SAT 型制約ソルバー Sugar と Scala インターフェイスについて

田村直之 丹生智也 番原睦則 (神戸大学)

2011 年 9 月 29 日 JSSST 第 28 回大会

CSP と SAT

制約充足問題 (CSP; Constraint Satisfaction Problem)

与えられたすべての制約を充足する変数への値割当てが存在するか否かを判定する問題

制約充足問題は，以下を満たす組 (X, D, C) として与えられる．

- X: **変数** (variable) **の有限集合．**
- D: **各変数の取り得る値の集合である**ドメイン (domain) **を定義する関数．**
- C: X **上の**制約 (constraint) **の有限集合．**

- **以下では $D(x)$ は整数の有限集合とする．**
- **また制約としては X 上の算術比較演算および論理演算で構成される論理式および** alldifferent **等のグローバル制約を用いる．**
  - alldifferent$(x_1, x_2, \ldots, x_n)$ **は x_i が互いに異なることを表す．**

制約モデリング言語と制約ソルバー

- CSP を記述する言語を**制約モデリング言語**という．
- 与えられた CSP を解くプログラムを**制約ソルバー** (あるいは CSP ソルバー) という．

- ほとんどの制約ソルバーは CSP だけでなく**制約最適化問題**も解くことができる．
  - 与えられた目的変数の値を最小化/最大化する解を求める．
- 多くの応用が存在する．
  - J リーグの対戦スケジュール作成 (IBM ILOG)
- これまで 4 回の**国際 CSP ソルバー競技会**が開催されるなど，世界中で活発に研究開発が行われている (2005, 2006, 2008, 2009)．

制約モデリング言語

制約モデリング言語には以下の 2 種類がある．

- **専用の制約モデリング言語**を用いるもの
  - 高機能な構文の導入により，より簡明に CSP を記述できる可能性がある．
  - 複数の制約ソルバーが入力言語として対応していれば，制約ソルバーを切換えることができ利便性が高い (XCSP など)．
- **汎用プログラミング言語**の枠内で記述するもの
  - ライブラリの利用などを含め既存言語の機能をすべて利用でき，柔軟な処理が可能になる．
  - 既存言語の構文の枠内での記述であり，言語によっては簡潔に CSP を記述できない．
  - Java 上の制約プログラミング API 標準化の試みとして JSR-331 [1] がある．

[1] Java Specification Request 331 (2010 The Most Innovative JSR 受賞)

JSR-331 でのプログラム例

JSR-331 (Java Specification Request 331) でのプログラム例

```
 1:  Var x = variable("X", 1, 10);
 2:  Var y = variable("Y", 1, 10);
 3:  Var z = variable("Z", 1, 10);
 4:  Var r = variable("R", 1, 10);
 5:  Var[] vars = { x, y, z, r };
 6:  post(x,"<",y);
 7:  post(z,">",4);
 8:  post(x.plus(y),"=",z);
 9:  postAllDifferent(vars);
10:  int[] coef1 = { 3, 4, -5, 2 };
11:  post(coef1,vars,">",0);
12:  post(vars,">=",15);
13:  int[] coef2 = { 2, -4, 5, -1 };
14:  post(coef2,vars,">",x.multiply(y));
```

SAT (Satisfiability Testing Problem)

与えられた命題論理式を充足する命題変数への真理値割当てが存在するか否かを判定する問題

- **通常，命題論理式は連言標準形** (CNF) **で与えられる．**
- CNF **式は複数の節の連言であり，各節は複数のリテラルの選言，各リテラルは命題変数あるいは命題変数の否定である．**

- **与えられた** SAT **問題** (CNF **式**) **の充足可能性を判定するプログラムを** SAT **ソルバー**という．充足可能な場合は，命題変数への真理値割当てを解として出力する．
- 2001 **年頃に数十倍から数百倍の高速化が実現された．**
- **元の問題を** SAT **に符号化し，**SAT **ソルバーを用いて求解する** SAT **型システム**が成功を収めている．
- **参考**: **人工知能学会誌** 2010 **年** 1 **月号 特集「最近の** SAT **技術の発展」**

SAT 型システム

- **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 3.4 **に利用されている．**
- **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** (iProver, Darwin)
- **一階モデル発見** (Paradox)
- **制約充足問題** (Sugar) [Tamura et al. 2006]

なぜ SAT 型システムか? (個人的意見)

- CDCL (Conflict Driven Clause Learning)，**非時間順バックトラック法，リスタート，監視リテラル，変数選択ヒューリスティック等の様々な技術の導入**
- **実装技術の進歩．SAT 競技会で前回の優勝ソルバーは優勝できない**
  - 2004 **年** ZChaff, 2005 **年** SatELiteGTI, 2007 **年** Rsat, 2009 **年** PrecoSAT, 2011 **年** glucose
- **キャシュを意識した実装** [Zhang & Malik 2003]
  - **例えば，ある** 260 **万節の** SAT **問題は，**MiniSat **で** 4 **秒以内で求解でき** L2 **キャッシュへのヒットレートは** 99%**以上である．**

```
$ valgrind --tool=cachegrind minisat gp10-10-1091.cnf
L2 refs:       42,842,531  ( 31,633,380 rd +11,209,151 wr)
L2 misses:     25,674,308  ( 19,729,255 rd + 5,945,053 wr)
L2 miss rate:         0.4% (        0.4%   +        1.0%  )
```

なぜ SAT 型システムか? (個人的意見)

SAT ソルバーをエンジンとして用いるアプローチは，1980 年代に Patterson らが提唱した RISC アプローチに似ている．

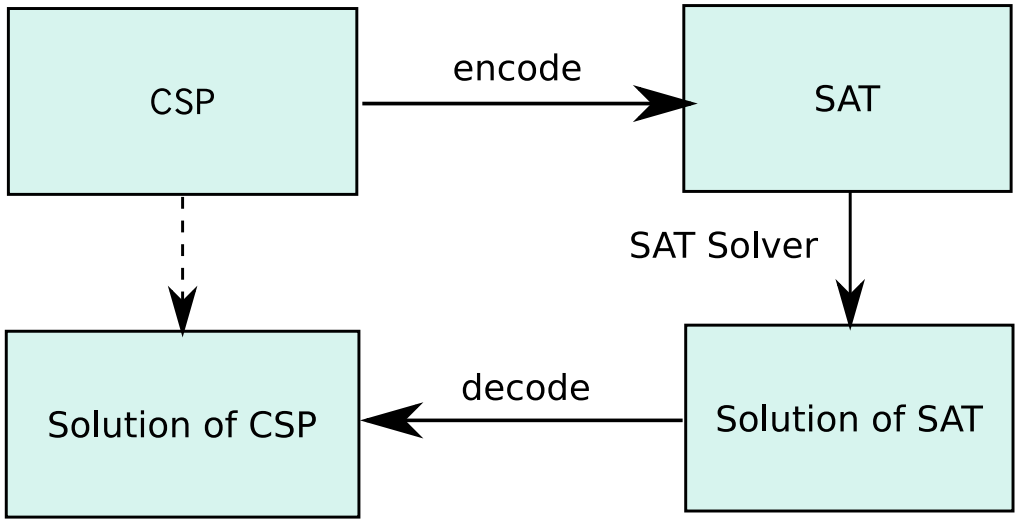
- **RISC**: Reduced Instruction Set Computer
  Patterson は “reduced” かつ高速な命令セットの計算機のほうが，“complex” な命令セットの計算機 (**CISC**) よりも高速になると主張した．

