⏩ Fault Localization in Constraint Programs

Nadjib LAZAAR*, Arnaud GOTLIEB*, Yahia LEBBAH**

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

ICTAI'10, Arras, France
27 October

UNIVERSITÉ DE RENNES 1

INRIA

Motivations

- Numerous CP modeling languages and platforms have been developed

(OPL, SICStus Prolog, ZINC, GECODE, CHOCO…)

for solving combinatorial problems that arise in optimization, planning, scheduling…

Motivations

- Numerous CP modeling languages and platforms have been developed

  (OPL, SICStus Prolog, ZINC, GECODE, CHOCO…)

for solving combinatorial problems that arise in optimization, planning, scheduling…

- CP programs begin to be used in business-critical systems (e.g., combinatorial auctions)

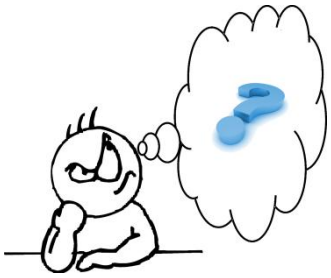
Motivations

- Numerous CP modeling languages and platforms have been developed

  (OPL, SICStus Prolog, ZINC, GECODE, CHOCO…)

for solving combinatorial problems that arise in optimization, planning, scheduling…

- CP programs begin to be used in business-critical systems (e.g., combinatorial auctions)
- Refinement in CP



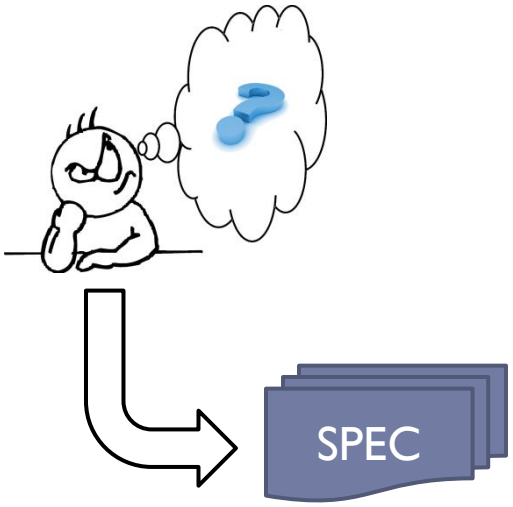
Motivations

- Numerous CP modeling languages and platforms have been developed

  (OPL, SICStus Prolog, ZINC, GECODE, CHOCO…)

for solving combinatorial problems that arise in optimization, planning, scheduling…

- CP programs begin to be used in business-critical systems (e.g., combinatorial auctions)
- Refinement in CP



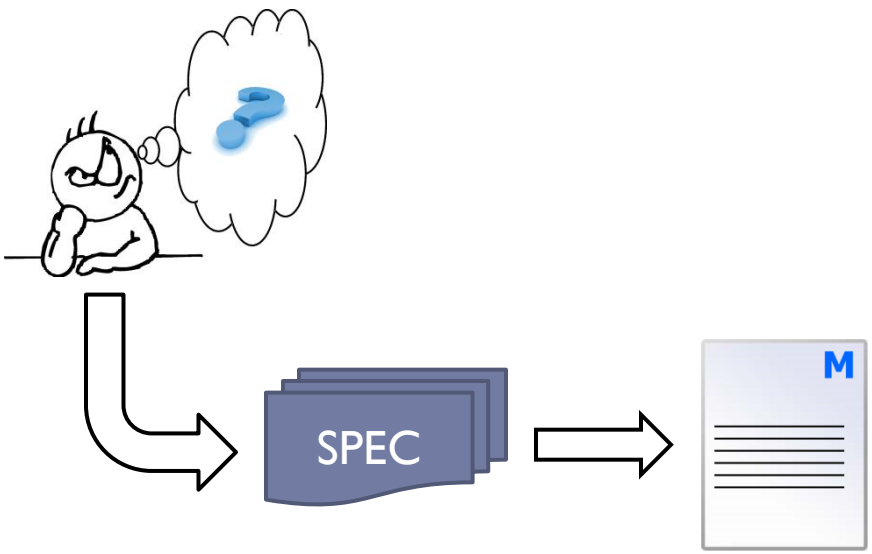
Motivations

- Numerous CP modeling languages and platforms have been developed

  (OPL, SICStus Prolog, ZINC, GECODE, CHOCO…)

for solving combinatorial problems that arise in optimization, planning, scheduling…

- CP programs begin to be used in business-critical systems (e.g., combinatorial auctions)
- Refinement in CP



