Switch to full screen

Contrats de programmes réactifs

Florence Maraninchi, Lionel Morel
Marc Vareille, Jacques NDjeng NDjeng
Paul Caspi et Jan Mikac

Plan

- La conception par contrats de [Meyer 89] à nos jours
- Contrats de systèmes réactifs : besoins
- Un tour d'horizon de notions liées
- Les contrats Lustre et l'outillage associé
- Les contrats dans ALIDECS

1: Contrats

La conception par contrats [Meyer 89]

```
procedure Empiler (p : in out Pile ; x : in Element) is
   -- Precondition :  p=p0, p0 n'est pas pleine
   -- Postcondition : p=p1, p1 = p0&x
```

Métaphore du contrat :
Si l'appelant assure qu'il passe à `Empiler` des paramètres qui vérifient la précondition alors
la procédure `Empiler` se termine et
les paramètres de sortie sont conformes à la postcondition.

Variante "objet"

```
class Pile {
   private ... p ;
   Pile () { p = new ... ; }
   public void Empiler (Element x) {
   // Precondition :  p=p0, p0 n'est pas pleine
   // Postcondition : p=p1, p1 = p0&x
   }
```

Métaphore du contrat :
Si les paramètres d'entrée et l'état interne de l'objet vérifient la précondition alors
la méthode `Empiler` se termine et
les paramètres de sortie et l'état interne de l'objet sont conformes à la postcondition.

Types d'assertions

On distingue :

- Les pré et post conditions de méthodes
- Les invariants de classes
  (vrais "entre deux appels" de méthodes)

Langages, outils, etc.

Eiffel [Meyer] – contrats font partie du langage

JContract, IContract (langage de spécification des contrats, génération de code défensif par transformation de programmes Java)

C♯ ??
C `assert` pour la programmation défensive

OCL dans UML (contraintes statiques simples, invariants de classes, pré-post des méthodes)

Limitations du langage des contrats (I)

Comment exprimer : la méthode `m2` ne peut être appelée que si la méthode `m1` a été appelée au moins une fois auparavant ?

On introduit un nouvel attribut mémoire M dans la classe (init faux).
`m2` a comme pré-condition : M vrai
`m1` a comme post-condition : M vrai

Cas plus compliqués : on doit coder à la main l'automate reconnaisseur des séquences d'appels de méthodes corrects.

Limitations du langage des contrats (2)

Pour les systèmes réactifs, on va avoir besoin de parler des exécutions successives de la même portion de code qui constitue le noyau réactif du programme (ou même chose pour certains des composants).

Chercher des notions de contrats qui permettent de parler des exécutions successives d'une méthode, ou des appels successifs aux méthodes d'une même classe, ...

Composants et contrats

On trouve souvent une distinction entre les types suivants de contrats :

- Les contrats dits "syntaxiques" :

  détrompeurs et un peu de typage

- Les contrats fonctionnels

- Les contrats de synchronisation

- Les contrats de qualité de service

  coût en temps ou mémoire, consommation, ...
  problème de calculs compositionnels avec ça.

2: Contrats pour les systèmes réactifs
Premières idées et besoins

Caractéristiques des systèmes réactifs

On s'intéresse à des contrats fonctionnels
(et de synchronisation, considérée comme faisant partie de la fonction).

Les systèmes réactifs sont :

- bouclés (système+environnement)
- non terminants
- temporels (temps discret) = raisonnement sur des séquences

Contrats pour les systèmes réactifs ?

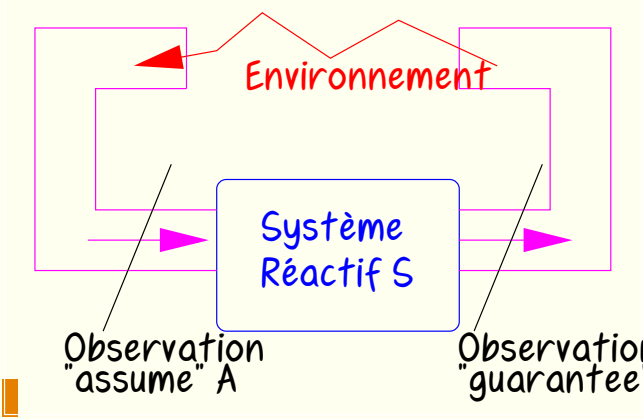
Les contrats peuvent oublier les histoires de terminaison

Les contrats doivent parler de temps :
On doit pouvoir supposer que :

- la température augmente
- les entrées `i` et `j` alternent
- ...

Une première proposition

pré- et post- deviennent assume et guarantee.



Sémantique : partout dans la séquence $(\forall t) : A \wedge S \Longrightarrow G$

Sur la boucle principale du code séquentiel

Pour des A et G combinatoires :

```
init memory M
while true loop
    get inputs i
    -- Assume (i)
    compute outputs o = f (M, i)
    -- Guarantee (i, o)
    update memory M' = g (M, i)
end loop
```

Pour A et G séquentiels, ajouter leur mémoire.

3 : Un tour d'horizon de notions liées

Assume/Guarantee (McMillan)

Circular assume/guarantee reasoning =
recurrence sur le temps discret.

La seule règle de preuve compositionnelle qui s'applique vraiment.

Interface Automata (Alfaro et al)

Pas de séparation claire entre
la partie "assume" et la partie "garantie".

