Scala 上で実現されたSAT型制約プログラミングシステムのための開発ツール Scarab について

宋 剛秀　番原 睦則　田村 直之
Daniel Le Berre　Stéphanie Roussel

Scarab は制約プログラミングのためのドメイン特化言語, SAT 符号化モジュール, SAT ソルバーへのインターフェースから構成されており，SAT 型制約プログラミングシステム開発者を対象に，表現性，変更性，効率性を備えたワークベンチを提供することを目的としたツールである．Scarab は著者らが開発した Scala 上の制約プログラミング用ドメイン特化言語である Copris と同様のツールである．しかし Scarab では，SAT ソルバー以外の部分は，SAT 符号化の部分を含めてすべて Scala でコンパクトに記述されており変更が容易であるという特長を持つ．また SAT ソルバーとして利用している Sat4j に対する API を使用することにより，インクリメンタル解法などの SAT 技術を用いた高度な解法が実現可能である点も特長の一つである．本論文では，Scarab の構成とプログラム例を通してその機能と特長について説明を行う．

1　はじめに

命題論理の充足可能性判定 (**SAT**; Boolean (or propositional) satisfiability (testing)) は人工知能や計算機科学の分野において重要である [3] [8]．この十数年で SAT 問題を解くためのプログラムである **SAT ソルバー** の性能は飛躍的に向上しており [15]，このような性能向上は与えられた問題を SAT 問題に符号化し，SAT ソルバーを用いて解を求める **SAT 型 システム** の開発を促進してきた．例えば論理合成，プランニング，スケジューリング，ハードウェア・ソフトウェアの検証，制約充足問題などで SAT 型システムが成功をおさめている [8] [2] [22] [14]．

本論分では Scala 上で実装された SAT 型制約プログラミングシステムのための開発ツールである Scarab[†1] を提案する．Scarab は制約プログラミングのための **ドメイン特化言語** (**DSL**; Domain-specific Language) である Scarab DSL, SAT 符号化モジュール，そしてバックエンドの SAT ソルバーへのインターフェースから構成される．現在，Scarab では SAT 符号化として順序符号化 [19] ，SAT ソルバーには Sat4j [10] を利用可能である．

Scarab の設計方針は SAT 型システム開発者に表現性，効率性，変更性，可搬性を備えたワークベンチを提供することである．

表現性:　Scarab DSL と Scala の両方を用いて与えられた問題を記述することが可能である．

効率性:　Scarab は最適化された順序符号化法を用いているという点で効率的である [19]．順序符号化は 2008 年，2009 年に CSP ソルバー競技会のグローバル部門で優勝した Sugar [20] に採用されている．

変更性:　Scarab では SAT 型システム開発者が自前の制約を定義し，変更・改良することが可能である．また Scarab のソースコードは全体で 500 行ほどであり，Scarab 本体の変更も可能である．

可搬性:　Scarab と Sat4j は両方とも JVM 上で動作し，可搬性のあるシステムを実現可能である．

Scarab: A Prototyping Tool for SAT-based Constraint Programming Systems in Scala

Takehide Soh, Mutsunori Banbara, Naoyuki Tamura, 神戸大学情報基盤センター, Information Science and Technology Center, Kobe University.
Daniel Le Berre, Stéphanie Roussel, CRIL-CNRS, UMR 8188, Université d'Artois.

†1 http://kix.istc.kobe-u.ac.jp/~soh/scarab/

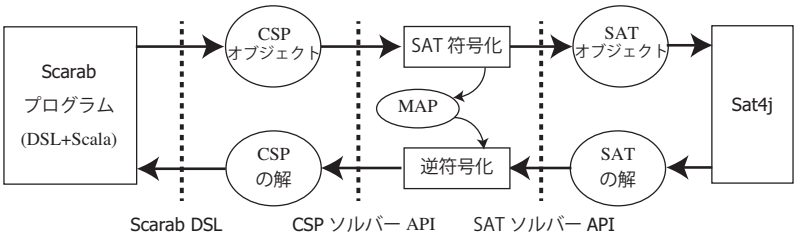

図 1　Scarab の構成

図 1 に Scarab の構成図を示す．まず SAT 型システム開発者が Scala と Scarab DSL を用いて記述したプログラム (Scarab プログラム) によって CSP オブジェクトが生成される．次に Scarab プログラムによって CSP ソルバーが求解のために API を通して呼ばれた時に，CSP オブジェクトは SAT オブジェクトへと変換される．続いて CSP ソルバーから SAT ソルバー Sat4j が API を通して実行される．解が存在すれば逆符号化を通して CSP の解が返される．

