Scala 上で実現された SAT 型制約プログラミングシステム Scarab について

宋 剛秀 [1]
共同研究者
Daniel Le Berre[2]　Stéphanie Roussel[2]
番原 睦則 [1]　**田村 直之** [1]

[1] **神戸大学 情報基盤センター**

[2]CRIL-CNRS, UMR 8188, Université d'Artois

2014/09/05
ScalaMatsuri

目次

命題論理の充足可能性判定 (SAT) 問題

- **SAT は与えられた論理式を真にするような値割当てが存在するかどうかを判定する問題である.**

- SAT **は** [Cook, 1971] **により，初めてその** NP **完全性が証明された問題であり，計算機科学において理論的にも実践的にも基礎となる問題である.**

SAT **問題のインスタンスは連言標準形** (CNF) **で与えられる.**

- CNF **式は節の連言である.**
- **節はリテラルの選言である.**
- **リテラルは命題変数もしくはその否定である.**

命題論理の充足可能性判定 (SAT) 問題

- **SAT は与えられた論理式を真にするような値割当てが存在するかどうかを判定する問題である**.

- SAT **は** [Cook, 1971] **により，初めてその** NP **完全性が証明された問題であり，計算機科学において理論的にも実践的にも基礎となる問題である**.

SAT 問題の例 (CNF)

- **ブール変数**: $a, b, c \in \{True, False\}$
- **質問**: **以下の** CNF **式は充足可能か**?

$$(a \vee b \vee c) \wedge$$
$$(\neg a \vee \neg b) \wedge$$
$$(\neg a \vee \neg c) \wedge$$
$$(\neg b \vee \neg c)$$

- **解**: **充足可能** (SAT).
- **全ての節を充足する値割当て** $(a, b, c) = (True, False, False)$ **が存在する**.

