplus(\text{device} = \text{switch-4}\,;\ \text{interface} = \text{fe04})\\
&\ \wedge \oplus(\text{device} = \text{switch-3}\,,\ \text{interface} = \text{fe06}\,;\ \text{switchport mode} = \text{trunk})\\
&\ \wedge \oplus(\text{device} = \text{switch-4}\,,\ \text{interface} = \text{fe04}\,;\ \text{switchport mode} = \text{trunk})\\
&\ \wedge (\\
&(\oplus(\text{device} = \text{switch-3}\,,\ \text{interface} = \text{fe06}\,;\ \text{switchport encapsulation} = \text{dot1q}) \wedge\\
&\ \oplus(\text{device} = \text{switch-4}\,,\ \text{interface} = \text{fe04}\,;\ \text{switchport encapsulation} = \text{dot1q})) \vee\\
&(\oplus(\text{device} = \text{switch-3}\,,\ \text{interface} = \text{fe06}\,;\ \text{switchport encapsulation} = \text{isl}) \wedge\\
&\ \oplus(\text{device} = \text{switch-4}\,,\ \text{interface} = \text{fe04}\,;\ \text{switchport encapsulation} = \text{isl}))\\
&)
\end{aligned} \tag{2.4}$$

This last set of actions is interesting, since it imposes addition of nodes not only under the newly connected `switch-4`, but also under `switch-3` which is already part of the working VLAN. The reason is that connecting `switch-3` to `switch-4` involves setting up a trunk, an operation that requires configuration modifications on both ends of the wire.

Therefore, the actual course of actions will depend on the architecture chosen: if

the self-configuration module depicted in Figure 2.3 is decentralized in every switch, then each switch must ignore the actions that modify other devices, assuming that the concerned devices will take proper measures to correct the situation on their side. If the self-configuration module is rather centralized for some number of devices, the changes can be sent by the device communication manager to multiple switches at once.

The global plan that must be executed to make the rules true is the conjunction of equations 2.1–2.4.

2.5.2 Generating the Configuration

It can be seen that the previous plan is nondeterministic. For example, if a formula $\langle \mathrm{p} = x \rangle\, \varphi \vee \psi$ is false, there are two different ways of making it true: either by making φ true, or by making ψ true. In such a case, both solutions are explored and propagated up the recursion stack. This happens with the actions produced by the Formal VTP Self-Rule 1 that either define the switch as client or as server. In the same way, the actions produced by the Formal VTP Self-Rule 4 rule impose that the trunk encapsulation in `switch-3` and `switch-4` be either both dot1q or both isl, but since none of them is already configured and no further constraint applies, the choice is free.

Moreover, a separate plan has been generated for each violated self-rule; however, it is well possible that these plans are mutually contradictory and that no global solution exists. Finally, one must make sure that executing the plans not only makes the violated rules true, but in turn does not produce side effects that could falsify other rules that were previously true.

To this end, two further steps are taken before committing any configuration to the device. First, a satisfying assignment of the global plan must be found. This can be easily obtained by applying a standard Boolean satisfiability (SAT) solver such as

zChaff (Moskewicz *et al.*, 2001) or SATO (Zhang et Stickel, 2000). This assignment designates the actions that are actually chosen from the many alternatives suggested by the plan to make the formulæ true.

Once this assignment has been found, the plan is tentatively executed on the configuration, and the whole set of self-rules is then revalidated against this modified configuration. If one of the rules is violated, the satisfying assignment is discarded and the procedure backtracks to find a new assignment. This is the case if the following operations are chosen as the action to execute on the configuration (for clarity, the actions on `switch-3` have been omitted):

$$\oplus(\text{device} = \text{switch-4}\,;\ \text{vtp mode} = \text{server})$$

$$\oplus(\text{device} = \text{switch-4}\,;\ \text{vtp mode} = \text{client})$$

$$\oplus(\text{device} = \text{switch-4}\,;\ \text{vtp domain} = \text{accounting})$$

$$\oplus(\text{device} = \text{switch-4}\,;\ \text{interface} = \text{fe04})$$

$$\oplus(\text{device} = \text{switch-4}\,,\ \text{interface} = \text{fe04}\,;\ \text{switchport mode} = \text{trunk})$$

$$\oplus(\text{device} = \text{switch-4}\,,\ \text{interface} = \text{fe04}\,;\ \text{switchport encapsulation} = \text{dot1q})$$

The configuration resulting of the previous operations violates Formal VTP Self-Rule 2, since it assigns `switch-4` both to the role of client and server at the same time. It violates Formal VTP Self-Rule 2 for another reason, as a server already exists in the network.

However, a second possible truth assignment to the global plan is the following:

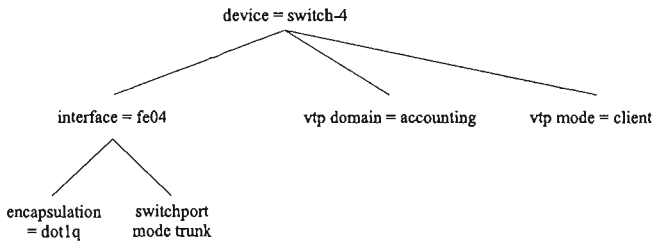


Figure 2.5: The configuration of `switch-4` after automatic configuration generation.

$$\oplus(\text{device} = \text{switch-4}\,;\ \text{vtp mode} = \text{client})$$

$$\oplus(\text{device} = \text{switch-4}\,;\ \text{vtp domain} = \text{accounting})$$

$$\oplus(\text{device} = \text{switch-4}\,;\ \text{interface} = \text{fe04})$$

$$\oplus(\text{device} = \text{switch-4}\,,\ \text{interface} = \text{fe04}\,;\ \text{switchport mode} = \text{trunk})$$

$$\oplus(\text{device} = \text{switch-4}\,,\ \text{interface} = \text{fe04}\,;\ \text{switchport encapsulation} = \text{dot1q})$$

The configuration resulting of these operations validates all formal VTP self-rules and is therefore admissible. In such a case, the tentative configuration is committed as the running configuration of the switch. The resulting configuration tree obtained from these autonomic configuration steps is shown in Figure 2.5.

One can remark that this method can also work in a self-healing context where a working network is disturbed; the recursive plan generation method provided here ensures that the configurations of the devices can be put back to a state where they fulfil all the rules that must apply on the network.

2.5.3 Complexity of the Method and Experimental Results

We now evaluate the global complexity of the method. Although Boolean satisfiability is an NP-complete problem, it is not the bottleneck of the procedure, as the global Boolean formula that is generated never has more than a few dozens, possibly a hundred, configuration operations. Therefore, the size of the Boolean instance to be processed is negligible by SAT standards, and can be processed in fractions of a second by SAT solvers accustomed to problems topping the million of variables.

Each configuration operation in itself consists solely in the addition of a single node in the configuration tree of a device and can also be considered instantaneous. Hence, the core of the complexity of this method resides in the validation of CL formulæ on multiple incarnations of configurations for many devices.

In order for this method to be tractable, the validation of CL formulæ must be quick and simple. We have already shown that CL model checking of a formula is polynomial in the size of the tree (Hallé *et al.*, 2006b).

We give in Table 2.2 the validation times for a network composed of 10 to 80 switches. These configurations were generated by a parameterizable script and then loaded into ValidMaker. All results have been obtained on a Pentium IV of 2.4 GHz with 512 Mb of RAM running Windows XP Professional; they are consistent with the polynomiality result already demonstrated.

As one can see from these results, the validation times are reasonable and do not exceed 25 milliseconds per device for the largest network of 80 switches. One should compare these figures with the time required for actually injecting a new configuration in a switch and restarting it to make it effective, which is at least a few seconds. Moreover, it should be noted that the validation experiments shown here involve the validation of the whole network, and not only of the few devices that might be concerned when a new

Switches	Validation time per device (ms)			
	Rule 1	Rule 2	Rule 3	Rule 4
10	0.024	0.012	0.106	0.564
20	0.029	0.012	0.312	1.715
40	0.044	0.024	1.011	6.482
80	0.078	0.041	3.655	21.948

Table 2.2: Validation time of each formal VTP self-rule in a network formed of a varying number of switches

switch is connected; this is especially true for VTP Self-Rule 4.

2.6 Conclusion and Future Work

In this paper, we have shown how to self-configure a network device by validating constraints on the configuration of other devices expressed as trees of parameters and values. We showed how a new switch connected to an VLAN can discover its configuration by selecting a suitable plan that tries to fulfil the existing configuration constraints. We also showed by experimental results with the ValidMaker configuration tool that the validation of self-rules is a tractable operation that requires negligible time to execute.

At the moment, the SAT instances forming the possible plans are treated separately from the CL validation in an independent solver embedded into the Configuration generator. A natural extension of this work is to enrich this active model by dealing with side effects of the application of a self-rule and support node deletion and modification, for example by integrating CL components into an existing SAT solver in the context of SAT modulo theories (Nieuwenhuis et Oliveras, 2005).

CHAPITRE III

SATISFYING A FRAGMENT OF XQUERY BY BRANCHING-TIME REDUCTION

Ce chapitre est tiré de l'article suivant :

> Hallé, S., Villemaire, R. (2008). Satisfying a Fragment of XQuery by Branching-Time Reduction. Soumis au *15th International Symposium on Temporal Representation and Reasoning (TIME 2008)*, janvier 2008.

Dans ce chapitre, la logique CL est revisitée selon l'angle de la satisfiabilité de ses formules. On a vu au chapitre précédent comment une logique satisfiable nous permet de générer de manière systématique une structure vérifiant une formule quelconque. Cette propriété nous permet, dans le cas de la gestion des configurations réseaux, de produire à partir de zéro une configuration satisfaisant un ensemble de contraintes. Il est démontré ici que le problème de la satisfiabilité de CL est décidable et qu'il peut être ramené à un problème de satisfiabilité de CTL, à l'instar de la construction présentée au chapitre 1.

Abstract

Most works relating tree languages to temporal logics consider navigational fragments where the content of tree nodes is not taken into account. In contrast, Configuration Logic (CL) is a fragment of the XML Query Language (XQuery) that allows first-order quantification over node labels. In this paper, we study CL satisfiability and seek a deterministic decision procedure to build models of satisfiable CL formulæ. To this end, we show how to revert CL satisfiability into an equivalent CTL satisfiability problem in order to leverage existing model construction algorithms for CTL formulæ and further the bond between temporal logics and logics over tree structures.

3.1 Introduction

The similarity between tree languages and temporal logic has long been recognized; early works on the subject even precede the advent of the XML technologies that made formalisms on so-called *semi-structured data* popular (Blackburn, 1993). More recently, a connection between the XML Path Language (XPath) (Clark et DeRose, 1999) and the Computation Tree Logic CTL (Clarke *et al.*, 2000) has been suggested in previous works (Miklau et Suciu, 2002; Gottlob *et al.*, 2002); fragments of XPath have also been interpreted in terms of CTL (Afanasiev *et al.*, 2004) and Propositional Dynamic Logic (PDL) (Afanasiev *et al.*, 2005). Complexity and decidability results on semi-structured data have been revisited from a modal logic perspective (Marx, 2003; Alechina *et al.*, 2003). Finally, correspondences between XML Schemata and multi-modal hybrid logic formulæ have also been established (Bidoit *et al.*, 2004). The main interest of such connections is the "well-behaved" nature of modal logics (Marx et Venema, 2007) and temporal logics in particular, an observation that has led to the development of guarded logic (Andréka *et al.*, 1998).

The majority of works relating tree languages to modal logics are considered *navigational*: they are appropriate for describing complex conditions on the structure of the tree, but are less flexible in stating relations between *values* of different tree nodes. In contrast, Configuration Logic (CL) is a strict subset of the XML Query Language (XQuery) (Boag *et al.*, 2005) that includes first-order quantification over node labels in addition to the usual Boolean and parent-child connectives; it can also be interpreted as an extension of first-order logic where the variables form a forest instead of a set (Villemaire *et al.*, 2006). CL was introduced as an appropriate fragment of XQuery that expresses data properties in configuration files of network routers (Villemaire *et al.*, 2005a).

The satisfiability of a tree logic is an important question in a number of practical contexts. In network routers, satisfiability is needed to ensure that a set of requirements expressed as CL formulæ can actually be fulfilled by a configuration; one is also interested in finding an actual model of the requirements when it exists. Benedikt *et al.* (2005) argue about the importance of satisfiability by citing other critical consistency problems: database query optimization —where the unnecessary calculation of inconsistent queries can be avoided,— information leakage in security views and consistency of XML specifications .

In this paper, we attempt to solve the problem of CL satisfiability from a temporal logic perspective. We first recall the syntax and semantics of CL in Section 3.2; however, the first-order quantification capability of CL makes it immediately undecidable, even though its expected models (forests of trees) are more restrictive than, for example, hybrid logic. In the spirit of ten Cate and Franceschet (2005), we seek ways of "taming" the full-blown language CL. This task is delicate, since adding even a simple hierarchical structure over the variables of a decidable first-order language is sometimes enough to render it undecidable: for example, while classical first-order logic with a single equivalence relation is decidable, under the same conditions, CL is not. Nevertheless,

it can be shown that satisfiability of CL with unary predicates is complete for non-deterministic exponential time. We compare CL with existing results on various other fragments of tree languages in this respect.

However, it is important to note that all previously mentioned applications assume a *deterministic* decision procedure for the language in question. Therefore, when the optimal complexity class of the problem is in non-deterministic time, this procedure must still be determinized to be used in practice. This is why, for example, SAT solvers run in worst-case deterministic exponential time, although they work on a problem complete for non-deterministic polynomial time. Therefore, from a practical point of view, we should not be content with the NEXPTIME complexity result for CL and should also look for a deterministic procedure. We show in Section 3.3 that given a CL formula φ, there exists CTL formulæ $\omega(\varphi)$ and ψ, with $\omega(\varphi)$ polynomial in the size of φ and ψ exponential on the number of variables in φ such that φ is satisfiable if and only if $\omega(\varphi) \wedge \psi$ is satisfiable. Our construction provides a 2EXPTIME decision procedure which, under widely-held complexity theoretic assumptions, is optimal for *deterministic* time and restates the problem in a pure CTL setting. Furthermore, alternate techniques developed for CTL, such as the classical CTL decision algorithm (Emerson et Clarke, 1982) or symbolic construction of OBDDs (Marrero, 2005), can be leveraged and applied to CL.

3.2 Satisfiability of Configuration Logic

CL is a logic over tree structures whose nodes are name-value pairs. It has been introduced in an earlier paper (Villemaire *et al.*, 2005a) as an appropriate language to express constraints on the configuration of network routers. We start by briefly recalling the syntax and semantics of CL.

3.2.1 Syntax and Semantics

A *configuration* is a forest of *nodes*, each formed of two parts: a *name* and a *value*, represented in the form "name = value". We associate to each node of T a unique symbol that we call a *number*. This symbol does not play a role in the logic properly speaking, but will be used in the decision procedure we present in Section 3.3. We assume that each forest is topped with an additional source node and consider that a configuration is a tree. From now on, the terms "tree" and "configuration" will be used interchangeably.

The succession of name-value pairs from the source to an arbitrary node is called a *named path*; two nodes are considered identical if they have the same named path. Hence, a tree T can be alternately defined as a set of constant named paths closed by prefix. A named path can also have one or more variables in place of the "value" part of some nodes. In this case, depending on the values taken by those variables, the named path will designate different nodes. Variables standing for the "name" part of nodes are not authorized.

In addition to the traditional Boolean connectives, CL allows existential ($\langle\,\rangle$) and universal ($[\,]$) quantification on the "value" part of "name = value" pairs. Existential quantification takes the form $\langle \overline{p}\,;\, n = x\rangle\,\varphi$, where $\overline{p}$ is a (possibly empty) named path, n is a name and x is a variable free in φ. Note that in this quantification, only x is bound; the other variables possibly occurring in $\overline{p}$ are considered free. Therefore, as stated above, the actual admissible values for x depend on how the named path that precedes it is valuated. Universal quantification follows a similar syntax. In line with the interpretation of CL as a tree generalization of first-order logic, the presence of named paths in a quantifier amounts to locating, within the hierarchy, the first-order variable to be quantified; this location can depend on previously quantified variables.

A sentence is a closed CL formula. We suppose in this case that formulæ are *well-*

named —that is, each variable is quantified only once. The formal semantics of CL is shown below, where T is a tree, ρ is a valuation, R_i is a predicate, $\overline{x}$ is a tuple of variables matching the arity of R_i, $\overline{p}$ is a (possibly empty) named path, and V is the set of node values. The notation $\rho(\overline{p})$ is used to denote a named path where all occurrences of variables have been replaced by their value according to ρ.

$$
\begin{aligned}
T, \rho \models R_i(\bar{x}) &\Leftrightarrow R_i(\rho(\bar{x})) \text{ holds} \\
T, \rho \models \varphi \wedge \psi &\Leftrightarrow T, \rho \models \varphi \text{ and } T, \rho \models \psi \\
T, \rho \models \varphi \vee \psi &\Leftrightarrow T, \rho \models \varphi \text{ or } T, \rho \models \psi \\
T, \rho \models \neg\varphi &\Leftrightarrow T, \rho \not\models \varphi \\
T, \rho \models \langle \bar{p}; p = x \rangle \varphi &\Leftrightarrow \exists v \in V : \rho(\bar{p}), p = v \in T \text{ and} \\
&\quad\ T, \rho[x/v] \models \varphi \\
T, \rho \models [\bar{p}; p = x] \varphi &\Leftrightarrow \forall v \in V : \rho(\bar{p}), p = v \in T, \\
&\quad\ T, \rho[x/v] \models \varphi
\end{aligned}
$$

One can observe that CL is a fragment of the XML Query Language XQuery (Boag *et al.*, 2005) without recursion; it is indeed straightforward to map a configuration tree to an XML document, and to use XQuery primitives to represent the alternate syntax defined in the previous semantics.

3.2.2 Necessary Conditions for Decidability

In the following, we study the complexity of the decision problem for CL: given a formula φ with unary predicates, determine whether there exists a tree that satisfies φ. Trakhtenbrot showed that for any first-order language with n-ary relations ($n \geq 2$) other

than equality, satisfiability for finite structures is undecidable (Trakhtenbrot, 1950). CL being a generalization of first-order logic, this theorem also applies to it. However, while in the case of first-order logic, restricting the signature of the language to unary relations and equality is sufficient for ensuring decidability, this condition no longer holds with CL.

Theorem 3.1. *There exists a CL formula $\varphi(x, y)$ such that for every structure $\langle S, R\rangle$, where R is a binary relation over S, there exists a configuration T such that $R(s_1, s_2) \Leftrightarrow T \models \varphi(s_1, s_2)$.*

Proof. Let T be the tree constructed with dummy names p and q as follows. The top-level nodes of T consist of all nodes $p = x$, where x iteratively takes each value in S exactly once. For each top-level node $p = u$ and each $v \in S$ such that $R(u, v)$, there exists one and only one descendant of $p = u$ of the form $p = (u, v)$ (the value of that node is a tuple), and exactly one descendant of the form $q = (v, u)$. Therefore, $R(u, v)$ is equivalent as stating that $p = u$ has a p-descendant and that $p = v$ has a q descendant with the same value. In CL, this can be expressed by the formula $\varphi(u, v) = \langle p = u\,;\, p = x\rangle\langle p = v\,;\, q = y\rangle x = y$. □

Using the fact that variables in CL are organized in a tree structure instead of a flat set, CL with unary relations and equality over infinite domains can simulate first-order logic with arbitrary binary relations and is therefore undecidable. To restore decidability, CL must be restricted to unary predicates. In turn, equality over finite domains of size k can be obtained by defining k unary predicates p_i such that $p_i(d_i)$ is true, and $p_i(d_j)$ is false for $i \neq j$. In the remainder of this paper, the use of equality symbols will be used as a shorthand notation.

3.2.3 Complexity of CL Satisfiability

Once CL is restricted to unary predicates and equality over finite domains, decidability comes as no surprise since there exist a finite number of trees that can possibly satisfy a given formula.

Theorem 3.2. *The satisfiability problem for CL is NEXPTIME-complete.*

Proof. (Lower bound.) It suffices to remark that CL with unary predicates is a generalization of monadic first-order logic, whose satisfiability is NEXPTIME-complete (Börger *et al.*, 1997).

(Upper bound.) Let A and B be a set of names and values, static and given beforehand. Let φ be a CL formula using only names and values in A and B. Define $C = |A||B|$. By the semantics of CL, the arity of trees is bounded by C (since no two children of the same parent can have the same "name=value" pair), and the depth of the trees is bounded by $|\varphi|$ (since recursion on paths is not allowed). The maximum size of a tree is $\sum_{i=1}^{|\varphi|} C^i \leq C^{|\varphi|+1}$. Since C is fixed, then the size of a tree is bounded by $O(2^{|\varphi|})$ It suffices to guess a tree of such a size; model checking of a single tree of size $|T|$ is in $O(|\varphi| \cdot |T|^{|\varphi|}) = O(|\varphi| \cdot 2^{|\varphi||\varphi|})$ and the problem is in NEXPTIME (Hallé *et al.*, 2006b). □

Numerous works have considered the satisfiability of related fragments of tree languages; each of them differs from the others in the set of features they support: first-order quantification on node values ($\exists$), "next-sibling" relation ($\rightarrow$), number and type of Boolean connectives ($\vee$, $\neg$, $\wedge$), "child" relation ($\downarrow$), recursion or transitive closure ($*$), equality between node values ($=$). For example, CL is denoted by $\{\neg, \wedge, \vee, \downarrow, =, \exists\}$. Using such a notation, we can attempt a rough comparison; however the reader is referred to the original papers for precisions on each language.

Fragment	Satisfiability	References
$\{\neg, \vee, \downarrow\}$	Decidable	(Calcagno *et al.*, 2003)
$\{\neg, \vee, \downarrow, \rightarrow, =\}$	Decidable	(Marx, 2005a)
$\{\wedge, \downarrow, \rightarrow\}$	NP-complete	(Hidders, 2003)
$\{\vee, \downarrow, \rightarrow\}$	NP-complete	(Lakshmanan *et al.*, 2004)
$\{\neg, \wedge, \downarrow, \rightarrow\}$	NP-hard	(Hidders, 2003)
$\{\neg, \vee, \downarrow, \rightarrow, =, *\}$	EXPTIME-complete †	(Marx, 2004)
$\{\neg, \vee, \downarrow, \rightarrow, *\}$	EXPTIME-complete †	(Afanasiev *et al.*, 2005)
$\{\neg, \vee, \downarrow, =, \exists\}$	NEXPTIME-complete †	This paper
mCTL*	2EXPTIME-complete †	(Kupferman et Vardi, 2006)
$\{\neg, \vee, \downarrow, \exists\}$	Undecidable	(Conforti et Ghelli, 2004; Charatonik et Talbot, 2001)

Table 3.1: Satisfiability results for various related logics. The † symbol indicates that the result concerns finite structures or finite domains.

The complexity of the satisfiability problem of a number of these fragments has been studied by analyzing them as fragments of first-order logic (Marx, 2005b), and in particular two-variable first-order logic with words on data (Marx, 2005a; Bojanczyk *et al.*, 2006). Also worthy of mention is mCTL* (Kupferman et Vardi, 2006), an extension of CTL* that allows "memoryful" path quantification similar in some respects to the construction presented in the next section. Koch (2006) has further shown that Core XQuery is NEXPTIME-hard. Complexity results are summarized in Table 3.1.

3.2.4 Hybrid interpretation of CL

We end this section by showing how CL with unary predicates corresponds to a fragment of the hybrid logic $HL(@, \downarrow)$ (Blackburn, 2000). Hybrid logic is an extension of classical modal logic with operators for uniquely naming states ($\downarrow$) and for stating that a formula is true in some specific state ($@$). This naming apparatus can be used to simulate the named path quantification mechanism of CL. In fact, one can provide a linear embedding τ from the set of CL formulæ into the set of hybrid logic formulæ as follows. The syntactic translation presented is in the spirit of the translation of CTL* into the similarly-defined Temporal Logic with Reference Pointers (Goranko, 2000).

$$
\begin{aligned}
\tau([p_1 = x_1, \ldots, p_n = x_n \,;\, p = x]\ \varphi) &= @_{\hat{x_n}} \Box_1 (\hat{\alpha} = p \wedge \downarrow \hat{x}.\tau(\varphi)) \\
\tau([\,;\, p = x]\ \varphi) &= @_{root} \Box_1 (\hat{\alpha} = p \wedge \downarrow \hat{x}.\tau(\varphi)) \\
\tau(\langle p_1 = x_1, \ldots, p_n = x_n \,;\, p = x \rangle\ \varphi) &= @_{\hat{x_n}} \Box_2 (\hat{\alpha} = p \rightarrow \downarrow \hat{x}.\tau(\varphi)) \\
\tau(\langle\,;\, p = x \rangle\ \varphi) &= @_{root} \Box_2 (\hat{\alpha} = p \rightarrow \downarrow \hat{x}.\tau(\varphi)) \\
\tau(t(x_1, \ldots, x_k)) &= @_{x_1} \hat{t}(\hat{x_2}, \ldots, \hat{x_k})
\end{aligned}
$$

Table 3.2: Translation of CL into the fragment $HL(@, \downarrow)$.

We designate the root of the tree by the nominal "root". The "name" part of the "name=value" pair of each node in the CL formula is replaced by a discrete variable $\hat{\alpha}$ that contains the name of the node represented by the current state. Each variable x in a CL formula becomes a state variable $\hat{x}$ in HL; n-ary predicates $t(x_1, \ldots, x_k)$ in the CL formula are mapped into $(n-1)$-ary predicates $\hat{t}$ such that $t(x_1, \ldots, x_k)$ holds if and only if $@_{x_1} \hat{t}(\hat{x_2}, \ldots, \hat{x_k})$. This way, unary predicates in CL become Boolean state variables in HL. Finally, we define the two modalities $\Box_1$ and $\Box_2$ as follows: $M, s' \models \varphi$, and $M, s \models \Box_1 \varphi$ if and only if for each s' such that $R(s, s')$, $M, s' \models \varphi$, and $M, s \models \Box_2 \varphi$ if and only if there exists s' such that $R(s, s')$. The recursive translation τ from CL to HL becomes direct: Boolean connectives translate into themselves in the standard way, and CL quantifiers are translated as shown in Table 3.2. Clearly, φ has a model if and only if $\tau(\varphi)$ has one.

However, it has long been known that full hybrid logic is undecidable (Areces *et al.*, 2001). Even by restricting CL to unary relations, the hybrid logic correspondence language, which becomes nothing but the classical $HL(@, \downarrow)$ with only Boolean state variables and no relations, is still undecidable. The source of undecidability has been identified (ten Cate et Franceschet, 2005): the presence of a $\Box \downarrow \Box$ pattern is shown to be a necessary condition for undecidable hybrid formulæ. The translation of the CL existential quantifier produces exactly such a sequence of alternating $\Box$ and $\downarrow$. Therefore, although the pattern $\Box \downarrow \Box$ is a necessary condition for undecidability, this example

shows it is not a sufficient one.

3.3 CL Satisfiability as a Temporal Logic Problem

The decision procedure presented in Theorem 3.2 is non-deterministic. A straightforward determinization consists of generating all possible finite trees in a sequence until one that satisfies the desired formula is found. Termination is guaranteed by the fact that the number of candidates to search is finite, due to the semantics of CL with unary predicates.

In this section, CL satisfiability is translated into a CTL decision problem. Since CTL is decidable in simple deterministic exponential time (Emerson et Clarke, 1982), it provides an alternate, deterministic CL decision procedure. The advantage of using CTL decision procedures is twofold: it leverages algorithms for a well-studied logic, and the decision algorithms (especially (Marrero, 2005)) create a model by decomposing the original formula, therefore making better "educated guesses" on the candidate model than a brute-force enumeration.

3.3.1 Reducing CL to CTL

The first part of the translation consists in converting a configuration T and a CL formula φ with n quantified variables to a Kripke structure $K_{T,\varphi}$, and to translate φ into a CTL formula $\omega(\varphi)$ such that that $T \models_{\text{CL}} \varphi$ if and only if $K_{T,\varphi} \models_{\text{CTL}} \omega(\varphi)$. The construction we use is based on the reduction of CL model checking to CTL model checking presented in (Hallé *et al.*, 2006b).

First, we take a configuration T and a CL formula φ with n quantified variables and generate a Kripke structure $K_{T,\varphi} = (S, I, R, L)$, where S is a set of states, $I \subseteq S$ is a

set of initial states, $R \subseteq S^2$ is a transition relation and L is a labelling function which shall be defined later.

Let A, B and Γ be respectively the sets of all names, values and numbers appearing in T. From these sets, we create the sets A', B' and Γ' by adding to each a special, unused symbol $\#$ that will stand for "undefined". A state of $K_{T,\varphi}$ is uniquely identified by the values of $n+3$ state variables: $\alpha \in A'$ contains the name of the node in the original tree, $\beta \in B'$ contains the value of the node in the original tree, $\gamma \in \Gamma'$ contains the number of the node in the original tree, and $x_1, \ldots, x_k \in B'$ stand respectively for each quantified variable in the original CL formula φ; they can hold either a node value, or the "undefined" symbol $\#$. For each state variable x, we define a function $L_x : S \rightarrow Dom(x)$ that assigns a unique value to every variable in every state.

A state of $K_{T,\varphi}$ corresponds to a node of the original tree and a valuation of the quantified variables $x_1, \ldots, x_n$ of φ. Formally, let $a \in A'$, $b \in B'$, $c \in \Gamma'$ and $(b_1, \ldots, b_n) \in B'^n$. Let s be a state such that $L_\alpha(s) = a$, $L_\beta(s) = b$, $L_\gamma(s) = c$ and for all $1 \leq i \leq n$, $L_{x_i} = b_i$. Then s is a state in S if and only if there exists a node in T with name a, value b and number c. Hence, for each node in T, there are multiple states in $K_{T,\varphi}$ associated to it, namely one for each possible valuation of the $x_1, \ldots, x_k$; we call these the *copies* of T.

Definition 3.1 (Copy). *Let $(b_1, \ldots, b_n) \in B'^n$; the subset $S_{(b_1,\ldots,b_n)} \subseteq S$ defined by*

$$\{s : L_{x_1}(s) = b_1 \wedge \cdots \wedge L_{x_n}(s) = b_n\}$$

is called a copy.

We extend the term and call a copy the sub-Kripke structure obtained by taking the restriction of R to $S_{(d_1,\ldots,d_n)}$. There are as many copies in $K_{T,\varphi}$ as there are possible ways of valuating the x_i.

As the name implies, each copy is intended to be a mirror image of the tree T from which K is built. Since each node in T possesses a unique number, all states in $K_{T,\varphi}$ where γ takes the same value c are all images of the same node in T. Therefore, we can take γ as a "representative" that maps each state of $K_{T,\varphi}$ back to its unique corresponding node in T. It follows that γ unequivocally determines the values of α and β for a given state.

Definition 3.2 (Representative). *Variable γ is a* representative *of α and β: for every $d \in \Gamma'$, there exist values $a \in N'$ and $b \in V'$ such that for every state $s \in S$, $L_\gamma(s) = d$ implies that $L_\alpha(s) = a$ and $L_\beta(s) = b$.*

The transition relation R is composed of two kinds of transitions. *Tree* transitions link the states belonging to the same copy of T.

Definition 3.3 (Tree transition). *A tuple $(s_1, s_2) \in S^2$ is a* tree transition *if and only if s_1 and s_2 belong to the same copy.*

We ensure that, inside each copy, states are connected according to their original structure in T. For each tree transition $t = (s_1, s_2)$ such that $L_\gamma(s_1) = c_1$ and $L_\gamma(s_2) = c_2$ and for each $1 \leq i \leq n$, $L_{x_1}(s_1) = \#$, $t \in R$ if and only if node numbered c_2 is the child of node numbered c_1 in T. Thus, K_T is composed of disjoint, identical copies of T. Then, *freeze* transitions connect copies that differ only in the value of one of the x_i.

Definition 3.4 (Freeze transition). *A transition $(s_1, s_2) \in R$ is a* freeze transition *if and only if there exists $1 \leq j \leq n$ such that $L_{x_j}(s_1) \neq L_{x_j}(s_2)$ and for every $1 \leq i \leq n$ $(i \neq j)$, $L_{x_i}(s_1) = L_{x_i}(s_2)$.*

Moreover, we impose that all freeze transitions return to a copy of the source node of T; these transitions are called *resetting*.

Definition 3.5 (Resetting transition). *A freeze transition $(s_1, s_2) \in R$ is* resetting *if and only if $L_\gamma(s_2) = \#$.*

Freeze transitions restrict the behaviour of variables $x_1, \ldots, x_n$; such variables are aptly called *freeze* variables. A freeze variable acts as some kind of permanent "memory device": it is originally set to some undefined value, and once it takes a definite value, it keeps ("freezes") it for the remainder of the execution of the system.