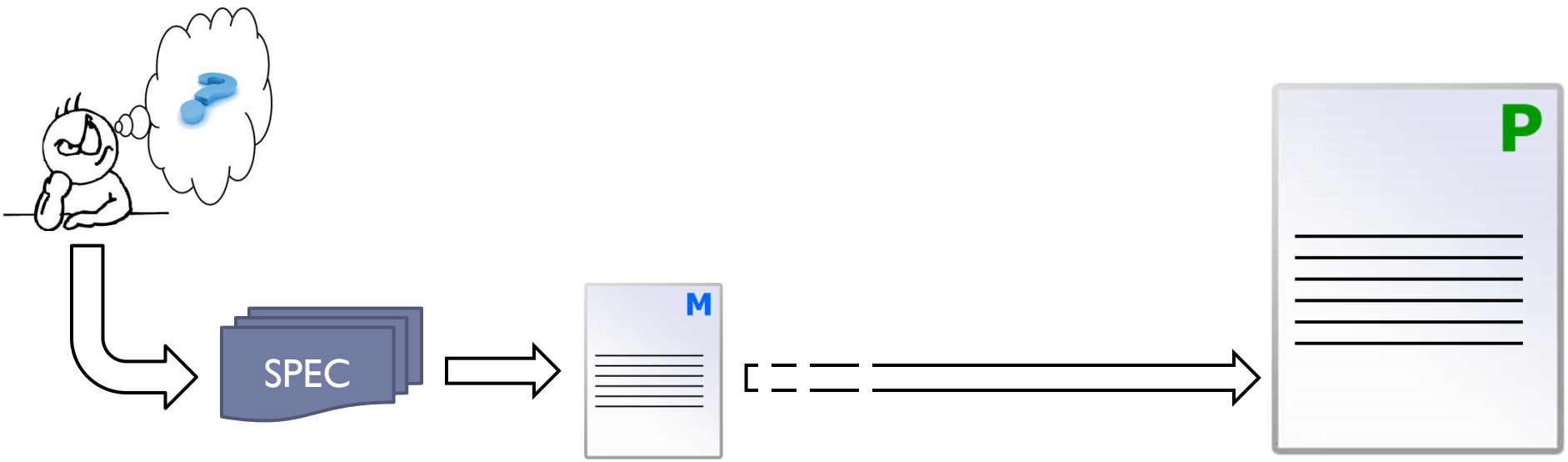
Motivations

- Numerous CP modeling languages and platforms have been developed

  (OPL, SICStus Prolog, ZINC, GECODE, CHOCO…)

for solving combinatorial problems that arise in optimization, planning, scheduling…

- CP programs begin to be used in business-critical systems (e.g., combinatorial auctions)
- Refinement in CP



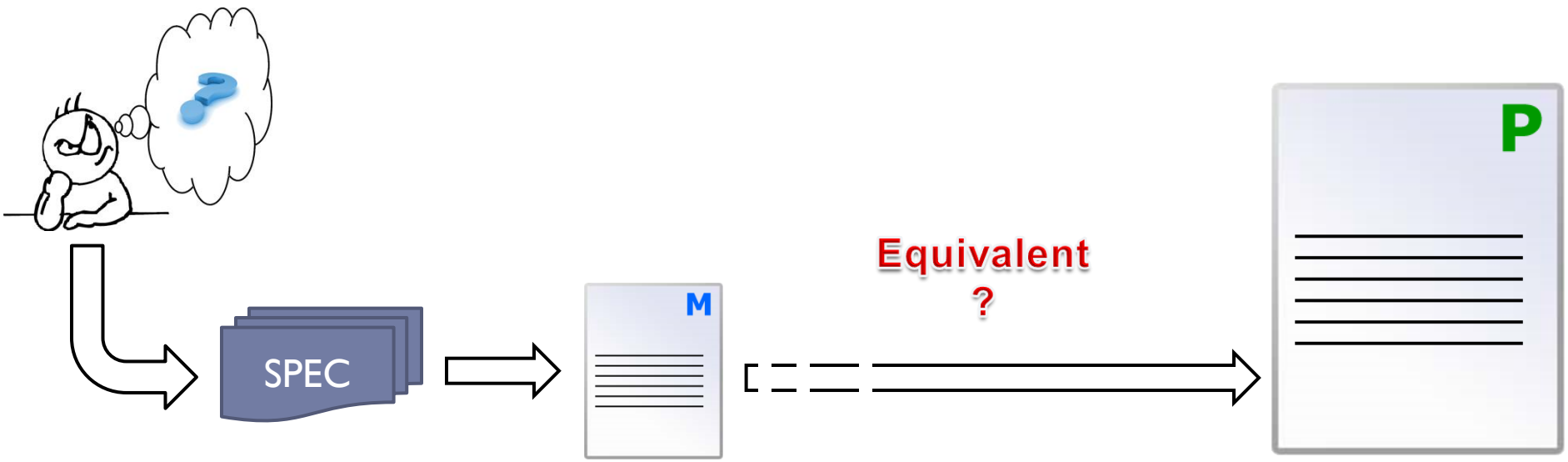
Motivations

- Numerous CP modeling languages and platforms have been developed

  (OPL, SICStus Prolog, ZINC, GECODE, CHOCO…)

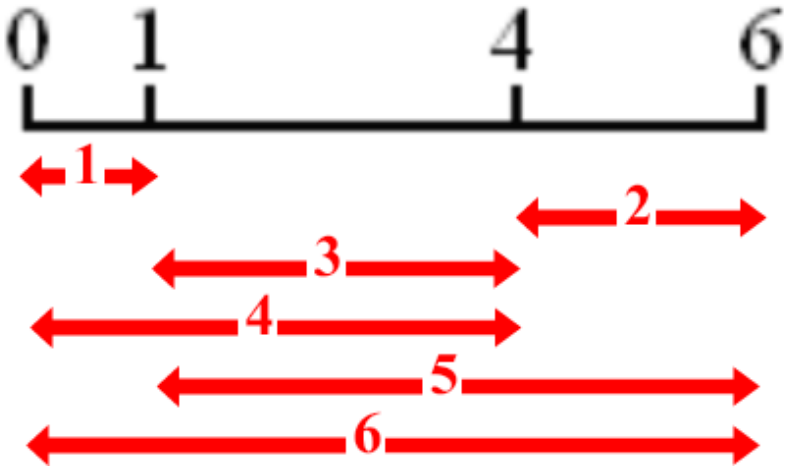
for solving combinatorial problems that arise in optimization, planning, scheduling…

- CP programs begin to be used in business-critical systems (e.g., combinatorial auctions)
- Refinement in CP



Golomb Rulers

Golomb Rulers



Golomb Rulers in **O**ptimization **P**rogramming **L**anguage

```
using CP;

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];
}
```

Golomb Rulers in **O**ptimization **P**rogramming **L**anguage

M

```
using CP;

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

```
using CP;

int m=...;
tuple indexerTuple {int i;
                    int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize x[m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
            (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
                        d[ind1] == d[ind2]+d[ind3];
(7) forall(ind1,ind2,ind3,ind4 in indexes2 :
            (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]);
(8) forall(ind in indexes2, k in 1..m div 2)
            (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);

