Formalisation d'une Approche Compositionnelle de Patrons de Propriétés

Djamila Baroudi[1], Philippe Dhaussy[2], Nait Bahloul Safia[3],
[1] UMAB, Univérsité de Mostaganem, Algérie,
baroudi.djamila@univ-mosta.dz
[2] Lab-STICC, UMR CNRS 6285 ENSTA Bretagne,France,
philippe.dhaussy@ensta-bretagne.fr
[3] Lab-LITIO, Université Oran 1, Ahmed BENBELLA,Algérie
bahloul.safia@univ-oran.dz

Abstract

Pour faciliter et encadrer l'expression des propriétés formelles, des alternatives aux logiques temporelles, telles que LTL ou CTL, ont été proposés, au prix de la réduction de l'expressivité. Celles-ci proposent des langages d'expression de propriétés basés sur des patrons de définition. Dans le but d'étendre l'expressivité des patrons proposés, nous avons conçu un langage de type DSL (*Domain Specific Language*) nommé CDL (*Context Description Language*). Ce langage permet une expression de propriétés de sûreté basée sur une extension des patrons proposés par Dwyer et al. Les propriétés sont traduites en automates observateurs et exploités par l'explorateur de modèles OBP (*Observer-Based Prover*). Pour pouvoir valider formellement la transformation des propriétés, nous avons formalisé, avec l'outil COQ, la sémantique des patrons dans une approche compositionnelle.
Mots Clés. Patrons de propriétés, observateurs, sémantique, composition, OBP/CDL, COQ.

1 Introduction

Un défi bien connu dans le domaine des méthodes formelles est d'améliorer leur intégration dans les processus industriels de développement logiciel. En particulier, les approches de vérification formelle de type $model - checking$ nécessitent que les exigences à vérifier sur des modèles de logiciels soient exprimées, sous la forme de propriétés formelles, dans des formalismes faciles d'accès et compréhensibles pour les ingénieurs.

Une difficulté de mise en œuvre d'une technique par model-checking est d'exprimer les exigences à vérifier de façon aisée. Les artefacts produits à travers les activités du processus de développement industriel (exigences, spécification, modèles de conception) ne sont pas directement exploitables par les outils de vérification. En particulier, les exigences sont en général disponibles sous une forme textuelle qui les rendent inexploitables sans une ré-écriture préalable plus formelle. Traditionnellement, les techniques de $model - checking$ implique de pouvoir exprimer les exigences à l'aide de formalismes de type logique temporelle comme par exemple LTL ou CTL. Bien que ces langages aient une grande expressivité et permettent une expression rigoureuse, ils ne sont pas faciles à utiliser par des ingénieurs lors de leurs activités

de vérification. En effet, dans un contexte industriel, une exigence peut référencer de nombreux états et évènements, liés à l'exécution du modèle ou de l'environnement, et est dépendante d'un historique d'exécution à prendre en compte au moment de sa vérification.

Une autre méthode, pour l'expression d'exigences et popularisée par le langage LUSTRE, repose sur la notion d'observateur [1]. Un observateur est un automate ou un programme transformant une propriété d'invariance (*toujours p*) en une vérification d'accessibilité (*la situation not p est elle accessible ?*). Si l'expression d'une propriété d'invariance est effectivement plus simple sous la forme d'un automate observateur, l'écriture d'un tel automate peut se révéler malgré tout complexe dans le cas d'une propriété elle-même complexe. Un automate observateur réaliste peut rapidement dépasser la vingtaine d'états. Il est donc nécessaire de faciliter l'expression des exigences avec des langages adéquats, permettant d'encadrer l'expression des propriétés au prix de la réduction de l'expressivité. De nombreux auteurs ont fait ce constat et certains [2, 3, 4] ont proposé de formuler les propriétés à l'aide de patrons de définition. Un patron est une structure syntaxique textuelle qui permet un mode d'expression d'une propriété plus proche des langages que les ingénieurs ont l'habitude de manipuler. Chaque propriété, basée sur un patron, peut être associée à un $scope$ qui précise les instants de l'exécution où la propriété est sensée être vérifiée ($Globally, Before, After, Between, After/Unless$). Dywer et al. [2] fournissent une sémantique formelle des combinaisons offertes par les patrons, par transformation en une formule logique exprimée dans différents langages comme LTL, CTL.

