# 制約充足問題の SAT 符号化

田村直之　番原睦則　宋剛秀

神戸大学

論理と推論の理論，実装，応用に関する合同セミナー
2013 年 7 月 24 日

# 目次

# SAT と SAT ソルバー

# SAT

SAT (Boolean satisfiability testing) **は，与えられた命題論理式を真にする値割当てが存在するか否かを判定する問題である．**

- SAT **は，理論上も実際上も計算機科学の中心的課題である．**
- SAT **は** NP-**完全であることが最初に証明された問題である** [Cook 1971]**．**
- MiniSat **等，非常に効率の良い** SAT **ソルバーが実装されている．**
- SAT **ソルバーをエンジンとした** SAT **型システムが多くの分野で利用されるようになってきた．**

# SAT 問題 (SAT Instances)

**具体的な** SAT **の問題は，連言標準形** (CNF) **で与えられる．**

## CNF 式

- CNF **式は，複数の節の連言である．**
- **節** (clause) **は複数のリテラルの選言である．**
- **リテラル** (literal) **は，命題変数かあるいはその否定である．**

**標準的フォーマットとしては** DIMACS CNF **が用いられる．**

```
p cnf 3 4       ; Number of variables and clauses
1 2 3 0         ; p1 ∨ p2 ∨ p3
-1 -2 0         ; ¬p1 ∨ ¬p2
-1 -3 0         ; ¬p1 ∨ ¬p3
-2 -3 0         ; ¬p2 ∨ ¬p3
```

# SAT ソルバー (SAT Solvers)

- SAT **ソルバー**は，与えられた SAT 問題が充足可能 (SAT) か充足不能 (UNSAT) かを判定するプログラムである．
- 通常，充足可能であればその値割当てを解として出力する．

- 系統的 SAT ソルバーは，SAT あるいは UNSAT を判定する．
  - ほとんどは DPLL アルゴリズムを用いている．
- 確率的 SAT ソルバーは，SAT のみを判定する (UNSAT は判定できない)．
  - ローカルサーチアルゴリズムが用いられる．

# 近代的 SAT ソルバー (Modern SAT Solvers)

- **近代的** SAT **ソルバーでは，**DPLL **に以下の技術が導入され大幅な性能向上が実現されている．**
  - **CDCL** (Conflict Driven Clause Learning) [Silva 1996]
  - **非時間順バックトラック法** [Silva 1996]
  - **ランダムリスタート** [Gomes 1998]
  - **監視リテラル** [Moskewicz & Zhang 2001]
  - **変数選択ヒューリスティック** [Moskewicz & Zhang 2001]
- Chaff **と** zChaff **は十倍から百倍の高速化を実現した．**
- 2002 **年以降** SAT **競技会が開催され，**SAT **ソルバーの実装技術の向上に貢献している．**
- MiniSat **は** C++**で約** 2000 **行というコンパクトなサイズにもかかわらず，**2005 **年の** SAT **競技会で優れた性能を示した．**
- **近代的** SAT **ソルバーは** $10^6$ **個の命題変数** $10^7$ **個の節を含む** SAT **問題も取り扱うことができる．**

# SAT 競技会 (SAT Competitions)

| | Gold | Silver | Bronze |
|---|---|---|---|
| Application | | | |
| SAT | Lingeling | **ZENN** | satUZK |
| Certified UNSAT | glucose | **glueminisat** | Riss3g |
| SAT+UNSAT | Lingeling | Lingeling | **ZENN** |
| Hard-combinatorial | | | |
| SAT | glucose | gluebit_clasp | BreakIDGlucose |
| Certified UNSAT | Riss3g | glucose | forl |
| SAT+UNSAT | BreakIDGlucose | gluebit_clasp | glucose |
| Random | | | |
| SAT | probSAT | sattime | Ncca+ |
| Certified UNSAT | **dk-SAT11** | march_br | |
| SAT+UNSAT | CSHCrandMC | MIPSat | march_vflip |

- 2013 **年** SAT **競技会** Core Solvers, Sequential **トラックの結果**
- 2011 **年と比べて，いくらか傾向が変わったようだ**

# SAT 型システム (SAT-based Systems)

- **プランニング** (SATPLAN, Blackbox) [Kautz & Selman 1992]
- **自動テストパターン生成** [Larrabee 1992]
- **ジョブショップスケジューリング** [Crawford & Baker 1994]
- **ソフトウェア検証** (Alloy)
- **有界モデル検査** [Biere 1999]
- **ソフトウェアパッケージの依存解析** (SATURN)
  - Sat4j **は** Eclipse **で利用されている．**
- **書換えシステム** (AProVE, Jambox)
- **解集合プログラミング** (clasp, Cmodels-2)
- **一階論理定理証明，モデル発見** (iProver, Darwin, Paradox)
- **制約充足問題** (Sugar) [Tamura et al. 2006]
- **プログラミング言語のコンパイラ**
  - Scala 2.11 **コンパイラで** Sat4j **が利用される**

# 制約充足問題の SAT 符号化

# Finite linear CSP

- **整数変数** (**有限ドメイン**)
  - $\ell(x)$ : $x$ **の下限**
  - $u(x)$ : $x$ **の上限**
- **命題変数**
- **算術演算**: $+$, $-$, **定数倍**, etc.
- **比較演算**: $=$, $\neq$, $\geq$, $>$, $\leq$, $<$
- **論理演算**: $\neg$, $\wedge$, $\vee$, $\Rightarrow$

- **一般性を失うことなしに算術演算と比較を** $\sum a_i x_i \leq c$ **の形に制限できる．ここで** $x_i$ **は整数変数で** $a_i$ **と** $c$ **は整数定数である．**

# SAT 符号化 (SAT encodings)

CSP の SAT 符号化については，種々の方法が提案されている．

- **直接符号化** (Direct encoding) **は，最も広く用いられている方法である** [de Kleer 1989]．
- **順序符号化** (Order encoding) **は，多くの問題に対して優れた性能を示す新しい方法である** [Tamura et al. 2006]．
  - **順序符号化はジョブショップ・スケジューリング問題に対して最初に用いられた** [Crawford & Baker 1994]．
  - **応用例**: **ショップ・スケジューリング，2 次元パッキング，テストケース生成，時間割問題，クリーク幅，パズルなど**
- **他の符号化**:
  - **多値符号化** (Multivalued encoding) [Selman 1992]
  - **支持符号化** (Support encoding) [Kasif 1990]
  - **対数符号化** (Log encoding) [Iwama 1994]
  - **対数支持符号化** (Log-support encoding) [Gavanelli 2007]
  - **コンパクト順序符号化** (Compact order encloding) [Tanjo et al. 2010]

# 直接符号化 (Direct encoding)

**直接符号化では，各整数変数 $x$ とそのドメインの各値 $i$ に対して，$x = i$ を表す命題変数 $p(x = i)$ を用いる** [de Kleer 1989]**.**

**各整数変数 $x$ に対して用いる命題変数**

$$p(x = i) \qquad (\ell(x) \leq i \leq u(x))$$

**たとえば整数変数 $x \in \{2, 3, 4, 5, 6\}$ に対しては，以下の 5 つの命題変数を用いる.**

**$x \in \{2, 3, 4, 5, 6\}$ に対する 5 つの命題変数**

$$p(x = 2) \quad p(x = 3) \quad p(x = 4) \quad p(x = 5) \quad p(x = 6)$$

# 直接符号化 (続き)

**命題変数 $p(x = i)$ が $x = i$ の時かつその時に限って真となるように，以下の** at-least-one **節と** at-most-one **節を追加する．**

## 各整数変数 $x$ に対して追加する節

$$p(x = \ell(x)) \vee \cdots \vee p(x = u(x))$$
$$\neg p(x = i) \vee \neg p(x = j) \qquad (\ell(x) \leq i < j \leq u(x))$$