SAT ソルバー $\Longleftrightarrow$ **RISC**
SAT 符号化 $\Longleftrightarrow$ **最適化コンパイラ**

- この意味で，SAT ソルバーと SAT 符号化の双方が重要な研究テーマといえる．

SAT 型制約ソルバー Sugar

SAT 型制約ソルバー Sugar



- CSP を SAT に符号化し，SAT ソルバーを用いて求解するシステム
  - SAT への符号化は順序符号化を用いる [Tamura et al. 2006]
  - 制約最適化問題 (COP), Max-CSP にも対応している
- 2008 年および 2009 年の CSP ソルバー競技会のグローバル制約部門，2008 年 Max-CSP ソルバー競技会の 3 部門で優勝
- オープン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の問題で未知の最適解を発見

Sugar の制約モデリング言語

- Lisp 風の表記
- 配列やループがなく，複雑な問題の記述が面倒
- Sugar に入力する CSP ファイルを生成するプログラムを Perl, Ruby 等で作成していた

4-Queens の記述例

```
(int q_1 1 4)
(int q_2 1 4)
(int q_3 1 4)
(int q_4 1 4)
(alldifferent q_1 q_2 q_3 q_4)
(alldifferent (+ q_1 1) (+ q_2 2) (+ q_3 3) (+ q_4 4))
(alldifferent (- q_1 1) (- q_2 2) (- q_3 3) (- q_4 4))
```

Copris: Scala 上の制約プログラミング用 DSL [2]

[2] DSL: Domain-Specific Language

Scala 言語の概要

- Martin Odersky による関数型オブジェクト指向言語
- 言語の特徴
  - 関数型言語とオブジェクト指向言語の融合
  - 強力な型推論，高階関数
  - Immutable Collections
  - Actor による並行計算
  - 埋込み DSL の実装に適している
    - ケースクラス，演算子の多重定義，暗黙変換，名前呼び，オブジェクトのメソッドインポート，動的変数，可変長引数など
- 処理系の特徴
  - JVM [3] へのコンパイラ処理系
  - インタラクティブな実行環境 (REPL [4]) も用意されている
  - Java のクラス・ライブラリをそのまま利用できる

[3] JVM: Java Virtual Machine
[4] REPL: Read Eval Print Loop

Copris (Constraint Programming in Scala)

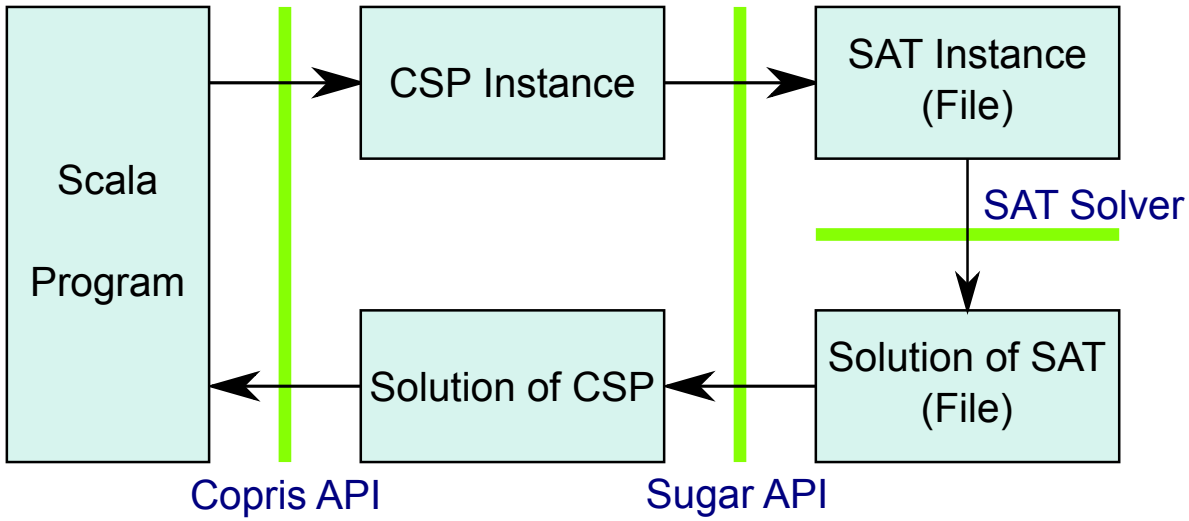
Scala 上の制約プログラミング用 DSL [1]

特徴

- Scala の機能により**簡明な制約モデリング**が可能
- SAT 型制約ソルバーである Sugar を利用することにより，**高性能な制約解消**を実現
- SAT ソルバーとして Sat4j を用いれば，すべてが JVM [2] 上で稼働
- 自分で SAT 符号化を実装するのが面倒な場合，簡単に SAT 問題を生成できる

[1] DSL: Domain-Specific Language
[2] JVM: Java Virtual Machine

Copris の構成



1. Scala **プログラム中で** CSP **を定義する．**
2. Scala **プログラムから制約ソルバー** Sugar **を呼び出す．**
   - Sugar **は** CSP **を** SAT **に符号化し，**SAT **ソルバーを呼び出す．**
   - SAT **ソルバーとしては** Sat4j **あるいは** MiniSat, PrecoSAT, GlueMiniSat **等の外部ソルバーを指定できる．**
   - SAT **ソルバーの解は** CSP **の解に逆符号化される．**
3. CSP **の解は** Scala **プログラムに返される．**

Copris DSL: インポート

```
import jp.kobe_u.copris._
import jp.kobe_u.copris.dsl._
```

- 1 行目は名前空間をインポートする (Java と同様).
- 2 行目は dsl オブジェクトに定義されているメソッドをインポートする.
  - 整数変数を定義するメソッド int
  - 制約を追加するメソッド add
  - 解を探索するメソッド find
  - 解を返すメソッド solution

```
import jp.kobe_u.copris.sugar.dsl._
```

- Sugar 専用のメソッド (SAT ソルバーの指定等) をインポートする.