(cf. exemples de Lionel Morel)

Reactive Modules (Henzinger)

Plutôt un format interne comme cible de langages de plus haut niveau : pas de vrai support langage pour décrire des spécifications.

(Sequential) don't care des circuits

Exactement ce dont on a besoin.
Utilisés surtout pour de l'optimisation, parfois pour la validation.
Support langage assez pauvre.

OCL et extensions

OCL = Object Constraint Language (dans UML)

BOTL et autres variantes : extensions en logique temporelle de OCL, pour la prise en compte du temps logique.

Problème essentiel : on parle de la séquence globale d'exécution d'un programme objet parallèle, sans jamais dire comment on l'obtient à partir des exécutions des composants.

Les spécifications dans la méthode B ?

Pas vraiment sous forme de contrats
(séparation assume/guarantee)

JASS (Java with assertions)

Intéressant pour la prise encompte explicite des traces (temps logique et appels successifs de la même méthode)

4 : Contrats Lustre et outillage

Ecrire les contrats directement en Lustre

- on a toute la puissance du langage
  (variables locales, calculs, mémoire, ...)

- synchrone (définition claire de LA séquence globale, et tout le monde a la même échelle de temps)

Exemple : programme Lustre avec contrat

```
node N (i) returns (o)
var
    v : ...            -- variables ordinaires
    a, g : bool ;      -- contrat
let
    a = A_N (i) ;
    -- equations de v, o :
    ...
    g = G_N (i, o)
tel
```

Commentaires

- `A` et `G` sont des observateurs, qui peuvent être des noeuds quelconques en Lustre
- `A` ne dépend que des entrées...
- Les mémoires de `A`, `G` et celle du corps sont disjointes. Peut conduire à de la duplication.

Avantages

Pour la preuve, un contrat est utilisable à la place du composant fini.
On fait donc la preuve avec la spécification non déterministe, pas avec une implantation particulière déterministe.

Exemple (combinatoire) tiré d'une étude de cas Airbus : procédure de sélection d'un élément maximum dans un tableau.
N'importe quelle implantation Lustre doit établir une priorité statique. La spécification est non déterministe.

Travail de Marc Vareille (magistère 2002)

- Définition de la forme des contrats
- Exemples significatifs en Lustre
- Expériences de preuves compositionnelles avec Lesar et Nbac
- Connexion au débogueur Ludic (avec Fabien Gaucher) pour l'interprétation défensive et l'aide à la localisation de bugs

Travail de Jan Mikac (Thèse 2002-2005)

Définition d'un principe de raffinement de noeud Lustre (au sens de B), avec spécification des noeuds par des contrats (les mêmes !)

Exemple de noeud à raffiner

```
node N (i) returns (o)
refines M
with  -- invariant de liaison
assumes
   var
   let ... tel
guarantees
   var
   let ... tel
body
   var
   let ... tel
```

Travail de Lionel Morel (Thèse 2001-2004)

Tableaux et contrats, exploitation des régularités et des spécifications locales pour l'extraction d'obligations de preuve.

--- Contrôles de cohérence et de branchements grâce aux contrats (en particulier dans les itérations sur les tableaux)
--- Extraction d'obligations de preuve, à la preuve compositionnelle
--- Etudes de cas : Airbus ELMU (Electrical Load mangement Unit) et gyroscope (Avions indiens)

Travail des testeurs (Pascal, Yvan, Erwan)

Mécanique utilisable pour l'exécution précoce de programmes dont certaines parties ne sont données que par leur contrat.

Travail de Jacques NDjeng (DEA 2003)

Vers un modèle de composants réactifs (avec construction dynamique) - Exécution précoce de contrats temporels et validation formelle

Aspects dynamiques complètement inspirés par les Sugar-Cubes (Boussinot, Susini).

5: Les contrats dans ALIDECS pour notre future notion de composant

A intégrer

- Les contrats fonctionnels simples pour composants statiques
- Des contrats plus sophistiqués décrivant des dépendances temporelles complexes entre entrées/sorties
- Des contrats pour composants dynamiquement configurables, décrivant ce qu'on a le droit de faire en configurant les composants
- Des contrats pour composants dans un contexte de construction complètement dynamique

Coordination avec ASSERT

Tâche "contrats et exécution précoce"

--- Etude de cas = ATV de EADS lanceurs,
par David Lesens, en Lustre !
--- Intérêt manifesté par Prover Technologies

Liens avec d'autres travaux

Equipe commune avec ST microelectronics Central R&D Crolles.

Conception et analyse de SoCs au niveau transactionnel (TLM).
Composants pour les SoC (SystemC)

A lire...

La biblio "composants" (cf. Gregor G.)
Fractal, EJB, .com, ... Corba, ...

Contrats de langages fonctionnels d'ordre supérieur
Reactive modules, interface automata,
don't care, BOTL, JASS, ...

Conposants : Liste d'Alain

Reactive modules (Henzinger, ...) formel, Lg faible
Ptolemy (Lee) simu
Prometheus (Greg et Joseph) pas de compil.
Shift (Berkeley) simu.
Reactive Machines (Boussinot)
Les (software)ADL ... (MetaH est devenu A(vionics)ADL étudié dans ASSERT)
Metropolis
Koala – introspection
(Enterprise) Java Beans
Fractal et Fraktal, Julia, Think
RMA – radio logicielle et reconfiguration dynamique
Et une liste de langages ...