FROM RESEARCH TO INDUSTRY

cea tech

list

SOPRANO Janvier 2015

Benjamin Blanc, Patricia Mouy et Bruno Marre

INSTITUT CARNOT CEA LIST

digiteo

Introduction

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

- **Aujourd'hui: Environnement de V&V pour des modèles Scade6**
  - Extensions successives au gré des projets/études de cas
  - Intégration de différentes fonctionnalités de test et de vérification

- **Basé sur un moteur de résolution dédié aux modèles flots de données synchrones:**
  - Efficace sur des modèles à temporisations
  - Mécanismes dédiés pour la prise en compte des State-Machines
  - Plusieurs arithmétiques cibles: entiers, reéls, flottants , entiers modulaires

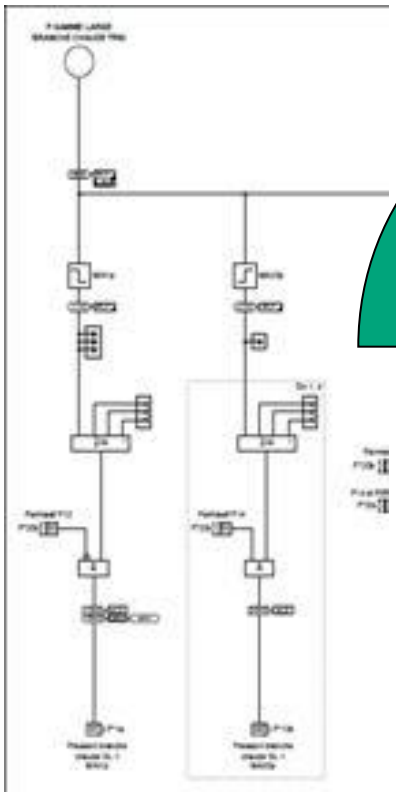
Evaluation des campagnes de test

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

- **Comment :**
  - Modélisation d'un DFP en Lustre
  - Validation du modèle (simulation, comparaison aux sorties de tests constructeur)
  - Identification de cas de test structurels/fonctionnels (en vérifiant leur atteignabilité)
  - Mesure de la couverture des cas de test sur les tests constructeur

Modélisation d'un DFP

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



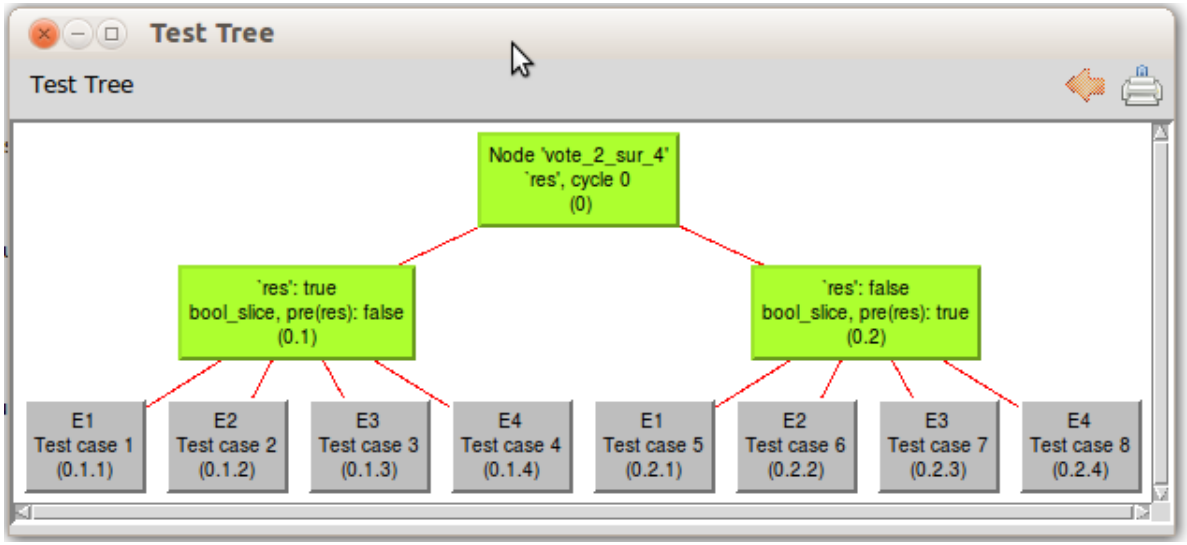
```
node seuil_haut(E,seuil,hys: int) returns (Seuil_haut: bool);
var seuil_inf : int;
let
seuil_inf = seuil - hys;
Seuil_haut = if (E <= seuil_inf)
                then false
                else if E > seuil
                    then true
                    else (false -> pre(Seuil_haut));
tel;
…
```

