Vers une Théorie du Test des Programmes à Contrainte

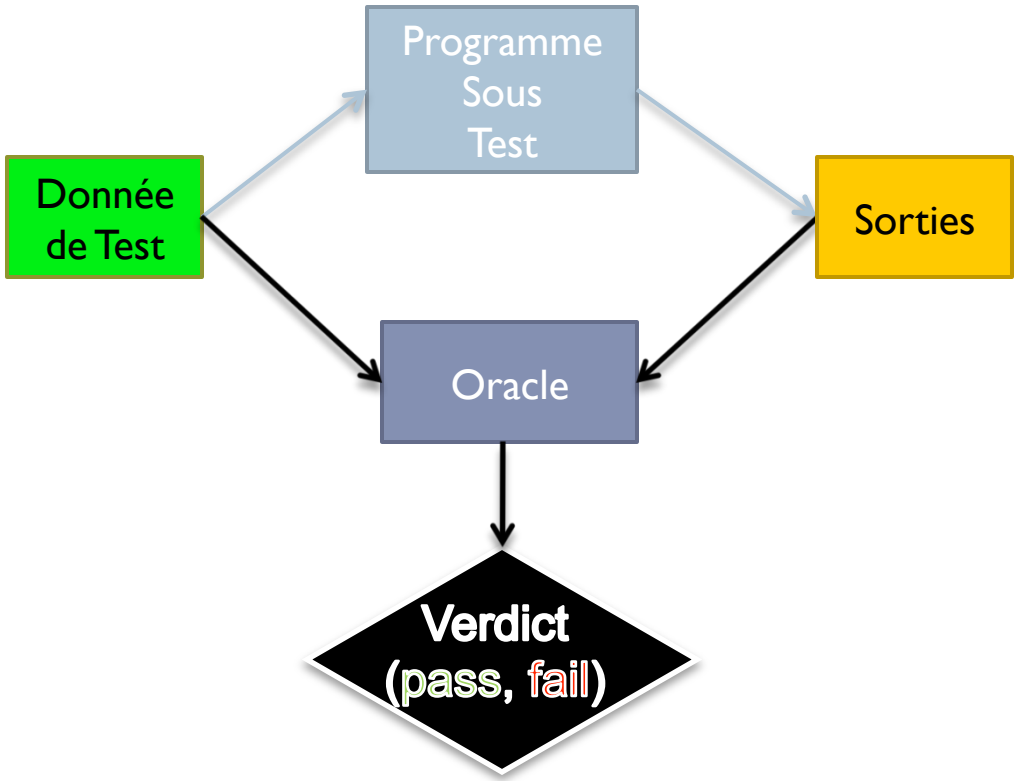
Nadjib LAZAAR*, A. GOTLIEB*, Y. LEBBAH**

*INRIA Rennes Bretagne Atlantique ** Université d'Oran Es-Senia

JFPC 09

IRISA

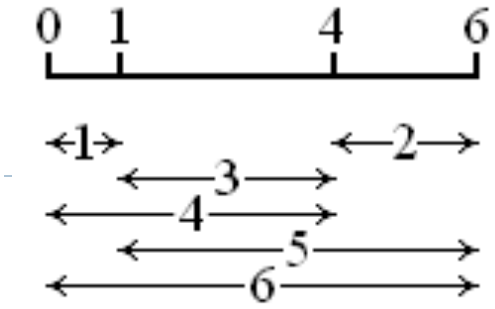
Test Logiciel



Développement PPC

- Langages et plateformes commercialisés :
  - **OPL** (Optimization Programming Language)
  - **COMET** (www.Dynadec.net)
  - **SICStus Prolog**
- Développement d'outils académiques :
  - **ZINC** [Nethercote et al, CP07][Marriott et al, Constraints, 08]
  - **GECODE** [Schulte, Phd, 00][Tack, Phd, 09]
  - **CHOCO** [Laburthe et al, JNPC00]