Dans le cadre d'expérimentations industrielles, nous avons souhaité étendre l'expressivité de ces patrons pour pouvoir exprimer plus aisément certaines exigences qui référencent un grand nombre d'évènements. Pour cela, nous avons proposé un langage prototype, de type DSL (*Domain Specific Language*), nommé CDL (*Context Description Language*) [5].

2 Motivations

CDL reprend les principaux concepts proposés par [2, 3, 4] et permet l'expression de propriétés temporisée de sûreté, d'invariance ou vivacité bornée, basée sur une extension des patrons proposés. Les patrons CDL[1] permettent d'exprimer, comme [2], des propriétés temporisées de réponse ($Response$), de prérequis ($Precedence$), d'absence et d'existence. Les propriétés font référence à des prédicats et des évènements détectables tels que des envois ou réceptions de valeurs entre processus du modèle, des changements d'état de processus ou de variables. Ces patrons (ou formes) de base ont été enrichis par des éléments optionnels ($options$) (*Pre-arity, Post-arity, Immediacy, Precedency, Nullity, Repeatability*) à l'aide d'annotations dans l'esprit de [3]. Des opérateurs an et all précisent respectivement si un évènement ou tous les évènements, ordonnés ($ordered$) ou non ($combined$), d'un ensemble d'évènements sont concernés par la propriété comme proposé par [6].

Pour mettre en œuvre la vérification de propriétés, notre outillage OBP (*Observer-Based Prover*) transforme les propriétés en automates observateurs non intrusifs dotés d'états de rejet ($reject$). Un observateur est construit de manière à encoder une propriété logique [7] et a pour rôle d'observer, partiellement et de manière non intrusive, les évènements significatifs survenant dans le modèle du système à valider. Lors de la simulation, il est composé, de manière synchrone,

[1] L'outil OBP et la documentation CDL sont accessibles librement sur : http://www.obpcdl.org.

au modèle à observer. La vérification de propriété consiste alors en une analyse d'accessibilité des états de rejet sur le graphe d'exploration qui résulte de la composition du modèle à valider et des observateurs.

Dans un travail précédent, une sémantique, dite de référence, des patrons CDL a été décrite et formalisée avec l'assistant COQ sous la forme de systèmes de transitions[2] représentant les automates observateurs qui sont générés par OBP. Nous précisons, en section 4, les principes adoptés pour la description de la sémantique de référence. Pour pouvoir maintenant implanter et valider les transformations de propriétés en des automates observateurs, nous souhaitons exprimer la sémantique des patrons CDL avec plus de facilité, c'est à dire avec une approche compositionnelle qui permet d'exprimer leur sémantique de manière extensible. Le but est de pouvoir, d'une part, construire, à l'aide d'opérateurs de composition, des observateurs complexes à partir de patrons élémentaires et d'éléments optionnels et, d'autre part, en déduire les algorithmes corrects de génération des automates pour notre outil OBP. Nous souhaitons disposer ainsi d'une base formelle pour enrichir l'expressivité des patrons et pouvoir valider formellement les transformations. Le travail, présenté ici, décrit les principes de cette approche. Le but est d'apporter ensuite la preuve de la préservation de la sémantique suite à la composition. Cette preuve sera basée, d'une part, sur une preuve de bisimulation entre les automates encodant la sémantique et le résultats de la composition patrons-options que nous avons implantée en COQ. D'autre part, une preuve par récurrence permettra de prouver que les opérateurs de composition préservent la sémantique lorsque la complexité des propriétés augmente.