これまで SAT 型システムのための制約プログラミング DSL はいくつか提案されている: Copris (in Scala) [23], Numberjack (in Python) [7], Bee (in Prolog) [13], and B-Prolog (in Prolog) [28]．この中でも，著者らが開発した Copris は Scala 上の埋込み DSL を備えており Scarab と同様のツールである．Copris と Scarab の違いとして，Scarab は 500 行で実装されているなど，よりコンパクトで変更容易であることが挙げられる．また Scarab と SAT ソルバー Sat4j の親和性が高いことも Scarab の大きな特長である．第 4 節で説明されるようにこの高い親和性により高度な解法が利用可能になる．

以下では第 2 節で Scarab プログラムと二つの例を説明する．次に SAT 符号化モジュールについて第 3 節で述べる．第 4 節は Sat4j を用いた高度な解法について紹介し，第 5 節では汎対角線ラテン方陣を用いた性能評価を示す．

2　Scarab プログラム

Scarab を用いた SAT 型制約プログラミングシステム開発の最も基本的な方法は Scarab プログラムを記述することである．Scarab プログラムの記述には Scala と制約プログラミングのための DSL である Scarab DSL の両方を利用可能である．本節では Scala と Scarab DSL を説明した後，Scarab プログラムの二つの例について説明を行う．

2.1　Scala の概要

Scala は†2 Martin Odersky により設計された比較的新しいプログラミング言語で，Twitter の分散 DB フレームワークに用いられたことなどもあり，近年注目を集めている [16]．言語上の特長としては，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，型安全性，高階関数，不変コレクションなどが挙げられる．

処理系としては，JVM (Java Virtual Machine) へのコンパイラとインタラクティブな実行環境 (REPL; Read Eval Print Loop) が用意されている．Java との親和性は高く，Java のクラス・ライブラリをそのまま利用できる．

さらに Scala は，関数型言語およびオブジェクト指向言語としての高度な記述能力，リスト，マップ，集合等の豊富なコレクションフレームワーク，演算子の多重定義や柔軟な構文，オブジェクトのメソッドのインポート機能など，埋込みの **ドメイン特化言語** (**DSL**; Domain-specific Language) を実装するために適した機能を数多く備えている [12]．特に Scarab DSL ではケースクラス，暗黙変換，シングルトンオブジェクトおよびそのオブジェクトからのインポートなどの機能を利用しているが，これらの詳細については文献 [16] を参照されたい．

†2 http://www.scala-lang.org/

T ::= V | `-` T | T `+` *Int* | T `+` T | T `-` *Int* | T `-` T | T `*` *Int* | `Sum(`V, ...`)` | `Sum(`*Seq*[V]`)`

V ::= `Var(`*String*, *String*, ...`)` | V`(`*Any*, ...`)`

C ::= B | T *op* T | `!` C | C `&&` C | C `||` C |
`And(`C, ...`)` | `And(`*Seq*[C]`)` | `Or(`C, ...`)` | `Or(`*Seq*[C]`)` | `alldiff(`*Seq*(T, ...`))`

op ::= `<=` | `<` | `>=` | `>` | `===` | `!==`

B ::= `Bool(`*String*, *String*, ...`)` | B`(`*Any*, ...`)`

図 2　Scarab における制約の構文規則

2.2　Scarab DSL

Scarab DSL は Scala 上の埋込み DSL (embedded DSL) として実装されている．CSP における項 ($x+y$ 等の数式) および制約 ($(x>0)\vee(y>0)$ 等の論理式) を Scala 中で表現するために，次のようなクラスを定義する: `Term` (項を表すクラス), `Var` (整数変数を表すクラス), `Bool` (ブール変数を表すクラス), `Constraint` (制約を表すクラス)．これらのクラスは `jp.kobe_u.scarab.csp` パッケージに属している．

図 2 に Scarab DSL における制約の構文規則を示す．ここで T，V，C，B は上述の `Term`，`Var`，`Constraint`，`Bool` クラスを表す．また *Int*，*String*，*Seq* をそれぞれ Scala における整数 (Integer)，文字列 (String)，列 (Sequence) クラスとする．*Any* は全てのクラスの親クラスとする．論理演算子は`||` と `Or` が論理和，`&&` と `And` が論理積を表している．加えて比較演算子として`<=` ($\leq$), `<`, `>=` ($\geq$), `>`, `===` ($=$), `!==` ($\neq$) を用意している．

Scarab DSL における制約定義の例を Scala の REPL の実行例を通して説明する (`import jp.kobe_u.scarab.csp._` の実行を仮定している)．整数変数を表すクラス `Var` は，変数名に加え文字列のリストを添字として指定可能である．また，すでにある `Var` オブジェクトに任意の添字を追加した新しい `Var` オブジェクトを作成できるように，`apply` メソッドを定義している．

```
scala> val x = Var("x")
x: jp.kobe_u.scarab.csp.Var = x
scala> x(1,2)    // x.apply(1,2) と同じ
res0: jp.kobe_u.scarab.csp.Var = x(1,2)
```