Règles de Golomb



M

```
int m=...;
dvar int x[1..m] in 0..m*m;
minimize x[m];
subject to
{
(1)   forall (i in 1..m-1)
         x[i] < x[i+1];
(2)   forall (i in 1..m, j in 1..m,
              k in 1..m, l in 1..m:
              (i < j, k < l))
         x[j] - x[i] != x[l] - x[k];
}
```

P

```
int m=...;
dvar int x[1..m] in 0..m*m;
tuple indexerTuple { int i;  int j;}
{indexerTuple} indexes = {<i,j> | i,j in 1..m : i<j};
dvar int d[indexes];
minimize x[m];
subject to {
 (1) forall (i in 1..m-1) x[i] < x[i+1];
 (2) forall(ind in indexes) d[ind]==x[ind.j]-x[ind.i];
 (3) x[1]=0;
 (4) x[m] >= (m * (m - 1)) / 2;
 (5) allDifferent(all(ind in indexes ) d[ind]);
 (6) x[2] <= x[m]-x[m-1];
 (7) forall(ind1 in indexes, ind2 in indexes, ind3 in
indexes :
(ind1.i==ind2.i)&&(ind2.j==ind3.j)&&(ind1.j==ind3.j)&&
 ( ind1.i<ind2.j<ind1.j))  d[ind1]=d[ind2]+d[ind3];
 (8) forall(ind1,ind2,ind3,ind4 in indexes :
(ind1.i==ind2.i)&&(ind1.j==ind3.j)&&(ind2.j==ind4.j)&&
(ind3.i==ind4.i)&&(ind1.i<m-1)&&(3<ind1.j<m+1)&&
 (2<ind2.j<m)&&(1<ind3.i<m-1)&&(ind1.i < ind3.i <
ind2.j < ind1.j))  d[ind1]==d[ind2]+d[ind3]-d[ind4];
 (9) forall(i in 2..m, j in 2..m, k in 1..m : i  < j)
x[i]=x[i-1]+k => x[j] != x[j-1]+k;
}
```

Plan

I Introduction

II Notations et Relation de Conformité

III Critères de Test

IV Validation Expérimentale

V Conclusion

II- Notations

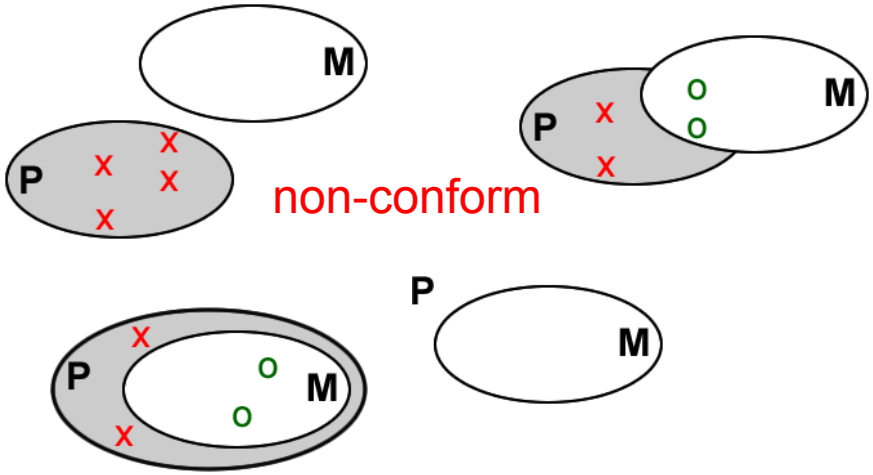
$M_x(k) \equiv C_1(x) \wedge \ldots C_n(x).\ k \in \mathcal{K}$

- sol(M): ensemble des solutions.
- $quasi_f(M,I)$: ensemble des quasi-optimaux.