Definition 3.6 (Freeze variable). *A variable x_i in $K_{T,\varphi}$ is a* freeze variable *if and only if whenever $L_s(x_i) = \#$, the following holds: $\forall s \in I\ \forall (s_1, s_2) \in R : L_{s_1}(x_i) = \# \vee L_{s_1}(x_i) = L_{s_2}(x_i)$.*

Moreover, we want to arrange the x_i so that the values they remember are those of the state variable β.

Definition 3.7 (β-memory). *A freeze variable x_i is a* memory *of state variable β if and only if for every transition $(s_1, s_2) \in R$, either $L_{s_1}(x_i) = L_{s_2}(x_i)$ or $L_{s_2}(x_i) = L_{s_1}(\beta)$.*

Finally, since each copy in K is meant to be an image of the original tree T, we must ensure that all such images are identical with the concept of *symmetry*.

Definition 3.8 (Symmetry). *K is symmetrical if any tree transition $(s_1, s_2) \in R$ such that $L_{s_1}(\gamma) = c_1$ and $L_{s_2}(\gamma) = c_2$ either exists in all copies or in no copy of K.*

This translation procedure can be seen as the construction of a finite transition system modelling an algorithm that checks a CL formula on a particular tree. The algorithm traverses the tree multiple times; it stores in variables α and β the "name=value" pair of the current tree node, and the x_i are used to store the value of some node. Once a new value is put into one of the x_i, the algorithm goes back to the head of the tree.

No constraint is imposed yet on the actual paths that the algorithm takes on each traversal: these constraints are enforced by the translation of the CL formula into CTL. Once the Kripke structure $K_{T,\varphi}$ is obtained, the translation function ω of the CL formula

$$
\begin{array}{rcl}
\omega(\langle n = x_i\,,\,\overline{p}\,;\,m = x_i\rangle\varphi) & = & \mathbf{EX}\,(\alpha = n \wedge \beta = x_i \wedge \omega(\langle \overline{p}\,;\,m = x_i\rangle\varphi)) \\
\omega([n = x_i\,,\,\overline{p}\,;\,m = x_i]\,\varphi) & = & \mathbf{EX}\,(\alpha = n \wedge \beta = x_i \wedge \omega([\overline{p}\,;\,m = x_i]\,\varphi)) \\
\omega(\langle\,;\,n = x_i\rangle\varphi) & = & \mathbf{EX}\,((\alpha = n \wedge x_i = \#) \wedge \mathbf{EX}\,(x_i \neq \# \wedge \omega(\varphi))) \\
\omega([\,;\,n = x_i]\,\varphi) & = & \mathbf{AX}\,((\alpha = n \wedge x_i = \#) \rightarrow \mathbf{EX}\,(x_i \neq \# \wedge \omega(\varphi)))
\end{array}
$$

Table 3.3: Embedding of CL quantifiers into CTL.

φ into CTL becomes simple. It is defined recursively on the structure of the formula; ground equality testing and Boolean connectives are translated in the traditional way; the quantifiers are translated as shown in Table 3.3.

Theorem 3.3 (Hallé *et al.*, 2006). *Let T be a tree, φ be a CL formula and ω be the embedding defined previously. Let $K_{T,\varphi}$ be the Kripke structure built as described above. Then $T \models_{CL} \varphi$ if and only if $K_{T,\varphi} \models_{CTL} \omega(\varphi)$.*

3.3.2 Bad Models of $\omega(\varphi)$

The mapping produced by ω produces a CTL formula linear in the size of the original CL formula. By the EXPTIME-completeness of CTL satisfiability and Theorem 3.2, providing a model of $\omega(\varphi)$ is not sufficient to show there exists a tree T that satisfies φ unless EXPTIME = NEXPTIME.

To illustrate our approach, let us consider the example of an unsatisfiable CL formula φ for which $\omega(\varphi)$ is a satisfiable CTL formula. Consider the CL formula $\langle\,;\,\mathrm{a} = x_1\rangle\,[\,;\,\mathrm{a} = x_2]\,x_1 \neq x_2$. No tree can satisfy this formula. The reason is that any value taken by x_1 eventually reappears as x_2, since both x_1 and x_2 quantify over the same set of nodes. However, $\omega(\varphi)$ is the following:

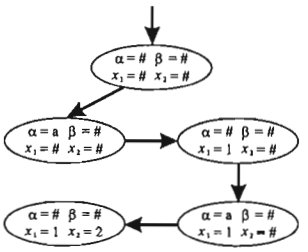


Figure 3.1: A Kripke structure satisfying formula 3.1. Values of state variable γ are not shown.

$$\begin{aligned}
&\mathbf{EX}\,((\alpha = \mathrm{a} \wedge x_1 = \#) \wedge \\
&\quad \mathbf{EX}\,(x_1 \neq \# \wedge \\
&\quad\quad \mathbf{AX}\,((\alpha = \mathrm{a} \wedge x_2 = \#) \rightarrow \\
&\quad\quad\quad \mathbf{EX}\,(x_2 \neq \# \wedge x_1 \neq x_2))))
\end{aligned} \tag{3.1}$$

This CTL formula is satisfiable, and Figure 3.1 shows a small Kripke structure that models it. We give below a property ψ such that $\omega(\varphi) \wedge \psi$ is satisfiable in CTL if and only if φ is satisfiable in CL, thus giving a sufficient condition for a Kripke structure to be $K_{T,\varphi}$ for some tree T. Moreover, we shall see that ψ only depends on the number of variables in φ.

Theorem 3.4. *Let φ be a CL formula with $n \geq 1$ variables, and $K = (S, I, R, L)$ be a Kripke structure with state variables α, β, γ and $x_1, \ldots, x_n$ that fulfils the following conditions:*

- *In the initial state, all state variables take value #*
- *The graph $\langle S, R \rangle$ is a tree*
- *γ is a representative of α and β*
- *The x_i are freeze variables that memorize β*
- *All freeze transitions are resetting*
- *K is symmetrical*

Then K is a tree encoding*: there exists a configuration tree T such that K is $K_{T,\varphi}$.*

Proof. We build T from the copy $K_{\#,\ldots,\#}$, i.e. the subset of states where all freeze variables are undefined, by creating a one-to-one mapping M_1 defined as follows: for s a state in $K_{\#,\ldots,\#}$, $M_1(s)$ corresponds to a node of T with name $L_s(\alpha)$, value $L_s(\beta)$ and number $L_s(\gamma)$. The structure of T is derived from that of $K_{\#,\ldots,\#}$ as follows: for s_1, s_2 two states of $K_{\#,\ldots,\#}$, node $M_1(s_2)$ is a child of node $M_1(s_1)$ in T if and only if (s_1, s_2) is a transition in $K_{\#,\ldots,\#}$.

Consider now $K_{T,\varphi} = (S', I', R', L')$, the Kripke structure constructed from T as described in the previous section, and take $K'_{\#,\ldots,\#}$ the copy in $K_{T,\varphi}$ where all freeze variables are undefined. By definition, there exists a second one-to-one mapping M_2 from nodes of T to states of $K_{T,\varphi}$ that translates name, value and number for every node into variables α, β and γ, and preserves the parent-child relationship of T in so doing. But then the composition $M_1 \circ M_2$ is the identity and therefore $K_{\#,\ldots,\#} = K'_{\#,\ldots,\#}$. But then, since both K and $K_{T,\varphi}$ are symmetrical and have the same freeze transitions, they must be identical overall. □

3.3.3 Tree Encodings

In order to infer CL's satisfiability from CTL's, we must arrange for the CTL decision procedure to return only tree encodings. A possible solution would involve the modification of the CTL decision procedure itself. However, the following lemmas show that each of the conditions in Theorem 3.4 can actually be expressed as CTL formulæ, so that the property of being a tree encoding is by itself a CTL formula ψ. Therefore, searching for a model of $\omega(\varphi)$ that is a tree encoding simply amounts to finding an arbitrary model of $\omega(\varphi) \wedge \psi$. Unsurprisingly, the auxiliary formula ψ is exponential in the size of the original CL expression; more precisely, we shall see that it is a conjunction

composed of an exponential number of formulæ of length polynomial in the number of variables of φ.

Lemma 3.1. *Let K be the Kripke structure defined as above. If K satisfies the following CTL formula, then γ is a representative in K.*

$$\bigwedge_{d \in \Gamma'} \bigvee_{a \in A', b \in B'} \mathbf{AG}\,(\gamma = d \rightarrow (\alpha = a \wedge \beta = b)) \tag{3.2}$$

Lemma 3.2. *Let K be a Kripke structure defined as above and x_i be a state variable with domain B'. If the following CTL formula holds, x_i is a freeze variable in K.*

$$x_i = \# \wedge \bigwedge_{k \in B} \mathbf{AG}\,(x_i = \# \vee (x_i = k \wedge \mathbf{AX}\, x_i = k)) \tag{3.3}$$

Taken as is, the x_i can freeze any arbitrary value. We further restrict their behaviour by imposing that they remember only values of state variable β.

A CTL formula can impose that restriction on K.

Lemma 3.3. *Let K be a Kripke structure defined as above and x_i be a freeze variable with domain B'. If the following CTL formula holds, x_i is a memory of state variable β in K.*

$$\bigwedge_{k \in B} \mathbf{AG}\,((x_i = \# \wedge \beta = k) \rightarrow \mathbf{AX}\,(x_i \neq \# \rightarrow x_i = k)) \tag{3.4}$$

As for the existence of a representative, symmetry can be imposed by a CTL formula. Special care must be taken, since "terminal" copies (where all freeze variables are initialized) do not have successors.

Lemma 3.4. *Let K be a Kripke structure defined as above, with γ a representative in*

K. If the following CTL formula holds, then K is symmetrical.

$$\bigwedge_{d_1,d_2\in\Gamma'} ((\bigvee_{i=1}^{n} x_i \neq \#) \rightarrow ((\mathbf{AG}\,(\gamma = d_1 \rightarrow \mathbf{EX}\,\gamma = d_2)) \vee (\mathbf{AG}\,(\gamma = d_1 \rightarrow \neg(\mathbf{EX}\,\gamma = d_2))))) \quad (3.5)$$

Lemma 3.5. *Let K be a Kripke structure defined as above, γ be a representative. If the following CTL formula holds, all freeze transitions of K are resetting.*

$$\bigwedge_{i=1}^{n} \mathbf{AG}\,((x_i = \# \rightarrow \mathbf{AX}\,(x_i \neq \# \rightarrow \gamma = \#)) \quad (3.6)$$

Since the Kripke structure constructed by the CTL decision procedure can be arbitrary, it is not excluded that such structure contains cycles. We must therefore impose two additional conditions restricting the structure to trees.

Lemma 3.6. *Let K be a Kripke structure. If the following CTL formula holds, K is acyclic.*

$$\bigwedge_{d\in\Gamma'} \bigwedge_{k_1\in B} \cdots \bigwedge_{k_n\in B} \mathbf{AG}\,((\gamma = d \wedge x_1 = k_1 \wedge \cdots \wedge x_n = k_n) \rightarrow \neg\mathbf{EF}\,(\gamma = d \wedge x_1 = k_1 \wedge \cdots \wedge x_n = k_n)) \quad (3.7)$$

Lemma 3.7. *Let K be an acyclic Kripke structure. If the following CTL formula holds, K is a tree.*

$$\bigwedge_{d_1,d_2\in\Gamma'} \mathbf{EF}\,(\gamma = d_1 \wedge \mathbf{EX}\,\gamma = d_2) \rightarrow (\mathbf{AG}\,(\gamma \neq d_1 \rightarrow \neg\mathbf{EX}\,\gamma = d_2)) \quad (3.8)$$

Theorem 3.5. *Let φ be a CL formula with k quantified variables. Let $\varphi' = \omega(\varphi)$ be its CTL translation as described in Section 3.2. Let ψ be the conjunction of Lemmas 3.2–3.8. φ is satisfiable in CL if and only if $\varphi' \wedge \psi$ is satisfiable in CTL.*

Proof. If φ is satisfiable, then there exists some tree T such that $T \models \varphi$; by Theorem 3.3, $K_{T,\varphi} \models \varphi' \wedge \psi$. Conversely, if $\varphi' \wedge \psi$ if satisfiable, then the CTL decision procedure returns a model of it, K. By Lemmas 3.1-3.7, K satisfies the hypotheses of Theorem 3.4; therefore, there exists a tree T such that K is $K_{T,\varphi}$, and this tree is a model of φ. □

By this theorem, a model for a CL formula can be constructed directly using decision procedures for CTL. It should be noted that in the expression of ψ, the conjunction of Lemmas 3.2–3.8 is exponential in n (exponentiality comes from the conjunction over Γ', whose maximum size is exponential in the length of the original formula. But since $n \leq |\varphi|$, then $|\varphi_{\mathrm{CTL}} \wedge \psi| \in O(2^{|\varphi|})$. In turn, satisfiability for CTL is EXPTIME-complete; hence the resulting decision procedure for CL is in deterministic double exponential time. Unless EXPTIME = NEXPTIME, this result is optimal for a *deterministic* CL model construction procedure.

3.4 Conclusion

In this paper, we have shown how to reduce the satisfiability of CL with unary predicates to the satisfiability of CTL. This leads to a 2EXPTIME upper bound on the *deterministic* decision procedure. As far as satisfiability is concerned, the result obtained is as sharp as possible for finite models, since the extension of CL to general n-ary predicates leads to undecidability, even with a single equivalence relation. Moreover, the reduction of CL satisfiability to CTL satisfiability, although it requires an exponential translation of the original formula, still yields a decision procedure which, under widely-held complexity theoretic assumptions, is optimal for deterministic time.

A possible way of pushing the envelope of our CTL encoding would be to consider *regular trees* that represent unfoldings of directed graphs with cycles. Finally, the universal fragment of CL, that translates into LTL formulæ, could also be studied in itself, since the satisfiability of this fragment could be reduced to a lower complexity class.

CHAPITRE IV

MODELLING THE TEMPORAL ASPECTS OF NETWORK CONFIGURATIONS

Ce chapitre est tiré de l'article suivant :

Hallé, S., Deca, R., Cherkaoui, O., Villemaire, R., Puche , D. (2007). Modelling the Temporal Aspects of Network Configuration. In *Network Control and Engineering for QoS, Security and Mobility (NetCon 2005)*, Springer Verlag : IFIP International Federation for Information Processing, 229, 269-282.

Jusqu'à présent, les contraintes présentées dans les chapitres précédents étaient statiques : il s'agissait de propriétés qui se réfèrent à un état (ou un message) particulier d'un système. Les chapitres qui suivent étendent le problème aux contraintes dynamiques, qui font référence à une séquence d'états (ou de messages) du système. Le présent chapitre formule un premier énoncé de cette question en analysant un ensemble de contraintes tirées de la configuration d'un service réseau.

C'est ici qu'entre pleinement en jeu la logique temporelle déjà présentée aux chapitres 1 et 3. L'utilisation des logiques temporelles LTL et CTL pour exprimer des dépendances entre plusieurs états d'une procédure, d'un système ou d'un protocole est commune aux chapitres 4 à 6.

Abstract

One of the main issues with the existing management configuration is the absence of a transactional model, which should allow the network configuration data to retain their integrity and consistency during the configuration process. In this paper, we propose a mathematical framework based on lattice theory allowing the structuring of configuration operations leading to the concept of component and validation checkpoint, and present polynomial-time algorithms for studying these structures. We will illustrate the model by two examples of configuration operations: the deployment of a VLAN service through SNMP and the deployment of a VPN service through the Netconf protocol.

4.1 Introduction

Among other network management functions, configuration management is still mainly accomplished by proprietary means, be it Command Line Interface (CLI), JunOS or TL1. Recent alternatives like SNMPConf, COPS and Netconf have not yet succeeded in bringing a standard configuration solution. This situation is due to numerous causes: security (SNMPv3), absence of an adequate configuration information model, proprietary equipment instrumentation semantics. Another overlooked factor is the absence of a simple transactional model between agents and managers allowing for the association between management protocol operations (SNMP, COPS and Netconf) and management information.

When configuring a network service that involves multiple equipments, there is an important temporal aspect of complexity of the network service configuration. There are many sequences of configuration commands or operations that must be performed on multiple network elements or equipments, temporal dependencies among these sequences, semantic constraints among their parameters. Moreover, specific groups of

commands and parameters belong to the same service or sub-service and must thus be performed together, in an atomic way, even though they affect multiple components or network elements form different network devices. The atomic character of the configuration operations involving a number of parameters is relevant both at device and at network levels.

In this paper, we propose a mathematical framework based on lattice theory allowing the structuring of configuration operations in terms of configuration dependencies. The concepts of component and milestone that we define in terms of paths in the lattice structure help us to simplify the analysis of possible solution paths and provide us with a sound criterion for dividing the deployment of a service into natural macro-steps that serve as validation checkpoints. We will illustrate the model by two examples of configuration operations: the deployment of a VLAN service through SNMP and the deployment of a VPN service through the Netconf protocol, and show preliminary results for validation checkpoints for the latter of these cases.

In section 2, we show by two examples why current management approaches are inadequate for dealing with the sequential aspect of network configuration. In section 3, the lattice-based mathematical framework for modelling temporal constraints is detailed, and polynomial-time algorithms for studying the resulting structures are presented. Section 4 shows possible applications of this framework and preliminary results obtained for the case of simple Virtual Private Networks, while section 5 concludes and indicates further directions of work.

4.2 Motivation and Related Work

The deployment of a service over a network basically consists in altering the configuration of one or many equipments to implement the desired functionalities. We can

presuppose without loss of generality that all properties of a given configuration are described by attribute-value pairs hierarchically organized in a tree structure (Hallé *et al.*, 2004).

Possible alterations to the configuration typically include deleting or adding new parameters to the configuration of a device, or changing the value of existing parameters. In most cases, the parameters involved in such modifications are both syntactically and semantically interdependent. For instance, the value of some parameter might be required to depend in a precise way on the value of another parameter; the simplest example of such dependency is the fact that an IP address must match the subnet mask that comes with it. More complex dependencies might constrain the existence of a parameter on the existence of another. Recent works have shown how such dependencies can be automatically checked by logical tools on a given configuration snapshot (Hallé *et al.*, 2004).

However, the situation becomes more complex when one wants to actually *deploy* a service from scratch. In addition to constraints on the values of parameters, the dependencies may also impose that the modifications be performed in a specific order. When done in an uncoordinated way, changing, adding or removing components or data that implement network services can bring the network in an inconsistent or undefined state. This fact becomes acutely true in the cases where operations must be distributed on multiple network elements, as they cannot be modified all at once. Moreover, while a single inconsistent device can ultimately be restarted when all else fails, there is no such "restart" option when an entire network configuration becomes inconsistent.

However, one of the main issues with the existing management paradigms is the absence of a *transactional model*, which should allow the network configuration data to retain their integrity and consistency during the configuration process.

The network community has proposed different approaches for ensuring the consis-

tency and integrity of the network configuration during the management of network services. The policy-based management using the Ponder language (Damianou *et al.*, 2001) incorporates OCL (Object Constraint Language) (Warmer et Kleppe, 2003) to express the dependencies among configuration parameters. Other approaches, based on ontologies (de Vergara *et al.*, 2002; Noy *et al.*, 2001), use the Protégé Axiom Language and Toolset (PAL) (Crubézy, 2002) for expressing configuration parameter constraints and queries. Many other constraint languages and tools are also available and can be used to express configuration parameter dependences, such as the Alloy language (Jackson, 2000) and the constraint analyzer based on it, Alcoa (Jackson *et al.*, 2000). However, these approaches use constraint languages borrowed from other domains, designed for other purposes. Therefore, they are not adequate to the specifics of network configuration, and they do not tackle the transactional aspect of network configuration.

We will study two examples of configuration management using different paradigms, and stress their weaknesses in this regard. Based on this evidence, we will show what the transactional model can accomplish and what its benefits are in the area of network configuration.

4.2.1 VLAN Configuration with SNMP

The deployment of a Virtual Local Area Network (VLAN) (Cisco Systems, 2005; IEEE, 2003) on a network involves a number of configuration operations falling into four categories:

- specification of the Virtual Trunk Protocol (VTP) domain and operation mode
- VLAN creation
- port allocation
- trunk creation

However, these operations cannot be performed in any order. Some ordering constraints are imposed.

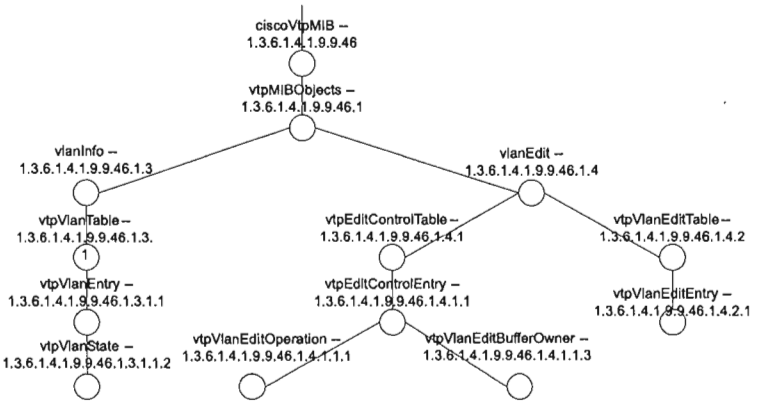
Clearly, the operations belonging to the first group (VTP) are preparation operations on which the operations of the second group (VLAN creation) rely. In the same fashion, the port allocation cannot be done before the VLAN has been created. This leads to the formulation of a first set of two temporal constraints:

Temporal Constraint 4.1. *All VTP operations must have been performed before any VLAN creation parameter is added to the configuration.*

Temporal Constraint 4.2. *All VLAN creation parameters must have been added to the configuration before any port allocation or trunk creation parameters are added.*

These two constraints entail that the VLAN be created in an *atomic* way: the name, number and other parameters must be specified together and the editing must be done by one manager at a time. If the configuration were to be modified by means of the command line interface, this atomic property would be achieved by having both number and name parameters mandatory within the same command (for example, the Cisco IOS command `vlan <number> <name>`). A similar reasoning can be done for the other modification operations, leading to more temporal constraints.

However, despite these temporal constraints, the SNMP paradigm (Case *et al.*, 1990) allows the parameters to be configured independently and has no semantics for the configuration operations. It does not have a transactional model and thus allows inconsistent evolution of the network configuration. The way SNMP ensures atomicity is by providing an editing buffer for VLAN creation (the `vtpVlanEditingTable`) within the VLAN Management Information Base (MIB) (Cisco Systems, 2005), as Figure 4.1 shows. Only one manager at a time is allowed to own and edit this buffer.



Contents of a *vtpVlanEditEntry*									
#	1	2	3	4	5	6	...	11	...
Name			vtpVlanEditType	vtpVlanEditName		vtpVlanEditDot10Said		vtpVlanEditRowStatus	
Value			ethernet (1)	vlan11		000186AB		createAndGo (4)	

Figure 4.1: Structure of the VLAN edit table, edit control table and VLAN state object

This example illustrates several facts. First, temporal constraints impose that some of the VLAN parameters be grouped; SNMP does not elegantly enforce this and rather uses an ad-hoc editing buffer mechanism for this purpose.

Second, there are two levels of validation: the first level ensures that each operation has been correctly made by confirming that the `apply` buffer operation has succeeded; the second level validates the overall operation and checks whether the VLAN has actually been created.

4.2.2 Example 2: VPN Configuration with Netconf

In this section we analyze the problems encountered by the Netconf protocol when dealing with network services that involve multiple equipments and introduce a transactional model that solves these problems.

We illustrate this with an example of an MPLS Virtual Private Network (VPN) ser-

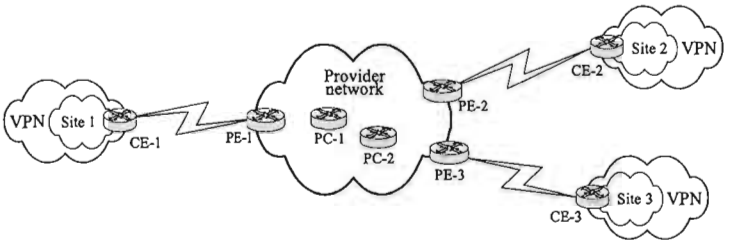


Figure 4.2: Example of a network topology with an MPLS VPN service

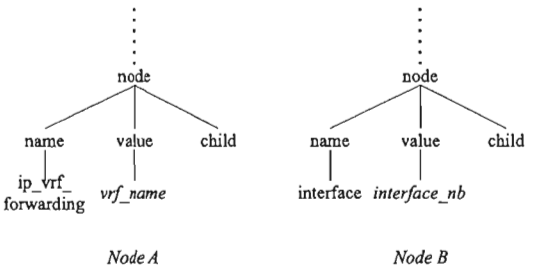


Figure 4.3: Two configuration nodes that must be added for deploying a VPN.

vice deployment (Rosen et Rekhter, 1999). A VPN is a private network constructed within a public network such as a service provider's network. A customer might have several sites, which are contiguous parts of the network, dispersed throughout the Internet and would like to link them together by a protected communication. The VPN ensures the connectivity and privacy of the customer's communications between sites. Figure 4.2 presents a sample MPLS VPN service.

Such a service consists of multiple configuration operations that involve setting the routing tables and the VPN forwarding tables, setting the MPLS, BGP and IGP connectivity on multiple equipments having various roles, such as the customer edge (CE), provider edge (PE) and provider core (PC) routers. In total, a minimum of about 30 parameters must be added or changed in each device involved in the deployment of the VPN. As an example, Figure 4.3 shows two leaf nodes that must be added, each in its own position, to the configuration tree of a PE router.

Leaf node A is one of the configuration parameters that contributes to the creation of the VPN routing and forwarding tables on the PEs of the service provider. It cannot be added to the configuration of a PE router before the corresponding interface has been configured, which entails, among other things, the addition of leaf node B. Therefore, one can extract a temporal constraint from this relation:

Temporal Constraint 4.3. *The node* `ip_vrf_forwarding` *cannot be added to a configuration tree before node* `interface/number` *has been set.*

Similar dependencies can be extracted for many other pairs of nodes among the 30 parameters involved, based on

- the semantic dependencies among the various components and parameters of the configuration;
- the spatial distribution of the configuration components and parameters;
- the choices of topology and technologies (protocols, device roles and types, vendor software, etc.).

These interdependencies imply a logically simultaneous configuration of the respective parameters on all these equipments. Since these equipments are spatially distributed and configuration operations can only be performed sequentially, this goal can only be achieved by "synchronizing" the configurations on different equipments by carefully setting up *validation points* during the configuration procedure.

The Netconf protocol (Enns, 2005) defines a simple mechanism for network device management. However, its transactional model, which includes a validation capability, is device-centred, and does not provide a mechanism to ensure the consistency of the configuration that involves correlated configuration steps on multiple devices.

Netconf provides two phases of successful configuration transaction during a service configuration procedure: preparation and commitment. During preparation, the configurations are retrieved from the network devices. When all the configurations have been

retrieved, the edition starts at service level. The validation at this stage ensures that the network configuration is consistent before the proposed modifications required by the service. To ensure the integrity of the configuration edition, the device configurations are locked, edited and subsequently unlocked. When the service edition has been successfully accomplished, the commitment starts. The validation at this stage ensures that the network configuration remains consistent after the successive modifications of the network configurations. Therefore, taken as is, Netconf does not provide any indication as to where and what to validate.

The previous examples have shown that many configuration operations must be done in a specific sequence, others must be performed together notwithstanding the order and others are mutually exclusive. Therefore, we need a clear temporal representation of the operations to be performed, which will describe all the temporal dependencies, indicate the possible procedural order of operation for various groups of configuration parameters on various devices and indicate the optimal temporal order and distribution of these operations.

4.3 A Theoretical Model

In this section, we present a theoretical study of the temporal issues described in section 4.2 by providing a theoretical model of the situation.

4.3.1 States and Transitions

Let S be a set of "states" representing a unit situation at a given time. In the case of network configuration, states are labelled trees as described in Section 4.2.

We call *transition* from a state s_1 to a state s_2 the structural modifications that

transform s_1 into s_2. Formally, transitions can be defined as a subset of tuples $T \subseteq S \times S$; there exists a transition from s_1 to s_2 if and only if $(s_1, s_2) \in T$. The tuple (S, T) forms a directed graph G that we call a *transition diagram.*

In the case of the labelled trees we use for modelling device configurations, structural modifications are limited to addition of a labelled node to a leaf, or deletion of a leaf node in the tree. These modifications intuitively refer to addition, deletion or modification of a parameter in the configuration of a device. Therefore, it is possible that no transition exists in either direction between two given states: this explains why T is only a subset of all possible pairs of states.

A *path* is a finite sequence of states $s_1, \ldots, s_n$ such that, for any s_i, s_{i+1}, there exists a $t \in T$ such that $t = (s_i, s_{i+1})$. The *distance* between two states, noted $\Delta(s_1, s_2)$ is the length of the shortest directed path linking them.

The configuration problem of the previous section becomes, in this system, the study of all paths that start from a given configuration, s_s, and end at a target configuration s_t. In addition, one might want to find the shortest of such paths.

However, this system, taken as is, is too general for any practical use. In particular, we must make sure that only solutions that progress towards the target are possible. We hence use path constraints to limit our study to sequences of states that have a meaning and are not degenerate.

A solution is to remove all tuples $(s_1, s_2) \in T$ such that $\Delta(s_1, s_t) < \Delta(s_2, s_t)$. This condition makes sure that the parameters that are actually added are part of the solution, but not of the start state, and that parameters that are removed are part of the start state, but not part of the solution. Any other modification is out of the way of an acceptable solution. This distance restriction also has for effect of removing any loops in the paths.

4.3.2 Temporal Constraints

Now that G has been trimmed of any inappropriate states and paths, we can add further restrictions by imposing on the remaining transitions the semantic constraints related to the situation we are trying to model.

For example, in order to Temporal Constraint 3 in the case of the VPN deployment exposed in section 4.2.2, we must remove all transitions that lead to states where node A of Figure 4.3 is added while node B is not present.

More semantic constraints can be added to further trim the state graph from unwanted states and transitions. The remaining paths satisfy all defined constraints. Intuitively, these so-called *acceptable* paths can be seen as a semantically desirable candidate solution for transforming the start state into the target final state.

4.3.3 Structuring Operations

In the minimal VPN example described previously, containing only two provider edge and two customer edge routers, the resulting state graph is composed of over 15,000 states spanning a proportional number of paths. These figures suggest that raw state graphs are far from being meaningful and manageable by hand. However, it is possible to simplify further this graph by studying patterns that can be found in it.

First, we can observe that the remaining graph $G' = (S', T')$, induces a (strict) partial ordering $\sqsubset$ in S' defined in the following way:

$$x \sqsubseteq y \Leftrightarrow (x, y) \in T'$$

Intuitively, the relation T' is transitive: if a set of operations α transforms configuration x into configuration y, and a set of operations α' transforms configuration y into configuration z, then the composition of α and α' transforms x into z. T' is antisymmetric, since the restriction to acceptable paths forces transitions to strictly decrease the distance from the goal. For that same reason T' is not reflexive.

Experimentally, we conjecture that the tuple $L = (S', \sqsubset)$ forms a bounded lattice (Davey et Priestley, 1990); the graph G can also be seen as the Hasse diagram of L. This lattice of possible states and transitions on states can then be studied for interesting properties. It is from these properties that a global procedure describing legal transformations to the configurations will be deduced.

4.3.3.1 Components

The first step is to recognize the presence of *components*, i.e. of closed groups of interleaved actions.

For instance, the parameters that contribute to the creation of the VPN routing and forwarding tables on the PEs of the service provider are bound by constraints of equations (1) and (2). They impose that all VRF actions on a router be done before passing on to another router, and that all VRF configuring must be done before going on to another aspect of the configuration.

Therefore, all actions of VRF configuration on a single router may be done in any order, but must all be done before doing anything else. Such a set of n actions thus forms a component and appears as a Boolean n-dimensional cube in the Hasse diagram of the lattice, as shown in Figure 4.4. The left structure shows a component where three actions, α, β and γ, can be performed in any order, resulting in 6 different paths. The right structure shows a similar Boolean cube for 4 different actions.

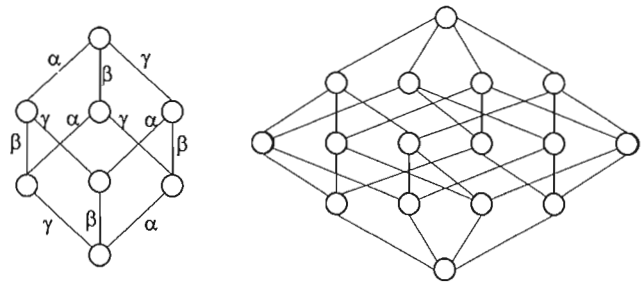


Figure 4.4: A closed set of interchangeable operations forms a component

These components act as a "capsule" of operations that must be performed atomically (i.e., that must not be mixed with any other operation).

We see that identifying components is an important tool to reduce the complexity of the state graph. Each component has a unique start and end point, and can therefore be assimilated to the single edge linking these two points.