Copris DSL: 整数変数とドメインの定義

```
int('x, 0, 7)       // 変数名と上下限によるドメイン定義
int('y, Set(0,2)) // 集合によるドメイン定義
```

- 'x 等は Scala の Symbol オブジェクトであり，暗黙変換により Copris の整数変数を表す Var オブジェクトに変換される．
- int メソッドで定義された変数はデフォールトの CSP に追加される．

```
int('a(1), 0, 7)     // 添字付き変数の定義
int('b(1,2), 0, 7)  // 添字付き変数の定義
```

- Var オブジェクトに添字を与えると，新しい Var オブジェクトが生成される．

Copris DSL: 制約の定義

```
add('x + 'y === 7)
add('x < 'y || 'y >= 'z)
add(Alldifferent('x, 'y, 'z))
```

- Scala の演算子多重定義により，算術・比較・論理演算子を用いて制約を記述できる．
  - ただし等号，非等号，含意には ===, !==, ==> を用いる．
- alldifferent 等のグローバル制約には，対応するクラスを用いる．
- add メソッドにより制約はデフォールトの CSP に追加される．

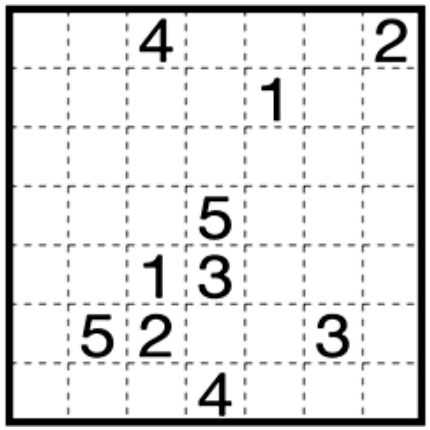
Copris DSL: 解の探索

```
find        // 最初の解の探索
findNext    // 他の解の探索
findOpt     // 最適解の探索
```

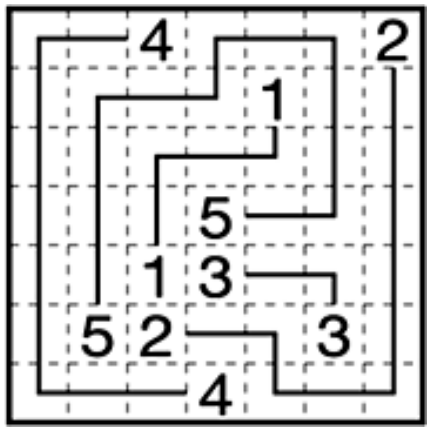
- `find` は CSP を SAT に符号化し，SAT ソルバーを起動して解を求める．
- `findNext` はこれまで求まった解の否定を制約として追加し，再度 SAT ソルバーを起動することで他の解を求める．
  - CSP 全体を再符号化するのでなく，必要部分だけを符号化して追加する．
- `findOpt` は与えられた目的変数の最小値あるいは最大値を 2 分探索により求める．

Copris プログラム例: ナンバーリンク [1]

問題



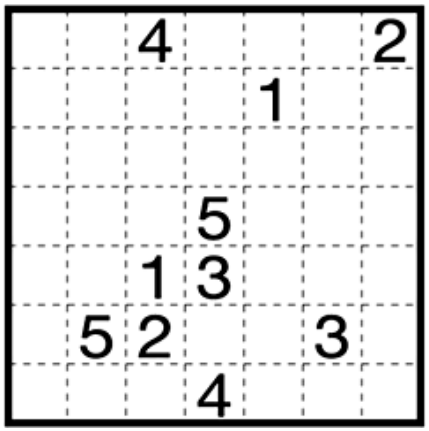
解答



1. 同じ数字どうしを線でつなげます．
2. 線はタテヨコに引き，マスの中央を通ります．
3. 線は 1 マスに 1 本だけ通過できます．線をワクの外に出したり，交差や枝分かれさせてはいけません．また，線は数字が入っているマスを通過してもいけません．

[1] ナンバーリンクはニコリの登録商標

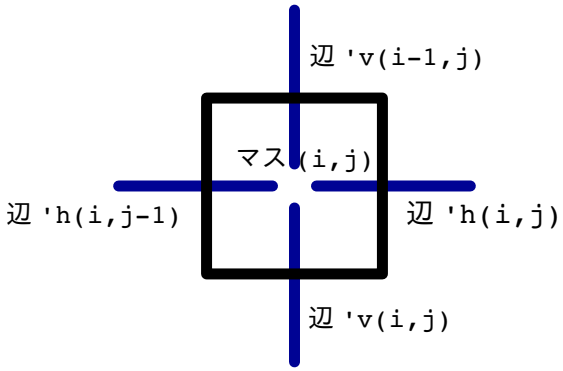
Copris プログラム例: ナンバーリンク



入力データ

```
val puzzle = List(
  List(0,0,0,4,0,0,2),
  List(0,0,0,0,1,0,0),
  ...)
val m = 7  // 行数
val n = 7  // 列数
val k = 5  // 数字の最大値
```

Copris プログラム例: ナンバーリンク



- **各マス** (i,j) **から下のマス** (i+1,j) **への辺を** 0-1 **変数** 'v(i,j) **で表す．**1 **は辺がつながっていることを意味する．**
- **各マス** (i,j) **から右のマス** (i,j+1) **への辺を** 0-1 **変数** 'h(i,j) **で表す．**1 **は辺がつながっていることを意味する．**

```
for (i <- 0 until m; j <- 0 until n) {
  if (i < m - 1) int('v(i, j), 0, 1)
  if (j < n - 1) int('h(i, j), 0, 1)
}
```

Copris プログラム例: ナンバーリンク

- 各マスの次数を表す変数 'd(i,j) を導入し，白マスの次数は 1，数字マスの次数は 0 または 2 である制約を追加する．

```
def degree(i: Int, j: Int) = {
  val vs = for (i0 <- i-1 to i; if 0 <= i0 && i0 < m - 1)
           yield 'v(i0, j)
  val hs = for (j0 <- j-1 to j; if 0 <= j0 && j0 < n - 1)
           yield 'h(i, j0)
  Add(vs ++ hs)
}

for (i <- 0 until m; j <- 0 until n) {
  if (puzzle(i)(j) > 0)
    int('d(i, j), 1)
  else
    int('d(i, j), Set(0, 2))
  add('d(i, j) === degree(i,j))
}
```

