第57回 プログラミング･シンポジウム 2016.1.8-10

SATソルバーを用いた制約プログラミングシステムとその応用

宋 剛秀[1,a)]

概要：近年 SAT ソルバーの求解性能は飛躍的に向上しており，様々な分野で応用が進んでいる．しかし，SAT ソルバーは連言標準形の命題論理式を入力としており，算術制約を含むような問題を直接記述して解くことには向いていない．このため，より表現力のある入力形式に対応できるように SAT ソルバーを利用・拡張したシステムが研究されている．本稿で説明する SAT 型制約プログラミングシステム Scarab は，制約充足問題 (CSP) を簡潔に表現できる高い記述性，CSP を SAT に符号化した後，密に連携した SAT ソルバーを実行することによる高い求解性をもっている．本稿では Scarab の設計方針と制約記述のためのドメイン特化言語を説明した後，いくつかのプログラム例を通して Scarab を用いたアプリケーション開発の利点を明らかにする．

キーワード：SAT ソルバー，制約プログラミング，ドメイン特化言語

1. はじめに

命題論理の充足可能性判定 (**SAT**; propositional satisfiability) 問題は計算理論において中心的であり，人工知能や計算機科学の分野においても重要な問題である [3], [7]．2000 年以降 SAT 問題を解くためのプログラムである **SAT** ソルバーの性能が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．これまで論理合成，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などで SAT 型システムが成功をおさめている [1], [7], [14], [24]．特に近年の話題として，SAT ソルバーはインテル社のコア i7 プロセッサの設計 [8]，統合開発環境 Eclipse のコンポーネントの依存性解析 [10]，ラムゼー数の下界更新 [5] などにも使われている．また Knuth による “The Art of Computer Programming” の最新刊である 4B 巻 [9] (2015 年 12 月出版予定) では SAT が 318 ページにわたって取り上げられており，序文には「SAT 問題は，非常に多くの問題を解くためのキーであることから，明らかに “killer app” である」と述べられている．

SAT 型システムは効果的な方法であるが，与えられた問題に対して提案した解法を毎回はじめから実装するのは効率が悪い．そこで制約充足問題 (**CSP**; constraint satisfaction problem) などの汎用的な形式で問題を記述して解くことができる **SAT** 型制約プログラミングシステムが開発されてきた [6], [13], [23], [25], [27].

[1] 神戸大学情報基盤センター
Information Science and Technology Center, Kobe University
[a)] soh@lion.kobe-u.ac.jp

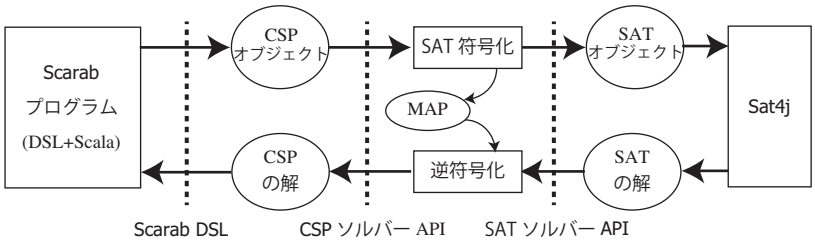

図 1 Scarab の構成

例えば Sugar[23] は専用の制約言語で制約充足問題 (**CSP**; constraint satisfaction problem) を記述し，順序符号化 [22] によって SAT 問題へと符号化した後に SAT ソルバーによって解を求める SAT 型制約プログラミングシステムである．2008 年と 2009 年に CSP ソルバー競技会 [11] のグローバル制約部門で優勝し，また近年では 2013 年にパッキング配列の最良解を更新するのに用いられた [17]. Sugar は専用の記述言語を用いることでユーザにとって理解し易い制約表現を可能にしているが，一方でループなどの汎用プログラミング言語で提供されている制御構造を扱うことができない．また SAT ソルバーとの連携はファイルを介して外部プロセスを起動することによって行なうなど SAT ソルバーをブラックボックスとして扱っている．これはいろいろな種類の SAT ソルバーをコストをかけずに切り替えることが可能であり，利点の一つでもあるが SAT ソルバー内部の低レベル機能の利用と拡張には対応することが難しくなる．

