cptest4choco : test et mise au point des modèles à contraintes

Nadjib Lazaar*, Arnaud Gotlieb, Yahia Lebbah

*CNRS-LIRMM -- lazaar@lirmm.fr

12 Juin 2013

Langages de Modélisation en PPC

- conçus avec un haut niveau d'abstraction permettant une grande expressivité (OPL, COMET, ZINC, CHOCO...)
- proposent des solutions robustes à des problèmes réels.
- sont utilisés dans des applications critiques

Langages de Modélisation en PPC

- conçus avec un haut niveau d'abstraction permettant une grande expressivité (OPL, COMET, ZINC, CHOCO...)
- proposent des solutions robustes à des problèmes réels.
- sont utilisés dans des applications critiques

Planication des tâches de production
(Sanlaville, ROADEF05)

Langages de Modélisation en PPC

- conçus avec un haut niveau d'abstraction permettant une grande expressivité (OPL, COMET, ZINC, CHOCO…)
- proposent des solutions robustes à des problèmes réels.
- sont utilisés dans des applications critiques

Planication des tâches de production
(Sanlaville, ROADEF05)

Systèmes de recommendation
(McSherry & Aha, IJCAI07; Felfernig & Burke, ICEC08)

Enchères en ligne
(Lozano et al., CP10)

e-Portfeuille
(Flener et al., Constraints07)

Langages de Modélisation en PPC

- conçus avec un haut niveau d'abstraction permettant une grande expressivité (OPL, COMET, ZINC, CHOCO...)
- proposent des solutions robustes à des problèmes réels.
- sont utilisés dans des applications critiques

Planication des tâches de production
(Sanlaville, ROADEF05)

Systèmes de recommendation
(McSherry & Aha, IJCAI07; Felfernig & Burke, ICEC08)
Enchères en ligne
(Lozano et al., CP10)
e-Portfeuille
(Flener et al., Constraints07)

Trafic aérien
(Flener et al., JATM07)
Trafic ferroviaire
(Chiu et al., Constraints02; Rodriguez & Kermad, Comprail98)

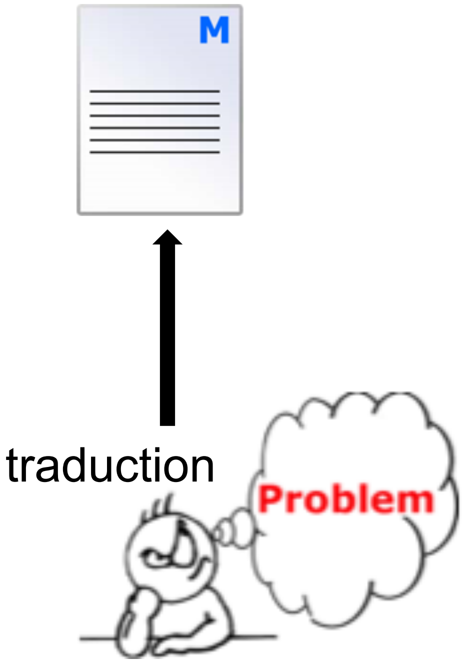
LANGAGES DE MODÉLISATION EN PPC

- conçus avec un haut niveau d'abstraction permettant une grande expressivité (OPL, COMET, ZINC, CHOCO...)
- proposent des solutions robustes à des problèmes réels.
- sont utilisés dans des applications critiques

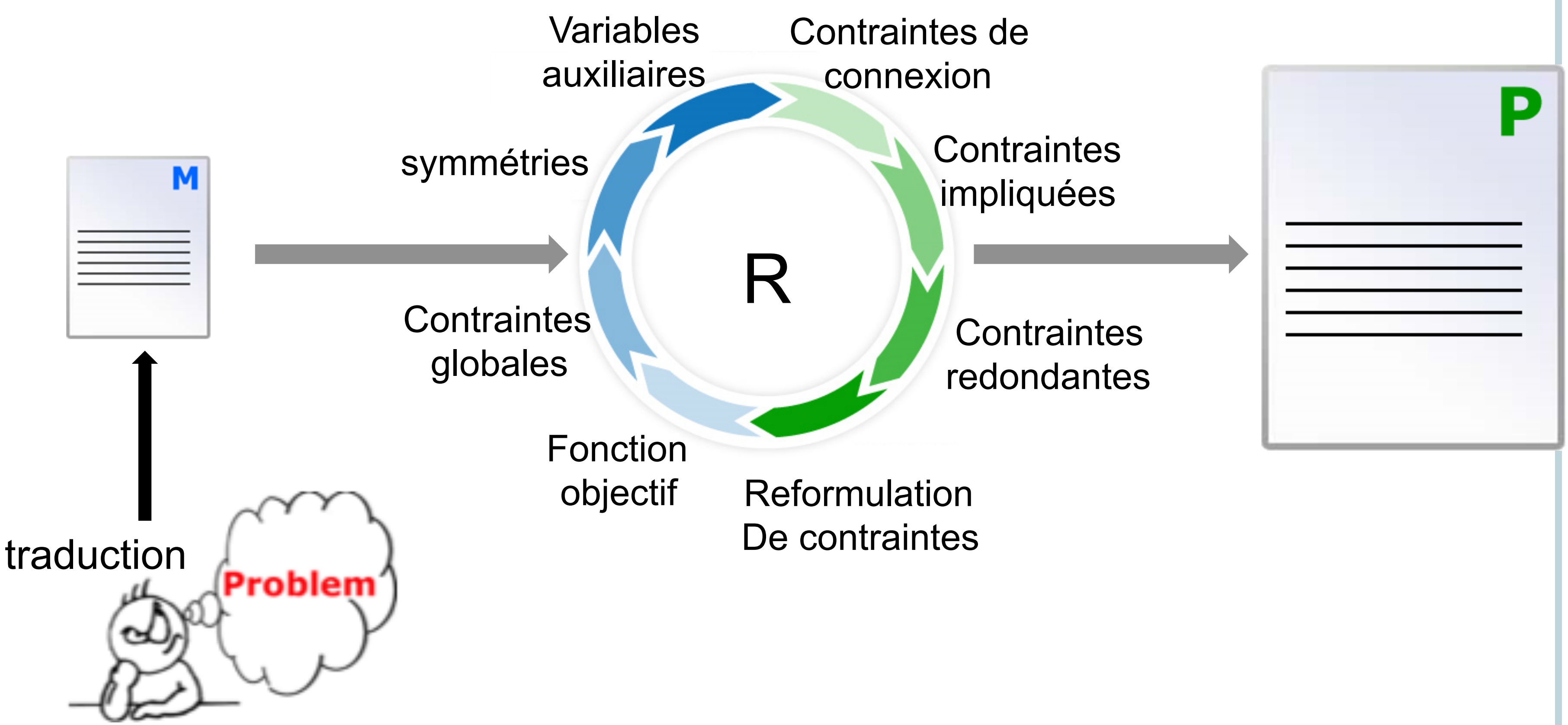
Le besoin,

•Test:	détection de faute
•Localisation:	explication de faute
•Correction:	reformulation de contraintes

