FROM RESEARCH TO INDUSTRY

cea tech

list

GATeL et la génération de séquences de tests fonctionnels

Tests et méthodes formelles Juin 2015

Bruno Marre

INSTITUT CARNOT CEA LIST

digiteo

Introduction

- **Développé dans le cadre d'une collaboration IRSN depuis 1998 : aide à l'évaluation des fonctions de protection de contrôle-commande nucléaire**

- **Aujourd'hui:** environnement de V&V pour des modèles flots de données synchrones décrits en Lustre/Scade
  - Extensions successives au gré des projets/études de cas
  - Intégration de différentes fonctionnalités de test et de vérification

- **Basé sur un moteur de résolution de contraintes dédié à la génération de séquences de test à partir de modèles flots de données synchrones**
  - Efficace sur des modèles à temporisations (plusieurs milliers de cycles)
  - Mécanismes dédiés pour la prise en compte des State-Machines
  - Plusieurs arithmétiques cibles: entiers, réels, flottants simples/doubles

Problématique IRSN

Objectif : évaluation des campagnes de test constructeur pour les fonctions (critiques) des systèmes de protection

- Pouvoir définir des critères de couverture adaptés à chaque évaluation
- Elaborer des cas intéressants (fonctionnels, discriminants pour les erreurs usuelles …)
- Effectuer une évaluation des tests indépendante de celle du constructeur

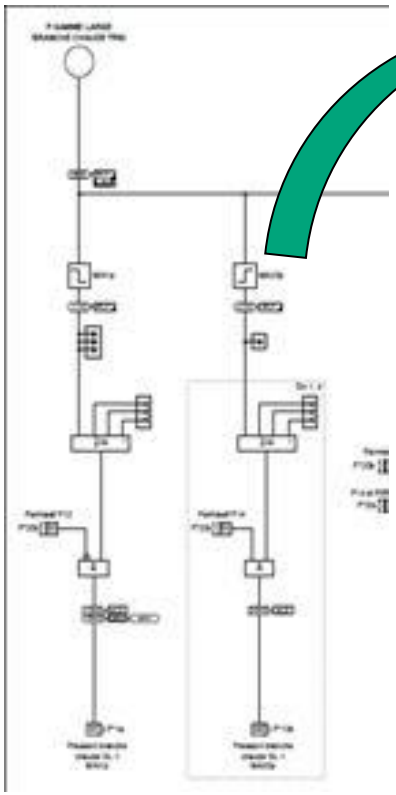
Méthode :

- Modélisation d'un DFP en Lustre/Scade (systèmes réactifs synchrones)
- Validation du modèle : simulation, comparaison aux sorties des tests constructeur
- Identification de cas de test structurels/fonctionnels
- Mesure de la couverture des cas de test sur les tests fournis par le constructeur
- Pour chaque cas de test non couvert, vérification de leur atteignabilité par génération d'une séquence témoin

Choix : Interprétation CLP + Procédure de résolution dédiée

Modélisation d'un DFP

Diagramme Fonctionnel de Protection : réseau d'opérateurs métier décrivant la logique de décision du mécanisme de protection (taille réduite)