**たとえば $x \in \{2, 3, 4, 5, 6\}$ に対しては，以下の 11 節を追加する．**

## $x \in \{2, 3, 4, 5, 6\}$ に対して追加する 11 節

$$p(x = 2) \vee p(x = 3) \vee p(x = 4) \vee p(x = 5) \vee p(x = 6)$$
$$\neg p(x = 2) \vee \neg p(x = 3) \qquad \neg p(x = 2) \vee \neg p(x = 4) \qquad \neg p(x = 2) \vee \neg p(x = 5)$$
$$\neg p(x = 2) \vee \neg p(x = 6) \qquad \neg p(x = 3) \vee \neg p(x = 4) \qquad \neg p(x = 3) \vee \neg p(x = 5)$$
$$\neg p(x = 3) \vee \neg p(x = 6) \qquad \neg p(x = 4) \vee \neg p(x = 5)$$
$$\neg p(x = 4) \vee \neg p(x = 6) \qquad \neg p(x = 5) \vee \neg p(x = 6)$$

# 直接符号化 (続き)

**制約は違反点** (conflict points) **を列挙することで符号化できる．**

## 制約節

$x_1 = i_1, \ldots, x_k = i_k$ **が制約に違反する時，以下の節を追加する．**

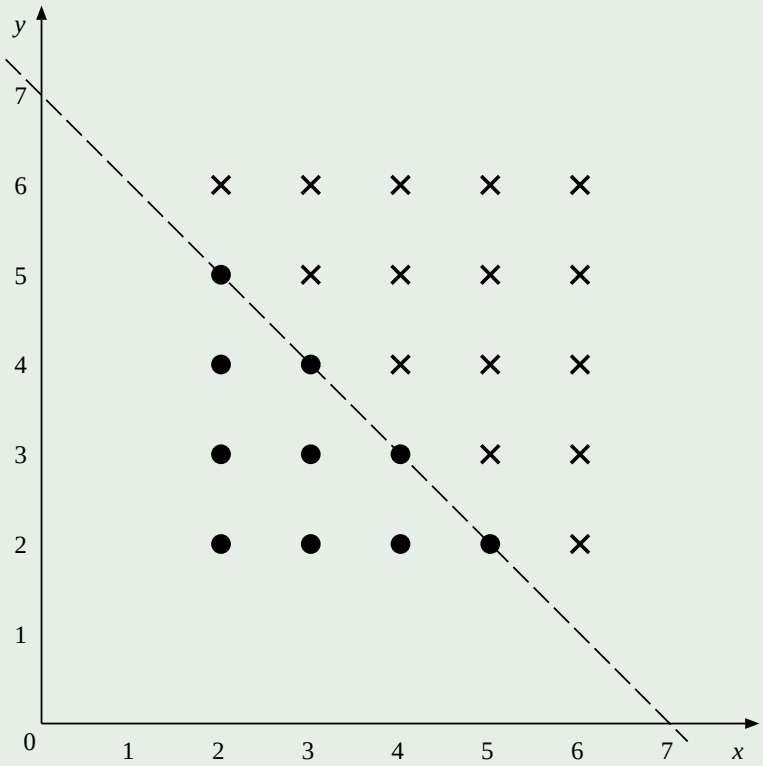
$$\neg p(x_1 = i_1) \vee \cdots \vee \neg p(x_k = i_k)$$

# 直接符号化 (続き)

**制約 $x + y \leq 7$ は，違反点 (図中の $\times$ 点) を列挙することで以下の 15 節に符号化される．**

## $x + y \leq 7$ の直接符号化

$$
\begin{aligned}
&\neg p(x = 2) \vee \neg p(y = 6)\\
&\neg p(x = 3) \vee \neg p(y = 5)\\
&\neg p(x = 3) \vee \neg p(y = 6)\\
&\neg p(x = 4) \vee \neg p(y = 4)\\
&\neg p(x = 4) \vee \neg p(y = 5)\\
&\neg p(x = 4) \vee \neg p(y = 6)\\
&\neg p(x = 5) \vee \neg p(y = 3)\\
&\neg p(x = 5) \vee \neg p(y = 4)\\
&\neg p(x = 5) \vee \neg p(y = 5)\\
&\neg p(x = 5) \vee \neg p(y = 6)\\
&\neg p(x = 6) \vee \neg p(y = 2)\\
&\neg p(x = 6) \vee \neg p(y = 3)\\
&\neg p(x = 6) \vee \neg p(y = 4)\\
&\neg p(x = 6) \vee \neg p(y = 5)\\
&\neg p(x = 6) \vee \neg p(y = 6)
\end{aligned}
$$



# 順序符号化 (Order encoding)

**順序符号化では，各整数変数 $x$ とそのドメインの各値 $i$ に対して，$x \leq i$ を表す命題変数 $p(x \leq i)$ を用いる** [Tamura et al. 2006]**.**

## 各整数変数 $x$ に対して用いる命題変数

$$p(x \leq i) \qquad (\ell(x) \leq i < u(x))$$

**たとえば整数変数 $x \in \{2, 3, 4, 5, 6\}$ に対しては，以下の 4 つの命題変数を用いる.**

## $x \in \{2, 3, 4, 5, 6\}$ に対する 4 つの命題変数

$$p(x \leq 2) \quad p(x \leq 3) \quad p(x \leq 4) \quad p(x \leq 5)$$

- **$x \leq 6$ は常に真のため，命題変数 $p(x \leq 6)$ は不要である.**

# 順序符号化 (続き)

**命題変数 $p(x \leq i)$ が $x \leq i$ の時かつその時に限って真となるように，以下の節を追加する．**

## 各整数変数 $x$ に対して追加する節

$$\neg p(x \leq i-1) \vee p(x \leq i) \quad (\ell(x) < i < u(x))$$

**たとえば $x \in \{2,3,4,5,6\}$ に対しては，以下の 3 節を追加する．**

## $x \in \{2,3,4,5,6\}$ に対して追加する 3 節

$$\neg p(x \leq 2) \vee p(x \leq 3)$$
$$\neg p(x \leq 3) \vee p(x \leq 4)$$
$$\neg p(x \leq 4) \vee p(x \leq 5)$$

# 順序符号化 (続き)

**追加された節を充足する可能な値割当ては以下の通りである.**

$$\neg p(x \leq 2) \vee p(x \leq 3)$$
$$\neg p(x \leq 3) \vee p(x \leq 4)$$
$$\neg p(x \leq 4) \vee p(x \leq 5)$$

## 充足する値割当て

| $p(x \leq 2)$ | $p(x \leq 3)$ | $p(x \leq 4)$ | $p(x \leq 5)$ | **解釈** |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | $x = 2$ |
| 0 | 1 | 1 | 1 | $x = 3$ |
| 0 | 0 | 1 | 1 | $x = 4$ |
| 0 | 0 | 0 | 1 | $x = 5$ |
| 0 | 0 | 0 | 0 | $x = 6$ |

# 順序符号化 (続き)

## 充足する部分値割当て

| $p(x \leq 2)$ | $p(x \leq 3)$ | $p(x \leq 4)$ | $p(x \leq 5)$ | 解釈 |
|---|---|---|---|---|
| – | – | – | – | $x = 2..6$ |
| – | – | – | 1 | $x = 2..5$ |
| – | – | 1 | 1 | $x = 2..4$ |
| – | 1 | 1 | 1 | $x = 2..3$ |
| 0 | – | – | – | $x = 3..6$ |
| 0 | 0 | – | – | $x = 4..6$ |
| 0 | 0 | 0 | – | $x = 5..6$ |
| 0 | – | – | 1 | $x = 3..5$ |
| 0 | – | 1 | 1 | $x = 3..4$ |
| 0 | 0 | – | 1 | $x = 4..5$ |

“–” は未定義を表す．

- 命題変数に対する部分値割当ては，整数変数の範囲を表している．

# 順序符号化 (続き)

**制約は，違反点ではなく違反領域** (conflict region) **を列挙することで符号化できる．**

## 制約節

- **範囲** $i_1 < x_1 \leq j_1, \ldots, i_k < x_k \leq j_k$ **中のすべての点** $(x_1, \ldots, x_k)$ **が制約に違反する時，以下の節を追加する．**

$$p(x_1 \leq i_1) \vee \neg p(x_1 \leq j_1) \vee \cdots \vee p(x_k \leq i_k) \vee \neg p(x_k \leq j_k)$$

