Constraint Programming and Abstract Intepretation

Séminaire Master Science Informatique
Rennes
4 novembre 2019

Charlotte Truchet

LS2N, UMR 6004, Université de Nantes

Based on joint works with

Marie Pelleau
MCF U. Nice

Ghiles Ziat
PhD U. Paris 6

Antoine Miné
Prof. U. Paris 6

Pierre Talbot
Post-doc U. Nantes

Mathieu Vavrille
Master ENS Lyon

Outline

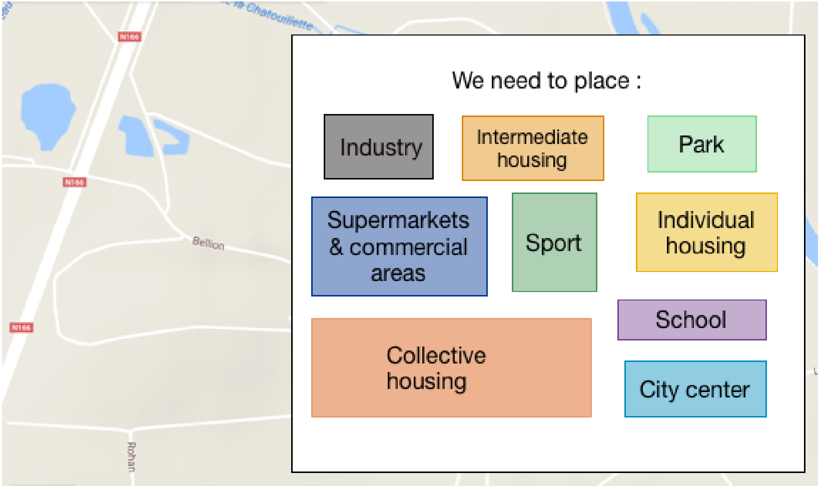
Introduction to CP

Complete solving
- Consistency
- Backtrack search

Abstract solving
- Abstract Domains for CP
- Octagons
- Combining abstract domains

CP on an example: Urban planning

CP on an example: Urban planning



CP on an example: Urban planning

For each urban form, we know:

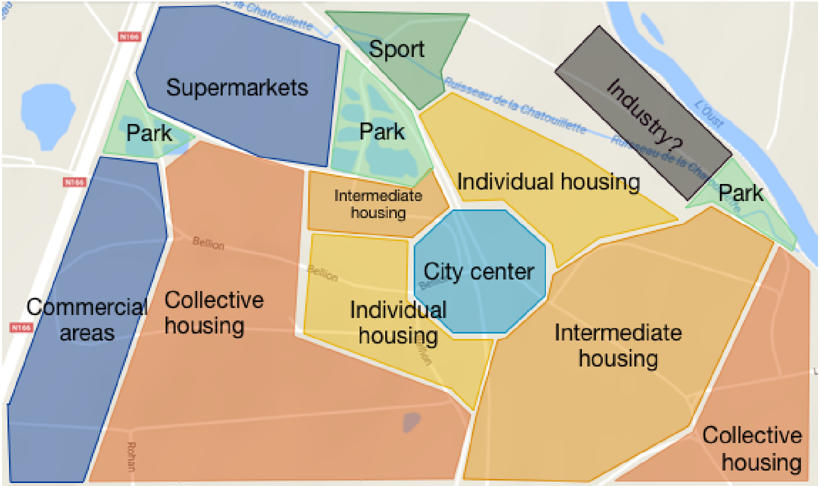
- the surface they need, as a number of blocks,
- a series of preferences,

  industries like rivers and roads, schools have to be near housing areas, etc
- some hard constraints.

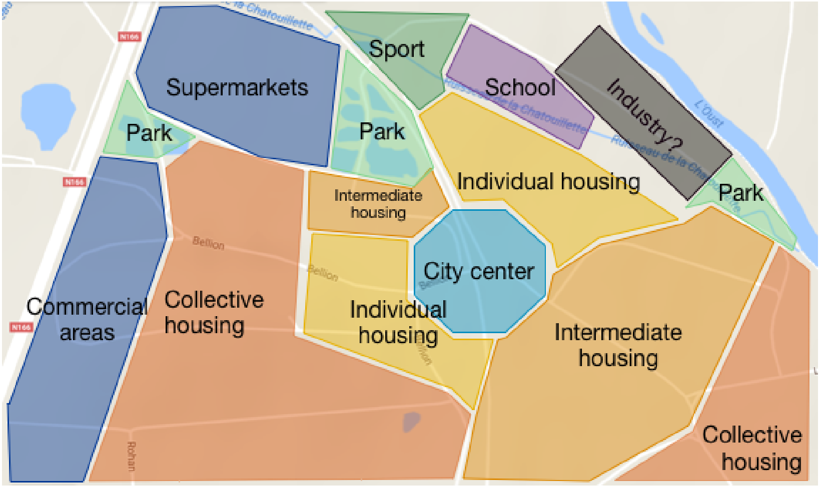
  a housing block has to be at a walking distance from a park, some urban forms must have a minimum size, etc

