

Introduction de techniques de tests en FOCAL

Carlier Matthieu

Thèse
CNAM - CEDRIC

Encadrant : Catherine Dubois (Laboratoire CEDRIC)

Plan

I - Contexte

II - Test des propriétés

III - Couverture

IV - Perspectives

Pourquoi tester en Focal ?

- ▷ Des propriétés dans Focal ne sont pas prouvées.
 - porte sur du code importé de Ocaml.
 - hors de portée du prouveur.

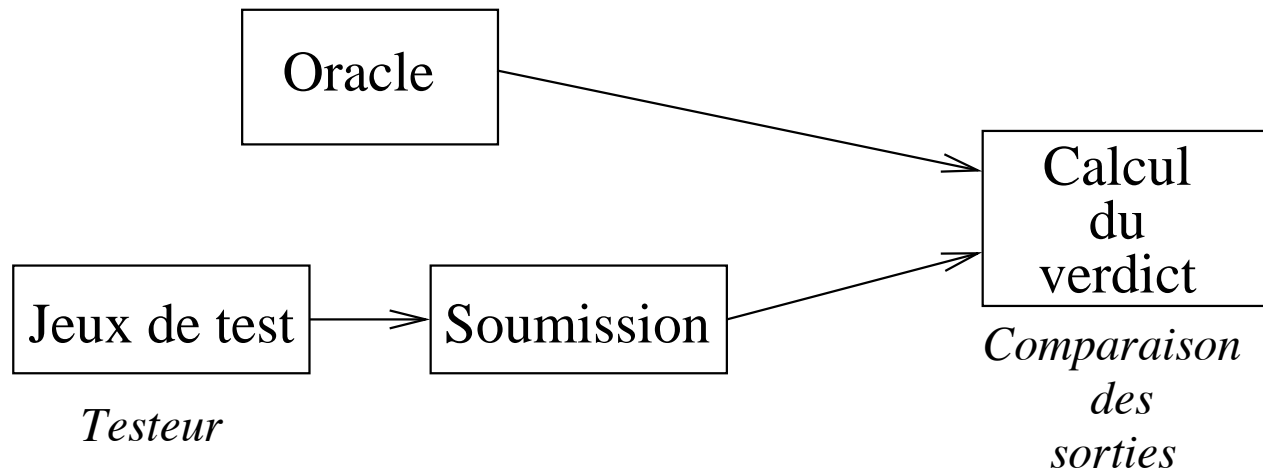
- ▷ Pendant le processus de développement
 - preuve qui échoue, on teste pour vérifier la propriété (chercher un contre-exemple).

- ▷ Pour éprouver le comportement du programme
 - ajout de propriétés supposées correctes mais ne faisant pas partie de la spécification.

But

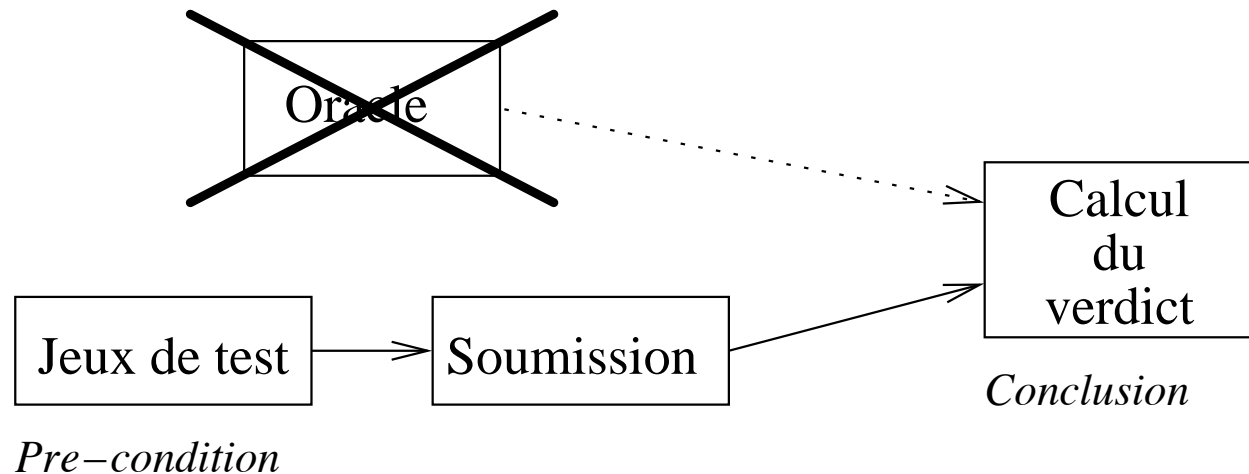
- ▷ fournir un outil de test qui permet de vérifier une propriété face à une implantation.
 - Test automatique.
 - Possibilité de test manuel.
 - Création de rapports de test.
 - Réinjection des jeux de test déjà soumis.