Identification of such components leads to the construction of a reduced state graph where some edges no longer represent a single transition, but rather whole sets of transitions (Boolean cubes) that can be performed in arbitrary order. Figure 4.5 shows how a reduced state graph can be obtained from a state graph. One can identify 2, 3 and 4-dimensional Boolean cubes that are linked together. For example, points P1 and P2 are the endpoints of a component: all states between P1 and P2 have no contact with any other state. Since these cubes represent components made of swappable actions, they can be identified as such and be identified with their endpoints to form a reduced state graph, as shown on the right part of the picture.

4.3.3.2 Milestones

The notion of component naturally leads to that of a *milestone*. A milestone is an element $x \in S$ such that for all $y \in S$, either $x \sqsubseteq y$ or $y \sqsubseteq x$. For example, in Figure 4.5,

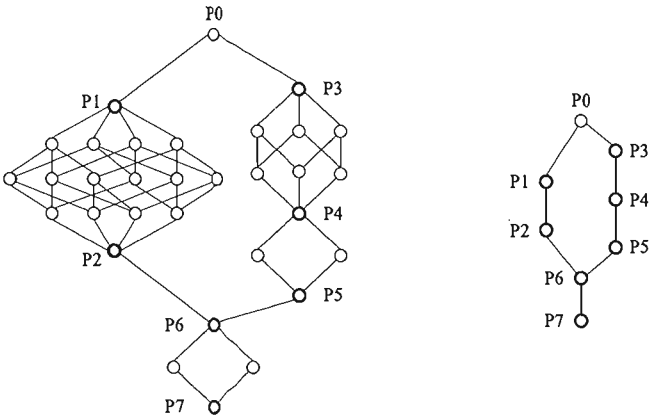


Figure 4.5: A complete state graph and its associated reduced state graph.

states P0, P6 and P7 are milestones.

Milestones can be thought of as unavoidable steps in the path from start to solution, since all acceptable paths must eventually pass by those points, in the order they appear. Therefore, milestones are good candidates to divide the modelled process into natural macro-steps of which they are the boundaries. In the case of Figure 4.5, two macro-steps can be identified: the transition from the start state P0 to state P6, and the transition from P6 to P7. The word "natural" is used here, since these milestones emerge from the set of temporal constraints imposed on the lattice. Different temporal constraints generally lead to different milestones.

The concept of milestone can also be applied to any sub-lattice of L. In particular, inside each macro-step, there can be local milestones that may be viewed as natural sub-steps. The process can be recursively repeated and yields a natural, hierarchical decomposition of the whole process into nested natural blocks of swappable operations. Hence, states P1-P5 are sub-milestones, or milestones of order 2.

4.3.4 Computing Components and Milestones

In order to be efficient, the structures defined in the previous section must be found easily and automatically by means of algorithms. In this section, we provide algorithms for finding milestones and for identifying components, and give an overview of their complexity.

4.3.4.1 Finding Milestones

The first algorithm we present is aimed at finding milestones in a lattice. Its principle is easy: for a state s, we can mark all states reachable from s by a (depth-first or breadth-first) graph traversal starting at s. Similarly, the set of states from which we can reach s can be computed and marked by a traversal of the transpose graph (going "up" instead of "down"). The chosen vertex is a milestone if and only if all vertexes of the lattice are marked at the end of the algorithm.

A brief analysis of this algorithm shows that determining whether a given vertex is a milestone takes no more execution steps than the total number n of vertices's in the lattice. Therefore, for finding all milestones, it suffices to repeat the algorithm starting at each vertex; the resulting complexity is therefore in $O(n^2)$. A more complex, but *linear* algorithm can also be found (Habib *et al.*, 1995).

4.3.4.2 Finding Components

In the same way as the definition of milestones is linked to that of components, the algorithm for finding components relies on the milestone-finding algorithm. We proceed to the same reachability analysis than in the previous section, except that instead of merely marking a vertex as visited, we explicitly state from which original vertex it was

reached. At the end of the process, each node has a list of the nodes from which it is reachable, either forwards or backwards. A vertex m is a milestone if all the vertexes in L have m in their list.

Then, for each milestone m_1 and its immediate milestone successor m_2, we consider the subgraph of all points between m_1 and m_2. This subgraph is divided into a number of disjoint sub-lattices $L_1, L_2, \ldots, L_n$. If one of these L_i has no milestone (which can be easily obtained by analyzing the lists a second time), then the sub-lattice $L_i \cup \{m_1, m_2\}$ is a component.

As we can see, the overall complexity of this procedure depends on the maximum level of nesting where milestones can be found. However, a rough worst case can be calculated by supposing there can be no more nesting levels than there are elements in L. At the first step, it takes $O(n^2)$ operations to find the milestones of the first level. It again takes a time proportional to the square of their size to find all sub-milestones found in these sub-lattices. However, all nested sub-lattices found after removing the first-order milestones are disjoint. Therefore, the total time needed to find all second order sub-milestones is again in $O(n^2)$, where n is still the total size of L. Since the nesting level of any component is at most n, the total number of steps required is in $O(n^3)$.

4.4 Applications

The main advantage of the analysis of the lattice that arises from temporal constraints is that it induces a way of synthesizing a protocol for the implementation of a service. By placing validation points at milestones, we ensure such checkpoints are optimally placed in semantically sound locations throughout the deployment process. Since these checkpoints reflect the structure imposed by the temporal constraints, they

also make optimal points to roll back in case a failure occurs.

We have succeeded in analyzing the deployment of a Virtual Private Network for the basic case of four routers and identified six main milestones. This is helpful in practice. For instance, in an existing tool called NetconfMaker (Cherkaoui *et al.*, 2004), the user must manually set validation point in order to obtain a transactional model on top of the Netconf protocol. Due to the large number of possible solutions, this is not an easy task. However, by using the approach presented in this paper, it is possible to feed NetconfMaker with scripts enabling it to proceed automatically to the discovery of these validation points. Figure 4.6 shows an example of this transactional model in the case of the deployment of a VPN.

The granularity of the configuration components and validation operations depends on how tightly the semantic dependences are coupled within the components and the complexity of these components. For instance, in the case of the VPN example, the BGP component can be split into two subcomponents: the first dealing with the creation of the BGP process and the second with the neighbour information configuration. Another component refers to the mutual redistribution of the routing information between the IGP, the static routing used, or the connectivity between PE-CEs and the BGP process. If we take into account the initial underlying sub-services (establishing the connectivity between PE-CEs, between PE-PCs and MLPS), we obtain six components, to which we add the initial constraints, obtained form the customer and the service provider choices.

One aspect of establishing validation points takes into account the hierarchy existing among the configuration transactions. Establishing the network-level validation points ensures the consistency and integrity of the configuration transactions that involve multiple equipments, roles and configuration parameters.

In this hierarchy, two validation points are set at network level. The first one, which is placed during the preparation phase upon getting all the initial device configurations,

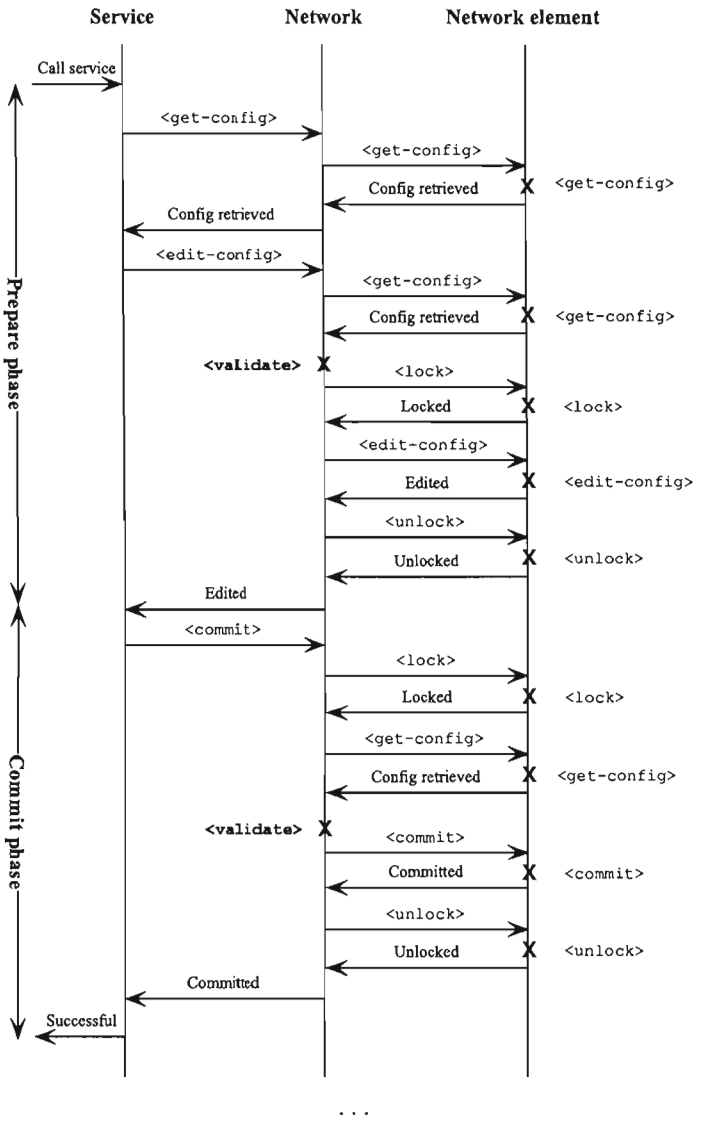


Figure 4.6: Transactional model for the Netconf protocol. Calls to <validate> are placed at validation points.

ensures that the initial multiple device configurations are consistent before edition. The second one, which occurs during the commitment phase upon getting all the modified device configurations, ensures that the modified multiple device configurations are consistent before committing them.

Moreover, the knowledge of milestones and components for a given service allows for the creation of more structured Management Information Bases (MIBs) and Policy Information Bases (PIBs), where the access mechanisms to configuration parameters could be designed according to the temporal dependencies discovered.

4.5 Conclusion

In this paper, we have shown using examples that configuration parameters in network devices are subject to syntactical and semantic dependencies which, when deploying a network service, may impose that some of the configuration operations be done in a specific order. We also explained how a mathematical framework using lattice theory can model these ordering constraints. The concepts of components and milestones, defined in terms of paths in the lattice structure, help to simplify the analysis of possible solution paths and provide a sound criterion for dividing the deployment of a service into natural macro-steps that serve as validation checkpoints.

In particular, a deeper study of the implementation of an MPLS VPN in a simple case was found to be divided into six ordered main natural components whose internal configuration operations are mutually swappable. These results are in accordance with the intuitive vision of the deployment of this service.

Further work on this concept can lead to a thorough study of the deployment of a number of network services that could suggest the location of optimal validation points.

CHAPITRE V

SEQUENTIAL DEPENDENCIES IN CONFIGURATION OPERATIONS

Ce chapitre est tiré de l'article suivant :

> Hallé, S., Deca, R., Cherkaoui, O., Villemaire, R., Puche, D. (2006). Sequential Dependencies in Configuration Operations. In *Actes du 7^e Colloque francophone de Gestion de Réseaux et de Services (GRES 2006)*, 112-123.

Ce chapitre reprend les principes du chapitre 4 en les formalisant davantage. Alors qu'au chapitre précédent, des notions élémentaires de treillis étaient utilisées pour représenter les composantes et les *milestones*, un pas supplémentaire est effectué ici en les réexprimant au moyen de formules de logique temporelle, suivant le principe fédérateur de ce travail.

De plus, on y trouve les premiers résultats expérimentaux de cette thèse en ce qui a trait à la vérification de propriétés dynamiques ; des résultats plus poussés seront présentés au chapitre suivant. Enfin, on donne des résultats empiriques préliminaires sur la génération automatique d'une trace satisfaisant une propriété temporelle donnée, faisant ainsi écho au problème de décidabilité d'une logique temporelle soulevé au chapitre 3.

Abstract

The deployment of a network service is subject to a number of semantical and sequential dependencies. However, one of the main issues with existing configuration management approaches is the lack of a transactional model, which should allow the network configuration data to retain their integrity during the configuration process. In this paper, we introduce the notion of sequential dependency, propose a mathematical framework based on model checking that allows the structuring of configuration operations leading to the concept of milestone state, and suggest how the Netconf protocol can be enhanced with a transactional component.

5.1 Introduction

The deployment and configuration of network services is a complex and error-prone task that is subject to constraints at different levels. For instance, *semantical* dependencies between parameters dispersed among multiple configuration operations appear in even the simplest management tasks (Hallé *et al.*, 2004). Although these dependencies are not currently captured by management protocols such as Netconf (Enns, 2005), it has been shown how tree logics can help in automating their formalization and verification on a given configuration snapshot (Hallé *et al.*, 2005).

However, even if semantical dependencies can be automatically verified, an important part of the complexity of deploying a service still remains. In general, the configuration operations must be performed in a specific order that is determined by the connected nature of the network, or even by some requirements of the devices' operating system. Therefore, in addition to semantical dependencies, there are *sequential* dependencies that

We gratefully acknowledge the financial support of the National Sciences and Engineering Research Council of Canada and Cisco Systems.

need to be formalized and checked in a similar fashion. The importance of checking these sequential dependencies is heightened by the fact that an increasing number of services, such as Virtual LANs and Virtual Private Networks, involve configuration changes on multiple devices at the same time.

While the semantic dependences in network device configurations have been widely debated in the network management literature (Damianou *et al.*, 2001; Warmer et Kleppe, 2003; Crubézy, 2002; Jackson *et al.*, 2000; Hallé *et al.*, 2005), the sequential dependences have not yet been extensively covered. Among the works on the subject, (Couch et Sun, 2003) examine a convergent approach to automated configuration and provide an algebraic model of configuration management. According to this model, the managed processes can be decomposed into regions or intents of non-conflicting, stateless actions. Each of these non-commutative regions can then be processed separately. Using this model, procedural processes composed of non-commutative operations can be redesigned as declarative processes, which are composed of commutative operations. The authors illustrate their approach with examples from file editing.

The transactional aspect of the device configuration process is taken into account by the Netconf configuration protocol (Enns, 2005), which is a new protocol designed for manipulating network device configurations. However, this protocol provides transactional operations at the device level, but does not currently have similar operations for the network level.

The purpose of this paper is twofold. First, we raise the question of sequential constraints in network configuration operations and show by some of examples that their presence is as common as semantical ones; using concepts borrowed from the field of model checking, we also demonstrate how these constraints can be accurately formalized in Kripke structures by temporal logic formulæ. Second, we define the notion of *milestone* states in a Kripke structure and show that these states make good candidates

for validation, synchronization and rollback points during the deployment of a service, and illustrate how the Netconf protocol could be enhanced by the addition of a similar transactional component.

The paper is structured as follows. In section 2, we briefly overview the concepts of configuration tree and semantical dependencies and introduce by means of concrete examples the concept of sequential dependency. We also describe the mathematical framework of model checking and show how sequential dependencies can be modelled by temporal logic. In section 3, we introduce the concept of milestone and apply it to the Netconf protocol. Finally, Section 4 shows some experimental results and Section 5 concludes with future work.

5.2 The Sequential Aspect of Network Management

The deployment of a service over a network basically consists in altering the configuration of one or many equipments to implement the desired functionalities. We can assume without loss of generality that all properties of a given configuration are described by attribute-value pairs hierarchically organized in a tree structure (Hallé *et al.*, 2004; Villemaire *et al.*, 2005a) like the one shown in Figure 5.1.

Possible alterations to the configuration typically include deleting or adding new parameters to a device configuration, or changing the value of existing parameters. In most cases, the parameters involved in such modifications are both syntactically and semantically interdependent. For instance, the value of some parameter might be required to depend in a precise way on the value of another parameter; the simplest example of such dependency is the fact that an IP address must match the subnet mask that comes with it. More complex dependencies might constrain the existence of a parameter to the existence of another. Recent works have shown how such dependencies

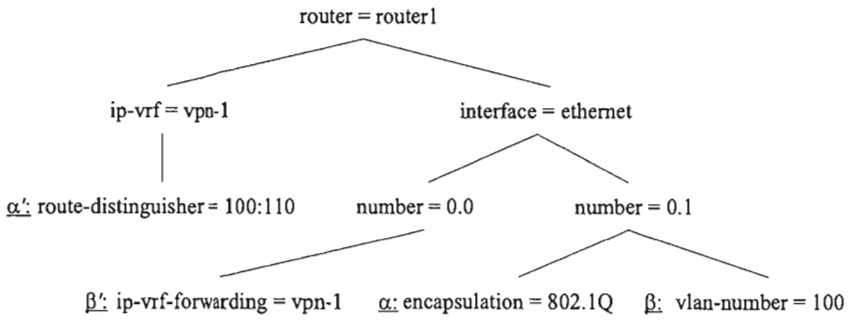


Figure 5.1: A sample configuration tree. Nodes labelled α, α', β and β' are not present initially, but are added in the process of deploying the network services given later as examples.

can be automatically checked by logical tools on a given configuration snapshot (Hallé *et al.*, 2005).

5.2.1 Sequential Dependencies at the Service Level

The situation becomes quite complex when one wants to actually *deploy* a service from scratch. In addition to constraints on the values of parameters, the dependencies may also impose that the modifications be performed in a specific order. When done in an uncoordinated way, changing, adding or removing components or data that implement network services can bring the network in an inconsistent or undefined state. This fact becomes acutely true in the case where operations must be distributed on multiple network elements, as they cannot be modified all at once. Moreover, while a single inconsistent device can ultimately be restarted when all else fails, there is no such "restart" option when an entire network configuration becomes inconsistent.

We illustrate the concept of sequential dependencies by means of two examples taken from the deployment of network services. For each of these examples, a sequential dependency is extracted and formalized.

5.2.1.1 Example 1: Virtual LANs

A Virtual LAN (VLAN) is a group of devices spanning multiple LAN segments that are configured to communicate as if they were connected to the same wire. Each VLAN works as a completely separate entity that can only be joined by a router. Since VLANs are logically (rather than physically) structured, they are extremely flexible. Among the several protocols designed for this purpose, IEEE 802.1Q (IEEE, 2003) has become the standard.

When configuring a router on a VLAN, the sub-interface that is connected to a VLAN *trunk* must be set to support the 802.1Q protocol; since each sub-interface is attached to a specific VLAN, the number of this VLAN must also be specified when configuring the trunk. Therefore, configuring a VLAN trunk will have for effect of adding nodes α and β in the configuration tree of Figure 5.1.

However, 802.1Q frames are designed so that they must contain the VLAN number; therefore, encapsulation and VLAN number must be configured together. From this simple example, one can deduce this first sequential rule:

> **Sequential Rule 1.** *In a router, the VLAN number must be set at the same time the encapsulation protocol is enabled.*

In the case of Figure 5.1, this means that nodes α and β must be added to the tree in the same step.

5.2.1.2 Example 2: Virtual Private Networks

A VPN is a private network constructed within a public network such as a service provider's network (Rosen et Rekhter, 1999). A customer might have several sites, which

are contiguous parts of the network, dispersed throughout the Internet, and would like to link them together by a protected communication. The VPN ensures the connectivity and privacy of the customer's communications between sites.

Such a service consists of multiple configuration operations; in the case of Layer 3 VPNs, it involves setting the routing tables and the VPN forwarding tables, setting the MPLS, BGP and IGP connectivity on multiple equipments having various roles, such as the customer edge (CE), provider edge (PE) and provider core (PC) routers. An average of 10 parameters must be added or changed in each device involved in the deployment of the VPN.

As an example, for a Layer 3 VPN using MPLS, Figure 5.1 shows two leaf nodes that must be added, each in its own position, to the configuration tree of a PE router. Node α' corresponds to the creation of the Virtual Routing and Forwarding Tables (VRF) necessary for the proper functioning of the VPN; node β' associates this VRF to a specific interface on the router. Semantically, it is clear that one cannot associate a VRF to an interface before the VRF is created in memory. Therefore, trying to add node β' to the configuration before node α' is created is nonsensical and generates an error. From this situation, we can elicit a second sequential rule:

Sequential Rule 2. *To add node* `ip-vrf-forwarding` *to a configuration tree, the node* `route-distinguisher` *must already be present.*

We must stress that the node `route-distinguisher` has to be present in the tree *before* node `ip-vrf-forwarding` is added, which rules out the possibility that both nodes be added in a single operation.

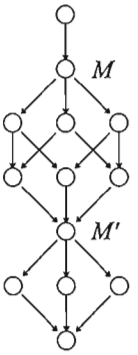


Figure 5.2: A Kripke structure with multiple paths from a start state to a target state. Each state represents a labelled tree.

5.2.2 Formalizing Sequences of Configuration Operations

To formalize the sequences of operations, we first need to introduce some basic concepts taken from the model checking theory (Clarke *et al.*, 2000).

Let S be a set of *states* representing a unit situation at a given time. In the context of network configuration, states are labelled trees, as described previously.

We call *transition* from a state s_1 to a state s_2 the structural modifications that transform s_1 into s_2. Formally, transitions can be defined as a subset of tuples $T \subseteq S \times S$; there exists a transition from s_1 to s_2 if and only if $(s_1, s_2) \in T$. The tuple (S, T) forms a directed graph G called a *Kripke structure*. Figure 5.2 shows an example of a Kripke structure.

In the case of the labelled trees we use for modelling device configurations, structural modifications are limited to addition of a labelled node to a leaf, deletion of a leaf node and change in a node's value. These modifications intuitively refer to addition, deletion or modification of a parameter in the configuration of a device.

A *path* is a finite sequence of states $\langle s_1, \ldots, s_n \rangle$ such that, for any s_i, s_{i+1}, there

exists a $t \in T$ such that $t = (s_i, s_{i+1})$.

The deployment of a service is a path in such a structure that starts from a given configuration, s_s, and ends at a target configuration s_t. For example, in the case of Figure 5.1, a possible start state could be the tree without any of α, α', β, β', and a possible target state could be the same tree with all these nodes. A valid deployment sequence could be a sequence of addition of the nodes that s, among other things, Sequential Rules 1 and 2.

5.2.3 Formalizing Sequential Dependencies

The state space generated by spanning all possible transitions between a start and target state is fairly large. For the 4 nodes of Figure 5.1, there are 24 possible unconstrained paths, and in general, for n possible operations, there are $n!$ possible paths. We must now restrict our study to acceptable paths —that is, paths that the elicited sequential constraints. For this purpose, we use the Linear Temporal Logic (LTL) commonly used in model checking (Clarke *et al.*, 2000).

LTL is a logic aimed at describing sequential properties along paths in a given Kripke structure. Its syntax is based on classical propositional logic, to which modal path operators have been added.

The first such modal operator is **G**, which means "globally". Formally, the formula **G** φ is true on a given path π when, for all states along this path, the formula φ is true. A second commonly used modal operator is **X** ("next"). The formula **X** φ is true on a given path π of the Kripke structure when the next state along π satisfies φ. Finally, a formula of the form **F** φ ("eventually") is true on a path when at least one state of the path satisfies φ. Other modal operators exist, but go beyond the scope of this paper.

A LTL formula is a well-formed combination of the classical $\vee$ (disjunction), $\wedge$ (con-

junction), $\neg$ (negation), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence) operators with modal operators. The *atoms* of LTL are the base-level Boolean expressions over which the formulæ are built. In the present case, since states are labelled trees, we take the atoms to be formulæ themselves, based on a tree logic such as CL (Villemaire *et al.*, 2005a).

Equipped with LTL, it is now possible to formalize sequential constraints into logical formulæ. Without delving into further details, suppose that φ_n is a CL formula that is true if and only if node n is present in a given configuration tree. Then, the Sequential Constraint 1 presented in the previous section can be translated into the following formula:

Sequential Formula 1.

$$\mathbf{G}\ \varphi_\alpha \leftrightarrow \varphi_\beta$$

This formula means that in all states of all paths where either α or β exists, the other node must also exist. All temporal rules described earlier can similarly be translated into LTL formulæ. For example, Sequential Rule 2 becomes:

Sequential Formula 2.

$$(\mathbf{G}\ (\neg\varphi_{\alpha'} \rightarrow \mathbf{X}\ (\varphi_{\alpha'} \rightarrow \neg\varphi_{\beta'}))) \vee (\mathbf{G}(\varphi_{\beta'} \rightarrow \varphi_{\alpha'}))$$

telling that the presence of node β' implies that node α' is present, and that is must also have been present at least in the previous step in the deployment.

5.3 Transactional Aspects of the Netconf Protocol

We now show how transactional aspects presented in the previous section can be applied to the Netconf protocol by enhancing it with a transactional component.

5.3.1 Overview of the Netconf Protocol

To send configuration commands to a router, Netconf provides a set of "remote procedure calls" (RPC) and RPC-replies. In a simplified way, an RPC is a block of XML data whose opening tag contains an identifier that either asks the router to return a portion of its configuration file, or tells it to replace a part of its configuration with a snippet provided by the user and carried in the body of the RPC. Netconf offers other built-in operations, such as commands allowing to lock a part of a router's configuration so that only the current user can modify it, and subsequently unlock it. The RPC-reply is the XML block that is returned to the user.

The Netconf protocol defines a simple mechanism for device management. However, its transactional model, which includes a validation capability, is device-centred, and does not provide a mechanism to ensure the consistency of the sequence of operations with to the rules elicited in section 2.

In order to bestow transactional semantics on the update operations of multiple configurations, it is important to determine the optimal points of validation, commitment and roll-back during the update process of the network device configurations. We propose to determine these points by analyzing special properties of the Kripke structure induced by the sequential dependencies.

5.3.2 Components and Milestones

We first introduce the notion of *milestone state*. A milestone state is a state m by which all valid paths must eventually pass. Formally, let ψ be some LTL temporal rule, and τ_m be a CL formula that is true only on state m. Then, in a Kripke structure G, m is such that for every path beginning at the start state and that satisfies ψ, the formula F τ_m is true.

Milestones can be seen as unavoidable steps in the path from start to solution, since all acceptable paths must eventually pass by those points, in the order they appear. In the case of Figure 5.2, we see two milestones, labelled M and M'.

We argue that milestones are good candidates to divide the modelled process into natural macro-steps of which they are the boundaries; complementary to milestone states are *components*, i.e. sets of states and transitions comprised between two milestone states. The word "natural" is used here, since these milestones emerge from the set of temporal constraints imposed on the lattice. Different temporal constraints generally lead to different milestones.

5.3.3 Towards a Transactional Model

The main advantage of analyzing the lattice that arises from temporal constraints is that it induces a way of synthesizing a protocol for the implementation of a service. By placing validation checkpoints at milestones, we ensure such checkpoints are placed in semantically sound locations throughout the deployment process. Since these checkpoints reflect the structure imposed by the temporal constraints, they also make good points to roll back in case a failure occurs.

These points are important since they represent optimal places of validation in the flow of operations. Thus, a validation performed at of these points can check all or most of the dependences that apply on the multiple flow streams that converge towards the validation point. Moreover, these convergence points are unavoidable during the configuration and, since they concentrate the flow paths, a validation performed at such points provides a maximum extent of coverage for those flows.

Intuitively, we suggest that a validation point be used to validate the operations that are situated along the flow path connecting it to a previous upstream milestone, in which

a validation has been already done. If the validation has been successful, the update information generated by the operations is committed. Otherwise, if the validation or the commitment fails, the update information generated by the operations is discarded and the configurations are rolled back to the latest points of successful validation.

Netconf provides two phases of successful configuration transaction during a service configuration procedure: preparation and commitment. During preparation, the configurations are retrieved from the network devices. When all configurations have been retrieved, edition starts at service level. This validation stage ensures that the network configuration is consistent before the proposed modifications required by the service. To ensure integrity of the configuration edition, the device configurations are locked, edited and subsequently unlocked. When the service edition has been successfully accomplished, commitment starts. This validation stage ensures that the network configuration remains consistent after the successive modifications of the network configurations.

Since the network service update affects multiple device configurations, a two-phase commit is required. The first phase stores the update information on temporary storage and validates it before entering the second phase. If the validation is successful, the update information is transferred onto the real configurations, otherwise this information is discarded. In case of erroneous transfers during the second phase, the configurations are rolled back and the second phase can be resumed.

This semantics can be used with the Netconf configuration protocol to ensure the transactional properties of the service updates on multiple devices. As already mentioned, the Netconf protocol defines transactional operations for device level but does not provide similar operations for network level, i.e. for the multiple device configurations supporting a network service.

Obviously, the higher-level validations may involve multiple devices. For instance,

the routing table and the protocol information in a device depend on the network addresses and the protocol information from other devices. Similarly, parameters such as protocol neighbours' IP addresses, autonomous system numbers, protocols' process number and IP addresses must accurately correspond on more than one device, in order for the network service that is deployed over that network to be consistent.

In this case, defining an operation that can validate multiple parameters situated on several devices might be highly recommendable. This multi-device validation operation would replace multiple single-device validation operations and would allow performing complex validation queries directly within the given configuration protocol. Figure 5.3 presents a sequence diagram of transactional operations for multiple devices using the Netconf configuration protocol.

5.4 Experimental Results

Since structures generated by service deployments are Kripke structures and since sequential constraints can be formalized in LTL, it is possible to submit the problem directly to a model checker like NuSMV (Cimatti *et al.*, 2002). Using this tool, we generated sample deployment sequences and checked that these deployments ed a set of constraints similar to Sequential Rules 1 and 2. For each test, we varied the number of nodes in the sequence and the number of sequential constraints imposed on each nodes. The validation times for these experiments are summarized in Figure 5.4. All times given in this section have been calculated on an AMD Athlon XP-M 2200+ running NuSMV 2.1.2 under Cygwin.

One can see that validation times for large sequences of operations (up to 150 nodes) remain under the reasonable bound of 10 seconds, and that augmenting the number of constraints is not the key factor that makes the computation longer.

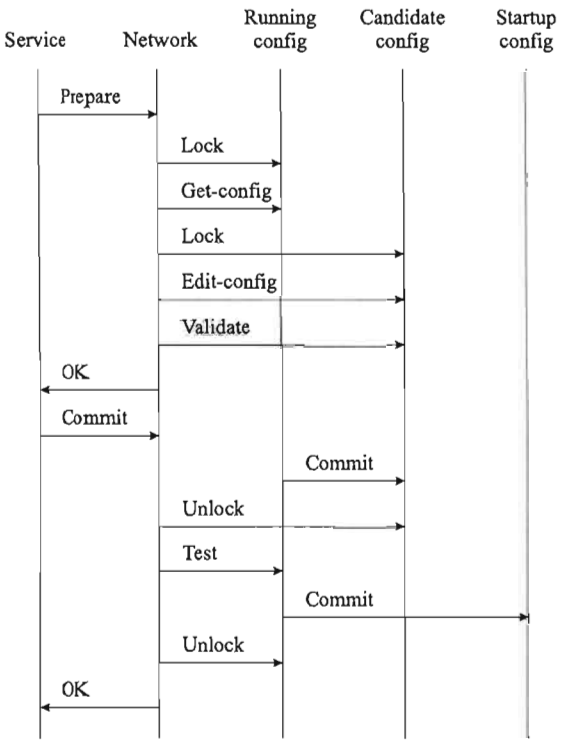


Figure 5.3: Transactional operations for multiple devices using the Netconf configuration protocol. Some `OK` replies have been omitted.

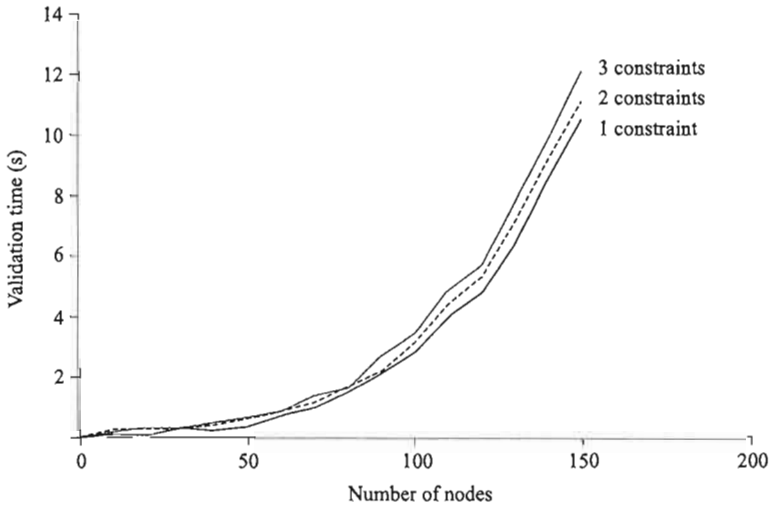


Figure 5.4: Validation time of a deployment sequence in terms of number of nodes to alter and constraints per node.

Additionally, it is possible to benefit from the counter-example generation mechanism of NuSMV to find a deployment sequence that does not violate any constraint. As a matter of fact, when a LTL property of the form $\mathbf{G}\ p$ is false, NuSMV provides the user with an execution trace on the Kripke structure for which p is false. If p is the LTL property one wants to verify on a structure, it suffices to submit the formula $\mathbf{G}\ \neg p$ for verification. If there exists a trace for which p is true, then such a trace is a counter-example for the formula $\mathbf{G}\ \neg p$, and therefore NuSMV will display it to the user, giving by the same occasion a valid deployment sequence.