             }
```

Golomb Rulers in **O**ptimization **P**rogramming **L**anguage

M

```
using CP;

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];
}
```

1)- Does P conform to M ?

P

```
using CP;

int m=...;
tuple indexerTuple {int i;
                    int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize x[m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
            (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
                        d[ind1] == d[ind2]+d[ind3];
(7) forall(ind1,ind2,ind3,ind4 in indexes2 :
            (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]);
(8) forall(ind in indexes2, k in 1..m div 2)
            (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);

             }
```

Golomb Rulers in **O**ptimization **P**rogramming **L**anguage

M

```
using CP;

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

```
using CP;

int m=...;
tuple indexerTuple {int i;
                    int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize x[m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
            (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
                        d[ind1] == d[ind2]+d[ind3];
(7) forall(ind1,ind2,ind3,ind4 in indexes2 :
            (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]);
(8) forall(ind in indexes2, k in 1..m div 2)
            (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);

             }
```

1)- Does P conform to M ?

m=8

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

Golomb Rulers in **O**ptimization **P**rogramming **L**anguage

M

```
using CP;

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

```
using CP;

int m=...;
tuple indexerTuple {int i;
                    int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize x[m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
            (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
                        d[ind1] == d[ind2]+d[ind3];
(7) forall(ind1,ind2,ind3,ind4 in indexes2 :
            (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]);
(8) forall(ind in indexes2, k in 1..m div 2)
            (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);

             }
```

1)- Does P conform to M ?

m=8

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

1 1

Fault detected in P !

Golomb Rulers in **O**ptimization **P**rogramming **L**anguage

M

```
using CP;

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];
}
```

1)- Does P conform to M

m=8

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

1 1

Fault detected in P !

P

```
using CP;

int m=...;
tuple indexerTuple {int i;
                    int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize x[m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
            (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
                        d[ind1] == d[ind2]+d[ind3];
(7) forall(ind1,ind2,ind3,ind4 in indexes2 :
            (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]);
(8) forall(ind in indexes2, k in 1..m div 2)
            (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);

             }
```

2)-Where is the fault

Golomb Rulers in **O**ptimization **P**rogramming **L**anguage

M

```
using CP;

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];
}
```

1)- Does P conform to M

m=8

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

1 1

Fault detected in P !

P

```
using CP;

int m=...;
tuple indexerTuple {int i;
                    int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize x[m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)    x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
            (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
                        d[ind1] == d[ind2]+d[ind3];
(7) forall(ind1,ind2,ind3,ind4 in indexes2 :
            (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]);
(8) forall(ind in indexes2, k in 1..m div 2)
            (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);