II- Relation de Conformité

- Une Seule Solution ($conf_{one}$):

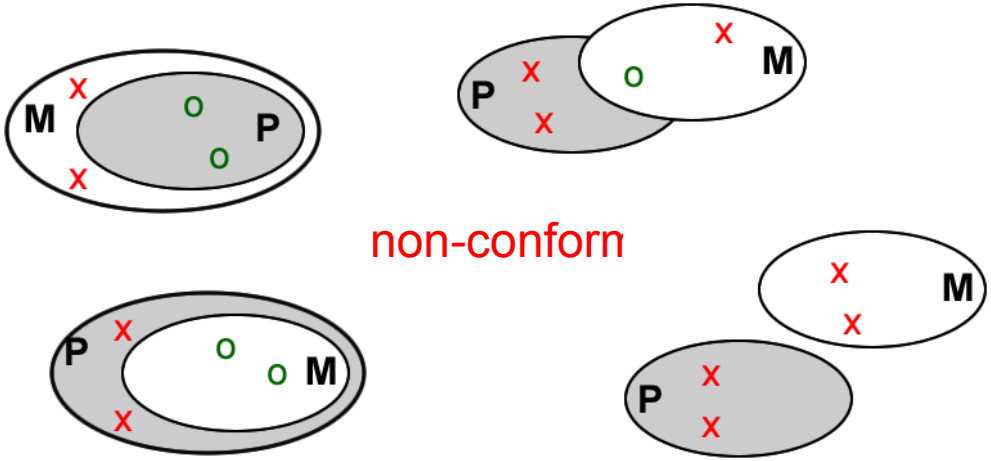
$P\ conf_{one}\ M \Leftrightarrow \forall k \in \mathcal{K}: sol(P_x(k)) \neq \emptyset \wedge sol(P_x(k)) \subseteq sol(M_x(k))$

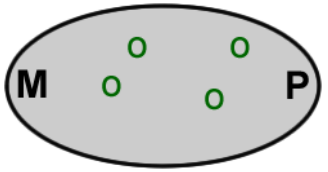


II- Relation de Conformité

- Toutes les Solutions ($conf_{all}$):

$P\ conf_{all}\ M \Leftrightarrow \forall\ k \in \mathcal{K}:\ \ sol(P_x(k)) = sol(M_x(k))$



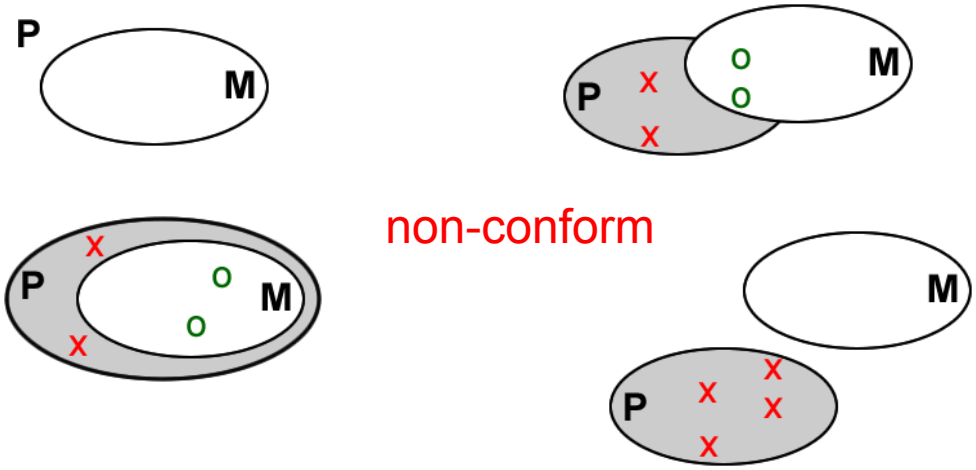


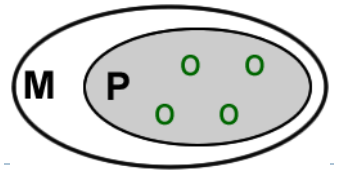
conform

II- Relation de Conformité

- Une Meilleure Solution ($conf_{best}$):

$$P\ conf_{best}\ M \Leftrightarrow \forall\ k \in \mathcal{K} : quasi_f(P_x(k),I) \neq \emptyset \wedge\ quasi_f(P_x(k),I) \subseteq quasi_f(M_x(k),I)$$





conform

II- Relation de Conformité

$$P\ conf_{one}\ M \Leftrightarrow \forall k \in \mathcal{K}: sol(P_x(k)) \neq \emptyset \wedge sol(P_x(k)) \subseteq sol(M_x(k))$$

$$P\ conf_{all}\ M \Leftrightarrow \forall k \in \mathcal{K}: sol(P_x(k)) = sol(M_x(k))$$

$$P\ conf_{best}\ M \Leftrightarrow \forall k \in \mathcal{K}: quasi_f(P_x(k),l) \neq \emptyset \wedge quasi_f(P_x(k),l) \subseteq quasi_f(M_x(k),l)$$