Méthodologie traditionnelle



- Un testeur écrit des jeux de test.
- Un oracle prédit un résultat (testeur, autres programmes, ...).
- Le verdict est calculé par comparaison des résultats.

Méthodologie adaptée



- On met en évidence une partie pré-condition et conclusion de la propriété.
- Un jeu de test doit au minimum satisfaire une pré-condition.
- Le verdict est obtenu à l'aide de la conclusion.
- L'oracle est supprimé.

Hypothèses

- ▷ On prend des espèces complètes qui peuvent être paramétrées
 - le type support est connu
 - toutes les méthodes sont définies

- ▷ Les propriétés portent sur des variables de type :
 - *self*
 - les paramètres de l'espèce
 - type de base (*int*, *string*, *bool*, ...)

Les propriétés

- ▷ On restreint la forme des propriétés.
→ les propriétés exécutables (pas de \exists notamment)
- ▷ On s'intéresse aux formules de la forme :

$$\forall X_1 \dots X_n. \varphi$$

avec φ de la forme :

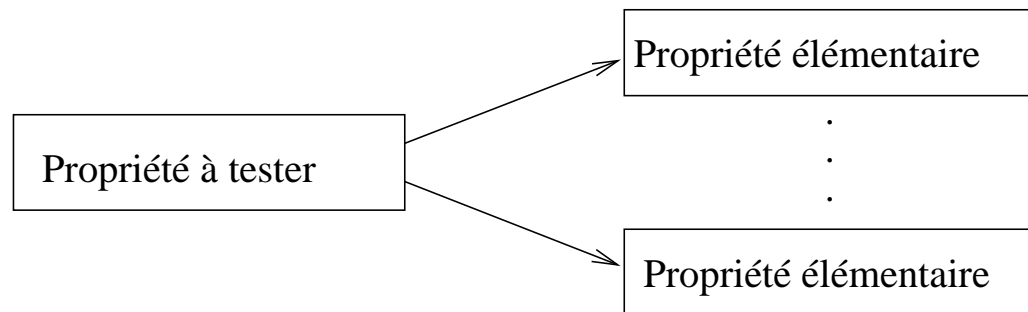
$$\alpha_1 \Rightarrow \dots \alpha_n \Rightarrow (A_1^1 \vee \dots \vee A_{n_1}^1) \wedge \dots \wedge (A_1^m \vee \dots \vee A_{n_m}^m) \text{ et}$$

$$\alpha ::= \alpha \vee \alpha \mid \alpha \wedge \alpha \mid A \mid \neg A$$

Les A_i^j sont des appels de méthodes.

Découpage de la propriété

- ▷ On découpe la propriété en plusieurs propriétés plus simples :



→ les propriétés obtenues sont celles qui vont être testées.