            }
```

2)-Where is the fault

Faulty constraint!!

Contributions

- A first framework for fault localization in constraint programs
  - Context
  - Fault localization process

Contributions

- A first framework for fault localization in constraint programs
  - Context
  - Fault localization process

- A method and a tool, called CPTEST:
  - to detect non-conformities (e.g., X= [0 1 3 6 10 26 27 28])
  - to localize faulty constraints
  - short demonstration of CPTEST

Contributions

- A first framework for fault localization in constraint programs
  - Context
  - Fault localization process

- A method and a tool, called CPTEST:
  - to detect non-conformities (e.g., X= [0 1 3 6 10 26 27 28])
  - to localize faulty constraints
  - short demonstration of CPTEST

- An experimental validation on classical benchmarks (Golomb rulers, car sequencing, n-queens, social golfer)

Context

- M implements the problem specification (testing oracle)
- The underlying constraint solver is correct

Context

$M \equiv C_1 \wedge C_2 ... \wedge C_n$

$sol(M): \ set\ of\ solution\ of\ M$

Constraint solving problem

$P \equiv C'_1 \wedge C'_2 ... \wedge C'_m$

$sol(P): \ set\ of\ solution\ of\ P$

- M implements the problem specification (testing oracle)
- The underlying constraint solver is correct

Context

$M \equiv C_1 \wedge C_2 ... \wedge C_n$

$sol(M): \ set\ of\ solution\ of\ M$

Constraint solving problem

$P \equiv C'_1 \wedge C'_2 ... \wedge C'_m$

$sol(P): \ set\ of\ solution\ of\ P$

$M \equiv C, \quad minimize(f(x))$

$bounds_{lw,up,f}(M) = \{x : x \in sol(C) \wedge f(x) \in [lw, up]\}$

Optimization problem

$P \equiv C', \quad minimize(f'(x))$

$bounds_{lw,up,f'}(P) = \{x : x \in sol(C') \wedge f'(x) \in [lw, up]\}$

- M implements the problem specification (testing oracle)
- The underlying constraint solver is correct

Context*

- Conformity relation

one solution:

$$P\ conf\ M \Leftrightarrow sol(P) \neq \emptyset \wedge sol(P) \subseteq sol(M)$$

* N. Lazaar, A. Gotlieb, and Y. Lebbah. On testing constraint programs. In Proc. of Principles of Constraint Programming, CP'2010, Sept. 2010.

Context*

- Conformity relation

one solution:

$$P\ conf\ M \Leftrightarrow sol(P) \neq \emptyset \wedge sol(P) \subseteq sol(M)$$

all solutions:

$$P\ conf\ M \Leftrightarrow sol(P) = sol(M)$$

* N. Lazaar, A. Gotlieb, and Y. Lebbah. On testing constraint programs. In Proc. of Principles of Constraint Programming, CP'2010, Sept. 2010.

Context*

- Conformity relation

one solution:

$$P\ conf\ M \Leftrightarrow sol(P) \neq \emptyset \wedge sol(P) \subseteq sol(M)$$

all solutions:

$$P\ conf\ M \Leftrightarrow sol(P) = sol(M)$$

best solution:

$$P\ conf\ M \Leftrightarrow bounds_{lw,up,f'}(P) \neq \emptyset \wedge bounds_{lw,up,f'}(P) \subseteq bounds_{lw,up,f}(M)$$

* N. Lazaar, A. Gotlieb, and Y. Lebbah. On testing constraint programs. In Proc. of Principles of Constraint Programming, CP'2010, Sept. 2010.

Fault localization (suspicious constraints)

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

Fault localization (suspicious constraints)

$$C_i \text{ is suspicious in } P \text{ w.r.t. } M \equiv M \wedge \underbrace{\{C_1, C_2 ..., C_{i-1}, C_{i+1} ... C_n\}}_{P \setminus \{C_i\}} \text{ is satisfiable.}$$

Objective: $SuspiciousSet(P) \equiv \{C \in P : C \text{ is suspicious in } P \text{ w.r.t. } M\}$

Fault localization (suspicious constraints)

$$C_i \text{ is suspicious in } P \text{ w.r.t. } M \equiv M \wedge \underbrace{\{C_1, C_2 ..., C_{i-1}, C_{i+1} ... C_n\}}_{P \setminus \{C_i\}} \text{ is satisfiable.}$$

Objective: $SuspiciousSet(P) \equiv \{C \in P : C \text{ is suspicious in } P \text{ w.r.t. } M\}$

- $SuspiciousSet(P) = \{C_i\} \Rightarrow$ the singleton C_i is the faulty constraint

Fault localization (suspicious constraints)

$$C_i \text{ is suspicious in } P \text{ w.r.t. } M \equiv M \wedge \underbrace{\{C_1, C_2 ..., C_{i-1}, C_{i+1} ... C_n\}}_{P \setminus \{C_i\}} \text{ is satisfiable.}$$

Objective: $SuspiciousSet(P) \equiv \{C \in P : C \text{ is suspicious in } P \text{ w.r.t. } M\}$

- $SuspiciousSet(P) = \{C_i\} \Rightarrow$ the singleton C_i is the faulty constraint
- $SuspiciousSet(P) = \{C_j, ...C_k\} \Rightarrow$ $C_j ... C_k$ are suspicious constraints

Fault localization (suspicious constraints)

$$C_i \text{ is suspicious in } P \text{ w.r.t. } M \equiv M \wedge \underbrace{\{C_1, C_2 ..., C_{i-1}, C_{i+1} ... C_n\}}_{P \setminus \{C_i\}} \text{ is satisfiable.}$$

Objective: $SuspiciousSet(P) \equiv \{C \in P : C \text{ is suspicious in } P \text{ w.r.t. } M\}$

- $SuspiciousSet(P) = \{C_i\} \Rightarrow$ the singleton C_i is the faulty constraint