Copris プログラム例: ナンバーリンク

- 以上で，数字マスを結ぶ交差しないパスを求めることができるが，同じ数字を結ぶ条件が含まれていない．
- 各マスがどの数字と結ばれているかを表す変数 `'x(i,j)` を導入する．
- 辺で結ばれているマスは同一の値になる制約を追加する．

```
for (i <- 0 until m; j <- 0 until n) {
  int('x(i, j), 1, k)
  if (puzzle(i)(j) > 0)
    add('x(i,j) === puzzle(i)(j))
  if (i < m - 1)
    add(('v(i, j) !== 0) ==> ('x(i, j) === 'x(i+1, j)))
  if (j < n - 1)
    add(('h(i, j) !== 0) ==> ('x(i, j) === 'x(i, j+1)))
}
```

Copris プログラム例: ナンバーリンク

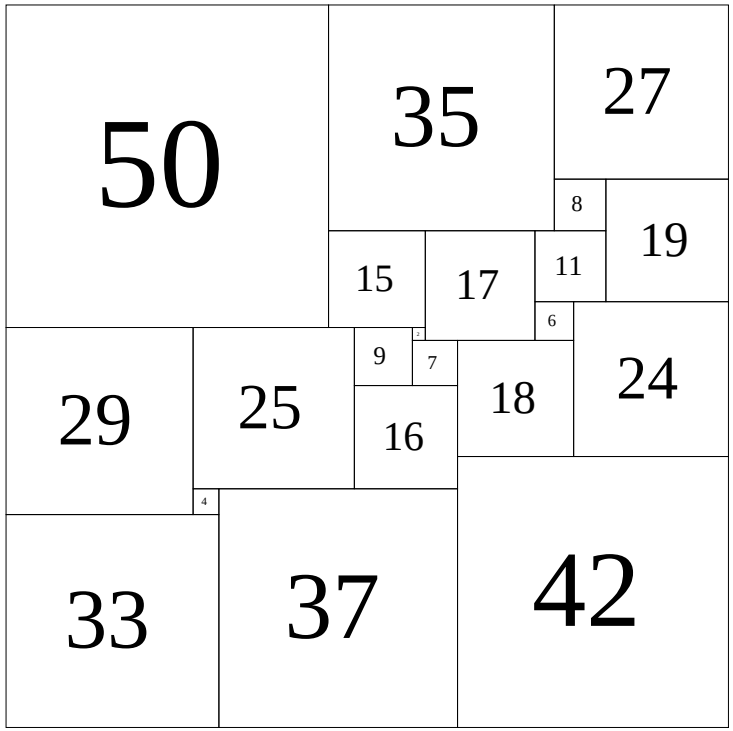
```
 1:  def degree(i: Int, j: Int) = {
 2:    val vs = for (i0 <- i-1 to i; if 0 <= i0 && i0 < m - 1) yield 'v(i0, j)
 3:    val hs = for (j0 <- j-1 to j; if 0 <= j0 && j0 < n - 1) yield 'h(i, j0)
 4:    Add(vs ++ hs)
 5:  }
 6:  for (i <- 0 until m; j <- 0 until n) {
 7:    if (i < m - 1) int('v(i, j), 0, 1)
 8:    if (j < n - 1) int('h(i, j), 0, 1)
 9:  }
10:  for (i <- 0 until m; j <- 0 until n) {
11:    if (puzzle(i)(j) > 0) int('d(i, j), 1) else int('d(i, j), Set(0, 2))
12:    add('d(i, j) === degree(i,j))
13:  }
14:  for (i <- 0 until m; j <- 0 until n) {
15:    int('x(i, j), 1, k)
16:    if (puzzle(i)(j) > 0)
17:      add('x(i,j) === puzzle(i)(j))
18:    if (i < m - 1)
19:      add(('v(i, j) !== 0) ==> ('x(i, j) === 'x(i+1, j)))
20:    if (j < n - 1)
21:      add(('h(i, j) !== 0) ==> ('x(i, j) === 'x(i, j+1)))
22:  }
```

Copris プログラム例: ナンバーリンク

- 15×15 の問題でも 1 分程度で解くことができる.
- さらに，辺を有向辺にし，盤面を 4 分割した部分領域での入次数・出次数に関する制約を追加することで速度が大幅に改善され，http://www.janko.at 中の最大サイズの問題 (25×42) を 1 分程度で解くことができた.

Copris プログラム例: 完全正方形詰込み問題

2, 4, 6, 7, 8, 9, 11, 15, 16, 17, 18, 19, 24, 25, 27, 29, 33, 35, 37, 42, 50 **のサイズの** 21 **個の正方形を** 112×112 **の正方形に隙間なく詰込む問題**



21 : 112A AJD 1978

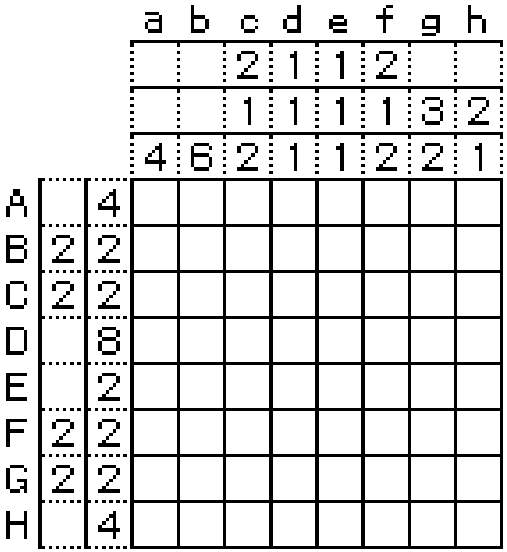
Copris プログラム例: 完全正方形詰込み問題

```
 1:  val size = 112
 2:  val s = List(2,4,6,7,8,9,11,15,16,17,
 3:    18,19,24,25,27,29,33,35,37,42,50)
 4:  val n = s.size
 5:  for (i <- 0 to n-1) {
 6:    int('x(i), 0, size - s(i))
 7:    int('y(i), 0, size - s(i))
 8:  }
 9:  for (i <- 0 to n-1; j <- i+1 to n-1)
10:    add('x(i) + s(i) <= 'x(j) ||
11:        'x(j) + s(j) <= 'x(i) ||
12:        'y(i) + s(i) <= 'y(j) ||
13:        'y(j) + s(j) <= 'y(i))
```