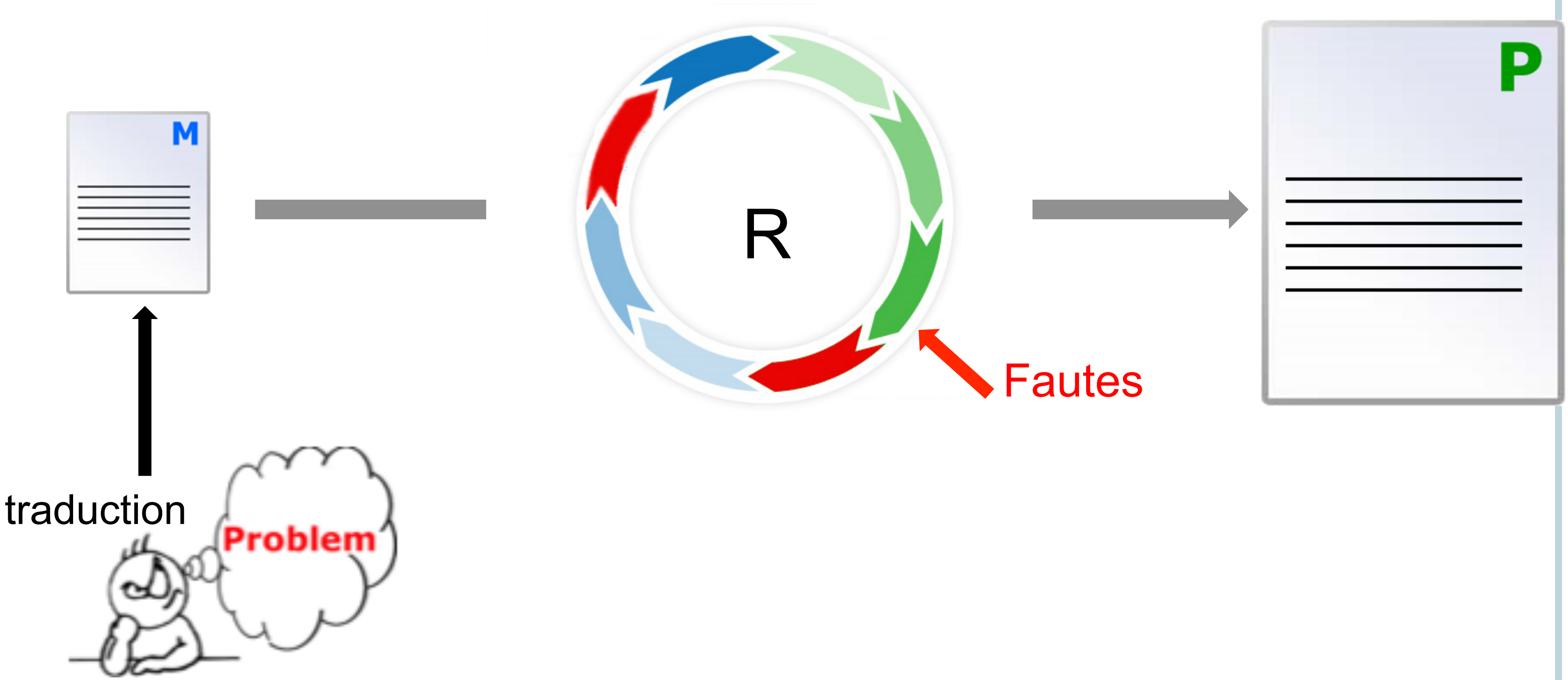
Raffinement en PPC



RAFFINEMENT EN PPC

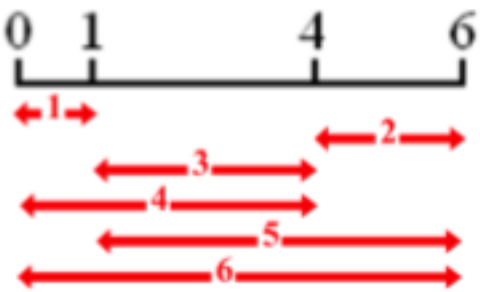


RAFFINEMENT EN PPC



Exemple : Règles de Golomb

- Une règle de Golomb est définie comme un ensemble de m entiers $0 = x_1 < x_2 < \ldots < x_m$ tel que les $m(m-1)/2$ distances $\{x_j - x_i \mid i < j\}$, sont différentes. Une telle règle d'ordre m est dite de longueur x_m . L'objectif est de trouver une règle de longueur minimale.



Exemple : Règles de Golomb

M

1)
```
MO.addConstraint(Choco.lt(marks[i], marks[i+1]));
```

2)
```
MO.addConstraint(Choco.neq(Choco.minus(marks[j],marks[i]),
                 Choco.minus(marks[l],marks[k])));
```

Exemple : Règles de Golomb

M

1)
```
MO.addConstraint(Choco.lt(marks[i], marks[i+1]));
```
2)
```
MO.addConstraint(Choco.neq(Choco.minus(marks[j],marks[i]),
                 Choco.minus(marks[l],marks[k])));
```

P

1)
```
CPUT.addConstraint(Choco.lt(marks[i], marks[i+1]));
```
2)
```
CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
                   Choco.minus(marks[j-1],marks[i-1])));
```
3)
```
CPUT.addConstraint(Choco.geq(marks[m-1], m*(m-1)/2));
```
4)
```
CPUT.addConstraint(Choco.leq(marks[1], Choco.minus(marks[m-1], marks[m-2])));
```
5)
```
CPUT.addConstraint(Choco.allDifferent(distance));
```
6)
```
CPUT.addConstraint(Choco.eq(marks[0], 0));
```
7)
```
CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
        Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
            distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```
8)
```
CPUT.addConstraint(Choco.eq(
    Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (l-i))],
    distance[((m*(m-1) / 2) - ((m-l+1)*(m-l) / 2) + (j-l))]),
    Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
        distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```

Exemple : Règles de Golomb

M

```
1) MO.addConstraint(Choco.lt(marks[i], marks[i+1]));
2) MO.addConstraint(Choco.neq(Choco.minus(marks[j],marks[i]),
                    Choco.minus(marks[l],marks[k])));
```

a)- **P** est il conforme à **M** ?

P

```
1) CPUT.addConstraint(Choco.lt(marks[i], marks[i+1]));
2) CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
                        Choco.minus(marks[j-1],marks[i-1])));
3) CPUT.addConstraint(Choco.geq(marks[m-1], m*(m-1)/2));
4) CPUT.addConstraint(Choco.leq(marks[1], Choco.minus(marks[m-1], marks[m-2])));
5) CPUT.addConstraint(Choco.allDifferent(distance));
6) CPUT.addConstraint(Choco.eq(marks[0], 0));
7) CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
              Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
                  distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
8) CPUT.addConstraint(Choco.eq(
        Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (l-i))],
        distance[((m*(m-1) / 2) - ((m-l+1)*(m-l) / 2) + (j-l))]),
        Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
            distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```

EXEMPLE : RÈGLES DE GOLOMB

M

1)
```
MO.addConstraint(Choco.lt(marks[i], marks[i+1]));
```
2)
```
MO.addConstraint(Choco.neq(Choco.minus(marks[j],marks[i]),
                 Choco.minus(marks[l],marks[k])));
```

a)- P est il conforme à M ?

m=8

X= [0 1 3 6 10 26 27 28]

P

1)
```
CPUT.addConstraint(Choco.lt(marks[i], marks[i+1]));
```
2)
```
CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
                   Choco.minus(marks[j-1],marks[i-1])));
```
3)
```
CPUT.addConstraint(Choco.geq(marks[m-1], m*(m-1)/2));
```
4)
```
CPUT.addConstraint(Choco.leq(marks[1], Choco.minus(marks[m-1], marks[m-2])));
```
5)
```
CPUT.addConstraint(Choco.allDifferent(distance));
```
6)
```
CPUT.addConstraint(Choco.eq(marks[0], 0));
```
7)
```
CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
           Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
                 distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```
8)
```
CPUT.addConstraint(Choco.eq(
      Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (l-i))],
      distance[((m*(m-1) / 2) - ((m-l+1)*(m-l) / 2) + (j-l))]),
      Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
           distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```

EXEMPLE : RÈGLES DE GOLOMB

M

1)
```
MO.addConstraint(Choco.lt(marks[i], marks[i+1]));
```
2)
```
MO.addConstraint(Choco.neq(Choco.minus(marks[j],marks[i]),
                 Choco.minus(marks[l],marks[k])));
```

a)- P est il conforme à M ?

P

1)
```
CPUT.addConstraint(Choco.lt(marks[i], marks[i+1]));
```
2)
```
CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
                 Choco.minus(marks[j-1],marks[i-1])));
```
3)
```
CPUT.addConstraint(Choco.geq(marks[m-1], m*(m-1)/2));
```
4)
```
CPUT.addConstraint(Choco.leq(marks[1], Choco.minus(marks[m-1], marks[m-2])));
```
5)
```
CPUT.addConstraint(Choco.allDifferent(distance));
```
6)
```
CPUT.addConstraint(Choco.eq(marks[0], 0));
```
7)
```
CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
           Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
                 distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```
8)
```
CPUT.addConstraint(Choco.eq(
      Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (l-i))],
      distance[((m*(m-1) / 2) - ((m-l+1)*(m-l) / 2) + (j-l))]),
      Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
           distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```

m=8

X= [0 1 3 6 10 26 27 28]

1 1

Faute détectée dans P!

EXEMPLE : RÈGLES DE GOLOMB

M

1)
```
MO.addConstraint(Choco.lt(marks[i], marks[i+1]));
```
2)
```
MO.addConstraint(Choco.neq(Choco.minus(marks[j],marks[i]),
                Choco.minus(marks[l],marks[k])));
```

a)- P est il conforme à M ?

P

1)
```
CPUT.addConstraint(Choco.lt(marks[i], marks[i+1]));
```
2)
```
CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
                Choco.minus(marks[j-1],marks[i-1])));
```
3)
```
CPUT.addConstraint(Choco.geq(marks[m-1], m*(m-1)/2));
```
4)
```
CPUT.addConstraint(Choco.leq(marks[1], Choco.minus(marks[m-1], marks[m-2])));
```
5)
```
CPUT.addConstraint(Choco.allDifferent(distance));
```
6)
```
CPUT.addConstraint(Choco.eq(marks[0], 0));
```
7)
```
CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
        Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
            distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```
8)
```
CPUT.addConstraint(Choco.eq(
    Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (l-i))],
    distance[((m*(m-1) / 2) - ((m-l+1)*(m-l) / 2) + (j-l))]),
    Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
        distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```

b)-Où est la faute ?

EXEMPLE : RÈGLES DE GOLOMB

M