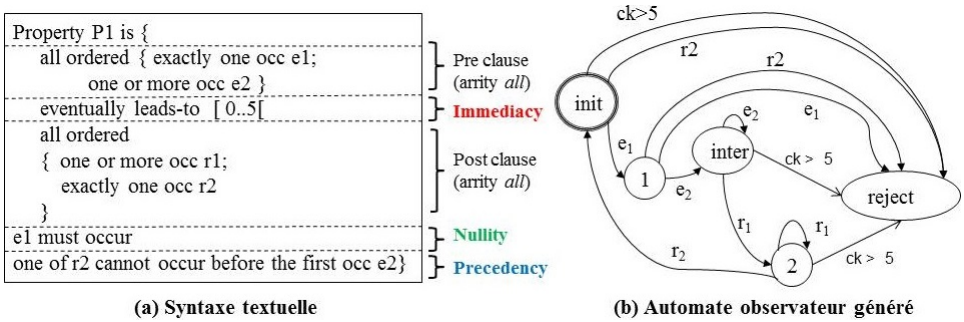
3 Un exemple de patron de propriétés CDL

En guise d'illustration, nous présentons un exemple simple de propriété ($P1$) basé sur un patron de type $Response$ de la forme : *all [*e_1*, *e_2*] leads-to [0..5[all [*r_1, r_2*].* Celui-ci définit une réponse (clause $Post$) constituée d'un ensemble d'évènements détectables $[r_1, r_2]$ qui doit avoir lieu suite à une action (clause Pre) décrite par un ensemble d'évènements [e_1, e_2]. La réponse doit intervenir dans un délai [0..5[, relatif au temps de simulation (ou d'exploration) du modèle. En examinant de plus près cette propriété, nous constatons qu'elle peut révéler plusieurs détails cachés qui nécessitent notre attention pour établir sa sémantique précise. Par exemple, si l'évènement e_1 est détecté plusieurs fois, l'ensemble d'évènements $[r_1, r_2]$ doivent-ils être considérés de manière répétitive ? ou encore : Un événement autre que e_1, e_2, r_1, ou r_2 peut-il se produire après e_1 et avant $[r_1, r_2]$? Dans [3], les auteurs abordent ces questions en étendant la notion de patron par l'ajout d'options évoquées précédemment.

Nous avons été inspirés par ce travail, en reprenant les options de [3], et en étendant la notion d'ensemble d'évènements (ordonné ou non). Les différentes formes d'options prévues dans CDL, avec leur variantes, sont les suivantes : *Pre-arity* (resp. *Post-arity*) détermine si un évènement e peut être détecté une fois (*exactly one occ e*) ou plusieurs fois (*one or more occ e*) dans la clause Pre (resp. $Post$). Dans ces clauses, des ensembles d'évènements peuvent être précisés pour spécifier si un seul évènement (an) de l'ensemble est à considérer ou tous les évènements (all) de l'ensemble, ordonnés ($ordered$) ou non ($combined$). $Immediacy$ détermine si un évènement, autre que ceux présents dans les clauses Pre et $Post$ peut survenir (*eventually*) ou

[2]Le code COQ (environ 1300 lignes de Gallina) de cette sémantique est disponible sous http://www.obpcdl.org.

non (*immediately*) avant un évènement de la clause $Post$. $Immediacy$ précise aussi le délai attendu de la réponse dans le cas de propriétés temporisées. $Nullity$ détermine si un évènement de la clause Pre doit arriver (*e must occur*) ou non (*e may never occur*) dans le délai spécifié par $Immediacy$. $Precedency$ détermine si un évènement de la clause $Post$ peut survenir (*one of list$_{post}$ may occur before the first one of list$_{pre}$*) ou non (*one of list$_{post}$ cannot occur before the first one of list$_{pre}$*) avant un évènement de la clause Pre. *list$_{pre}$* (resp. *list$_{post}$*) désigne ici une liste d'évènements pouvant potentiellement survenir dans la clause Pre (resp. $Post$). [3] considèrent l'option $Repeatability$ avec deux valeurs possibles : $true$ et $false$. Nous considérons ici que, pour des propriétés de sûreté, seule l'option à $true$ est pertinente. La figure 1 illustre la propriété précédente $P1$ sous la forme d'un patron *Response* exprimé avec le langage CDL. Dans cet exemple (figure 1.a), la clause Pre (resp. $Post$) référence deux événements e_1, e_2 (resp. une liste d'évènements $[r_1, r_2]$). La figure 1.b montre l'observateur qui est généré par l'outil OBP.