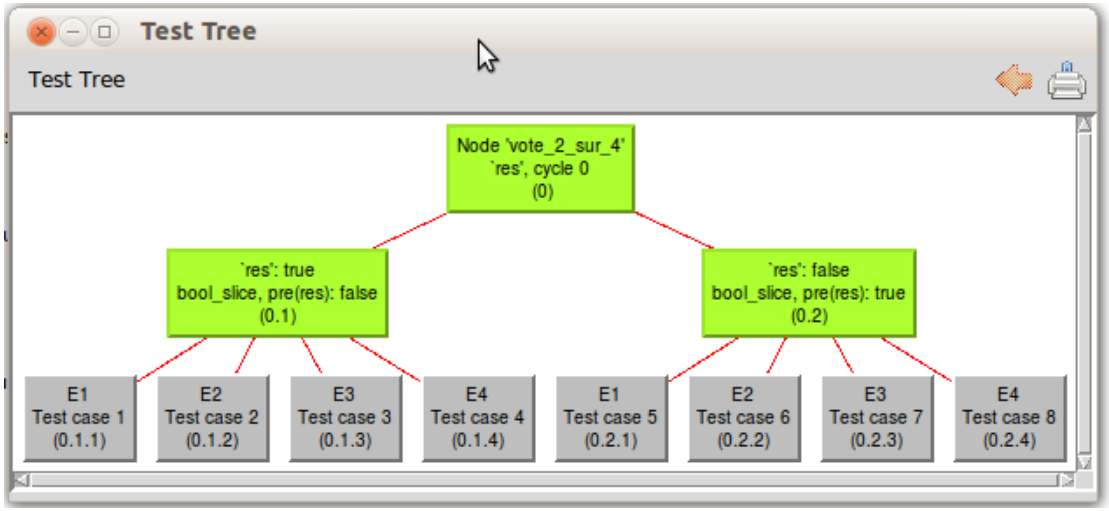


```
node seuil_haut(E,seuil,hys: int) returns (Seuil_haut: bool);
var seuil_inf : int;
let
seuil_inf = seuil - hys;
Seuil_haut = if (E <= seuil_inf)
                then false
                else if E > seuil
                    then true
                    else (false -> pre(Seuil_haut));
tel
…
```

Couverture de la logique de décision d'une sortie booléenne : plusieurs critères automatisés spécifiques IRSN

Exemple : MC/DC généralisé pour un voteur deux sur quatre

```
node vote_2_sur_4 (E1, E2, E3, E4: bool) returns (res: bool);
var
  r1,r2,r3,r4,sum: int;
let
  r1 = if E1 then 1 else 0;
  r2 = if E2 then 1 else 0;
  r3 = if E3 then 1 else 0;
  r4 = if E4 then 1 else 0;
  sum = r1 + r2 + r3 + r4;
  res = sum >= 2;
tel
```



Test case 1 :

res $\wedge$ not(pre(res)) $\wedge$ E1 <> pre(E1)) – *front montant de res, basculement de E1*

$\wedge$ E2 = pre(E2) $\wedge$ E3 = pre(E3) $\wedge$ E4 = pre(E4) – *E2, E3 et E4 stables*

Identification de cas de test fonctionnels

Définition déclarative du cas de test par une directive GATeL d'atteignabilité

`reach(Obj)`

L'expression booléenne Obj devient vrai au dernier cycle

Exemples :

- *Franchissement d'un seuil*

```
reach (pre(Val) < seuil  and  Val >= seuil)
```

- *Pas d'état invalide depuis le franchissement du seuil*

```
reach NEVER_SINCE_LAST(INVALID(Val), RISING_EDGE(Val >= seuil ))
```

Exploration interactive de sous-cas

But : raffiner les cas de tests obtenus

Techniques de raffinement : activées interactivement par l'utilisateur

- **Structurelles :** décomposition des cas par analyse de cas prédéfinis

  X = **if** Cond **then** Exp1 **else** Exp
  - Cond = true ∧ X = Exp1
  - Cond = false ∧ X = Exp2

- **Fonctionnelles :** description de scénarios fonctionnels attachés à une variable de flot (directive GATeL)

  split X **with** [sc1,…,scn]

```
split Seuil_haut with [
    -- E sous la zone à hystérésis
    -- E était dans la zone à hystérésis et franchit le seuil haut
    -- E avait franchit le seuil haut et revient dans la zone à hystérésis
    ….. ]
```

Utilisation d'invariants d'environnement

Considérer des cas de test réalistes : prise en compte d'un environnement excluant les comportements impossibles

Utilisation d'invariants d'environnement définis par des assertions

Propriétés implicites statiques :