```
1)  MO.addConstraint(Choco.lt(marks[i], marks[i+1]));
2)  MO.addConstraint(Choco.neq(Choco.minus(marks[j],marks[i]),
                          Choco.minus(marks[l],marks[k])));
```

a)- P est il conforme à M ?

P

```
1)  CPUT.addConstraint(Choco.lt(marks[i], marks[i+1]));
2)  CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
                                    Choco.minus(marks[j-1],marks[i-1])));
3)  CPUT.addConstraint(Choco.geq(marks[m-1], m*(m-1)/2));
4)  CPUT.addConstraint(Choco.leq(marks[1], Choco.minus(marks[m-1], marks[m-2])));
5)  CPUT.addConstraint(Choco.allDifferent(distance));
6)  CPUT.addConstraint(Choco.eq(marks[0], 0));
7)  CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
                Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
                      distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
8)  CPUT.addConstraint(Choco.eq(
          Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (l-i))],
          distance[((m*(m-1) / 2) - ((m-l+1)*(m-l) / 2) + (j-l))]),
          Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
               distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```

b)-Où est la faute ?

EXEMPLE : RÈGLES DE GOLOMB

M

```
1) MO.addConstraint(Choco.lt(marks[i], marks[i+1]));
2) MO.addConstraint(Choco.neq(Choco.minus(marks[j],marks[i]),
                    Choco.minus(marks[l],marks[k])));
```

a)- P est il conforme à M ?

c)-Comment corriger auto. la faute ?

P

```
1) CPUT.addConstraint(Choco.lt(marks[i], marks[i+1]));
2) CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
                      Choco.minus(marks[j-1],marks[i-1])));
3) CPUT.addConstraint(Choco.geq(marks[m-1], m*(m-1)/2));
4) CPUT.addConstraint(Choco.leq(marks[1], Choco.minus(marks[m-1], marks[m-2])));
5) CPUT.addConstraint(Choco.allDifferent(distance));
6) CPUT.addConstraint(Choco.eq(marks[0], 0));
7) CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
            Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
                distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
8) CPUT.addConstraint(Choco.eq(
        Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (l-i))],
        distance[((m*(m-1) / 2) - ((m-l+1)*(m-l) / 2) + (j-l))]),
        Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
            distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```

b)-Où est la faute

EXEMPLE : RÈGLES DE GOLOMB

M

1)
```
MO.addConstraint(Choco.lt(marks[i], marks[i+1]));
```
2)
```
MO.addConstraint(Choco.neq(Choco.minus(marks[j],marks[i]),
                    Choco.minus(marks[l],marks[k])));
```

a)- P est il conforme à M ?

c)-Comment corriger auto. la faute ?

Processus de correction

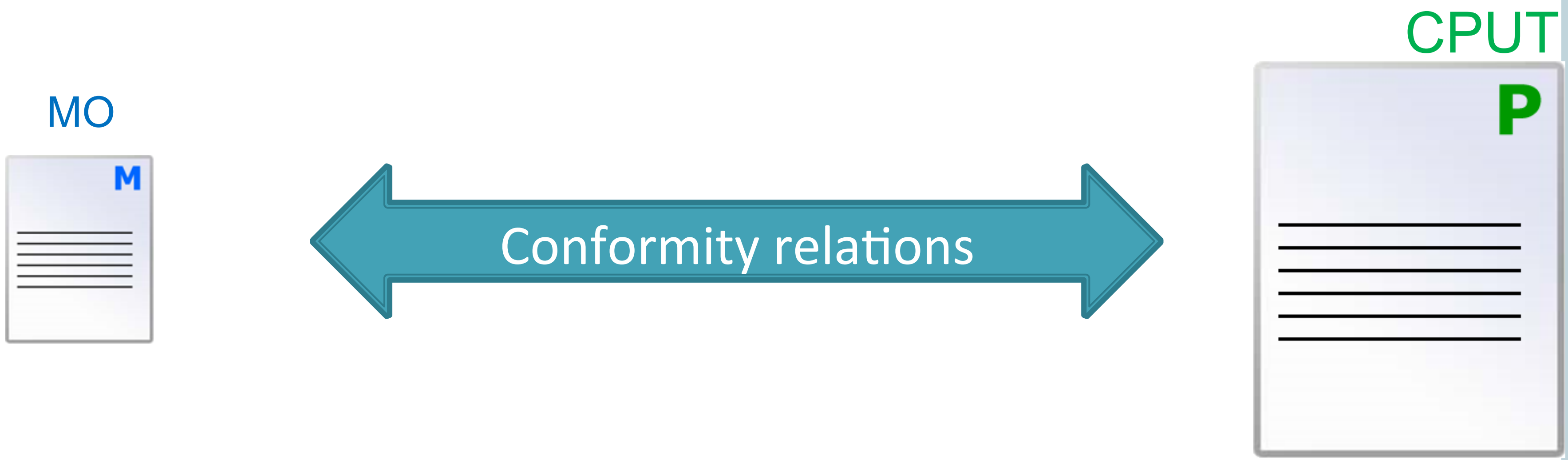
Contraintes Correctrices

P

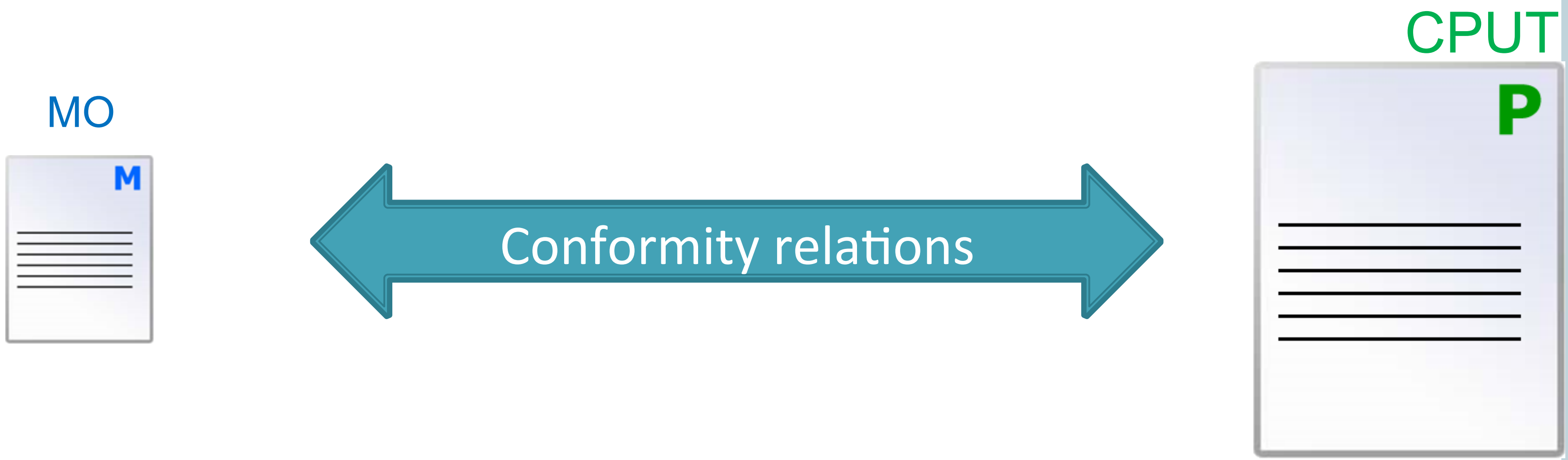
1)
```
CPUT.addConstraint(Choco.lt(marks[i], marks[i+1]));
```
2)
```
CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
                    Choco.minus(marks[j-1],marks[i-1])));
```
3)
```
CPUT.addConstraint(Choco.geq(marks[m-1], m*(m-1)/2));
```
4)
```
CPUT.addConstraint(Choco.leq(marks[1], Choco.minus(marks[m-1], marks[m-2])));
```
5)
```
CPUT.addConstraint(Choco.allDifferent(distance));
```
6)
```
CPUT.addConstraint(Choco.eq(marks[0], 0));
```
7)
```
CPUT.addConstraint(Choco.eq(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (j-i))],
          Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
                 distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```
8)
```
CPUT.addConstraint(Choco.eq(
     Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (l-i))],
     distance[((m*(m-1) / 2) - ((m-l+1)*(m-l) / 2) + (j-l))]),
     Choco.plus(distance[((m*(m-1) / 2) - ((m-i+1)*(m-i) / 2) + (k-i))],
          distance[((m*(m-1) / 2) - ((m-k+1)*(m-k) / 2) + (j-k))])));
```