We have conducted experiments with NuSMV on sample deployment sequences with constraints of the same form as Sequential Formulæ 1 and 2. We varied the size of the configurations and the number of sequential constraints per node imposed on the structure, and computed the time NuSMV took to provide a correct deployment sequence. The results of these experiments are presented in Figure 5.5. Each curve corresponds to the generation time of a valid deployment sequence involving some number of nodes,

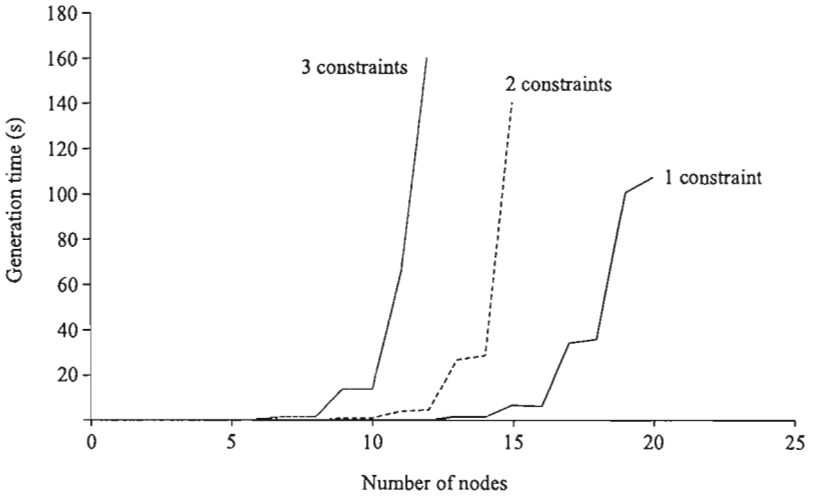


Figure 5.5: Generation time of a valid deployment sequence in terms of number of nodes to alter and constraints per node

with 1, 2 or 3 sequential constraints imposed on each *node* —that is, the total number of constraints actually increases with the number of nodes.

As expected, generating a valid sequence is much harder than validating an existing one. Moreover, the number of sequential constraints on each node does matter in this case, and can change a rather simple situation into an intractable one. One can see that, for sequences that involve the addition or modification of about 10 nodes, which is comparable to deployment of a simple VPN on a router, up to three sequential constraints per node can be imposed without the generation time becoming prohibitive.

These findings suggest that model checking is indeed an interesting tool for on-the-fly validation of deployment sequences, and for offline, *a priori* synthesis of valid sequences for network services with a complexity comparable to a Virtual Private Network.

5.5 Conclusion

In this paper, we have shown how Linear Temporal Logic applied to Kripke structures can accurately formalize sequential constraints in the deployment of network services. Using these model checking concepts, we defined the notion of *milestone* states in a Kripke structure and gave arguments for using these points as validation, synchronization and rollback points during the deployment of a service, and illustrated how the Netconf protocol could be enhanced by the addition of a transactional component based on milestones.

Empirical results on sample network configurations demonstrate the feasibility of validating deployment sequences using model checking tools, and show that finding a deployment sequence that validates a set of constraints is a computationally hard problem.

We plan future work on these concepts in order to further use milestones in a hierarchical decomposition of a service deployment. In such a setting, each component could contain sub-milestones that would further divide a process into sub-steps based on the same principle. Moreover, the current methodology could be extended by considering all possible orderings of operations in a component and eventually reduce the study to one specific ordering, in the same way *partial order reduction* reduces the state space in model checking (Clarke *et al.*, 2000).

CHAPITRE VI

SPECIFYING AND VALIDATING DATA-AWARE TEMPORAL WEB SERVICE PROPERTIES

Ce chapitre est tiré de l'article suivant :

Hallé, S., Villemaire, R., Cherkaoui, O. (2008). Specifying and Validating Data-Aware Temporal Web Service Properties. Soumis aux *IEEE Transactions on Software Engineering*, février 2008.

Cet article est lui-même la synthèse des deux publications suivantes :

Hallé, S., Villemaire, R., Cherkaoui, O., Ghandour, B. (2007). Model-checking Data-Aware Temporal Workflow Properties with CTL-FO+. In *Proceedings of the 11^{th} IEEE International EDOC Conference (EDOC 2007)*, IEEE Computer Society Press, 267-278.

Hallé, S., Villemaire, R., Cherkaoui, O., Tremblay, J., Ghandour, B. (2007). Extending Model Checking to Data-Aware Temporal Properties of Web Services. In *Proceedings of the 4^{th} International Workshop on Web Services and Formal Methods (WS-FM 2007)*, Springer Verlag : Lecture Notes in Computer Science 4937, 31-45.

Cet article constitue l'aboutissement de la démarche débutée au chapitre 1. En effet, la logique CTL-FO$^+$ qui y est formellement introduite utilise les résultats des chapitres précédents.

Utilisant l'exemple de l'environnement UCLP, ce chapitre démontre l'existence de contraintes pour lesquelles la séquence des opérations invoquées est importante, étendant ainsi les principes énoncés aux chapitres 4 et 5. Cependant, contrairement à ces deux chapitres, les contraintes font également référence au contenu des messages, qui sont des structures XML échangées avec des partenaires ; pour ce faire, un fragment de la logique CL présentée dans un contexte concret au chapitre 2 est utilisé.

Finalement, la procédure de *model checking* développée à la section 6.6 utilise la technique de quantification « freeze » présentée aux chapitres 1 et 3.

Abstract

Most works that extend workflow validation beyond syntactical checking consider constraints on the sequence of messages exchanged between services. These constraints are expressed only in terms of message names and abstract away their actual data content. However, motivated by the context of the User-controlled Lightpath initiative (UCLP) hosted by the CANARIE consortium, we provide examples of real-world "data-aware" web service constraints where the sequence of messages and their content are interdependent.

To this end, we present CTL-FO$^+$, an extension over Computation Tree Logic that includes first-order quantification on state variables in addition to temporal operators. We show how CTL-FO$^+$ is adequate for expressing data-aware constraints, give a sound and complete model checking algorithm for CTL-FO$^+$ and establish its complexity to be PSPACE-complete.

A "naïve" translation of CTL-FO$^+$ into CTL leads to a serious exponential blow-up of the problem that prevents existing validation tools to be used. We provide an alternate translation of CTL-FO$^+$ into CTL where the construction of the workflow model depends on the property to validate. We show experimentally how this translation is significantly more efficient for complex formulæ and makes model checking of data-aware temporal properties on real-world web service workflows tractable using off-the-shelf tools.

6.1 Introduction

There exists a large number of web service orchestration tools available over the Internet; these tools allow a syntactical validation of the service invocations in a workflow, since all input and output messages are publicized by service providers in WSDL

documents whose form and content is regulated by standards bodies such as the W3C. This "first generation" of web service technologies, as it is called by Zaha *et al.* (2006), concentrates on single request-response patterns of messages specified by various means such as Message Exchange Patterns (MEPs).

However, it has long been argued that syntactical correctness does not give a complete picture of the necessary conditions for a successful interaction with a service (Meredith et Bjorg, 2003). Nothing prevents a BPEL process from sending to a peer syntactically valid messages in a sequence that prevents an actual composition from taking place. This led the authors of (Greenfield *et al.*, 2003) to call for future work on a formal language to express and advertise the *protocol* imposed on the use of a service, and a methodology to check as much as possible that an orchestration script satisfies this protocol before it is allowed to execute. Depending on the authors, this concept has been dubbed *operating guidelines*, *conversation specification*, *user contract*, *protocol of interaction*. Irrelevant of the terminology, a wide consensus exists to the effect that specification of these constraints is beyond the expressive power of existing standards and available design tools.

A "second generation" of web service technologies has given rise to a variety of standards taking into account the sequence of message exchanges allowed by a service. The SOAP Service Description Language (SSDL) (Parastatidis *et al.*, 2005) is a notable example of this approach. Classical temporal languages such as Linear Temporal Logic (LTL), Computation Tree Logic (CTL) or the π-calculus have been suggested as appropriate notations for expressing temporal or conversational dependencies between message exchanges. Numerous automated validation tools have also been developed that can guarantee conformance of a given workflow to some set of operating guidelines (Lohmann *et al.*, 2006).

In Section 6.2, we briefly review related work and show why current model checking

solutions based on traditional temporal logics are not adequate for the validation task at hand. Although this new generation of technologies allows for a much more realistic specification and enforcement of interaction constraints, we shall see that most efforts still abstract the actual content that transits inside the messages of a given conversation. In other words, current protocol specifications treat messages as atomic units represented by their names; they are not "data-aware".

In this paper, we argue that "data-awareness" of protocol specifications is a fundamental part of ensuring workflow correctness. Using the context of the User-controlled Lightpath initiative (UCLP) hosted by the CANARIE consortium, we provide in Section 6.3 examples of real-world web service protocols where both the sequence of messages and their content are interdependent.

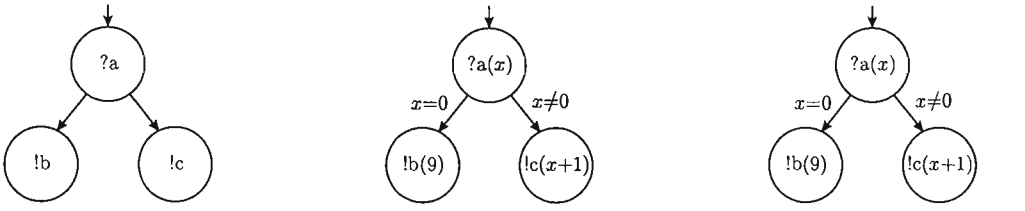
In Section 6.4, we present an extension of the well-known Computation Tree Logic (CTL) that introduces a general first-order quantification on values of state variables, called CTL-FO$^+$, as an appropriate formal language for the expression of temporal constraints on web service invocations. Contrarily to the classical temporal formalisms used in most web service validation approaches, CTL-FO$^+$ retains the full temporal flexibility of the CTL logic, while allowing to refer to the content of messages inside the temporal properties. We provide in Section 6.5 an explicit algorithm for model checking CTL-FO$^+$ formulæ on a given workflow model. We show how it differs from other methodologies suggested to model data-awareness and establish the complexity of the problem of model checking a CTL-FO$^+$ formula to be PSPACE-complete. This result places CTL-FO$^+$ model checking on a par, complexity-wise, with the Linear Temporal Logic (LTL) used by widely accepted tools such as SPIN (Holzmann, 2003). Therefore, we argue that data-awareness in web service validation is as tractable as modelling sequential properties in LTL, an approach that is already tackled by many second-generation workflow tools.

There exist numerous ways to transform the CTL-FO$^+$ model checking problem back into classical CTL model checking to leverage existing workflow tools and standard model checkers. For example, the CTL formula $\mathbf{AG}\,(a = x \rightarrow \mathbf{AF}\,b = x)$ correlates the values of state variables a and b at two different moments in time; it is a valid CTL formula when x is a static constant, but it cannot be used to express the same thing "for all x" unless the formula is repeated for every possible static value. This limited form of quantification is called *explicit*. Unfortunately, any such transformation results in an exponential blowup and shifts the original problem to the higher EXPTIME-hard class, unless P = NP. This result seems to suggest that data-aware properties are beyond the reach of existing tools.

However, in Section 6.6, we present a reduction of CTL-FO$^+$ to CTL that modifies the translation of a workflow into a finite-state system using the concept of "freeze quantification": the construction of the system becomes dependent on the property to validate. In Section 6.7, we compare this freeze quantification approach with the more straightforward *explicit quantification* suggested above. Although both translations are ultimately exponential, we empirically demonstrate that freeze quantification is several orders of magnitude more efficient. We illustrate our claim by showing a technology chain using two off-the-shelf tools, the VERBUS (Arias-Fisteus *et al.*, 2005) workflow translator coupled with the NuSMV (Cimatti *et al.*, 2002) model checker, to validate constraints on sample web service workflows. We conclude that despite the theoretical lower bound, it is nevertheless possible to model and validate data-aware properties in web services using existing technologies and a suitable reduction to CTL.

6.2 Related Work and Existing Solutions

The modelling and validation of constraints of various kinds on web service workflows has spawned a large amount of both theoretical and practical works. With respect to



?a ≫ !c
?a ≫ !b

(a) Propositional workflow, propositional properties

?a ≫ !c
?a(0) ≫ !b(9)

(b) Data-aware workflow, propositional properties

$\forall x, y : ?a(x) \gg (!c(y) \wedge y = x + 1)$
?a(0) ≫ !b(9)

(c) Data-aware workflow, data-aware properties

Figure 6.1: Workflow modelling with various degrees of data-awareness

the goal of this paper, they can be classified into three categories corresponding to the degree of data-awareness they exhibit. We mention here a few of them.

To support our point, we illustrate each of these categories using the simple example of Figure 6.1. We consider a web service workflow which receives from some partner a message labelled "a" that contains an integer value. If this received value is 0, then the service returns a message "b" with value 9. If the received value is not 0, then the service returns a message "c" that increments the received value incremented by 1. For the needs of the example, we employ a simplified notation to refer to messages sent (!a) or received (?a) and indicate the value of a message between parentheses. The ≫ symbol means "the next message".

6.2.1 Propositional Workflows, Propositional Properties

A first step is to use classical automata-theoretic constructions or model checking tools and languages to model the behaviour of a web service and its interaction with other services. This is exemplified in Figure 6.1a. The messages are considered *atomic*: their actual data content is abstracted away. We call such a model *propositional*, since

the external behaviour of web services is represented by the transmission or reception of messages that are identified by propositional letters standing for their names.

This entails that the choice between sending message "b" and message "c", since it depends on message content, is seen as non-deterministic by the model. For the same reason, the behavioural properties of the service can only be expressed in terms of message names; we call them *propositional* properties. The two formulæ at the bottom of Figure 6.1a respectively mean that when message "a" is received, the next message is "b", or that when message "a" is received, the next message is "c". Neither of these formulæ are always true on the modelled workflow.

A fair number of works use the propositional approach. Conversation specification (Bultan *et al.*, 2003) is an example of sequence of intertwined messages received and sent by multiple agents. Message Sequence Charts (MSC) have been modelled into finite state processes (Foster *et al.*, 2006). A similar approach has been done with use of the BPE-calculus and the Concurrency Workbench (CWB) (Koshkina et van Breugel, 2004) and Petri nets (Hinz *et al.*, 2005). Zaha *et al.* (2006) tackle the formal specification of a protocol of interaction between services expressed as a pattern of messages.

These works have been dubbed "data-agnostic" solutions (Deutsch *et al.*, 2006). It is important to note, however, that although they do not model data, this abstraction is an appropriate simplification to tackle problems that are outside the scope of the present paper. For example, (Zaha *et al.*, 2006) provides an algorithm that determines whether services are "locally enforceable"; modelling the data content in messages in such a work is an open problem, and would render such a question much more complex and perhaps intractable in practice.

6.2.2 Data-aware Workflows, Propositional Properties

A refinement over the previous solutions is to consider that the actual data exchanged in the messages of a web service can actually influence the control flow of that service: the workflow model becomes "data-aware". This refinement is illustrated in Figure 6.1b. The choice between sending message "b" or "c" is now unambiguous and determined by the value inside message "a". Moreover, the model correctly represents that the value inside message "c" is always incremented by 1.

This category constitutes the bulk of formal web services models. Kazhamiakin *et al.* (2006) model web service compositions by finite-state systems and studies them from the angle of synchronicity; it takes the content of variables and message parts into account by extending the original message alphabet. Berardi *et al.* (2005) model web services in Propositional Dynamic Logic (PDL) and are interested in generating automated compositions between services. Duan *et al.* (2004) propose a restricted BPEL semantics for which it is possible to automatically generate the composition of tasks. The controllability of a business process has also been studied (Lohmann *et al.*, 2006); the *operating guidelines* of a process P is the automaton that includes as its subgraphs all the possible controllers of P. Nakajima (2004) proposes techniques to extract a behavioural specification from a BPEL process and to verify it with model checking techniques.

Other works also present automated tools for the validation of the properties. Turner (2005) formalizes BPEL web service workflows using a language called CHISEL which is then transformed into LOTOS for automated validation. Multi-agent web services have been modelled using a custom protocol language called MAP which is then translated into SPIN models and model-checked (Walton, 2004). A process algebra approach is used by Brogi *et al.* (2004) to model web service choreographies using the Calculus of Communicating Systems (CCS). Pistore *et al.* (2004b) use a formal language called

Tropos and validate properties in NuSMV (Cimatti *et al.*, 2002). Finally, model checking of LTL formulæ expressed in Promela on BPEL specifications is attempted using SPIN (Holzmann, 2003) in a recent paper from Fu *et al.* (2004a). The approach is extended in a subsequent paper (Fu *et al.*, 2004c) and constitutes the basis of the Web Service Analysis Tool (WSAT). VERBUS (Arias-Fisteus *et al.*, 2005) is another tool that translates a web service workflow into a finite-state structure. Finally, (Johnson *et al.*, 2004) studies the two-phase commit protocol and models it using the Temporal Logic of Actions (TLA^+).

Although these works take data into account when modelling the web services' interactions, this data does not play a role when expressing the properties. The temporal formulæ are still propositional. Actual data content can be referred to, but only statically by extending the original message alphabet. This is shown by the properties in Figure 6.1b. It is now possible to state that when "a" contains 0, then "b" is sent with value 9, since 0 and 9 are fixed constants: a(0) and b(9) are simply modelled as two new message names. It is not possible, however, to compare the values inside two different messages except by explicitly stating their value; therefore, one cannot say "for all x, the value inside message "a" is x, and later the value inside message "c" is $x + 1$" without resorting to explicitly name each possible static value.

6.2.3 Data-aware Workflows, Data-aware Properties

A further extension with respect to expressiveness of properties is to model the transfer and transformation of data inside the control flow of the modelled service, but also to allow quantification on data inside temporal properties. We call these properties "data-aware" to indicate that the actual message content can be known and fully accessed by the temporal formulæ. Figure 6.1c illustrates this. Knowledge about the internal workflow generally remains unchanged with respect to the previous category. However, the properties can now fully express the constraint between messages "a" and "c": when

"c" is sent, it contains the value of "a" incremented by 1.

We are only aware of a limited number of works that address this question. In Deutsch's work (Deutsch *et al.*, 2004; Deutsch *et al.*, 2006), extensions to the temporal logics CTL and LTL, respectively called CTL-FO and LTL-FO, are introduced. These logics include a form of first-order quantification on data. The model presented is very rich: it contains a *database* represented as a variable set of first-order predicates; however this richness is achieved at the price of complexity. The problem of model checking a CTL-FO formula φ on a web service $\mathcal{W}$ (as defined in Deutsch *et al.* (2004)) is undecidable. The problem of model checking a formula φ without any quantification is in CO-NEXPTIME if the formula is propositional CTL, and in EXPSPACE if the formula is propositional CTL*.

We show in this paper how a simpler modelling of the services, coupled with a more expressive logic than CTL-FO, called CTL-FO$^+$, is sufficient for model checking important data-aware properties in real-world scenarios. Theorem 6.2 will demonstrate that CTL-FO$^+$ model checking is PSPACE-complete, a considerably lower complexity.

The Artifact Behavioral Specification Language (ABSL) is another extension of CTL that includes a form of first-order quantification (Gerede et Su, 2007). However, ABSL is developed in a context of artifact-centric business processes and is suited to express properties of *intra*-artifact behaviours, not *inter*-message constraints; the optimality of the ABSL model checking algorithms also remains to be shown.

Another work of interest is Venzke's (2004), which defines specifications using XQuery on traces (SXQT). In this work, the SOAP messages exchanged by services are aligned into a large XML sequence. XQuery can then be used to refer to and compare complex elements of specific messages along the trace in a powerful manner; temporal operators are translated into specific XQuery expressions. However, this approach allows the validation of one specific trace at a time and does not constitute a complete model checking

of the service workflow itself.

6.3 A Web Service Scenario

To measure the importance of data-awareness in web service workflow validation, we introduce a representative real-world scenario. We study this scenario from the angle of data constraints and show that data-aware properties arise naturally and are essential to correctly specify and validate.

6.3.1 UCLP Web Service Architecture

To make a more efficient and flexible use of network resources, a growing trend is to offer users and applications services that can be reserved and composed according to specific needs. This is particularly visible in the GRID initiative which offers computing, data ware-housing and transmission resources for high-performance applications.

The web service paradigm is an ideal setting for such service-oriented networks. Based on this observation, the User-Controlled Lightpath (UCLP) research project (Boutaba *et al.*, 2004) develops an environment that allows end users to self provision and dynamically reconfigure optical networks resources within a single domain or across multiple independent management domains. To this end, network resources from a specific provider are virtualized and exposed to the end user as instances of *web services* that implement functionalities related to lightpath manipulation; such services are called Lightpath Objects (LPOs).

Simply put, a lightpath is a point-to-point, high-speed optical link. We concentrate on two main operations provided to manage Lightpath Objects.

LPO Concatenation. In order to build an end-to-end link, two adjacent LPOs

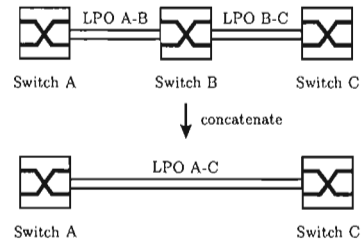


Figure 6.2: The result of the concatenation operation is an LPO that is considered as one single link.

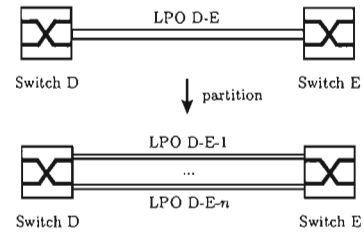


Figure 6.3: The partition operation splits an LPO into fragments of smaller bandwidth.

can be *concatenated*, as is exemplified in Figure 6.2. The result of the concatenation operation is an LPO that is considered as one single link.

Since the same traffic flows through all the link's segments, the concatenated LPOs must have the same bandwidth. Furthermore, the concatenation operation gives rise to a new LPO during whose lifetime the original LPOs cannot be used individually in some other operation.

LPO Partition. An LPO's bandwidth can be *partitioned* into links of equal bandwidth. For instance, an OC-3 LPO (155.52 Mbps) can be partitioned into three OC-1 LPOs (51.84 Mbps). This is shown in Figure 6.3.

In order to be partitioned into OC-1 links, an LPO must be of an OC-1's multiple bandwidth. Furthermore, as before, during the partition's life-time the original LPO cannot be used in other operations.

Within the UCLP environment, a customer can use a graphical interface where available network resources are shown to the user, who can operate on them to create the desired connection.

Once the link is finished, the sequence of operations required to create it from the initial resources can be saved as a script and "played back" at a later time at the request of the user to provide him with the desired connection. Under the hood of this graphical engine is a web service environment. Each provider gives access to its resources in an Articulated Private Network (APN) via an LPO-factory web service from which LPOs can be controlled and consumed. Each LPO is identified by a unique ID, and the UCLP operations usually manipulate these IDs.

The script built by the user in the graphical interface is actually a BPEL process that invokes LPO operations by means of XML messages like for any other web service interaction. The corresponding BPEL operation for LPO concatenation takes as input an array of LPOs to concatenate. A simplified version of the concatenateRequest message structure is shown below:

<message>
  <operation>concatenateRequest</operation>
  <LPO-ID>i_1</LPO-ID>
  <LPO-ID>i_2</LPO-ID>
  ...
  <LPO-ID>i_n</LPO-ID>
</message>

The response from such an operation is the following:

<message>
<operation>concatenateResponse</operation>
<LPO-ID>i</LPO-ID>
</message>

Similarly, the corresponding BPEL operation for LPO partition takes as input the reference to an LPO and returns an array of references to spawned lightpaths, each of the desired bandwidth. A request is therefore of the following form:

<message>
<operation>partitionRequest</operation>
<LPO-ID>i</LPO-ID>
<bandwidth>b</bandwidth>
</message>

where i is the ID of the LPO to partition, b is the bandwidth of the desired fragments. The response from this request is a message of the following form:

<message>
<operation>partitionResponse</operation>
<LPO-ID>i_1</LPO-ID>
<LPO-ID>i_2</LPO-ID>
...
<LPO-ID>i_n</LPO-ID>
</message>

Therefore, the GUI is just a lightpath-oriented rendition of a standard BPEL workflow design environment. Although the interface is adapted for LPO manipulation, the processes contain full-fledged BPEL code that can have loops, conditional branching,

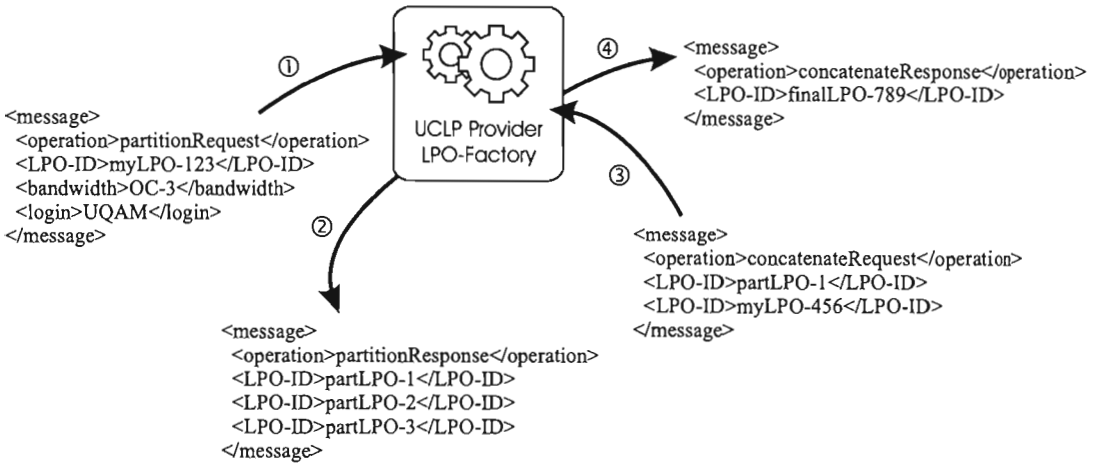


Figure 6.4: Pattern of messages exchanged between a customer BPEL process and a provider LPO-Factory service.

and even interactions with other, non-UCLP web services. Moreover, users can bypass the GUI and program their own scripts involving UCLP resources using the BPEL environment of their choice. Figure 6.4 presents a simple pattern of messages exchanged between a customer BPEL process and a provider LPO-Factory service.

6.3.2 UCLP Service Constraints

While a service oriented network offers much more flexible use of network resources, a major issue is making sure that these resources are correctly used. The consumer of lightpaths from a provider is subject to two kinds of restrictions on its use and composition of LPOs:

- Technical constraints: these constraints arise because of the physical or logical nature of the resources involved in the operations.
- Policy constraints: these constraints arise for non-technical reasons, often dealing with business logic; this may include membership restrictions, QoS requirements

or other reasons.

These constraints are crucial to avoid publicizing erroneous services that could modify in a wrong way the physical resources they represent.

One could argue that it is simpler to let any user-created script allowed to run as long as it properly traps any errors and inconsistencies returned by the web service operations. However, tracing errors and misuses after deployment can be a daunting task. What is really needed is a set of sufficient conditions on web service compositions that can act as guarantees for a workflow to properly interact with a service. Therefore, in order to achieve true interoperability between services of different providers, a *sine qua non* condition is to:

1. advertise the restrictions imposed on the use of a service using some formal language;
2. enforce these restrictions by developing a *general* methodology to check as much as possible that an orchestration script created by some user satisfies the required constraints before it is even allowed to execute.

We now proceed to show that a number of these constraints are data-aware temporal properties.

Let us examine the case of partition as a first example. This operation takes as input the ID of some LPO x and returns new LPOs y, z corresponding to the results of the partition. From there on, it does not make sense to again use x as an argument of a UCLP operation such as concatenate. Although the LPO still physically exists, it has been logically superseded by its fragments y, z. The script could even have applied further operations on y and z, like concatenating them to other LPOs or further partitioning them. In this context, invoking an operation with x is at best semantically

unsound and at worst plain dangerous for the reliability of the whole UCLP environment. We must therefore enforce the following constraint on any UCLP script:

UCLP Service Constraint 6.1. *Any LPO ID appearing in any partition request must be different from any LPO ID appearing in any future concatenate request.*

In that sentence, the first and third occurrences of the word "any" indicate a quantification over message content, while the second and fourth occurrences represent a quantification over messages in an execution sequence: data and temporal modalities are intertwined and the constraint is indeed data-aware.

A second constraint involves the concatenate operation. As has been said earlier, concatenated LPOs must have identical bandwidths. This can be formulated in the following way:

UCLP Service Constraint 6.2. *Every LPO occurring as an input of the concatenate operation must be of the same bandwidth.*

Constraints can also link together invocations of different operations. For example, the semantics of the concatenate operation supposes that the LPOs to be concatenated are adjacent (i.e. they have exactly one common end). Therefore, although it would be syntactically perfectly valid, it does not make sense to take two LPOs originating from the same partition operation and attempt to concatenate them, as these two LPOs are actually the same end-to-end connection. This calls for a third, mixed constraint:

UCLP Service Constraint 6.3. *If two LPOs are the result of the same partition response, they cannot be involved together in a common concatenate request.*

These first three constraints are in the realm of technical restrictions: they arise because of the specific nature of the web services involved.

Business logic can also be a source of data-aware temporal properties. Suppose a small UCLP resource provider wants to limit the management overhead of its LPOs; it might want to expect from all users of its resources to avoid over-partitioning its links by imposing that LPOs can only be partitioned once.

UCLP Service Constraint 6.4. *If an LPO is the result of a partition response, it cannot be involved in a subsequent partition request.*

This constraint clearly has nothing to do with the semantics of the partition operation, but rather with some additional business logic imposed by one particular service provider.

Another constraint related to business logic is the following. Once an LPO is partitioned, the IDs for the spawned LPOs consisting of the fragments of the original LPO are returned to the user of the script. Because of UCLP Service Constraint 1, the original LPO cannot be used by anyone during the lifetime of its fragments. Therefore, partitioning an LPO without using its fragments in a future operation in a script wastes resources by making the original LPO unavailable to anybody without ever using it. A UCLP resource provider might therefore impose the following constraint:

UCLP Service Constraint 6.5. *All LPOs contained in a partition response must be involved in an operation at some point later in the script.*

With this solution, each UCLP service provider can define for its own services custom rules that take into account the specifics of each service. These rules act both as restrictions that prevent a user from using a service incorrectly, and as guarantees that if obeyed, the service should behave as expected.

6.4 A Data-Aware Temporal Logic

The use of temporal logic to express the behaviour of a system is common to many related works of web service workflow verification. Temporal logics are commonly used in model checking for describing behavioural properties of systems. However, classical temporal formalisms are propositional, and Section 6.2 has shown how these languages are only partially appropriate for modelling and validating data-aware properties. In this section, we introduce CTL-FO$^+$, an extension of the classical temporal logic CTL that was first presented in an earlier paper (Hallé *et al.*, 2007b).

6.4.1 Workflow Modelling

The logic will be defined in relation to a suitable model of a web service workflow. In the present context, this suitable representation should model the actual messages that are exchanged. In addition, the values of the internal variables used in the original process, since they can influence the control flow, and hence the messages that can be sent or received, should also be kept.

We start by defining a set Π of parameters and a set Ω of values that are used to represent the content of messages. We define a special symbol # that stands for "no-value". Couples of parameters and values form a message element:

Definition 6.1 (Message elements). *The set of defined message elements is* $\mathcal{E}_d = \Pi \times (\Omega \cup \{\#\})$. *We also consider the set of undefined message elements, which is the singleton* $\mathcal{E}_u = \{(\#, \#)\}$. *A* message element *is a member of* $\mathcal{E} = \mathcal{E}_d \cup \mathcal{E}_u$.

The concept of message element closely parallels the structure of a (flat) XML message. The parameters stand for the tag names, while the values represent the data inside the tag. For that reason, the definition of a message element excludes the possibility

that a value has no corresponding parameter. In the UCLP example, "operation", "bandwidth" and "LPO-ID" are examples of parameters; "concatenateResponse", "myLPO-123" and "OC-3" are examples of values. A message is simply an ordered sequence of message elements:

Definition 6.2 (k-messages). *Let k be a positive integer, and for $i < k$, define $D_i = \{(e_1, \ldots, e_k) : e_i \in \mathcal{E}_u \wedge e_{i+1} \in \mathcal{E}_d\}$. The set of k-messages is defined as*

$$M_k = \mathcal{E}^k \setminus \left(\bigcup_{i=1}^{k-1} D_i \right)$$

Note that this representation does not allow nested tags, and imposes an upper bound k on the number of elements inside a single message. Empty elements are simply ignored; we impose the restriction that all undefined elements be grouped at the end of the k-uple, and therefore one XML message maps to exactly one message of M_k.

A workflow messaging model is simply a standard Kripke structure whose states represent the values of each of the internal variables and the message that is being sent or received in that state. That message can be the empty k-message $((\#, \#), \ldots, (\#, \#))$, indicating that no message is neither received nor sent in that particular state of the model.