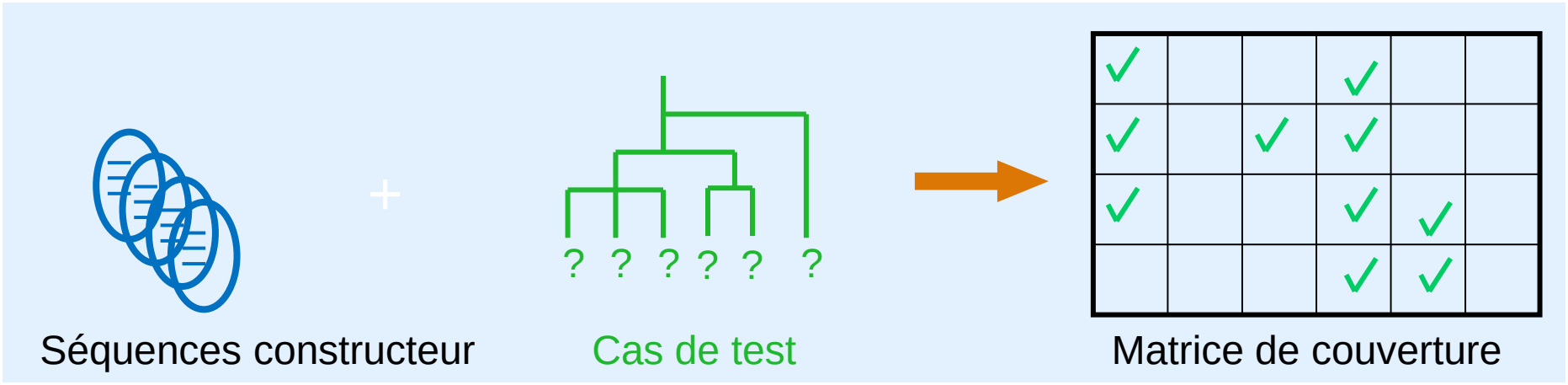
- Entrées incompatibles : not(set and reset)
- Domaine de définition de variables numériques : 0.0 < temp < 100.0

Propriétés implicites temporelles :

- Domaine de variation d'une entrée : abs(x – pre(x)) < delta
- Entrée constante si une autre entrée est vraie

Mesure de la couverture des cas de test

- **Pour chaque séquence de test constructeur :**
  - Evaluation des sorties selon le modèle IRSN et comparaison aux sorties de la séquence constructeur (alarme si différence)
  - Pour chaque cas de test à couvrir, vérifier si ses contraintes sont satisfaites par les entrées de la séquence



- **Si un cas de test n'est pas couvert, on cherche à générer une séquence témoin**