b)-Où est la faute

Relation de Conformité



Relation de Conformité

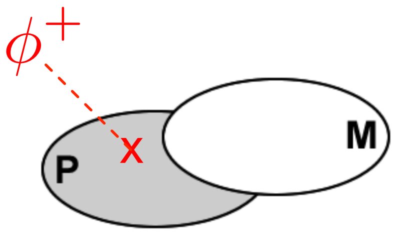


$$\mathrm{CPUT}\ conf\ \mathrm{MO} \Leftrightarrow sol(\mathrm{CPUT}) \neq \emptyset \ \wedge \ sol(\mathrm{CPUT}) \subseteq sol(\mathrm{MO})$$

MODÈLE DE FAUTE

- Faute positive

$$\phi^+ \triangleq sol(\textsc{cput}) \backslash sol(\textsc{mo}) \neq \emptyset$$



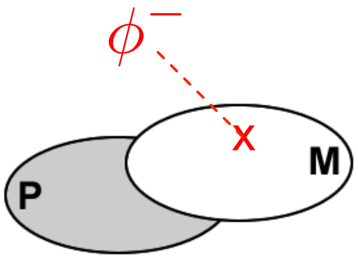
Modèle de Faute

- Faute positive

$$\phi^{+} \triangleq sol(\textsc{cput}) \backslash sol(\textsc{mo}) \neq \emptyset$$

- Faute négative

$$\phi^{-} \triangleq sol(\textsc{mo}) \backslash sol(\textsc{cput}) \neq \emptyset$$



MODÈLE DE FAUTE

- Faute positive

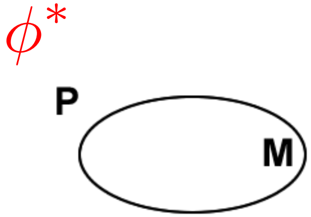
$$\phi^+ \triangleq sol(\text{CPUT}) \setminus sol(\text{MO}) \neq \emptyset$$

- Faute négative

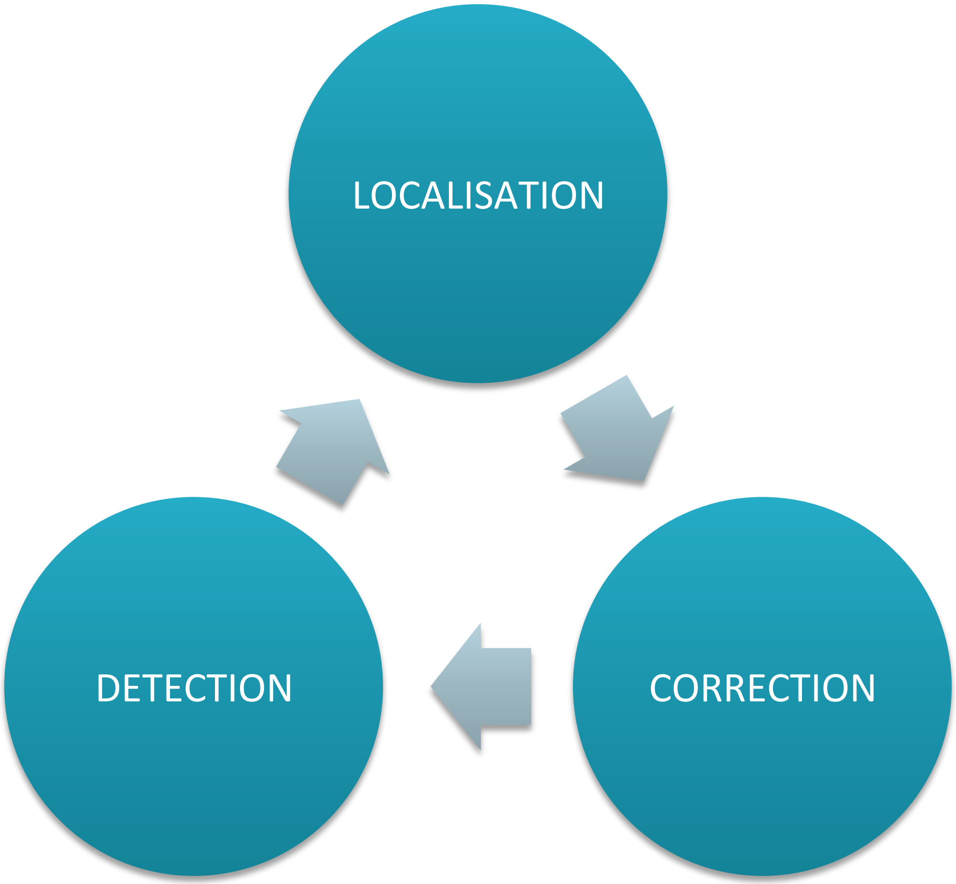
$$\phi^- \triangleq sol(\text{MO}) \setminus sol(\text{CPUT}) \neq \emptyset$$