Definition 6.3 (Workflow messaging model). *Let k be a positive integer, Π be a set of parameter names, Ω be a set of value names, $\mathcal{P} = \{p_1, \ldots, p_k\}$ a set of parameter variables, $\mathcal{V} = \{v_1, \ldots, v_k\}$ a set of value variables, $\mathcal{I}$ be a set of internal variables. A* workflow messaging model *is a Kripke structure $\mathcal{M}_k = (S, I, R, L)$ such that:*

- *S is a set of states*
- *$I \subseteq S$ is a set of initial states*
- *$R \subseteq S^2$ is a transition relation over the states*
- *$L = (S \times (\mathcal{P} \cup \mathcal{V} \cup \mathcal{I})) \rightarrow (\Pi \cup \Omega)$ is a labelling function such that for every $s \in S$,*

$((L(s,p_1),L(s,v_1)),\ \ldots,\ (L(s,p_k),L(v_k)))$ *is a* k*-message*

We further suppose that L *uniquely identifies every state; that is, there does not exist distinct states* $s_0, s_1 \in S$ *such that* $L(s_0,\alpha) = L(s_1,\alpha)$ *for every* $\alpha \in \mathcal{P} \cup \mathcal{V} \cup \mathcal{I}$.

A path $\pi = s_0 s_1 \ldots$ is a sequence of states in S such that $(s_i, s_{i+1}) \in R$ for every $i \geq 0$. A workflow messaging model can be seen as a generalized construction of a classical *Moore machine* (Hopcroft *et al.*, 2000), where the symbol to be emitted in a state is replaced with the k-message encoded by the values of state variables of $\mathcal{P}$ and $\mathcal{V}$. Any path in the system corresponds to a possible sequence of messages in a service interaction. Properties about message sequences become properties on sequences of states that can then be expressed using temporal logics.

The translation of a business process expressed in a language like BPEL into a workflow messaging model is beyond the scope of this paper; we assume it is given. There exist numerous works providing a formal semantics of BPEL to this end (Ouyang *et al.*, 2007; Lucchi et Mazzara, 2007), and some others describe the transformation of a transition system with variables and internal actions into a "classical" system (Constant *et al.*, 2007).

6.4.2 Syntax and Semantics of CTL-FO$^+$

The Computation Tree Logic with Full First-order Quantification (CTL-FO$^+$) is an extension of the well-known temporal logic CTL (Clarke *et al.*, 2000). CTL and a related logic called LTL are the most commonly used languages for describing sequentialities in finite-state systems. All major model checking tools, such as SPIN (Holzmann, 2003) and NuSMV (Cimatti *et al.*, 2002), verify temporal formulæ expressed in one of these logics. The reader is referred to a book by Clarke *et al.* (Clarke *et al.*, 2000) for a deeper coverage of CTL and other temporal logics. CTL-FO$^+$ is aimed at describing

sequentialities in a finite-state system while allowing full quantification over data.

Formulæ are built from Boolean variables and the constants *true* and *false* using the classical connectors: $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implies) and $\neg$ (not). CTL-FO$^+$ further provides *temporal operators* that can be used on top of traditional propositional logic formulæ to specify the temporal conditions.

We briefly recall these operators, which are taken directly from CTL. They can be divided into two classes. The first class is composed of *universal* operators that assert properties about all executions starting from the current state. The first of these operators is **AG**, which means "globally". For example, the formula $\mathbf{AG}\,\varphi$ means that formula φ is true in every state of every execution starting at the current state. The operator **AF** means "eventually"; the formula $\mathbf{AF}\,\varphi$ is true whenever for all executions, φ holds for some future state. The operator **AX** means "next"; it is true whenever φ holds in any possible next state of the current state. Finally, the **AU** operator means "until"; the formula $\mathbf{A}\,\varphi\,\mathbf{U}\,\psi$ is true if, in any execution sequence, φ holds for all states until ψ holds.

The second class of operators are called *existential* and are designated by **EG**, **EF**, **EX** and **EU**; they are defined in the same way as their universal equivalents, except that the condition holds only for some instead of all possible sequences.

We extend the expressiveness of the traditional CTL by adding first-order quantification. The resulting language has the following formal syntax and semantics.

Definition 6.4 (Syntax). *The language CTL-FO$^+$ (Computation Tree Logic with Full First-order Quantification) is obtained by closing CTL under the following construction rules:*

1. *If x and y are variables or constants, then $x = y$ is a CTL-FO$^+$ formula;*

2. *If* φ *and* ψ *are CTL-FO*$^+$ *formulæ, then* $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, $\mathbf{AG}\,\varphi$, $\mathbf{EG}\,\varphi$, $\mathbf{AF}\,\varphi$, $\mathbf{EF}\,\varphi$, $\mathbf{AX}\,\varphi$, $\mathbf{EX}\,\varphi$, $\mathbf{A}\,\varphi\,\mathbf{U}\,\psi$, $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$, *are CTL-FO*$^+$ *formulæ;*
3. *If* φ *is a CTL-FO*$^+$ *formula,* x_i *is a free variable in* φ, $p \in \Pi$ *is a parameter name, then* $\exists_p x_i : \varphi$ *and* $\forall_p x_i : \varphi$ *are CTL-FO*$^+$ *formulæ.*

Definition 6.5 (Semantics). *Let* $\mathcal{M}_k$ *be a workflow messaging model, and* $s_0 \in S$ *be a state. For* $p \in \Pi$, *let* $Dom_{s_0}(p) = \{L(s_0, v_i) : L(s_0, p_i) = p, 1 \leq i \leq k\}$, *and* c_1 *and* c_2 *be constants. Let* $X = \{x_1, \ldots, x_n\}$ *be the set of variables in* φ *and* $\nu : X \rightarrow \Omega \cup \{\#\}$ *a valuation that maps each variable to a possible value. We denote by* $\nu[a/x_j]$ *the valuation that agrees with* ν *on every* $x_i \in X$ *with the exception of* x_j *for which it returns* a. *We say the triplet* $\mathcal{M}_k, s_0, \nu$ *satisfies the CTL-FO*$^+$ *formula* φ, *and write* $\mathcal{M}_k, s_0, \nu \models \varphi$ *if and only if it satisfies the rules given in Table 6.1.*

By extension, we write $\mathcal{M}_k \models \varphi$ *if all initial states* $s_0 \in I$ *of* $\mathcal{M}_k$ *are such that, given the empty valuation* $\nu(x) = \#$ *for all* $x \in X$, *we have* $\mathcal{M}_k, s_0, \nu \models \varphi$.

The set of operators $\neg$, $\vee$, $\mathbf{AF}$, $\mathbf{EX}$, $\mathbf{EU}$, and $\exists$ is called an *adequate* set of connectives in that all other operators can be derived from a combination of them with the following identities: $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$, $\mathbf{EG}\,\varphi \equiv \neg\mathbf{AF}\,\neg\varphi$, $\mathbf{EF}\,\varphi \equiv \mathbf{E}\,[true\,\mathbf{U}\,\varphi]$, $\mathbf{AX}\,\varphi \equiv \neg\mathbf{EX}\,\neg\varphi$, $\mathbf{AG}\,\varphi \equiv \neg\mathbf{E}\,[true\,\mathbf{U}\,\neg\varphi]$, $\mathbf{A}\,[\varphi\,\mathbf{U}\,\psi] \equiv \neg(\mathbf{E}\,[(\neg\varphi)\,\mathbf{U}\,\neg(\varphi \vee \psi)] \vee \mathbf{EG}\,\neg\psi)$, $\forall_p x_i : \varphi \equiv \neg(\exists_p x_i : \neg\varphi)$. This result is classical (Schnoebelen, 2003).

Without loss of generality, we assume that CTL-FO$^+$ formulæ φ with n quantified variables are *well-named*: each variable is quantified only once and in the order $x_1, x_2, \ldots, x_n$. Every CTL-FO$^+$ formula can be transformed by a simple renaming of its variables to a well-named formula. Then, the valuations ν used in the previous semantics can be restricted to *ordered* valuations, which define variables progressively in the order $x_1, x_2, \ldots, x_n$. An ordered t-valuation is an ordered valuation for which exactly the first t variables $x_1, x_2, \ldots, x_t$ have been defined. We then define $p_i \in \Pi$ as the parameter name appearing in the quantification of variable x_i in φ.

$\mathcal{M}_k, s_0, \nu \models c_1 = c_2$	$\Leftrightarrow$	c_1 is equal to c_2
$\mathcal{M}_k, s_0, \nu \models x_i = c_1$	$\Leftrightarrow$	$\nu(x_i)$ is equal to c_1
$\mathcal{M}_k, s_0, \nu \models x_i = x_j$	$\Leftrightarrow$	$\nu(x_i)$ is equal to $\nu(x_j)$
$\mathcal{M}_k, s_0, \nu \models \neg\varphi$	$\Leftrightarrow$	$\mathcal{M}_k, s_0, \nu \not\models \varphi$
$\mathcal{M}_k, s_0, \nu \models \varphi \vee \psi$	$\Leftrightarrow$	$\mathcal{M}_k, s_0, \nu \models \varphi$ or $\mathcal{M}_k, s, \nu \models \psi$
$\mathcal{M}_k, s_0, \nu \models \mathbf{AF}\,\varphi$	$\Leftrightarrow$	for all $\pi = s_0 s_2 \ldots$, $\mathcal{M}_k, s_i, \nu \models \varphi$ for some i
$\mathcal{M}_k, s_0, \nu \models \mathbf{EX}\,\varphi$	$\Leftrightarrow$	there exists $\pi = s_0 s_1 s_2 \ldots$ such that $\mathcal{M}_k, s_1, \nu \models \varphi$
$\mathcal{M}_k, s_0, \nu \models \mathbf{E}\,\varphi\,\mathbf{U}\,\psi$	$\Leftrightarrow$	there exists $\pi = s_0 s_1 s_2 \ldots$ such that $\mathcal{M}_k, s_j, \nu \models \psi$ for some j and $\mathcal{M}_k, s_i, \nu \models \varphi$ for $i < j$
$\mathcal{M}_k, s_0, \nu \models \exists_p x_i : \varphi$	$\Leftrightarrow$	there exists $a \in Dom_{s_0}(p)$ such that $\mathcal{M}_k, s_0, \nu[a/x_i] \models \varphi$

Table 6.1: Formal semantics of CTL-FO$^+$.

CTL-FO$^+$ is reminiscent of a logic called EQCTL (Kupferman, 1995) that extends CTL by allowing existential quantification over *state variables*. EQCTL is not closed under negation; therefore, universal quantification cannot be obtained. CTL-FO$^+$ quantifies over *values* and is closer to true first-order quantification. Furthermore, the model checking of EQCTL is NP-complete, while we show later that model checking in CTL-FO$^+$ is in a higher complexity class. A closer work is QCTL (Rensink, 2006) which extends CTL by including first-order quantification and monadic second-order quantification over arbitrary *algebraic data structures*; such expressiveness is not required in our case. Finally, CTL-FO$^+$ can freely mix temporal and data quantification without restriction. This is an extension over CTL-FO (Deutsch *et al.*, 2004), which does not allow formulæ containing temporal operators to be existentially quantified. The inclusion could be strict: for example, it is not known whether CTL-FO can express UCLP Constraint 2, while we shall see later that CTL-FO$^+$ does.

6.4.3 Formalizing Web Service Properties

The values of variables appearing in a CTL-FO$^+$ formula are quantified according to specific parts of the XML message that is received or sent in the current state of the system. A quantifier like $\forall_{\text{LPO-ID}}x$ therefore means "for all values x of elements named LPO-ID in the current message". This form of quantification is sufficient, since when referring to message data, it is never necessary to quantify over all values of all elements in the message; rather, we normally want to quantify for all values of a specific element name. As an example, UCLP Service Constraint 1 becomes the following CTL-FO$^+$ formula:

UCLP Formal Service Constraint 6.1.

$$\begin{aligned}
&\mathbf{AG}\,(\forall_{operation}\, x_1 : x_1 = concatenateRequest \rightarrow \\
&\forall_{LPO\text{-}ID}\, x_2 : \mathbf{AX}\ \mathbf{AG}\ (\forall_{operation}\, x_3 : \\
&\quad (x_3 = partitionRequest \vee x_3 = concatenateRequest) \\
&\quad \rightarrow \forall_{LPO\text{-}ID}\, x_4 : x_2 \neq x_4))
\end{aligned}$$

This formula states that at any time in any execution of the script, if the operation x_1 of the message is concatenateRequest, then for every LPO ID x_2 appearing in this message, we have that for every future message whose operation element value x_3 is partitionRequest or concatenateRequest, any value x_4 for its LPO ID is different from x_2. In other words, once an LPO has been concatenated, no further partition or concatenation involves this LPO, which is indeed equivalent to UCLP Service Constraint 1. A similar formula constrains the use of the results from a partitionRequest.

In the previous example, one can see how the quantification of the variables in CTL-FO$^+$ depends on the message part. The variable x_1 is defined on `operation` elements; its domain is the set of all values appearing inside such elements somewhere in the process. In contrast, the variable x_2 is defined on elements of name `LPO-ID`; its domain is the set of all possible values that can occur in this part the current message. Consider for example the sequence formed of a partitionRequest and a partitionResponse XML messages as shown in Section 6.3.1. In the first message of the pattern, variable x_1 can only take the value "partitionRequest", since x_1 is defined in UCLP Formal Service Constraint 1 as a variable on root element names. In the same way, x_2 can only take the value i, since x_2 is defined as a variable on elements of name LPO-ID. In the second message, x_2 can take the values $i_1, \ldots, i_n$.

UCLP Service Constraint 2 can be enforced in various ways; in the present context,

we limit ourselves to assuming that all original LPOs have the same bandwidth at the beginning, and check for example that no attempt is made to concatenate an LPO resulting from a partition operation with an LPO that has not been partitioned. We obtain the following CTL-FO$^+$ formula:

UCLP Formal Service Constraint 6.2.

$$\begin{aligned}
&\mathbf{AG}\,(\forall_{operation}\, x_1 : x_1 = \mathit{partitionResponse} \rightarrow \\
&\forall_{LPO\text{-}ID}\, x_2 : \neg\mathbf{EF}\,(\exists_{operation} x_3 : \\
&(x_3 \neq \mathit{partitionResponse} \wedge \\
&(\exists_{LPO\text{-}ID} x_4 : \mathbf{EF} \\
&(\exists_{operation} x_5\, \exists_{LPO\text{-}ID} x_6\, \exists_{LPO\text{-}ID} x_7 : \\
&x_5 = \mathit{concatenateRequest} \wedge \\
&x_2 = x_6 \wedge x_4 = x_7)))))
\end{aligned}$$

This formula states that at any time in any execution of the script, if a message received is a partition response, then for every LPO ID x_2 appearing in this message, at no point in the future can an LPO x_4 coming from some operation other than a partition response can later appear with x_2 in a concatenate request. In other words, a partitioned LPO can only be concatenated with other partitioned LPOs, which is equivalent to UCLP Service Constraint 2. Remark that since XML messages are considered flat, the choice of values for x_6 and x_7 does not depend on the choice of x_5; therefore, the order in which x_5, x_6 and x_7 are picked from the message is irrelevant.

UCLP Service Constraints 3 to 5 can be obtained in the same way. We include them for the sake of completeness:

UCLP Formal Service Constraint 6.3.

$$\begin{aligned}
&\mathbf{AG}\,(\forall_{operation}\, x_1 : x_1 = partitionResponse \rightarrow \\
&\forall_{LPO\text{-}ID}\, x_2\, \forall_{LPO\text{-}ID}\, x_3 : \mathbf{AX}\ \mathbf{AG} \\
&(\forall_{operation}\, x_4\, \forall_{LPO\text{-}ID}\, x_5\, \forall_{LPO\text{-}ID}\, x_6 \\
&x_4 = concatenateRequest \rightarrow \\
&(x_2 \neq x_5 \wedge x_3 \neq x_6)))
\end{aligned}$$

UCLP Formal Service Constraint 6.4.

$$\begin{aligned}
&\mathbf{AG}\,(\forall_{operation}\, x_1 : x_1 = partitionResponse \rightarrow \\
&\forall_{LPO\text{-}ID}\, x_2 : \mathbf{AX}\ \mathbf{AG}\ (\forall_{operation}\, x_3 : \\
&x_3 = partitionRequest \\
&\rightarrow \forall_{LPO\text{-}ID}\, x_4 : x_2 \neq x_4))
\end{aligned}$$

UCLP Formal Service Constraint 6.5.

$$\begin{aligned}
&\mathbf{AG}\,(\forall_{operation}\, x_1 : x_1 = partitionResponse \rightarrow \\
&\forall_{LPO\text{-}ID}\, x_2 : \mathbf{AX}\ \mathbf{AF}\ (\exists_{operation}\, x_3 : \\
&x_3 = concatenateRequest \wedge \\
&\forall_{LPO\text{-}ID}\, x_4 : x_2 = x_4))
\end{aligned}$$

6.5 Validating CTL-FO$^+$ Properties

In this section, we show how CTL-FO$^+$ formulæ can be actually validated on a web service workflow by presenting a complete model checking algorithm. The complexity of this algorithm is then established and discussed. In particular, we show that CTL-FO$^+$ model checking is a problem as tractable as the LTL model checking problem.

Finally, we show that any web service model that uses a data-aware workflow, but only *propositional* properties cannot efficiently simulate data-awareness.

6.5.1 Model Checking CTL-FO$^+$

Our algorithm for model checking CTL-FO$^+$ formulæ is derived from the classical CTL model checking algorithm defined in Huth's book (2000) and is presented in Table 6.2. Given a valuation ν, the procedure CHECK performs structural recursion on the CTL-FO$^+$ formula φ and consists in forming recursively the set of states s such that $\mathcal{M}_k, s, \nu \models \varphi$. If the subformula to check is of the form $\neg\varphi$, $\varphi \wedge \psi$, $\mathbf{AF}\,\varphi$, $\mathbf{EX}\,\varphi$, $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$, the algorithm is identical to the model checking of a CTL formula.

Differences arise when the formula's main operator is an existential quantifier, $\exists x : \varphi(x)$. In such a case, the algorithm successively applies the model checking of $\varphi(a)$ a in the domain of x and returns states which satisfy at least one $\varphi(a)$. Finally, model checking of ground terms is done through equality testing. Depending on the valuation and the terms to be compared, either the entire model satisfies it if the assertion is true, or no state satisfies it if the assertion is false.

A workflow messaging model $\mathcal{M}_k$ satisfies the global CTL-FO$^+$ formula φ if and only if all its initial states are in the set returned by CHECK(φ, ν), with ν the empty valuation.

Theorem 6.1. CHECK *is sound and complete.*

Proof. The proof is by structural induction on the formula φ. For each case, we need to show that for every state $s \in S$ in a workflow messaging model $\mathcal{M}_k$ and every valuation ν, $\mathcal{M}_k, s, \nu \models \varphi$ if and only if $s \in$ CHECK(φ, ν).

Base case: suppose φ is of the form $c_1 = c_2$ for c_1 and c_2 two constants. If c_1 and

```
Procedure CHECK(c_1 = c_2, ν)
  If c_1 = c_2
    Return S
  Else
    Return ∅
  End if
End procedure

Procedure CHECK(x_i = c_1, ν)
  If ν(x_i) = c_1
    Return S
  Else
    Return ∅
  End if
End procedure

Procedure CHECK(x_i = x_j, ν)
  If ν(x_i) = ν(x_j)
    Return S
  Else
    Return ∅
  End if
End procedure

Procedure CHECK(¬φ, ν)
  Return S \ CHECK(φ, ν)
End procedure

Procedure CHECK(φ ∨ ψ, ν)
  Return CHECK(φ, ν) ∪ CHECK(ψ, ν)
End procedure

Procedure CHECK(∃_p x : φ, ν)
  N := ∅
  For each s ∈ S
    For each a ∈ Dom_s(p)
      If s ∈ CHECK(φ, ν[a/x])
        N := N ∪ {s}
      End if
    End for
  End for
  Return N
End procedure

Procedure CHECK(EX φ, ν)
  M := CHECK(φ, ν)
  N := ∅
  For each (s_1, s_2) ∈ R
    If s_2 ∈ M
      N := N ∪ {s_1}
    End if
  End for
  Return N
End procedure

Procedure CHECK(E φ U ψ, ν)
  M := CHECK(ψ, ν)
  Do
    N := ∅
    For each (s_1, s_2) ∈ R
      If s_2 ∈ M
        N := N ∪ {s_1}
      End if
    End for
    M := M ∪ N
  Loop until N = ∅
  Return M
End procedure

Procedure CHECK(AF φ)
  M := CHECK(φ)
  Do
    N := ∅
    For each s_1 ∈ S
      flag := true
      For each s_2 ∈ S
        If (s_1, s_2) ∈ R and s_2 ∉ M
          flag := false
        End if
      End for
      If flag = true
        N := N ∪ {s_1}
      End if
    End for
    M := M ∪ N
  Loop until N = ∅
  Return M
End procedure
```

Table 6.2: The recursive model checking procedure for CTL-FO$^+$.

c_2 are the same value, then φ is a tautology and therefore every state $s \in S$ is such that $\mathcal{M}_k, s, \nu \models c_1 = c_2$, but then $s \in \textsc{Check}(\varphi, \nu)$ since the procedure returns S in such a case. Conversely, if c_1 and c_2 are different, then φ is a contradiction and no state $s \in S$ is such that $\mathcal{M}_k, s, \nu \models c_1 = c_2$; since $\textsc{Check}(\varphi, \nu)$ returns $\emptyset$, the equivalence is respected. The two other ground equality testings, $x_i = x_j$ and $x_i = c_1$, are proven in the same way, using the application of valuation ν on variables.

Induction step: suppose that CHECK is sound and complete for every formula of length less than ℓ. Let φ be a formula of length ℓ. For the cases where φ is of the form $\neg\varphi'$, $\varphi' \vee \psi$, $\mathbf{EX}\,\varphi'$, $\mathbf{E}\,\varphi'\,\mathbf{U}\,\psi$ and $\mathbf{AF}\,\varphi'$, the procedure CHECK is identical to the one described in Huth's book (2000) and its soundness and completeness are assumed. The remaining case not covered is that of existential quantification. The lines 4-6 of the procedure $\textsc{Check}(\exists_p x : \varphi, \nu)$ are such that a state s is added to set N if and only if there exists at least one value $a \in Dom_s(p)$ such that $s \in \textsc{Check}(\varphi, \nu[a/x])$. By the induction hypothesis, this is the case if and only if $\mathcal{M}_k, s, \nu[a/x] \models \varphi$. Since this loop is repeated for every $s \in S$, at the end of the procedure we have that $s \in N = \textsc{Check}(\exists_p x : \varphi, \nu)$ if and only if there exists $a \in Dom_s(p)$ such that $\mathcal{M}_k, s, \nu[a/x] \models \varphi$. By Definition 6.5, this in turn is equivalent to $\mathcal{M}_k, s, \nu \models \exists_p x : \varphi$.

□

6.5.2 CTL-FO$^+$ Model Checking is PSPACE-complete

We now establish the complexity of model checking CTL-FO$^+$ formulæ and show that data-aware properties cannot be modelled effectively by propositional properties.

Theorem 6.2. *Let φ be a CTL-FO$^+$ formula and $\mathcal{M}_k$ be a workflow messaging model. Determining whether $\mathcal{M}_k \models \varphi$ is PSPACE-complete.*

Proof. We first show that the model checking problem is PSPACE-hard by reducing the

quantified Boolean formula problem (QBF), known to be PSPACE-complete (Garey et Johnson, 1979), to CTL-FO$^+$ model checking. A quantified Boolean formula φ is of the form $Q^1x_1Q^2x_2\ldots Q^nx_n\varphi$, where Q^i is either the existential ($\exists$) or the universal ($\forall$) quantifier and the x_i are Boolean variables (their domain is $\{0,1\}$). Consider the workflow messaging model $\mathcal{M}_2$ consisting of a single state s (which is also the initial state), a transition relation $\{(s,s)\}$ and where the 2-message in state s is $((p,0),(p,1))$ for some dummy parameter name p. Then, by rewriting the quantifiers Qx_i in the original QBF to $Q^i_px_i$, φ becomes a CTL-FO$^+$ formula where each variable x_i has domain $Dom_s(p) = \{0,1\}$. Therefore, φ is satisfiable if and only if $\mathcal{M}_2 \models \varphi$.

The second step consists in showing that the procedure CHECK is in PSPACE. Each recursive call receives as arguments a subformula bounded by the size of the original formula φ and a valuation ν whose cardinality is fixed. Depending on the case to be considered, each call uses at most two subsets of S during its computation, and returns as output a subset of S. Therefore, the space consumed by each recursive call is linear in the size of both the formula and the structure. Since the number of calls is bounded by the length of the formula, this algorithm is polynomial in the size of the CTL-FO$^+$ formula and the transition system. Remark that the PSPACE class of decision problems only requires polynomial use of *memory space*; the algorithm is clearly exponential with respect to time. □

The PSPACE-completeness result places CTL-FO$^+$ model checking for finite domains in the same complexity class as model checking of an LTL formula (Clarke *et al.*, 2000). LTL is a temporal logic widely used in industry, for example in conjunction with the SPIN model checker, and many works with *propositional* properties mentioned in Section 6.2.2 use LTL as their language for expressing constraints on message sequences. Therefore, although CTL-FO$^+$ allows to fully access the data content of the messages, its model checking procedure has an equivalent complexity to many other existing solutions

that do not provide data-aware temporal capabilities.

6.5.3 Simulating Data-awareness with Propositional Properties

Studying the complexity of the CTL-FO$^+$ model checking algorithm can teach us more. Since the domain for each variable is finite, it is possible to use the semantics of Definition 6.5 and convert each quantifier into a conjunction or a disjunction of a finite number of terms. The resulting expression is a plain CTL formula where all references to data are static; we call this approach *explicit quantification*. In turn, this expansion of a CTL-FO$^+$ formula φ into a propositional CTL formula φ' is exponential in the number of quantifiers, since each quantified subformula must be repeated once for each possible value in the domain. However, the model checking algorithm of a CTL formula in P: it has a worst-case running time linear in the size of the formula to check, and linear in the size of the Kripke structure (Schnoebelen, 2003). Therefore, using CTL model checking on φ' takes exponential time, which is no worse than the runtime of CTL-FO$^+$ model checking on φ.

One might then think that CTL-FO$^+$ is simply CTL with an additional level of syntactic sugar, and that data-aware workflows with *propositional* properties, as described in Section 6.2.2, are already sufficient to model any data-aware property by simply extending the message alphabet. However, this is not the case; the following theorem shows that it is highly unlikely that any polynomial reduction of CTL-FO$^+$ to CTL exists.

Theorem 6.3. *If there exists a polynomial reduction of CTL-FO$^+$ model checking to CTL model checking, then $P = NP$.*

Proof. A polynomial reduction of CTL-FO$^+$ model checking to CTL model checking entails that for every workflow messaging model $\mathcal{M}_k$ and every CTL-FO$^+$ formula φ,

there exists a Kripke structure K' and a CTL formula φ' such that $\mathcal{M}_k \models \varphi$ if and only if $K' \models \varphi'$. Moreover, the size of K' and φ are respectively polynomial in the sizes of $\mathcal{M}_k$ and φ. Since CTL-FO$^+$ model checking is PSPACE-complete and CTL model checking is in P, we have PSPACE $\subseteq$ P. The result follows since P $\subseteq$ NP $\subseteq$ PSPACE. $\square$

Therefore, unless P = NP, any attempt at using data-aware workflows with propositional properties to model data-aware properties will either blow the size of the formulæ or the size of the model by an exponential factor; the space used is no longer polynomial and therefore the translation is not optimal. In other words, CTL-FO$^+$ is exponentially more succinct than any propositional modelling of data-awareness.

Furthermore, with explicit quantification, the translation of constraints into temporal logic becomes tightly coupled with the actual script on which it has to be checked. This is because the translation of the quantifiers shown depends on the values occurring in the script. It is, however, unrealistic that a UCLP resource provider advertises its constraints in such a manner: one would have to know in advance all possible LPO values occurring in scripts prepared by third-parties to include them in the large disjunction. Moreover, standard model checkers such as NuSMV (Cimatti *et al.*, 2002) can easily handle systems with very large state spaces and reasonably short temporal formulæ, but are far less efficient for checking exponentially long formulæ.

6.6 An Efficient Reduction of CTL-FO$^+$ to CTL

Theorem 6.3 indicates that, in fact, *any* attempt to use standard model checkers to validate data-aware workflow properties is "doomed" to an exponential blow-up of the original problem, and not only the explicit quantification method suggested above.

Nevertheless, in this section, we show an alternate translation of the CTL-FO$^+$

model checking problem to CTL. In this approach, the original CTL-FO$^+$ formula is transformed into a CTL formula, but the original workflow messaging model is also transformed by adding new state variables. These additional variables are used to "freeze" the value of a state variable at some point in the execution for future reference; consequently the transformation technique used is called "freeze quantification". It has been originally developed by Alur and Henzinger (1994) for timed transitions systems and further studied by Demri *et al.* (2005). We previously used this technique to reduce a subset of XPath to CTL (Hallé *et al.*, 2006b).

We proceed in two steps: first, we show how to convert a workflow messaging model $\mathcal{M}_k$ and a CTL-FO$^+$ formula φ with n variables into a *freeze workflow messaging model* $\widehat{\mathcal{M}}_k^n$; then, we show how a CTL-FO$^+$ formula φ can be translated to a CTL formula $\widehat{\varphi}$ and show that φ is true for $\mathcal{M}_k$ if and only if $\widehat{\varphi}$ is true for $\widehat{\mathcal{M}}_k^n$, thereby reducing the problem of CTL-FO$^+$ model checking to CTL model checking.

The number of states in $\widehat{\mathcal{M}}_k^n$ is exponential with respect to the number of states in $\mathcal{M}_k$ and hence the reduction is still in EXPTIME; however, the original CTL-FO$^+$ formula φ is transformed into a CTL formula whose size is *linear* with respect to φ. We shall see in Section 6.7 that, for this reason, this reduction performs much better than the explicit quantification approach for complex formulæ. The presentation given below is a refinement over the basic ideas first introduced in an earlier paper (Hallé *et al.*, 2007c).

6.6.1 Transforming a Kripke Structure

Let $\mathcal{M}_k = (S, I, R, L)$ be a workflow messaging model defined over parameters Π and values Ω, with sets of state variables $\mathcal{P}$, $\mathcal{V}$ and $\mathcal{I}$ defined as previously. We construct a *freeze* workflow messaging model $\widehat{\mathcal{M}}_k^n = (\widehat{S}, \widehat{I}, \widehat{R}, \widehat{L})$ by including an additional set of state variables $\mathcal{F} = \{\widehat{\nu}_1, \ldots, \widehat{\nu}_n\}$, called the "freeze" variables, since they are intended

to capture the value of some part of a message at a given point in the execution of the workflow. Intuitively, the $\widehat{\nu}_i$ will be used to represent inside the workflow messaging model the possible ordered valuations ν of the variables $x_1, \ldots, x_n$ in the original CTL-FO$^+$ formula; more precisely, in every state, $\nu(x_i) = \widehat{\nu}_i$.

The labelling function $\widehat{L}$ is extended to the freeze variables and is defined as $\widehat{L} : (\widehat{S} \times (\mathcal{P} \cup \mathcal{V} \cup \mathcal{I} \cup \mathcal{F})) \rightarrow (\Pi \cup \Omega)$. For $0 \leq t \leq n$, we define the set $\Omega_t^n \subset (\Omega \cup \{\#\})^n$ such that $(\widehat{L}(\widehat{\nu}_1), \ldots, \widehat{L}(\widehat{\nu}_n)) \in \Omega_t^n$ if and only if $\widehat{\nu}_i \neq \#$ for $1 \leq i \leq t$ and $\widehat{\nu}_i = \#$ otherwise. The set Ω_t^n contains all possible ordered t-valuations.

The set of system states $\widehat{S}$ and the behaviour of the labelling function $\widehat{L}$ on this set are defined as follows:

Definition 6.6 (Set of system states, labelling). *Let $s \in S$ be a state of $\mathcal{M}_k$ and t be an integer such that $0 \leq t \leq n$. Then $\widehat{s} \in \widehat{S}$ is a state of $\widehat{\mathcal{M}}_k^n$ if and only if:*

- $L(s, \alpha) = \widehat{L}(\widehat{s}, \alpha)$ *for every* $\alpha \in \mathcal{P} \cup \mathcal{V} \cup \mathcal{I}$
- $(\widehat{L}(\widehat{s}, \widehat{\nu}_1), \ldots, \widehat{L}(\widehat{s}, \widehat{\nu}_n)) \in \Omega_t^n$.

This definition entails that the relation between S and $\widehat{S}$ is surjective: for every state $s \in S$ there exists multiple "copies" $\widehat{s}_\nu \in \widehat{S}$ that agree on the labelling function for s for every state variable of $\mathcal{M}_k$, and for which the freeze variables encode every possible ordered valuation ν of the x_i. Therefore, for every $0 \leq t \leq n$ and every $\nu \in \Omega_t^n$, the sets $\widehat{S}_\nu = \left\{ \widehat{s}_\nu : \widehat{L}(\widehat{s}_\nu, \widehat{\nu}_i) = \nu(x_i), 1 \leq i \leq n \right\}$ form a partition of $\widehat{S}$; each $\widehat{S}_\nu$ is a "copy" of S where the $\widehat{\nu}_i$ encode one specific ordered valuation, ν.

