tel-00821836, version 1 - 13 May 2013

Université Paris-Sud

École Doctorale Informatique Paris-Sud

Laboratoire de Recherche en Informatique

Thèse de Doctorat

Semantics-Based Testing for Circus

Synthèse

Présentée par :
Abderrahmane Feliachi
pour l'obtention du
Doctorat de l'université Paris-Sud XI

Jury

Pr. Rob Hierons	Brunel University, UK	Rapporteur
Pr. Stephan Merz	INRIA Nancy	Rapporteur
Dr. Ana Cavalcanti	University of York, UK	Examinatrice
Pr. Pascale Le Gall	Université d'Evry	Examinatrice
Pr. Claude Marché	INRIA Saclay	Examinateur
Pr. Marie-Claude Gaudel	Université Paris-Sud XI	Directrice de thèse
Pr. Burkhart Wolff	Université Paris-Sud XI	Co-Encadrant

2009 – 2012

tel-00821836, version 1 - 13 May 2013

Resumé

Le travail présenté dans cette thèse est une contribution aux méthodes formelles de spécification et de vérification. Les spécifications formelles sont utilisées pour décrire un logiciel, ou plus généralement un système, d'une manière mathématique sans ambiguïté. Des techniques de vérification formelle sont définies sur la base de ces spécifications afin d'assurer l'exactitude d'un système donné. Cependant, les méthodes formelles ne sont souvent pas pratique et facile à utiliser dans des systèmes réels. L'une des raisons est que de nombreux formalismes de spécification ne sont pas assez riches pour couvrir à la fois les exigences orientées données et orientées comportement. Certains langages de spécification ont été proposés pour couvrir ce genre d'exigences. Le langage *Circus* se distingue parmi ces langues par une syntaxe et une sémantique riche et complètement intégrées.

L'objectif de cette thèse est de fournir un cadre formel pour la spécification et la vérification de systèmes complexes. Les spécifications sont écrites en *Circus* et la vérification est effectuée soit par des tests ou par des preuves de théorèmes. Des environnement similaires de spécification et de vérification ont déjà été proposées dans la littérature. Une spécificité de notre approche est de combiner des preuves de théorème avec la génération de test. En outre, la plupart des méthodes de génération de tests sont basés sur une caractérisation syntaxique des langages étudiés. Notre environnement est différent car il est basé sur la sémantique dénotationnelle et opérationnelle de *Circus*. L'assistant de preuves Isabelle/HOL constitue la plateforme formelle au dessus de laquelle nous avons construit notre environnement de spécification et de vérification.

La première contribution principale de notre travail est l'environnement formel de spécification et de preuve Isabelle/*Circus*, basé sur la sémantique dénotationnelle de *Circus*. Sur la base de Isabelle/HOL nous avons fourni une intégration vérifiée de UTP, la base de la sémantique de *Circus*. Cette intégration est utilisée pour formaliser la sémantique dénotationnelle du langage *Circus*. L'environnement Isabelle/*Circus* associe à cette sémantique des outils de *parsing* qui aident à écrire des spécifications *Circus*. Le support de preuve de Isabelle/HOL peut être utilisée directement pour raisonner sur ces spécifications grâce à la représentation *superficielle* de la sémantique (shallow embedding). Nous présentons une application de l'environnement à des preuves de raffinement sur des processus *Circus* (impliquant à la

fois des données et des aspects comportementaux). Une version de Isabelle/*Circus* est déjà publiée dans le cadre de l'AFP (archives de preuves formelles) d'Isabelle [1].

La deuxième contribution est l'environnement de test *CirTA* construit au dessus de Isabelle/*Circus*. Cet environnement fournit deux tactiques de génération de tests symboliques qui permettent la vérification de deux notions de raffinement : l'inclusion des traces et la réduction de blocages. L'environnement est basé sur une formalisation symbolique de la sémantique opérationnelle de *Circus* avec Isabelle/*Circus*. Plusieurs définitions symboliques et tactiques de génération de test sont définis dans le cadre de *CirTA*. L'infrastructure formelle permet de représenter explicitement les théories de test ainsi que les hypothèses de sélection de test. Des techniques de preuve et de calculs symboliques sont la base des tactiques de génération de test.

L'environnement de génération de test a été utilisé dans une étude de cas pour tester un système existant de contrôle de message. Une spécification du système est écrite en *Circus*, et est utilisé pour générer des tests pour les deux relations de conformité définies pour *Circus*. Les tests sont ensuite compilées sous forme de méthodes de test JUnit qui sont ensuite exécutées sur une implémentation Java du système étudié.

Cette thèse est une importante étape vers, d'une part, le développement d'outils de tests sophistiqués utilisant des preuves de propriétés de la spécification pour réaliser et justifier des stratégies de test et, d'autre part, l'intégration de test et de preuves dans des développements logiciels formellement vérifiées.

Mots clefs

Spécification et vérification de logiciels, preuves de théorèmes, test formel.

1. `http://afp.sourceforge.net/entries/Circus.shtml`

tel-00821836, version 1 - 13 May 2013

tel-00821836, version 1 - 13 May 2013

CHAPITRE 1

Introduction

Les systèmes logiciels et matériels ne cessent d'augmenter en taille et en complexité. En conséquence, le nombre des erreurs augmente également dans ces systèmes. L'activité de test vise à trouver ces erreurs. Le test formel désigne l'utilisation de méthodes et techniques formelles dans le test. Les méthodes formelles comprennent les langages formel, ainsi que les techniques et outils pour la spécification et la vérification de systèmes logiciels et matériels.