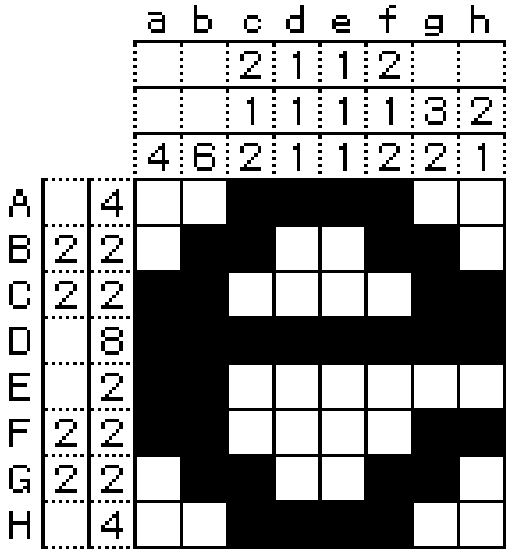
- 'x(i), 'y(i) は i 番目の正方形の左下の座標を表す.
- 10-13 行目は i 番目と j 番目の正方形が重ならない条件を表している.
- 求解時間は約 40 秒.

Copris プログラム例: ノノグラム

問題



解答



- Survey of Paint-by-Number Puzzle Solvers (http://webpbn.com/survey/) **中の他ソルバーよりも高速だった (唯一 全問題を解くことができた).**
- http://www.comp.lancs.ac.uk/~ss/nonogram/puzzles **の** 100×100 **の問題でも** 30 **秒程度で求解できた.**

Copris プログラム例: ノノグラムの比較結果

puzzle	size	Copris	Wolter	Olsak	Simpson	BGU	Lagerkvist Gecode	Kjellerstrand Gecode	Kjellerstrand Lazyfd
1: Dancer	5 x 10	1.01s	0	0	0	0.06s	0.01s	0.01s	0.04s
6: Cat	20 x 20	4.03s	0	0	0	0.08s	0.01s	0.02s	0.44s
21: Skid	14 x 25	4.47s	0	0	0	0.08s	0.01s	0.02s	0.64s
27: Bucks	27 x 23	8.64s	0	0	0	0.16s	0.01s	0.02s	1.1s
23: Edge	10 x 11	1.38s	0	0	0	0.09s	0.01s	0.01s	0.08s
2413: Smoke	20 x 20	4.58s	0	0	0	0.19s	0.01s	0.02s	0.60s
16: Knot	34 x 34	10.66s	0	0	0	0.20s	0.01s	0.02s	5.5s
529: Swing	45 x 45	13.34s	0	0	0	0.25s	0.02s	0.04s	13.2s
65: Mum	34 x 40	11.85s	0	0	0.01s	0.26s	0.02s	0.04s	11.0s
7604: DiCap	52 x 63	14.87s	0	0	0	2.1s	0.02s	0.06s	+
1694: Tragic	45 x 50	12.42s	0	0.03s	0.54s	0.62s	0.14s	3.6m	32.1s
1611: Merka	55 x 60	16.18s	0.01s	0.01s	0	0.35s	0.03s	+	1.1m
436: Petro	40 x 35	11.50s	0.06s	15.2s	0.10s	0.58s	1.4m	1.6s	6.2s
4645: M&M	50 x 70	12.64s	0.07s	0.10s	0.02s	0.84s	0.93s	0.28s	48.1s
3541: Signed	60 x 50	13.43s	0.04s	1.1s	5.4m	0.68s	0.57s	0.56s	1.0m
803: Light	50 x 45	8.60s	0.38s	+	0.02s	1.1s	+	+	4.7s
6574: Forever	25 x 25	9.03s	3.7s	2.0s	18.9s	44.3s	4.7s	2.3s	1.7s
2040: Hot	55 x 60	16.58s	0.83s	+	1.2m	0.93s	+	+	2.6m
6739: Karate	40 x 40	11.77s	0.80s	17.3s	18.3m	0.61s	56.0s	57.8s	13.6s
8098: 9-Dom	19 x 19	3.65s	11.0s	4.0m	+	2.8m	12.6m	3.8s	1.8s
2556: Flag	65 x 45	15.16s	0.55s	1.5s	0	24.2s	3.4s	+	4.2s
2712: Lion	47 x 47	14.36s	6.3s	+	+	7.8s	+	+	+
10088: Marley	52 x 63	18.19s	+	+	+	22.0s	+	+	2.9m
9892: Nature	50 x 40	18.72s	+	18.6m	+	+	+	+	2.4m

- Copris は Intel Core i5 vPro 2.0GHz × 2, その他は 2.6GHz AMD Phenom II X4 810 quad-core で実行． + はタイムアウト (30 分)

Copris プログラム例: ノノグラムのプログラム

```
 1:  for (i <- 0 until m; j <- 0 until n)
 2:    int('x(i,j), 0, 1)
 3:  for (i <- 0 until m; k <- 0 until rows(i).size)
 4:    int('r(i,k), 0, n-rows(i)(k))
 5:  for (i <- 0 until m; k <- 0 until rows(i).size-1)
 6:    add('r(i,k) + rows(i)(k) < 'r(i,k+1))
 7:  for (j <- 0 until n; k <- 0 until cols(j).size)
 8:    int('c(j,k), 0, m-cols(j)(k))
 9:  for (j <- 0 until n; k <- 0 until cols(j).size-1)
10:    add('c(j,k) + cols(j)(k) < 'c(j,k+1))
11:  for (i <- 0 until m; j <- 0 until n) {
12:    val rs = for (k <- 0 until rows(i).size)
13:             yield 'r(i,k) <= j && 'r(i,k) + rows(i)(k) > j
14:    add('x(i,j) > 0 <==> Or(rs))
15:    val cs = for (k <- 0 until cols(j).size)
16:             yield 'c(j,k) <= i && 'c(j,k) + cols(j)(k) > i
17:    add('x(i,j) > 0 <==> Or(cs))
18:  }
```

- 'x(i,j) は i 行目 j 列目のマスの色を表す (1 が黒).
- 'r(i,k) は i 行目 k 番目のブロックの左端の位置を表す.
- 'c(j,k) は j 列目 k 番目のブロックの上端の位置を表す.
- 13 行目および 17 行目は (i,j) のマスが'r(i,k) および 'c(j,k) のブロックに含まれる条件を表す.