Réécriture

▷ But : transformer la propriété à tester pour obtenir des propriétés *élémentaires*.

$$\begin{array}{l}
 \alpha_1 \Rightarrow \dots \Rightarrow (\beta_1 \vee \dots \vee \beta_m) \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma \quad \mapsto \quad \left\{ \begin{array}{l} \alpha_1 \Rightarrow \dots \Rightarrow \beta_1 \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma \\ \alpha_1 \Rightarrow \dots \Rightarrow \beta_2 \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma \\ \vdots \\ \alpha_1 \Rightarrow \dots \Rightarrow \beta_m \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma \end{array} \right. \\
 \\
 \alpha_1 \Rightarrow \dots \Rightarrow (\beta_1 \wedge \dots \wedge \beta_m) \Rightarrow \dots \Rightarrow \gamma \quad \mapsto \quad \alpha_1 \Rightarrow \dots \Rightarrow \beta_1 \Rightarrow \dots \Rightarrow \beta_m \Rightarrow \dots \Rightarrow \gamma \\
 \\
 \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow (\gamma_1 \wedge \dots \wedge \gamma_m) \quad \mapsto \quad \left\{ \begin{array}{l} \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma_1 \\ \vdots \\ \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma_m \end{array} \right.
 \end{array}$$

Exemple

$$\forall x y . !equal(x, y) \Rightarrow !equal(x, !plus(y, !zero)) \wedge !equal(x, !plus(!zero), y) \wedge !equal(!plus(x, !zero), y) \wedge !equal(!plus(!zero, x), y)$$

donne :

$$\left\{ \begin{array}{l} \forall x y . !equal(x, y) \Rightarrow !equal(x, !plus(y, !zero)) \\ \forall x y . !equal(x, y) \Rightarrow !equal(x, !plus(!zero), y) \\ \forall x y . !equal(x, y) \Rightarrow !equal(!plus(x, !zero), y) \\ \forall x y . !equal(x, y) \Rightarrow !equal(!plus(!zero, x), y) \end{array} \right\}$$

Pour Focal

$$\left. \begin{array}{l} \forall X_1 \dots X_n, \quad A_1(X_1, \dots, X_n) \Rightarrow \dots \Rightarrow A_m(X_1, \dots, X_n) \end{array} \right\} \text{pré-condition}$$
$$\Rightarrow$$
$$\left. \begin{array}{l} B_1(X_1, \dots, X_n) \vee \dots \vee B_{m'}(X_1, \dots, X_n) \end{array} \right\} \text{conclusion}$$

- ▷ On veut tester cette propriété
→ vérifier la propriété en certaines valeurs (X_1, \dots, X_n) .
- ▷ Un jeu de test : un vecteur de valeurs X_1, \dots, X_n .
- ▷ La partie pré-condition doit au minimum être satisfaite.
→ si la pré-condition est fausse alors propriété *vraie* d'emblée. *Inintéressant*

Obtention des Jeux de test

- ▷ Il y a deux grands moyens d'obtenir les jeux de tests.
 - générer les jeux de test aléatoirement (le code reste caché).
 - obtenir les jeux de test autrement (analyse du code).

Test aléatoire

$$\left. \begin{array}{l} \forall X_1 \dots X_n, \quad A_1(X_1, \dots, X_n) \Rightarrow \dots \Rightarrow A_m(X_1, \dots, X_n) \end{array} \right\} \text{pré-condition}$$
$$\Rightarrow$$
$$\left. \begin{array}{l} B_1(X_1, \dots, X_n) \vee \dots \vee B_{m'}(X_1, \dots, X_n) \end{array} \right\} \text{conclusion}$$

▷ Les propriétés sont vues comme des prédicats :

→ on génère des valeurs aléatoires des variables (les X_i) ;

→ on regarde si les pré-conditions (les A_i) sont vérifiées ;

→ on calcule la valeur de la conclusion (les B_i) qui donne le verdict.