Dans le test formel, différents formalismes peuvent être utilisés pour décrire formellement le comportement ("correct") désiré du système. Les langages de spécification sont largement utilisés comme formalismes de description dans les techniques de test basées sur les spécifications. Afin de couvrir les différents aspects du comportement du système, différents langages de spécification peuvent être utilisés. Certains langages orientés données portent essentiellement sur la représentation des données complexes (Z par exemple). Autres langages portent davantage sur les aspects comportementaux complexes du système (CCS et CSP par exemple). Comme les systèmes réels combinent des données et des comportements complexes, certaines langages ont été proposés pour couvrir ces deux aspects.

C'est le cas de la langage *Circus*, qui combine des représentations de données à la Z avec une algèbre de processus à la CSP. Le langage définit aussi une notion particulière de raffinement. Le langage est décrit avec des sémantique dénotationnelle et opérationnelle bien établies. Une théorie de test formel est défini pour *Circus* sur la base de sa sémantique.

Une façon naturelle d'implémenter les méthodes et outils formels est l'utilisation des assistants de preuves. Ces assistants de preuve offrent une base formelle "de confiance" pour tout autre développement formel. Un des assistants de preuve les plus connus est l'assistant de preuve Isabelle/HOL pour les logiques d'ordre supérieur.

1.1 Motivations

Dijkstra a déclaré que le test "peut être utilisé pour indiquer la présence des erreurs, mais jamais de montrer leur absence". Un des objectifs du test formel est de montrer que cette thèse n'est pas toujours vrai. En fait, si le test couvre tous les comportements possibles du système et, sous certaines hypothèses, il peut garantir l'absence des erreurs. Cette méthode de test est également appelée le test exhaustif. Il combine les techniques de test et de preuve pour définir une notion de correction c-à-d l'absence d'erreurs.

Un système est dit correct par rapport à une spécification s'il se comporte de manière satisfaisante par rapport à cette spécification. On dit alors que le système est *conforme* à la spécification donnée selon une *relation de conformité*. Un théorème de test énonce que, pour des cas de test et des hypothèses test donnés, le système se comporte correctement par rapport à une spécification donnée.

L'utilisation d'assistant de preuve rend possible la description des théorèmes et des hypothèses de test, et fournit une base solide pour les stratégies de test. Les démonstrateurs de théorèmes offrent aussi un puissant fondement logique formelle. Ceci peut être utilisé pour formaliser les langages de spécification riches tels que *Circus*.

Dans cette thèse, nous explorons la combinaison de ces deux idées c-à-d l'utilisation d'un assistant de preuve comme base pour spécifier et tester des systèmes complexes.

1.2 Propositions et Contributions

Le but de ce travail est de fournir un cadre formel et complet pour la spécification, le test et le raisonnement sur des systèmes complexes. On a choisi le langage de spécification *Circus* pour couvrir les différents aspects de ces systèmes. On croit que *Circus* fournit une combinaison exemplaire de deux aspects de données et de comportements complexes. Notre approche de test est basée sur l'assistant de preuve Isabelle/HOL. Il est utilisé pour formaliser la sémantique de *Circus* et sa théorie de test. Il est également utilisé pour générer des tests exhaustifs à partir de spécifications écrites en *Circus*.

Dans ce travail, nous avons voulu une approche basée sur la sémantique pour la spécification, la vérification et le test. Traditionnellement, les langages de spécification sont définis formellement avec une sémantique et des propriétés bien établies. Certains outils peuvent être développés pour ces langages, couvrant d'analyse, la vérification de type, l'animation ou la génération de test. Le problème est que, dans la plupart des cas, il existe un écart entre les définitions formelles "sur papier" et les outils, souvent, non vérifiés. De plus, les outils sont généralement basés sur la syntaxe, avec très peu de support de la sémantique. Dans notre travail, l'écart entre la théorie et

tel-00821836, version 1 - 13 May 2013

les outils est extrêmement réduit. En utilisant l'environnement formel Isabelle/HOL, toutes les définitions formelles peuvent être automatisées et mises en œuvre. Dans ce cadre formel, il est également possible d'associer une sémantique à la syntaxe. Ainsi, il est possible d'effectuer un raisonnement plus élaboré sur les spécifications.

Cette thèse présente un travail pionnier dans le domaine des environnements de spécification, vérification formelle et de test. La représentation formelle basée sur la sémantique d'un langage de spécification complexe et riche dans un démonstrateur est une importante contribution. Les contributions principales peuvent être résumées en cinq parties :

- Une première contribution de ce travail est l'intégration *deep* de UTP, la base de la sémantique *Circus*, dans Isabelle/HOL.

- La représentation de la sémantique dénotationnelle de *Circus* au-dessus de UTP est également une contribution importante de ce travail, avec la définition de plusieurs notions clés comme les variables d'état, la portée, les canaux de communication et des ensembles de noms.