CP on an example: Urban planning

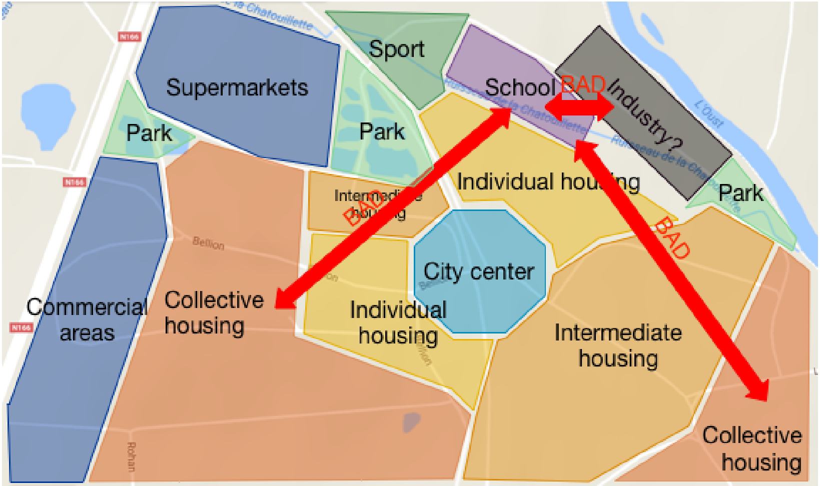
CP on an example: Urban planning



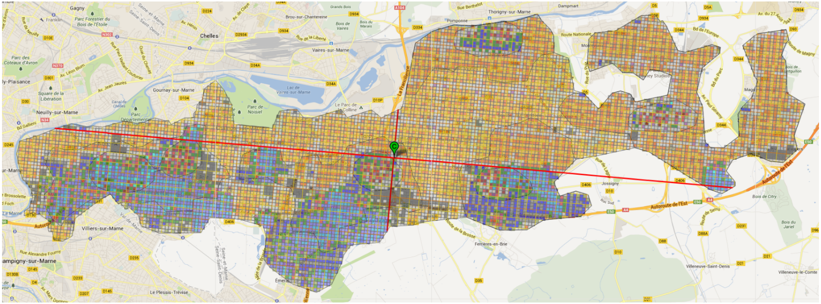
CP on an example: Urban planning



CP on an example: Urban planning



Sustain project



Sustain Projet, simulation on Marne-la-Vallée,
a city of 8728 hectares, 230 000 inhabitants, $\sim$ 10 000 cells.
PhD of Bruno Belin, 2011-2014

Constraint Programming

In practice:

- combinatorial problem:
  - we need to make choices,
  - choices may have consequences long after they have been made,
  - it must be possible to revise the choices (mark them).
- declarative problem:
  - checking is easy, based on rules or user knowledge,
  - efficiently building is difficult.

CP

Constraint Programming (CP) is both:

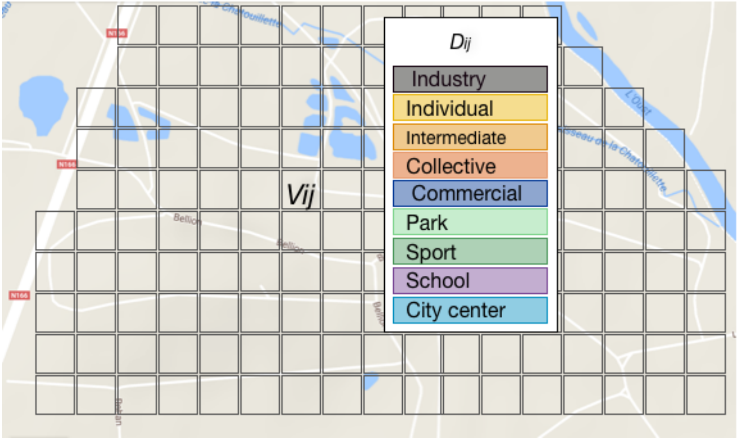
- **AI** an efficient tool for declarative programming,
- **OR** a series of algorithms for combinatorial (sub)structures.

Many applications on a wide range of problems:

- logistics/planning: vehicle routing, nurse roastering, matching...
- sustainable development: energy optimization, lifetime...
- arts, music, computer graphics: automatic harmonization, CAD...
- verification/software engineering: test generation, floating point abstractions...
- medicine, football games, cryptography, ...

Definitions

A variable is an unknown of the problem. It has a given domain, set of values the variable can take.



Definitions

A variable is an unknown of the problem. It has a given domain, set of values the variable can take.

A constraint is a logical relation on variables.

Definitions

A variable is an unknown of the problem. It has a given domain, set of values the variable can take.

A constraint is a logical relation on variables.

A consistent domain for a given constraint is a domain which does not contain infeasible values.

Sustain project

In collaboration with urban planners from EPAMarne

- model of the problem based on urban planners' expertise,
- solver based on a parallelized local search algorithm,
- interactive mode to re-compute partially modified solutions.

PhD of Bruno Belin
collaboration with Marc Christie, Frédéric Benhamou

Sustain project

Other examples of real-life applications

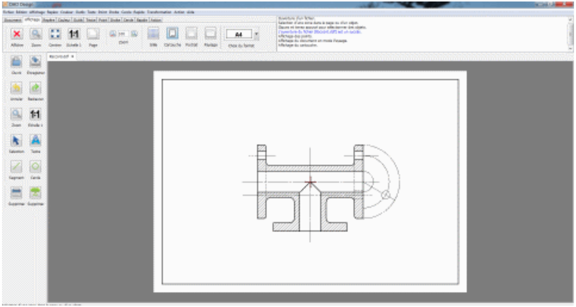
Placement of VMs on real machines (BtrPlace, Entropy project), solver Choco

Other examples of real-life applications

Planning of medical examinations (radio, etc), Medicalis, solver Choco

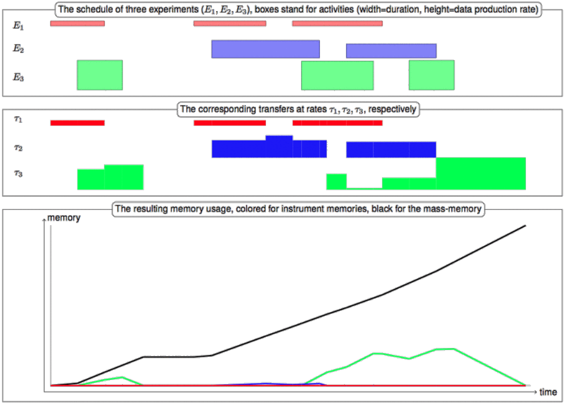
Other examples of real-life applications

Computation of geometrical measures in CAO, DaoDesign (free), solver Choco



Other examples of real-life applications

Scheduling for the Philae robot (for instance, data transfer) with ressource constraints (memory, energy).



Constraint Satisfaction Problem

A *Constraint Satisfaction Problem* (CSP) is given by:

- variables $V_1 \ldots V_n$ (n fixed),
- domains $D_1 \ldots D_n$, where D_i is the set of values that variable V_i can take,
  often finite subsets of $\mathbb{N}$, **or** subsets of $\mathbb{R}$,
- constraints $C_1 \ldots C_p$, logical relations on the variables.

A *solution* of the problem is an instanciation of values of the domains, to the variables, such that the constraints are satisfied.

Constraint Satisfaction Problem

For continuous variables, if solutions are not computer-representable, a solution can be given

- by an over-approximation of the solution set (complete solver),
- by an inner-approximation (correct solver).