- Faute zéro

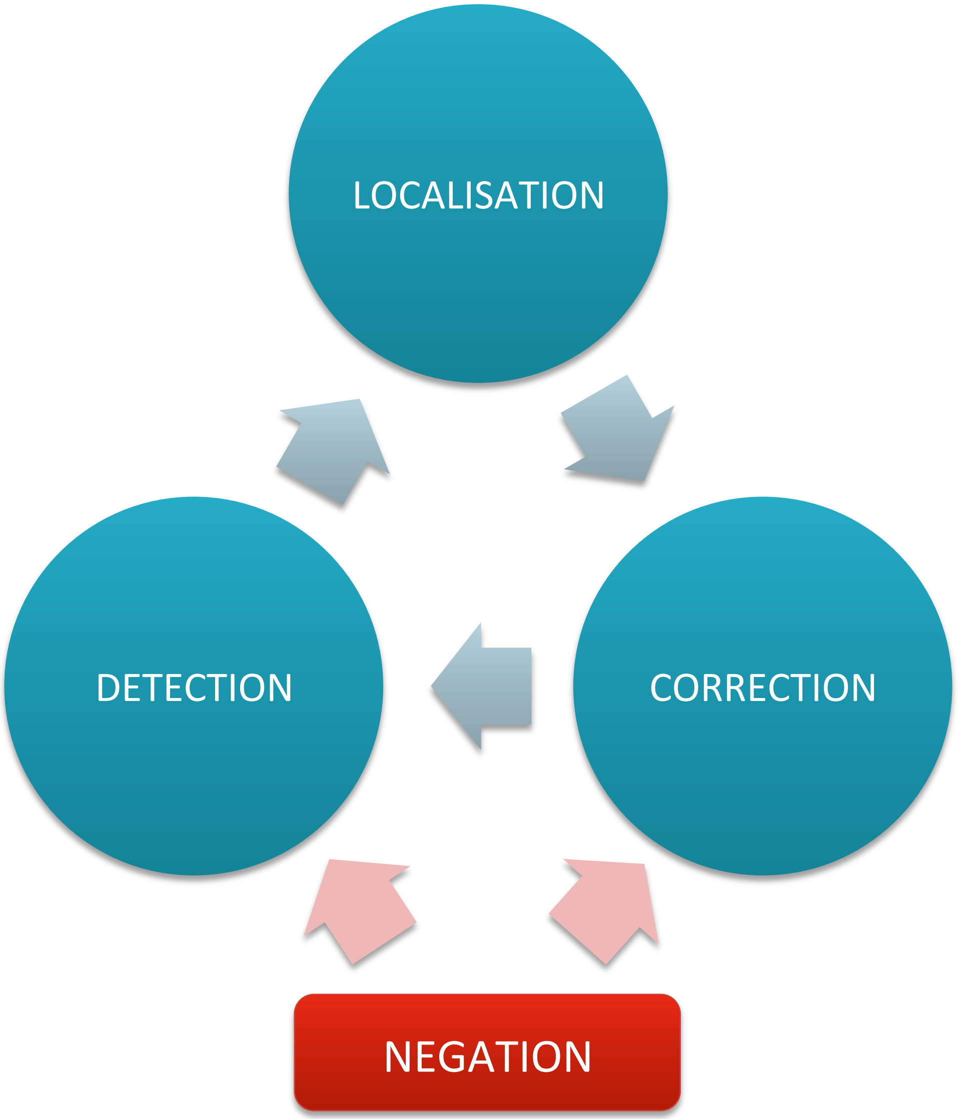
$$\phi^* \triangleq sol(\text{CPUT}) = \emptyset$$



CPTEST4CHOCO : LES PAQUETAGES



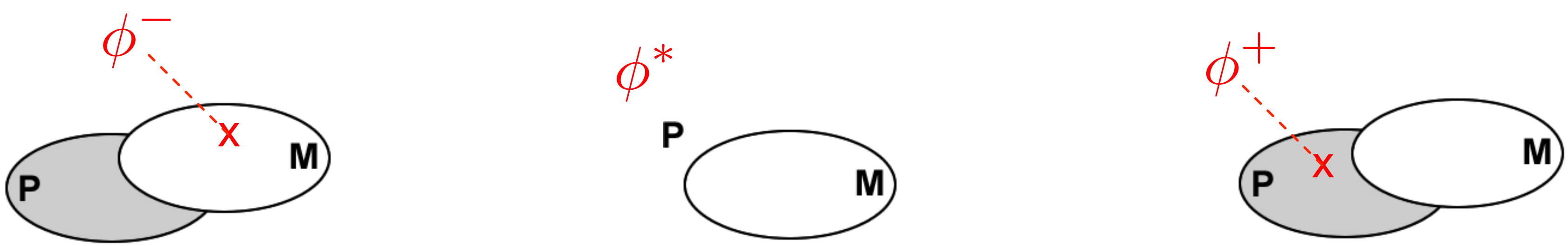
CPTEST4CHOCO : LES PAQUETAGES



CPTEST4CHOCO : DÉTECTION DE FAUTE

Algorithm 1: one_negated(CPUT, MO)

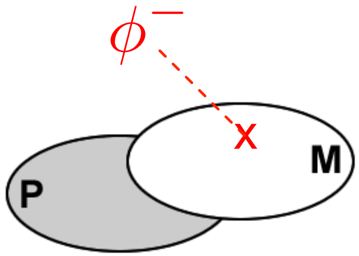
foreach $C_i \in$ MO **do**
  $x \leftarrow solve(\text{CPUT} \wedge \neg C_i)$
  if x **then return** x
return $\emptyset$

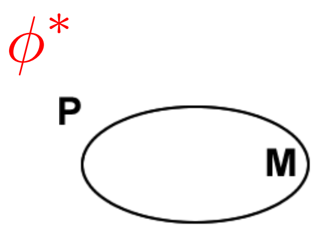


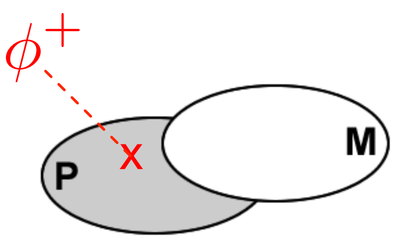
CPTEST4CHOCO : LOCALISATION DE FAUTE

- Contrainte suspecte C_i :

$$\begin{array}{ll} \phi^+, \phi^* & : sol(\textsc{mo}) \cap sol(\textsc{cput} \backslash C_i) \neq \emptyset \\ \phi^- & : sol(\textsc{mo} \wedge \neg C_i) \neq \emptyset \end{array}$$







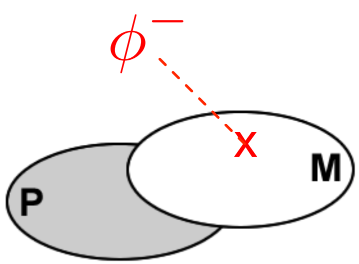
CPTEST4CHOCO : LOCALISATION DE FAUTE

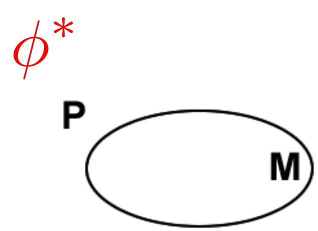
- Contrainte suspecte C_i :

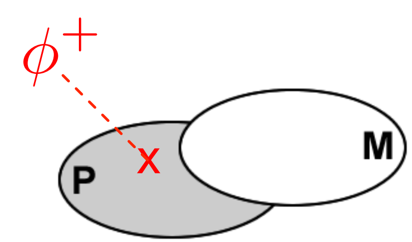
$$\begin{array}{ll} \phi^+, \phi^* & : sol(\textsc{mo}) \cap sol(\textsc{cput} \setminus C_i) \neq \emptyset \\ \phi^- & : sol(\textsc{mo} \wedge \neg C_i) \neq \emptyset \end{array}$$

Algorithm 2: locate(CPUT, MO, nc)

$SuspiciousSet \leftarrow \emptyset$
if $(\phi^+ \vee \phi^*)$ **then**
    foreach $C_i \in$ CPUT **do**
        if $sol(\textsc{mo} \wedge \textsc{cput} \setminus C_i) \neq \emptyset$ **then**
            $SuspiciousSet \leftarrow SuspiciousSet \cup \{C_i\}$
else
    foreach $C_i \in$ CPUT **do**
        if $sol(\textsc{mo} \wedge \neg C_i) \neq \emptyset$ **then**
            $SuspiciousSet \leftarrow \{C_i\}$
            break
if CPUT $\equiv SuspisiousSet$ **then** $SuspisiousSet \leftarrow \emptyset$
return $SuspiciousSet$