- L'environnement de vérification Isabelle/*Circus*, basé sur cette représentation, est un environnement de preuve pour *Circus* qui peut être utilisé pour raisonner sur des spécifications *Circus*. Un exercice intéressant est exploré, où l'environnement est étendu avec des règles de raffinement sur des processus *Circus*.

- Une autre contribution importante de ce travail est la représentation de la sémantique opérationnelle et la théorie de test de *Circus*. Les règles de la sémantique opérationnelle sont définies sur des actions *Circus* et quelques règles inversées très utiles sont déduites. La théorie de test contient les définitions de nombreuses notions utiles (par exemple des traces et des initiales) ainsi que quelques théorèmes de test. Ces définitions sont basées sur une représentation et une manipulation symbolique *shallow* basées sur l'infrastructure symboliques d'Isabelle. Des tactiques de génération automatique de tests sont définis pour les processus *Circus*.

- La dernière contribution est l'application de l'environnement de test à une étude de cas réaliste. L'application cible est un module de surveillance des messages d'un système de télésurveillance médicale. Une spécification du système est écrit en *Circus*, puis est utilisée comme base pour la génération de tests. Les tests générés sont exécutés, en utilisant JUnit sur l'implémentation Java du système.

tel-00821836, version 1 - 13 May 2013

1.3 Plan

L'objectif de cette thèse est de présenter une approche de test basée sur la sémantique et un environnement de test formel pour *Circus*. Ceci est réalisé en deux étapes, couverts par deux contributions principales. La première étape est la définition d'un environnement de spécification formelle et de vérification pour *Circus*. La deuxième étape, présente un framework de test pour *Circus* au-dessus de l'environnement de spécification. La thèse est organisée autour de ces deux contributions principales. Elle est composée de cinq chapitres (2-6) organisés comme suit :

Chapitre 2 : Les méthodes formelles et le test

Dans ce premier chapitre, nous introduisons certains aspects de la complémentarité entre les méthodes formelles et le test, avec une attention particulière pour les tests formels. Après quelques informations générales sur les théories de test, nous discutons de l'état de l'art des principales techniques de tests formels. La relation entre les techniques de vérification traditionnels et le test est discutée comme un type particulier de complémentarité. En particulier, nous étudions quelques model-checkers et assistants de preuve utilisés pour la génération de tests.

Chapitre 3 : Contexte

Nous présentons dans ce chapitre les deux principaux concepts sur lesquels la thèse est basée. Le premier concept est le langage de spécification formelle *Circus* et le second est l'assistant de preuve Isabelle/HOL. Dans la première partie de ce chapitre, nous présentons la syntaxe et la sémantique de *Circus*. Cette sémantique est basée sur un cadre sémantique appelé UTP (Unifying Theories of Programming), qui est également introduit. La première partie du chapitre est consacrée à l'assistant de preuve générique Isabelle. Nous nous concentrons sur l'extension de l'assistant avec des logiques d'ordre supérieur appelée Isabelle/HOL. Nous présentons des extensions de Isabelle/HOL qui sont liées à notre travail : HOL-Z, HOL-CSP et HOL-TestGen. Enfin, le chapitre résume quelques considérations générales qui guideront les chapitres suivants.

Chapitre 4 : Isabelle/Circus

Ce chapitre présente la première contribution de la thèse : l'environnement de spécification et de vérification Isabelle/*Circus*. Dans la première partie de ce chapitre, nous discutons des choix conceptuels de la représentation. Ensuite, nous développons notre intégration d'une partie substantielle de UTP au-dessus d'Isabelle/HOL. La définition de la sémantique dénotationnelle de *Circus* est introduite sur la base de l'intégration de UTP. La sémantique complète, enrichie avec une interface de haut

tel-00821836, version 1 - 13 May 2013

niveau, constitue notre environnement de spécification Isabelle/*Circus*. Cet environnement permet de raisonner sur des spécifications *Circus* en utilisant les techniques de preuves fournit par l'assistant.

Nous présentons une première application de l'environnement Isabelle/*Circus* aux preuves de raffinement. Nous introduisons la notion de raffinement de données et d'action dans *Circus* ainsi que quelques lois générales de raffinement. Un exemple de preuve de raffinement est illustré sur un petit exemple.

Chapitre 5 : Test basé sur la sémantique

La deuxième contribution de cette thèse est expliquée dans ce chapitre. Sur la base de l'environnement Isabelle/*Circus*, le système de test *CirTA* est défini pour les spécifications *Circus*. Nous présentons d'abord notre approche de génération de tests basée sur un assistant de preuve inspirée de HOL-TestGen. Une partie importante de ce chapitre est ensuite consacré à la sémantique opérationnelle de *Circus*. Notre représentation des définitions et des manipulations symboliques est également discutée. La méthode de test symbolique est expliquée et les principales définitions sont introduites. Un petit exemple est utilisé pour illustrer les tactiques de génération de tests présentées dans ce chapitre.