# 順序符号化 (続き)

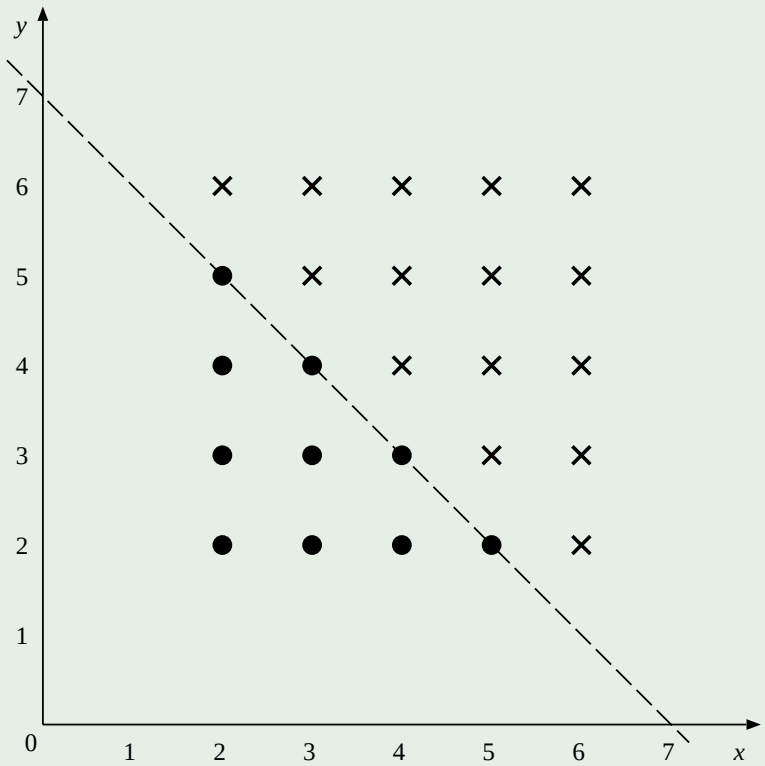
**具体的には，以下の関係を利用し再帰的に符号化する．**

$$\sum_{i=1}^{n} a_i x_i \leq c \Longleftrightarrow$$

$$\begin{cases} (x_1 \leq \lfloor c/a_1 \rfloor) & (n = 1, a_1 > 0) \\ \neg(x_1 \leq \lceil c/a_1 \rceil + 1) & (n = 1, a_1 < 0) \\ \bigwedge_{d \in Dom(x_1)} \left( (x_1 \leq d-1) \vee \sum_{i=2}^{n} a_i x_i \leq c - a_1 d \right) & (n \geq 2, a_1 > 0) \\ \bigwedge_{d \in Dom(x_1)} \left( \neg(x_1 \leq d) \vee \sum_{i=2}^{n} a_i x_i \leq c - a_1 d \right) & (n \geq 2, a_1 < 0) \end{cases}$$