これらの整数変数を用いた制約 $x\leq 1\vee x_{1,2}\leq 1$ の記述例を以下に示す．

```
scala> x <= 1 || x(1,2) <= 1
res1: jp.kobe_u.scarab.csp.Or =
 Or(LeZero(Sum(-1+x)),LeZero(Sum(-1+x(1,2))))
```

ここで注意されたいのは Scarab では制約中の線形制約は定義されるとまず $\sum_i a_i x_i - c\leq 0$ (但し a_i は非零の整数 c は整数値，x_i は整数変数) の形に変換されるということである．線形制約のクラスは `LeZero` である．上記の例では $x\leq 1\vee x_{1,2}\leq 1$ は $x-1\leq 0\vee x_{1,2}-1\leq 0$ に変換される．

この他に Scarab では Scala の暗黙変換を利用してシンボル (' で始まる記号) を整数変数として利用できるようになっており，次のように簡潔に制約を記述できる．

```
scala> 'y <= 0 || 'y(1,2) <= 0
res2: jp.kobe_u.scarab.csp.Or =
 Or(LeZero(Sum(+y)),LeZero(Sum(+y(1,2))))
```

最後に Scarab DSL を用いた CSP の定義例を以下に示す．

```
// CSP の生成
scala> val csp = CSP()
// 整数変数 x, y ∈ {1, ..., 3} の追加
scala> csp.int('x, 1, 3)
scala> csp.int('y, 1, 3)
// 制約 x + y ≤ 2 の追加
scala> csp.add('x + 'y <= 2)
```

Scarab では CSP の解を計算するために CSP ソルバーのクラス `Solver` を用意している．`Solver` は `jp.kobe_u.scarab.solver` パッケージに属しており，CSP の解を計算する `find` メソッドを提供している．`find` は与えられた CSP を SAT に符号化し，SAT ソルバーを呼び出すことで解の計算を行う．Scarab における SAT 符号化については第 3 節で説明を行う．

```
 1:  val n = 15; val s =36
 2:
 3:  for (i <- 1 to n) {
 4:    int('x(i),0,s-i)
 5:    int('y(i),0,s-i)
 6:  }
 7:
 8:  for (i <- 1 to n; j <- i+1 to n)
 9:    add(('x(i) + i <= 'x(j)) ||
10:        ('x(j) + j <= 'x(i)) ||
11:        ('y(i) + i <= 'y(j)) ||
12:        ('y(j) + j <= 'y(i)))
13:
14:  if (find) println(solution)
```

図 3 $SP(15,36)$ **の Scarab プログラム**

2.3 Scarab プログラム例

本節では Scala と Scarab DSL を用いたプログラム例を説明する．なお本節のプログラム例では次のインポート文の実行を仮定している．

```
import jp.kobe_u.scarab.csp._
import jp.kobe_u.scarab.solver._
import jp.kobe_u.scarab.sapp._
```

ここで `jp.kobe_u.scarab.sapp` は Scarab のアプリケーションのためのシングルトンオブジェクトであり，CSP，CSP ソルバーそれぞれのデフォルトオブジェクトや，それらに対する `int`, `add`, `find` 等のメソッドを提供している．これによりデフォルトオブジェクトに対する操作を簡潔に記述可能となる．

2.3.1 正方形詰込み問題

一つめの例は正方形詰込み問題を解くプログラムである．正方形詰込み問題 $SP(n,s)$ は一辺の長さ 1 から n まで 1 ずつ増加する正方形の集合を一辺の長さ s の正方形の枠内に重なりなく配置する問題である．最も素直なモデリングは整数変数 $x_i, y_i \in \{0,\ldots,s-i\}$ をそれぞれの正方形 $i (1 \le i \le n)$ に $(x_i,\ y_i)$ が正方形 i の左下の座標を指すようにするものである．以下の制約は任意の二つの正方形 i と j (但し $1 \le i < j \le n$) が重なることを禁止する．

$$\begin{array}{ll} (x_i + i \le x_j) & \vee \\ (x_j + j \le x_i) & \vee \\ (y_i + i \le y_j) & \vee \\ (y_j + j \le y_i) & \end{array} \qquad (1)$$