TEST de Non-conformité

II- Relation de Conformité

- Exemple:

Paramètres Variables

$$(\forall x \in R: x = val) \wedge (\forall y \in X / R: y \neq val) \quad ,R \subseteq X: |R| = N$$

(val=3, N=2):

$M_x((2,3))$:

```
(x[1]==3; x[2]==3; x[3] !=3)||
(x[1]==3; x[2]!=3; x[3] ==3)||
(x[1]!=3; x[2]==3; x[3] ==3);
```

$P_x((2,3))$:

```
forall(i in 3)
count(3, x[i]) =< 2;
```

```
atmost(2, 3, X) in OPL 6.x
```

II- Relation de Conformité

- Exemple:

Paramètres Variables

$$(\forall\, x \in R:\ x = val) \wedge (\forall\, y \in X / R:\ y \neq val) \qquad ,R \subseteq X: |R| = N$$

(val=3, N=2):

$M_x((2,3))$:

```
(x[1]==3; x[2]==3; x[3] !=3)||
(x[1]==3; x[2]!=3; x[3] ==3)||
(x[1]!=3; x[2]==3; x[3] ==3);
```

$P_x((2,3))$:

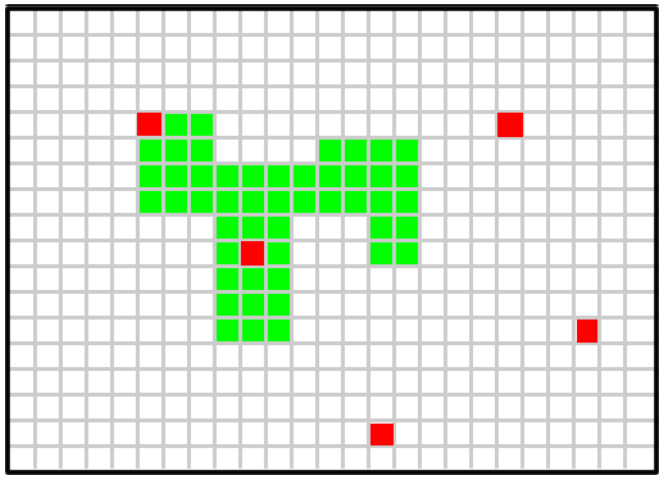
```
forall(i in 3)
count(3, x[i]) =< 2;
```

```
x[1]==3; x[2]==9; x[3]==84;
```

Point de non-conformité

III- Critères de Test

Fautes



Critère de Test A

Données de Test

III- Critères de Test

- 1er Critère de test : Toutes_les_contraintes
- 2ème Critère de test : Tous_les_reveils
- 3ème Critère de test : Une_contrainte_niée

III- Critères de Test

- Toutes_les_contraintes

Le critère Toutes_les_contraintes exige que chaque contrainte d'un modèle P ait contribué au moins une fois à la résolution du système de contraintes.

$P_x(m) \equiv$

```
(1)  forall( i in 1..m-1)
        x[i] < x[i+1];

(2)  if(m < 10)
      then
        {x[m]==10;}
      else
        {x[m]==100;}
```

III- Critères de Test

- Tous_les_reveils

Le critère Tous_les_reveils exige que chaque contrainte d'un modèle P ait contribué au moins deux fois à la résolution du système de contraintes.