Couverture de la logique de décision d'une sortie booléenne : MC/DC généralisé et appliqué pour tout basculement

Exemple: vote 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;
```



Identification de cas fonctionnels

Atteignabilité:

- Une sortie devient vraie au dernier cycle (fin d'un processus, etc…)
- Un seuil vient d'être franchi
- Un état d'une state-machine devient actif

Ex: franchissement d'un seuil

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

Exercice d'un scenario déclaratif:

Utilisation d'opérateurs temporels

Ex: pas d'état invalide depuis le franchissement du seuil

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

Exploration interactive de sous-cas

But: Raffiner les cas obtenus

Techniques de raffinement :

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

  Ex: X = **if** Cond **then** Exp1 **else** Exp
  - Cond = true /\ X = Exp1
  - Cond = false /\ X = Exp2

- **Fonctionnelles:** description de scénarios fonctionnels attachés à une variable

  split X **with** [sc1,…,scn]

  Ex: 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

Attachés à des assume de Scade

Propriétés implicites statiques:

- Entrées incompatibles (set/reset, ouvert/fermé, etc)
- Domaine de définition de variables numériques

Propriétés implicites temporelles:

- Domaine de variation d'une entrée ($|x-pre(x)| < \eta$)
- Entrée constante si une autre entrée est vraie

Restrictions à des comportements spécifiques:

Etudier les cas nominaux seulement

Aperçu de l'outil

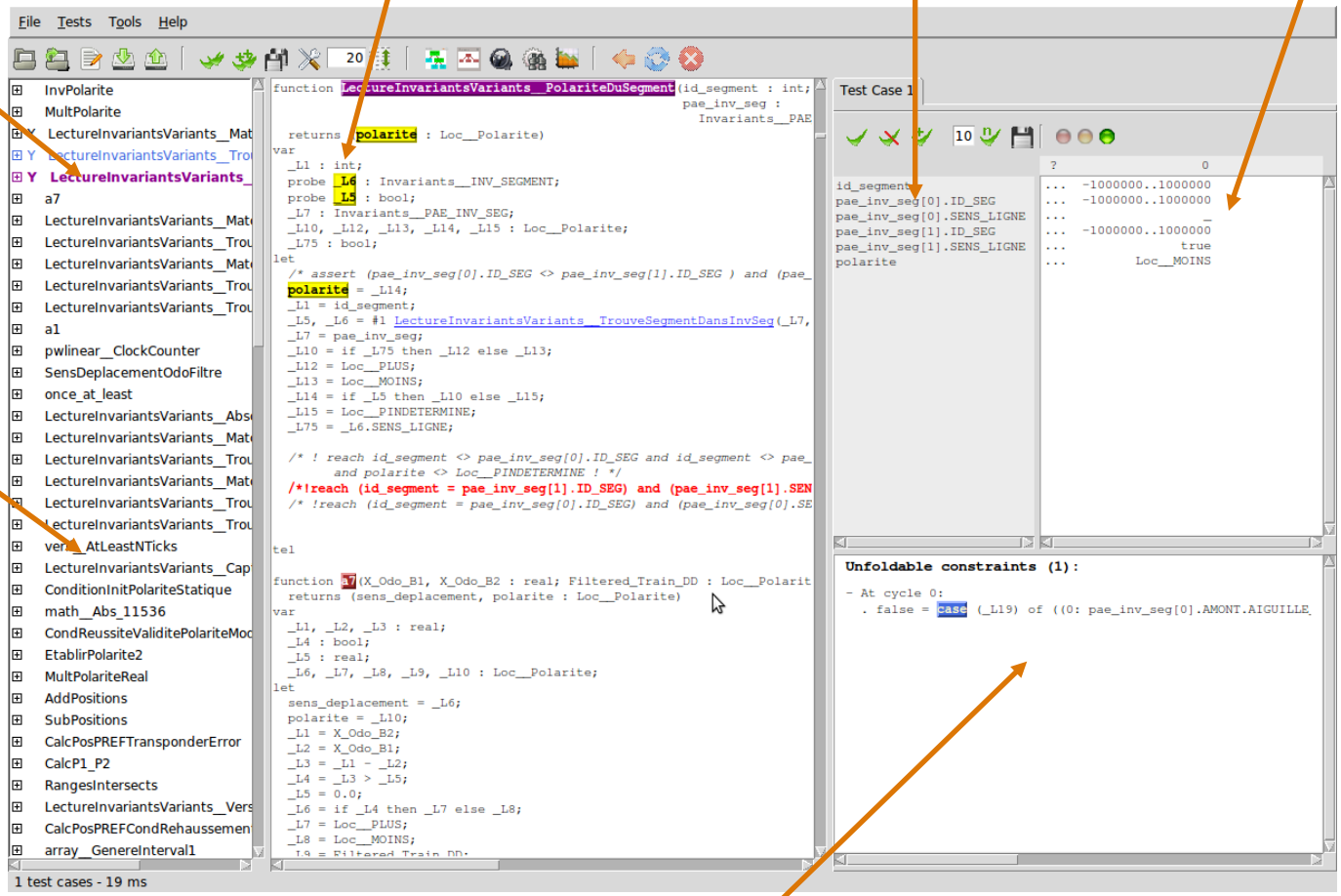
Choix d'un nœud racine

Variables observables

Valeurs courantes

Variables concernées

Liste des nœuds



Contraintes du problème

Plus de détails…

- **Un BMC standard:** (domaines finis, séquences bornées)
  - Constructeur de problèmes de décision
  - Solveur de problème

1. Construction d'un problème:

- Compilation du modèle vers un « $\mathcal{IL}+$ »
- Compilation et prise en compte des contraintes d'environnement (assume)
- Définition manuelle ou interactive d'un objectif

2. Résolution du problème:

- Schéma DPLL classique
- Contraintes généralistes (pas que SAT):
  - Définition de règles de propagation pour chaque construction $\mathcal{IL}+$
  - Intégration du solveur numérique COLIBRI
  - Intégration de domaines pour les compteurs, les horloges
  - Génération d'invariants inductifs

Compilation GATeL

Mise au format du noyau de GATeL:

- Disparition des itérateurs
- Disparition des appels de nœud
- « Mise a plat » des structures, tableaux
- Formatage des sélections dynamiques

Simplifications statiques : propagation de constantes, réécritures, analyse statique

- **`If false then T else E`** est simplifié en **`E`**
- Analyse statique des expressions numériques:
  Intervalles, variations affines (fct du nombre de cycles)
  Ex: Cpt = 0 -> pre cpt + 1 → cpt :: [0 .. MaxInt], (0+n..0+n)

IL+:

- Non-lineaire, overflows/underflows
- Merge, restarts, horloges
- « -> » est un opérateur cible

Procédure de résolution

On cherche une séquence en arrière depuis le cycle final:

Introduction **paresseuse** des définitions 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 de choix)
2. Instanciation dans son domaine (point de choix)
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 => true = Exp1 /\ true = Exp2
- X = Y + 50 /\ X € [0..50] /\ Y € [0..100] => X = 50 /\ Y = 0
- V = Exp1 -> Exp2 et pas initial => V = Exp1

K-induction

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

- **K-Induction 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) !*)
```