(a) Syntaxe textuelle **(b) Automate observateur généré**

Figure 1: $P1$: un exemple de propriété CDL de type Réponse

4 L'expression de la sémantique des propriétés CDL

Une sémantique des patrons CDL a été exprimée suivant des principes que nous décrivons dans la suite. Mais notons, tout d'abord, que compte tenu des différentes options et du nombre d'évènements pouvant intervenir dans les clauses Pre et $Post$ d'un patron, le nombre de combinaisons potentielles, dans l'expression de ce dernier, peut être très important. Par exemple, pour un patron de type Response qui référence un évènement dans chaque clause, Pre et $Post$, il existe 8 combinaisons possibles. Si le nombre d'évènements, dans chaque clause, Pre et $Post$, est de 2 (resp. 3 et 4), le nombre de combinaisons de patrons et d'options est de 96 (resp. 192 et 384), moins un certain nombre de combinaisons à supprimer car non significatives.

Pour décrire la sémantique, dite de référence, nous avons d'abord décrit, par un automate, toutes les combinaisons (Pre, $Post$ et options) disponibles et pour des complexités réduites, c'est à dire, pour un nombre d'évènements, inférieur à 3, dans les clauses Pre et $Post$. A partir de la sémantique de ces patrons de base, nous avons défini, par récurrence, la sémantique des patrons obtenus lorsque le nombre d'évènements pris en compte dans les clauses Pre ou $Post$ augmente. Par exemple, dans le cas de la propriété $P1$, figure 1, l'ajout d'un évènement dans la clause Pre ou $Post$ donne lieu à un nouvel automate observateur. Nous pouvons donc définir, de manière inductive sous la forme d'une fonction, le calcul qui permet de générer le nouvel automate pour chaque combinaison de base, c'est à dire pour les 96 combinaisons de base pour un nombre de 2 évènements dans chaque clause Pre et $Post$. Ce raisonnement par induction nous

permet en théorie d'établir formellement la sémantique de base des patrons CDL pour n'importe quel nombre d'évènements référencés dans les clauses Pre et $Post$.

Mais ce mode d'expression de la sémantique nous pose problème pour la raison suivante. Dans le cas où nous voudrions apporter une variante à la sémantique d'un patron et de ces options, nous devons modifier l'automate de base et la fonction inductive associés à chaque combinaison. Compte tenue de cette difficulté, nous optons pour une démarche compositionnelle et donc extensible pour exprimer la sémantique des patrons. L'idée est (1) d'exprimer la sémantique de référence de toutes les combinaisons CDL possibles (Pre, $Post$ et options) pour des complexités réduites (nombre d'évènements maximum égale à 2 dans Pre et $Post$), et les fonctions pour la définition par induction de la sémantique, (2) d'identifier un ensemble d'opérateurs élémentaires de composition, pour construire les automates observateurs à partir de la sémantique des éléments de base Pre, $Post$ et des options, (3) de démontrer la bisimulation du résultat de la composition avec la sémantique de référence pour un niveau de complexité réduite, et enfin (4) de démontrer par récurrence que la sémantique est préservée par la composition lorsque la complexité des propriétés augmente. Nous détaillons cette idée dans la suite du papier.

5 L'approche compositionnelle

Nous reprenons l'idée développée par [8] qui proposent d'exprimer la sémantique des patrons de [2] par une composition d'automates de Büchi décrivant séparément les patrons et les $scopes$. Mais nous l'adaptons pour l'appliquer à la composition de patrons de base (Pre et $Post$) et d'options CDL. Nous ne considérons le $scope$ qu'avec uniquement le type $Globally$, qui précise que la propriété exprimée doit tenir pendant toute l'exploration du modèle à valider. Nous présentons ici les règles de composition entre les patrons de base et les options. Pour cela, nous présentons les automates décrivant la sémantique des éléments Pre, $Post$ et des options constituant un patron CDL.