III- Critères de Test

- Une_contrainte_niée

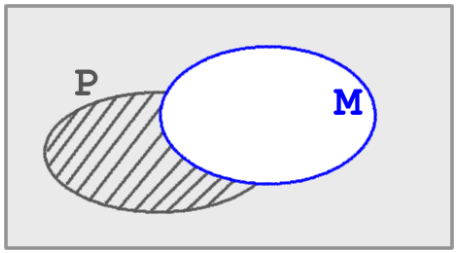
Le critère Une_contrainte_niée exige que :

$\exists i,\ sol(P_x(k) \wedge \neg C_i) \neq \emptyset$ une seule solution

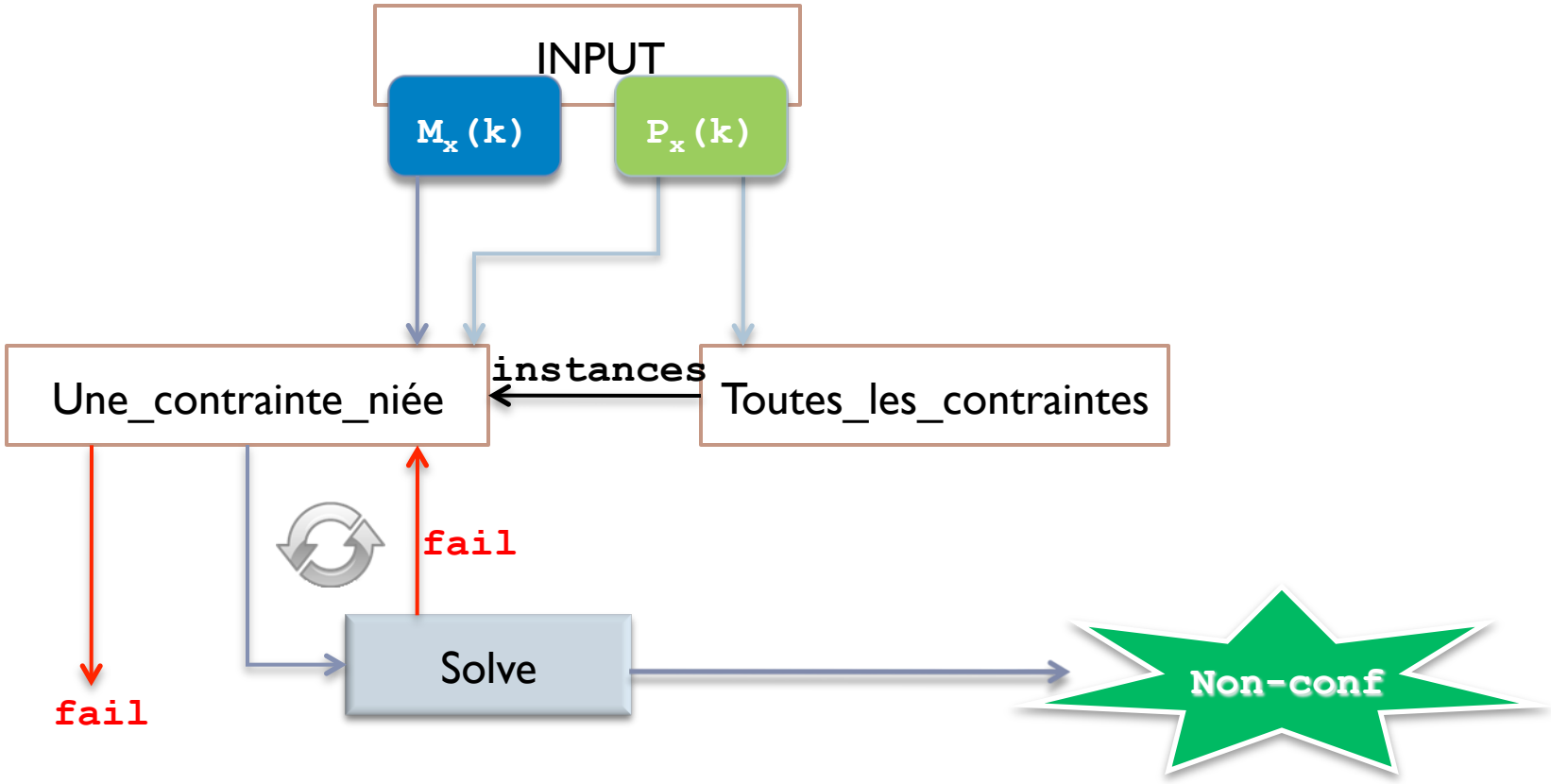
$\exists\ i,j\ sol(P_x(k) \wedge \neg C_i \vee M_x(k) \wedge \neg C_j') \neq \emptyset$ toutes les solutions