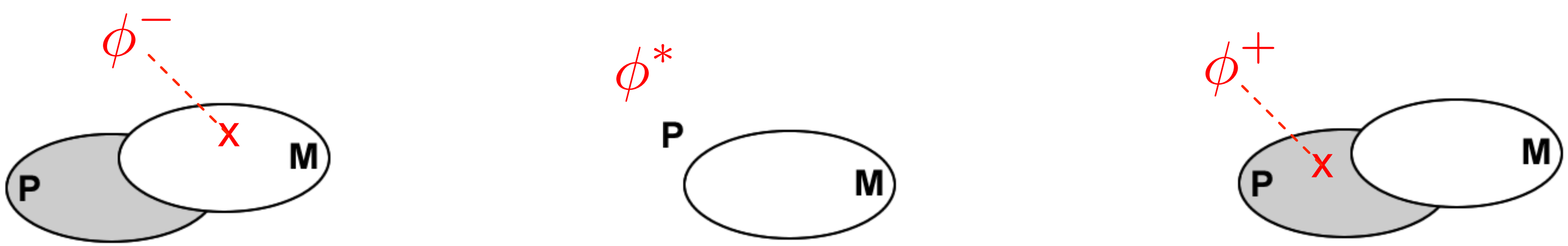




CPTEST4CHOCO : CORRECTION DE FAUTE

- Contrainte correctrice C_i :

$$sol((\textsc{cput} \backslash suspiciousSet) \wedge \neg C_i) \neq \emptyset$$



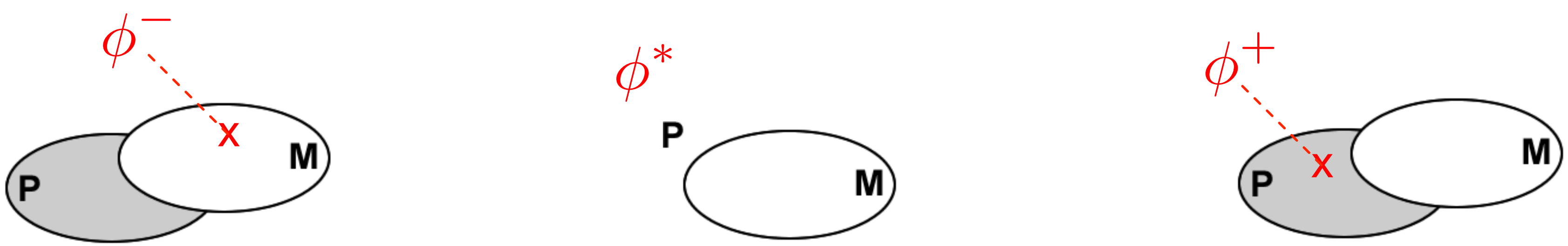
CPTEST4CHOCO : CORRECTION DE FAUTE

- Contrainte correctrice C_i :

$$sol((\text{CPUT} \backslash suspiciousSet) \wedge \neg C_i) \neq \emptyset$$

Algorithm 3: correction(MO, CPUT, $SuspiciousSet$)

```
correctionSet ← ∅
P ← CPUT\SuspiciousSet
foreach C_i ∈ MO do
    if sol(P ∧ ¬C_i) ≠ ∅ then
        correctionSet ← correctionSet ∪ {C_i}
return correctionSet
```



CPTEST4CHOCO : Négation des contraintes

- Négation d'une contrainte élémentaire est une contrainte élémentaire

$$\neg x < y \equiv x \geq y$$

CPTEST4CHOCO : NÉGATION DES CONTRAINTES

- Négation d'une contrainte élémentaire est une contrainte élémentaire

$$\neg x < y \equiv x \geq y$$

- Négation des contraintes globales?

CPTEST4CHOCO : NÉGATION DES CONTRAINTES

- Négation d'une contrainte élémentaire est une contrainte élémentaire

$$\neg x < y \equiv x \geq y$$

- Négation des contraintes globales?
  - ➔ Reformulation $\neg atLeast(n, X, val) \equiv atMost(n-1, X, val)$

$$\neg atMost(n, X, val) \equiv atLeast(n+1, X, val)$$

CPTEST4CHOCO : NÉGATION DES CONTRAINTES

- Négation d'une contrainte élémentaire est une contrainte élémentaire

$$\neg x < y \equiv x \geq y$$

- Négation des contraintes globales?
  - ➔ Reformulation (done!)
  - ➔ Réécriture

$$allDiff(x) \xrightarrow{\text{réécriture}} \forall i,j : \; x_i \neq x_j \xrightarrow{\text{négation}} \exists i,j : x_i = x_j$$

CPTEST4CHOCO : Négation des contraintes

- Négation d'une contrainte élémentaire est une contrainte élémentaire

$$\neg x < y \equiv x \geq y$$

- Négation des contraintes globales?
  - ➔ Reformulation (done!)
  - ➔ Réécriture (done!)
  - ➔ Réification

CPTEST4CHOCO : NÉGATION DES CONTRAINTES

- Négation d'une contrainte élémentaire est une contrainte élémentaire

$$\neg x < y \equiv x \geq y$$

- Négation des contraintes globales?
  - ➔ Reformulation (done!)
  - ➔ Réécriture (done!)
  - ➔ Réification (done!)

CPTEST4CHOCO : NÉGATION DES CONTRAINTES

- Négation d'une contrainte élémentaire est une contrainte élémentaire

$$\neg x < y \equiv x \geq y$$

- Négation des contraintes globales?
  - ➔ Reformulation (done!)
  - ➔ Réécriture (done!)
  - ➔ Réification (done!)
  - ➔ générique (todo!)

CPTEST4CHOCO : RÉSULTATS

	Mutants	Fault injected	Detection		Localization		Correction		
			non-conformity	time	susp. cstr	time	removed cstr	added cstr	time
Golomb rulers (m=8)	Mut1	cc2	ϕ^+ : [0 1 2 3 4 5 6 7]	0.87	∅	2.48	∅	368 EC	42.13
	Mut2	cc3	ϕ^+ :[0 1 2 3 4 5 6 7]	0.11	∅	1.69	∅	431 EC	43.53
	Mut3	cc3	ϕ^- : [0 7 15 17 18 31 37 58]	0.08	∅	1.72	∅	466 EC	46.59
	Mut4	cc5	ϕ^+ : [0 4 8 12 16 20 24 28]	0.16	∅	1.46	∅	340 EC	14.25
	Mut5	cc1	ϕ^* :sol(Mut5)=∅	0.35	cc1	315.85	cc1	763 EC	30.49
	Mut6	cc7	ϕ^* :sol(Mut6)= ∅	0.37	cc7	137.07	cc7	763 EC	34.81
	Mut7	cc9	ϕ^* :sol(Mut7)= ∅	35.10	cc9	13.50	cc9	∅	108.45
n-queens (n=8)	Mut1	cc12	ϕ^* : sol(Mut1)=∅	0.34	cc12	0.87	cc12	∅	0.17
	Mut2	cc11	ϕ^* :sol(Mut2)=∅	0.39	cc11	0.99	cc11	∅	0.44
	Mut3	cc12	ϕ^* :sol(Mut3)=∅	0.20	cc12	0.40	cc12	∅	0.36
	Mut4	cc11	ϕ^* :sol(Mut2)=∅	0.01	cc11	0.18s	cc11	∅	0.30
	Mut5	cc6	ϕ^+ :[8 7 6 5 4 3 2 1]	0.52	cc6	0.23	cc6	28 EC	0.67
	Mut6	cc7	ϕ^- [1 5 8 6 3 7 2 4]	0.24	∅	0.31	∅	28 EC	0.44

CPTEST4CHOCO : RÉSULTATS

détection

	Mutants	Fault injected	Detection: non-conformity	Detection: time	Localization: susp. cstr	Localization: time	Correction: removed cstr	Correction: added cstr	Correction: time
Golomb rulers (m=8)	Mut1	cc2	ϕ^+ : [0 1 2 3 4 5 6 7]	0.87	∅	2.48	∅	368 EC	42.13
	Mut2	cc3	ϕ^+ :[0 [illegible] 7]	0.11	∅	1.69	∅	431 EC	43.53
	Mut3	cc3	ϕ^- : [0 7 15 17 18 31 37 58]	0.08	∅	1.72	∅	466 EC	46.59
	Mut4	cc5	ϕ^+ : [0 4 8 12 16 20 24 28]	0.16	[illegible]	[illegible]6	∅	340 EC	14.25
	Mut5	cc1	ϕ^* :sol(Mut5)=∅	0.35	cc1	315.85	cc1	763 EC	30.49
	Mut6	cc7	ϕ^* :sol(Mut6)= ∅	0.37	cc7	137.07	cc7	763 EC	34.81
	Mut7	cc9	ϕ^* :sol(Mut7)= ∅	35.10	cc9	13.50	cc9	∅	108.45
n-queens (n=8)	Mut1	cc12	ϕ^* : sol(Mut1)=∅	0.34	cc12	0.87	cc12	∅	0.17
	Mut2	cc11	ϕ^* :sol(Mut2)=∅	0.39	cc11	0.99	cc11	∅	0.44
	Mut3	cc12	ϕ^* :sol(Mut3)=∅	0.20	cc12	0.40	cc12	∅	0.36
	Mut4	cc11	ϕ^* :sol(Mut2)=∅	0.01	cc11	0.18s	cc11	∅	0.30
	Mut5	cc6	ϕ^+ :[8 7 6 5 4 3 2 1]	0.52	cc6	0.23	cc6	28 EC	0.67
	Mut6	cc7	ϕ^- [1 5 8 6 3 7 2 4]	0.24	∅	0.31	∅	28 EC	0.44