Sémantique des clauses Pre et $Post$. Les formes de bases (Pre et $Post$) des patrons sont décrits par des automates comme illustré figures 2.a et 2.b pour la propriété $P1$. L'état $init$ représente l'état initial. L'état $inter$ est le nœud de connexion entre les clauses Pre et $Post$. Les transitions bouclant sur les états représentent les arités du type *one or more*. La transition e_1 (figure 2.a) menant à l'état $reject$ représente l'arité du type *exactly one*.

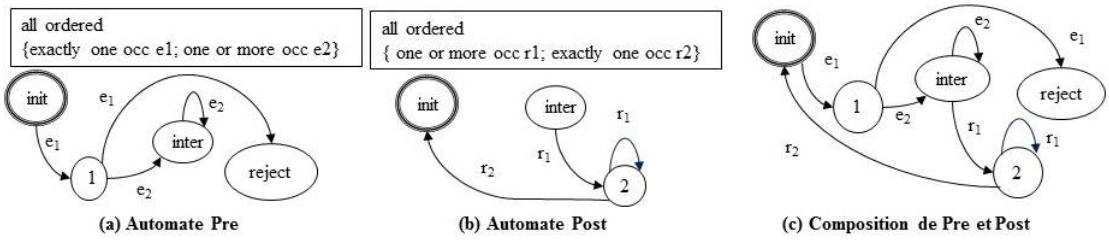


Figure 2: Automates pour les clauses Pre et Post de la propriété $P1$ et leur composition.

Sémantique des options. Les options $Immediacy$, $Precedency$, et $Nullity$ sont également décrits par des automates comme illustré figure 3 pour un patron de type $Response$. La figure 3.a

illustre un automate représentant l'option $Immediacy$ dans le cas de la variante $immediately$ (variante notée $Immediacy = true$). Le caractère '*' représente des évènements d'une liste autres que ceux apparaissant dans les clauses Pre et $Post$. L'idée est de pouvoir spécifier des évènements dont l'occurrence met en erreur ($reject$) la propriété. Cet automate référence aussi la contrainte d'horloge qui peut faire passer l'automate dans l'état $reject$. L'étiquette *"* or ck > 5"* signifie l'occurrence * ou le dépassement du délai 5.

La figure 3.b illustre un automate représentant l'option $Immediacy$ dans le cas de la variante $eventually$ (variante notée $Immediacy = false$). La figure 3.c illustre un automate représentant l'option $Nullity$ dans le cas de la variante *may never occur* (variante notée $Nullity = true$). *not a* représente l'absence de l'évènement a. Elle permet de spécifier l'évènement (ici a) qui est à prendre en compte dans l'option. Pour l'instant, cet évènement est unique mais cette spécification pourrait être étendue dans une version ultérieure à une liste d'évènements. La figure 3.d illustre un automate représentant l'option $Nullity$ dans le cas de la variante *must occur* (variante notée $Nullity = false$). La figure 3.e illustre un automate représentant l'option $Precedency$ dans le cas de la variante *may occur before* (variante notée $Precedency = true$). Cette option spécifie l'évènement r_2 qui peut être accepté avant e_2. La figure 3.f illustre un automate représentant l'option $Precedency$ dans le cas de la variante *cannot occur before* (variante notée $Precedency = false$). Cette option spécifie l'évènement r_2 qui met en erreur la propriété en cas d'occurrence de e_2.

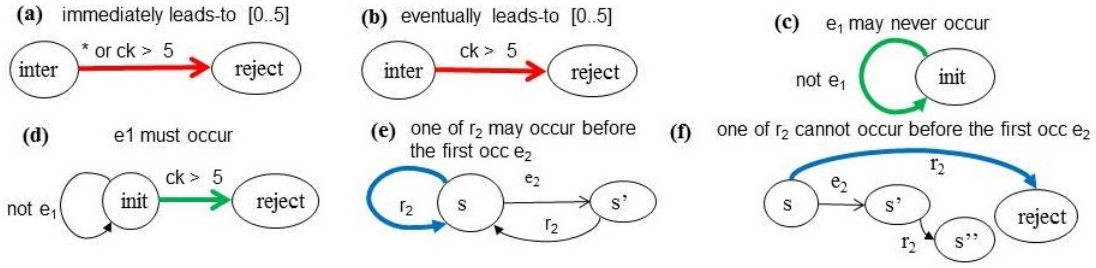


Figure 3: Automates pour les options applicables au patron $Response$ de la propriété $P1$.