$\exists i,\ quasi_f((P_x(k) \wedge \neg C_i), I) \neq \emptyset$ une meilleure solution

$\exists\ k, x: P_x(k) \wedge \neg M_x(k)$ ⇒ $\exists\ k, x: P_x(k) \wedge \neg C_1$ $\vee$ ... $P_x(k) \wedge \neg C_i$ $\vee$ ... $P_x(k) \wedge \neg C_n$



Processus de Test



IV- Validation (Règles de Golomb)

M

```
int m=...;
dvar int x[1..m] in 0..m*m;
minimize x[m];
subject to
{
(1)  forall (i in 1..m-1)
        x[i] < x[i+1];
(2)  forall (i in 1..m, j in 1..m,
           k in 1..m, l in 1..m:
           (i < j, k < l))
        x[j] - x[i] != x[l] - x[k];
}
```

P

```
int m=...;
dvar int x[1..m] in 0..m*m;
tuple indexerTuple { int i;  int j;}
{indexerTuple} indexes = {<i,j> | i,j in 1..m : i<j};
dvar int d[indexes];
minimize x[m];
subject to {
 (1) forall (i in 1..m-1) x[i] < x[i+1];
 (2) forall(ind in indexes) d[ind]==x[ind.j]-x[ind.i];
 (3) x[1]=0;
 (4) x[m] >= (m * (m - 1)) / 2;
 (5) allDifferent(all(ind in indexes ) d[ind]);
 (6) x[2] <= x[m]-x[m-1];
 (7) forall(ind1 in indexes, ind2 in indexes, ind3 in
indexes :
(ind1.i==ind2.i)&&(ind2.j==ind3.j)&&(ind1.j==ind3.j)&&
 ( ind1.i<ind2.j<ind1.j))  d[ind1]=d[ind2]+d[ind3];
 (8) forall(ind1,ind2,ind3,ind4 in indexes :
(ind1.i==ind2.i)&&(ind1.j==ind3.j)&&(ind2.j==ind4.j)&&
(ind3.i==ind4.i)&&(ind1.i<m-1)&&(3<ind1.j<m+1)&&
 (2<ind2.j<m)&&(1<ind3.i<m-1)&&(ind1.i < ind3.i <
ind2.j < ind1.j))  d[ind1]==d[ind2]+d[ind3]-d[ind4];
 (9) forall(i in 2..m, j in 2..m, k in 1..m : i  < j)
x[i]=x[i-1]+k => x[j] != x[j-1]+k;
}
```

IV- Validation (Règles de Golomb)

k = 8	Faute injectée	non_conf	Violation	T (ms)	M (Mo)
CPUT1	`d[ind] != x[ind.j]-d[ind.i]`	0 1 3 6 10 15 21 30	3-0=6-3	630	1.4
		0 1 8 12 14 19 25 28	8-1=19-12	310	1.4
CPUT2	`d[ind] == x[ind.j]+x[ind.i]`	0 2 8 11 16 20 21 42	16-11=21-16	3 020	3.8
		0 1 7 16 20 25 28 30	25-20=30-25	2 950	1.3
CPUT3	`x[i] == x[i-1]+k || x[j] != x[j-1]+k`	---	---	46 390	32.5
		---	---	50 150	34.2

IV- Validation (Ordonnancement des véhicules)