- $SuspiciousSet(P) = \{C_j, ...C_k\} \Rightarrow$ $C_j ... C_k$ are suspicious constraints
- $SuspiciousSet(P) = P \Rightarrow$ calculate $addSet$ form M

Fault localization (suspicious constraints)

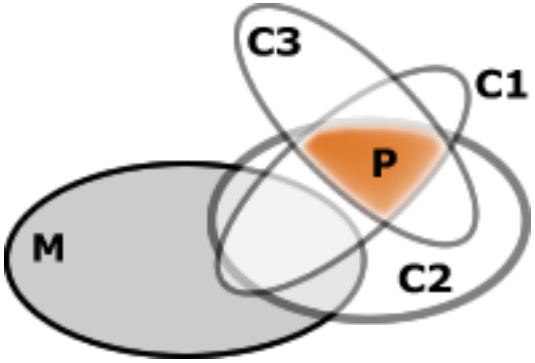
$$C_i \text{ is suspicious in } P \text{ w.r.t. } M \equiv M \wedge \underbrace{\{C_1, C_2 ..., C_{i-1}, C_{i+1} ... C_n\}}_{P \setminus \{C_i\}} \text{ is satisfiable.}$$

Objective: $SuspiciousSet(P) \equiv \{C \in P : C \text{ is suspicious in } P \text{ w.r.t. } M\}$

- $SuspiciousSet(P) = \{C_i\} \Rightarrow$ the singleton C_i is the faulty constraint
- $SuspiciousSet(P) = \{C_j, ...C_k\} \Rightarrow$ $C_j ... C_k$ are suspicious constraints
- $SuspiciousSet(P) = P \Rightarrow$ calculate $addSet$ form M

Hypothesis: P contains one faulty constraint

Fault localization (intuitions)



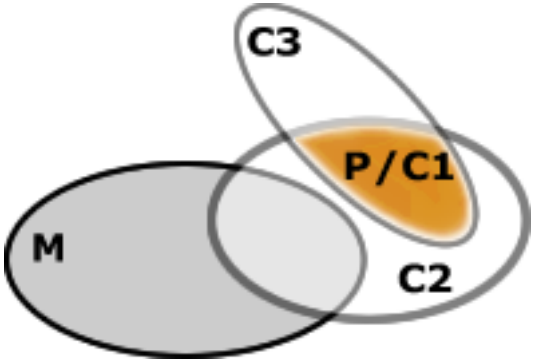
$$SuspiciousSet(P) = \emptyset$$

$$addSet = \emptyset$$

M: set of solution of the Model-Oracle
C1: set of solution of constraint C1
C2: set of solution of constraint C2
C3: set of solution of constraint C3

Fault localization (intuitions)

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



$SuspiciousSet(P) = \emptyset$

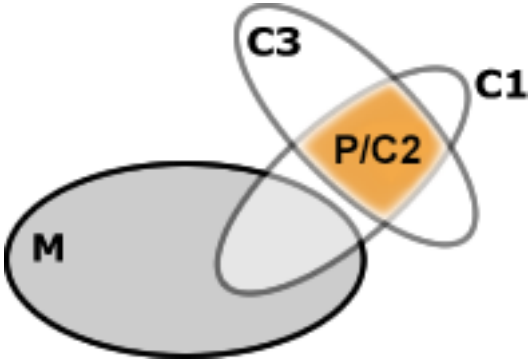
$addSet = \emptyset$

M: set of solution of the Model-Oracle
C1: set of solution of constraint C1
C2: set of solution of constraint C2
C3: set of solution of constraint C3

Fault localization (intuitions)

$$sol(M \wedge P/\{C_1\}) = \emptyset$$

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



$$SuspiciousSet(P) = \emptyset$$

$$addSet = \emptyset$$

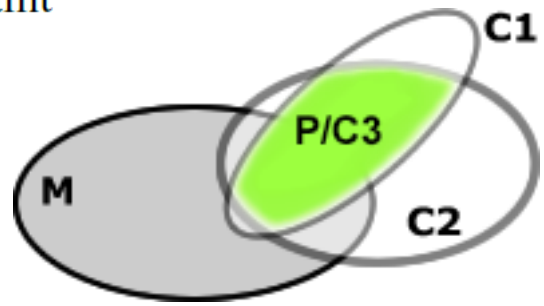
M: set of solution of the Model-Oracle
C1: set of solution of constraint C1
C2: set of solution of constraint C2
C3: set of solution of constraint C3

Fault localization (intuitions)

$$sol(M \wedge P/\{C_1\}) = \emptyset$$

$$sol(M \wedge P/\{C_2\}) = \emptyset$$

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



$$SuspiciousSet(P) = \{C_3\}$$

$$addSet = \emptyset$$

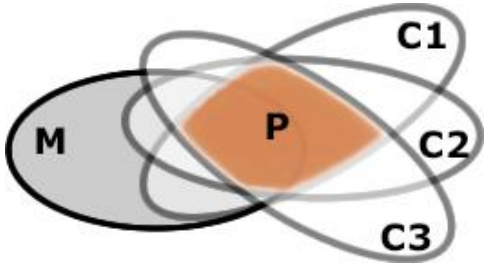
C_3 is the faulty constraint!!

M: set of solution of the Model-Oracle
C1: set of solution of constraint C1
C2: set of solution of constraint C2
C3: set of solution of constraint C3

Fault localization (intuitions)

$$SuspiciousSet(P) = \emptyset$$

$$addSet = \emptyset$$



M: set of solution of the Model-Oracle
C1: set of solution of constraint C1
C2: set of solution of constraint C2
C3: set of solution of constraint C3

Fault localization (intuitions)

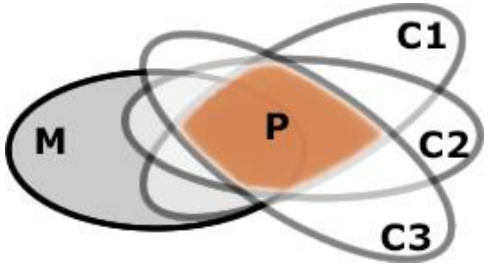
$sol(M \wedge P/\{C_1\}) \neq \emptyset$ C_1 is a suspicious constraint

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

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

$SuspiciousSet(P) = \{C_1, C_2, C_3\}$

$addSet = \emptyset$



M: set of solution of the Model-Oracle
C1: set of solution of constraint C1
C2: set of solution of constraint C2
C3: set of solution of constraint C3

Fault localization (intuitions)

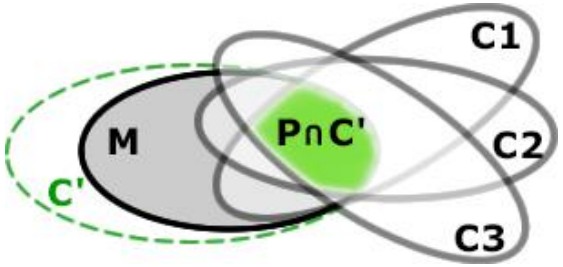
$sol(M \wedge P/\{C_1\}) \neq \emptyset$ C_1 is a suspicious constraint

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

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

$SuspiciousSet(P) = \{C_1, C_2, C_3\}$

$addSet = \{C'\}$



M: set of solution of the Model-Oracle
C1: set of solution of constraint C1
C2: set of solution of constraint C2
C3: set of solution of constraint C3

Fault localization (algorithm)

Algorithm: locate(M, P, nc)

$SuspiciousSet \leftarrow \emptyset$
$addSet \leftarrow \emptyset$
foreach $C_i \in P$ **do**
  if $sol(M \wedge P \backslash \{C_i\}) \neq \emptyset$ **then**
    $SuspisiousSet \leftarrow SuspisiousSet \cup \{C_i\}$
if $(SuspisiousSet = P) \wedge (nc \neq \emptyset)$ **then**
  $addSet \leftarrow$ **checking**(M, nc)
return $(rev(SuspisiousSet), addSet)$

Fault localization (algorithm)

Algorithm: locate(M, P, nc)

$SuspiciousSet \leftarrow \emptyset$
$addSet \leftarrow \emptyset$
foreach $C_i \in P$ **do**
  if $sol(M \wedge P \backslash \{C_i\}) \neq \emptyset$ **then**
    $SuspisiousSet \leftarrow SuspisiousSet \cup \{C_i\}$
if $(SuspisiousSet = P) \wedge (nc \neq \emptyset)$ **then**
  $addSet \leftarrow$ **checking**(M, nc)
return $(rev(SuspisiousSet), addSet)$

checking(B, nc):

if $B = \emptyset$ **then**
  return $\emptyset$
else
  foreach $C_i \in B$ **do**
    if $nc \models C_i$ **then**
      return ***checking***$(M \backslash \{C_i\}, nc)$
    else
      return $\{C_i\} \cup$ **checking**$(M \backslash \{C_i\}, nc)$

CPTEST tool

DEMO

Validation (Golomb Rulers)

- Manual fault injection

P