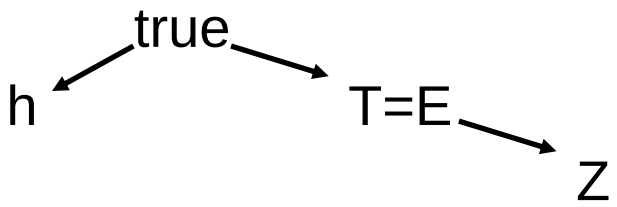
Pas de solution: borne sur la longueur des séquences insuffisante ?

- **Invariant :** $\forall$n.en_route = entree
- **Observation de la propagation** pour définir un invariant potentiel
  - A chaque cycle on propage $/\backslash_i V_i = c_i$
  - Est-ce que Prop :: $\backslash/_i V_i =/= c_i$ est invariante ?
- **Preuve par k-induction**
- Si ok alors pas de solution pour tout n (fail) et apprentissage de Prop

Spécificités: domaine des horloges

- **Les déclarations d'horloges définissent une hiérarchie:**

```
Y: int when h;
Z: bool when (T=E);
W: int when Z;
```



- **Pour chaque horloge H, on mémorise le plus grand numéro de cycle connu ainsi que son statut (init/non_init) à ce cycle**

- **Propriétés sur la hiérarchie:**
  - Ordre (si $H_i \geq H_j$ alors $Max_i \geq Max_j$)
  - Initialisation (si $H_i \geq H_j$, $Max_i = Max_j$, et St_i = init alors St_j = init)
  - Complémentarité (types énumérés)
  - …