Chapitre 6 : Étude de cas

Le dernier chapitre illustre une étude de cas de l'application du système de test proposé. Un spécification *Circus* d'un système de suivi de message existant est fournie et est utilisée pour la génération de tests. Des procédures de test JUnit sont automatiquement extraites des cas de test générés à l'aide de *CirTA*. Ces procédures de test sont exécutées sur l'implémentation Java existante du système de surveillance.

tel-00821836, version 1 - 13 May 2013

tel-00821836, version 1 - 13 May 2013

CHAPITRE 2

Les Méthodes Formelles et le Test

Les méthodes formelles et le test de logiciels ne sont pas de très vieux amis : ils ont même été considérés comme des rivaux. Depuis plusieurs années, ils avaient échoué à converger. Entre les années soixante-dix et les années quatre-vingt, quelques propositions pour l'utilisation de ces deux approches en même temps ont vu le jour. Depuis lors, les méthodes formelles et le test de logiciels sont de plus en plus intégré dans ce qui est maintenant connu comme *le test formel*.

Les méthodes formelles peuvent être définies comme les langages, techniques et outils pour la spécification et la vérification de systèmes logiciels et matériels. Ceci englobe essentiellement deux types de méthodes formelles : des langages de spécification formelle et les techniques de vérification formelle. Nous croyons que les tests formels est l'une des techniques de vérification formelle. Cependant, nous utilisons le mot vérification dans ce chapitre pour parler des techniques traditionnelles de vérification formelle et surtout par model checking et preuve de théorèmes.

Le Test formel rassemble tous les impacts et les contributions que les méthodes formelles peuvent offrir au test. Ces contributions peuvent être distingués en deux catégories en fonction de leur objectif initial. Tout d'abord, les définitions et techniques formelles introduites dans le cadre du test, y compris les théories de test, la génération de tests, les oracles de test et les définitions des verdicts. La deuxième catégorie comprend les méthodes et techniques formelles développées à d'autres fins, mais réutilisées pour le test. Cela inclut les langages de spécification formelle essentiellement utilisées comme base pour les tests et les techniques de vérification formelle réutilisés pour le test.

Ce chapitre donne un bref aperçu de la relation entre les méthodes formelles et le test. Les méthodes formelles considérées dans ce chapitre sont de deux catégories : langages de spécification formelle et méthodes de vérification formelles (traditionnelles). Un troisième type de méthodes formelles est introduit dans ce chapitre qui est les méthodes de test formel. Ceux-ci fournissent un environnement de test formel contenant des définitions et la génération de tests et les techniques d'évaluation des résultats (les verdicts et les oracles).

Certaines relations entre le test formel et les autres types de méthodes formelles

sont introduits dans ce chapitre. Tout d'abord, certaines méthodes de spécification formelle sont utilisés comme base pour des définitions et des techniques de génération de tests. Ces formalismes de spécification sont utilisés pour décrire les données ou les aspects comportementaux (ou les deux) des systèmes. Un autre type d'interactions apparaît lorsque certaines méthodes et techniques de vérification formelles sont déviés pour être utilisé pour le test. Des exemples de cette interaction sont les model checkers et les assistants de preuve utilisés pour le test.

tel-00821836, version 1 - 13 May 2013

tel-00821836, version 1 - 13 May 2013

CHAPITRE 3

Contexte

L'objectif principal de cette thèse est de fournir un cadre formel pour *Circus* sur la base d'Isabelle/HOL. Ce chapitre présente le contexte et les bases de cet environnement : le langage *Circus* et l'assistant de preuve Isabelle/HOL. *Circus* est un langage de spécification formelle bien défini, couvrant plusieurs aspects de spécifications des systèmes. Isabelle/HOL est un puissant cadre formel de raisonnement logique et de preuve de théorèmes et de nombreuses autres applications.

La première section présente *Circus*, y compris sa syntaxe, sémantique, son fondement UTP et ses théories de raffinement et de test. La deuxième section donne un aperçu de Isabelle/HOL et de certaines de ses notions importantes utilisées dans cette thèse.

3.1 Le Langage Circus

Circus est un langage de spécification formelle qui intègre les notions d'état et de types de données complexes dans un style similaire à Z et des processus parallèles communicant inspirés de CSP. De Z, le langage hérite la notion de schéma utilisés pour representer des ensembles d'états ainsi que la capacité de décrire les modifications d'état via des pré et post conditions. De CSP, le langage hérite le concept d'événements et de communication et des canaux de communication typés, les notions de choix déterministe et non déterministe (reflétée par les combinateurs $P \Box P'$ et $P \sqcap$ P'), le concept de la dissimulation $P \backslash A$ des événements A survenant dans l'évolution du processus P. En raison de la présence de variables d'état, l'opérateur parallèle de *Circus* est légèrement différent de celui de CSP. Sa syntaxe est donnée par : $P \llbracket n \mid c \mid n' \rrbracket P'$ signifie que P et P' communiquent via les canaux mentionnés dans c ; En outre, P peut modifier les variables mentionnées dans n seulement, et P' dans n' seulement, n et n' étant des ensembles disjoints de noms. De plus, le langage est définit avec une notion formelle de raffinement basé sur sa sémantique dénotationnelle.

Le langage suite la sémantique de *failure/divergence*, (mais exprimée en termes de UTP), fournissant une notion de trace d'exécution tr, ensemble de refus ref, et les

divergences. La sémantique est exprimée en termes de UTP ce qui permet d'utiliser les notions de raffinement propre à UTP. La sémantique permet un riche ensemble de règles algébriques pour les spécifications et leurs transitions vers des modèles de programmes.