CPTEST4CHOCO : RÉSULTATS

localisation

correction

	Mutants	Fault injected	Detection		Localization		Correction		
			non-conformity	time	susp. cstr	time	removed cstr	added cstr	time
Golomb rulers (m=8)	Mut1	cc2	ϕ^{+} : [0 1 2 3 4 5 6 7]	0.87	∅	2.48	∅	368 EC	42.13
	Mut2	cc3	ϕ^{+} :[0 1 2 3 4 5 6 7]	[illegible]	∅	1.69	[illegible]	[illegible] EC	43.53
	Mut3	cc3	ϕ^{-} : [0 7 15 17 18 31 37 58]	0.08	∅	1.72	∅	466 EC	46.59
	Mut4	cc5	ϕ^{+} : [0 4 8 12 16 [illegible]	[illegible]	∅	1.46	∅	340 EC	14.25
	Mut5	cc1	ϕ^{*} :sol(Mut5)=∅	0.35	cc1	315.85	cc1	763 EC	30.49
	Mut6	cc7	ϕ^{*} :sol(Mut6)= ∅	0.37	cc7	137.07	cc7	763 EC	34.81
	Mut7	cc9	ϕ^{*} :sol(Mut7)= ∅	35.10	cc9	13.50	cc9	∅	108.45
n-queens (n=8)	Mut1	cc12	ϕ^{*} : sol(Mut1)=∅	0.34	cc12	0.87	cc12	∅	0.17
	Mut2	cc11	ϕ^{*} :sol(Mut2)=∅	0.39	cc11	0.99	cc11	∅	0.44
	Mut3	cc12	ϕ^{*} :sol(Mut3)=∅	0.20	cc12	0.40	cc12	∅	0.36
	Mut4	cc11	ϕ^{*} :sol(Mut2)=∅	0.01	cc11	0.18s	cc11	∅	0.30
	Mut5	cc6	ϕ^{+} :[8 7 6 5 4 3 2 1]	0.52	cc6	0.23	cc6	28 EC	0.67
	Mut6	cc7	ϕ^{-} [1 5 8 6 3 7 2 4]	0.24	∅	0.31	∅	28 EC	0.44

CPTEST4CHOCO : RÉSULTATS

détection

	Mutants	Fault injected	Detection: non-conformity	Detection: time	Localization: susp. cstr	Localization: time	Correction: removed cstr	Correction: added cstr	Correction: time
omb rulers (m=8)	Mut1	cc2	ϕ^{+} : [0 1 2 3 4 5 6 7]	0.87	∅	2.48	∅	368 EC	42.13
	Mut2	cc3	ϕ^{+} :[0 1 2 3 4 5 6 7]	0.11	∅	1.69	∅	431 EC	43.53
	Mut3	cc3	ϕ^{-} : [0 7 15 17 18 31 37 58]	0.08	∅	1.72	∅	466 EC	46.59
	Mut4	cc5	ϕ^{+} : [0 4 8 12 16 20 24 28]	0.16	∅	1.46	∅	340 EC	14.25
	Mut5	cc1	ϕ^{*} :sol(Mut5)=∅	0.35	cc1	315.85	cc1	763 EC	30.49
	Mut6	cc7	ϕ^{*} :sol(Mut6)= ∅	0.37	[illegible]	[illegible]7.07	cc7	763 EC	34.81
	Mut7	cc9	ϕ^{*} :sol(Mut7)= ∅	35.10	cc9	13.50	cc9	∅	108.45
n-queens (n=8)	Mut1	cc12	ϕ^{*} : sol(Mut1)=∅	0.34	cc12	0.87	cc12	∅	0.17
	Mut2	cc11	ϕ^{*} :sol(Mut2)=∅	0.39	cc11	0.99	cc11	∅	0.44
	Mut3	cc12	ϕ^{*} :sol(Mut3)=∅	0.20	cc12	0.40	cc12	∅	0.36
	Mut4	cc11	ϕ^{*} :sol(Mut2)=∅	0.01	cc11	0.18s	cc11	∅	0.30
	Mut5	cc6	ϕ^{+} :[8 7 6 5 4 3 2 1]	0.52	cc6	0.23	cc6	28 EC	0.67
	Mut6	cc7	ϕ^{-} [1 5 8 6 3 7 2 4]	0.24	∅	0.31	∅	28 EC	0.44

CPTEST4CHOCO : RÉSULTATS

	Mutants	Fault injected	Detection		Localization		Correction		
			non-conformity	time	susp. cstr	time	removed cstr	added cstr	time
Golomb rulers (m=8)	Mut1	cc2	ϕ^+ : [0 1 2 3 4 5 6 7]	0.87	∅	2.48	∅	368 EC	42.13
	Mut2	cc3	ϕ^+ :[0 1 2 3 4 5 6 7]	0.11	∅	1.69	∅	431 EC	43.53
	Mut3	cc3	ϕ^- : [0 7 15 17 18 31 37 58]	0.08	∅	1.72	∅	466 EC	46.59
	Mut4	cc5	ϕ^+ : [0 4 8 12 16 20 24 28]	0.16	∅	1.46	∅	340 EC	14.25
	Mut5	cc1	ϕ^* :sol(Mut5)=∅	[illegible]	[illegible]	315.85	cc1	[illegible]	30.49
	Mut6	cc7	ϕ^* :sol(Mut6)= ∅	[illegible]	cc7	137.07	cc7	763 EC	34.81
	Mut7	cc9	ϕ^* :sol(Mut7)= ∅	35.10	cc9	13.50	cc9	∅	108.45
n-queens (n=8)	Mut1	cc12	ϕ^* : sol(Mut1)=∅	0.34	cc12	0.87	cc12	∅	0.17
	Mut2	cc11	ϕ^* :sol(Mut2)=∅	0.39	cc11	0.99	cc11	∅	0.44
	Mut3	cc12	ϕ^* :sol(Mut3)=∅	0.20	cc12	0.40	cc12	∅	0.36
	Mut4	cc11	ϕ^* :sol(Mut2)=∅	0.01	cc11	0.18s	cc11	∅	0.30
	Mut5	cc6	ϕ^+ :[8 7 6 5 4 3 2 1]	0.52	cc6	0.23	cc6	28 EC	0.67
	Mut6	cc7	ϕ^- [1 5 8 6 3 7 2 4]	0.24	∅	0.31	∅	28 EC	0.44

localisation

correction

CPTEST4CHOCO : RÉSULTATS

	Mutants	Fault injected	Detection		Localization		Correction		
			non-conformity	time	susp. cstr	time	removed cstr	added cstr	time
Golomb rulers (m=8)	Mut1	cc2	ϕ^+ : [0 1 2 3 4 5 6 7]	0.87	∅	2.48	∅	368 EC	42.13
	Mut2	cc3	ϕ^+ :[0 1 2 3 4 5 6 7]	0.11	∅	1.69	∅	431 EC	43.53
	Mut3	cc3	ϕ^- : [0 7 15 17 18 31 37 58]	0.08	∅	1.72	∅	466 EC	46.59
	Mut4	cc5	ϕ^+ : [0 4 8 12 16 20 24 28]	0.16	∅	1.46	∅	340 EC	14.25
	Mut5	cc1	ϕ^* :sol(Mut5)=∅	0.35	cc1	315.85	cc1	763 EC	30.49
	Mut6	cc7	ϕ^* :sol(Mut6)= ∅	0.37	cc7	137.07	cc7	763 EC	34.81
	Mut7	cc9	ϕ^* :sol(Mut7)= ∅	35.10	cc9	13.50	cc9	∅	108.45
n-queens (n=8)	Mut1	cc12	ϕ^* : sol(Mut1)=∅	0.34	cc12	0.87	cc12	∅	0.17
	Mut2	cc11	ϕ^* :sol(Mut2)=∅	0.39	cc11	0.99	cc11	∅	0.44
	Mut3	cc12	ϕ^* :sol(Mut3)=∅	0.20	cc12	0.40	cc12	∅	0.36
	Mut4	cc11	ϕ^* :sol(Mut2)=∅	0.01	cc11	0.18s	cc11	∅	0.30
	Mut5	cc6	ϕ^+ :[8 7 6 5 4 3 2 1]	0.52	cc6	0.23	cc6	28 EC	0.67
	Mut6	cc7	ϕ^- [1 5 8 6 3 7 2 4]	0.24	∅	0.31	∅	28 EC	0.44

