# An introduction to Choco

## A *java* Constraint Programming Library

G. Rochart, N. Jussien, X. Lorca
Charles Prud'homme, Hadrien Cambazard, Guillaume Richaud, Julien Menana, Arnaud Malapert

Bouygues SA, École des Mines de Nantes (LINA CNRS UMR 6241)

# Outline

1. The Choco constraint solver
   - A little bit of history
   - The design of Choco
   - General features
2. The practice of Choco
3. Choco around the World
4. The future of Choco
5. Acknowledgements

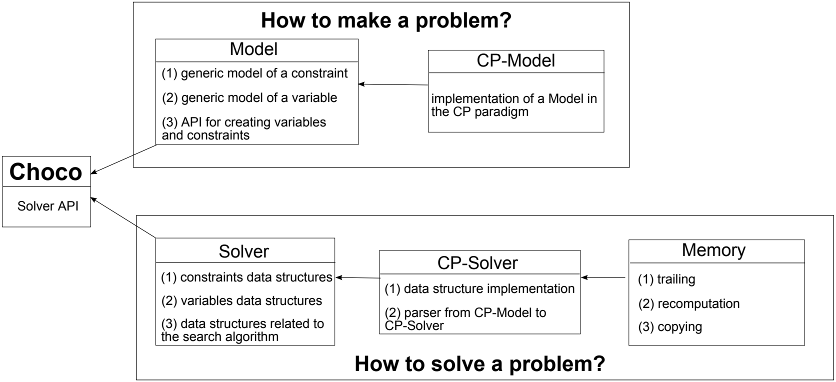
# A solver for teaching and research

- 1999 : a first CLAIRE implementation within the OCRE project
  an national initiative for an open constraint solver for both teaching and research (Nantes, Montpellier, Toulouse, Bouygues, ONERA)
- 2003 : a first Java implementation
  portability, ease of use for newcomers, etc.
- 2008 : Choco V2
  a clear separation between the model and the solving machinery ; a complete re-factoring ; a user-oriented version

# An open constraint solver

- An open system
    - a source forge project
    - BSD license for all possible usages
- A glass box
    - designed for both teaching and research
    - efficient yet readable
    - readable yet efficient

# General Schema of Choco's Architecture



How to make a problem?

Model

(1) generic model of a constraint

(2) generic model of a variable

(3) API for creating variables and constraints

CP-Model

implementation of a Model in the CP paradigm

Choco

Solver API

Solver

(1) constraints data structures

(2) variables data structures

(3) data structures related to the search algorithm

CP-Solver

(1) data structure implementation

(2) parser from CP-Model to CP-Solver

Memory

(1) trailing

(2) recomputation

(3) copying

How to solve a problem?

# Embedded Variable Types

A wide **variety of variable** paradigms :

- Integer variables : enumerated and bounded,
- Set variables : enumerated and bounded,
- Real variables,
- Composite variables : integer expression and real expression composed with operators like *plus, mult, minus, scalar, sum, power,* . . . (first-class citizen)

**Work in progress** :

- Intervals list.

# A Large Choice of Implemented Constraints

About **70 available constraints** in Choco :

- Classical **arithmetic constraints** : equal, not equal, less or equal, greater or equal,
- A large set of useful **global constraints** : AllDifferent and BoundAllDifferent, GlobalCardinality and BoundGCC, AtMostNvalue, Cumulative, Occurence, Element, . . .
- **Exclusive** constraints : Tree, . . .
- **Reified** constraints : and, or, not, implies, ifOnlyIf, . . .

# Search-related tools

User can use predefined search methods :

- Searching CSP solutions : **solve** for searching first solution and **solveAll** for searching all solutions
- Optimizing a problem by maximizing ou minimizing a variable value (**maximize** and **minimize**) with or without restart
- Some new feature in next release like solve with restarts (useful for heuristics with learning). . .