3.2 Isabelle/HOL

Isabelle/HOL est un environnement logique formelle utilisé comme base pour plusieurs environnements formels. Toutes les théories et les développements formels de cette thèse sont construites au-dessus de cet assistant de preuve. Outre Isabelle/HOL, un certain nombre d'environnements logiques sont utilisés pour la formalisation de théories mathématiques et des preuves. Des systèmes comme HOL, Coq, PVS et ProofPower sont de bons exemples. Tous ces systèmes offrent une base formelle pour la spécification et le raisonnement, mais avec des différences (plus ou moins importante).

Parmi tous les systèmes disponibles, nous avons fait le choix d'utiliser Isabelle/HOL comme base de notre travail pour des raisons différentes. Tout d'abord, Isabelle/HOL est l'un des assistants de preuve les plus riches et puissants. Il offre un grand nombre de bibliothèques de théories qui peut être réutilisé pour des applications particulières. Il intègre également plusieurs outils externes (par exemple solveurs de contraintes et les model checkers) et prend en charge la génération de code. Isabelle/HOL compte avec une très large communauté d'utilisateurs et de développeurs ce qui fait qu'il est très maintenu et soutenu. Une autre raison est que notre travail a été inspiré par certaines applications existantes basées sur Isabelle/HOL, par exemple HOL-Z, HOL-CSP et HOL-TesGen qui est u environnement formel de génération de tests.

tel-00821836, version 1 - 13 May 2013

tel-00821836, version 1 - 13 May 2013

CHAPITRE 4

Isabelle/Circus

Ce chapitre présente la première contribution de la thèse : l'environnement de spécification et de vérification Isabelle/*Circus*. Il est basé sur l'assistant de preuve générique Isabelle. Cet environnement permet d'écrire des spécifications *Circus* et de prouver des théorèmes sur eux. Il contient des règles de preuve et un soutien tactique qui permet de preuves de raffinement pour des processus *Circus* (impliquant à la fois des données et des aspects comportementaux).

Les actions *Circus* sont définies dans l'environnement Isabelle/*Circus* comme des entités sémantiques, définies sur les types HOL. Cela permet aux variables et canaux *Circus* d'avoir des types HOL arbitraires. Par conséquent, l'environnement Isabelle/*Circus* offre une représentation de la sémantique dénotationnelle de *Circus* dans Isabelle/HOL. Comme on le voit dans le chapitre 3, cette sémantique dénotationnelle a été définie en termes de UTP, un cadre sémantique basée sur un calcul relationnelle alphabétisé. Pour cette raison, nous introduisons d'abord une formalisation de UTP en Isabelle/HOL.

L'environnement Isabelle/*Circus* est défini en trois couches : (i) la première couche est la formalisation (d'une partie substantielle des) théories UTP, (ii) la seconde couche contient la représentation de la sémantique dénotationnelle de *Circus* et, (iii) la couche la plus supérieure est un système de preuve pour *Circus* contenant des règles de raffinement et de simulation.

Ce chapitre présente notre intégration *shallow* de la sémantique dénotationnelle de *Circus* dans Isabelle/HOL. Cette intégration repose sur une formalisation de UTP dans Isabelle/HOL. En particulier, la représentation du concept de l'alphabet UTP sous forme d'enregistrements *record* extensibles, nous proposons une représentation typée assez compacte du langage. Cette représentation permet d'attribuer des type HOL (d'ordre supérieur) arbitraires pour les canaux, les événements et les variables d'état, telles que, par exemple, des ensembles de relations. En outre, le changement de nom systématique des variables locales est évitée en les compilant à des variables globales à l'aide d'une pile d'instances de variables. Les preuves nécessaires pour démontrer que les définitions sont conformes c-à-d satisfont la propriété *is_CSP_healthy* ont été effectués, en utilisant un certain nombre de lois de

simplification algébriques sur les actions *Circus*.

Comme l'effort de codage peut être caché par des mécanismes d'extension flexibles d'Isabelle, il est possible d'avoir une notation compacte pour les spécifications et les preuves. De plus, les tactiques existantes standards de Isabelle tels que *auto*, *simp* et *metis* peuvent être réutilisés puisque notre représentation de la sémantique de *Circus* est très proche de HOL. Ainsi, nous offrons un environnement qui peut supporter des raffinements concernant les données et les comportements. Enfin, nous montrons un petit exemple de spécification de processus et de preuve de raffinement.

La formalisation de l'ensemble de UTP et de *Circus* au-dessus de celui-ci, constitue l'environnement Isabelle/*Circus*. Il peut être utilisé en tant que base d'une multitude d'application (formellement définis). Une première application a été montré dans ce chapitre : la définition d'une notion de raffinement pour *Circus* qui permet d'effectuer des preuves de raffinement. Le chapitre suivant va présenter une application substantielle basée sur l'environnement Isabelle/*Circus*. Cette extension contient les notions de base de test pour *Circus* avec des techniques de génération de tests.

tel-00821836, version 1 - 13 May 2013

tel-00821836, version 1 - 13 May 2013

CHAPITRE 5

Test Basé sur la Sémantique

Ce chapitre explique en détail la deuxième contribution de cette thèse : le test à partir de spécifications *Circus* avec Isabelle / HOL. La théorie des tests de *Circus* est basée sur la sémantique dénotationnelle et opérationnelles de celui-ci. Isabelle/-HOL est un cadre formel qui permet de définir une approche de test basée sur la sémantique de *Circus*. Pour distinguer ce type d'approches des autres approches, plus basées sur la syntaxe, nous l'appellerons *Test Basé sur la Sémantique*.