Dans chaque automate représentant une option (figure 3), nous illustrons en gras une transition particulière qui participera à l'automate généré par les règles de composition décrite ci-dessous.

Règles informelles de la composition. Le principe de la composition repose sur plusieurs compositions élémentaires. Ces opérations de composition reviennent à l'union des transitions et des nœuds des différents automates Pre et $Post$ et l'ajout de transitions selon les options $Nullity$, $Immediacy$ et $Precedency$. Nous illustrons ce principe sur l'exemple des clauses Pre et $Post$ de l'exemple figure 2 et pour les options $Immediacy = false$, $Nullity = false$ et $Precedency = false$ illustrées respectivement dans les figures 3.b, 3.d et 3.f. Nous expliquons, ci-dessous, les règles appliquées, pour les options choisies. Le résultat est illustré figure 1.b.

La composition Pre et $Post$ est construite par une union des états et des transitions des automates Pre et $post$. Nous nommons A_p le résultat de cette composition et qui, pour l'exemple, est illustrée figure 2.c. La composition de $Nullity$ avec A_p est construite de la manière suivante.

Pour chaque état s de Pre, $inter$ et $reject$ exclus, précédent l'état source[3] de la transition e_1 (c'est à dire comportant l'étiquette e_1), une transition t, étiquetée avec la contrainte contenue dans l'option $immediacy$, est ajoutée à A_p telle que $src(t) = s$ et $tgt(t) = reject$. Dans le cas de l'exemple, la transition ($init$, $ck > 5$, $reject$) est ajoutée à A_p. Nous nommons A_n le résultat de cette composition. La composition de $Immediacy$ avec A_n est construite de la manière suivante. Pour chaque état s de $Post$, excepté $init$, une transition, étiquetée avec la contrainte contenue dans l'option $immediacy$, est ajoutée à A_n telle que $src(t) = s$ et $tgt(t) = reject$. Dans le cas de l'exemple, les transitions ($inter$, $ck > 5$, $reject$) et (2, $ck > 5$, $reject$) sont ajoutées à A_n. Nous nommons A_i le résultat de cette composition. La composition de $Precedency$ avec A_i est construite de la manière suivante. Pour chaque état s de Pre, excepté $inter$ et $reject$, qui n'ont pas en entrée la transition e_2, une transition t, étiquetée avec l'évènement r_2, est ajoutée à A_p telle que $src(t) = s$ et $tgt(t) = reject$. Dans le cas de l'exemple, les transitions ($init$, r_2, $reject$) et (1, r_2, $reject$) sont ajoutées à A_i. Nous nommons A_c le résultat de cette composition. Le résultat final A_c des opérations décrites précédemment est illustrée figure 1.b. Notons que la règle de composition des clauses Pre et $Post$ s'applique la première, et ensuite, dans le désordre, les règles associées à $Immediacy$, $Nullity$ et $Precedency$.

Les règles de composition ont été implantées sous COQ pour l'instant uniquement pour les patrons de type $Response$. Les règles génèrent la description des systèmes de transitions représentant les propriétés spécifiées. Dans la suite de notre travail, nous devons poursuivre par l'implantation des preuves qui démontrent, d'une part, la bisimulation entre les systèmes de transitions générés par les opérateurs de composition avec les systèmes de transitions décrivant la sémantique des patrons, pour un niveau de complexité donné. D'autre part, la préservation de la sémantique, lors de l'augmentation de la complexité des patrons, doit être démontrée.