The initial states of $\widehat{\mathcal{M}}_k^n$ are the copies of the initial states of $\mathcal{M}_k$ where the $\widehat{\nu}_i$ encode the empty valuation.

Definition 6.7 (Initial states). *Let $s \in S$ be a state of $\mathcal{M}_k$ and $\widehat{s}_\nu \in \widehat{S}$ be a state of $\widehat{\mathcal{M}}_k^n$ such that $\nu \in \Omega_0^n$. Then $s \in I$ if and only if $\widehat{s}_\nu \in \widehat{I}$.*

The transition relation $\widehat{R}$ is defined as the union of two relations, $\widehat{R}_w$ and $\widehat{R}_f$. The transitions contained in the first part, $\widehat{R}_w$, reproduce in each partition $\widehat{S}_\nu$ the original transition relation R. They are called *workflow transitions.*

Definition 6.8 (Transition relation: workflow transitions). *Let $s, s' \in S$, $\nu \in \Omega_t^n$ for some $0 \leq t \leq n$ and $\widehat{s}_\nu, \widehat{s}'_\nu \in \widehat{S}_\nu$. Then $(s, s') \in R$ if and only if $(\widehat{s}_\nu, \widehat{s}'_\nu) \in \widehat{R}_w$.*

The transitions contained in the second part, $\widehat{R}_f$, simulate the definition of a new variable into the valuation.

Definition 6.9 (Transition relation: freeze transitions). *Let $s \in S$. Let $\nu \in \Omega_t^n$ for some $0 \leq t \leq n$, $\nu' \in \Omega_{t+1}^n$ such that $\nu(x_i) = \nu'(x_i)$ for $1 \leq i \leq t$. Then $(\widehat{s}_\nu, \widehat{s}_{\nu'}) \in \widehat{R}_f$ if and only if $\nu'_{t+1} \in Dom_s(\pi_{t+1})$.*

These are called *freeze transitions*, since the workflow messaging model switches between two copies $\widehat{s}_\nu$, $\widehat{s}_{\nu'}$ of the same original state $s \in S$. Thus, the action of the original workflow messaging model is suspended while a variable $\widehat{\nu}_i$ takes a value. We say that $\widehat{\mathcal{M}}_k^n$ is in a *freezing phase* when it advances to its next state through a freeze transition.

Following the semantics of CTL-FO$^+$, the domain of each freeze variable is dependent on the message part on which it is defined: the definition imposes that in state s, if the variable that takes a value is $\widehat{\nu}_i$, then this value must be from $Dom_s(\pi_{i+1})$.

The actual value assigned to either of these special variables in each state is non-deterministic among all possible values in $Dom_s(\pi_{i+1})$. In addition, each variable may or may not take a value —that is, variables can stay undefined. However, once a variable has taken a definite value, it keeps this value for the remainder of the execution trace. Finally, any number of freeze transitions can be taken before resuming the execution of $\mathcal{M}_k$ by taking again a workflow transition. This entails that any number of variables

can be assigned in a freezing phase, provided that they are assigned in lexicographical order and that their domain is not empty for that state.

Definitions 6.6 to 6.9 completely specify $\widehat{\mathcal{M}}_k^n$ from $\mathcal{M}_k$. It is important to remark that $\widehat{\mathcal{M}}_k^n$ also depends on the CTL-FO$^+$ formula to check, φ, but only in the number of variables and the parameters π_i on which each x_i is quantified.

6.6.2 Converting a CTL-FO$^+$ Formula

Once a workflow messaging model $\mathcal{M}_k$ has been translated into a freeze workflow messaging model $\widehat{\mathcal{M}}_k^n$, the CTL-FO$^+$ formula on $\mathcal{M}_k$ can be translated into a standard CTL formula on $\widehat{\mathcal{M}}_k^n$.

We first define a class of auxiliary formulæ γ_t^n, called the *guards*. Intuitively, γ_t^n describes the fact that the variables $\widehat{\nu}_1, \ldots, \widehat{\nu}_n$ encode an ordered t-valuation in Ω_t^n.

Definition 6.10 (Guard). *Let t,n be positive integers such that $0 \leq t \leq n$. The* guard *γ_t^n is the CTL formula:*

$$\gamma_t^n = \left(\bigwedge_{i=1}^{t} \widehat{\nu}_i \neq \# \right) \wedge \left(\bigwedge_{i=t+1}^{n} \widehat{\nu}_i = \# \right)$$

It can be observed that by definition, we have that γ_t^n holds in a state $\widehat{s} \in \widehat{S}$ if and only if $(\widehat{L}(\widehat{s}, \widehat{\nu}_1), \ldots, \widehat{L}(\widehat{s}, \widehat{\nu}_n)) \in \Omega_t^n$.

We define a linear embedding ω_t^n of CTL-FO$^+$ into CTL formulæ performed by structural induction on the original CTL-FO$^+$ formula φ. In the same way as the semantics of CTL-FO$^+$ depends on the valuation of the variables ν, the translation ω_t^n depends on t, the number of variables whose value is already defined. Therefore, $\omega_t^n(\varphi)$ returns the CTL translation of φ, given that t out of n variables are already defined.

Let φ_1 and φ_2 be CTL-FO$^+$ subformulæ, c_1, c_2 be constants in Ω, t, n be integers defined as above, $p \in \Pi$ be a parameter name and $x_1, \ldots x_n$ be the n quantified variables in the CTL-FO$^+$ formula φ. Translating the Boolean connectives and the ground equality testings is direct.

$$\omega_t^n(c_1 = c_2) \equiv c_1 = c_2 \tag{6.1}$$

$$\omega_t^n(x_i = c_1) \equiv \widehat{\nu}_i = c_1 \tag{6.2}$$

$$\omega_t^n(x_i = x_j) \equiv \widehat{\nu}_i = \widehat{\nu}_j \tag{6.3}$$

$$\omega_t^n(\neg\varphi_1) \equiv \neg\omega_t^n(\varphi_1) \tag{6.4}$$

$$\omega_t^n(\varphi_1 \vee \varphi_2) \equiv \omega_t^n(\varphi_1) \vee \omega_t^n(\varphi_2) \tag{6.5}$$

The translation of the CTL temporal operators requires more work; we explain them one by one. The semantics of the **EX** operator requires that there exists one execution path in $\mathcal{M}_k$ for which the next state fulfils some property. In $\widehat{\mathcal{M}}_k^n$, not all possible execution paths are admissible; remember that in *freeze* transitions $(\widehat{s}_0, \widehat{s}_1) \in \widehat{R}_f$ the states $\widehat{s}_0$ and $\widehat{s}_1$ are two copies of the same original state in $\mathcal{M}_k$, and do not represent an actual progression of the execution of $\mathcal{M}_k$. Therefore, the next states reached through these freeze transitions are not "real" next states of the execution and must be discarded. Only next states reached through workflow transitions $(\widehat{s}_0, \widehat{s}_1) \in \widehat{R}_w$ must be considered. These states can be characterized by the fact that the $\widehat{\nu}_i$ encode a t-valuation which, by Definition 6.9, must be the same as that in $\widehat{s}_0$. Hence, only next states that verify both γ_t^n and $\omega_t^n(\varphi)$ are valid candidates. This yields the following equation:

$$\omega_t^n(\mathbf{EX}\,\varphi_1) \equiv \mathbf{EX}\,(\gamma_t^n \wedge \omega_t^n(\varphi_1)) \tag{6.6}$$

A similar adaptation must be done to preserve the semantics of the **AF** operator. In the original semantics, $\mathbf{AF}\,\varphi$ requires that every execution path in $\mathcal{M}_k$ starting from the current state is such that there exists a state that verifies φ. Again, not all paths must be considered: states accessible through freeze transitions must be discarded. The only paths that must fulfil $\mathbf{F}\,\varphi$ are those which do not take a freeze transition:

$$\omega_t^n(\mathbf{AF}\,\varphi_1) \equiv \mathbf{A}\,[\gamma_t^n\,\mathbf{U}\,(\neg\gamma_t^n \vee (\gamma_t^n \wedge \omega_t^n(\varphi_1)))] \tag{6.7}$$

The translation of the **AF** operator is a generalization of the traditional CTL **AF**, defined as $\mathbf{AF}\,\varphi \equiv \mathbf{A}\,[\mathit{true}\,\mathbf{U}\,\varphi]$; it suffices to replace γ_t^n by *true* to recover the original definition. Therefore, the guard γ_t^n can be seen as a filter that determines which paths are admissible.

The case of **EU** is adapted following the same principle:

$$\omega_t^n(\mathbf{E}\,\varphi_1\,\mathbf{U}\,\varphi_2) \equiv \mathbf{E}\,[(\gamma_t^n \wedge \omega_t^n(\varphi_1))\,\mathbf{U}\,(\gamma_t^n \wedge \omega_t^n(\varphi_2))] \tag{6.8}$$

Equation 6.8 imposes the existence of a path where no freeze transition is taken, by adding the guard γ_t^n to both subformulæ φ_1 and φ_2.

The quantification on variables becomes a quantification on some execution paths. Indeed, a quantifier like $\exists_p x_i : \varphi$ actually means "there exists a value a that variable x_i can take in the current state such that φ holds". According to the Kripke structure $\mathcal{M}_k$ defined previously, this simply amounts to asserting that in the current state, there exists a way for $\widehat{\nu}_i$ of changing from # to some definite value, such that the translation of φ is true. By Definition 6.9, the only values $\widehat{\nu}_i$ can change to are in $Dom_{\widehat{s}}(\pi_i)$ for s for the current state $\widehat{s}$. This translates as follows:

$$\omega_t^n(\exists_p x_i : \varphi_1) \equiv \mathbf{EX}\ \left(\gamma_{t+1}^n \wedge \omega_{t+1}^n(\varphi_1)\right) \tag{6.9}$$

Using this embedding, UCLP Formal Constraint 1 is recursively translated to the following CTL expression. The translation for **AG** and **AX** has been obtained from the above equations using the classical identities mentioned in Section 6.4.2.

$$\begin{aligned}
&\neg\mathbf{E}\,[\gamma_0^4\,\mathbf{U}\,(\gamma_0^4 \wedge \\
&\quad(\neg(\mathbf{AX}\,(\gamma_1^4 \rightarrow (x_1 = \text{partitionResponse} \rightarrow \\
&\quad\quad(\mathbf{AX}\,(\gamma_2^4 \rightarrow (\mathbf{AX}\,(\gamma_2^4 \rightarrow \\
&\quad\quad\quad\mathbf{A}\,[\gamma_2^4\,\mathbf{U}\,(\gamma_2^4 \vee (\mathbf{AX}\,(\gamma_3^4 \rightarrow \\
&\quad\quad\quad\quad(x_3 = \text{concatenateRequest} \rightarrow \\
&\quad\quad\quad\quad\quad(\mathbf{EX}\,(\gamma_4^4 \wedge x_2 = x_4))))))]))))))))))]
\end{aligned} \tag{6.10}$$

We do not expect data aware constraints to be expressed directly in CTL in such a way. Instead, the translation from CTL-FO$^+$ to CTL can be automated, and the next theorem proves that the overall construction preserves the validity of the original problem.

Theorem 6.4. *Let $\mathcal{M}_k$ be a workflow messaging model, $s \in S$ be a state of $\mathcal{M}_k$, φ be a CTL-FO$^+$ formula in n variables, ν be an ordered t-valuation for some $0 \leq t \leq n$. Let $\widehat{\mathcal{M}}_k^n$ be the freeze workflow messaging model built from $\mathcal{M}_k$, $\widehat{s}_\nu \in \widehat{S}$ be a state of $\widehat{\mathcal{M}}_k^n$ such that $\widehat{L}(\widehat{s}_\nu, \widehat{\nu}_i) = \nu(x_i)$ for all $1 \leq i \leq n$ and $L(s,\alpha) = \widehat{L}(\widehat{s}_\nu, \alpha)$ for every $\alpha \in \mathcal{P} \cup \mathcal{V} \cup \mathcal{I}$. Then $\mathcal{M}_k, s, \nu \models \varphi$ if and only if $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models \omega_t^n(\varphi)$.*

Proof. The proof is done by structural induction on φ.

Base case: the three ground equality testings must be verified.

1. $c_1 = c_2$: Suppose $\mathcal{M}_k, s, \nu \models c_1 = c_2$, where c_1 and c_2 are constants. By Definition 6.5, then c_1 and c_2 are the same. By equation 6.1, $\omega_t^n(c_1 = c_2) \equiv c_1 = c_2$. Since c_1 and c_2 are the same, then $c_1 = c_2$ is a tautology and $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models c_1 = c_2$. The case where $\mathcal{M}_k, s, \nu \not\models c_1 = c_2$ is proven in the same way.
2. $x_i = c_1$: Suppose $\mathcal{M}_k, s, \nu \models x_i = c_1$. By Definition 6.5, $\nu(x_i) = c_1$. By equation 6.2, $\omega_t^n(x_i = c_1) \equiv \widehat{\nu}_i = c_1$. By hypothesis, $\widehat{L}(\widehat{s}_\nu, \widehat{\nu}_i) = \nu(x_i)$; therefore, $\widehat{L}(\widehat{s}_\nu, \widehat{\nu}_i) = c_1$; hence, $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models \nu_i = c_1$. The case where $\mathcal{M}_k, s, \nu \not\models x_i = c_1$ is proven in the same way.
3. $x_i = x_j$: Suppose $\mathcal{M}_k, s, \nu \models x_i = x_j$. By Definition 6.5, $\nu(x_i) = \nu(x_j)$. By equation 6.3, $\omega_t^n(x_i = x_j) \equiv \widehat{\nu}_i = \widehat{\nu}_j$. By hypothesis, $\widehat{L}(\widehat{s}_\nu, \widehat{\nu}_i) = \nu(x_i)$ and $\widehat{L}(\widehat{s}_\nu, \widehat{\nu}_j) = \nu(x_j)$; therefore, $\widehat{L}(\widehat{s}_\nu, \widehat{\nu}_i) = \widehat{L}(\widehat{s}_\nu, \widehat{\nu}_j)$; hence, $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models \nu_i = \nu_j$. The case where $\mathcal{M}_k, s, \nu \not\models x_i = x_j$ is proven in the same way.

Induction step: Suppose the equivalence is satisfied for all formulæ of length less than ℓ. Let φ be a formula of length ℓ. We must show that the application of ω_t^n in each possible case for φ preserves the satisfiability of the formula.

1. $\neg\varphi'$: By Definition 6.5, $\mathcal{M}_k, s, \nu \models \neg\varphi'$ if and only if $\mathcal{M}_k, s, \nu \not\models \varphi'$. By the induction hypothesis, $\mathcal{M}_k, s, \nu \not\models \varphi'$ if and only if $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \not\models \omega_t^n(\varphi')$. By the classical semantics of CTL, $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \not\models \omega_t^n(\varphi')$ if and only if $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models \neg\omega_t^n(\varphi')$. By equation 6.4, $\neg\omega_t^n(\varphi') \equiv \omega_t^n(\neg\varphi')$.
2. $\varphi' \vee \psi$: By Definition 6.5, $\mathcal{M}_k, s, \nu \models \varphi' \vee \psi$ if and only if $\mathcal{M}_k, s, \nu \models \varphi'$ or $\mathcal{M}_k, s, \nu \models \psi$. By the induction hypothesis, $\mathcal{M}_k, s, \nu \models \varphi'$ if and only if $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models \omega_t^n(\varphi')$, and $\mathcal{M}_k, s, \nu \models \psi$ if and only if $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models \omega_t^n(\psi)$. By the classical semantics of CTL, this is the case if and only if $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models \omega_t^n(\varphi') \vee \omega_t^n(\psi)$. By equation 6.4, $\omega_t^n(\varphi') \vee \omega_t^n(\psi) \equiv \omega_t^n(\varphi' \vee \psi)$.
3. $\mathbf{EX}\,\varphi'$: By Definition 6.5, $\mathcal{M}_k, s, \nu \models \mathbf{EX}\,\varphi'$ if and only if there exists $s' \in S$ such that $(s, s') \in R$ and $\mathcal{M}_k, s', \nu \models \varphi'$. By Definition 6.8, $(s, s') \in R$ if and

only if there exists a state $\widehat{s}'_\nu \in \widehat{S}$ such that $(\widehat{s}_\nu, \widehat{s}'_\nu) \in \widehat{R}$, $L(s', \alpha) = \widehat{L}(\widehat{s}'_\nu, \alpha)$ for every $\alpha \in \mathcal{P} \cup \mathcal{V} \cup \mathcal{I}$, and $\widehat{L}(\widehat{s}'_\nu, \widehat{\nu}_i) = \widehat{L}(\widehat{s}_\nu, \widehat{\nu}_i)$ for every $\widehat{\nu}_i \in \mathcal{F}$; this last condition entails that $\widehat{\mathcal{M}}^n_k, \widehat{s}'_\nu \models \gamma^n_t$. By the induction hypothesis, we know that $\mathcal{M}_k, s', \nu \models \varphi'$ if and only if $\widehat{\mathcal{M}}^n_k, \widehat{s}'_\nu \models \omega^n_t(\varphi')$. By the classical semantics of CTL, the previous two results are equivalent to $\widehat{\mathcal{M}}^n_k, \widehat{s}'_\nu \models \gamma^n_t \wedge \omega^n_t(\varphi')$. This in turn is equivalent to $\widehat{\mathcal{M}}^n_k, \widehat{s}_\nu \models \mathbf{EX}\,(\gamma^n_t \wedge \omega^n_t(\varphi'))$. Finally, $\mathbf{EX}\,(\gamma^n_t \wedge \omega^n_t(\varphi')) \equiv \omega^n_t(\mathbf{EX}\,\varphi')$ by equation 6.6.

4. $\mathbf{AF}\,\varphi'$: Suppose $\mathcal{M}_k, s, \nu \models \mathbf{AF}\,\varphi'$. By Definition 6.5, every path $\pi = ss_1s_2\ldots$, is such that $\mathcal{M}_k, s_i, \nu \models \varphi'$ for some i. Alternatively, this is equivalent to the fact that there is no path $\pi = s_1s_2\ldots$ with $s = s_1$ such that $\mathcal{M}_k, s_i, \nu \not\models \varphi'$ for every $i \geq 1$.

   By equation 6.7, $\omega^n_t(\mathbf{AF}\,\varphi') \equiv \mathbf{A}\,[\gamma^n_t\,\mathbf{U}\,(\neg\gamma^n_t \vee \omega^n_t(\varphi'))]$. By the classical CTL semantics, the formula $\mathbf{A}\,[\gamma^n_t\,\mathbf{U}\,(\neg\gamma^n_t \vee \omega^n_t(\varphi'))]$ if true if and only if for every path $\widehat{\pi} = \widehat{s}_1\widehat{s}_2\ldots$ with $\widehat{s}_\nu = \widehat{s}_1$, there exists an $m \geq 1$ such that $\widehat{M}_k, \widehat{s}_i \models \gamma^n_t$ for every $i < m$, and either $\widehat{M}_k, \widehat{s}_m \not\models \gamma^n_t$ or $\widehat{M}_k, \widehat{s}_m \models \omega^n_t(\varphi')$.

   Let $\widehat{\pi} = \widehat{s}_\nu\widehat{s}_1\widehat{s}_2\ldots$ be a path. Define $1 \leq m_1 \leq \infty$ such that $\widehat{M}_k, \widehat{s}_{m_1} \models \omega^n_t(\varphi')$, and $\widehat{M}_k, \widehat{s}_i \not\models \omega^n_t(\varphi')$ for every $i < m_1$. Similarly, define $1 \leq m_2 \leq \infty$ such that $\widehat{M}_k, \widehat{s}_{m_2} \not\models \gamma^n_t$, and $\widehat{M}_k, \widehat{s}_i \models \gamma^n_t$ for every $i < m_2$. Three cases must be considered:

   - $m_1 \leq m_2$ and $m_1 < \infty$: then $\widehat{M}_k, \widehat{s}_i \models \gamma^n_t$ for every $i < m_1$, and $\widehat{M}_k, \widehat{s}_{m_1} \models \gamma^n_t \wedge \omega^n_t(\varphi')$.

     Let $m = m_1$, and the path fulfils the definition.
   - $m_1 > m_2$: then $\widehat{M}_k, \widehat{s}_i \models \gamma^n_t$ for every $i < m_2$, and $\widehat{M}_k, \widehat{s}_{m_2} \not\models \gamma^n_t$. Let $m = m_2$, and the path fulfils the definition.
   - $m_1 = m_2 = \infty$: then $\widehat{\pi}$ is a path $\widehat{s}\widehat{s}_1\widehat{s}_2\ldots$ such that $\widehat{\mathcal{M}}^n_k, \widehat{s}_i \models \gamma^n_t$ and $\widehat{\mathcal{M}}^n_k, \widehat{s}_i \not\models \omega^n_t(\varphi')$ for every $i \geq 1$. By Definition 6.8, $\widehat{\pi}$ is a path in $\widehat{M}_k$ if and only if there exists a path $\pi = ss_1s_2\ldots$ in $\mathcal{M}_k$ such that, for every $i > 1$, $L(s_i, \alpha) = \widehat{L}(\widehat{s}_i, \alpha)$ for every $\alpha \in \mathcal{P} \cup \mathcal{V} \cup \mathcal{I}$ and $(s_{i-1}, s_i) \in R$. By the induction hypothesis, since

$\widehat{\mathcal{M}}_k^n, \widehat{s}_i \not\models \omega_t^n(\varphi')$ for every $i \geq 1$, then $\mathcal{M}_k, s_i, \nu \not\models \varphi'$ for every $i \geq 1$. This contradicts the hypothesis that no such path exists in $\mathcal{M}_k$.

Therefore, all paths in $\widehat{\mathcal{M}}_k^n$ satisfy the condition, and $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models \omega_t^n(\varphi')$.

Conversely, suppose $\mathcal{M}_k, s, \nu \not\models \mathbf{AF}\,\varphi'$. By Definition 6.5, there exists a path $\pi = ss_1s_2\ldots$ in $\widehat{M}_k$ such that $\mathcal{M}_k, s_i, \nu \not\models \varphi'$ for every $i \geq 1$. By Definition 6.8, π is a path in $\mathcal{M}_k$ if and only if there exists a path $\widehat{\pi} = \widehat{s}_\nu\widehat{s}_1\widehat{s}_2\ldots$ such that:

- $(\widehat{s}_\nu, \widehat{s}_1) \in \widehat{R}$ and for every $j \geq 1$, $(\widehat{s}_j, \widehat{s}_{j+1}) \in \widehat{R}$
- for every $j \geq 1$ and every $\alpha \in \mathcal{P} \cup \mathcal{V} \cup \mathcal{I}$, $\widehat{L}(\widehat{s}_j, \alpha) = L(s_j, \alpha)$
- for every $j \geq 1$ and every $\widehat{\nu}_i \in \mathcal{F}$, $\widehat{L}(\widehat{s}_j, \widehat{\nu}_i) = \nu(x_i)$

Therefore, for every $i \geq 1$, we have $\widehat{M}_k, \widehat{s}_i \models \gamma_t^n$, and by the induction hypothesis, $\widehat{\mathcal{M}}_k^n, \widehat{s}_i \not\models \omega_t^n(\varphi')$. By the classical CTL semantics, we then have that $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \not\models \mathbf{A}\,[\gamma_t^n\,\mathbf{U}\,(\neg\gamma_t^n \vee (\gamma_t^n \wedge \omega_t^n(\varphi')))]$.

5. $\mathbf{E}\,[\varphi'\,\mathbf{U}\,\psi]$: By Definition 6.5, $\mathcal{M}_k, s, \nu \models \mathbf{E}\,[\varphi'\,\mathbf{U}\,\psi]$ holds if and only if there exists a path $\pi = ss_1s_2\ldots$ and an $m \geq 1$ such that $\mathcal{M}_k, s_i, \nu \models \varphi'$ for all $i < m$ and $\mathcal{M}_k, s_m, \nu \models \psi$. By Definition 6.8, π is a path in $\mathcal{M}_k$ if and only if there exists a path $\widehat{\pi} = \widehat{s}_\nu\widehat{s}_1\widehat{s}_2\ldots$ such that
   - $(\widehat{s}_\nu, \widehat{s}_1) \in \widehat{R}$ and for every $j \geq 1$, $(\widehat{s}_j, \widehat{s}_{j+1}) \in \widehat{R}$
   - for every $j \geq 1$ and every $\alpha \in \mathcal{P} \cup \mathcal{V} \cup \mathcal{I}$, $\widehat{L}(\widehat{s}_j, \alpha) = L(s_j, \alpha)$
   - for every $j \geq 1$ and every $\widehat{\nu}_i \in \mathcal{F}$, $\widehat{L}(\widehat{s}_j, \widehat{\nu}_i) = \nu(x_i)$

   Therefore, every state $\widehat{s}_i$ along $\widehat{\pi}$ is such that $\widehat{M}_k, \widehat{s}_i \models \gamma_t^n$. Moreover, by the induction hypothesis, $\mathcal{M}_k, s_i, \nu \models \varphi'$ for all $i < m$ if and only if $\widehat{M}_k, \widehat{s}_i \models \omega_t^n(\varphi')$ for all $i < m$, and $\mathcal{M}_k, s_m, \nu \models \psi$ if and only if $\widehat{M}_k, \widehat{s}_m \models \omega_t^n(\psi)$. By the classical CTL semantics, this is equivalent to $\widehat{M}_k, \widehat{s}_\nu \models \mathbf{E}\,[(\gamma_t^n \wedge \omega_t^n(\varphi))\,\mathbf{U}\,(\gamma_t^n \wedge \omega_t^n(\psi))]$, and by equation 6.8, this in turn is equivalent to $\widehat{M}_k, \widehat{s}_\nu \models \omega_t^n(\mathbf{E}\,[\varphi'\,\mathbf{U}\,\psi])$.

6. $\exists_p x_i : \varphi'$: Since φ is well-named, the quantification of variable x_i entails that all variables $x_1, \ldots, x_{i-1}$ are already defined by ν; hence $i = t+1$ and $p = \pi_{t+1}$. By Definition 6.5, $\mathcal{M}_k, s, \nu \models \exists_{\pi_{t+1}} x_{t+1} : \varphi'$ if and only if there exists $a \in Dom_s(\pi_{t+1})$

such that $\mathcal{M}_k, s, \nu[a/x_{t+1}] \models \varphi'$; $\nu[a/x_i]$ is the $(t+1)$-valuation that agrees with the ordered t-valuation ν for all x_i with $1 \leq i \leq t$, and which maps x_{t+1} to a.

By the induction hypothesis, $\mathcal{M}_k, s, \nu[a/x_{t+1}] \models \varphi'$ holds if and only if $\widehat{\mathcal{M}}_k^n, \widehat{s}_{\nu[a/x_i]} \models \omega_{t+1}^n(\varphi')$, where $\widehat{s}_{\nu[a/x_i]} \in \widehat{S}$ is such that $\widehat{L}(\widehat{s}_{\nu[a/x_i]}, \alpha) = L(s, \alpha)$ for every $\alpha \in \mathcal{P} \cup \mathcal{V} \cup \mathcal{I}$, and $\widehat{L}(\widehat{s}_{\nu[a/x_i]}, \widehat{\nu}_j) = \nu[a/x_i](x_j)$ for every $\widehat{\nu}_j \in \mathcal{F}$; moreover, by definition $\widehat{s}_{\nu[a/x_i]}$ is such that $\widehat{\mathcal{M}}_k^n, \widehat{s}_{\nu[a/x_i]} \models \gamma_{t+1}^n$. By Definition 6.9, this is true if and only if $(\widehat{s}_\nu, \widehat{s}_{\nu[a/x_{t+1}]}) \in \widehat{R}$. By the classical semantics of CTL, this is true if and only if $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models \mathbf{EX}\,(\gamma_{t+1}^n \wedge \omega_{t+1}^n(\varphi'))$, which by equation 6.9 is equivalent to $\widehat{\mathcal{M}}_k^n, \widehat{s}_\nu \models \omega_t^n(\exists_p x_{t+1} : \varphi')$.

□

Corollary 6.1. *Let $\mathcal{M}_k$ be a workflow messaging model, φ be a CTL-FO$^+$ formula in n variables, $\widehat{\mathcal{M}}_k^n$ be the freeze workflow messaging model built from $\mathcal{M}_k$, and $\widehat{\varphi} = \omega_0^n(\varphi)$. Then $\mathcal{M}_k \models \varphi$ if and only if $\widehat{\mathcal{M}}_k^n \models \widehat{\varphi}$.*

Proof. From Definition 6.7, $s \in I$ if and only if $\widehat{s} \in \widehat{I}$, with $L(s, \alpha) = \widehat{L}(\widehat{s}, \alpha)$ for every $\alpha \in \mathcal{P} \cup \mathcal{V} \cup \mathcal{I}$, and $\widehat{L}(\widehat{s}, \widehat{\nu}_i) = \#$ for every $\nu_i \in \mathcal{F}$. But then by Theorem 6.4, $\mathcal{M}_k, s, \nu \models \varphi$ if and only if $\widehat{\mathcal{M}}_k^n, \widehat{s} \models \widehat{\varphi}$, with ν the empty valuation. □

Contrarily to explicit quantification, the freeze quantification approach does not cause an exponential blow-up of the original formula. The embedding ω is linear: that is, if we denote by $|\varphi|$ the length of a CTL-FO$^+$ formula φ, then $|\omega_0^n(\varphi)| \in O(|\varphi|)$. It suffices to remark that each translation rule consumes at least one symbol of the original CTL-FO$^+$ formula and contributes a fixed number of symbols in the resulting CTL formula.

6.7 Experimental Results

To confirm the soundness of our approach, we conducted a set of simple experiments that involved the validation of the UCLP constraints detailed in Section 6.3.2 on sample BPEL processes taken from real-world UCLP use cases. This section shows results intended to present an overview of explicit quantification compared to freeze quantification.

6.7.1 Methodology

The goal of these experiments is twofold: first, show that validating UCLP constraints can be done using the freeze quantification solution presented in this paper; second, exhibit sample properties for which the explicit quantification approach quickly becomes inadequate. Therefore, we conducted the following steps for both approaches:

1. Select a UCLP BPEL process for validation
2. Generate a Kripke structure from this BPEL process
3. Apply modifications to this Kripke structure
4. Translate the CTL-FO$^+$ formula into the corresponding CTL formula
5. Model-check the resulting formula against the new Kripke structure

The experiments were made using only readily available and open source tools. NuSMV (Cimatti *et al.*, 2002) was chosen as the model checker because of its robustness, good performance, and especially its CTL model checking capability. VERBUS (Arias-Fisteus *et al.*, 2005) was chosen to generate the Kripke structure from the original BPEL processes; the choice was influenced mostly by its ability to model the content of variables and messages in a process and its capability to directly output the Kripke structure

as a NuSMV file. Only minor bug-solving modifications were made to the original VERBUS code to make it work in our situation. In addition, the system variables produced by VERBUS, but that were not necessary either for the validation of explicit or freeze quantifications, were removed from the files.

The resulting SMV files were then modified according either to the explicit or the freeze quantification approach. In the explicit quantification, no modification other than appending the desired CTL formula at the end of the file was made. In the freeze approach, freeze variables and freeze transitions were added throughout the file conforming to Definitions 6.6–6.9, and the corresponding CTL formula was also added at the end.

6.7.2 Results and Discussion

We then followed this method using as input various BPEL processes. Each process consisted of a short sequence of operations on a given number of LPOs. For $n = 1$, the result of all operations in the process is deterministic: for example, the partition operation with LPO A always returns the same LPO IDs B, C, D. We then varied this number n of LPOs, without changing the process, by making the operations non-deterministically return IDs taken from a set of possible values. This generalization of the process is a natural step to take, since a UCLP operation on an LPO is dependent of the global situation of the UCLP network and, therefore, need not return the same value on every invocation.

Figures 6.5 and 6.6 present the validation times for the freeze and the explicit quantification approaches of two UCLP formal constraints on processes with n ranging from 1 to 10 (all figures are semi-logarithmic plots). All times have been obtained with NuSMV 2.4.0 on an AMD Athlon 2200+ CPU running under Windows XP (Cygwin). Since NuSMV takes several dozens of seconds only to display the explicitly quantified

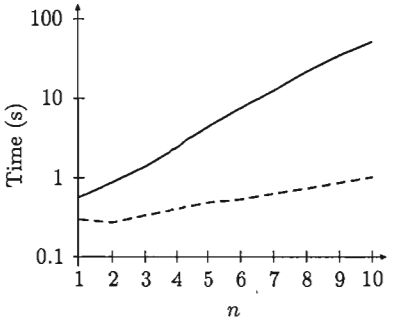


Figure 6.5: Validation time (in seconds) for UCLP Property 5 with respect to the size n of the domain, using respectively the explicit quantification (dashed curve) and the freeze quantification approach (solid curve).