まとめ

- SAT **型制約ソルバー** Sugar **の紹介**
- Scala **上の制約プログラミング用** DSL **である** Copris **の紹介**
- **将来的には，以下も実現したい．**
  - Max-CSP, Weighted CSP **への対応**
  - JSR-331 **準拠の** CSP **ソルバーとの接続**
  - XCSP **の出力**

デモ

デモの内容

1. Scala REPL [2] からの利用
2. 8-Queens 問題
3. 完全正方形詰込み問題
4. 数独 [5]
5. ノノグラム [6]
6. ナンバーリンク [7]
7. 倉庫番 [8]

[5] 数独はニコリの登録商標
[6] ノノグラムは西尾徹也，石田伸子の創案
[7] ナンバーリンクはニコリの登録商標
[8] 倉庫番はファルコンの登録商標

Scala REPL からの利用

```
/* Copris クラスや DSL 用メソッドのインポート */
scala> import jp.kobe_u.copris._
scala> import jp.kobe_u.copris.dsl._

/* CSP の定義 */
scala> int('x, 0, 7)
scala> int('y, 0, 7)
scala> add('x + 'y === 7)
scala> add('x * 2 + 'y * 4 === 20)

/* 解の探索と表示 */
scala> find
  res: Boolean = true
scala> solution
  res: Solution = Solution(Map(x -> 4, y -> 3),Map())
scala> solution('x)
  res: Int = 4
```

8-Queens 問題

8 × 8 のチェス盤に 8 個のクイーンを互いに取られないように配置する問題

```
 1:  val n = 8
 2:  for (i <- 1 to n) int('q(i), 1, n)
 3:  add(Alldifferent((1 to n).map(i => 'q(i))))
 4:  add(Alldifferent((1 to n).map(i => 'q(i)+i)))
 5:  add(Alldifferent((1 to n).map(i => 'q(i)-i)))
 6:  if (find) {
 7:    do {
 8:      println(solution)
 9:    } while (findNext)
10:  }
```

- 'q(i) は i 行目に配置するクイーンの列位置を表す.
- 100-Queens の初解の求解時間は約 5 秒.

完全正方形詰込み問題

21 個の正方形を 112 × 112 の正方形に隙間なく詰込む問題

```
 1:  val size = 112
 2:  val s = List(2,4,6,7,8,9,11,15,16,17,
 3:    18,19,24,25,27,29,33,35,37,42,50)
 4:  val n = s.size
 5:  for (i <- 0 to n-1) {
 6:    int('x(i), 0, size - s(i))
 7:    int('y(i), 0, size - s(i))
 8:  }
 9:  for (i <- 0 to n-1; j <- i+1 to n-1)
10:    add('x(i) + s(i) <= 'x(j) ||
11:        'x(j) + s(j) <= 'x(i) ||
12:        'y(i) + s(i) <= 'y(j) ||
13:        'y(j) + s(j) <= 'y(i))
```

- 'x(i), 'y(i) は i 番目の正方形の左下の座標を表す.
- 10-13 行目は i 番目と j 番目の正方形が重ならない条件を表している.
- 求解時間は約 40 秒.

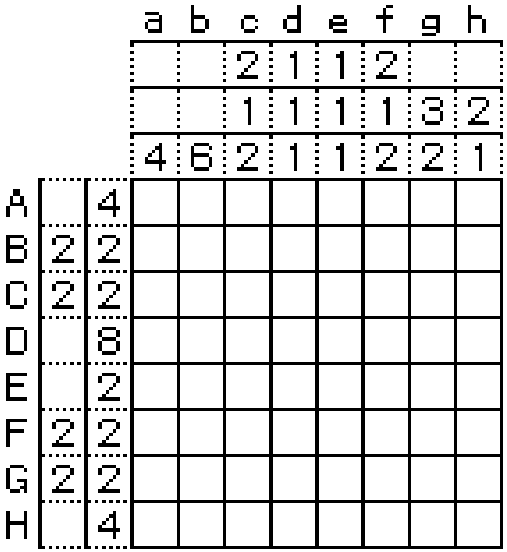
数独

```
 1:  val m = 3
 2:  val n = m*m
 3:  val puzzle: Seq[Seq[Int]] = /* パズルの定義 */
 4:  for (i <- 0 until n; j <- 0 until n)
 5:    int('x(i,j), 1, n)
 6:  for (i <- 0 until n)
 7:    add(Alldifferent((0 until n).map('x(i,_))))
 8:  for (j <- 0 until n)
 9:    add(Alldifferent((0 until n).map('x(_,j))))
10:  for (i <- 0 until n by m; j <- 0 until n by m) {
11:    val xs = for (di <- 0 until m; dj <- 0 until m)
12:      yield 'x(i+di,j+dj)
13:    add(Alldifferent(xs))
14:  }
15:  for (i <- 0 until n; j <- 0 until n; if puzzle(i)(j) > 0)
16:    add('x(i,j) === puzzle(i)(j))
```

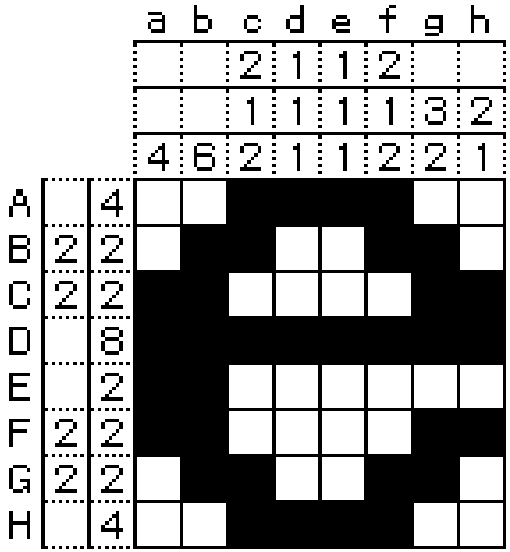
- 'x(i,j) は i 行目 j 列目のマスの値を表す．
- 各行，各列，各ブロックで互いに値が異なる条件を alldifferent 制約で表している．

ノノグラム

問題



解答



- Survey of Paint-by-Number Puzzle Solvers (http://webpbn.com/survey/) **中の他ソルバーよりも高速だった.**
- http://www.comp.lancs.ac.uk/~ss/nonogram/puzzles **の** 100×100 **の問題でも** 30 **秒程度で求解できた.**