# 順序符号化 (続き)

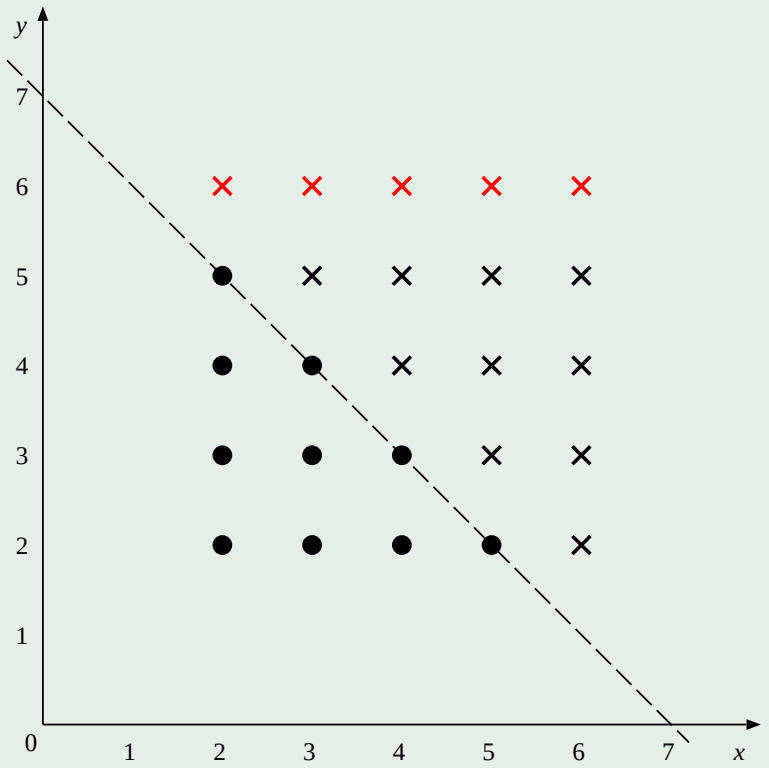
## $x + y \leq 7$ の順序符号化



# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

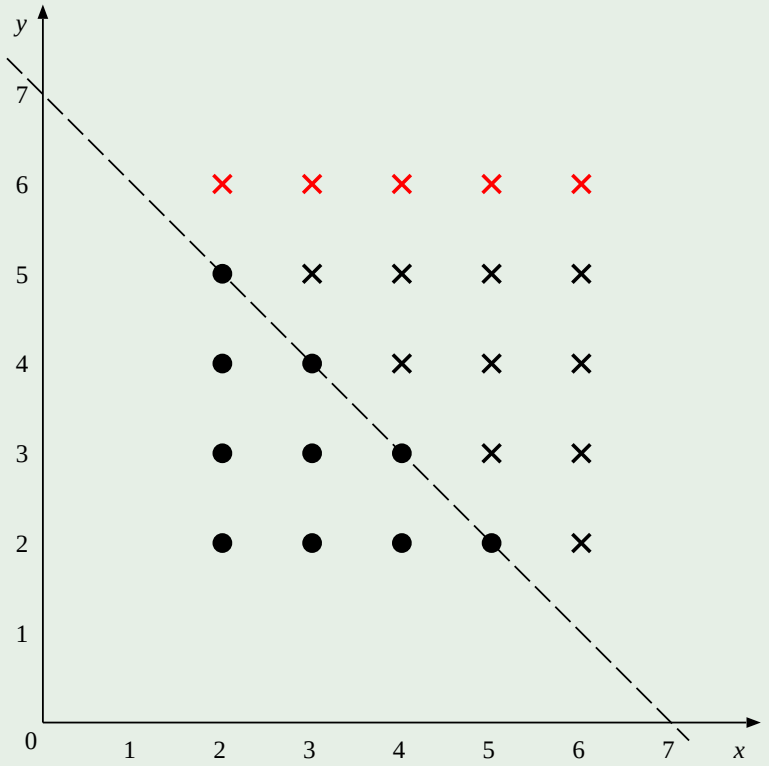
$\neg(y \geq 6)$



# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

$p(y \leq 5)$

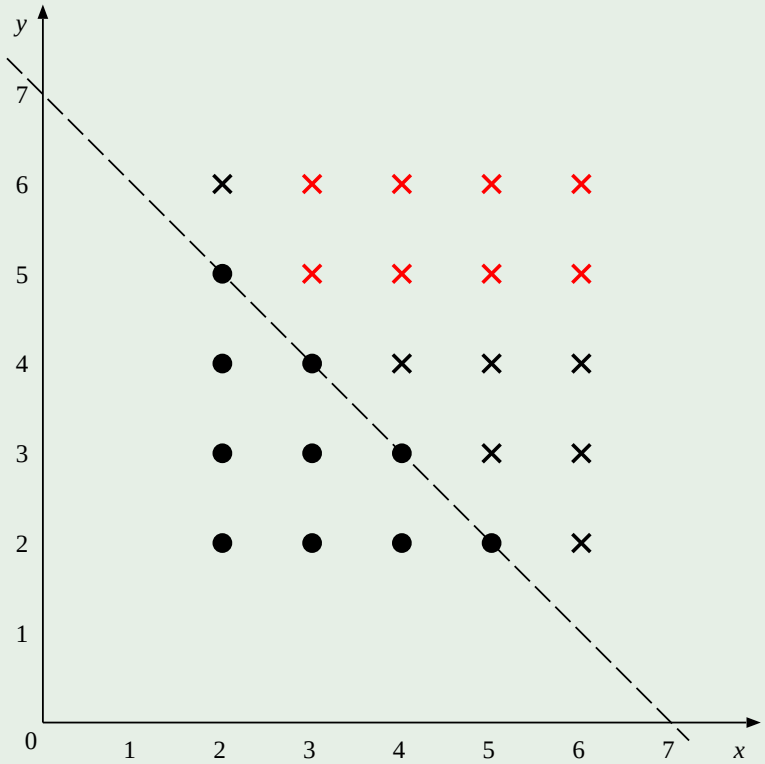


# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

$p(y \leq 5)$

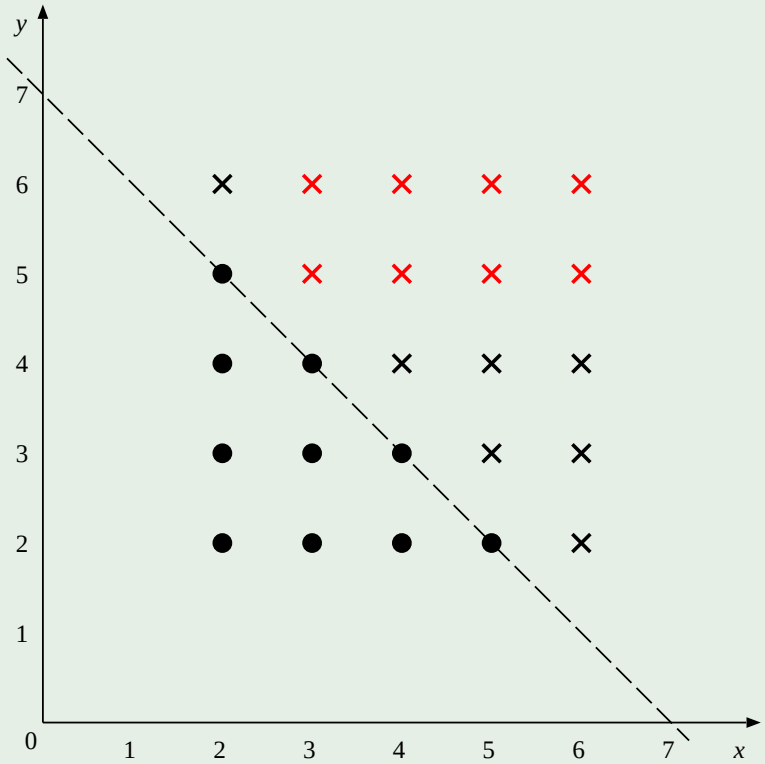
$\neg(x \geq 3 \wedge y \geq 5)$



# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

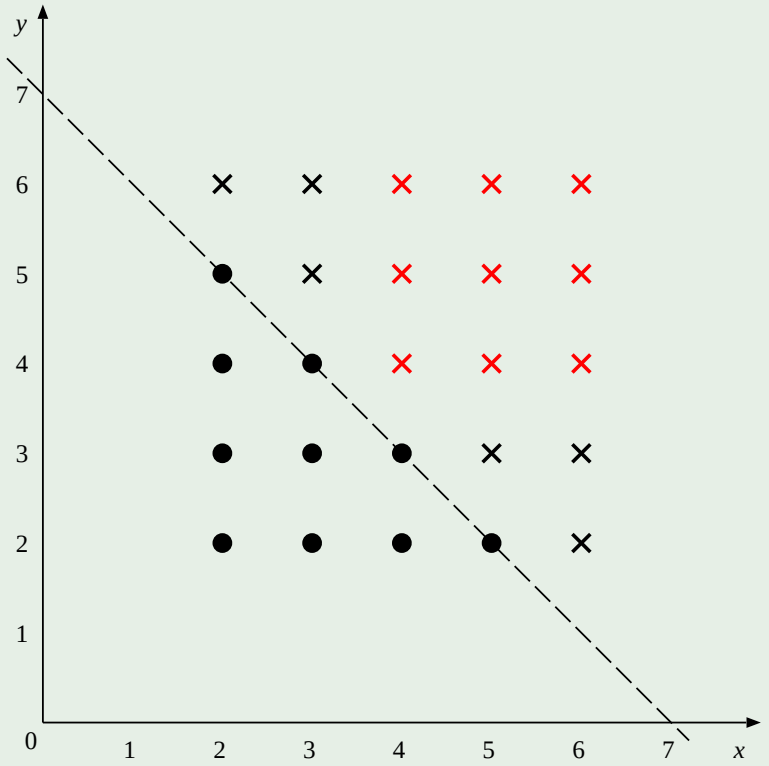
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$



# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

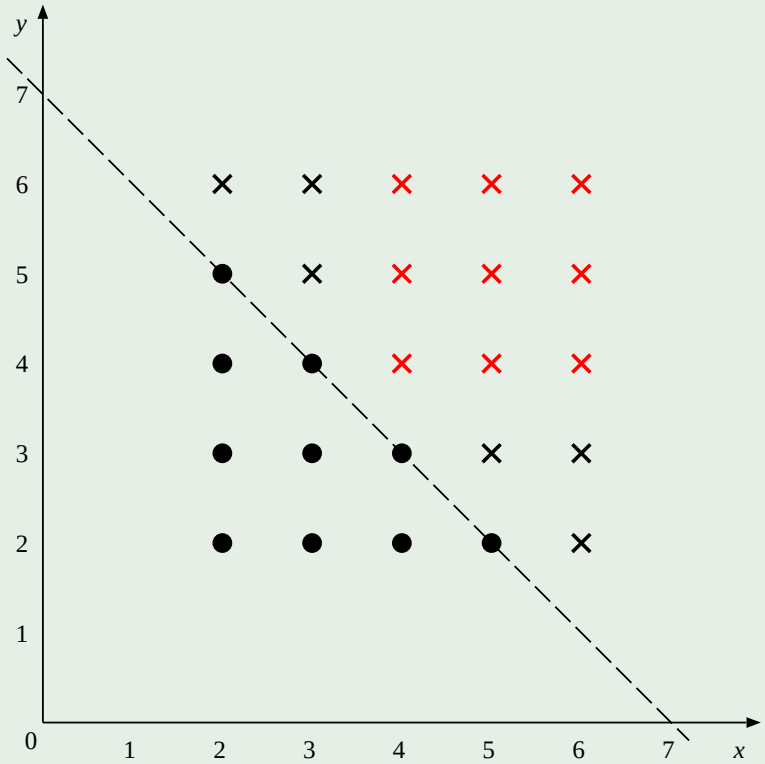
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$
$\neg(x \geq 4 \wedge y \geq 4)$



# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

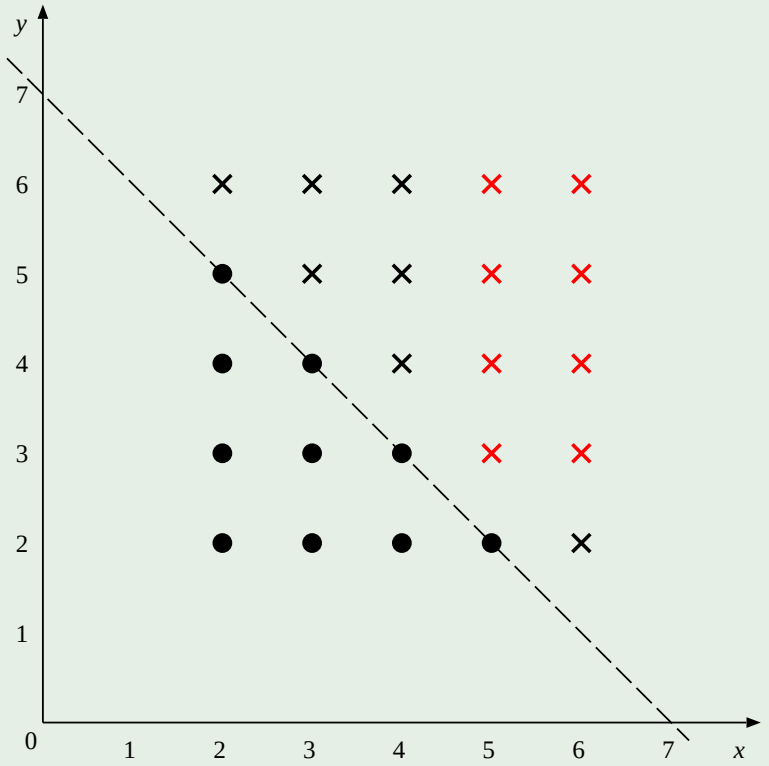
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$
$p(x \leq 3) \vee p(y \leq 3)$



# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

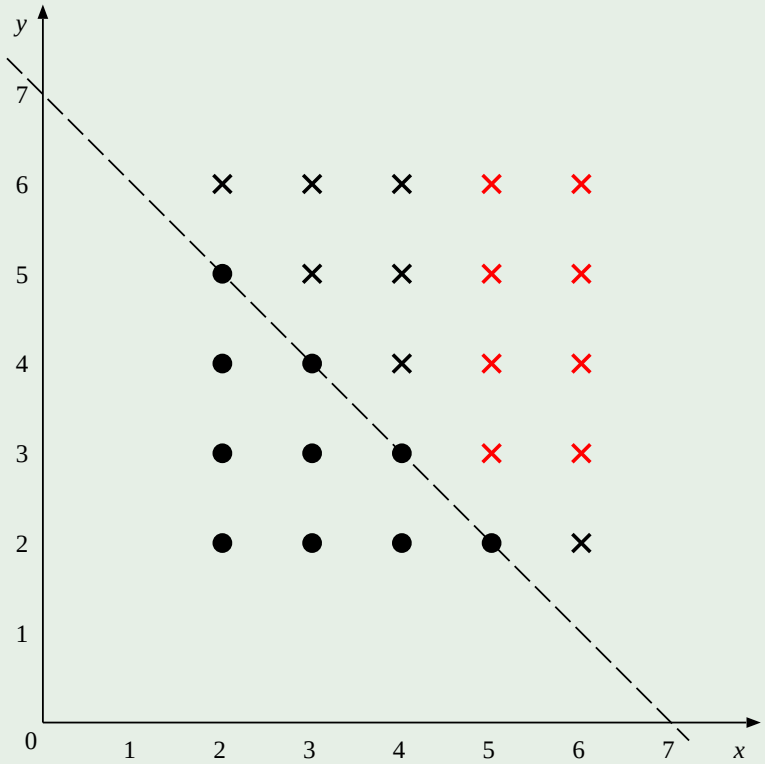
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$
$p(x \leq 3) \vee p(y \leq 3)$
$\neg(x \geq 5 \wedge y \geq 3)$



# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

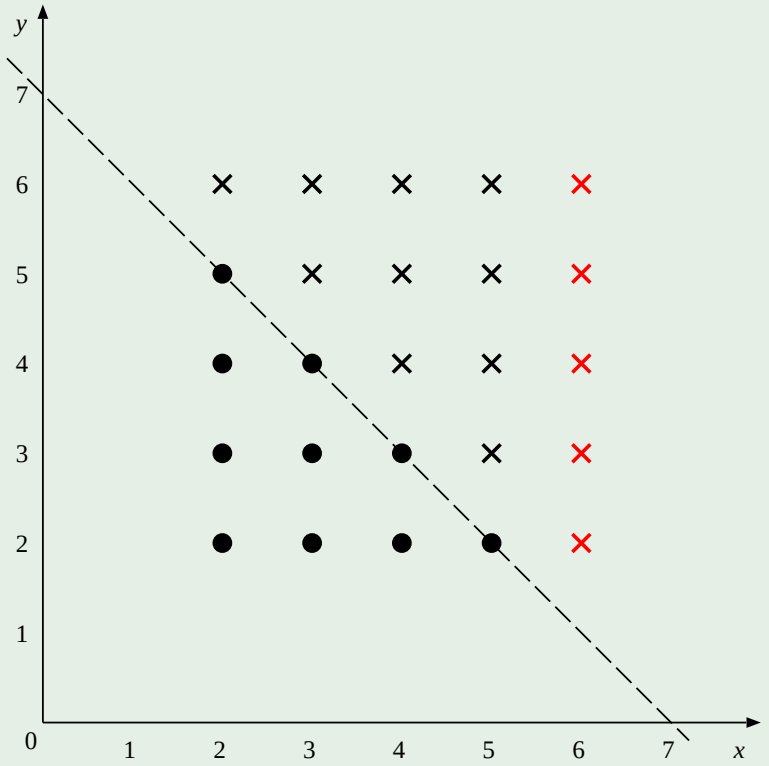
$$p(y \leq 5)$$
$$p(x \leq 2) \vee p(y \leq 4)$$
$$p(x \leq 3) \vee p(y \leq 3)$$
$$p(x \leq 4) \vee p(y \leq 2)$$



# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

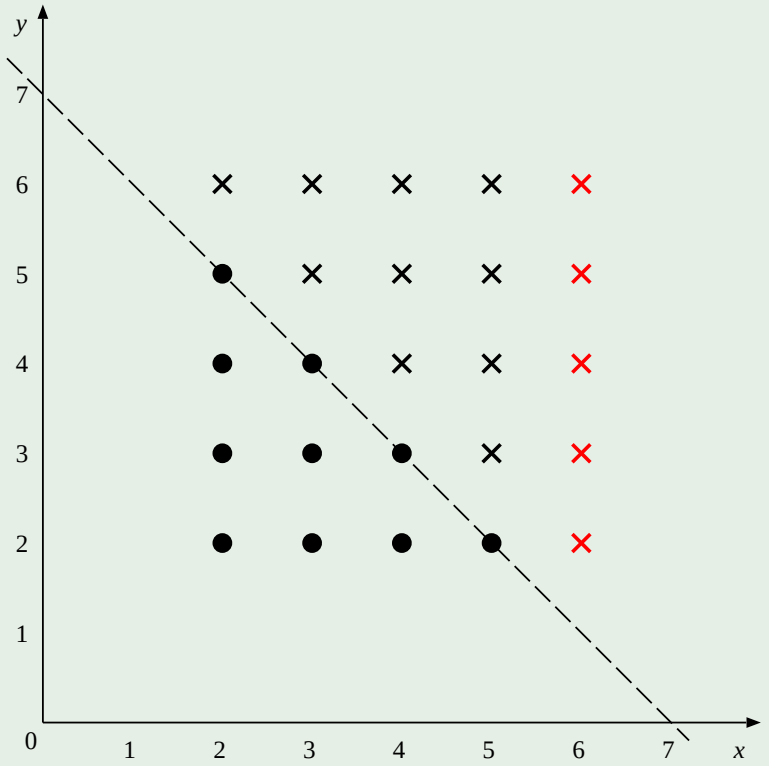
$p(y \leq 5)$
$p(x \leq 2) \vee p(y \leq 4)$
$p(x \leq 3) \vee p(y \leq 3)$
$p(x \leq 4) \vee p(y \leq 2)$
$\neg(x \geq 6)$



# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

$$
\begin{aligned}
& p(y \leq 5) \\
& p(x \leq 2) \vee p(y \leq 4) \\
& p(x \leq 3) \vee p(y \leq 3) \\
& p(x \leq 4) \vee p(y \leq 2) \\
& p(x \leq 5)
\end{aligned}
$$