# Embedded Search Heuristics

Choco proposes a set of implemented **Search Heuristics**. Two kinds are distinguished :

- ***Variable* choice :** MinDomain, RandomIntVarSelector, StaticVarOrder, DomOverDeg, DomOverDynDeg, DomOverWDeg, DomOverFailureDeg, LexIntVarSelector, . . .
- ***Value* choice for a *Variable* :** DecreasingDomain, IncreasingDomain, MaxVal, MinVal, MidVal, RealIncreasingDomain, RandomIntValSelector, RandomSetValSelector, . . .

# Outline

1. The Choco constraint solver

2. The practice of Choco
    - A First Problem
    - The Nqueens problem
    - Custom branching
    - Good Practice

3. Choco around the World

4. The future of Choco

5. Acknowledgements

# $x + y = z$

- Two parts : the *Model* and a *Solver*.

```
Model m = new CPModel() ;
Solver s = new CPSolver() ;
```

## $x + y = z$

- Two parts : the *Model* and a *Solver*.

```
Model m = new CPModel();
Solver s = new CPSolver();
```

- *Model* = Variables + Expressions + Constraints.

```
IntegerVariable v1 = makeBoundIntVar(“v1”,1,5);
IntegerVariable v2 = makeBoundIntVar(“v2”,1,5);
IntegerVariable v3 = makeBoundIntVar(“v3”,1,5);
IntegerExpressionVariable e1 = plus(v1,v2);
Constraint c1 = eq(v3,e1);
```

# $x + y = z$

- Two parts : the *Model* and a *Solver*.

```
Model m = new CPModel() ;
Solver s = new CPSolver() ;
```

- *Model* = Variables + Expressions + Constraints.

```
IntegerVariable v1 = makeBoundIntVar(“v1”,1,5) ;
IntegerVariable v2 = makeBoundIntVar(“v2”,1,5) ;
IntegerVariable v3 = makeBoundIntVar(“v3”,1,5) ;
IntegerExpressionVariable e1 = plus(v1,v2) ;
Constraint c1 = eq(v3,e1) ;
```

- Linking *Variables* and *Constraints*

```
m.addVariable(v1,v2,v3) ;
m.addConstraint(c1) ;
```

# $x + y = z$

- Two parts : the *Model* and a *Solver*.

```
Model m = new CPModel() ;
Solver s = new CPSolver() ;
```

- *Model* = Variables + Expressions + Constraints.

```
IntegerVariable v1 = makeBoundIntVar(“v1”,1,5) ;
IntegerVariable v2 = makeBoundIntVar(“v2”,1,5) ;
IntegerVariable v3 = makeBoundIntVar(“v3”,1,5) ;
IntegerExpressionVariable e1 = plus(v1,v2) ;
Constraint c1 = eq(v3,e1) ;
```

- Linking *Variables* and *Constraints*

```
m.addVariable(v1,v2,v3) ;
m.addConstraint(c1) ;
```

- Feeding the *Model* to the *Solver*

```
s.read(m) ;
s.solve() ;
```

# Problem and Variables declaration

A well-known problem : the nqueens problem

- *Model* and *Solver* declarations

```
Model m = new CPModel();
Solver s = new CPSolver();
```

- A model using three kinds of *Variables*

```
IntegerVariable[] q = new IntegerVariable[n];
IntegerVariable[] d1 = new IntegerVariable[n];
IntegerVariable[] d2 = new IntegerVariable[n];
∀i ∈ [1, n], q[i] = makeEnumIntVar(“q”+i,1,n);
∀i ∈ [1, 2 ∗ n], d1[i] = makeEnumIntVar(“d1-”+i,1,2 ∗ n);
∀i ∈ [1, 2 ∗ n], d2[i] = makeEnumIntVar(“d2-”+i,−n,n);
```

# Constraint declaration

- *Model* and *Solver* declarations
- A model using three kinds of *Variables*
- *Equality (channeling) constraints* are defined :

```
Constraint[] equalities = new Constraint[2*n];
int i,j = 0;
while (i < n) {
         equalities[j] = eq(d1[i],plus(q[i],i));
        equalities[j+1] = eq(d2[i],minus(q[i],i));
         i++; j+=2;
}
```