Aperçu de l'outil

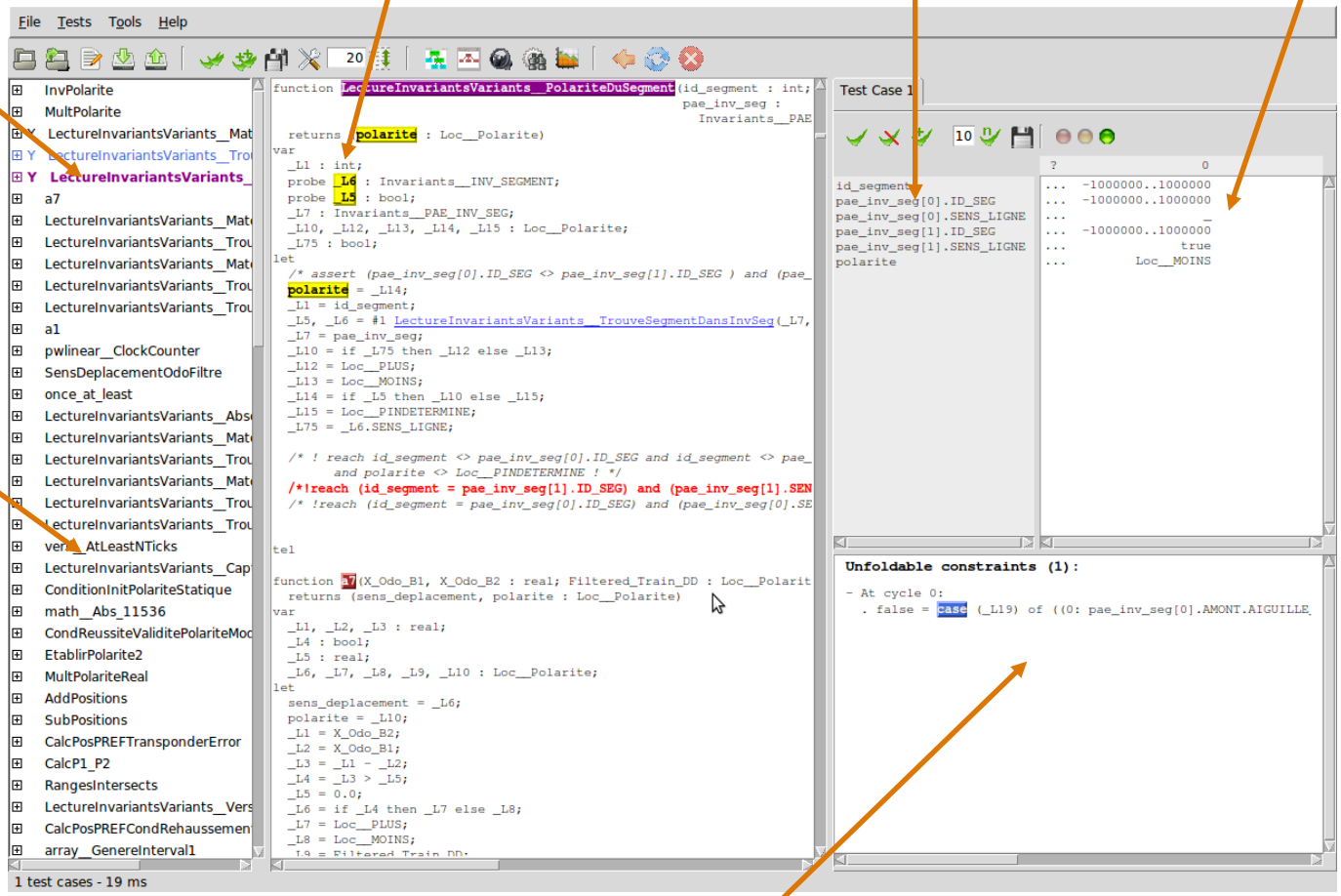
Choix d'un nœud racine

Variables observables

Valeurs courantes

Variables concernées

Liste des nœuds



Contraintes du problème

Plus de détails…

Un BMC standard : domaines finis, séquences bornées

1. **Construction d'un problème :**
   - Traduction « modèle + environnement + objectif » vers un langage intermédiaire « plat » (expansion des nœuds, structures, tableaux …)
   - Simplifications statiques : propagation de constantes, réécritures, analyse statique des expressions numériques (intervalles, variations affines: fonction du nombre de cycles)

     cpt = 0 -> pre cpt + 1 ⟶ cpt :: [0 .. MaxInt], (0+n .. 0+n)

     Exploitation dynamique : cpt >= 10 ⟶ au moins 11 cycles dans le passé

2. **Résolution du problème :**

   Schéma DPLL classique, contraintes généralistes (pas que SAT)
   - Règles de propagation pour chaque construction du langage
   - Intégration du solveur de contraintes numériques COLIBRI
   - Intégration de domaines pour les compteurs, les horloges
   - Génération d'invariants inductifs

Procédure de résolution

On cherche une séquence en arrière depuis le cycle final :

- introduction **paresseuse** des définitions de flots à partir de l'objectif,
- interprétation en **contraintes** des expressions de flots

Procédure de résolution DPLL :