図 3 に $SP(15,36)$ の時の Scarab プログラムを示す．Scarab DSL を用いることで簡潔に上記のモデルを表現できている．4，5 行目の `int` メソッドは整数変数をデフォルト CSP に定義している．ここで `'x`，`'y` は Scala におけるシンボルオブジェクトを表しており，前述のように Scala の暗黙変換により整数変数オブジェクトへと変換される．9 から 12 行目の `add` メソッドは式 (1) の制約を定義している．14 行目の `find` メソッドは，定義された CSP を SAT へと符号化した後に解を計算，逆符号化している．逆符号化された解は `solution` によって返され, Scala の `println` メソッドにより出力される．

```
 1:  val n: Int = 5
 2:  val elems = (0 until n)
 3:
 4:  for (i <- elems; j <- elems)
 5:    int('x(i,j),1,n)
 6:  for (i <- elems) {
 7:    add(alldiff(elems.map(j => 'x(i,j))))
 8:    add(alldiff(elems.map(j => 'x(j,i))))
 9:    add(alldiff(
10:      elems.map(j => 'x(j,(i+j+n-1)%n))))
11:    add(alldiff(
12:      elems.map(j => 'x(j,(i-j+n-1)%n))))
13:  }
14:
15:  if (find) println(solution)
```

図 4 $LS(5)$ **の Scarab プログラム**

2.3.2 汎対角線ラテン方陣

もう一つの例は CSP ソルバー競技会 [11] で使用された汎対角線ラテン方陣 (Pandiagonal Latin Square) である．汎対角線ラテン方陣 $LS(n)$ は n 行 n 列の行列に 1 から n までの n 個の異なる整数を，各整数が各行，各列，各汎対角線に 1 回だけ現れるように配置する問題である．いま $LS(n)$ が与えられたとき，整数変数の n 行 n 列の行列 $x_{i,j} \in \{1,\ldots,n\}$ $(1 \le i,j \le n)$ を用いてモデリングを行う．各整数が 1 回だけ現れる制約は *alldiff* 制約 [17] によって表す．この制約は制約プログラミングの分野で最もよく知られているグローバル制約の一つであり，与えられた n 個の整数変数が互いに異なることを意味する [26].

図 4 は $LS(5)$ に対する Scarab プログラムを表している．Scarab DSL は *alldiff* を用いたモデリングを Scala の特長を用いて簡潔に表している．例えば 7 行目の `alldiff(elems.map(j => 'x(i,j)))` は各行においてそれぞれの変数が異なることを表す制約 $\mathit{alldiff}(x_{i,1}, x_{i,2}, \ldots, x_{i,n})$ を表している．

$$\sum_{i=1}^{n} a_i x_i \le c = \begin{cases} x_1 \le \lfloor c/a_1 \rfloor & (n=1, a_1 > 0) \quad (2a) \\ \neg\,(x_1 \le \lceil c/a_1 \rceil - 1) & (n=1, a_1 < 0) \quad (2b) \\ \bigwedge_{b=lb(x_1)}^{ub(x_1)} \left((x_1 \le b-1) \vee \sum_{i=2}^{n} a_i x_i \le c - a_1 b \right) & (n \ge 2, a_1 > 0) \quad (2c) \\ \bigwedge_{b=lb(x_1)}^{ub(x_1)} \left(\neg(x_1 \le b) \vee \sum_{i=2}^{n} a_i x_i \le c - a_1 b \right) & (n \ge 2, a_1 < 0) \quad (2d) \end{cases}$$

図 5　順序符号化における変換

3　Scarab における SAT 符号化

Scarab では与えられた CSP を，前処理により一旦 CSP の CNF 式に変換している．CSP の CNF 式におけるリテラルは，ブール変数，ブール変数の否定，$\sum_{i=1}^{n} a_i x_i \le c$ の形の線形制約のいずれかである．ここで a_i は非零の整数, c は整数値, そして x_i は整数変数を表す．なお変換は Tseitin 変換 [25] [21] を用いて新しいブール変数を導入する方法で行っている．以下では Scarab で採用している SAT 符号化である順序符号化とその実装について概要を説明する．

3.1　順序符号化

順序符号化法 [19] ではブール変数 $p(x \le c)$ がそれぞれの整数変数 x と整数値 $c \in \{lb(x)-1, \ldots, ub(x)\}$ に対して導入される．ここで $lb(x)$ と $ub(x)$ はそれぞれ x の下限値と上限値である．さらに以下の整数変数における値の順序関係を表す制約が導入される．

$$\bigwedge_{c=lb(x)+1}^{ub(x)-1} (\neg p(x \le c-1) \vee p(x \le c))$$

順序符号化法では線形制約 $\sum_{i=1}^{n} a_i x_i \le c$ を符号化するために図 5 に示す変換式を用いる．この変換を再帰的に適用し，変換式 (2a) と (2b) の $x_1 \le \lfloor c/a_1 \rfloor$ と $x_1 \le \lceil c/a_1 \rceil - 1$ をそれぞれブール変数 $p(x_1 \le \lfloor c/a_1 \rfloor)$ と $p(x_1 \le \lceil c/a_1 \rceil - 1)$ に変換することで最終的に線形制約はブール変数上の連言標準形式へと変換される．