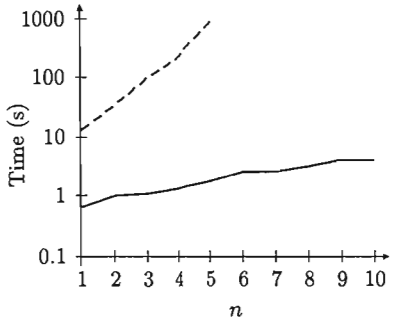


Figure 6.6: Validation time (in seconds) for UCLP Property 2 with respect to the size n of the domain, using respectively the explicit quantification (dashed curve) and the freeze quantification approach (solid curve).

formulæ, display from NuSMV was disabled.

We can distinguish two situations. In the case of Property 5 (Figure 6.5), the use of freeze quantification yields manageable results (a growth on the order of 1.7^n), but the explicit quantification approach fares better (with a growth on the order of 1.1^n). This can be explained by the fact that the addition of freeze variables and transitions imposes an initial overhead on the Kripke structure that the explicit quantification does not have. In this situation, the CTL formulæ to validate are still too short to represent a serious load on the model checker. Properties 1 and 4 behave similarly on this respect.

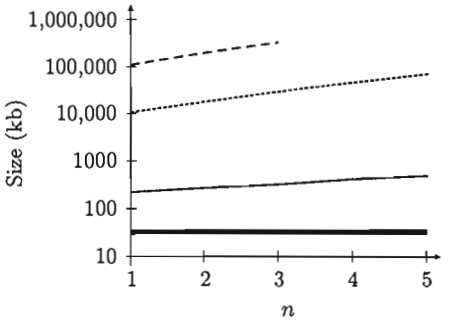


Figure 6.7: Size (in kilobytes) of the SMV files (including the formula to validate) with respect to the size n of the domain. Starting from the top: UCLP Properties 3, 2 and 1 using explicit quantification, and the same properties using freeze quantification (all in the same region at the bottom of the graph).

On the other hand, for Property 2 (Figure 6.6), using the explicit quantification approach to produce the CTL formula quickly takes its toll on the validation time. As expected, the translation of the CTL-FO$^+$ formula is heavily dependent on n and its validation time grows much faster than for freeze quantification: on the order of 4.6^n and 1.2^n, respectively. Even a completely deterministic process ($n = 1$) needs more than 10 seconds to be validated using the explicit quantification. With UCLP operations returning only 5 different possible values, validation time takes almost 15 minutes.

The size of the NuSMV files required to validate some of the process is shown in Figure 6.7. This size includes the size of the formula freeze or explicit to validate.

This latter figure helps explain the differences measured in the previous validation times. It confirms that, as expected, all explicitly quantified formulæ increase exponentially in length from roughly the same factor; the crucial factor is the initial complexity of the formula, since it determines the starting value for each curve. For simple CTL-FO$^+$ formulæ like UCLP Constraints 1, 4 and 5 (third curve from the top), the file size remains manageable; we have already seen that in these situations, the explicit quantification approach wins over freeze quantification in terms of validation time.

However, for more complex CTL-FO$^+$ formulæ such as Properties 2 and 3, the file size quickly explodes using the explicit quantification approach, reaching more than 70 megabytes for UCLP Property 2. We could not get any times for UCLP Formal Constraint 3 since NuSMV crashed before processing the whole file, most probably due to its size. The generation of the formulæ for $n = 4$ and $n = 5$ was manually aborted after the file size reached 500 Mb. Comparatively, the freeze approach requires only minimal modifications to the original NuSMV file produced by VERBUS; most importantly, these modifications do not depend on n.

Investigation of the exact causes of the complexity in each situation is beyond the scope of this paper. However, these findings are sufficient to deduce that the size of the formula is a major factor influencing the performance of the validation. In that respect, the freeze quantification approach, whose translation produces a CTL formula linear in the size of the original CTL-FO$^+$ formula, seems more promising to handle complex properties. Moreover, our results show that the validation time of the freeze quantification approach, compared to explicit quantification, is much less sensitive to the complexity of the CTL-FO$^+$ formula, and that there exists situations where explicit quantification cannot reasonably be used as is.

6.8 Conclusion

In this paper, we have shown how "data-aware" temporal properties can be used to express constraints on the behaviour of web service compositions. These properties enable complex temporal relationships to be expressed, while at the same time allowing full first-order quantification on the content of the messages. We presented a real-world scenario where data-aware properties arise naturally, and showed how existing related work is only partially appropriate for the validation of such properties. To this end, we introduced the logic CTL-FO$^+$, showed its model checking algorithm and studied

its complexity. We conclude that model checking data-aware temporal properties is a tractable problem and that any web service model that uses a data-aware workflow, but only use *propositional* properties cannot efficiently simulate data-awareness.

We have demonstrated by empirical testing on BPEL processes how a suitable reduction of CTL-FO$^+$ to CTL can be used to validate them in reasonable time compared to classical approaches.

This project lends itself to many further developments. An appropriate modification of the workflow messaging model could take into account structured (instead of flat) messages. Moreover, static, *a priori* model checking on scripts could be complemented with runtime monitoring of CTL-FO$^+$ properties for a given transaction.

Acknowledgements

The authors gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada.

They would like to thank the Université du Québec à Montréal–University of Ottawa UCLP Development Team, and in particular Jérôme Tremblay and Boubker Ghandour, for having provided the theoretical and practical environment required for testing the validation rules.

CONCLUSION

Dans ce travail, on a vu comment le passage d'une approche impérative à une approche déclarative de la conception des systèmes d'information entraîne la nécessité de résoudre trois problèmes fondamentaux : l'expression de contraintes dans un langage formel, la validation de ces contraintes sur une structure, et enfin la construction d'une structure satisfaisant une contrainte donnée.

En se basant sur deux contextes concrets, la gestion des configurations réseaux et le déploiement de services web, on a également démontré l'existence de contraintes de trois types : les contraintes statiques énoncent des propriétés sur les valeurs d'un seul état d'un système ; les contraintes dynamiques imposent des règles quant à la succession d'états d'un système ; enfin les contraintes hybrides sont la fusion des deux précédentes et relient entre elles des valeurs précises de deux états du système séparés dans le temps. L'objectif du travail consistait à développer un formalisme basé sur la logique permettant l'expression et la validation de contraintes à la fois statiques, dynamiques et hybrides, de même que la construction automatique de structures satisfaisant une contrainte donnée. Le formalisme développé devait pouvoir être traduit en logique temporelle afin de réutiliser au maximum les outils existants.

Au terme de cette thèse, il convient de revenir sur ces trois grands problèmes et mesurer le chemin parcouru.

Résumé des contributions

Pour énoncer les contraintes statiques, un fragment du langage de requête sur les arbres XQuery a été étudié plus en détail. On a vu aux chapitres 1 et 3 comment les problèmes de *model checking* et de satisfiabilité pour ce fragment, appelé Configuration

Logic (CL), peuvent être réduits de manière efficace aux problèmes de *model checking* et de satisfiabilité de CTL.

Au chapitre 1, on a présenté et étudié la logique CL, un fragment du langage XQuery permettant d'exprimer des contraintes statiques sur des structures de données arborescentes. Ces structures arborescentes se manifestent dans un grand nombre de contextes concrets, en particulier les documents XML couramment utilisés dans la gestion des configurations réseaux grâce au protocole Netconf, ainsi que dans les transactions de services web par l'échange de messages SOAP définis dans des documents de type WSDL.

On a vu comment le problème de la vérification d'une formule CL pour une structure arborescente donnée était équivalent au problème du *model checking* d'une formule CTL pour une structure de Kripke donnée. La construction utilisée pour démontrer cette équivalence faisait appel à la technique de *freeze quantification* par laquelle l'ajout de variables d'état au système permettait de simuler l'affectation d'une valeur aux variables quantifiées de la formule CL originale. Ce chapitre étendait les résultats d'un travail mené indépendamment par des chercheurs européens (Afanasiev *et al.*, 2004). La technique de quantification devait être reprise avec quelques variantes dans les chapitres 3 et 6 pour ramener d'autres questions à des problèmes de logique temporelle. La complexité de ce problèmes a également été établie : le *model checking* de CL est PSPACE-complet.

Le chapitre 2 a introduit par un exemple concret l'importance du problème de la satisfiabilité de la logique CL. En se basant sur le contexte des réseaux autonomiques, et en particulier le domaine de la configuration automatique (*self-configuration*), le chapitre montre comment le problème de l'auto-configuration d'équipement réseau se ramène à la satisfiabilité d'un ensemble de formules CL.

Le bon fonctionnement d'un réseau local virtuel (VLAN) était formalisé par un ensemble de contraintes sur la configuration de ses équipements constituants ; ces contraintes étaient exprimées dans la logique CL. Le branchement d'un nouvel appareil à un VLAN

existant déclenchait ensuite une procédure automatique qui cherchait à bâtir de zéro la configuration de cet appareil. La configuration devait respecter les contraintes du VLAN, c'est-à-dire satisfaire les formules CL définissant le VLAN. L'algorithme développé pour ce faire constitue une procédure simple de satisfiabilité d'une formule CL construisant une formule booléenne représentant l'ensemble des « options » permises. Cette formule booléenne pouvait ensuite être passée à un solveur SAT dont la réponse éventuelle constituait une manière de créer la configuration tout en respectant les contraintes originales. À l'exception du travail de Narain (2005), cet article constitue l'un des rares travaux connus à ce jour étudiant la *self-configuration* des équipements réseaux comme un problème de satisfiabilité logique.

Au moment de la publication du chapitre 2, la décidabilité du problème de satisfiabilité de CL n'était cependant pas encore démontrée. La procédure qui y était présentée fonctionnait de manière itérative sans toutefois fournir une garantie de complétude ou de terminaison. Le chapitre 3 s'est attaqué à cette question en étudiant les conditions dans lesquelles la satisfiabilité de CL était décidable. À cet effet, plusieurs liens ont été établis entre CL et d'autres fragments de logiques existantes, en particulier la logique gardée et la logique hybride.

On a vu que la satisfiabilité de CL était assurée moyennant l'utilisation de prédicats unaires ; la procédure de décision est démontrée complète pour la classe NEXPTIME. Contrairement à la logique classique du premier ordre, l'égalité permet à CL de simuler n'importe quelle relation binaire et entraîne donc l'indécidabilité du problème de satisfiabilité. Cependant, comme le chapitre 2 l'a montré, les applications pratiques demandent une procédure de décision déterministe ; une telle procédure fut ensuite construite par réduction à la satisfiabilité de CTL. Alors que les similitudes entre les logiques sur les arbres et les logiques modales ont été bien étudiées sur le plan du *model checking*, il en va tout autrement du problème de satisfiabilité ; à cet égard, ce chapitre constitue donc une avancée par rapport aux connaissances actuelles.

En particulier, une retombée inattendue de ce chapitre fut de démontrer que CL, pourtant décidable, pouvait être plongée dans un fragment de la logique hybride $HL(\downarrow, @)$ respectant une condition démontrée nécessaire pour obtenir l'indécidabilité. La condition n'est évidemment pas suffisante ; ce résultat pourrait éventuellement permettre de circonscrire davantage le « noyau » indécidable de la logique hybride.

Le chapitre 4 quitte le domaine des contraintes statiques pour étudier les contraintes dynamiques. Il suggère qu'il existe des dépendances séquentielles entre les opérations de configuration d'un service réseau et que ces dépendances doivent être explicitées, modélisées, et vérifiées automatiquement. L'article introduit également le concept de borne (*milestone*), en donne la définition mathématique et avance que ces bornes divisent les opérations de configuration en macro-étapes dont elles sont les frontières naturelles.

Le concept de dépendance séquentielle est un élément central de la suite de la thèse. En effet, si l'idée d'étudier les contraintes dans l'ordre des opérations de configuration d'un réseau informatique devait être justifiée par des exemples concrets (exemples fournis par le chapitre 4), la présence de telles contraintes d'ordonnancement dans les architectures orientées services est, quand à elle, évidente : prenons à témoin l'impressionnante littérature produite à ce sujet et mentionnée au début de cette thèse et au chapitre 6. La notion de borne est toujours en cours d'étude et a fait l'objet de publications récentes d'auteurs indépendants (Sun et Couch, 2006).

Au chapitre 5, les problèmes de la validation d'une séquence d'opérations de configuration de réseaux (*model checking*) de même que de la création d'une séquence d'opérations respectant une contrainte dynamique donnée (satisfiabilité) ont été abordés d'un point de vue expérimental. Alors qu'au chapitre précédent, des notions de treillis étaient utilisées pour représenter les composantes et les *milestones*, ces notions étaient ici exprimées au moyen de formules de logique temporelle.

Le chapitre fournit deux nouveaux exemples mettant en évidence la présence de

contraintes dynamiques dans la gestion des équipements réseau : la configuration d'un réseau privé virtuel (VPN) et d'un réseau local virtuel (VLAN). On remarquera au passage que les VLAN ont déjà été étudiés au chapitre 2 sous l'angle des contraintes statiques. De plus, bien qu'elles ne soient pas formalisées comme telles, les opérations de configuration décrites au chapitre 5 impliquent des modifications à la structure *arborescente* de la configuration et annoncent les contraintes hybrides abordées au chapitre suivant.

Le chapitre 6 a synthétisé les résultats précédents en présentant CTL-FO$^+$, un formalisme logique permettant l'expression de contraintes statiques, dynamiques, de même que hybrides ; des contraintes tirées de l'utilisation de services web dans des contextes concrets ont illustré l'approche. Fidèle au principe de cette thèse, et à partir des résultats obtenus dans les chapitres précédents, une réduction efficace du problème du *model checking* de CTL-FO$^+$ au *model checking* de CTL a été présentée, et sa complexité fut établie complète pour la classe PSPACE.

Pour ce faire, un modèle transactionnel appelé *workflow messaging model* a été introduit ; ce modèle permet de tenir compte à la fois des modifications au contenu des structures (c.-à-d. le contenu des messages), mais également à la séquence de ces modifications. La particularité de ce chapitre fut d'étudier deux approches différentes à la vérification des contraintes dynamiques : une traduction directe des formules CTL-FO$^+$, produisant une formule CTL de taille exponentielle, et une traduction nouvelle impliquant des modifications à la structure de Kripke modélisant la transaction. Parmi le faible nombre de travaux étudiant les contraintes dynamiques, il s'agit sous toutes réserves du premier travail présentant une réduction non-triviale au problème de *model checking* des formules CTL. Les résultats présentés dans cet article montrent que la construction développée rend possible la vérification de contraintes inaccessibles par une traduction « naïve » en CTL et s'avère donc une approche prometteuse.

On peut tirer des contributions de ces chapitres les conclusions suivantes.

D'abord, il est possible, au moyen de constructions appropriées, de traduire des problèmes de logique sur des données semi-structurées en problèmes équivalents de logique temporelle, en particulier CTL. La plupart de ces constructions entraînent une élévation de la classe de complexité mais demeurent dans des limites raisonnables. Au moyen d'exemples concrets, on a vu que les problèmes de *model checking* et de satisfiabilité de la logique CL permettent de résoudre des problèmes de configuration d'équipement réseau et de transactions de services web.

Ensuite, il existe des dépendances séquentielles (dynamiques) dans un grand nombre de contextes concrets. À cet égard, les logiques temporelles LTL et CTL sont appropriées pour modéliser ces dépendances.

Enfin, il existe également des contraintes hybrides faisant référence à la fois au contenu et à la séquence des opérations. Ces contraintes ont été mises en évidence en particulier dans les transactions de services web, mais peuvent également apparaître dans la gestion des configurations réseau grâce à l'utilisation du protocole Netconf. Au moyen de la technique de *freeze quantification*, on peut alors utiliser les traductions des logiques statiques sur les arbres et ramener le problème entier à un problème de logique temporelle. Des résultats expérimentaux ont montré que cette technique était réalisable et que des contraintes réelles pouvaient être vérifiées au moyen d'outils existants.

Travaux futurs

Les constructions développées durant cette thèse ouvrent la voie à de nombreux raffinements et à plusieurs applications. Nous en mentionnons quelques uns.

La syntaxe de la logique CL pourrait être étendue afin d'inclure une certaine forme de récursion ; les constructions présentées aux chapitres 1 et 3 pourraient être modifiées en

conséquence. Les modèles de telles formules sont des arbres réguliers, c.-à-d. des graphes contenant des cycles.

Une adaptation des algorithmes de *model checking* de formules CTL-FO$^+$ pour la vérification en temps réel permettrait la surveillance (*monitoring*) de l'exécution d'un service web ; cette fonctionnalité revêt une grande importance pour la communauté des services web à l'heure actuelle (Bianculli et Ghezzi, 2007) et rejoint la notion du moniteur (*watcher*) présentée par Bultan *et al.* (2003). Par exemple, il est bien connu qu'une formule LTL peut être traduite en un automate de Büchi ; il suffit d'utiliser cet automate comme observateur pour vérifier en temps réel qu'une trace donnée satisfait la propriété : LTL est donc un langage ω-régulier. À quelle classe de langages appartient CTL-FO$^+$?

En suivant les techniques exposées aux chapitres 3 et 5, il serait possible de réduire le problème de la satisfiabilité de CTL-FO$^+$ à la satisfiabilité de CTL. Comme le problème du *model checking* est plus « simple » que la satisfiabilité pour une même logique et que le *model checking* de CTL-FO$^+$ a été démontré PSPACE-complet au chapitre 6, il est raisonnable de conjecturer que sa satisfiabilité soit « au moins » NEXPTIME-complète. On pourrait ainsi résoudre, et formellement établir, la complexité du problème de la composition automatique de services web (Gerede *et al.*, 2004) selon une approche purement logique.

Le chapitre 3 a donné un certain nombre de conditions garantissant la décidabilité du fragment statique de CTL-FO$^+$, en particulier la finitude des domaines. À l'instar du développement de la logique gardée (Andréka *et al.*, 1998), il serait utile de développer un équivalent « gardé » de la logique CTL-FO$^+$ qui la rendrait décidable (ou moins complexe) pour un plus grand nombre de situations. Cette technique a déjà été suggérée pour d'autres logiques (Deutsch *et al.*, 2006), où elle porte le nom de *input boundedness*.

La question de la *réalisabilité* (Bultan *et al.*, 2003; Decker et Weske, 2007; Zaha *et al.*, 2006; Decker *et al.*, 2006) peut également être étudiée en utilisant CTL-FO$^+$ comme

langage de spécification des interactions. Les résultats déjà obtenus pour des langages similaires à LTL tiennent-ils toujours si on emploie plutôt CTL-FO$^+$?

On doit également souligner que depuis le début de ce projet, en 2005, un certain nombre d'équipes de recherche ont commencé à s'intéresser à des problèmes semblables à l'objectif de cette thèse. Mentionnons le développement du *artifact-centric business process management* (Gerede *et al.*, 2007), présentement mis de l'avant par les centres de recherche d'IBM à travers le monde ; une logique semblable en plusieurs points à CTL-FO$^+$, mais spécifique aux comportements d'objets appelés artefacts, y est même suggérée. En quoi la vérification de contraintes CTL-FO$^+$ sur un *workflow messaging model*, tel que décrite dans cette thèse, est-elle semblable ou différente du modèle artefact ? Il y a fort à parier que ces deux modèles possèdent une intersection commune ; il serait cependant intéressant d'étudier les situations où l'un ne peut être simulé par l'autre.

On peut revenir à l'article (Constant *et al.*, 2007) cité en début de thèse, et qui ouvre la porte à une multitude de questions. La méthode des auteurs permet d'atteindre un but similaire à celui visé par CTL-FO$^+$, mais de manière différente. On peut naturellement se demander : comment se compare l'expressivité de leurs observateurs vis-à-vis d'une formule CTL-FO$^+$? Les auteurs mentionnent que *certaines* propriétés LTL sont équivalentes à des observateurs, mais qu'en est-il pour d'autres fragments ou d'autres logiques ? La notion d'observateur correspond à la vision de vérification en temps réel d'une chorégraphie de services web mentionnée ci-dessus : il s'agit ni plus ni moins que d'un automate qui suit au fur et à mesure l'exécution d'un système et qui « sonne une cloche » quand quelque chose ne fonctionne plus. Répondre à la question précédente revient à se demander comment vérifier en temps réel des propriétés CTL-FO$^+$.

Certains chercheurs commencent également à explorer la réduction de la satisfiabilité au problème du *model checking* telle qu'ébauchée à grands traits dans la section 5.4 ; un

article récent à ce sujet est celui de Rozier et Vardi (2007). L'attention portée à ces questions par un grand nombre de chercheurs confirme l'intérêt suscité par la résolution de problèmes du paradigme déclaratif via une approche logique.

CONCLUSION – ENGLISH SUMMARY

The constructions developed in this thesis open the way to numerous refinements and applications:

- Adapting CTL-FO$^+$ model checking to the verification of properties in real time would allow runtime monitoring of the execution of a web service interaction, an important issue for the web service community (Bianculli et Ghezzi, 2007).
- By following the techniques described in Chapters 3 and 5, it would be possible to reduce the problem of CTL-FO$^+$ satisfiability to CTL satisfiability. Since model checking is a "simpler" problem than satisfiability for the same logic, and since CTL-FO$^+$ model checking has been shown PSPACE-complete in Chapter 6, its satisfiability could be reasonably expected to be NEXPTIME-complete or "more". It will then be possible to solve, and incidentally to establish the computational complexity, of the problem of automated web service composition (Gerede *et al.*, 2004) by a purely logical approach.
- Chapter 3 gave a number of conditions that guarantee the decidability of the static fragment of CTL-FO$^+$; finite domains was one of these conditions. Following the development of guarded logic (Andréka *et al.*, 1998), it would be useful to develop an equivalent "guarded" fragment of CTL-FO$^+$ which would ensure its decidability (or lower its complexity) in a wider range of situations. This technique has been suggested for other logics, for example in Deutsch *et al.* (2006), where the condition is called *input boundedness*.

We must also mention that since the beginning of this project, in 2005, many research groups worked on problems similar to the goals of this thesis. For example, *artifact-centric business process management* (Gerede *et al.*, 2007) is currently pioneered by joint academic and industrial teams from IBM around the world; a logic similar to CTL-FO$^+$

in many aspects, but specific to behaviours of so-called artifact objects, is also suggested. Others have explored the reduction of satisfiability to the problem of model checking as was roughly sketched in Section 5.4; a recent paper on this topic is the one by Rozier and Vardi (2007). The attention paid to these questions by a large number of researchers confirms the interest raised by the resolution of declarative problems through a logical approach.

RÉFÉRENCES

2006a. *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings.* IEEE Computer Society.

2006b. *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), 16-20 October 2006, Hong Kong.* IEEE Computer Society.

2007. *11th International Enterprise Distributed Object Computing Conference (EDOC 2007), 15-19 October 2007, Annapolis, USA, Proceedings.* IEEE Computer Society.

Aagesen, F. A., C. Anutariya et V. Wuwongse (dir. publ.). 2004. *Intelligence in Communication Systems, IFIP International Conference, INTELLCOMM 2004, Bangkok, Thailand, November 23-26, 2004, Proceedings*, *Lecture Notes in Computer Science*, vol. 3283. Berlin: Springer Verlag.

Abiteboul, S. 1997. «Querying semi-structured data». In *International Conference on Database Theory.* p. 1–18.

Afanasiev, L. 2004. «XML query evaluation via CTL model checking». In *European Summer School on Logic, Languages and Information.* p. 1–12.

Afanasiev, L., P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. Marx et M. de Rijke. 2005. «PDL for ordered trees». *Journal of Applied Non-Classical Logics*, vol. 15, no. 2, p. 115–135.

Afanasiev, L., M. Franceschet, M. Marx et M. de Rijke. 2004. «CTL model checking for processing simple XPath queries.» In *11th International Symposium on Temporal Representation and Reasoning (TIME 2004), 1-3 July 2004, Tatihou Island, Normandie, France.* IEEE Computer Society, p. 117–124.

Aiello, M., M. Aoyama, F. Curbera et M. P. Papazoglou (dir. publ.). 2004. *Service-Oriented Computing - ICSOC 2004, Second International Conference, New York, NY, USA, November 15-19, 2004, Proceedings.* ACM.

Alechina, N., S. Demri et M. de Rijke. 2003. «A modal perspective on path constraints». *J. Log. Comput.*, vol. 13, no. 6, p. 939–956.

Alonso, G., F. Casati, H. Kuno et V. Machiraju. 2004. *Web Services, Concepts, Architectures and Applications.* Berlin: Springer Verlag.

Alur, R. et T. A. Henzinger. 1994. «A really temporal logic». *Journal of the ACM*, vol. 41, no. 1, p. 181–204.

Andréka, H., J. van Benthem et I. Németi. 1998. «Modal languages and bounded fragments of predicate logic». *Journal of Philosophical Logic*, no. 27, p. 217–274.

Andrews, T., F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic et S. Weerawarana. 2003. «Business process execution language for web services, version 1.1».

Areces, C., P. Blackburn et M. Marx. 1999. «Hybrid logic is the bounded fragment of first order logic». In de Queiroz, R. J. G. B. et W. A. Carnielli (dir. publ.), *6th Workshop on Logic, Language, Information and Computation (WoLLIC'99)*. p. 33–50.

Areces, C., P. Blackburn et M. Marx. 2001. «Hybrid logics: Characterization, interpolation, and complexity». *Journal of Symbolic Logic*, vol. 66, no. 3, p. 977–1010.

Arias-Fisteus, J. 2005. «Definición de un modelo para la verificación formal de procesos de negocio». Thèse de doctorat, Universidad Carlos III de Madrid, Madrid, 268 p.

Arias-Fisteus, J., L. S. Fernández et C. D. Kloos. 2004a. «Formal verification of BPEL4WS business collaborations». In Bauknecht, K., M. Bichler et B. Pröll (dir. publ.), *EC-Web*, *Lecture Notes in Computer Science*, vol. 3182. Springer, p. 76–85.

Arias-Fisteus, J., L. S. Fernández et C. D. Kloos. 2005. «Applying model checking to BPEL4WS business collaborations». In Haddad, H., L. M. Liebrock, A. Omicini et R. L. Wainwright (dir. publ.), *ACM Symsposium on Applied Computing (SAC)*. ACM, p. 826–830.

Arias-Fisteus, J., A. Marin et C. D. Kloos. 2004b. «VERBUS: A formal model for business process verification». In *Information Resources Management Association Conference*.

Bajaj, S., D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, H. Prafullchandra, C. von Riegen, D. Roth, J. Schlimmer, C. Sharp, J. Shewchuk, A. Vedamuthu, Ü. Yalçinalp et D. Orchard. 2006. «Web services policy framework (WS-Policy), version 1.2».

Ball, T., B. Cook, V. Levin et S. K. Rajamani. 2004. «SLAM and static driver verifier: Technology transfer of formal methods inside Microsoft». In Boiten, E. A., J. Derrick et G. Smith (dir. publ.), *IFM*, *Lecture Notes in Computer Science*, vol. 2999. Springer, p. 1–20.

Ball, T. et S. K. Rajamani. 2000. «Boolean programs: A model and process for software analysis». Tech. Rep. MSR-TR-2000-14, Microsoft Research.

Benedikt, M. et G. Bruns. 2004. «On guard: Producing run-time checks from integrity constraints». In Rattray, C., S. Maharaj et C. Shankland (dir. publ.), *AMAST*, *Lecture Notes in Computer Science*, vol. 3116. Berlin: Springer Verlag, p. 27–41.

Benedikt, M., G. Bruns, J. Gibson, R. Kuss et A. Ng. 2002. «Automated update management for XML integrity constraints». In *PLAN-X: Programming Language Technologies for XML, Informal Proceedings, 3 October 2002, Pittsburgh, PA*. p. 14.

Benedikt, M., W. Fan et F. Geerts. 2005. «XPath satisfiability in the presence of DTDs». In Li, C. (dir. publ.), *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*. ACM, p. 25–36.

Berardi, D., D. Calvanese, G. D. Giacomo, R. Hull et M. Mecella. 2005. «Automatic composition of transition-based semantic web services with messaging». In (Böhm *et al.*, 2005), p. 613–624.

Bianculli, D. et C. Ghezzi. 2007. «Monitoring conversational web services». In *IW-SOSWE '07: 2nd international workshop on Service oriented software engineering*. New York, NY, USA: ACM, p. 15–21.

Bidoit, N., S. Cerrito et V. Thion. 2004. «A first step towards modeling semistructured data in hybrid multimodal logic». *Journal of Applied Non-Classical Logics*, vol. 14, no. 4, p. 447–476.

Blackburn, P. 1993. *Modal Logic and Attribute Value Structures*. Synthese Library 229. Kluwer Academic Publishers, p. 19–65.

Blackburn, P. 2000. «Representation, reasoning, and relational structures: A hybrid logic manifesto». *Logic Journal of the IGPL*, vol. 8, no. 3, p. 339–365.

Blackburn, P., B. Gaiffe et M. Marx. 2003. «Variable-free reasoning on finite trees». In *Mathematics of Language 8*.

Boag, S., D. Chamberlin et M. F. Fern. 2005. «XQuery 1.0: An XML query language, W3C working draft».

Böhm, K., C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson et B. C. Ooi (dir. publ.). 2005. *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*. ACM.

Bojanczyk, M., A. Muscholl, T. Schwentick, L. Segoufin et C. David. 2006. «Two-variable logic on words with data». In (DBL, 2006a), p. 7–16.

Boneva, I. et J.-M. Talbot. 2004. «On complexity of model-checking for the TQL logic». In *3rd IFIP International Conference on Theoretical Computer Science*, p. 381–394.

Börger, E., E. Grädel et Y. Gurevich. 1997. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Berlin: Springer Verlag.

Boutaba, R., W. Golab, Y. Iraqi et B. S. Arnaud. 2004. «Lightpaths on demand: A web-services-based management system». *IEEE Communications Magazine*, p. 2–9.

Boutaba, R., S. Omari et A. P. S. Virk. 2001. «SELFCON: An architecture for self-configuration of networks». *Journal of Communications and Networks*, vol. 3, no. 4, p. 317–323.

Brambilla, M., A. Deutsch, L. Sui et V. Vianu. 2005. «The role of visual tools in a web application design and verification framework: A visual notation for LTL formulae». In Lowe, D. et M. Gaedke (dir. publ.), *Web Engineering, 5th International Conference, ICWE 2005, Sydney, Australia, July 27-29, 2005, Proceedings*, *Lecture Notes in Computer Science*, vol. 3579. Springer, p. 557–568.

Bratko, I. 2000. *Prolog Programming for Artificial Intelligence, 3rd edition.* Addison Wesley.

Bravetti, M., M. Núñez et G. Zavattaro (dir. publ.). 2006. *Web Services and Formal Methods, Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006, Proceedings*, *Lecture Notes in Computer Science*, vol. 4184. Berlin: Springer Verlag.

Bray, T., J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeay et J. Cowan. 2004. «Extensible markup language (XML) 1.1».

Brogi, A., C. Canal, E. Pimentel et A. Vallecillo. 2004. «Formalizing web service choreographies». *Electr. Notes Theor. Comput. Sci.*, vol. 105, p. 73–94.

Bultan, T., X. Fu, R. Hull et J. Su. 2003. «Conversation specification: a new approach to design and analysis of e-service composition». In *Proceedings of the 12th international conference on World Wide Web, WWW 2003, Budapest, Hungary, May 20-24, 2003.* p. 403–410.

Bultan, T., J. Su et X. Fu. 2006. «Analyzing conversations of web services». *IEEE Internet Computing*, vol. 10, no. 1, p. 18–25.

Cacciagrano, D., F. Corradini, R. Culmone et L. Vito. 2006. «Dynamic constraint-based invocation of web services». In (Bravetti *et al.*, 2006), p. 138–147.

Calcagno, C., L. Cardelli et A. D. Gordon. 2003. «Deciding validity in a spatial logic for trees». In *TLDI.* p. 62–73.

Cardelli, L. 2001. «Describing semistructured data». *SIGMOD Record*, vol. 30, no. 4, p. 80–85.

Cardelli, L., P. Gardner et G. Ghelli. 2002. «A spatial logic for querying graphs». In Widmayer, P., F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz et R. Conejo (dir. publ.), *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, *Lecture Notes in Computer Science*, vol. 2380. Springer, p. 597–610.

Cardelli, L. et G. Ghelli. 2001. «A query language based on the ambient logic.» In *Programming Languages and Systems, 10th European Symposium on Programming, ESOP 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, *Lecture Notes in Computer Science*, vol. 2028. Berlin: Springer Verlag, p. 1–22.

Cardelli, L. et G. Ghelli. 2004. «A query language for semistructured data based on the ambient logic». *Mathematical Structures in Computer Science, 14(3)*, p. 285–327.

Cardelli, L. et A. D. Gordon. 1998. «Mobile ambients.» In Nivat, M. (dir. publ.), *Foundations of Software Science and Computation Structure, First International Conference, FoSSaCS'98, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, *Lecture Notes in Computer Science*, vol. 1378. Berlin: Springer Verlag, p. 140–155.

Case, J. D., M. Fedor, M. L. Schoffstall et J. R. Davin. 1990. «A simple network management protocol. (RFC 1157)».

Cha, Y., T. Kwon, C. H. Kim et J. Choi. 2004. «Management information and model of GSMP network open interface». In (Aagesen *et al.*, 2004), p. 309–318.