```
using CP;

int m=...;
tuple indexerTuple {int i;
                    int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize x[m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i] - x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) allDifferent(all(ind in indexes1 ) d[ind]);
(6) x[2] <= x[m] - x[m-1];
(7) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
            (ind1.j==ind3.j)&&( ind1.i<ind2.j<ind1.j))
                        d[ind1] == d[ind2] + d[ind3];
(8) forall(ind1,ind2,ind3,ind4 in indexes2 :
            (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(ind in indexes2, k in 1..m div 2)
        (x[ind.i+1] == x[ind.i]+k) => (x[ind.j+1] != x[ind.j]+k);

             }
```

Validation (Golomb Rulers)

- Manual fault injection

```
using CP;

int m=...;
tuple indexerTuple {int i;
                    int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize x[m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i] - x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) allDifferent(all(ind in indexes1 ) d[ind]);
(6) x[2] <= x[m] - x[m-1];
(7) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
            (ind1.j==ind3.j)&&( ind1.i<ind2.j<ind1.j))
                        d[ind1] == d[ind2] + d[ind3];
(8) forall(ind1,ind2,ind3,ind4 in indexes2 :
            (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(ind in indexes2, k in 1..m div 2)
        (x[ind.i+1] == x[ind.i]+k) => (x[ind.j+1] != x[ind.j]+k);