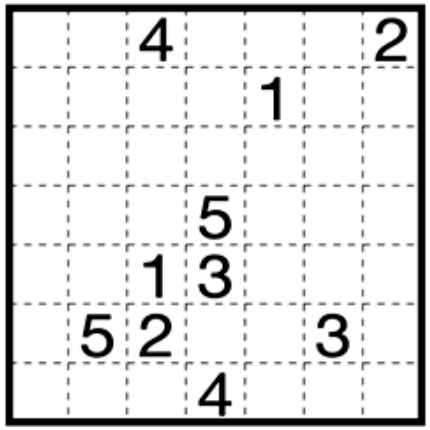
ノノグラム

```
 1:  for (i <- 0 until m; j <- 0 until n)
 2:    int('x(i,j), 0, 1)
 3:  for (i <- 0 until m; k <- 0 until rows(i).size)
 4:    int('r(i,k), 0, n-rows(i)(k))
 5:  for (i <- 0 until m; k <- 0 until rows(i).size-1)
 6:    add('r(i,k) + rows(i)(k) < 'r(i,k+1))
 7:  for (j <- 0 until n; k <- 0 until cols(j).size)
 8:    int('c(j,k), 0, m-cols(j)(k))
 9:  for (j <- 0 until n; k <- 0 until cols(j).size-1)
10:    add('c(j,k) + cols(j)(k) < 'c(j,k+1))
11:  for (i <- 0 until m; j <- 0 until n) {
12:    val rs = for (k <- 0 until rows(i).size)
13:             yield 'r(i,k) <= j && 'r(i,k) + rows(i)(k) > j
14:    add('x(i,j) > 0 <==> Or(rs))
15:    val cs = for (k <- 0 until cols(j).size)
16:             yield 'c(j,k) <= i && 'c(j,k) + cols(j)(k) > i
17:    add('x(i,j) > 0 <==> Or(cs))
18:  }
```

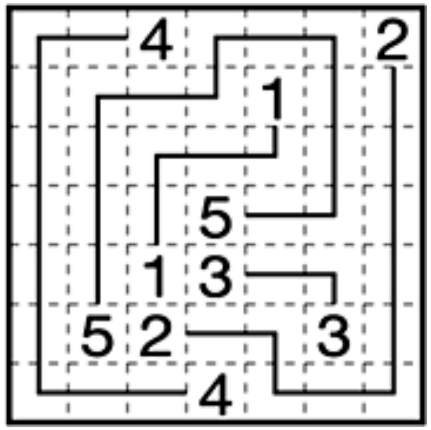
- `'x(i,j)` は i 行目 j 列目のマスの色を表す (1 が黒).
- `'r(i,k)` は i 行目 k 番目のブロックの左端の位置を表す.
- `'c(j,k)` は j 列目 k 番目のブロックの上端の位置を表す.
- 13 行目および 17 行目は (i,j) のマスが `'r(i,k)` および `'c(j,k)` のブロックに含まれる条件を表す.

ナンバーリンク

問題



解答



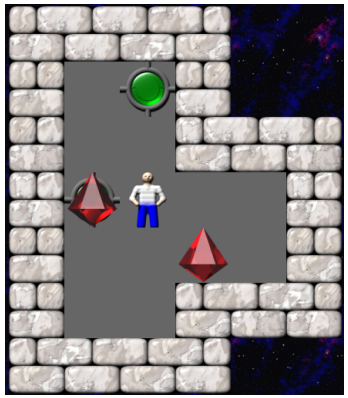
- **各マスにおける次数の条件 (白マスの次数は 0 または 2，数字マスの次数は 1) および辺で結ばれたマスの値が等しくなる条件を記述すれば良いが，必ずしも速くない．**
- **辺を有向辺にし，盤面を 4 分割した部分領域での入次数・出次数に関する制約を追加することで速度が改善され，** http://www.janko.at **中の最大サイズの問題 (25 × 42) を 1 分程度で解くことができた．**

ナンバーリンク

```
 1:  def degree(i: Int, j: Int) = {
 2:    val vs = for (i0 <- i-1 to i; if 0 <= i0 && i0 < m - 1) yield 'v(i0, j)
 3:    val hs = for (j0 <- j-1 to j; if 0 <= j0 && j0 < n - 1) yield 'h(i, j0)
 4:    Add(vs ++ hs: _*)
 5:  }
 6:  for (i <- 0 until m; j <- 0 until n) {
 7:    if (i < m - 1) int('v(i, j), 0, 1)
 8:    if (j < n - 1) int('h(i, j), 0, 1)
 9:  }
10:  for (i <- 0 until m; j <- 0 until n) {
11:    if (puzzle(i)(j) > 0) int('d(i, j), 1) else int('d(i, j), Set(0, 2))
12:    add('d(i, j) === degree(i,j))
13:  }
14:  for (i <- 0 until m; j <- 0 until n) {
15:    int('x(i, j), 1, k)
16:    if (puzzle(i)(j) > 0)
17:      add('x(i,j) === puzzle(i)(j))
18:    if (i < m - 1)
19:      add(('v(i, j) !== 0) ==> ('x(i, j) === 'x(i+1, j)))
20:    if (j < n - 1)
21:      add(('h(i, j) !== 0) ==> ('x(i, j) === 'x(i, j+1)))
22:  }
```

- 'v(i,j) はマス (i, j) からマス (i+1, j) への辺を表す.
- 'h(i,j) はマス (i, j) からマス (i, j+1) への辺を表す.
- 'd(i,j) はマス (i, j) の次数を表す.
- 'x(i,j) はマス (i, j) の値 (つながっている数字) を表す.

倉庫番



- **有界モデル検査と同様の手法で解くことができる．**

$$\Phi(t) \iff \bigwedge_{s=0}^{t} S(V_s) \wedge I(V_0) \wedge \bigwedge_{s=0}^{t-1} T(V_s, V_{s+1}) \wedge \bigvee_{s=0}^{t} G(V_s)$$

  - V_s: s **ステップ目の状態を表す変数**
  - $S(V_s)$: **状態であることの条件**
  - $I(V_s)$: **初期状態であることの条件**
  - $G(V_s)$: **目標状態であることの条件**
  - $T(V_s, V_{s+1})$: **状態遷移の条件**

Scala

クラス定義

項の定義

```
1:  abstract class Term
2:  case class Num(a: Int) extends Term
3:  case class Add(x0: Term, x1: Term) extends Term
```

- case class **を用いると以下が自動的に利用可能になる.**
  - equals, hashCode, toString
  - **アクセサー** (Add **に対しての** x0 **や** x1)
  - factory **メソッド** (new **なしのオブジェクト生成**)
  - **パターンマッチ** (**後述**)

クラス定義

項の定義

```
1:  abstract class Term
2:  case class Num(a: Int) extends Term
3:  case class Add(x0: Term, x1: Term) extends Term
```

- **実行例** (`scala>` **は入力行**)

```
scala> val t = Add(Num(1), Num(2))
t: Add = Add(Num(1),Num(2))
scala> t.x0
res: Term = Num(1)
scala> t.x0 == Num(1)
res: Boolean = true
```