1. Choix d'une variable du système de contraintes (heuristiques)
2. Instanciation dans son domaine
3. Point fixe des règles de simplification/propagation
4. Si échec « backtrack » en 2
5. Si plus de contraintes, alors OK
6. Sinon 1

Exemples de simplifications :

- true = Exp1 and Exp2 $\longrightarrow$ true = Exp1 $\wedge$ true = Exp2
- V = Exp1 -> Exp2 et non initial $\longrightarrow$ V = Exp2
- X = Y + 50 $\wedge$ X € [0..50] $\wedge$ Y € [0..100] $\longrightarrow$ X = 50 $\wedge$ Y = 0

K-induction

En tant que BMC, que dire si pas de solution pour *n* borné?

- **K-Induction classique pour réfutation d'invariants**
- **Induction en propagation**

```
debut = entree and not (false -> pre(entree));
en_route = debut -> if pre(en_route) then entree else debut;
(*! reach  entree and not(en_route) !*)
```

Pas de solution : longueur des séquences insuffisante ?

- **Invariant :** $\forall$n.en_route = entree
- **Observation de la propagation** pour définir un invariant potentiel
  - A chaque cycle on propage $\bigwedge_i v_i = c_i$
  - Est-ce que Prop :: $\bigvee_i v_i <> c_i$ est invariante ?
- **Preuve par k-induction**

  Si ok alors pas de solution pour tout n (fail) et apprentissage de Prop (nouvelle assertion)

Plus de détails : domaines numériques

Caractéristiques communes :

- Domaines finis : variables attribuées, handlers (unify, test_unify, copy …),
- Contraintes : suspensions, évenements de réveil, priorités
- Scheduling dynamique : priorité modifiée selon évenement de réveil

Arithmétique “Entière” : unions d’intervalles, bornes finies, arithmétique d’intervalles, simplifications algébriques, congruences

Arithmétiques “Réelles” : unions d’intervalles de flottants, arithmétique d’intervalles

- Réels : bornes doubles, élargissements vers +/-Inf, simplifications algébriques
- Flottants : bornes simples/doubles (pas de NaN ou zeros signés), round to nearest, simplifications algébriques, propriétés spécifiques (absorption, annulation ..)

Contraintes globales : Difference logic (X-Y =< cste), Simplex

Plus de détails : Entiers

Opérateurs : +, -, x, //, rem, abs, ^n, =, <>, =<, <, >=, >

Exemples de simplifications algébriques :

A + A = B	→	2 x A = B	
A *op* B = C ∧ A *op* B = D	→	C = D ∧ A *op* B = C	*(op est une fonction)*
A + B = C ∧ A + Y = C	→	B = Y ∧ A + B = C	
X =< Y ∧ X <> Y	→	X < Y	

Congruences : (Granger 91, Leconte & Berstel 06)

Insatisfiabilités et réductions de bornes

{ 2X + 4Y + 3Z = 1 (1) Z = 2T + 12 (2) }

(1) : 2X + 4Y ≡ 0[2], 3Z ≡ 1[2] donc **Z ≡ 1[2]**

(2) : **Z ≡ 0[2]**

Plus de détails : Réels / Flottants

Opérateurs : +, -, x, /, sqr, sqrt, =, <>, =<, <, >=, >, conversions int/real

- **Réels :** projections élargies +/- infini
- **Flottants :** projections spécifiques au mode nearest (Michel 2002)

Simplifications algébriques :

- **Réels :** idem Entiers
- **Flottants :** on garde les simplifications compatibles (~~associativité~~, ~~distributivité~~ …)

$A + A = B \rightarrow 2 \times A = B$

$A \mathit{op} B = C \wedge A \mathit{op} B = D \rightarrow C = D \wedge A \mathit{op} B = C$

Propriétés spécifiques aux flottants (absorption, annulation …)

$16.0 + X = 16.0 \rightarrow X : [-8.88\ldots e\text{-}16 .. 1.77\ldots e\text{-}15]$

$X + Y = 0.05 \rightarrow$ X et Y dans [-0.1249… .. 0.175…]