            }
```

```
forall(i in m..3*m)
    count(all(j in indexes1)d[j],i)==1;
```

Validation (Golomb Rulers)

	Golomb rulers	# consts	fault injected	non-conformity detected
Test	CPUT1	2	p1_ct2	[0 1 3 6]
	CPUT2	4	p2_ct3	[0 1 6 11]
	CPUT3	5	p3_ct3	[0 5 6 11]
	CPUT4	9	p4_ct5	$sol(P4) = \emptyset$
	CPUT5	9	p5_ct1	$sol(P5) = \emptyset$
	CPUT6	9	p6_ct2	[0 1 6 11]
	CPUT7	9	p7_ct9	$sol(P7) = \emptyset$
Localization	Golomb rulers	fault localized		time
	CPUT1	(rev(P1), (m_ct2))		0.32s
	CPUT2	(rev(P2), (m_ct2))		0.82s
	CPUT3	(rev(P3), (m_ct2))		1.15s
	CPUT4	(rev(p4_ct5),$\emptyset$)		1.87s
	CPUT5	(rev(p5_ct1),$\emptyset$)		1.62s
	CPUT6	(rev(P6), (m_ct2))		2.14s
	CPUT7	(rev(p7_ct6,p7_ct9),$\emptyset$)		1.92s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (Golomb Rulers)

	Golomb rulers	# consts	fault injected	non-conformity detected
Test	CPUT1	2	p1_ct2	[0 1 3 6]
	CPUT2	4	p2_ct3	[0 1 6 11]
	CPUT3	5	p3_ct3	[0 5 6 11]
	CPUT4	9	p4_ct5	$sol(P4) = \emptyset$
	CPUT5	9	p5_ct1	$sol(P5) = \emptyset$
	CPUT6	9	p6_ct2	[0 1 6 11]
	CPUT7	9	p7_ct9	$sol(P7) = \emptyset$
Localization	Golomb rulers	fault localized		time
	CPUT1	(rev(P1), (m_ct2))		0.32s
	CPUT2	(rev(P2), (m_ct2))		0.82s
	CPUT3	(rev(P3), (m_ct2))		1.15s
	CPUT4	(rev(p4_ct5),∅)		1.87s
	CPUT5	(rev(p5_ct1),∅)		1.62s
	CPUT6	(rev(P6), (m_ct2))		2.14s
	CPUT7	(rev(p7_ct6,p7_ct9),∅)		1.92s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (Golomb Rulers)

	Golomb rulers	# consts	fault injected	non-conformity detected
Test	CPUT1	2	p1_ct2	[0 1 3 6]
	CPUT2	4	p2_ct3	[0 1 6 11]
	CPUT3	5	p3_ct3	[0 5 6 11]
	CPUT4	9	p4_ct5	$sol(P4) = \emptyset$
	CPUT5	9	p5_ct1	$sol(P5) = \emptyset$
	CPUT6	9	p6_ct2	[0 1 6 11]
	CPUT7	9	p7_ct9	$sol(P7) = \emptyset$

	Golomb rulers	fault localized	time
Localization	CPUT1	(rev(P1), (m_ct2))	0.32s
	CPUT2	(rev(P2), (m_ct2))	0.82s
	CPUT3	(rev(P3), (m_ct2))	1.15s
	CPUT4	(rev(p4_ct5),∅)	1.87s
	CPUT5	(rev(p5_ct1),∅)	1.62s
	CPUT6	(rev(P6), (m_ct2))	2.14s
	CPUT7	(rev(p7_ct6,p7_ct9),∅)	1.92s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (Golomb Rulers)

	Golomb rulers	# consts	fault injected	non-conformity detected
Test	CPUT1	2	p1_ct2	[0 1 3 6]
	CPUT2	4	p2_ct3	[0 1 6 11]
	CPUT3	5	p3_ct3	[0 5 6 11]
	CPUT4	9	p4_ct5	$sol(P4) = \emptyset$
	CPUT5	9	p5_ct1	$sol(P5) = \emptyset$
	CPUT6	9	p6_ct2	[0 1 6 11]
	CPUT7	9	p7_ct9	$sol(P7) = \emptyset$
Localization	Golomb rulers	fault localized		time
	CPUT1	(rev(P1), (m_ct2))		0.32s
	CPUT2	(rev(P2), (m_ct2))		0.82s
	CPUT3	(rev(P3), (m_ct2))		1.15s
	CPUT4	(rev(p4_ct5),∅)		1.87s
	CPUT5	(rev(p5_ct1),∅)		1.62s
	CPUT6	(rev(P6), (m_ct2))		2.14s
	CPUT7	(rev(p7_ct6,p7_ct9),∅)		1.92s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (Car Sequencing)

	car sequencing	# constraints	fault injected	non-conformity detected
Test	CPUT1	2	p1_ct2	slots1
	CPUT2	3	p2_ct3	slots2
	CPUT3	3	p3_ct2	slots3
	CPUT4	3	p4_ct2	$sol(P4) = \emptyset$
	CPUT5	6	p5_ct1	$sol(P5) = \emptyset$
	CPUT6	6	p6_ct6	$sol(P6) = \emptyset$
	CPUT7	6	p7_ct5	$sol(P7) = \emptyset$
		slots1= [4 5 3 6 4 6 5 1 3 2] slots2= [4 6 3 1 5 2 3 5 4 6] slots3= [5 2 3 6 1 4 3 6 4 5]		
Localization	cSeq	fault localized		time
	CPUT1	(rev(P1), (m_ct2))		0.46s
	CPUT2	(rev(P2), (m_ct2))		0.23s
	CPUT3	(rev(P3), (m_ct2))		0.67s
	CPUT4	(rev(p4_ct2),$\emptyset$)		0.87s
	CPUT5	(rev(p5_ct1),$\emptyset$)		1.29s
	CPUT6	(rev(p6_ct6),$\emptyset$)		1.34s
	CPUT7	(rev(p7_ct5, p7_ct6),$\emptyset$)		1.28s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (Car Sequencing)

	car sequencing	# constraints	fault injected	non-conformity detected
Test	CPUT1	2	p1_ct2	slots1
	CPUT2	3	p2_ct3	slots2
	CPUT3	3	p3_ct2	slots3
	CPUT4	3	p4_ct2	$sol(P4) = \emptyset$
	CPUT5	6	p5_ct1	$sol(P5) = \emptyset$
	CPUT6	6	p6_ct6	$sol(P6) = \emptyset$
	CPUT7	6	p7_ct5	$sol(P7) = \emptyset$

slots1= [4 5 3 6 4 6 5 1 3 2]
slots2= [4 6 3 1 5 2 3 5 4 6]
slots3= [5 2 3 6 1 4 3 6 4 5]

	cSeq	fault localized	time
Localization	CPUT1	(rev(P1), (m_ct2))	0.46s
	CPUT2	(rev(P2), (m_ct2))	0.23s
	CPUT3	(rev(P3), (m_ct2))	0.67s
	CPUT4	(rev(p4_ct2),$\emptyset$)	0.87s
	CPUT5	(rev(p5_ct1),$\emptyset$)	1.29s
	CPUT6	(rev(p6_ct6),$\emptyset$)	1.34s
	CPUT7	(rev(p7_ct5, p7_ct6),$\emptyset$)	1.28s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (Car Sequencing)

	car sequencing	# constraints	fault injected	non-conformity detected
Test	CPUT1	2	p1_ct2	slots1
	CPUT2	3	p2_ct3	slots2
	CPUT3	3	p3_ct2	slots3
	CPUT4	3	p4_ct2	$sol(P4) = \emptyset$
	CPUT5	6	p5_ct1	$sol(P5) = \emptyset$
	CPUT6	6	p6_ct6	$sol(P6) = \emptyset$
	CPUT7	6	p7_ct5	$sol(P7) = \emptyset$

slots1= [4 5 3 6 4 6 5 1 3 2]
slots2= [4 6 3 1 5 2 3 5 4 6]
slots3= [5 2 3 6 1 4 3 6 4 5]

	cSeq	fault localized	time
Localization	CPUT1	(rev(P1), (m_ct2))	0.46s
	CPUT2	(rev(P2), (m_ct2))	0.23s
	CPUT3	(rev(P3), (m_ct2))	0.67s
	CPUT4	(rev(p4_ct2),∅)	0.87s