# 順序符号化 (続き)

## $x + y \leq 7$ の順序符号化

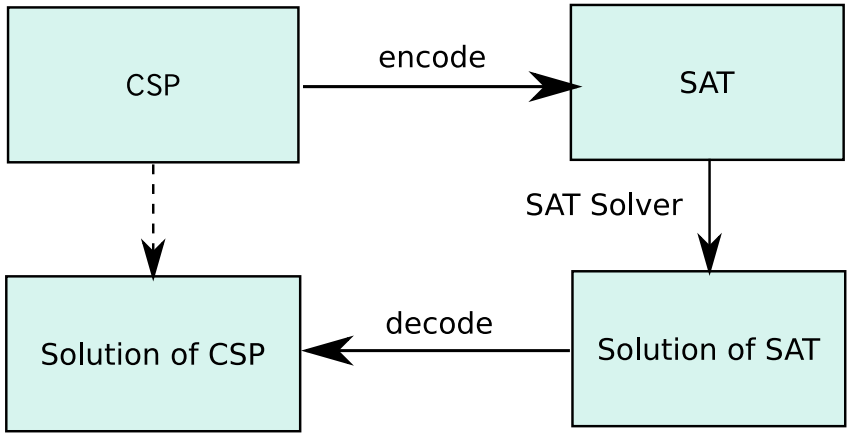
$$
\begin{array}{ll}
C_1: & p(y \leq 5) \\
C_2: & p(x \leq 2) \vee p(y \leq 4) \\
C_3: & p(x \leq 3) \vee p(y \leq 3) \\
C_4: & p(x \leq 4) \vee p(y \leq 2) \\
C_5: & p(x \leq 5)
\end{array}
$$

- $p(x \leq 3)$ が偽 (すなわち $x \geq 4$) となれば，$C_3$ に対する**単位伝播** (unit propagation) により $p(y \leq 3)$ が真 (すなわち $y \leq 3$) となる．
- これは制約ソルバーにおける**範囲伝播** (bound propagation) に対応している．
- 順序符号化は tractable CSP を tractable SAT に符号化する既存で唯一の方法である [Petke and Jeavons 2011]．

# SAT 型制約ソルバー

- Sugar
- Copris

# Sugar: SAT 型制約ソルバー



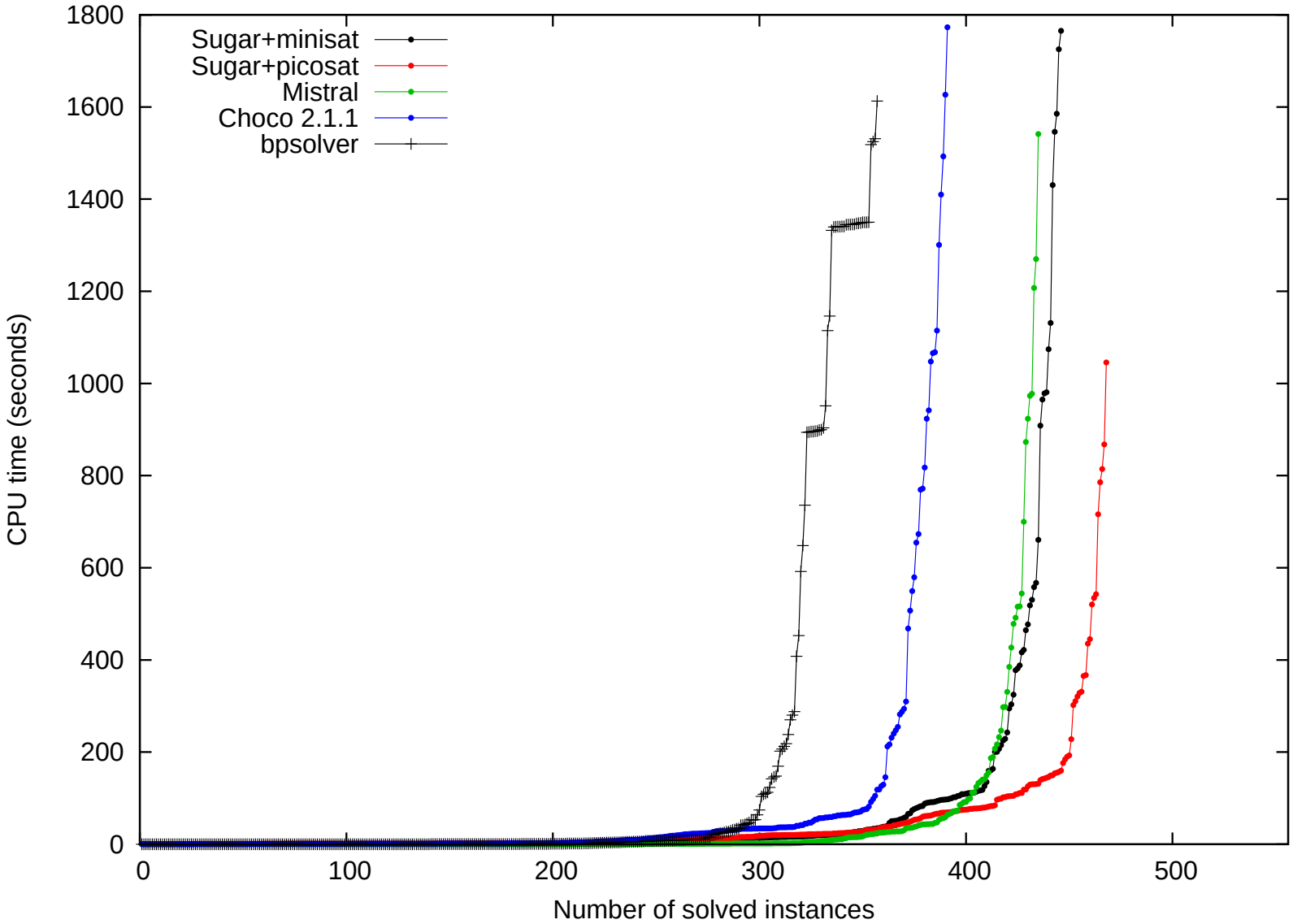

- **Sugar は順序符号化を用いた SAT 型の制約ソルバーである．**
- **2008 年，2009 年 CSP ソルバー競技会のグローバル制約部門で優勝，2008 年 Max-CSP ソルバー競技会の 3 部門で優勝．**
- **様々な問題で良い性能を示している (ショップスケジューリング，2 次元パッキング，テストケース生成など)．**