Constraints

Constraint languages include in general:

- arithmetic expressions and "reasonable" functions,
- comparison operators: $<, \leq, >, \geq, =, \neq$,
  $V_1 + 7 = V_3$,
  $V_1 * V_3 < 10$
  $\sum_i V_i < M$

Constraints

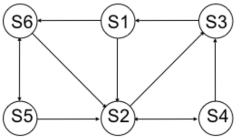
Constraint languages include in general:

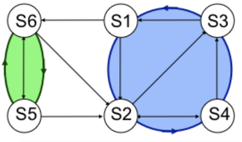
- arithmetic expressions and "reasonable" functions,
- comparison operators: $<, \leq, >, \geq, =, \neq$,
- global constraints:

Constraints

Constraint languages include in general:

- arithmetic expressions and "reasonable" functions,
- comparison operators: $<, \leq, >, \geq, =, \neq$,
- global constraints:
  - on graphs : `tree`, `forest`, `circuit`...

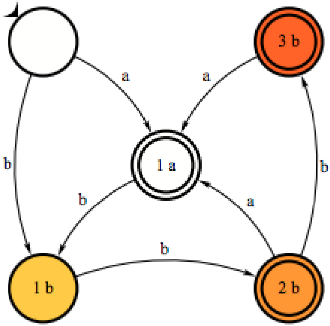




Constraints

Constraint languages include in general:

- arithmetic expressions and "reasonable" functions,
- comparison operators: $<, \leq, >, \geq, =, \neq$,
- global constraints:
  - on graphs : `tree`, `forest`, `circuit`...
  - on words: `regular`, `cost-regular`, ...



Constraints

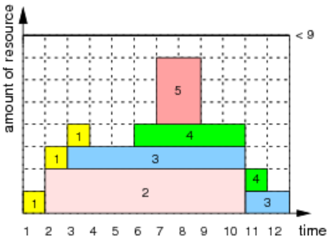
Constraint languages include in general:

- arithmetic expressions and "reasonable" functions,
- comparison operators: $<, \leq, >, \geq, =, \neq$,
- global constraints:
  - on graphs : `tree`, `forest`, `circuit`...
  - on words: `regular`, `cost-regular`, ...
  - for practical reasons : `element`, `table`...

Constraints

Constraint languages include in general:

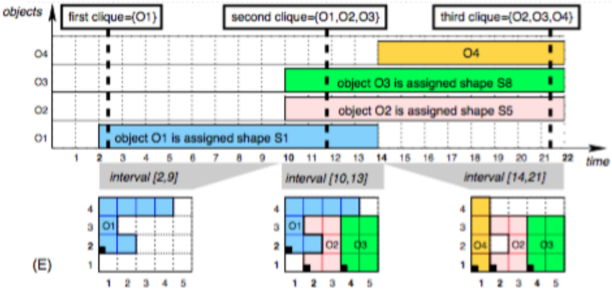
- arithmetic expressions and "reasonable" functions,
- comparison operators: $<, \leq, >, \geq, =, \neq$,
- global constraints:
  - on graphs : `tree`, `forest`, `circuit`...
  - on words: `regular`, `cost-regular`, ...
  - for practical reasons : `element`, `table`...
  - specific to common problems : `cumulative`, `geost`,



Constraints

Constraint languages include in general:

- arithmetic expressions and "reasonable" functions,
- comparison operators: $<, \leq, >, \geq, =, \neq$,
- global constraints:
  - on graphs : `tree`, `forest`, `circuit`...
  - on words: `regular`, `cost-regular`, ...
  - for practical reasons : `element`, `table`...
  - specific to common problems : `cumulative`, `geost`,



Constraints

Constraint languages include in general:

- arithmetic expressions and "reasonable" functions,
- comparison operators: $<, \leq, >, \geq, =, \neq$,
- global constraints:
  - on graphs : `tree`, `forest`, `circuit`...
  - on words: `regular`, `cost-regular`, ...
  - for practical reasons : `element`, `table`...
  - specific to common problems : `cumulative`, `geost`,
  - on cardinality : `alldifferent`, `nvalue`, `atleast`, `gcc`...

Constraints

Constraint languages include in general:

- arithmetic expressions and "reasonable" functions,
- comparison operators: $<, \leq, >, \geq, =, \neq$,
- global constraints:
  - on graphs : `tree`, `forest`, `circuit`...
  - on words: `regular`, `cost-regular`, ...
  - for practical reasons : `element`, `table`...
  - specific to common problems : `cumulative`, `geost`,
  - on cardinality : `alldifferent`, `nvalue`, `atleast`, `gcc`...

Nearly all global constraints are indexed in the *Global Constraint Catalog*, with a common format and all the bibliography.
`http://sofdem.github.io/gccat/`

Outline

Introduction to CP

Complete solving
- Consistency
- Backtrack search

Abstract solving

Consistency on finite domains

A constraint $C(V_1 ... V_n)$ is generalized arc-consistent (GAC) for domains $D_1 ... D_n$ iff for every variable V_i, for every value $v^i \in D_i$, there exist values $v^1 \in D_1, ..., v^{i-1} \in D_{i-1}, v^{i+1} \in D_{i+1}, ..., v^n \in D_n$ such that $C(v^1, ... v^n)$.

Consistency on finite domains

A constraint $C(V_1 ... V_n)$ is generalized arc-consistent (GAC) for domains $D_1 ... D_n$ iff for every variable V_i, for every value $v^i \in D_i$, there exist values
$v^1 \in D_1, ..., v^{i-1} \in D_{i-1}, v^{i+1} \in D_{i+1}, ..., v^n \in D_n$ such that $C(v^1, ... v^n)$.

A constraint $C(V_1 ... V_n)$ is bound-consistent (BC) for domains $D_1 ... D_n$ iff the bounds of the domains are consistent (as defined above).

Consistency on continuous domains

A constraint C on variables $V_1 \dots V_n$, with domains $D_1 \dots D_n$ is Hull consistent (HC) iff $D_1 \times \cdots \times D_n$ is the smallest real box with floating point bounds, including solutions C, in $D_1 \times \cdots \times D_n$.

Remark: there are plenty of other consistencies (discrete: path-consistency, singleton arc-consistency, strong consistencies... / continuous: Box consistency, MOHCC...)