3.2　順序符号化の実装

図 6 は図 5 の変換を実装したプログラムを示している．1 行目は二つの入力を表しており，$\{(a_1, x_1), \ldots, (a_n, x_n)\}$ と整数値 c である．5，6 行目はそれぞれ変換式 (2a) と (2b) を表している．9 から 12 行目は変換式 (2c) を表している．11 行目では $\{(a_2, x_2), \ldots, (a_n, x_n)\}$ と $c - a_1 b$ を入力として `encodeLe` を再帰的に呼び出している．14 から 17 行目は同様に変換式 (2d) を表している．

```
 1:  def encodeLe(axs: Seq[(Int,Var)], c: Int):
 2:          Seq[Seq[Literal]] = axs match {
 3:
 4:    case Seq((a,x)) =>
 5:      if (a > 0) Seq(Seq(le(x, floorDiv(c, a))))
 6:      else Seq(Seq(le(x, ceilDiv(c, a)-1).neg))
 7:    case Seq((a,x), axs1 @ _*) => {
 8:      if (a > 0) {
 9:        for {
10:          b <- lb(x) to ub(x)
11:          clause <- encodeLe(axs1, c-a*b)
12:        } yield le(x, b-1) +: clause
13:      } else {
14:        for {
15:          b <- lb(x) to ub(x)
16:          clause <- encodeLe(axs1, c-a*b)
17:        } yield le(x, b).neg +: clause
18:      }
19:    }
20:  }
```

図 6　順序符号化の実装

例として正方形詰込み問題 $SP(2,6)$ の制約の一部である制約 $x_1 + 1 \le x_2$ を考える．ここで $x_1 \in \{0, \ldots, 5\}$ と $x_2 \in \{0, \ldots, 4\}$ とする．まず初めに $x_1 + 1 \le x_2$ は線形制約 $x_1 - x_2 \le -1$ に変形される．図 6 の `encodeLe` を入力 $\{(1, x_1), (-1, x_2)\}$ と -1 に入力として呼ぶことにより, 以下の順序符号化された CNF 式が得られる:

$$\begin{gathered} \neg p(x_2 \le 0) \ \wedge \\ (p(x_1 \le 0) \vee \neg p(x_2 \le 1)) \ \wedge \\ (p(x_1 \le 1) \vee \neg p(x_2 \le 2)) \ \wedge \\ (p(x_1 \le 2) \vee \neg p(x_2 \le 3)) \ \wedge \\ p(x_1 \le 3) \ \wedge \ p(x_1 \le 4) \end{gathered}$$

```
 1:  // CSP 定義
 2:  int('x, 1, 3) // x ∈ {1,2,3}
 3:  int('y, 1, 3) // y ∈ {1,2,3}
 4:  add(x === y)  // 制約の定義 x = y
 5:
 6:  // 4.1 インクリメンタル解法
 7:  find          // 充足可能: x = 3, y = 3
 8:  add(x !== 3)  // 制約の追加
 9:  find          // 充足可能: x = 2, y = 2
10:
11:  // 4.2 Assumption を用いた解法
12:  find(y === 3) // 充足不能
13:  find(x === 1) // 充足可能: x = 1, y = 1
14:
15:  // 4.3 コミットとロールバック
16:  commit        // コミットポイント生成
17:  add(x < y)    // x < y が追加される
18:  find          // 充足不能
19:  rollback      // x < y の追加をロールバック
20:  find          // 充足可能: x = 2, y = 2
```

図 7　Sat4j を用いた高度な解法の例

実際の Scarab 中の encodeLe の実装も 25 行以内とコンパクトであり，SAT 型システム開発者が SAT 符号化を変更し易いようになっている．また Scarab の符号化インターフェースを用いてこれまで提案されてきた他の SAT 符号化を実装することも可能である [9] [27] [6] [5] [1] [4] [24]．この際には整数変数と制約の SAT 符号化および逆符号化メソッドの実装が必要になる．

4　Sat4j を用いた高度な解法

Scarab では Sat4j をデフォルトの SAT ソルバーとして採用している．Scarab と Sat4j は共に JVM 上で実行可能であり，

Sat4j は Scarab から外部プロセスの起動なしに直接実行可能であり，このような SAT ソルバーとの融合およびこれによる高度な解法は他の SAT 型制約プログラミングツールにはない Scarab の特長になっている．本節では図 7 に示すプログラム例を用いて Sat4j に実装されている機能を利用した Scarab の解法について説明を行う．