Charatonik, W. et J.-M. Talbot. 2001. «The decidability of model checking mobile ambients». In Fribourg, L. (dir. publ.), *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings*, *Lecture Notes in Computer Science*, vol. 2142. Berlin: Springer Verlag, p. 339–354.

Cherkaoui, O., F. Bétouret et R. Deca. 2004. «On the transactional issues of the Netconf protocol». Tech. rep., Université du Québec à Montréal. Unpublished research document.

Christensen, E., F. Curbera, G. Meredith et S. Weerawarana. 2001. «Web services description language (WSDL) 1.1, W3C note».

Cimatti, A., E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani et A. Tacchella. 2002a. «NuSMV 2: An opensource tool for symbolic model checking». In Brinksma, E. et K. G. Larsen (dir. publ.), *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, *Lecture Notes in Computer Science*, vol. 2404. Berlin: Springer Verlag, p. 359–364.

Cisco Systems. 2002. «An OSS for packet networks». *Cisco Systems Packet Magazine*, vol. 14, no. 1, p. 25–27.

Cisco Systems. 2005. «Cisco SNMP object navigator». `http://tools.cisco.com/Support/SNMP/do/BrowseOID.do`.

Cisco Systems. 2006. «Configuring VTP». `http://www.cisco.com/en/US/products/hw\-/switches/ps708/products\_configuration\_guide\_chapter09186a008019f048.html`.

Clark, J. et S. DeRose. 1999. «XML path language (XPath) version 1.0, W3C recommendation».

Clarke, E. M., O. Grumberg et D. E. Long. 1994. «Model checking and abstraction». *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 5, p. 1512–1542.

Clarke, E. M., O. Grumberg et D. A. Peled. 2000. *Model Checking*. MIT Press.

Coen, C. S., P. Marinelli et F. Vitali. 2004. «Schemapath, a minimal extension to XML Schema for conditional constraints». In (Feldman *et al.*, 2004), p. 164–174.

Conforti, G. et G. Ghelli. 2004. «Decidability of freshness, undecidability of revelation». In Walukiewicz, I. (dir. publ.), *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, *Lecture Notes in Computer Science*, vol. 2987. Berlin: Springer Verlag, p. 105–120.

Constant, C., T. Jéron, H. Marchand et V. Rusu. 2007. «Integrating formal verification and conformance testing for reactive systems». *IEEE Trans. Software Eng.*, vol. 33, no. 8, p. 558–574.

Couch, A. L. et M. Gilfix. 1999. «It's elementary, dear Watson: Applying logic programming to convergent system management processes». In *Proceedings of the 13th Conference on Systems Administration (LISA-99), Seatlle, WA, November 7-12, 1999*. USENIX, p. 123–138.

Couch, A. L. et Y. Sun. 2003. «On the algebraic structure of convergence». In Brunner, M. et A. Keller (dir. publ.), *Self-Managing Distributed Systems, 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2003, Heidelberg, Germany, October 20-22, 2003, Proceedings*, *Lecture Notes in Computer Science*, vol. 2867. Berlin: Springer Verlag, p. 28–40.

Cridlig, V., H. Abdelnur, J. Bourdellon et R. State. 2005. «A Netconf network management suite: ENSUITE». In Magedanz, T., E. R. M. Madeira et P. Dini (dir. publ.), *IPOM*, *Lecture Notes in Computer Science*, vol. 3751. Berlin: Springer Verlag, p. 152–161.

Crubézy, M. 2002. «The Protégé axiom language and toolset ("PAL")». `http://protege.stanford.edu/plugins/paltabs/pal-documentation/index.html`

Damianou, N., N. Dulay, E. Lupu et M. Sloman. 2001. «The Ponder policy specification language». In (Sloman *et al.*, 2001), p. 18–38.

Davey, B. A. et H. A. Priestley. 1990. *Introduction to Lattices and Order*. Cambridge University Press.

de Vergara, J. L., V. Villagrá et J. Berrocal. 2002. «Semantic management: advantages of using an ontology-based management information meta-model». In *HP Openview University Association (HP-OVUA) Workshop*. `http://www.hpovua.org/PUBLICATIONS/PROCEEDINGS/9\_HPOVUAWS/`.

Deca, R., O. Cherkaoui et D. Puche. 2004. «A validation solution for network configuration». In *2nd Annual Conference on Communication Networks and Services Research (CNSR 2004), 19-21 May 2004, Fredericton, N.B., Canada*. IEEE Computer Society, p. 273–275.

Decker, G. et M. Weske. 2007. «Local enforceability in interaction Petri nets». In Alonso, G., P. Dadam et M. Rosemann (dir. publ.), *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings*, *Lecture Notes in Computer Science*, vol. 4714. Springer, p. 305–319.

Decker, G., J. M. Zaha et M. Dumas. 2006. «Execution semantics for service choreographies». In (Bravetti *et al.*, 2006), p. 163–177.

Demri, S., R. Lazic et D. Nowak. 2005. «On the freeze quantifier in constraint LTL: Decidability and complexity». In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*. IEEE Computer Society, p. 113–121.

Desai, N., R. Bradshaw et C. Lueninghoener. 2006. «Directing change using Bcfg2». In *Proceedings of the 20th Conference on Systems Administration (LISA 2006), Washington, D.C., USA, December 3-8, 2006*. USENIX, p. 215–220.

Deutsch, A., M. Marcus, L. Sui, V. Vianu et D. Zhou. 2005. «A verifier for interactive, data-driven web applications». In Özcan, F. (dir. publ.), *SIGMOD Conference*. ACM, p. 539–550.

Deutsch, A., L. Sui et V. Vianu. 2004. «Specification and verification of data-driven web services». In Deutsch, A. (dir. publ.), *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 14-16, 2004, Paris, France*. ACM, p. 71–82.

Deutsch, A., L. Sui, V. Vianu et D. Zhou. 2006. «Verification of communicating data-driven web services». In Vansummeren, S. (dir. publ.), *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, Maryland, USA*. ACM, p. 90–99.

Deutsch, A. et V. Tannen. 2001. «Containment and integrity constraints for XPath». In Lenzerini, M., D. Nardi, W. Nutt and D. Suciu (dir. publ.), *Proceedings of the 8th International Workshop on Knowledge Representation meets Databases (KRDB 2001), Rome, Italy, September 15, 2001*, vol. 45 de *CEUR Workshop Proceedings*. CEUR-WS.org.

Duan, Z., A. J. Bernstein, P. M. Lewis et S. Lu. 2004. «A model for abstract process specification, verification and composition». In (Aiello *et al.*, 2004), p. 232–241.

Emerson, E. et J. Halpern. 1985. «Decision procedures and expressiveness in the temporal logic of branching time». *Journal of Computer and System Sciences*, vol. 30, no. 1, p. 1–24.

Emerson, E. A. et E. M. Clarke. 1982. «Using branching time temporal logic to synthesize synchronization skeletons». *Sci. Comput. Program.*, vol. 2, no. 3, p. 241–266.

Enns, R. 2005. «Netconf configuration protocol, IETF working draft».

Eshuis, H. 2002. «Semantics and Verification of UML Activity Diagrams for Workflow Modelling». Thèse de doctorat, University of Twente, Twente, 240 p.

Fakhouri, S. A., G. S. Goldszmidt, M. H. Kalantar, J. A. Pershing et I. Gupta. 2001. «Gulfstream - a system for dynamic topology management in multi-domain server farms». In *2001 IEEE International Conference on Cluster Computing (CLUSTER 2001), 8-11 October 2001, Newport Beach, CA, USA*. IEEE Computer Society, p. 55–62.

Fallside, D. C. et P. Walmsley. 2004. «XML schema part 0: Primer second edition, W3C recommendation».

Feamster, N. et H. Balakrishnan. 2003. «Towards a logic for wide-area internet routing». *Computer Communication Review*, vol. 33, no. 4, p. 289–300.

Feldman, S. I., M. Uretsky, M. Najork et C. E. Wills (dir. publ.). 2004. *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*. ACM.

Ferrara, A. 2004. «Web services: A process algebra approach». *CoRR*, vol. cs.AI/0406055.

Filiot, E., J.-M. Talbot et S. Tison. 2007. «Satisfiability of a spatial logic with tree variables». In Duparc, J. et T. A. Henzinger (dir. publ.), *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, *Lecture Notes in Computer Science*, vol. 4646. Berlin: Springer Verlag, p. 130–145.

Foster, H., S. Uchitel, J. Magee et J. Kramer. 2003. «Model-based verification of web service compositions». In *ASE*. IEEE Computer Society, p. 152–163.

Foster, H., S. Uchitel, J. Magee et J. Kramer. 2006. «Model-based analysis of obligations in web service choreography». In *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), 19-25 February 2006, Guadeloupe, French Caribbean*. IEEE Computer Society, p. 149.

Franceschet, M. et M. de Rijke. 2006. «Model checking hybrid logics (with an application to semistructured data)». *J. Applied Logic*, vol. 4, no. 3, p. 279–304.

Franceschet, M. et E. Zimuel. 2005. «Modal logic and navigational XPath: an experimental comparison». In *Methods for Modalities*. p. 156–172.

Fu, X., T. Bultan et J. Su. 2003. «Conversation protocols: A formalism for specification and verification of reactive electronic services». In Ibarra, O. H. et Z. Dang (dir. publ.), *Implementation and Application of Automata, 8th International Conference, CIAA 2003, Santa Barbara, California, USA, July 16-18, 2003, Proceedings*, *Lecture Notes in Computer Science*, vol. 2759. Springer, p. 188–200.

Fu, X., T. Bultan et J. Su. 2004a. «Analysis of interacting BPEL web services». In (Feldman *et al.*, 2004), p. 621–630.

Fu, X., T. Bultan et J. Su. 2004b. «Conversation protocols: a formalism for specification and verification of reactive electronic services». *Theor. Comput. Sci.*, vol. 328, no. 1-2, p. 19–37.

Fu, X., T. Bultan et J. Su. 2004c. «Model checking XML manipulating software». In Avrunin, G. S. et G. Rothermel (dir. publ.), *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2004, Boston, Massachusetts, USA, July 11-14, 2004*. ACM, p. 252–262.

Gabbay, D., A. Pnueli, S. Shelah et J. Stavi. 1980. «On the temporal analysis of fairness». In *POPL*. p. 163–173.

Gaïti, D., G. Pujolle, M. Salaün et H. Zimmermann. 2005. «Autonomous network equipments». In Stavrakakis, I. et M. Smirnov (dir. publ.), *WAC, Lecture Notes in Computer Science*, vol. 3854. Berlin: Springer Verlag, p. 177–185.

Garey, M. R. et D. S. Johnson. 1979. *Computers and intractability, a guide to the theory of NP-completeness*. W. H. Freeman.

Gerede, C. E., K. Bhattacharya et J. Su. 2007. «Static analysis of business artifact-centric operational models». In *SOCA*. IEEE Computer Society, p. 133–140.

Gerede, C. E., R. Hull, O. H. Ibarra et J. Su. 2004. «Automated composition of e-services: lookaheads». In (Aiello *et al.*, 2004), p. 252–262.

Gerede, C. E. et J. Su. 2007. «Specification and verification of artifact behaviors in business process models». In Krämer, B. J., K.-J. Lin et P. Narasimhan (dir. publ.), *ICSOC, Lecture Notes in Computer Science*, vol. 4749. Berlin: Springer Verlag, p. 181–192.

Giunchiglia, F. et P. Traverso. 1999. «Planning as model checking». In Biundo, S. et M. Fox (dir. publ.), *ECP, Lecture Notes in Computer Science*, vol. 1809. Springer, p. 1–20.

Golab, W. et R. Boutaba. 2004. «Optical network reconfiguration using automated regression-based parameter value selection». In *Third International Conference on Networking, ICN 2004*.

Goranko, V. 2000. «Temporal logics with reference pointers and computation tree logics». *Journal of Applied Non-Classical Logics*, vol. 10, no. 3-4.

Gottlob, G., C. Koch et R. Pichler. 2002. «Efficient algorithms for processing XPath queries.» In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*. p. 95–106.

Governatori, G., Z. Milosevic et S. Sadiq. 2006. «Compliance checking between business processes and business contracts». In (DBL, 2006b), p. 221–232.

Grädel, E., P. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema et S. Weinstein. 2007. *Finite Model Theory and Its Applications*, vol. 13 de *Texts in Theoretical Computer Science. An EATCS Series*. Berlin: Springer Verlag.

Graml, T., R. Bracht et M. Spies. 2007. «Patterns of business rules to enable agile business processes». In (DBL, 2007), p. 365–378.

Greenfield, P., A. Fekete, J. Jang et D. Kuo. 2003. «Compensation is not enough». In *7th International Enterprise Distributed Object Computing Conference (EDOC 2003), 16-19 September 2003, Brisbane, Australia, Proceedings*. IEEE Computer Society, p. 232–239.

Greenfield, P., D. Kuo, S. Nepal et A. Fekete. 2005. «Consistency for web services applications». In (Böhm *et al.*, 2005), p. 1199–1203.

Guidi, C. 2007. "Formalizing languages for Service Oriented Computing." Thèse de doctorat, Università di Bologna, Bologne, 329 p.

Habib, M., M. Huchard et J. Spinrad. 1995. «A Linear Algorithm To Decompose Inheritance Graphs Into Modules». *Algorithmica*, vol. 13, no. 6, p. 573–591.

Hallé, S. 2005. «Formalismes logiques appliqués à la gestion de configurations réseaux». Mémoire de maîtrise, Université du Québec à Montréal, Montréal, 100 p.

Hallé, S., R. Deca, O. Cherkaoui et R. Villemaire. 2004. «Automated validation of service configuration on network devices». In Vicente, J. B. et D. Hutchison (dir. publ.), *Management of Multimedia Networks and Services: 7th IFIP/IEEE International Conference, MMNS 2004, San Diego, CA, USA, October 2004, Proceedings, Lecture Notes in Computer Science*, vol. 3271. Berlin: Springer Verlag, p. 176–188.

Hallé, S., R. Deca, O. Cherkaoui, R. Villemaire et D. Puche. 2005. «A formal validation model for the Netconf protocol». In *Utility Computing: 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2004, Davis, CA, USA, November 15-17, 2004. Proceedings, Lecture Notes in Computer Science*. Berlin: Springer Verlag, p. 147–158.

Hallé, S., R. Deca, O. Cherkaoui et R. Villemaire. 2006a. «Sequential Dependencies in Configuration Operations». In Krief, F. (dir. publ.), *GRES 2006: 7e Colloque Francophone de Gestion de Réseaux et de Services*, p. 112–123.

Hallé, S., R. Villemaire et O. Cherkaoui. 2006b. «CTL model checking for labelled tree queries». In *13th International Symposium on Temporal Representation and Reasoning (TIME 2006), 15-17 June 2006, Budapest, Hungary*. IEEE Computer Society, p. 27–35.

Hallé, S., E. Weenas, R. Villemaire et O. Cherkaoui. 2006c. «Self-configuration of Network Devices with Configuration Logic». In Vicente, J. B. et D. Hutchison (dir. publ.), *Autonomic Networking 2006, Proceedings, Lecture Notes in Computer Science*, vol. 4195. Berlin: Springer Verlag, p. 36–49.

Hallé, S., R. Deca, O. Cherkaoui, R. Villemaire et D. Puche. 2007a. «Modelling the Temporal Aspects of Network Configurations». In *Network Control and Engineering for QoS, Security and Mobility, IV, Fourth IFIP International Conference on Network Control and Engineering for QoS, Security and Mobility, Lannion, France, November 14-18, 2005. Proceedings*, *IFIP International Federation for Information Processing*. Berlin: Springer Verlag, p. 269–282.

Hallé, S., R. Villemaire, O. Cherkaoui et B. Ghandour. 2007b. «Model-checking data-aware temporal workflow properties with CTL-FO+». In (DBL, 2007), p. 267–278.

Hallé, S., R. Villemaire, O. Cherkaoui, J. Tremblay et B. Ghandour. 2007c. «Extending model checking to data-aware temporal properties of web services». In Dumas, M. et R. Heckel (dir. publ.), *Web Services and Formal Methods, Fourth International Workshop, WS-FM 2007, Brisbane, Australia, September 28-29, 2007, Proceedings*, *Lecture Notes in Computer Science*, vol. 4937. Berlin: Springer Verlag, p. 31–45.

Hallé, S. et R. Villemaire. 2008. «Satisfying a Fragment of XQuery by Branching-Time Reduction». In *15th International Symposium on Temporal Representation and Reasoning (TIME 2008), 16-18 June 2008, Montreal, Canada.* IEEE Computer Society, p. 72–76.

Hariri, S., L. Xue, H. Chen, M. Zhang, S. Pavuluri et S. Rao. 2003. «Autonomia: An autonomic computing environment». In *IPCCC*, p. 61–68.

Hartel, P. H. 2005. «A trace semantics for positive core XPath.» In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA.* p. 103–112.

Hidders, J. 2003. «Satisfiability of XPath expressions». In Lausen, G. et D. Suciu (dir. publ.), *Database Programming Languages, 9th International Workshop, DBPL 2003, Potsdam, Germany, September 6-8, 2003, Revised Papers*, *Lecture Notes in Computer Science*, vol. 2921. Berlin: Springer Verlag, p. 21–36.

Hinnelund, P. 2004. «Autonomic Computing». Mémoire de maîtrise, School of Computer Science and Engineering, Royal Institute of Engineering, Stockholm, 47 p.

Hinz, S., K. Schmidt et C. Stahl. 2005. «Transforming BPEL to Petri nets». In van der Aalst, W. M. P., B. Benatallah, F. Casati et F. Curbera (dir. publ.), *Business Process Management*, vol. 3649. p. 220–235.

Holzmann, G. J. 2003. *The SPIN Model Checker: Primer and Reference Manual.* Addison-Wesley Professional.

Hopcroft, J. E., R. Motwani et J. D. Ullman. 2000. *Introduction to Automata Theory, Languages, and Computation, Second Edition.* Addison Wesley.

Huth, M. R. A. et M. D. Ryan. 2000. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge, England: Cambridge University Press.

IEEE. 2003. «802.11Q: Virtual bridged local area networks standard». `http://standards.ieee.org/getieee802/download/802.1Q-2003.pdf`.

Jackson, D. 2000. «Alloy: A lightweight object modelling notation». Tech. Rep. 797, MIT Laboratory for Computer Science.

Jackson, D., I. Schechter et I. Shlyakhter. 2000. «Alcoa: the Alloy constraint analyzer». In *ICSE 2000, Proceedings of the 22nd International Conference on on Software Engineering, June 4-11, 2000, Limerick Ireland*, p. 730–733.

Jim, S. et H. Janelle. 2006. «Gartner predicts 2007: Align BPM and SOA initiative now to increase chances of becoming a leader in 2010». *Gartner Report*, p. 1–4.

Johnson, J. E., D. E. Langworthy, L. Lamport et F. H. Vogt. 2004. «Formal specification of a web services protocol». *Electr. Notes Theor. Comput. Sci.*, vol. 105, p. 147–158.

Kazhamiakin, R., M. Pistore et L. Santuari. 2006. «Analysis of communication models in web service compositions». In Carr, L., D. D. Roure, A. Iyengar, C. A. Goble et M. Dahlin (dir. publ.), *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23-26, 2006*. ACM, p. 267–276.

Keller, R. M. 2004. «Self-Configuring Services for Extensible Networks – A Routing-Integrated Approach». Thèse de doctorat, Swiss Federal Institute of Technology, Zürich, 216 p.

Koch, C. 2006. «On the complexity of nonrecursive XQuery and functional query languages on complex values.» *ACM Trans. Database Syst.*, vol. 31, no. 4, p. 1215–1256.

Konstantinou, A. V. 2003. «Towards Autonomic Networks». Thèse de doctorat, Columbia University, New York, 202 p.

Koshkina, M. et F. van Breugel. 2003. «Verification of business processes for web services». Tech. Rep. CS-2003-11, York University, Toronto, Canada.

Koshkina, M. et F. van Breugel. 2004. «Modelling and verifying web service orchestration by means of the concurrency workbench». *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 5, p. 1–10.

Kupferman, O. 1995. «Augmenting branching temporal logics with existential quantification over atomic propositions». In Wolper, P. (dir. publ.), *Computer Aided Verification, 7th International Conference, Liège, Belgium, July, 3-5, 1995, Proceedings, Lecture Notes in Computer Science*, vol. 939. Berlin: Springer Verlag, p. 325–338.

Kupferman, O. et M. Y. Vardi. 2006. «Memoryful branching-time logic». In (DBL, 2006a), p. 265–274.

Lakshmanan, L. V. S., G. Ramesh, H. Wang et Z. J. Zhao. 2004. «On testing satisfiability of tree pattern queries». In Nascimento, M. A., M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley et K. B. Schiefer (dir. publ.), *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*. Morgan Kaufmann, p. 120–131.

Le, F., S. Lee, T. Wong, H. S. Kim et D. Newcomb. 2006. «Minerals: Using data mining to detect router misconfigurations». Tech. Rep. CMU-CyLab-06-008, Carnegie Mellon University.

Li, Y. et H. Jagadish. 2003. «Compatibility determination in web services». In J. Gordijn, M. Janssen (dir. publ.), *ICEC 2003 E-Government and E-Services Workshop*, p. 7–14.

Lohmann, N., P. Massuthe, C. Stahl et D. Weinberg. 2006. «Analyzing interacting BPEL processes». In Dustdar, S., J. L. Fiadeiro et A. P. Sheth (dir. publ.), *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, *Lecture Notes in Computer Science*, vol. 4102. Berlin: Springer Verlag, p. 17–32.

Lucchi, R. et M. Mazzara. 2007. «A pi-calculus based semantics for WS-BPEL». *J. Log. Algebr. Program.*, vol. 70, no. 1, p. 96–118.

Marrero, W. 2005. «Using BDDs to decide CTL». In Halbwachs, N. et L. D. Zuck (dir. publ.), *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, *Lecture Notes in Computer Science*, vol. 3440. Berlin: Springer Verlag, p. 222–236.

Marx, M. 2003. «XPath and modal logics of finite DAG's». In Mayer, M. C. et F. Pirri (dir. publ.), *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2003, Rome, Italy, September 9-12, 2003. Proceedings*, *Lecture Notes in Computer Science*, vol. 2796. Berlin: Springer Verlag, p. 150–164.

Marx, M. 2004. «XPath with conditional axis relations». In Bertino, E., S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm et E. Ferrari (dir. publ.), *EDBT*, *Lecture Notes in Computer Science*, vol. 2992. Berlin: Springer Verlag, p. 477–494.

Marx, M. 2005a. «Conditional XPath». *ACM Trans. Database Syst.*, vol. 30, no. 4, p. 929–959.

Marx, M. 2005b. «First order paths in ordered trees». In Eiter, T. et L. Libkin (dir. publ.), *Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK, January 5-7, 2005, Proceedings*, *Lecture Notes in Computer Science*, vol. 3363. Berlin: Springer Verlag, p. 114–128.

Marx, M. et Y. Venema. 2007. *Local Variations on a Loose Theme: Modal Logic and Decidability*. Vol. 13 of (Grädel *et al.*, 2007), p. 371–429.

Melcher, B. et B. Mitchell. 2004. «Towards an autonomic framework: Self-configuring network services and developing autonomic applications». *Intel Technology Journal*, no. 4, p. 279–290.

Meredith, G. et S. Bjorg. 2003. «Contracts and types». *Commun. ACM*, vol. 46, no. 10, p. 41–47.

Miklau, G. et D. Suciu. 2002. «Containment and equivalence for an XPath fragment». In Popa, L. (dir. publ.), *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*. ACM, p. 65–76.

Moskewicz, M., C. Madigan, Y. Zhao, L. Zhang et S. Malik. 2001. «Chaff: Engineering an efficient SAT solver». In *Proceedings of the 39th Design Automation Conference*. p. 530–535.

Muscholl, A. et I. Walukiewicz. 2004. «An NP-complete fragment of LTL». In Calude, C., E. Calude et M. J. Dinneen (dir. publ.), *Developments in Language Theory*, *Lecture Notes in Computer Science*, vol. 3340. Berlin: Springer Verlag, p. 334–344.

Nakajima, S. 2004. «Model-checking of safety and security aspects in web service flows». In Koch, N., P. Fraternali et M. Wirsing (dir. publ.), *Web Engineering - 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004, Proceedings*, *Lecture Notes in Computer Science*, vol. 3140. Berlin: Springer Verlag, p. 488–501.

Narain, S. 2005. «Network configuration management via model finding». In *Proceedings of the 19th Conference on Systems Administration (LISA 2005), San Diego, USA, December 4-9, 2005*. USENIX, p. 155–168.

Narain, S., T. Cheng, B. A. Coan, V. Kaul, K. Parmeswaran et W. Stephens. 2003. «Building autonomic systems via configuration». In *Active Middleware Services*. IEEE Computer Society, p. 77–85.

Nieuwenhuis, R. et A. Oliveras. 2005. «Decision procedures for SAT, SAT modulo theories and beyond. the BarcelogicTools.» In Sutcliffe, G. et A. Voronkov (dir. publ.), *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*, *Lecture Notes in Computer Science*, vol. 3835. Berlin: Springer Verlag, p. 23–46.

Noy, N., M. Sintek, S. Decker, M. Crubézy, R. Fergerson et M. Musen. 2001. «Creating semantic web contents with Protégé-2000». *IEEE Intelligent Systems*, vol. 16, no. 2, p. 60–71.

Ouyang, C., E. Verbeek, W. M. P. van der Aalst, S. Breutel, M. Dumas et A. H. M. ter Hofstede. 2007. «Formal semantics and analysis of control flow in WS-BPEL». *Sci. Comput. Program.*, vol. 67, no. 2-3, p. 162–198.

Parashar, M. et S. Hariri. 2004. «Autonomic computing: An overview». In Banâtre, J.-P., P. Fradet, J.-L. Giavitto et O. Michel (dir. publ.), *Unconventional Programming Paradigms, International Workshop UPP 2004, Le Mont Saint Michel, France, September 15-17, 2004, Revised Selected and Invited Papers, Lecture Notes in Computer Science*, vol. 3566. Berlin: Springer Verlag, p. 257–269.

Parastatidis, S., J. Webber, S. Woodman, D. Kuo et P. Greenfield. 2005. «SOAP service description language (SSDL)». Tech. Rep. CS-TR-899, University of Newcastle, Newcastle upon Tyne.

Pesic, M. et W. M. van der Aalst. 2006. «A declarative approach for flexible business processes management». In Eder, J. et S. Dustdar (dir. publ.), *Business Process Management Workshops, Lecture Notes in Computer Science*, vol. 4103. Springer, p. 169–180.

Pistore, M., F. Barbon, P. Bertoli, D. Shaparau et P. Traverso. 2004a. «Planning and monitoring web service composition». In *Artificial Intelligence: Methodology, Systems, and Applications, 11th International Conference, AIMSA 2004, Varna, Bulgaria, September 2-4, 2004, Proceedings*. p. 106–115.

Pistore, M., M. Roveri et P. Busetta. 2004b. «Requirements-driven verification of web services». *Electr. Notes Theor. Comput. Sci.*, vol. 105, p. 95–108.

Pujolle, G. et D. Gaïti. 2004. «Intelligent routers and smart protocols». In (Aagesen *et al.*, 2004), p. 16–27.

Pujolle, G., O. Salvatori et J. Nozick. 2005. *Les Réseaux*. Paris: Eyrolles.

Rensink, A. 2006. «Model checking quantified computation tree logic». In Baier, C. et H. Hermanns (dir. publ.), *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings, Lecture Notes in Computer Science*, vol. 4137. Berlin: Springer Verlag, p. 110–125.

Rosen, E. C. et Y. Rekhter. 1999. «BGP/MPLS VPNs (RFC 2547)».

Rozier, K. Y. et M. Y. Vardi. 2007. «LTL satisfiability checking». In Bosnacki, D. et S. Edelkamp (dir. publ.), *Model Checking Software, 14th International SPIN Workshop, Berlin, Germany, July 1-3, 2007, Proceedings*, *Lecture Notes in Computer Science*, vol. 4595. Berlin: Springer Verlag, p. 149–167.

Schlingloff, B.-H., A. Martens et K. Schmidt. 2005. «Modeling and model checking web services». *Electr. Notes Theor. Comput. Sci.*, vol. 126, p. 3–26.

Schmidt, K. et C. Stahl. 2004. «A Petri net semantic for BPEL4WS validation and application». In *Proceedings of the 11th Workshop on Algorithms and Tools for Petri Nets (AWPN 04) / Ekkart Kindler (Ed.)*. Bericht tr-ri-04-251, Universität Paderborn, p. 1–6.

Schnoebelen, P. 2003. «The complexity of temporal logic model checking». *Advances in Modal Logic*, vol. 4, p. 393–436.

Sloman, M., J. Lobo et E. Lupu (dir. publ.). 2001. *Policies for Distributed Systems and Networks, International Workshop, POLICY 2001 Bristol, UK, January 29-31, 2001, Proceedings*, *Lecture Notes in Computer Science*, vol. 1995. Berlin: Springer Verlag.

Strassner, J. 1999. *Directory Enabled Networks*. Macmillan Technical Publishing.

Sun, Y. et A. Couch. 2006. «Dependency analysis in system administration (non publié)». `http://www.eecs.tufts.edu/~ysun/papers/dependency.pdf`.

ten Cate, B. et M. Franceschet. 2005. «On the complexity of hybrid logics with binders». In Ong, C.-H. L. (dir. publ.), *Computer Science Logic, 19th International Workshop, CSL 2005, 14th Annual Conference of the EACSL, Oxford, UK, August 22-25, 2005, Proceedings*, *Lecture Notes in Computer Science*, vol. 3634. Berlin: Springer Verlag, p. 339–354.

Trakhtenbrot, B. 1950. «Impossibility of an algorithm for the decision problem in finite classes». *Dok. Akad. Nauk SSSR*, vol. 70, p. 569–572.

Turner, K. J. 2005. «Formalising web services». In Wang, F. (dir. publ.), *Formal Techniques for Networked and Distributed Systems - FORTE 2005, 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5, 2005, Proceedings*, *Lecture Notes in Computer Science*, vol. 3731. Berlin: Springer Verlag, p. 473–488.

van Breugel, F. et M. Koshkina. 2006. «Models and verification of BPEL». Tech. rep., York University.

van der Aalst, W. M. P. et M. Pesic. 2006. «DecSerFlow: Towards a truly declarative service flow language». In (Bravetti *et al.*, 2006), p. 1–23.

Venzke, M. 2004. «Specifications using XQuery expressions on traces». *Electr. Notes Theor. Comput. Sci.*, vol. 105, p. 109–118.

Villemaire, R., S. Hallé et O. Cherkaoui. 2005a. «Configuration logic: A multi-site modal logic.» In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*. p. 131–137.

Villemaire, R., S. Hallé et O. Cherkaoui. 2005b. «A hierarchical logic for network configuration». In Sutcliffe, G. et A. Voronkov (dir. publ.), *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*, *Lecture Notes in Computer Science*, vol. 3835. Short paper proceedings.

Villemaire, R., S. Hallé, R. Deca et O. Cherkaoui. 2006. «Skolem functions and Herbrand universes for a tree generalization of first-order logic». In Gelbukh, A. et C. A. Reyes-García (dir. publ.), *Fifth Mexican International Conference on Artificial Intelligence, Special Session, November 13-17, Apizaco, Mexico*. IEEE Computer Society, p. 22–31.

Walton, C. D. 2004. «Model checking multi-agent web services». In *First International Semantic Web Services Symposium*. p. 8. `http://homepages.inf.ed.ac.uk/cdw/papers/aaai04/wsmodel.pdf`.

Wang, X., C. Sun, X. Liu et B. Xu. 2006. «Component composition based on web service and software architecture». In Shen, H. T., J. Li, M. Li, J. Ni et W. Wang (dir. publ.), *APWeb Workshops*, *Lecture Notes in Computer Science*, vol. 3842. Berlin: Springer Verlag, p. 987–990.

Warmer, J. B. et A. G. Kleppe. 1999. «OCL: The constraint language of the UML». *Journal of Object-Oriented Programming*, vol. 12, p. 10–13,28.

Warmer, J. B. et A. G. Kleppe. 2003. *The object constraint language*. Addison Wesley.

Woodman, S. J., D. J. Palmer, S. K. Shrivastava et S. M. Wheater. 2004. «Notations for the specification and verification of composite web services». In *8th International Enterprise Distributed Object Computing Conference (EDOC 2004), 20-24 September 2004, Monterey, California, USA, Proceedings*. IEEE Computer Society, p. 35–46.

Zaha, J. M., M. Dumas, A. ter Hofstede, A. Barros et G. Decker. 2006. «Service interaction modeling: Bridging global and local views». In (DBL, 2006b), p. 45–55.

Zhang, H. et M. E. Stickel. 2000. «Implementing the Davis-Putnam method.» *J. Autom. Reasoning*, vol. 24, no. 1/2, p. 277–296.

Zhao, H. et J. Shen. 2005. «OWL based web service component». *Computer Application*, vol. 25, no. 3, p. 634–636.