本稿では汎用プログラミング言語である Scala 上で実装された SAT 型制約プログラミングシステムである Scarab を紹介する．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 [22] ，SAT ソルバーには Sat4j[2] 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 [22]．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[23] 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．特に SAT 符号化の中心部分の実装は 25 行ほどで行われている．

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．

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．解が存在すれば復号化を通して CSP の解が返される．

汎用プログラミング言語を制約記述に用いる SAT 型制約プログラミングシステムの従来研究として Copris (in Scala) [25], Numberjack (in Python) [6], Bee (in Prolog) [13], B-Prolog (in Prolog) [27] が挙げられる．この中でも, 著者らが開発した Copris は Scala 上の埋込み DSL を備えている点で Scarab

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

と同様のツールである．Copris では Sugar を SAT 符号化モジュールとして用いており，SAT ソルバーを外部プロセスとして起動するなど SAT ソルバーとの連携は疎に行われる．一方，Scarab は，SAT 符号化，SAT ソルバーのグラスボックス化を設計方針としており，SAT 符号化モジュールを含めて Scala 上で 1000 行程度で実装されている．これによりコードはコンパクトで変更容易であり，新しいアイデアを実験し易い．また SAT ソルバーとの密な連携によりインクリメンタル解法 [4], [16] や組込み制約 [20] など SAT ソルバーの低レベル機能の利用ができる点，それらを Scala 上で拡張できる点において Copris と異なるなるツールになっている．

以下では 2 節で Scarab の入力にあたる Scarab プログラムを説明する．3 節ではと三つの問題記述例を説明する．次に 4 節では Scarab の特長である Sat4j を用いた高度な解法について説明を行う，5 節では性能評価を示す．

2. Scarab プログラム

Scarab を利用するため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，三つの問題記述例を通して Scarab の基本的機能の説明を行う．

2.1 Scala の概要

Scala [*1] は 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 [18]. 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 [18] を参照されたい．

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` 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,

[*1] `http://www.scala-lang.org/`

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．なお以下の REPL 実行例では，`import jp.kobe_u.scarab._` の実行を仮定していることに注意されたい．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.Var = x
scala> x(1,2)      // x.apply(1,2) と同じ
res0: jp.kobe_u.scarab.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.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.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 ∈ {1, ..., 3} の追加
scala> csp.int('x, 1, 3)
// 整数変数 y ∈ {1, ..., 3} の追加
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` 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．

以上 Scarab DSL について説明を行った．Scarab プログラムはこの Scarab DSL と Scala の両方を用いて記述される．ループはもちろんのこと Scala の全ての機能を用いたプログラミングが可能であり，CSP を簡潔に記述することができる．

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._
import dsl._
```

ここで `jp.kobe_u.scarab.dsl._` 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 および 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．

3.1 グラフ彩色問題

一つ目の例はグラフ彩色問題 (GCP) である．ここで $G = (V, E)$ をグラフとする．但し V は n 個の頂点の集合であり，E は辺の集合であり，m は許容する最大の彩色数を表すものとする．GCP は許容される数の色を用いて，必ず隣接頂点が異なる色になるように彩色する問題である．GCP が与えら

```
 1: val nodes = Seq(1,2,3,4,5)
 2: val edges = Seq((1,2),(1,5),(2,3),(2,4),(3,4),(4,5))
 3: var maxColor = 4
 4:
 5: for (i <- nodes)
 6:   int('x(i),1,maxColor)
 7: for ((i,j) <- edges)
 8:    add('x(i) !== 'x(j))
 9:
10: if(find) println(solution)
```

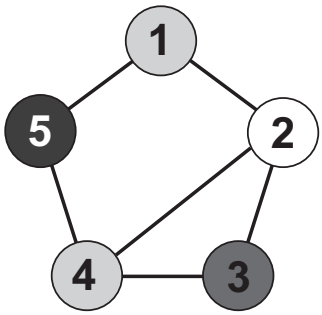

図 3　グラフ彩色問題例の Scarab プログラムと計算された解