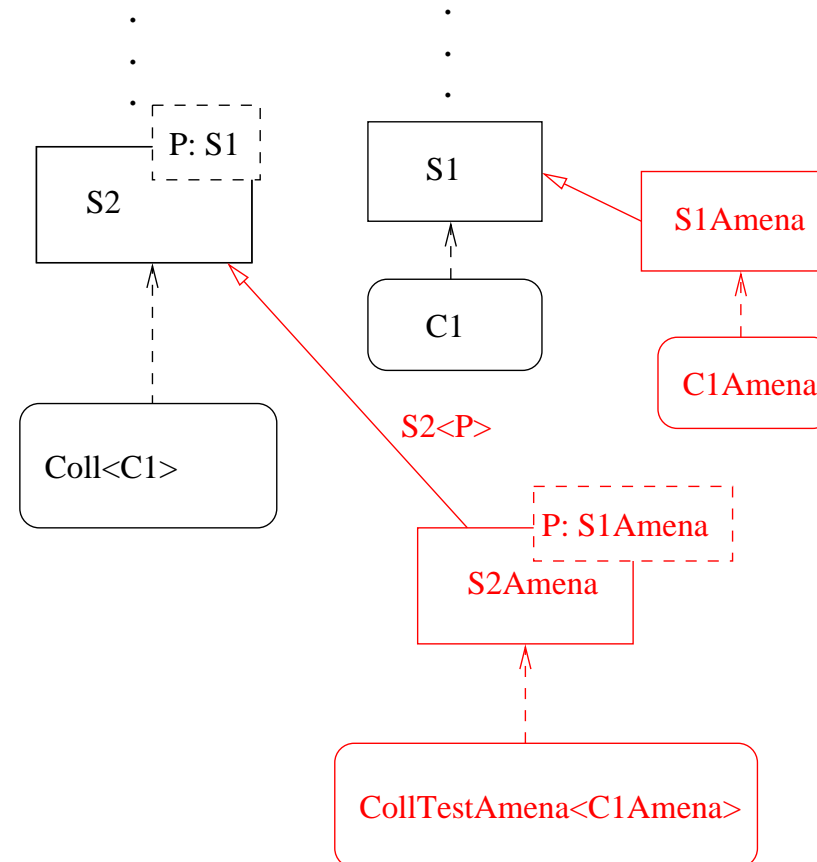
Génération des données

$$\left. \begin{array}{l} \forall X_1 \dots X_n, \quad A_1(X_1, \dots, X_n) \Rightarrow \dots \Rightarrow A_m(X_1, \dots, X_n) \end{array} \right\} \text{ pré-condition}$$
$$\Rightarrow$$
$$\left. \begin{array}{l} B_1(X_1, \dots, X_n) \vee \dots \vee B_{m'}(X_1, \dots, X_n) \end{array} \right\} \text{ conclusion}$$

- ▷ On teste les propriétés élémentaires avec des valeurs au hasard :
- on se donne les fonctions génératrices pour les types de base importés de CAML (int, float, string, ...);
 - on génère les fonctions aléatoires pour les types concrets;
 - on construit un générateur aléatoire pour les types *rep* composés (int * int, float list, int * string list option ...) en composant les générateurs des composants.

Point de vu de l'héritage

- ▷ On utilise l'héritage pour ajouter les méthodes nécessaires (aménagement).



Prototype

▷ Un prototype existe :

- prend des espèces paramétrées ;
- les propriétés de la forme définie plus haut ;
- génère des jeux de test pseudo-aléatoirement ;
- rattrape les exceptions ;
- génère des rapports de test au format XML.

▷ Expériences :

- découverte d'un bug dans la librairie standard (`geq_refines_order_sup` de l'espèce `lexicographic_product_additive_monoid`) (corrigé ??).
- compilations parfois longues.

Problème du critère de test

▷ Dans le test de logiciel on définit plusieurs stratégies.

→ Test partitionnel :

- le domaine d'entrée est coupé en une partition ;
- implique un jeu de test par composante de la partition.

→ Test aux limites :

- les bornes des valeurs d'entrée sont mises en évidence ;
- un jeu de test par borne.