単一オブジェクト定義

定数の定義

```
1:  object ZERO extends Num(0)
2:  object ONE extends Num(1)
```

- **実行例**

```
scala> ONE
res: ONE.type = Num(1)
scala> ONE == Num(1)
res: Boolean = true
```

メソッド定義

項の値の定義

```
1:  abstract class Term {
2:    def value: Int = this match {
3:      case Num(a) => a
4:      case Add(x0, x1) => x0.value + x1.value
5:    }
6:  }
```

- `obj match` は，パターンマッチ構文
- **実行例**

```
scala> val t = Add(Num(1), Num(2))
t: Add = Add(Num(1),Num(2))
scala> t.value
res: Int = 3
```

演算子定義

Add を infix に

```
1:  abstract class Term {
2:    .....
3:    def + (x: Term) = Add(this, x)
4:    def + (a: Int) = Add(this, Num(a))
5:  }
```

- **実行例**

```
scala> Num(1) + Num(2)
res: Add = Add(Num(1),Num(2))
scala> Num(1) + 2
res: Add = Add(Num(1),Num(2))
```

可変引数

Add を可変引数に

```
1:  abstract class Term {
2:    def value: Int = this match {
3:      case Num(a) => a
4:      case Add(xs @ _*) => xs.map(x => x.value).sum
5:    }
6:  }
7:  case class Add(xs: Term*) extends Term
```

- **実行例**

```
scala> val t = Add(Num(1), Num(2), Num(3))
t: Add = Add(WrappedArray(Num(1), Num(2), Num(3)))
scala> t.value
res: Int = 6
```

Factory メソッドの追加

Add() を ZERO に

```
1:  object Add {
2:    def apply() = ZERO
3:  }
```

- Add() は Add.apply() と同義
- 実行例

```
// 追加前
scala> Add()
res: Add = Add(WrappedArray())
// 追加後
scala> Add()
res: object ZERO = Num(0)
```

apply メソッド

変数クラスを定義

```
1:  case class Var(name: String, is: Int*)
2:  extends Term {
3:    def apply(is1: Int*) =
4:      Var(name, is ++ is1: _*)
5:  }
```

- obj(args) は obj.apply(args) と同義
- 実行例

```
scala> val x = Var("x")
x: Var = Var(x,WrappedArray())
scala> x(1,2)
res: Var = Var(x,ArrayBuffer(1, 2))
```

toString メソッド

toString メソッドをオーバーライド

```
 1:  case class Num(a: Int) extends Term {
 2:    override def toString = a.toString
 3:  }
 4:  case class Add(xs: Term*) extends Term {
 5:    override def toString =
 6:      xs.mkString(productPrefix + "(", ",", ")")
 7:  }
 8:  case class Var(name: String, is: Int*) ... {
 9:    override def toString = .....
10:  }
```

- **実行例**

```
scala> x(1) + x(2)
res: Add = Add(x(1),x(2))
```

コレクションと高階関数の利用

```
scala> (1 to 4).map(i => x(i))
res: ... = Vector(x(1), x(2), x(3), x(4))

scala> (1 to 4).map(x(_))
res: ... = Vector(x(1), x(2), x(3), x(4))

scala> (1 to 4).map(i => x(i)+i)
res: ... = Vector(Add(x(1),1), Add(x(2),2),
                  Add(x(3),3), Add(x(4),4))
```

- `i => x(i)` は `i` を `x(i)` に写像する無名関数
- `_` を用いて引数名を省略することもできる

コレクションと高階関数の利用

```
scala> for (i <- 1 to 4) yield x(i)
res: ... = Vector(x(1), x(2), x(3), x(4))

scala> for (i <- 1 to 4) yield x(i)+i
res: ... = Vector(Add(x(1),1), Add(x(2),2),
                  Add(x(3),3), Add(x(4),4))
```

- 上のように，for 内包表記を用いることもできる

```
scala> Add((1 to 4).map(x(_)): _*)
res: Add = Add(x(1),x(2),x(3),x(4))
```

- 可変引数メソッドへのパラメータ渡し

コレクションと高階関数の利用

```
scala> (1 to 4).reduceLeft[Any]((_,_))
res: Any = (((1,2),3),4)

scala> (1 to 4).map(x(_)).reduceLeft[Term](Add(_,_))
res: Term = Add(Add(Add(x(1),x(2)),x(3)),x(4))

scala> (1 to 4).foldLeft[Any](0)((_,_))
res: Any = ((((0,1),2),3),4)

scala> (1 to 4).map(x(_)).foldLeft[Term](ZERO)(_ + _)
res: Term = Add(Add(Add(Add(0,x(1)),x(2)),x(3)),x(4))
```

- reduceLeft, foldLeft **の使用例**

Map の利用

付値関数を再定義

```
1:  abstract class Term {
2:    def value(as: Map[Var,Int]): Int = this match {
3:      case Num(a) => a
4:      case Add(xs @ _*) => xs.map(x => x.value).sum
5:      case v: Var => as(v)
6:    }
7:  }
```

- **実行例**

```
scala> val as = Map(Var("x")->1, Var("y")->2)
as: Map[Var,Int] = Map((x,1), (y,2))
scala> (Var("x") + Var("y")).value(as)
res: Int = 3
```

Java ライブラリの利用

MiniSat を呼出す

```
1:  object MiniSat {
2:    val minisat = "minisat"
3:    def solve(cnf: String, out: String) = {
4:      val cmd = Array(minisat, cnf, out)
5:      val process = Runtime.getRuntime.exec(cmd)
6:      process.waitFor
7:    }
8:  }
```

- java.lang **は，自動的に** import **されている**

オブジェクトの import

Main オブジェクトのメソッドを import

```
1:  object Main {
2:    var list: Set[Term] = Set.empty
3:    def add(xs: Term*) =
4:      list = list ++ xs
5:    def clear =
6:      list = Set.empty
7:  }
```

- **実行例**

```
scala> import Main._
scala> add(Var("x")+1)
scala> add(Var("y")+2)
scala> list
res: Set[Term] = Set(Add(x,1), Add(y,2))
```

暗黙変換 (implicit conversion)

Symbol から Var への暗黙変換

```
1:  object Main {
2:    implicit def symbol2var(s: Symbol) = Var(s.name)
3:    .....
4:  }
```

- Symbol **で型エラーになる場合，**Var **へ変換される**
- **実行例**

```
scala> 'x
res: Symbol = 'x

scala> 'x + 'y + 1
res: Add = Add(Add(x,y),1)
```