```
 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)
```

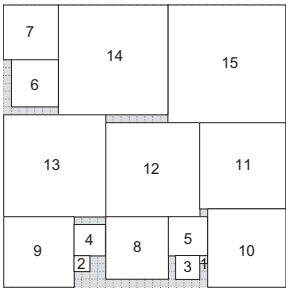

図 4　$SP(15, 36)$ の Scarab プログラムと計算された解

れた時に素直な制約モデルは各頂点 i $(1 \leq i \leq n)$ の色を表す整数変数 $x_i \in \{1, \ldots, m\}$ を用いる方法である．次に全ての隣接する頂点の対 $\{i, j\} \in E)$ に対して制約 $x_i \neq x_j$ を定義する．

図 2.2 の中で 1 行目と 2 行目は例として与えられた同じ図中のグラフにおける頂点と辺を定義している．3 行目は彩色数を 4 として定義している．以上によりグラフと彩色数の定義が完了したので，次は制約モデルである．まず初めに 5，6 行目では上述した各頂点の色を表す整数変数を定義している．ここで jp.kobe_u.scarab.dsl._のインポートにより，明示的に指定しない限りは，デフォルトの CSP オブジェクト csp にこれらの整数変数が追加されることに注意されたい．7，8 行目は同様に隣接頂点の色が異なるという制約を各辺に対して追加している．10 行目では find によって解の計算を行い，解が存在する場合は標準出力に印字する．このように Scarab では整数変数と制約を非常に簡潔に定義することができ，また SAT 符号化や SAT 解法の部分についてはバックエンドで処理されるため，ユーザは専用の符号化プログラムを記述する事無く簡単に SAT 技術を利用することが可能となる．

3.2　正方形詰込み問題

次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

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

2	3	5	1	4
5	1	4	2	3
4	2	3	5	1
3	5	1	4	2
1	4	2	3	5

図 5 $PLS(5)$ の Scarab プログラムと計算された解

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 $(1 \leq i \leq n)$ 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 $1 \leq i < j \leq n$) が重なることを禁止する．

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

図 4 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 行目から 12 行目の `add` メソッドは上述した式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．

3.3 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 $x_{i,j} \in \{1, \ldots, n\}$ $(1 \leq i, j \leq n)$ 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 [19] 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 5 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 $alldiff(x_{i,1}, x_{i,2}, \ldots, x_{i,n})$ を表している．

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，

```
 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 追加をロールバック
20:  find          // 充足可能: x = 2, y = 2
```

図 **6** Sat4j を用いた高度な解法の例

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 6 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 6 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 6 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 `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

```
 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 (findNext)
14:       println(solution)
```

図 7 仮説を用いたデクリメンタル探索