Pour démontrer la bisimulation, nous implantons les fonctions nécessaires à la vérification des règles définissant la bisimulation entre 2 systèmes de transitions. Pour démontrer la preuve de préservation par récurrence, le principe est le suivant. Soit $\mathcal{C}(k_{pre}, k_{post})$, représentant un niveau de complexité pour une propriété spécifiée avec les clauses Pre et $Post$ référençant respectivement k_{pre} et k_{post} évènements. Soit $\mathcal{C}(k_{pre} + 1, k_{post})$ (resp. $\mathcal{C}(k_{pre}, k_{post}$+1)), représentant un niveau de complexité pour cette propriété avec la clause Pre (resp. $Post$) référençant $k_{pre} + 1$ (resp. $k_{post} + 1$) évènements. Nous devons démontrer que l'ajout de l'évènement supplémentaire, soit dans Pre, soit dans $Post$, donnera lieu, lors de la composition, à la génération de toutes les transitions qui auraient été ajoutées dans les systèmes de transitions de la sémantique de référence. Ce qui revient à démontrer que la bisimulation est préservée. Cette démonstration doit s'effectuée pour tous les types de patrons ($Response$, $Precedence$, $Absence$ et $Existence$) et toutes les combinaisons avec les options. Dans la suite de notre travail, nous implanterons les règles démontrant cette préservation.

6 Conclusion et travaux futurs

Une sémantique des patrons CDL a été présentée, basée sur une approche compositionnelle. Nous formalisons la composition d'éléments de base constitutifs des patrons de propriétés

[3] On note $src(t)$ (resp. tgt (t)): état source (resp. cible) de la transition t.

avec des éléments optionnels qui viennent enrichir les patrons. Des règles de composition, implantées en COQ pour les patrons de type $Response$, permettent de générer des systèmes de transitions bisimilaires aux systèmes de transitions de référence, décrivant la sémantique des patrons. L'implantation reste à être finalisée pour les autres patrons de type $Precedence$, $Absence$ et $Existence$. Les fonctions COQ prouvant la bisimulation sont encore à valider, Aussi, les fonctions permettant d'exécuter la preuve par récurrence sont encore à implanter. Ce travail nous permet de disposer d'un cadre formel de construction de propriétés plus complexes à partir d'éléments de base et pouvoir valider formellement les transformations. Par la suite, nous pourrons donc continuer à concevoir des extensions de CDL en rajoutant des options supplémentaires. Le principe de construction d'un observateur par composition peut aussi s'étendre à la composition de plusieurs observateurs afin de pouvoir exprimer des propriétés plus complexes basées sur les propriétés élémentaires. Compte tenu du type d'exigences que nous avons à traiter dans certaines applications industrielles, notre objectif est d'expérimenter ces extensions pour mener des actions de validation formelle sur les modèles industriels.

References

[1] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*, pages 83–96, Twente, June 1993. Workshops in Computing, Springer Verlag.

[2] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *21st Int. Conf. on Software Engineering*, pages 411–420. IEEE Computer Society Press, 1999.

[3] R. Smith, G.S. Avrunin, L. Clarke, and L. Osterweil. Propel: An approach supporting property elucidation. In *24st Int. Conf. on Software Engineering(ICSE02), St Louis, MO, USA*, pages 11–21. ACM Press, 2002.

[4] S. Konrad and B. Cheng. Real-time specification patterns. In *27th Int. Conf. on Software Engineering (ICSE05), St Louis, MO, USA*, 2005.

[5] Philippe Dhaussy, Frédéric Boniol, Jean-Charles Roger, and Luka Leroux. Improving model checking with context modelling. *Adv. Software Engineering*, 2012:547157:1–547157:13, 2012.

[6] Wil Janssen, Radu Mateescu, Sjouke Mauw, Peter Fennema, and Petra Van Der Stappen. Model checking for managers. In *SPIN*, pages 92–107, 1999.

[7] L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen. The power of reachability testing for timed automata. In *Proc. 18th Conf. Found. of Software Technology and Theor. Comp. Sci. (FST&TCS'98), Chennai, India, Dec. 1998*, volume 1530, pages 245–256. Springer, 1998.

[8] S. Taha, J. Julliand, F. Dadeau, K. C. Castillos, and B Kanso. A compositional automata-based semantics and preserving transformation rules for testing property patterns. In *Formal Aspects of Computing*, volume 27, 2015.