	CPUT5	(rev(p5_ct1),∅)	1.29s
	CPUT6	(rev(p6_ct6),∅)	1.34s
	CPUT7	(rev(p7_ct5, p7_ct6),∅)	1.28s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (Car Sequencing)

	car sequencing	# constraints	fault injected	non-conformity detected
Test	CPUT1	2	p1_ct2	slots1
	CPUT2	3	p2_ct3	slots2
	CPUT3	3	p3_ct2	slots3
	CPUT4	3	p4_ct2	$sol(P4) = \emptyset$
	CPUT5	6	p5_ct1	$sol(P5) = \emptyset$
	CPUT6	6	p6_ct6	$sol(P6) = \emptyset$
	CPUT7	6	p7_ct5	$sol(P7) = \emptyset$

slots1= [4 5 3 6 4 6 5 1 3 2]
slots2= [4 6 3 1 5 2 3 5 4 6]
slots3= [5 2 3 6 1 4 3 6 4 5]

	cSeq	fault localized	time
Localization	CPUT1	(rev(P1), (m_ct2))	0.46s
	CPUT2	(rev(P2), (m_ct2))	0.23s
	CPUT3	(rev(P3), (m_ct2))	0.67s
	CPUT4	(rev(p4_ct2),∅)	0.87s
	CPUT5	(rev(p5_ct1),∅)	1.29s
	CPUT6	(rev(p6_ct6),∅)	1.34s
	CPUT7	(rev(p7_ct5, p7_ct6),∅)	1.28s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (n-queens)

	8-queens	# consts.	fault injected	non-conformity detected
Test	CPUT1	12	p1_ct11	$sol(P1) = \emptyset$
	CPUT2	12	p2_ct12	$sol(P2) = \emptyset$
	CPUT3	12	p3_ct11	$sol(P3) = \emptyset$
	CPUT4	12	p4_ct12	$sol(P4) = \emptyset$
	CPUT5	5	p5_ct3	q1
	CPUT6	3	p6_ct3	q2
	CPUT7	2	p7_ct1	q3
non-conformity		q1= [1 5 1 1 4 8 8 4]		
q2= [7 8 3 2 4 1 5 6], q3= [8 4 3 6 5 7 2 1]				
Localization	8-queens	fault localized		time
	CPUT1	(rev(p1_ct11),∅)		2.09s
	CPUT2	(rev(p2_ct12),∅)		1.64s
	CPUT3	(rev(p3_ct11),∅)		1.57s
	CPUT4	(rev(p4_ct12),∅)		1.59s
	CPUT5	(rev(p5_ct3),∅)		0.62s
	CPUT6	(rev(P6), (m_ct2,m_ct3))		4.12s
	CPUT7	(rev(P7), (m_ct2,m_ct3))		3.01s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (n-queens)

	8-queens	# consts.	fault injected	non-conformity detected
Test	CPUT1	12	p1_ct11	$sol(P1) = \emptyset$
	CPUT2	12	p2_ct12	$sol(P2) = \emptyset$
	CPUT3	12	p3_ct11	$sol(P3) = \emptyset$
	CPUT4	12	p4_ct12	$sol(P4) = \emptyset$
	CPUT5	5	p5_ct3	q1
	CPUT6	3	p6_ct3	q2
	CPUT7	2	p7_ct1	q3
non-conformity			q1= [1 5 1 1 4 8 8 4]	
			q2= [7 8 3 2 4 1 5 6], q3= [8 4 3 6 5 7 2 1]	

	8-queens	fault localized	time
Localization	CPUT1	(rev(p1_ct11),∅)	2.09s
	CPUT2	(rev(p2_ct12),∅)	1.64s
	CPUT3	(rev(p3_ct11),∅)	1.57s
	CPUT4	(rev(p4_ct12),∅)	1.59s
	CPUT5	(rev(p5_ct3),∅)	0.62s
	CPUT6	(rev(P6), (m_ct2,m_ct3))	4.12s
	CPUT7	(rev(P7), (m_ct2,m_ct3))	3.01s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (Social Golfer)

	social golfer	# constrs.	fault injected	non-conformity detected
Test	CPUT1	5	p1_ct1	$sol(P1) = \emptyset$
	CPUT2	5	p2_ct2	nc1
	CPUT3	5	p3_ct3	$sol(P3) = \emptyset$
	CPUT4	5	p4_ct4	$sol(P4) = \emptyset$
	CPUT5	5	p5_ct5	$sol(P5) = \emptyset$
non-conformity	nc1= [[1 1 1 1][1 2 2 2][1 3 3 3][2 1 3 3][2 2 2 1][2 2 1 1][3 3 3 2][3 1 1 2][3 3 2 3]]			
Localization	social golfer	fault localized		time
	CPUT1	(rev(p1_ct1),∅)		1.04s
	CPUT2	(rev(P2), (m_ct2))		0.18s
	CPUT3	(rev(p3_ct3),∅)		1.05s
	CPUT4	(rev(p4_ct4,p4_ct5),∅)		1.03s
	CPUT5	(rev(p5_ct5),∅)		1.15s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Validation (Social Golfer)

	social golfer	# constrs.	fault injected	non-conformity detected
Test	CPUT1	5	p1_ct1	$sol(P1) = \emptyset$
	CPUT2	5	p2_ct2	nc1
	CPUT3	5	p3_ct3	$sol(P3) = \emptyset$
	CPUT4	5	p4_ct4	$sol(P4) = \emptyset$
	CPUT5	5	p5_ct5	$sol(P5) = \emptyset$
non-conformity	nc1= [[1 1 1 1][1 2 2 2][1 3 3 3] [2 1 3 3][2 2 2 1][2 2 1 1][3 3 3 2][3 1 1 2][3 3 2 3]]			
Localization	social golfer	fault localized		time
	CPUT1	(rev(p1_ct1),$\emptyset$)		1.04s
	CPUT2	(rev(P2), (m_ct2))		0.18s
	CPUT3	(rev(p3_ct3),$\emptyset$)		1.05s
	CPUT4	(rev(p4_ct4,p4_ct5),$\emptyset$)		1.03s
	CPUT5	(rev(p5_ct5),$\emptyset$)		1.15s

Intel Core 2 Duo CPU, 2.39 GHz, 2.00 Go RAM

Conclusion

Fault localization in constraint programs is possible

But,
our method is based on the hypothesis that there is a single fault in the program!

Conclusion

Fault localization in constraint programs is possible

But,
our method is based on the hypothesis that there is a single fault in the program!

Perspectives

- Extend to multiple faults
  by considering Powset($\{C_1,\ldots C_n\}$)

- Extend to better correction
  by calculating the correction set

Conclusion

Fault localization in constraint programs is possible

But,
our method is based on the hypothesis that there is a single fault in the program!

Perspectives

- Extend to multiple faults
  by considering Powset($\{C_1,\ldots C_n\}$)

- Extend to better correction
  by calculating the correction set

Existing debugging tools

- Syntax checking

  (OPL studio [IBM ILOG]…)

- Host's language debuggers

  (gdb for Gecode, jdb for Choco…)

- Post-mortem trace analyzers

  (**Codeine** for Prolog[Langevine03], **Morphine** for Mercury,[Jahier02], ILOG **Gentra4CP**[Deransart 04], [*Jussien*96] **Jpalm/Jchoco** …)

- Trace visualization

  (CLPGUI[Fages04],CP-VIZ[Simonis10] …)