# CSC 2009 での結果

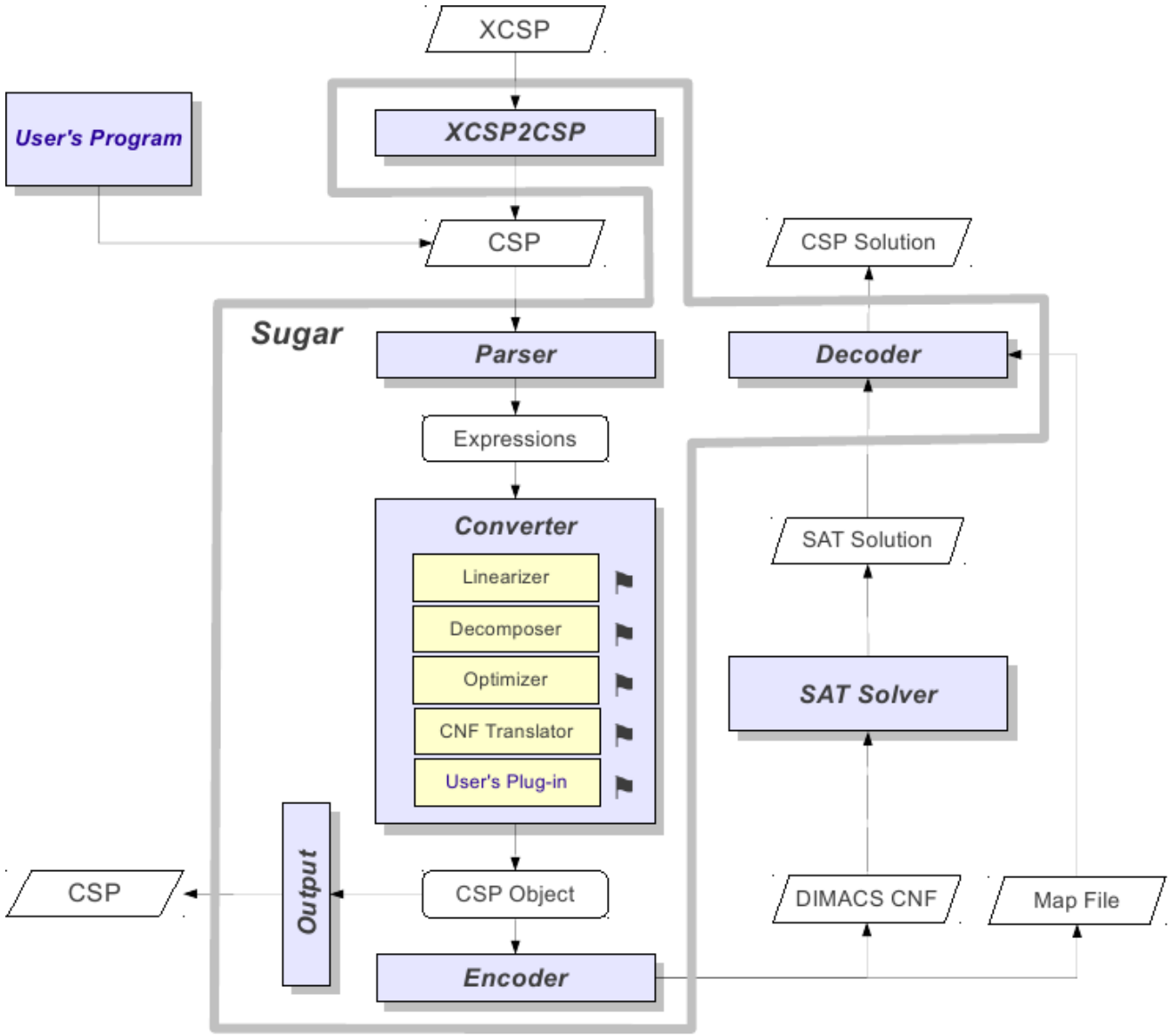
| Series | | Sugar+m | Sugar+p | Mistral | Choco | bpsolver |
|---|---|---|---|---|---|---|
| BIBD | ( 83) | 76 | 77 | 76 | 58 | 35 |
| Costas Array | ( 11) | 8 | 8 | 9 | 9 | 9 |
| Latin Square | ( 10) | 10 | 9 | 5 | 5 | 5 |
| Magic Square | ( 18) | 8 | 8 | 13 | 15 | 11 |
| NengFa | ( 3) | 3 | 3 | 3 | 3 | 3 |
| Orthogonal Latin Square | ( 9) | 3 | 3 | 3 | 2 | 3 |
| Perfect Square Packing | ( 74) | 54 | 53 | 40 | 47 | 36 |
| Pigeons | ( 19) | 19 | 19 | 19 | 19 | 19 |
| Quasigroup Existence | ( 35) | 30 | 29 | 29 | 28 | 30 |
| Pseudo-Boolean | (100) | 68 | 75 | 59 | 53 | 70 |
| BQWH | ( 20) | 20 | 20 | 20 | 20 | 20 |
| Cumulative Job-Shop | ( 10) | 4 | 4 | 2 | 1 | 0 |
| RCPSP | ( 78) | 78 | 78 | 78 | 77 | 75 |
| Cabinet | ( 40) | 40 | 40 | 40 | 40 | 40 |
| Timetabling | ( 46) | 25 | 42 | 39 | 14 | 1 |
| Total | (556) | 446 | 468 | 435 | 391 | 357 |

- **グローバル制約 3 部門での解けた問題数**
- Sugar+m : Sugar with MiniSat 2.0 backend
- Sugar+p : Sugar with PicoSAT 535 backend

# CSC 2009 での結果



# Sugar の構成


XCSP
User's Program
XCSP2CSP
CSP
CSP Solution
Sugar
Parser
Decoder
Expressions
Converter
SAT Solution
Linearizer
Decomposer
Optimizer
SAT Solver
CNF Translator
User's Plug-in
Output
CSP
CSP Object
DIMACS CNF
Map File
Encoder

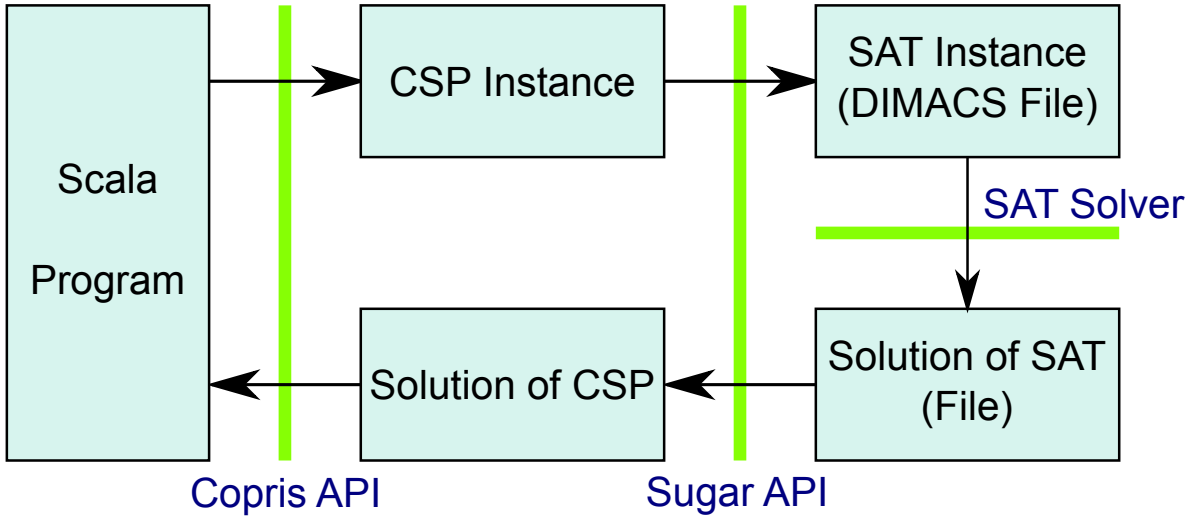
# Copris (Constraint Programming in Scala)

Copris は Scala 上で実現されている制約プログラミング用の DSL (Domain-Specific Language)

## Features

- 制約モデリングを容易にする機能を提供
- Sugar を利用することで高性能の制約解消を実現
- Sat4j を利用すれば，すべてが JVM 上で動作

# Copris の構成




1. **ユーザの** Scala **プログラムから** Copris API **を通じて** CSP **が定義される．**
2. Scala **プログラムから** Copris **ソルバーが呼び出される．**
   - **定義された** CSP **が** Sugar **を用いて** SAT **問題ファイルに符号化される．**
   - Copris **から** SAT **ソルバー (デフォールトは** Sat4j**) が呼び出される．**MiniSat, Glucose, GlueMiniSat **等の他の** SAT **ソルバーを利用することも可能．**
   - Sugar **により，**SAT **問題の解が** CSP **の解に変換される．**
3. CSP **の解がユーザのプログラムに戻される．**

# Copris での制約モデリング例

- **騎士の巡歴** (Knight's Tour)

# 騎士の巡歴 (Knight's Tour)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 60 | 5 | 56 | 53 | 20 | 35 | 8 |
| 4 | 57 | 2 | 19 | 34 | 7 | 54 | 21 |
| 61 | 64 | 59 | 6 | 55 | 52 | 9 | 36 |
| 58 | 3 | 62 | 33 | 18 | 11 | 22 | 51 |
| 63 | 32 | 17 | 12 | 23 | 50 | 37 | 10 |
| 16 | 13 | 46 | 49 | 42 | 39 | 24 | 27 |
| 31 | 48 | 15 | 44 | 29 | 26 | 41 | 38 |
| 14 | 45 | 30 | 47 | 40 | 43 | 28 | 25 |

- $n \times n$ のチェス盤の左上から出発し，すべてのマス目を桂馬飛びで巡回し，左上に戻ってくる閉路を求める．
- ハミルトン閉路問題の一種である．
- $n$ が奇数の場合，解は存在しない．

# 騎士の巡歴 (Knight's Tour)

**皆さんはどのような制約モデルを思い浮かべるだろうか?**

# 騎士の巡歴 (Knight's Tour)

皆さんはどのような制約モデルを思い浮かべるだろうか?