Nous décrivons dans ce chapitre, l'environnement *CirTA* pour la génération de tests à partir de spécifications *Circus*. Des définitions de tests introduites précédemment fournissent une base pour l'élaboration de stratégies de tests pour *Circus*. Notre approche de génération de test est basé sur l'environnement formel d'Isabelle/HOL. Dans le chapitre 4, nous avons présenté des formalisations de UTP et de *Circus* comme des théories Isabelle. Ces théories constituent la base sémantique de *CirTA*.

Les définitions de tests sont basés sur la sémantique opérationnelle de *Circus* définie comme une relation de transitions étiquetées. Nous proposons une formalisation des règles de la sémantique opérationnelle au-dessus de Isabelle/*Circus* et expliquons certains choix de représentation. Grâce à ces règles, nous dérivons un certain nombre de règles qui sont importantes pour la génération de tests. Les définitions des tests sont introduites puis des tactiques de génération automatiques sont définies comme des méthodes de preuve en Isabelle.

Isabelle/HOL est un démonstrateur mature et supporte facilement nos exigences de calcul symbolique, mais des efforts substantiels devaient être investis pour la construction de notre environnement de test formel. En ce qui concerne l'expérience des 10 - 20 dernières années de la communauté des assistant de preuve interactifs, cette ascension raide au départ est en fait assez commune, et nous pouvons anticiper d'éventuels avantages pour des exemples plus complexes. HOL, comme une logique, ouvre un large espace de modélisation mathématique riche, et Isabelle/HOL comme un environnement formel supporte de nombreux domaines mathématiques par des outils de preuve, par exemple pour la simplification et la résolution de contraintes. Plusieurs de ces outils de Isabelle sont déjà intégrés à d'autres externes, tels que Z3. Ainsi, nous profitons de cette riche collection d'outils de raisonnement formel pour

notre application particulière de test de conformité pour *Circus*, et nous exploitons, au maximum, le potentiel de la technologie de démonstrateur pour notre travail.

Par exemple, le choix d'une représentation symbolique superficielle *shallow*, en utilisant les installations symboliques de Isabelle pour les définitions et la génération de tests s'est avéré être un choix judicieux et puissant car il permet d'éviter un bon nombre de détails techniques lourds (en particulier dans la gestion des alphabets symboliques).

Un petit exemple est donné à la fin de ce chapitre pour illustrer la définition du test et le processus de génération. Les spécifications de test sont énoncées comme des objectifs de preuve et les hypothèses de sélection de tests sont exprimées comme règles d'introduction. Dans cette première présentation, nous considérons seulement le test hors-ligne, mais il est clair que des applications plus réalistes nécessitent des tests en ligne.

Dans le chapitre 6, l'environnement de génération de test est appliqué sur une étude de cas plus importantes. Une spécification *Circus* est écrite pour un système de surveillance des messages, implémenté comme une file d'attente. Les tactiques de génération de test sont utilisés pour engendrer des tests à partir de cette spécification. Une implémentation en Java du système a été testé en utilisant les tests générés.

tel-00821836, version 1 - 13 May 2013

tel-00821836, version 1 - 13 May 2013

CHAPITRE 6

Étude de Cas

Ce chapitre présente une application de notre environnement de spécification et de génération de tests (*CirTA*) pour une étude de cas réaliste. Le système étudié est un module de surveillance des messages implémenté comme une files d'attente FIFO. Ce module est une partie d'un système de surveillance à distance qui relie une variété de dispositifs, et en particulier des contrôleurs de stimulateur cardiaque sur des patients . L'ensemble du système est expliqué dans la première section, avec quelques détails sur le module de file d'attente. L'implémentation Java de cette file d'attente est utilisé pour exécuter les tests générés.

Une spécification *Circus* de la file d'attente est donnée dans la deuxième section de ce chapitre. Les opérations de la file d'attente sont représentés par des canaux de communication et la file d'attente est gérée par des variables d'état. La représentation de cette spécification dans la syntaxe de Isabelle/*Circus* est également donnée dans la deuxième section.

Le reste de ce chapitre traite de la génération de tests, leur préparation et leur exécution. Tout d'abord, la spécification de la file d'attente est utilisée pour générer l'ensemble des traces jusqu'à une longueur donnée. Pour chaque trace générée, les séries de tests correspondants sont générés pour les deux relations de conformité. Les tests sont ensuite traduits en méthodes Java afin d'être testé contre l'implémentation de la file d'attente. L'exécution des tests se fait en utilisant l'outil de test JUnit dans la plate-forme Eclipse.