→ Test nominal

- les jeux de test sont conçus avec les valeurs normales de fonctionnement.
-

pré-domaine

$$\left. \begin{array}{l} \forall X_1 \dots X_n, \quad A_1(X_1, \dots, X_n) \Rightarrow \dots \Rightarrow A_m(X_1, \dots, X_n) \\ \Rightarrow \\ B_1(X_1, \dots, X_n) \vee \dots \vee B_{m'}(X_1, \dots, X_n) \end{array} \right\} \begin{array}{l} \text{pré-condition} \\ \\ \text{conclusion} \end{array}$$

▷ Chaque propriété élémentaire donne lieu à la définition d'un domaine

Définition : On appelle pré-domaine d'une propriété P portant sur les variables $X_1 : T_1, \dots, X_n : T_n$, l'ensemble $\text{PreD}(P) \subseteq T_1 * \dots * T_n$ tel que $(X_1, \dots, X_n) \in \text{PreD} \Leftrightarrow A_1(X_1, \dots, X_n) \wedge \dots \wedge A_n(X_1, \dots, X_n)$ est vérifié.

propriétés sur les pré-domaines

▷ Soit $\text{Elem}_1, \dots, \text{Elem}_i$ les propriétés élémentaires issues d'une propriété P .

→ les pré-domaines peuvent se chevaucher.

$$\alpha_1 \Rightarrow \dots \Rightarrow (\beta_1 \vee \dots \vee \beta_m) \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma \quad \mapsto \quad \left\{ \begin{array}{l} \alpha_1 \Rightarrow \dots \Rightarrow \beta_1 \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma \\ \alpha_1 \Rightarrow \dots \Rightarrow \beta_2 \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma \\ \vdots \\ \alpha_1 \Rightarrow \dots \Rightarrow \beta_m \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma \end{array} \right.$$

$$(\text{mod}(x, 6) = 0 \vee \text{mod}(x, 21) = 0) \Rightarrow \text{not}(\text{prime}(x))$$

Donne deux propriétés élémentaires dont les pré-domaines ont 42 en commun.

propriétés sur les pré-domaines

- ▷ Soit $\text{Elem}_1, \dots, \text{Elem}_i$ les propriétés élémentaires issues d'une propriété P .
→ les pré-domaines Elem_i forment un multi-ensemble.

$$\alpha_1 \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow (\gamma_1 \wedge \dots \wedge \gamma_m) \quad \longmapsto \quad \left\{ \begin{array}{l} \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma_1 \\ \vdots \\ \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n \Rightarrow \gamma_m \end{array} \right.$$

La règle crée plusieurs propriétés sans changer la pré-condition.

propriétés sur les pré-domaines

▷ Soit $\text{Elem}_1, \dots, \text{Elem}_i$ les propriétés élémentaires issues d'une propriété P .

$$\bigcup_{i=1}^n \text{PreD}(\text{Elem}_i) = \text{PreD}(P)$$

→ l'union des pré-domaines Elem_i donne le pré-domaine de P .

→ on ne perd pas l'expressivité de P .

Critère de test

- ▷ Les pré-domaines Elem_i forment *presque* une partition du domaine de la propriété de départ.
- ▷ tester les propriétés individuellement revient à faire du test partitionnel.
 - Il est possible que certains pré-domaine soient vides (pré-condition fausse) à détecter.

Couverture sur la conclusion

- ▷ La conclusion est constituée de la disjonction de plusieurs appels de méthode.

$$B_1 \vee \dots \vee B_{m'}$$

- ▷ Pour chaque propriété élémentaire, on veut avoir au moins un jeu de test qui active chaque condition de la conclusion.
 - Certaines conditions ne peuvent pas être activables (inutile, ou erreur dans le code).
 - Des jeux de test peuvent n'activer aucune des conditions de la conclusion (faute dévoilée).

Conclusion et perspectives

- ▷ on justifie un critère de test de type partitionnel par la réécriture de propriétés.
- ▷ prototype utilisable.
- ▷ expériences positives (découverte de bug).
- ▷ réflexion sur la génération des jeux de test avec la programmation par contraintes.