```
 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 ; commit
11:    } else {
12:      lb = size + 1 ; rollback
13:    }
14:  }
```

図 8 commit/rollback を用いた二分探索

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 6 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 4 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 7 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 4 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 4 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．findNext 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．

図 8 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 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 (12 行目)．

5. 性能評価

Scarab の基本性能を評価するために図 5 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

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	4	5	6	7	8	9
	UNSAT	UNSAT	SAT	UNSAT	SAT	UNSAT	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	11	12	13	14	15	16
	UNSAT	SAT	UNSAT	SAT	UNSAT	UNSAT	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

に従い与えられた n 個の変数 $x_1, \ldots, x_n$ の入力に対し，$\bigwedge_{1 \le i < j \le n}(x_i \neq x_j)$ 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 $\neg\bigwedge(x_i > ub - n + 1)$ 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 [21] 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．

性能評価の制限時間は 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 Mac OS X 上で行っている．表 1 は $PLS(n)$ (但し $3 \le n \le 16$) 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．2009 年に開催された CSP ソルバー競技会では $n \le 12$ までを解いた Sugar を除くと $n > 8$ の $PLS(n)$ を 1800 秒以内にどの CSP ソルバーも解くことができなかったことから Scarab はこの問題において良い性能を示しているといえる．

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 とその応用について説明を行った．グラフ彩色問題，正方形詰込み問題，汎対角線ラテン方陣の三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応用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の実装に使うことができる．

本稿では触れなかったが Scarab の他の機能として CSP レベルで Minimal Unsatisfiable Subformula (MUS) を計算することが可能である．加えて埋め込みの制約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 SAT ソルバー Sat4j と連携して扱うことが可能である．Scarab とこれら Sat4j の機能の組合せはさらに SAT 技術の応用を広げる助けになると考えている．Scarab の詳細とソースコードは次から入手できる．

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

参考文献

[1] 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, pp. 122–129 (2010).

[2] Berre, D. L. and Parrain, A.: The Sat4j Library, release 2.2, *Journal on Satisfiability, Boolean Modeling and Computation*, Vol. 7, No. 2-3, pp. 59–64 (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] Eén, N. and Sörensson, N.: Temporal Induction by Incremental SAT Solving, *Electronic Notes in Theoretical Computer Science*, Vol. 89, No. 4 (2003).

[5] Fujita, H., Koshimura, M. and Hasegawa, R.:

SCSat: A Soft Constraint Guided SAT Solver, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, pp. 415–421 (2013).

[6] Hebrard, E., O'Mahony, E. and O'Sullivan, B.: Constraint Programming and Combinatorial Optimisation in Numberjack, *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010), LNCS 6140*, pp. 181–185 (2010).

[7] 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, pp. 57–67 (2010).

[8] Kaivola, R., Ghughal, R., Narasimhan, N., Telfer, A., Whittemore, J., Pandav, S., Slobodová, A., Taylor, C., Frolov, V. A., Reeber, E. and Naik, A.: Replacing Testing with Formal Verification in Intel CoreTM i7 Processor Execution Engine Validation, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, pp. 414–429 (2009).

[9] Knuth, D. E.: *The Art of Computer Programming, Volume 4B*, Addison-Wesley Professional (2015).

[10] Le Berre, D. and Rapicault, P.: Dependency Management for the Eclipse Ecosystem: Eclipse P2, Metadata and Resolution, *Proceedings of the 1st International Workshop on Open Component Ecosystems*, IWOCE '09, New York, NY, USA, ACM, pp. 21–30 (2009).

[11] Lecoutre, C., Roussel, O. and van Dongen, M. R. C.: Promoting Robust Black-box Solvers Through Competitions, *Constraints*, Vol. 15, No. 3, pp. 317–326 (2010).

[12] Mernik, M., Heering, J. and Sloane, A. M.: When and How to Develop Domain-Specific Languages, *ACM Computing Surveys*, Vol. 37, No. 4, pp. 316–344 (2005).

[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, pp. 465–483 (online), DOI: 10.1017/S1471068412000130 (2012).

[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, pp. 114–121 (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, pp. 68–76 (2010).

[16] Nabeshima, H., Soh, T., Inoue, K. and Iwanuma, K.: Lemma Reusing for SAT based Planning and Scheduling, *Proceedings of the International Conference on Automated Planning and Scheduling 2006 (ICAPS 2006)*, pp. 103–112 (2006).

[17] 則武治樹, 番原睦則, 宋 剛秀, 田村直之, 井上克巳：パッキング配列問題の制約モデリングと SAT 符号化, コンピュータソフトウェア, Vol. 31, No. 1, pp. 116–130 (2013).

[18] Odersky, M., Spoon, L. and Venners, B.: *Programming in Scala*, Artima, Inc., second edition (2010).

[19] Régin, J.-C.: A Filtering Algorithm for Constraints of Difference in CSPs, *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, pp. 362–367 (1994).

[20] Soh, T., Berre, D. L., Roussel, S., Banbara, M. and Tamura, N.: Incremental SAT-Based Method with Native Boolean Cardinality Handling for the Hamiltonian Cycle Problem, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, pp. 684–693 (2014).

[21] 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).

[22] Tamura, N., Taga, A., Kitagawa, S. and Banbara, M.: Compiling Finite Linear CSP into SAT, *Constraints*, Vol. 14, No. 2, pp. 254–272 (2009).

[23] Tamura, N., Tanjo, T. and Banbara, M.: System Description of a SAT-based CSP Solver Sugar, *Proceedings of the 3rd International CSP Solver Competition*, pp. 71–75 (2008).

[24] 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, pp. 77–85 (2010).

[25] 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, pp. 114–129 (2012).

[26] van Hoeve, W.-J. and Katrie, I.: Global Constraint, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, Elsevier, pp. 169–208 (2006).

[27] Zhou, N.-F.: The language features and architecture of B-Prolog, *Theory and Practice of Logic Programming*, Vol. 12, No. 1-2, pp. 189–218 (2012).