

MODULOGIC

Conception d'un atelier de construction

modulaire

et logiquement fondée

de systèmes logiciels

**pouvant répondre à des exigences élevées de
certification**

page web : modulogic@beaune.inria.fr

Journées de l'ACI SI - Rennes - Décembre 2003 – 1/15

Participants

- CPR, CEDRIC (CNAM) : Catherine Dubois
- LOGICAL, LIX (Polytechnique) : Gilles Dowek
- PROTHEO, LORIA : Isabelle Gnaedig
- MOSCOVA, INRIA Rocquencourt : Damien Doligez
- MIRO, INRIA Sophia : Luigi Liquori
- SPI, LIP6 (UPMC) : [Thérèse Hardin, coordonatrice](#)

Construction d'un système logiciel

- spécifier, concevoir, implanter des unités (“modules”)
- assembler des unités pour bâtir le système logiciel de manière logiquement fondée
- favoriser le partage entre différentes unités
- différencier des unités de souche commune :
Implantations de $Z/2Z$ sur `bool` et sur `int` doivent partager la connaissance sur $Z/2Z$ mais éviter de calculer (`true + 1`)
- interagir avec des compilateurs de langages de programmation, des systèmes de recherche et de vérification de preuves, ...

Construction d'un système logiciel

Et aussi :

- Mettre les preuves des propriétés à disposition des utilisateurs
- Engendrer automatiquement la liste des hypothèses sous lesquelles une propriété est vérifiée
- Engendrer automatiquement la documentation sur l'architecture du système (dépendances entre unités, instanciation de paramètres, ...)

Qu'est-ce qu'une unité?

Une unité définit un ensemble d'entités caractérisées par un ensemble de champs :

- le type de la représentation (une variable de type, un type générique, un type totalement défini)
- Déclaration d' un identificateur et de son type
- Définition d' un identificateur
- Énoncé de propriété
- Preuve de propriété

Spécification = déclarations + énoncés

Implantation = toute déclaration est associée à une définition,
toute propriété est prouvée

Construire une unité

- Introduction de nouveaux champs
- Héritage multiple
- Redéfinitions
- Paramétrage d'une unité par une autre unité
- Paramétrage par une entité d'une autre unité

[Un parfum de type abstrait algébrique et de langage à objets](#)

Cadre logique

- variables de type, principes de récurrence
- Dépendances entre déclarations et énoncés, définitions et preuves

théorie des types dépendants avec polymorphisme

... mais langage de propriétés en "simili" premier ordre

Déjà fait : Spécification en Coq de la hiérarchie des unités
(langage Foc: SPI + MOSCOVA + CEDRIC)

Interférence entre redéfinition et preuve peut conduire à des inconsistances logiques

Pb. résolu en imposant que toute unité puisse être associée à une **interface** constituée des types des champs

Traits objet et Encapsulation

Deux sortes d'unités:

- les espèces, déjà décrites
- les collections, obtenues par un mécanisme d'[encapsulation](#) d'[espèces](#) totalement implantées
- Paramétrage des espèces par des interfaces, instanciées par des collections ⇒ évite les ruptures d'invariant de représentation par exemple.

Déjà fait : Le langage Foc (bibliothèque de calcul formel écrite en Foc)

Apport de ModuLogic

Etendre Foc pour traiter plus de propriétés de sécurité

Spécifier formellement l'atelier

Déjà fait : rho-calcul (PROTHEO + MIRO)

- rho-calcul = réécriture + lambda-calcul + non-déterminisme
- théorie des types pour le rho-calcul
- permet de formaliser les paradigmes objet, les stratégies, ...

Apport de ModuLogic

Donner une spécification complète de l'atelier dans le rho-calcul

de manière à pouvoir étudier des extensions de la notion d'espèce

De bons outils de spécification/implantation

Déjà fait : spécifier et programmer en définissant des règles de manipulation d'expressions et des stratégies, disposer de mécanismes puissants de filtrage, de contraintes : langage Elan, PROTHEO

Apport de ModuLogic

Incorporer les spécificités de Elan et de Foc dans l'atelier

Elan *s'apparente aux types abstraits algébriques*

Déjà fait :

- sémantique de Elan dans le rho-calcul
- compilation de spécifications Elan en Java, en C (Tom)

Outil de recherche de preuve

Déjà fait

- Preuves de terminaison, de complétude, de confluence et d'atteignabilité, faites par réécriture (PROTHEO)
- Compilation d'un contexte Coq contenant toutes les informations disponibles pour prouver une propriété d'une espèce (Foc)

Apport de ModuLogic:

Un outil de recherche "automatique" de preuves fondé sur la déduction modulo

Déduction modulo

- Remplacer instantanément une formule par une formule équivalente mieux adaptée à la recherche de preuve (Dowek-Hardin-Kirchner)

$\forall x, y. x \times y = 0 \rightarrow \forall x, y. x = 0 \vee y = 0$)

$$\frac{\Gamma, P \vdash_{\text{RE}} \Delta \quad \Gamma, Q \vdash_{\text{RE}} \Delta}{\Gamma, R \vdash_{\text{RE}} \Delta} \vee\text{-I} \text{ if } R =_{\text{RE}} (P \vee Q)$$

- $=_{\text{RE}}$: réécriture de propositions + théorie équationnelle sur les termes
- Typiquement : les étapes “bateau” d’un raisonnement par récurrence faites par réécriture, étapes “intelligentes” laissées dans la preuve

Outil de vérification de preuve

Déjà fait

- compilation de preuves Elan en des termes-preuves de Coq (PROTHEO+LOGICAL)
- compilation d'un contexte Coq pour Foc

Apport de ModuLogic:

Preuves compilées vers Coq, pour être vérifiées

Le compilateur

Passé 1: vérification de la cohérence logique des espèces et collections (garantie sur la construction du logiciel), analyses statiques ...

Passes 2.*:

- vers un/des langages source (de préférence avec une sémantique bien étudiée) pour obtenir un exécutable
- vers le système de recherche de preuves
- production de documentation automatique en différents formats

Déjà fait

- Compilateur pour Foc, effectuant ces passes
- Compilateur pour Elan vers Java et C

Conclusion

Un projet d'une certaine ampleur, qui n'aurait pas pu être envisagé individuellement par chacune des équipes participantes

Challenges :

- Spécification formelle de tout un langage dans le rho-calcul
- Outil de recherche de preuves fondé sur la déduction modulo, effectuant aussi des preuves de terminaison
- Compilation de preuves (scripts d'application de règles) vers Coq

De manière à donner plus de confiance dans le logiciel