4.1　インクリメンタル解法

図 7 の 6〜9 行目に記載されるように, Scarab では一度解を計算した後に制約の追加が可能である．7 行目の 1 回目の find メソッドでは定義された CSP 全体が SAT 符号化され，生成された節集合が Sat4j へと追加される．SAT ソルバー Sat4j が求解を行う．9 行目の 2 回目の find メソッドでは 8 行目で追加された制約 $x \neq 3$ のみが符号化され，節集合が Sat4j に追加される．そして Sat4j が再び求解を行う．

ここで 1 回目の find メソッドによって生成された学習節は 2 回目の呼び出し時にも保持されており，学習節の再利用による効果的な解探索を期待できる．但し，1 回目の CSP が充足不能の場合には制約の追加は許可されないので注意されたい．

4.2　仮説を用いた解法

Scarab では制約を引数に持つ find(assump: Constraint) メソッドを用いる仮説 (Assumption) に基づく CSP の解探索が可能である．但し，現状は仮説に指定する制約はブール変数上のリテラルの連言に符号化される必要があり，それ以外の制約が指定された場合には例外が発生する．それぞれのリテラルは Sat4j へと渡されこの仮説リテラル集合に基づく SAT の解探索が Sat4j で行われる．

図 7 の例では，12 行目で仮説 $y = 3$ のもとで CSP の求解が行われている．しかし，この仮説はこれまで追加された制約 $(x = y) \wedge (x \neq 3)$ と矛盾するので充足不能が返される．続いて仮説 $x = 1$ のもとで CSP の求解が行われ，この場合には充足可能となる．

このように仮説で指定した制約は CSP に永続的に追加されるのではなく，求解中にのみ有効であり，また探索中に得られた学習節も保持される．後述するように，この仮説を用いた解法は最適解のデクリメンタル探索や二分法へと応用することができる．

4.3　制約のコミットとロールバック

Scarab では以下のような制約のコミット (commit) とロールバック (rollback) メソッドを提供している．

- commit メソッドは現在の CSP の状態 (整数変数，ブール変数，制約の数) を記録する．現在の Scarab の実装では一つのコミットポイントを作成可能である．
- rollback メソッドは CSP を最後のコミットポイントの状態まで戻す．同時に Sat4j の状態も

```
 1:  val lb = n
 2:  var ub = s
 3:  int('m, lb, ub)
 4:
 5:  for (i <- 1 to n)
 6:    add(('x(i)+i <= 'm) && ('y(i)+i <= 'm))
 7:
 8:  while (lb <= ub && find('m <= ub)) {
 9:    add('m <= ub)
10:    ub = solution.intMap('m) - 1
11:  }
12:
13:  while (find)
14:       println(solution)
```

図 8　仮説を用いたデクリメンタル探索

reset メソッドにより初期化されるので次回の find メソッド呼び出し時には Sat4j へと節を追加し直す必要がある．

これら commit/rollback メソッドを実行することで，これまでに追加した制約の削除が可能となり，動的な制約の変更が必要なアプリケーションに対して，より柔軟な解探索を提供することができると考えられる．しかし，符号化の手続きは繰り返し行う必要があり，CSP が充足不能になった場合には学習節は再利用できないので注意が必要である．

図 7 の例では，16 行目でコミットポイントが生成され，17 行目で制約 $x < y$ が追加されている．しかし，この制約はこれまで追加された制約 $(x = y)$ と矛盾するので充足不能が返される．次に 19 行目で rollback メソッドが呼ばれ CSP が 16 行目の状態まで戻される．すなわち制約 $(x = y)$ が削除される．20 行目で再び求解が行われ，この場合には CSP は充足可能となる．

4.4　最適値の探索

本節では上述の 3 つの機能を用いた応用として解の最適化を紹介する．例として，図 3 に示した正方形詰込み問題 $SP(n, s)$ について，制約を充足するような最小の正方形の枠の大きさを計算する．

図 8 は仮説を用いたデクリメンタル探索のプログラムを示している．このプログラムは図 3 のプログラムの最下部に追加することで実行可能であり，プログラム中の Scala の整数変数 n と s は図 3 で定義されているものである．

まずプログラム 3 行目の整数変数 $m \in \{lb, \ldots, ub\}$ は正方形の枠のサイズを表している．5，6 行目で定義される制約は全ての正方形がサイズ m の枠をはみ出ないことを保証する．8 行目では find メソッドが仮説 $m \leq ub$ とともに呼ばれている．もし解が存在するならば，制約 $m \leq ub$ が追加され，ub が m の最も最近の値から 1 つ小さい値に更新される (10 行目)．13，14 行目では全ての最適解が列挙されている．Scarab では，find メソッドが同じ CSP に対して呼ばれると 2 回目からは最後に得られた解の否定をブロック節として加えることで別の解の探索を行うようになっている．

```
 1:  var lb = n
 2:  var ub = s
 3:  commit
 4:
 5:  while (lb < ub) {
 6:    var size = (lb + ub) / 2
 7:    for (i <- 1 to n)
 8:      add(('x(i)+i<=size)&&('y(i)+i<=size))
 9:    if (find) {
10:      ub = size
11:      commit
12:    } else {
13:      lb = size + 1
14:      rollback
15:    }
16:  }
```