Examples

- $X = Y + 3 * Z$
  if $X = 10$, $Y = 4$ then $Z = -2$,

Examples

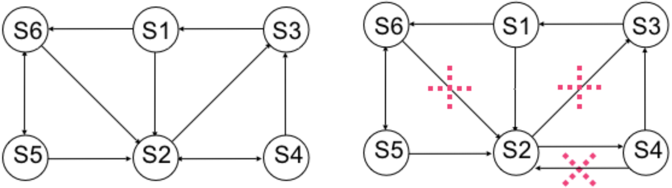
- $X = Y + 3 * Z$
  if $X = 10$, $Y = 4$ then $Z = -2$,
- $X = Y + 3 * Z$
  if $D_Z = \{1..5\}$ and $D_X = \{0..10\}$ then D_Y can be intersected with $\{-5, 7\}$,

Examples

- $X = Y + 3 * Z$
  if $X = 10$, $Y = 4$ then $Z = -2$,
- $X = Y + 3 * Z$
  if $D_Z = \{1..5\}$ and $D_X = \{0..10\}$ then D_Y can be intersected with $\{-5, 7\}$,
- $\texttt{alldifferent}(X_1, X_2, X_3)$
  if we know that D_1 **and** D_2 are $\{1, 2\}$, the values 1 and 2 can be removed from D_3.

Examples

- $X = Y + 3 * Z$
  if $X = 10$, $Y = 4$ then $Z = -2$,
- $X = Y + 3 * Z$
  if $D_Z = \{1..5\}$ and $D_X = \{0..10\}$ then D_Y can be intersected with $\{-5, 7\}$,
- `alldifferent`(X_1, X_2, X_3)
  if we know that D_1 **and** D_2 are $\{1, 2\}$, the values 1 and 2 can be removed from D_3.
- cycle constraint in a graph :



Propagation

Propagating a constraint C on domains $D_1 ... D_n$ is removing from $D_1 ... D_n$ all the inconsistent values for C.

For a conjunction of constraints, for each constraint the propagators are applied until a fixpoint is reached [Benhamou, 1996, Apt, 1999].

Propagation

All in all, a propagation loop mixes:

- generic propagators for atomic constraints,
- specific propagators for global constraints,
- generic methods (often event-based) to wake the propagators and efficiently combine them.

Designing an efficient propagation loop (fixpoint acceleration) is still a challenge [Schulte and Tack, 2001].

Solving ?

Consistency is not enough, in general, for computing a solution (all solutions).

Complete solving methods

Two phases are iterated

- propagation of the contraints (deductions),
- splits / instantiations : assertions on the domains, which may be invalidated later (backtrack).

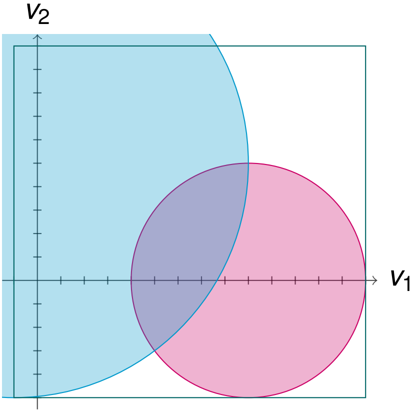
Continuous Solving Method

```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← Hull-Consistency(e)
  if e ≠ ∅ then
    if maxDim(e) ≤ r or isSol(e)
    then
      sols ← sols ∪ e
    else
      split e in two boxes e1 and
      e2
      push e1 and e2 in toExplore
```



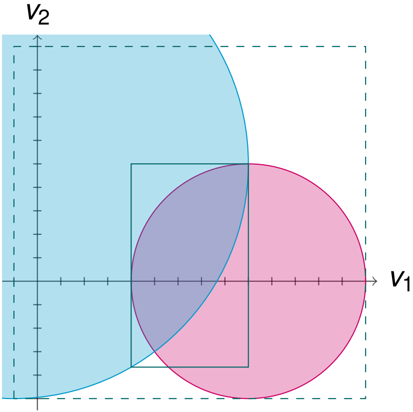
Continuous Solving Method

```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e)≤ r or isSol(e)
      then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



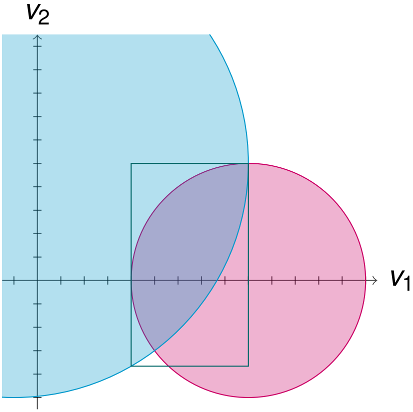
Continuous Solving Method

```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e) ≤ r or isSol(e)
      then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



Continuous Solving Method

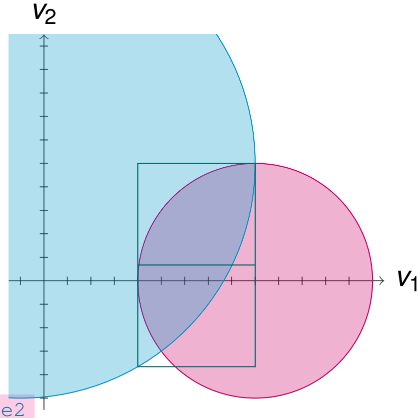
```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← Hull-Consistency(e)
  if e ≠ ∅ then
    if maxDim(e) ≤ r or isSol(e)
    then
      sols ← sols ∪ e
    else

      split e in two boxes e1 and e2
      push e1 and e2 in toExplore
```



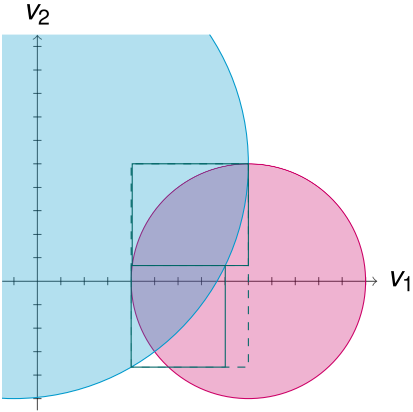
Continuous Solving Method

```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e)≤ r or isSol(e)
      then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



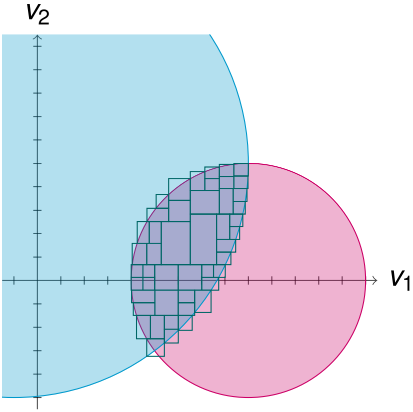
Continuous Solving Method

```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e)≤ r or isSol(e)
      then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



Heuristics

- dom: smallest domain first,
- deg, wdeg: most constrained variable first (possibly with weights),
- dom/wdeg: the previous ones combined,
- activity: dynamically adapts to the *efficiency* of the constraints,
- counting-based search: uses estimations (or ub) of the number of solutions for the global constraints (cardinality),
- on continuous domains, largest dimension first,
- *ad hoc* heuristics.