ここでは，3 通りの制約モデルを提示し比較する．

1. AD モデル
   - Alldifferent 制約を用いたモデル
2. ARC モデル
   - Alldifferent 制約の代わりに有効グラフの弧を用いたモデル
3. INC モデル
   - 有効グラフの弧の次数条件だけを用い，インクリメンタルにハミルトン閉路を求めるモデル

# 騎士の巡歴の制約モデル: AD モデル

1. **各マス目に 1 から $n^2$ の値を対応させる.**

$$x_{ij} \in \{1 .. n^2\} \quad (\forall i \in \{1 .. n\}\ \forall j \in \{1 .. n\})$$

2. **各マス目の値は互いに異なる.**

$$\mathsf{alldifferent}(x_{11}, \ldots, x_{nn})$$

3. **$x_{11}$ を始点，$x_{23}$ を 2 番目，$x_{32}$ を終点とする.**

$$x_{11} = 1,\ x_{23} = 2,\ x_{32} = n^2$$

4. **移動可能なマス目のうちの一つに次の値がある (終点を除く).**

$$\bigvee_{(di,dj)\in\Delta} x_{i+di,j+dj} = x_{ij} + 1 \quad (\forall i\ \forall j)^1$$

[1] $\Delta = \{(-2,-1),(-2,1),(-1,-2),(-1,2),(1,-2),(1,2),(2,-1),(2,1)\}$

# 騎士の巡歴の制約モデル: AD モデル

```
for (i <- 1 to n; j <- 1 to n)
  int('x(i,j), 1, n*n)
add(Alldifferent(for (i <- 1 to n; j <- 1 to n) yield 'x(i,j)))
add('x(1,1) === 1); add('x(2,3) === 2); add('x(3,2) === n*n)

for (i <- 1 to n; j <- 1 to n; if (i,j) != (3,2)) {
  val cs = for ((i1,j1) <- adj(i, j))
           yield 'x(i1,j1) === 'x(i,j) + 1
  add(Or(cs))
}
```

- Copris で記述した AD モデル

# 騎士の巡歴の制約モデル: AD モデルの実行結果

| $n$ | AD モデル |
|---|---|
| 6 | 1.68 |
| 8 | 64.89 |
| 10 | 1291.20 |
| 12 | - |

- Copris で実行した CPU 時間 (秒)
- Timeout: 1800 秒
- MiniSat 2.2 simp, Linux 32bit, Intel Xeon 3.47GHz

# 騎士の巡歴の制約モデル: ARC モデル

1. **桂馬飛び可能なマス目の対に，弧を表す変数を対応させる．**

$$e_{i,j,i+di,j+dj} \in \{0,1\} \quad (\forall i\ \forall j\ \forall (di,dj) \in \Delta)$$

2. **各マス目の入次数と出次数は 1 に等しい．**

$$\bigvee_{(di,dj)\in\Delta} e_{i+di,j+dj,i,j} = 1 \quad (\forall i\ \forall j)$$

$$\bigvee_{(di,dj)\in\Delta} e_{i,j,i+di,j+dj} = 1 \quad (\forall i\ \forall j)$$

3. **各マス目には 1 から $n^2$ の値を対応させる．**

$$x_{ij} \in \{1\,..\,n^2\} \quad (\forall i\ \forall j)$$

4. **各弧に沿って値が増加する (始点へ戻る弧を除く)．**

$$e_{i,j,i+di,j+dj} > 0 \Rightarrow x_{i,j} < x_{i+di,j+dj} \quad (\forall i\ \forall j\ \forall (di,dj) \in \Delta)$$

# 騎士の巡歴の制約モデル: ARC モデル

```
for (i <- 1 to n; j <- 1 to n)
  int('x(i,j), 1, n*n)
add('x(1,1) === 1); add('x(2,3) === 2); add('x(3,2) === n*n)

for (i <- 1 to n; j <- 1 to n; (i1,j1) <- adj(i,j))
  int('e(i,j,i1,j1), 0, 1)
for (i <- 1 to n; j <- 1 to n) {
  add(Add(adj(i,j).map{case (i1,j1) => 'e(i,j,i1,j1)}) === 1)
  add(Add(adj(i,j).map{case (i1,j1) => 'e(i1,j1,i,j)}) === 1)
}

for (i <- 1 to n; j <- 1 to n; if (i,j) != (3,2)) {
  for ((i1,j1) <- adj(i, j))
    add(('e(i,j,i1,j1) > 0) ==> ('x(i,j) < 'x(i1,j1)))
}
```

- Copris で記述した ARC モデル

# 騎士の巡歴の制約モデル: ARC モデルの実行結果

| $n$ | AD モデル | ARC モデル |
|---|---|---|
| 6 | 1.68 | 0.64 |
| 8 | 64.89 | 1.17 |
| 10 | 1291.20 | 1.59 |
| 12 | - | 3.03 |
| 14 | - | 5.53 |
| 16 | - | 23.64 |
| 18 | - | 123.11 |
| 20 | - | - |

- Copris で実行した CPU 時間 (秒)
- Timeout: 1800 秒
- MiniSat 2.2 simp, Linux 32bit, Intel Xeon 3.47GHz

# 騎士の巡歴の制約モデル: INC モデル

1. **弧と次数に関する条件のみを記述し，SAT 符号化する．**

$$e_{i,j,i+di,j+dj} \in \{0,1\} \qquad (\forall i\ \forall j\ \forall (di,dj) \in \Delta)$$

$$\bigvee_{(di,dj)\in\Delta} e_{i+di,j+dj,i,j} = 1 \qquad (\forall i\ \forall j)$$

$$\bigvee_{(di,dj)\in\Delta} e_{i,j,i+di,j+dj} = 1 \qquad (\forall i\ \forall j)$$

2. SAT **ソルバーを起動する．**
3. **求めた** SAT **解が単一の閉路から成る場合，それが解である．**
4. **求めた** SAT **解が複数の閉路から成る場合，それらの弧を排除する条件を符号化し，**SAT **問題に追加する．**
5. 2 **に戻る．**

# 騎士の巡歴の制約モデル: INC モデル

```
for (i <- 1 to n; j <- 1 to n; (i1,j1) <- adj(i,j))
  int('e(i,j,i1,j1), 0, 1)
for (i <- 1 to n; j <- 1 to n) {
  add(Add(adj(i,j).map{case (i1,j1) => 'e(i,j,i1,j1)}) === 1)
  add(Add(adj(i,j).map{case (i1,j1) => 'e(i1,j1,i,j)}) === 1)
}
var tour = None
if (find) {
  do {
    val cycles = getCycles
    if (cycles.size == 1) {
      tour = cycles.head
    } else {
      for (cycle <- cycles) {
        cycle を除外する条件を加える
        cycle の逆順を除外する条件を加える
      }
      SAT 問題に追加
    }
  } while (tour == None && solver.satSolve)
}
```

# 騎士の巡歴の制約モデル: INC モデルの実行結果

| $n$ | AD モデル | ARC モデル | INC モデル |
|---|---|---|---|
| 6 | 1.68 | 0.64 | 0.99 |
| 8 | 64.89 | 1.17 | 1.56 |
| 10 | 1291.20 | 1.59 | 1.38 |
| 12 | - | 3.03 | 2.26 |
| 14 | - | 5.53 | 2.23 |
| 16 | - | 23.64 | 3.27 |
| 18 | - | 123.11 | 3.63 |
| 20 | - | - | 4.78 |
| 30 | - | - | 17.52 |
| 40 | - | - | 42.34 |
| 50 | - | - | 133.87 |
| 60 | | | 118.90 |
| 70 | | | 387.63 |
| 80 | | | - |
| 90 | | | - |
| 100 | | | 613.87 |

- Copris で実行した CPU 時間 (秒)
- Timeout: 1800 秒
- MiniSat 2.2 simp, Linux 32bit, Intel Xeon 3.47GHz

# サマリー

1. SAT と SAT ソルバー
2. 制約充足問題の SAT 符号化
   - 直接符号化
   - 順序符号化
3. SAT 型制約ソルバー
   - Sugar
   - Copris
4. Copris での制約モデリング例
   - 騎士の巡歴