図 9　commit/rollback メソッドを用いた二分探索

図 9 は commit/rollback メソッドと制約の追加を用いた二分探索を示している．6 行目は現在の下限と上限のちょうど半分の値を計算している．7，8 行目は正方形の枠の大きさを制限する制約を追加している．9 行目では，find メソッドが呼ばれている．もし CSP が充足可能である場合，上限が更新され現在の CSP がコミットされる (10，11 行目)．もし充足不能である場合，下限が 1 だけ増加され最後のコミットポイントへと CSP がロールバックされる．

5　汎対角線ラテン方陣を用いた性能評価

Scarab の基本性能を評価するために図 4 で示した汎対角線ラテン方陣を用いて計算機実験を行った．ここでは二つの *alldiff* 制約の実装を用いた．一つめは naive 版で *alldiff* 制約の定義に従い与えられた n 個の変数 $x_1, \ldots, x_n$ の入力に対し，$\bigwedge_{1 \leq i < j \leq n}(x_i \neq x_j)$ なる制約を定義するものである．二つめは optimized 版で，順列制約 $\bigwedge_{i=lb}^{ub} \bigvee_{j=1}^{n}(x_j = i)$ と鳩ノ巣原理の制約 $\neg \bigwedge(x_i < lb + n - 1)$ and $\neg \bigwedge(x_i > ub - n + 1)$

表 1　汎対角線ラテン方陣における性能評価 (CPU 時間: 秒)

n	3 UNSAT	4 UNSAT	5 SAT	6 UNSAT	7 SAT	8 UNSAT	9 UNSAT
alldiff (naive)	0.164	0.153	0.183	0.398	0.210	T.O.	T.O.
alldiff (optimized)	0.230	0.209	0.236	0.264	0.221	0.212	0.235

n	10 UNSAT	11 SAT	12 UNSAT	13 SAT	14 UNSAT	15 UNSAT	16 UNSAT
alldiff (naive)	T.O.	0.347	T.O.	T.O.	T.O.	T.O.	T.O.
alldiff (optimized)	0.370	0.332	0.981	0.545	9.792	389.917	458.187

を naive 版に加えたものである．この鳩の巣原理の制約の効果は文献[18] で報告されている．なお naive 版は Scarab では 2 行で実装されており optimized 版でも 15 行程度である．このように制約の実装を簡潔に記述でき，改良したものを実験できることは Scarab で SAT 型システム開発を行う利点の一つである．

変換制限時間は 1 時間で，全ての計算時間は Xeon 2.93GHz の CPU，JVM に対して 2GB のメモリを割当てた MacOS X 上で行っている．参考として 2009 年に開催された CSP ソルバー競技会では $n \leq 12$ までを解いた Sugar を除くと $n > 8$ の $LS(n)$ をどの CSP ソルバーも解くことができなかった．

表 1 は $LS(n)$ (但し $3 \leq n \leq 16$) に対する Scarab の計算時間 (CPU 時間, 秒) を示している．optimized 版 *alldiff* 制約を用いたものは$n=16$までを解くことに成功している．

6　おわりに

この論文では Scala 上に実装された SAT 型制約プログラミングシステム開発ツールである Scarab の説明を行った．正方形詰込み問題と汎対角線ラテン方陣の二つの例によって示されるように Scarab DSL と Scala の特長を利用することで SAT 型システム開発者は簡潔に問題のモデリングを行うことができる．

また Sat4j の機能を用いることで, Scarab は次の機能を提供する: インクリメンタル探索; 仮説を用いた CSP の解探索; 制約のコミットとロールバック．これらの機能は Scarab において最適化や解列挙などの高度な機能を実装するのに使うことができる．

Scarab の興味深い拡張の一つとして Sat4j の機能をさらに導入することがある．例えば Sat4j は与えられた CNF 式が充足不能である時に Minimal Unsatisfiable Subformula (MUS) を計算することが可能である．加えて埋め込みの最適化メソッドおよび解列挙メソッド，また基数制約や擬似ブール制約なども入力として扱うことが可能である．Scarab が持つ制約表現および SAT 符号化モジュールとこれら Sat4j の機能の組合せは SAT 技術の応用を広げることを期待できる．Scarab の情報とソースコードは以下から入手できる．
http://kix.istc.kobe-u.ac.jp/~soh/scarab/.

参 考 文 献

[1] Ansótegui, C. and Manyà, F.: Mapping Problems with Finite-Domain Variables into Problems with Boolean Variables, *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004), LNCS 3542*, 2004.