There is no such thing as a Free Lunch.

Some active solvers

- Choco: java library, free
  `http://www.emn.fr/z-info/choco-solver/`
- gecode: C++ library, free
  `-http://www.gecode.org/`
- ORTools: C++, interface in Python, free,
  `https://code.google.com/p/or-tools/`
- Oscar: Scala, free,
  `https:`
  `//bitbucket.org/oscarlib/oscar/wiki/Home`
- Prolog family: ECLiPSe, Sicstus
- AbSolute, OCaml, free,
- plenty of others!

Disambiguation

	CP	**SAT/SMT**
Vars	int **or** real **or** symb	bool+MT
Const	various	clauses+MT
Solv	backtrack	DPLL
Propag	*ad hoc*	unit
Learning	nogoods	clause learning
Implem	support (AC6+)	watched literals

CP is good at: global reasoning on combinatorial problems, modeling tools, dirty problems.
CP is bad at: mixing variables of different types, learning.

Outline

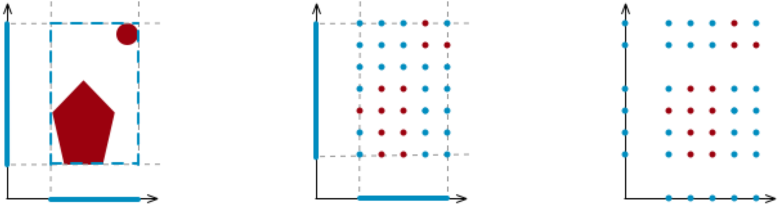
Introduction to CP

Complete solving

Abstract solving

- Abstract Domains for CP
- Octagons
- Combining abstract domains

Consistency



Hull-consistency *Bound-consistency* *Generalized arc-consistency*

Two key remarks

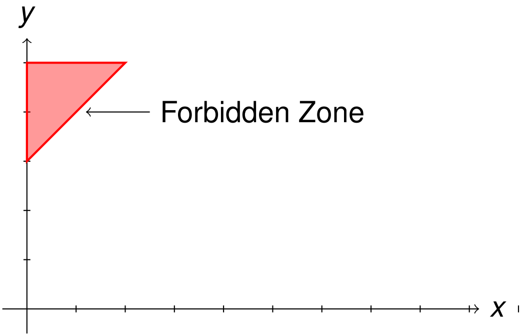
- consistency is not about where the solutions are, it is about where they are *not*,
- why square?

Abstract Interpretation

- Abstract Interpretation (AbsInt) is a theory of approximation of program semantics [Cousot and Cousot, 1976]
- Applied to static analysis and verification of software
- Goal: automatically prove that a program does not have execution errors
- Key idea: abstract the valuations of the programs variables

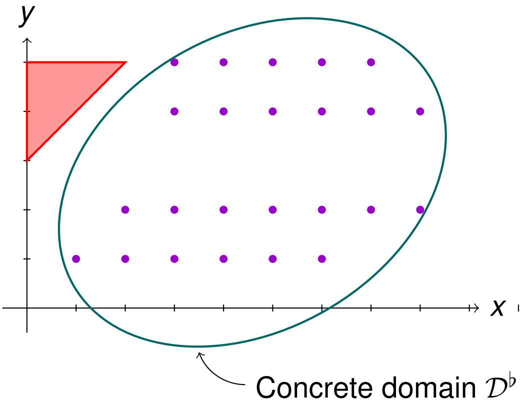
Abstract Domain

```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x-1
y ← y+1
```



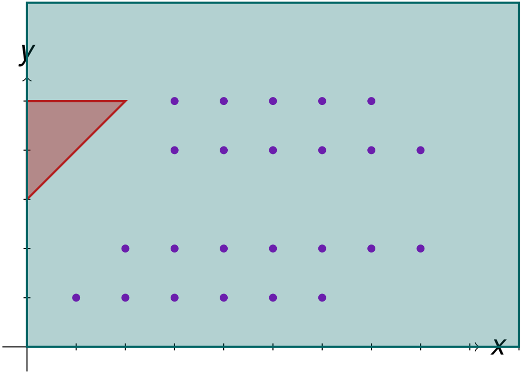
Abstract Domain

```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x-1
y ← y+1
```



Abstract Domain

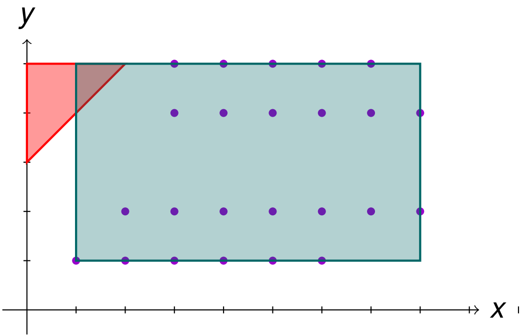
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x-1
y ← y+1
```



Boxes

Abstract Domain

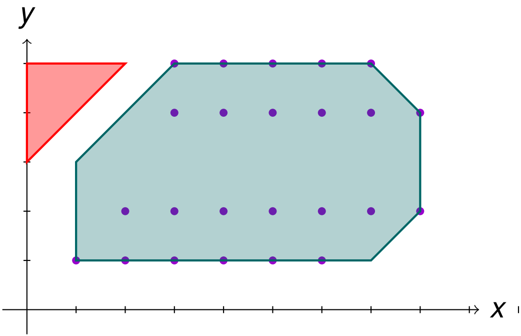
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x−1
y ← y+1
```