- *AllDifferent constraints* are defined

```
Constraint[] allDiff = new Constraint[3];
allDiff[0] = allDifferent(q);
allDiff[1] = allDifferent(d1);
allDiff[2] = allDifferent(d2);
```

# Relating *Variables* and *Constraints*

- *Model* and *Solver* declarations
- A model using three kinds of *Variables*
- *Constraint* declaration.
- Relating *Variables* and *Constraints* in the Model.

```
m.addVariable(q);
m.addVariable(d1);
m.addVariable(d2);
m.addConstraint(equalities);
m.addConstraint(allDiff);
```

# Search Heuristic and Resolution

- *Model* and *Solver* declarations
- A model using three kinds of *Variables*
- *Constraint* declaration.
- Relating *Variables* and *Constraints* in the Model.
- Feeding the *Model* to the *Solver* :

```
s.read(m);
```

- A search heuristic : choosing a *Variable* whose domain has a minimum size...

```
s.setVarIntSelector(new MinDomain(s,s.getVar(q)));
```

- Resolution begins...

```
s.solve();
```

# Customizing the search

Customizing the search can be done by custom branching (for instance)

- Creating an AbstractargeIntBranching class

```
 s.attachGoal(new DichotomicBranching(s.getVar(q)));
...
DichotomicBranching extends AbstractLargeIntBranching {
```

- Implementing some functions like

```
public int getFirstBranch(Object x) { return 1; }
public int getNextBranch(...) { return i+1; }
public boolean finishedBranching(..) { return i == 2; }

public void goDownBranch(..) ... {
      ...
      int middle = (var.getSup() + var.getInf()) / 2;
      if (i == 1) var.setSup(middle);
      else var.setInf(middle + 1);
```

# In a few words

Keep your mind, in Choco :

- Modeling and search are separated through *Model* and *Solver*
- *Variables* and *Constraints* are separated from *Model* and *Solver*

The Choco philosophy :

- an open, user-oriented constraint solver
- a clear separation between model and solver
- a living solver

# Outline

1. The Choco constraint solver
2. The practice of Choco
3. Choco around the World
4. The future of Choco
5. Acknowledgements

# Academic usage of Choco (as far as we know)

- in France :
  - Universities : Nantes, Montpellier, Rennes, Toulouse, Clermont-Ferrand
  - Engineering schools : ENSTA, Ecole des Mines de Nancy, Ecole des Mines de Nantes
- in Europe :
  - UK : University of Glasgow
  - Ireland : University of Cork

# Industry usage of Choco (as far as we know)

- Big companies : Bouygues, Amadeus, Dassault
- Research agencies : ONERA, NASA
- Software and Integrators : Kls-Optim, alfaplan GmbH

# Outline

1. The Choco constraint solver
2. The practice of Choco
3. Choco around the World
4. The future of Choco
5. Acknowledgements

# Choco diffusion

- A *ChocoDay* alongside the French CP days (in June)
- For the first time : contestant within the CP solver competition
- A dynamic website : downloads, teaching material, demo material, etc.

# Current hot topics inside choco

- integrating explanations (PaLM V2)
- implementing automatic reformulation techniques
- global constraint automatic and generic generation
- integration with LP !

# Outline

1. The Choco constraint solver
2. The practice of Choco
3. Choco around the World
4. The future of Choco
5. Acknowledgements

# The Choco team

- the founding fathers

  François Laburthe (Amadeus), Narendra Jussien (EMN, LINA)
- the core team

  Guillaume Rochart (Bouygues), Hadrien Cambazard (4C)
- the new generation

  Charles Prud'homme (EMN – project management), Xavier Lorca (EMN – teaching, training), Guillaume Richaud (EMN – dev.), Julien Menana (EMN – dev.), Arnaud Malapert (EMN – dev.)
- the funding fathers

  École des Mines de Nantes, Bouygues SA, Amadeus SA