Les premières campagnes de test ont soulevé deux questions importantes liées à la génération et à l'exécution des tests. Tout d'abord, le choix des règles de simplification utilisés dans la génération de test est très critique. L'utilisation d'un grand nombre de règles de simplification augmente la durée de la génération, surtout si le but de preuve est très complexe. De plus, certaines règles de simplification peut conduire à des simplifications superflues des hypothèses et donc une perte de contraintes. La raison en est que le simplificateur peut affiner le spécification de départ pour produire les sous-objectifs résultant. L'ensemble des règles de simplification a été réduit afin d'éviter ces problèmes. Seules des règles utiles et sûrs sont utilisées dans la génération de tests. L'ensemble des règles qui peuvent induire une

perte de contraintes sont supprimées de l'ensemble de règles de simplification. L'état final de la preuve doit être équivalente à la spécification de test de départ, et non pas seulement un raffinement de la dernière. Cette propriété peut être prouvée pour chaque règle de simplification utilisé par le simplificateur.

Une autre question importante est la définition d'une fonction d'oracle spécifique pour l'évaluation des résultats des tests. L'utilisation de la méthode de l'égalité définie dans l'implémentation du type de message a donné des résultats biaisés. Une fonction spécifique d'égalité, fondée sur l'égalité des nombres entiers, est définie et utilisée pour vérifier les des messages valeurs obtenues.

Les tests n'ont révélé aucune erreur, ce qui n'est pas une grande surprise étant donné que le système sous test est déjà en cours d'utilisation. Cependant, cette étude de cas présente une preuve de la technologie de la façon dont notre environnement peut être utilisé pour un système réel. Sur la base de cet environnement, il reste à introduire des stratégies de tests plus réalistes que l'exhaustivité. Ce sera fait en introduisant des hypothèses de test plus fortes, donc une certaine orientation de la tactique de génération de tests.

tel-00821836, version 1 - 13 May 2013

tel-00821836, version 1 - 13 May 2013

CHAPITRE 7

Conclusions et Travaux Futurs

Résumé des Contributions

Cette thèse présente une approche basée sur la sémantique pour la spécification, la vérification et le test. Nous profitons du cadre formel fourni par les assistant de preuve (Isabelle/HOL dans notre cas). Ces cadres formels offrent des bases vérifiées pour tout développement basé sur les logiques.

Le langage *Circus*, dont la sémantique est basée sur un calcul relationnel, est l'objet de nos développements formels. Ce langage fournit une notation et des techniques pour raisonner sur les conceptions et les implémentations de spécifications décrivant des données et des processus réactifs complexes. Sa sémantique riche et entièrement intégrée fait de lui un très bon formalisme pour la spécification des systèmes réels. L'intégration sémantique de ce langage dans Isabelle/HOL est la base de notre travail. En plus de cette intégration, nous formalisons plusieurs techniques de vérification et de test pour *Circus*.

Le travail présenté dans cette thèse apporte plusieurs contributions au domaine des environnements spécification, de vérification et de test formel. En soi, la représentation formelle basée sur la sémantique d'un langage de spécification complexe et riche dans un assistant de preuve est une réalisation importante.

Les principaux résultats de cette thèse sont les suivants :

UTP dans Isabelle/HOL La première contribution est l'intégration superficielle de UTP, la base de la sémantique de *Circus*, dans Isabelle/HOL. UTP (Unifying Theories of Programming) est un modèle sémantique basée sur un calcul relationnel alphabétisé. UTP propose un ensemble de théories qui peuvent être utilisées pour unifier la sémantique des langages de différents paradigmes. Il est utilisé, par exemple, pour définir la sémantique de *Circus*, combinant les notions de manipulations de données complexes et des descriptions des comportements complexes. Nous présentons une formalisation de UTP en Isabelle/HOL. Cette formalisation est exprimée en un encodage superficiel (*shallow embedding*) du calcul relationnel de UTP dans HOL.

La sémantique dénotationnelle de Circus dans Isabelle/HOL La représentation de la sémantique dénotationnelle de *Circus* au-dessus de l'encodage de UTP dans Isabelle est une autre contribution de ce travail, avec les définitions de quelques notions clés comme les variables d'état, la portée, les canaux de communication et les ensembles de noms. Ces définitions permettent d'introduire un encodage superficiel de la sémantique dénotationnelle de *Circus* dans Isabelle/HOL. Cette encodage superficiel est basé sur la formalisation UTP dans Isabelle/HOL. Les tactiques de preuve standards de Isabelle tels que *auto* et *simp* peuvent être réutilisés pour raisonner sur les spécifications *Circus*.

Isabelle/Circus Nous fournissons l'environnement Isabelle/*Circus*, basée sur l'incorporation de *Circus* dans Isabelle/HOL. Il peut être utilisé pour raisonner sur des spécifications *Circus*, ou en tant que base pour un certain nombre d'applications formelles combinant des données et des descriptions de comportement complexes. Un exercice intéressant est explorée dans le chapitre 4, avec la formalisation des notions de raffinement de *Circus* avec un certain support pour les preuves de raffinement.

La sémantique opérationnelle de Circus dans Isabelle/HOL Dans le chapitre 5, nous introduisons une autre application importante basée sur l'environnement Isabelle/*Circus*. Nous proposons une formalisation des règles de la sémantique opérationnelle au-dessus de Isabelle/*Circus* et expliquons certains choix de représentation. Grâce à ces règles, nous engendrons un certain nombre d'autres règles qui sont importantes pour la génération de tests.