Better boxes

Abstract Domain

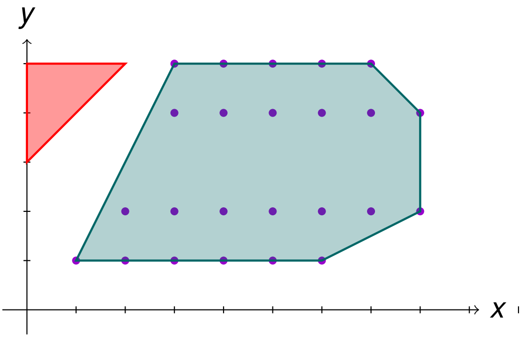
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x-1
y ← y+1
```



Octagons

Abstract Domain

```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x−1
y ← y+1
```



Convex polyhedra

AI ? CP ?

AI in a nutshell

We may not know where a program is going. But it is fine, as long as we know where the program is not going.

AI ? CP ?

AI in a nutshell

We may not know where a program is going. But it is fine, as long as we know where the program is not going.

CP in a nutshell

We make huge efforts to compute where solutions cannot be.

Links

$CP \cap AI$

Approximations of some spaces which are undecidable, or difficult to compute:

- solution space in CP,
- traces in AI.

$AI \setminus CP$

- many abstract domains,
- reduced products (combining abstract domains).

$CP \setminus AI$

- heuristics,
- precision.

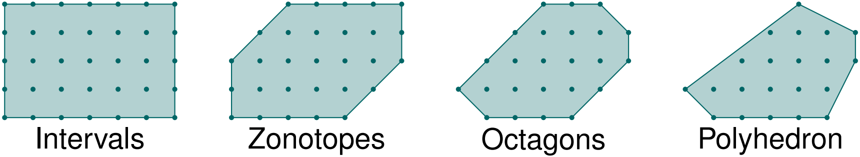
Abstract Solving Method

Central question

Given a CSP, is it possible to write a program such that a static analysis of this program gives the solutions of the CSP?

We define the resolution as a concrete semantics.

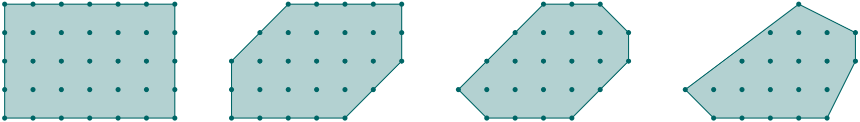
What already exist in AI



Abstract domains come with:

- transfer functions $\rho^\sharp$ (assignment, test, …)
- meet $\cap^\sharp$ and join $\cup^\sharp$
- widening $\triangledown^\sharp$ and narrowing $\vartriangle^\sharp$

What already exist in AI



Intervals | Zonotopes | Octagons | Polyhedron

Abstract domains come with:

- transfer functions $\rho^\sharp$ (assignment, test, ...)
- meet $\cap^\sharp$ and join $\cup^\sharp$
- widening $\nabla^\sharp$ and narrowing $\triangle^\sharp$

We need:

- a consistency/propagation ρ
- a splitting operator $\oplus$
- a size function τ

Abstract Solving Method

Propagation

- Constraint propagators are test transfer functions
  Hull consistency algorithm HC4 is exactly the same algorithm as Bottom-Up Top-Down in Abstract Interpretation [Cousot and Cousot, 1977]
- Propagation loop, fixpoint using local iterations [Granger, 1992]

Exploration

- Splitting operator in disjunctive completion: must be added
- Size function: must be added

Continuous Solving Method

```
Parameter:  float r

list of boxes  sols ← ∅
queue of boxes  toExplore ← ∅
box  e ← D

push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← propagate(e)
  if e ≠ ∅ then
    if maxDim(e)  ≤ r or isSol(e) then
      sols ← sols ∪ e
    else
      split e in two boxes e1 and e2
      push e1 and e2  in toExplore
```

Abstract Solving Method

Parameter: float r

~~list of boxes~~ disjunction sols ← ∅
~~queue of boxes~~ disjunction toExplore ← ∅
~~box~~ abstract element e ← ~~D~~ ⊤♯

push e in toExplore

while toExplore ≠ ∅ **do**
  e ← **pop**(toExplore)
  e ← ~~propagate(e)~~ ρ♯(e)
  if e ≠ ∅ **then**
    if ~~maxDim(e)~~ τ(e) ≤ r **or** isSol(e) **then**
      sols ← sols ∪ e
    else
      ~~split e in two boxes e1 **and** e2~~
      push ~~e1 **and** e2~~ ⊕(e) in toExplore

Under some conditions on the operators, this abstract solving method terminates, is correct and/or complete.

AbSolute

AbSolute is a solver:

- in OCaml
- based on the Apron library for numeric abstract domains [Jeannet and Miné, 2009],
- on abstract domains: boxes, octagons, polyhedra, BDDs (currently developed) some reduced products, and many others soon,
- with plenty of fun features: visualization, tikz generation...

`https://github.com/mpelleau/AbSolute`

Now in opam!

Solver architecture

Everything is made on abstract domains.

Abstract domain