- Ordonnancement des véhicules

$M_x(k)$

```
int  nbCars    = ...;
int   nbOptions = ...;
int   nbSlots   = ...;
range   Cars    = 1..nbCars;
range   Options = 1..nbOptions;
range   Slots   = 1..nbSlots;
int demand[Cars] = ...;
int option[Options,Cars] = ...;
tuple Tcapacity { int l; int u; };
Tcapacity capacity[Options] = ...;
int optionDemand[i in Options] = sum(j in Cars)
demand[j] * option[i,j];
dvar int slot[Slots] in Cars;
subject to {
(1) forall(c in Cars )
      sum(s in Slots ) (slot[s] == c) == demand[c];
(2) forall(o in Options, s in 1..(nbSlots -
capacity[o].u + 1) )
      sum(j in s..(s + capacity[o].u - 1))
        option[o][slot[j]] <= capacity[o].l;
};
```

$P_x(k)$

```
int  nbCars    = ...;
int   nbOptions = ...;
int   nbSlots   = ...;
range   Cars    = 1..nbCars;
range   Options = 1..nbOptions;
range   Slots   = 1..nbSlots;
int demand[Cars] = ...;
int option[Options,Cars] = ...;
tuple Tcapacity { int l; int u; };
Tcapacity capacity[Options] = ...;
int optionDemand[i in Options] = sum(j in Cars)
demand[j] * option[i,j];
dvar int slot[Slots] in Cars;
dvar int setup[Options,Slots] in 0..1;
subject to {
(1)  forall(c in Cars )
      sum(s in Slots ) (slot[s] == c) ==
demand[c];
(2)  forall(o in Options, s in 1..(nbSlots -
capacity[o].u + 1) )
      sum(j in s..(s + capacity[o].u - 1))
setup[o,j] <= capacity[o].l;
(3)  forall(o in Options, s in Slots )
      setup[o,s] == option[o][slot[s]];
(4)  forall(o in Options, i in
1..optionDemand[o])
      sum(s in 1 .. (nbSlots - i *
capacity[o].u)) setup[o,s] >= optionDemand[o] -
i * capacity[o].l;
};
```

IV- Validation (Ordonnancement des véhicules)

- Injection de fautes :

CPUT1	`setup[o,j]`	written	`setup[o,o]`
CPUT2	`capacity[o].l`	written	`capacity[o].u`
CPUT3	`capacity[o].u+1`	written	`capacity[o].u-1`
CPUT4	`capacity[o].u`	written	`capacity[o].l`
CPUT5	`s in Slots`	written	`s in Cars`
CPUT6	`c in Cars`	written	`c in Options`

IV- Validation (Ordonnancement des véhicules)

- Instance de couverture :

```
nbCars = 6;
nbOptions = 5;
nbSlots = 10;
demand = [1, 1, 2, 2, 2, 2];
option = [
            [ 1, 0, 0, 0, 1, 1],
            [ 0, 0, 1, 1, 0, 1],
            [ 1, 0, 0, 0, 1, 0],
            [ 1, 1, 0, 1, 0, 0],
            [ 0, 0, 1, 0, 0, 0]
         ];
capacity =
    [<1,2>,<2,3>,<1,3>,<2,5>,<1,5>];
```

IV- Validation (Ordonnancement des véhicules)

	non-conf	CC op.1	CC op.2	CC op.3	CC op.4	CC op.5
CPUT1	2 1 6 3 5 3 5 4 4	V	V	S	S	V
	2 1 6 3 4 5 3 6 4 5	V	V	S	V	V
CPUT2	3 6 5 1 2 4 3 5 4 6	V	S	V	V	S
	3 6 4 1 2 5 3 5 4 6	S	V	V	V	S
CPUT3	---	---	---	---	---	---
	4 5 3 6 2 5 4 6 1 3	V	S	S	V	S
CPUT4	---	---	---	---	---	---
	---	---	---	---	---	---
CPUT5	4 5 2 6 3 5 1 3 6 4	V	V	V	S	V
	4 6 2 5 3 6 1 5 4 3	V	S	V	S	S
CPUT6	---	---	---	---	---	---
	---	---	---	---	---	---

V- Conclusion

- Nous avons présenté les bases d'une première théorie du test pour les programmes à contraintes avec :
  - Relation de conformité
  - critères et processus de test et avec une première validation expérimentale

- Perspectives:
  - Définition d'autres critères de test
  - Traitement de la négation des contraintes
  - Automatisation du processus de test.

Merci pour votre attention!