CirTA Nous décrivons dans le chapitre5 CirTA (*Circus* Test Automation), un environnement de génération de test pour *Circus*. Des définitions des tests sont présentés et ensuite des tactiques de génération automatique sont définis comme des méthodes de preuves de Isabelle. Les définitions de tests et les tactiques de génération sont définis pour les processus *Circus*, en utilisant la sémantique opérationnelle. Le choix d'une représentation symbolique superficielle en utilisant les installations symboliques de Isabelle pour les définitions et la génération de test s'est avéré être un choix judicieux et pratique car elle permet d'éviter un bon nombre de détails techniques lourds (en particulier dans la gestion des alphabets symboliques).

Étude de Cas Dans le chapitre 6, l'environnement de génération de test (*CirTA*) est appliquée à une étude de cas réelle. Le système étudié est un module de surveillance des messages qui relie une variété de dispositifs, et en particulier les contrôleurs stimulateur cardiaque (pacemakers). Une spécification *Circus* est écrite pour ce système et est utilisée pour générer des tests jusqu'à une longueur donnée. Une implémentation en Java du système de suivi a été testé en utilisant les tests générés. L'exécution des tests se fait en utilisant l'outil de test JUnit dans la plate-forme Eclipse.

tel-00821836, version 1 - 13 May 2013

Travaux Futurs

Le travail présenté dans cette thèse constitue une avancée significative dans l'instrumentation de la spécification et la vérification de systèmes réels complexes. Cependant, il est clair qu'un certain nombre de questions intéressantes et des améliorations futures doivent être prises en compte. Ci-dessous, nous esquissons ceux que nous prévoyons de faire dans l'avenir.

La sémantique dénotationnelle Au niveau de la sémantique dénotationnelle de *Circus*, certaines actions qui ne sont pas encore considérés doivent être définis. Des opérations complexes de communication ne sont pas encore pris en compte. En plus de cela, certains opérateurs généralisés (par exemple le choix indexés) ne sont pas encore définies dans la théorie. L'ajout de ces actions rendra de notre langage plus facile à utiliser.

De plus, dans la version actuelle des théories, aucune opération sur les processus n'est prise en compte. Fondamentalement, ces opérations sont similaires à ceux sur les actions, mais avec un traitement spécial des états locaux. Ces opérations doivent être définis pour permettre l'écriture des spécifications complexes d'une manière plus modulaire.

Isabelle/Circus Au niveau de Isabelle/*Circus*, les techniques d'analyse doivent être améliorée afin de couvrir les définitions de schéma complexes. Des spécifications *Circus* externes peuvent être considérés par l'intégration de l'environnement avec l'analyseur CZT-parser pour *Circus*. Un soutien pour les preuves de raffinement peut être ajouté, y compris des tactiques automatiques de preuve de raffinement.

La sémantique opérationnelle Pour la sémantique opérationnelle, une réalisation importante serait de prouver la cohérence des règles avec les sémantique dénotationnelle. Ce problème a déjà été abordé dans le cadre de la sémantique de *Circus*, mais aucune preuve automatisée n'est fournie.

Une autre amélioration serait de couvrir les autres actions *Circus* mentionnées précédemment pour la sémantique dénotationnelle : de nouvelles règles doivent être ajoutés à la sémantique opérationnelle pour décrire le comportement de ces actions.

CirTA Au niveau de *CirTA*, plusieurs améliorations sont nécessaires. Tout d'abord, l'ensemble des activités de génération et de préparation de tests doivent être rendues plus automatiques, afin d'améliorer tout le temps dédié à la génération et à la compilation de tests.

L'intégration de l'environnement de génération de tests dans HOL-TestGen serait extrêmement fructueux pour les deux environnements. Une première (faible)

tel-00821836, version 1 - 13 May 2013

intégration a déjà été présentée dans l'étude de cas. Elle porte sur la réutilisation de HOL-TestGen pour la sélection et l'instanciation des tests.

De toute évidence, l'efficacité des procédures de génération doit être améliorée. L'étude de cas a révélé que le temps de génération de test est très important, principalement en raison de son exhaustivité. Mais c'est aussi en raison du nombre de règles de simplification qui sont appliquées à chaque étape de la génération. La taille de l'état de preuve joue également un rôle important dans l'efficacité de la génération. Plusieurs améliorations sont possibles, soit en ajoutant plus de parallélisation à la tactique de génération ou en allégeant l'ensemble des règles de simplification concernées. L'efficacité peut également être améliorée en utilisant des techniques de simplification *paresseuse*. Ceci nécessite la *congélation* de certaines parties de la spécification de test dans certains points, pour éviter au simplificateur de tenter toute simplification sur les parties gelées.

Nous avons maintenant les bases pour la définition de stratégies plus élaborées de sélection, de génération et d'exécution de tests. Une première étape est une combinaison des tests générés pour les deux relations de conformité afin de réduire le nombre de tests non concluants, une preuve que les résultats de l'exhaustivité sont préservés. Ceci nécessite la définition de nouvelles structures de test et des techniques de génération adaptées. Une autre étape possible est la formalisation des critères de couverture, et de leurs hypothèses de sélection associés, basée sur l'exploitation de Isabelle/*Circus* et de *CirTA*.

De plus, les techniques de test en ligne peuvent être utilisé afin de réduire le nombre de tests non concluants et de faux négatifs. Ceci nécessite que les testeurs soit connecté avec des solveurs de contraintes pour supporter les instanciations en ligne.

tel-00821836, version 1 - 13 May 2013