```
type domain

val init : prob -> domain
val propagate : domain -> constraints ->
   domain
val split : domain -> domain list
val size : domain -> bool
```

Outline

Introduction to CP

Complete solving

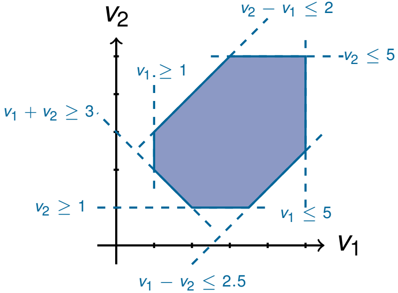
Abstract solving

- Abstract Domains for CP
- Octagons
- Combining abstract domains

Octagons

Definition (Octagon [Miné, 2006])

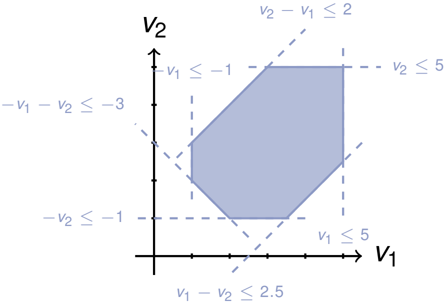
Set of points satisfying a conjunction of constraints of the form $\pm v_i \pm v_j \leq c$, called octagonal constraints



- In dimension n, an octagon has at most $2n^2$ faces
- An octagon can be unbounded

Octagons

Compact representation: smallest Difference Bound Matrix (DBM)

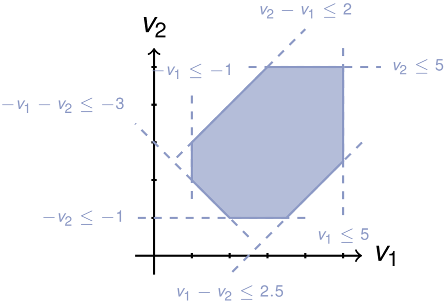


$$\begin{array}{c} \\ v_1 \\ -v_1 \\ v_2 \\ -v_2 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & -v_1 & v_2 & -v_2 \end{array} \\ \begin{pmatrix} 0 & -2 & 2 & -3 \\ 10 & 0 & +\infty & 2.5 \\ 2.5 & -3 & 0 & -2 \\ +\infty & 2 & 10 & 0 \end{pmatrix} \end{array}$$

- provides a normal form (smallest DBM),
- efficient propagation with Floyd-Warshall shortest path algorithm [Miné, 2006].

Octagons

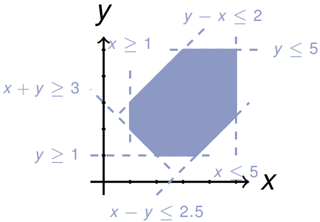
Compact representation: smallest Difference Bound Matrix (DBM)



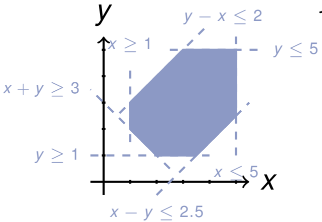
	v_1	$-v_1$	v_2	$-v_2$
v_1	0	-2	2	-3
$-v_1$	10	0	10	2.5
v_2	2.5	-3	0	-2
$-v_2$	10	2	10	0

- provides a normal form (smallest DBM),
- efficient propagation with Floyd-Warshall shortest path algorithm [Miné, 2006].

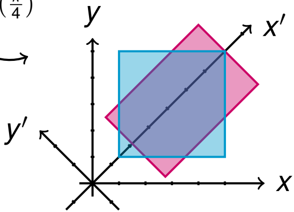
Octagons for CP



Octagons for CP



$$x' = x\cos\left(\frac{\pi}{4}\right) - y\sin\left(\frac{\pi}{4}\right)$$
$$y' = x\sin\left(\frac{\pi}{4}\right) + y\cos\left(\frac{\pi}{4}\right)$$



Representation for CP

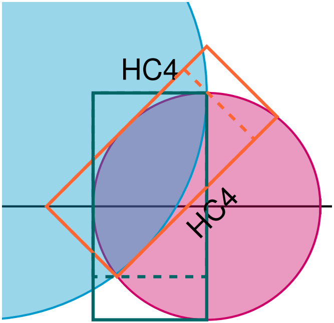
Representation in $\mathcal{O}(n^2)$ for a CSP with n variables and p constraints

- n^2 variables
- $p(n(n-1)+2)/2$ constraints

Back to the boxes: the constraints can be propagated in all the bases.

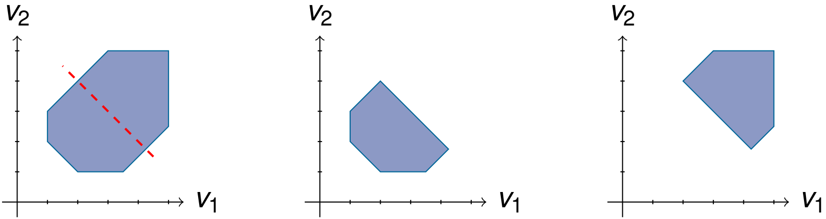
Octagonal Hull Consistency

Interleave the FW algorithm, and Hull-Consistency for each box:
each time a new bound is found by FW, it is replaced by the minimum of the bounds.

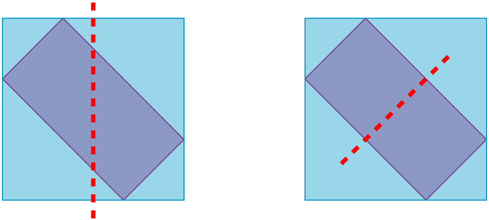


Octagonal Split

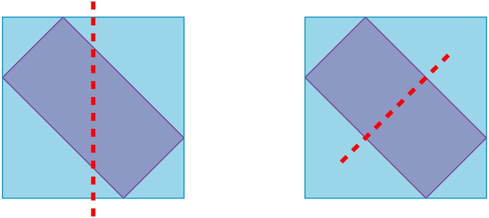
A splitting operator, splits a variable domain



Octagonal Heuristic



Octagonal Heuristic



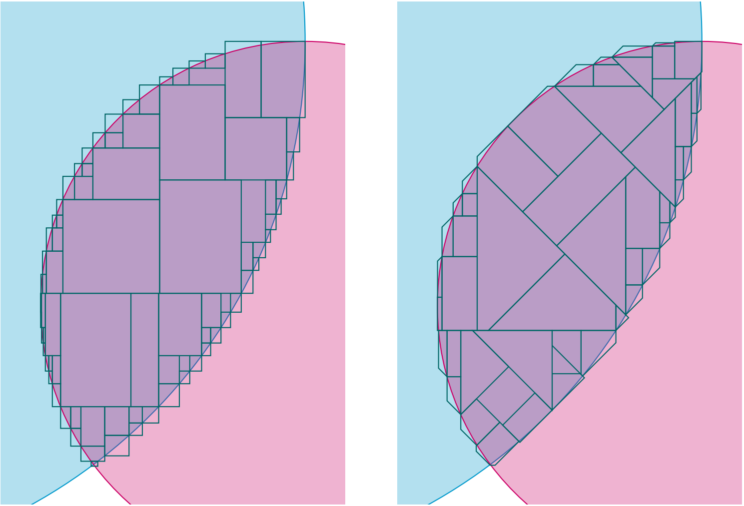
Take the "best" basis, *the box with the minimum of the maximum width*

Split the largest domain in this basis, *the domain with the maximum width*

Octagonal Solving

- We have:
  - an octagonal consistency
  - a splitting operator
  - a choice heuristic
  - a precision.
- We obtain an Octagonal Solver

Output



Same problem with the same time limit.

Experiments

Comparison of an ad-hoc implementation of the same solving algorithm, with the octagon abstract domain or the intervals.

			First solution		All the solutions	
name	nbvar	ctrs	$\mathbb{I}^n$	Oct	$\mathbb{I}^n$	Oct
h75	5	$\leq$	41.40	0.03	-	-
hs64	3	$\leq$	0.01	0.05	-	-
h84	5	$\leq$	5.47	2.54	-	7238.74
KinematicPair	2	$\leq$	0.00	0.00	53.09	16.56
pramanik	3	$=$	28.84	0.16	193.14	543.46
trigo1	10	$=$	18.93	1.38	20.27	28.84
brent-10	10	$=$	6.96	0.54	17.72	105.02
h74	5	$=$ $\leq$	305.98	13.70	1 304.23	566.31
fredtest	6	$=$ $\leq$	3 146.44	19.33	-	-

Solver: Ibex [Chabert and Jaulin, 2009].
Problems from the COCONUT benchmark.
CPU time in seconds, TO 3 hours.

Outline

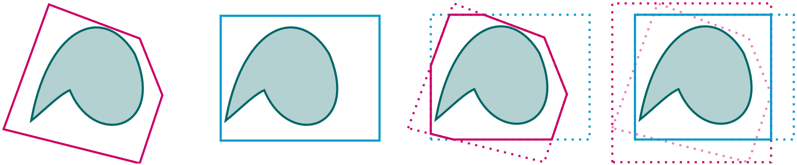
Introduction to CP

Complete solving

Abstract solving
- Abstract Domains for CP
- Octagons
- Combining abstract domains

Reduced Products