[2] 番原睦則, 田村直之: SAT によるシステム検証, 人工知能学会誌, Vol. 25, No. 1(2010).

[3] Biere, A., Heule, M., van Maaren, H., and Walsh, T.(eds.): *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications (FAIA), Vol. 185, IOS Press, 2009.

[4] Gavanelli, M.: The Log-Support Encoding of CSP into SAT, *Proceedings of the 13th International Joint Conference on Principles and Practice of Constraint Programming (CP 2007), LNCS 4741*, 2007, pp. 815–822.

[5] Gent, I. P. and Nightingale, P.: A New Encoding of Alldifferent into SAT, *Proceedings of the 3rd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, 2004.

[6] Gent, I. P.: Arc Consistency in SAT, *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, 2002, pp. 121–125.

[7] Hebrard, E., O'Mahony, E., and O'Sullivan, B.: Constraint Programming and Combinatorial Optimisation in NumberJack, *Proceedings of CPAIOR*,

2010, pp. 181–185.
[8] 井上克巳, 田村直之: SAT ソルバーの基礎, 人工知能学会誌, Vol. 25, No. 1(2010).
[9] Iwama, K. and Miyazaki, S.: SAT-Variable Complexity of Hard Combinatorial Problems, *Proceedings of the IFIP 13th World Computer Congress*, 1994, pp. 253–258.
[10] Le Berre, D. and Parrain, A.: The Sat4j library, release 2.2, *JSAT*, Vol. 7, No. 2-3(2010), pp. 59–6.
[11] Lecoutre, C., Roussel, O., and van Dongen, M. R. C.: Promoting robust black-box solvers through competitions, *Constraints*, Vol. 15, No. 3(2010), pp. 317–326.
[12] Mernik, M., Heering, J., and Sloane, A. M.: When and how to develop domain-specific languages, *ACM Comput. Surv.*, Vol. 37, No. 4(2005), pp. 316–344.
[13] Metodi, A. and Codish, M.: Compiling Finite Domain Constraints to SAT with BEE, *Theory and Practice of Logic Programming*, Vol. 12, No. 4-5(2012), pp. 465–483.
[14] 鍋島英知: SAT によるプランニングとスケジューリング, 人工知能学会誌, Vol. 25, No. 1(2010).
[15] 鍋島英知, 宋剛秀: 高速 SAT ソルバーの原理, 人工知能学会誌, Vol. 25, No. 1(2010).
[16] Odersky, M., Spoon, L., and Venners, B.: *Programming in Scala*, Artima, Inc., second edition, 2010.
[17] Régin, J.-C.: A Filtering Algorithm for Constraints of Difference in CSPs, *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, 1994, pp. 362–367.
[18] 田島宏史: SAT 変換に基づく制約ソルバーの高速化に関する研究, 修士論文, 神戸大学大学院自然科学研究科, 2006.
[19] Tamura, N., Taga, A., Kitagawa, S., and Banbara, M.: Compiling Finite Linear CSP into SAT, *Constraints*, Vol. 14, No. 2(2009), pp. 254–272.
[20] Tamura, N., Tanjo, T., and Banbara, M.: System Description of a SAT-based CSP Solver Sugar, *Proceedings of the 3rd International CSP Solver Competition*, 2008, pp. 71–75.
[21] 田村直之, 丹生智也, 番原睦則: SAT 変換に基づく制約ソルバーとその性能評価, コンピュータソフトウェア, Vol. 27, No. 4(2010), pp. 183–196.
[22] 田村直之, 丹生智也, 番原睦則: 制約最適化問題と SAT 符号化, 人工知能学会誌, Vol. 25, No. 1(2010).
[23] 田村直之, 丹生智也, 番原睦則: Scala 上の制約プログラミング用ドメイン特化言語 Copris について, コンピュータソフトウェア, Vol. 29, No. 4(2012), pp. 114–129.
[24] 丹生智也, 田村直之, 番原睦則: 位取り記数法に基づく整数有限領域上の制約充足問題のコンパクトかつ効率的な SAT 符号化, コンピュータソフトウェア, Vol. 30, No. 1(2013), pp. 211–230.
[25] Tseitin, G. S.: On the complexity of derivations in the propositional calculus, *Studies in Mathematics and Mathematical Logic Part II*, (1968), pp. 115—–125.
[26] van Hoeve, W.-J. and Katrie, I.: Global Constraint, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, Elsevier, 2006, pp. 169–208.
[27] Walsh, T.: SAT v CSP, *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP 2000)*, 2000, pp. 441–456.
[28] Zhou, N.-F.: The language features and architecture of B-Prolog, *Theory and Practice of Logic Programming*, Vol. 12, No. 1-2(2012), pp. 189–218.