Conclusions et perspectives

- Nos approches de test et de mise-au-points sont viables et peuvent être généralisées à d'autres langages PPC

Conclusions et perspectives

- Nos approches de test et de mise-au-points sont viables et peuvent être généralisées à d'autres langages PPC
- CPTEST des contraintes globales

CONCLUSIONS ET PERSPECTIVES

- Nos approches de test et de mise-au-points sont viables et peuvent être généralisées à d'autres langages PPC
- CPTEST des contraintes globales
- Test et correction : approches par réfutation ➔ Négation générique

CONCLUSIONS ET PERSPECTIVES

- Nos approches de test et de mise-au-points sont viables et peuvent être généralisées à d'autres langages PPC
- CPTEST des contraintes globales
- Test et correction : approches par réfutation → Négation générique
- Oracle de test → Acquisition de contraintes

CONCLUSIONS ET PERSPECTIVES

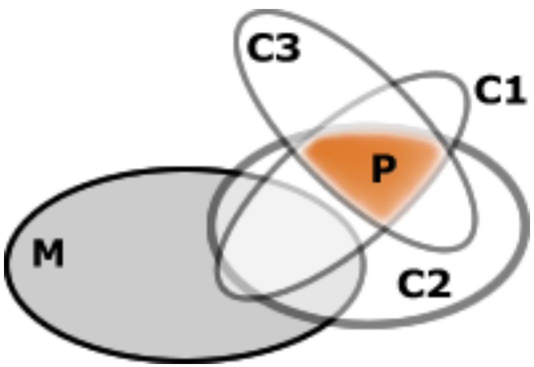
- Nos approches de test et de mise-au-points sont viables et peuvent être généralisées à d'autres langages PPC
- CPTEST des contraintes globales
- Test et correction : approches par réfutation ➔ Négation générique
- Oracle de test ➔ Acquisition de contraintes
- Critères de test (couverture, réveil, filtrage,…)

Conclusions et perspectives

- Nos approches de test et de mise-au-points sont viables et peuvent être généralisées à d'autres langages PPC
- CPTEST des contraintes globales
- Test et correction : approches par réfutation ➔ Négation générique
- Oracle de test ➔ Acquisition de contraintes
- Critères de test (couverture, réveil, filtrage,...)

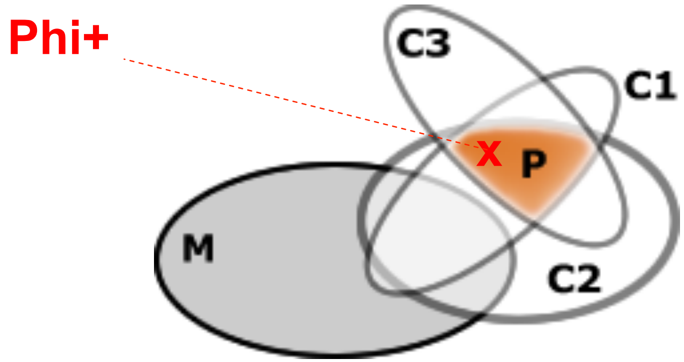
Fault localization (simple fault)

$$P \equiv C_1 \wedge C_2 \wedge C_3$$



Fault localization (simple fault)

$$P \equiv C_1 \wedge C_2 \wedge C_3$$

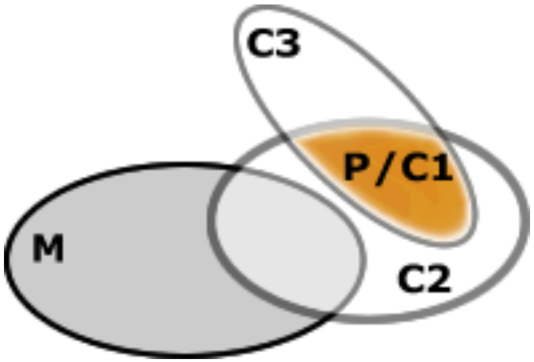


Fault localization (simple fault)

$$P \equiv C_1 \wedge C_2 \wedge C_3$$

Proposition:

C_i is suspicious in P w.r.t. $M \equiv M \wedge \{\overbrace{C_1, C_2 ..., C_{i-1}, C_{i+1} ... C_n}^{P \backslash C_i}\}$ is satisfiable.



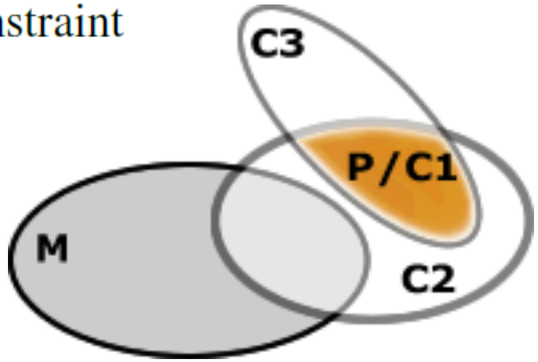
Fault localization (simple fault)

$$P \equiv C_1 \wedge C_2 \wedge C_3$$

Proposition:

C_i is suspicious in P w.r.t. $M \equiv M \wedge \{\overbrace{C_1, C_2..., C_{i-1}, C_{i+1}...C_n}^{P\backslash C_i}\}$ is satisfiable.

- $sol(M \wedge P/\{C_1\}) = \emptyset$ C_1 is not a suspicious constraint



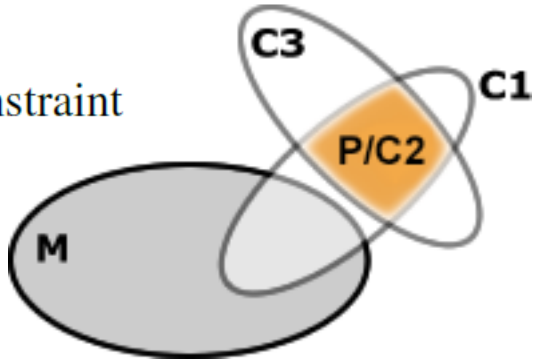
Fault localization (simple fault)

$$P \equiv C_1 \wedge C_2 \wedge C_3$$

Proposition:

C_i is suspicious in P w.r.t. $M \equiv M \wedge \{\overbrace{C_1, C_2 ..., C_{i-1}, C_{i+1} ... C_n}^{P \backslash C_i}\}$ is satisfiable.

- $sol(M \wedge P/\{C_1\}) = \emptyset$
- $sol(M \wedge P/\{C_2\}) = \emptyset$ C_2 is not a suspicious constraint



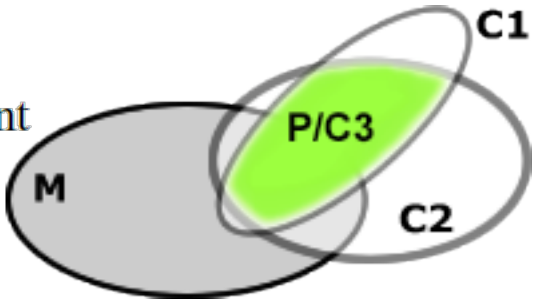
Fault localization (simple fault)

$$P \equiv C_1 \wedge C_2 \wedge C_3$$

Proposition:

C_i is suspicious in P w.r.t. $M \equiv M \wedge \{\overbrace{C_1, C_2 ..., C_{i-1}, C_{i+1} ... C_n}^{P \backslash C_i}\}$ is satisfiable.

- $sol(M \wedge P/\{C_1\}) = \emptyset$
- $sol(M \wedge P/\{C_2\}) = \emptyset$
- $sol(M \wedge P/\{C_3\}) \neq \emptyset$ C_3 is a suspicious constraint



C_3 is the faulty constraint!!