- Thèse C. Junke
- Blanc B., Junke C., Marre B., Le Gall P., Andrieu O. « Handling State-Machine Specifications with GATeL » (MBT - 2010)

Plus de détails : domaines numériques dans COLIBRI

Caractéristiques communes :

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

Arithmétiques "Entières": unions d'intervalles, bornes finies,

- **Entiers finis :** arithmétique d'intervalles
- **Entiers modulaires** (2^N) **:** arithmétiques signée et non signée, simulation par des contraintes sur les entiers ($A +_M B = C \Leftrightarrow A + B = C + K* 2^N$)

Arithmétiques "Réelles" : unions d'intervalles de flottants, pas de NaN ou zeros signés

- **Réels :** bornes doubles, élargissement vers +/-Inf
- **Flottants :** bornes simples/doubles, round to nearest

Contraintes globalles :

- **Contraintes de différence** (entiers, réels, flottants) : graphe de distances
- **Simplex** (entiers) : librairie "eplex(clpcbc)" fournie par ECLiPSe

Plus de détails : Entiers

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

- **Simplifications algébriques :**

A + A = B	→	2 x A = B
A x A = B	→	A^2 = B
A *op* B = C ∧ A *op* B = D	→	C = D ∧ A *op* B = C
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
  Modification intrusive du domaine entier (unification, projections).
  Handlers prioritaires aux intervalles d'entiers.

  { 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 nearest (Michel 2002)

- **Simplifications algébriques :**
  - **Réels :** idem Entiers
  - **Flottants :** on garde les simplifications compatibles (non-associativité …)

$$A + A = B \rightarrow 2 \times A = B$$
$$A \; op \; B = C \wedge A \; op \; B = D \rightarrow C = D \wedge A \; op \; B = C$$
$$X =< Y \wedge X <> Y \rightarrow X < Y$$

- **Propriétés spécifiques aux flottants (absorbtion, annulation …)**

$$16.0 + X = 16.0 \rightarrow X : [-8.88\ldots e\text{-}16 \; .. \; 1.77\ldots e\text{-}15]$$
$$X + Y = 0.05 \rightarrow X \text{ et } Y \text{ dans } [-0.1249\ldots \; .. \; 0.175\ldots]$$

- Michel C. « Exact projection functions for floating point number constraints » IAMAI, 2002.
- Marre B. Michel C. « Improving the floating point addition and subtraction constraints » CP 2010

Plus de détails : Contraintes de différence

Graphe de distances entre variables (entiers, réels, flottants)

Sommets : variables numériques

Arcs : intervalles encadrant les distances min/max

= D : distance exacte
Min..Max : Min =< D =< Max
Min..Max : Min =< D =< Max et D <> 0

Construction : extraction des *déplacements* induits par les contraintes

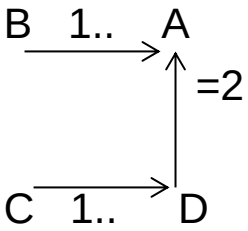
Insatisfiabilités : détection de cycles à coût non nul

Normalisation du graphe : instanciation, unification **…**

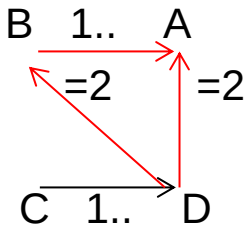
Lecture du graphe : les contraintes peuvent se transformer, réduire des domaines en observant les arcs de leurs arguments

Analyse du graphe : Nouvel arc entre deux sommets connexes, ou réduction d'une distance entre deux sommets d'un cycle connu

{ A > B, C > D, A = D + 2 }



si ajout de B = D + 2



Plus de détails : Interactions, collaborations de domaines

- **Intervalles d'entiers, Congruences :** réductions de bornes

- **Intervalles d'entiers, Distances :** réductions de bornes + projections / unions

  A+B = C, A: $[min_a..a_1, a_2..max_a]$, B: $[min_b..b_1, b_2..max_b]$, d(A,B) = 0..n, $a_2 > b_1$
  $\rightarrow$ C' : dom(C) $\cap$ ($[min_a..a_1]+[min_b..b_1, b_2..max_b] \cup [a_2..max_a]+ [b_2..max_b]$)

- **Réels/Flottants, Distances** (en nombre de flottants) **:** Projections / unions

- **Distances, Simplex, Intervalles d'entiers :**
  - Appel de min/max sur réduction de borne (min/max) d'un sommet du graphe avant analyse du graphe
  - Si succès, exploitation du min/max calculé pour réduction du domaine