A Reduced Product combines two (or more) abstract domains, with reduction operators to transfer information from one to the other [Cousot and Cousot, 1979].



(a) Polyhedra (b) Boxes (c) Reduced Product

Promising Reduced Products

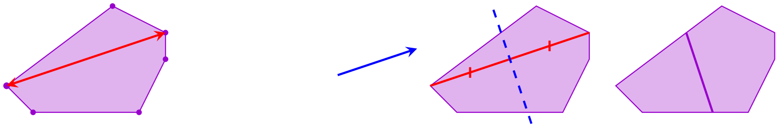
- Box-Polyedra: mixes CP and Operation Research techniques (linear programming & integer linear programming),
  implemented by Ghiles Ziat,
- Integer-Real Boxes: solves problem with both continuous and discrete variables.
  current work: a clever reduced product heuristic, Ghiles Ziat
- Boxes-Integer octagons, with reified constraints: other ways for the domains to communicate
  current work: new ways of learning constraints, Pierre Talbot

Polyedra abstract domain $\mathcal{P}^\sharp$

We use the already existing Polyedra Abstract Domain in double representation (constraints and generators).

$$\tau_p(X^\sharp) = \max_{vfill \in X^\sharp} ||g_i - g_j||$$

$$\oplus_p(X^\sharp) = \left\{ X^\sharp \cup \left\{ \sum_i \beta_i v_i \leq h \right\}, X^\sharp \cup \left\{ \sum_i \beta_i v_i \geq h \right\} \right\}$$

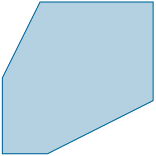


Box Polyedra Reduced Product

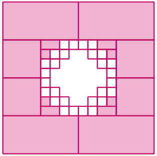
$$y \leq 2x + 10$$
$$2y \geq x - 8$$

$$x^2 + y^2 \geq 3$$

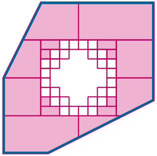
$$x, y \in [-5, 5]$$



(e) Consistent polyhedron



(f) Solving the non-linear part



(g) Intersection of the domains

Box Polyedra Reduced Product

problem	#var	#ctrs	time, AbS	time, Ibex	#sols AbS	#sols, Ibex
booth	2	2	3.026s	26.36s	19183	1143554
exnewton	2	3	0.158s	26.452s	14415	1021152
supersim	2	3	0.7s	0.008s	1	1
aljazzaf	3	2	0.008s	0.02s	42	43
bronstein	3	3	0.01s	0.004s	8	4
eqlin	3	3	0.07s	0.008s	1	1
cubic	2	2	0.007s	0.009	9	3
hs23	2	6	2.667s	2.608s	27268	74678
powell	4	4	0.007s	0.02	4	1
combustion	10	10	0.007s	0.012s	1	1

Other works

current
: Solution counting for global constraints (PhD Giovanni Lo Bianco)

future
: Solution counting in Abstract Domains, solvers which enumerate solutions in a random order

current
: Application to flow-chemistry (post-doc Daniel Cortes Borda)

future
: Application to mixed problems

Conclusion

What we (CP) gain:

- new, relational abstract domains: octagons, polyedra, BDDs...
- reduced products to combine domains in a sound way: Boxes+Polyedra, Real+Int boxes,
- new heuristics inspired from AI: elimination.

What AI gains:

- new operators on abstract domains to use on other verification problems: split for computing inductive invariants,
- new tools on the abstract domains which can be defined as constraints: size, enumeration of feasible points...

Further Research

Develop AbSolute

- improve the integer domain, add solution counting,
- generalize the reduced products mechanism (constraint allocation),

Is CP a decision procedure? Investigate the links with SMT:

- use SMT learning with abstract domains (comparable to ACDCL),
- compare the landscape analysis/heuristics to build efficient combined models,
- define CP as an MT, to retrieve the logic part of CP: back to constraint logic programming!

Apt, K. R. (1999).
The essence of constraint propagation.
Theoretical Computer Science, 221.

Benhamou, F. (1996).
Heterogeneous constraint solvings.
In Proceedings of the 5th International Conference on Algebraic and Logic Programming, pages 62–76.

Chabert, G. and Jaulin, L. (2009).
Contractor programming.
Artificial Intelligence, 173:1079–1100.

Cousot, P. and Cousot, R. (1976).
Static determination of dynamic properties of programs.
In Proceedings of the 2nd International Symposium on Programming, pages 106–130.

Cousot, P. and Cousot, R. (1977).
Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.