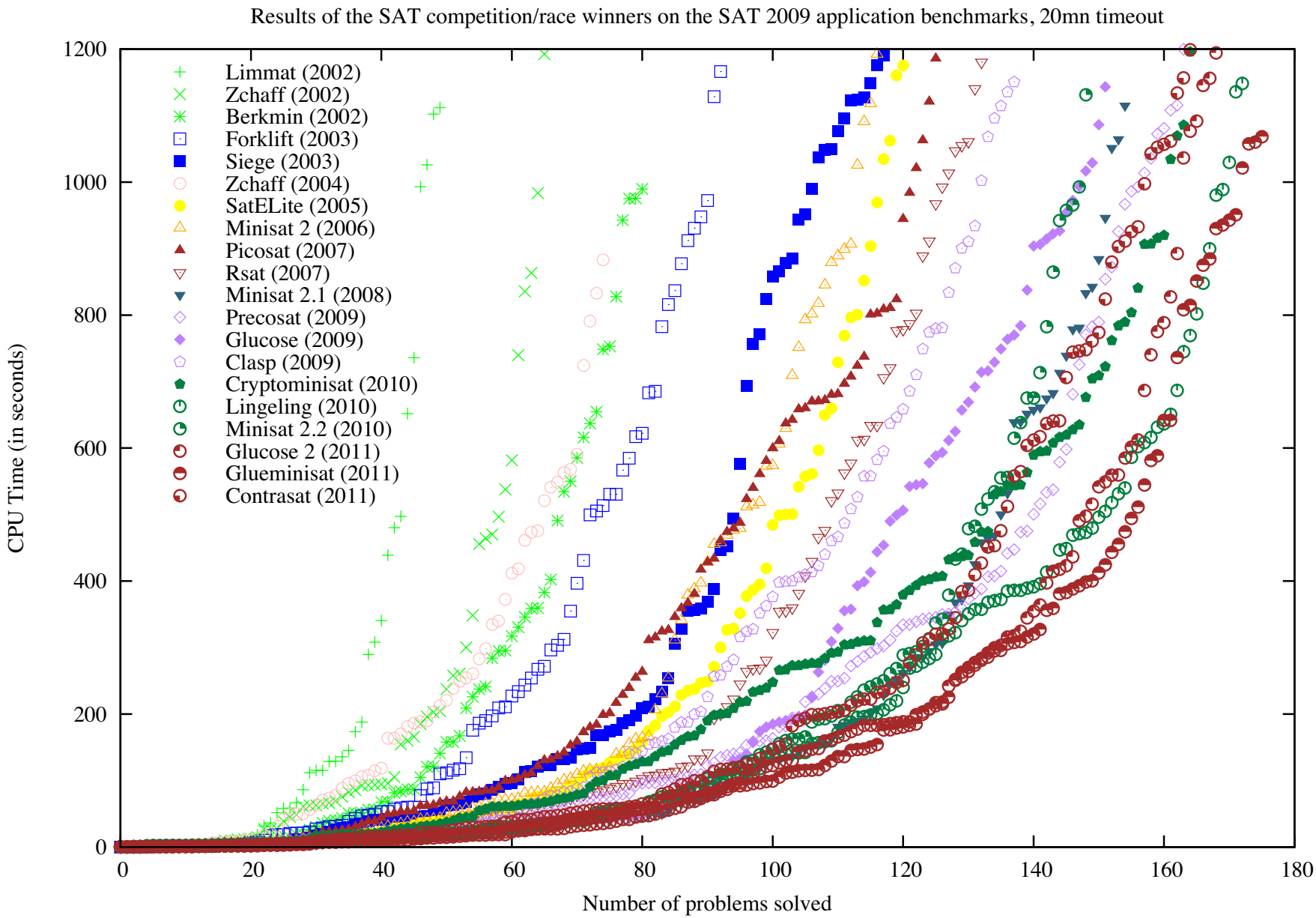
SAT ソルバー

- 値割当てには 2^n の組合せがある.
- では小さな n (e.g. $n = 100$) でも解けないのだろうか?
- SAT ソルバーは実際的にその能力の限界を更新し続けている.

> SAT ソルバーとは与えられた SAT 問題が SAT かどうかを判定するプログラム.
>
> ほとんどの SAT ソルバーは充足可能 (SAT) を返す時にはその値割当ても返す.

- 系統的探索を行う SAT ソルバーは DPLL [Davis et al., 1962] に基づく.
- 2000 年頃から SAT ソルバーの性能は様々な技術とともに毎年改善されてきた.
- 現代の SAT ソルバはだいたい 10^6 変数，10^7 節の問題を扱える.

SAT ソルバーの進歩 ([Simon 2011] より引用)



カクタスプロット [Simon 2011]

SAT の応用

この十数年の SAT ソルバーの飛躍的な性能向上を背景に，様々な問題に対する SAT 型システムの開発が行われている:

- **プランニング** (SATPLAN, Blackbox) [Kautz & Selman 1992]
- **ジョブショップスケジューリング** [Crawford & Baker 1994]
- **有界モデル検査** [Biere 1999]
- **書換えシステム** (AProVE) [Giesl et al. 2004]
- **制約充足問題** (Sugar) [Tamura et al. 2006]
- **その他**
  - **テストケース生成**
  - **システム生物学**
  - **時間割問題**
  - **パッキング問題**
  - **パズル** etc...

SAT に関する他のニュース

- Java で実装された SAT ソルバー Sat4j が Eclipse にプラグインの依存解析を行うために統合された.
- ドナルド・クヌース先生が招待講演 “Satisfiability and The Art of Computer Programming” を SAT の理論と実践に関する国際会議で行った.
- また The Art Of Computer Programming の 4b 号に SAT が取り上げられる予定.

SAT の参考文献

- Biere, A., Heule, M., van Maaren, H., and Walsh, T., editors (2009). Handbook of Satisfiability, volume 185 of Frontiers in Artificial Intelligence and Applications (FAIA). IOS Press.
- (in 日本語) Recent Advances in SAT Techniques, Journal of the Japan Society for Artificial Intelligence, Special Issue, 25(1), 2010.

目次

1. **命題論理の充足可能性判定** (SAT) **問題について**
2. Scarab: Scala **上に実現された** SAT **型制約プログラミングシステム**
3. Scarab **における制約モデリング**: **事例紹介**
4. Sat4j **との連携**

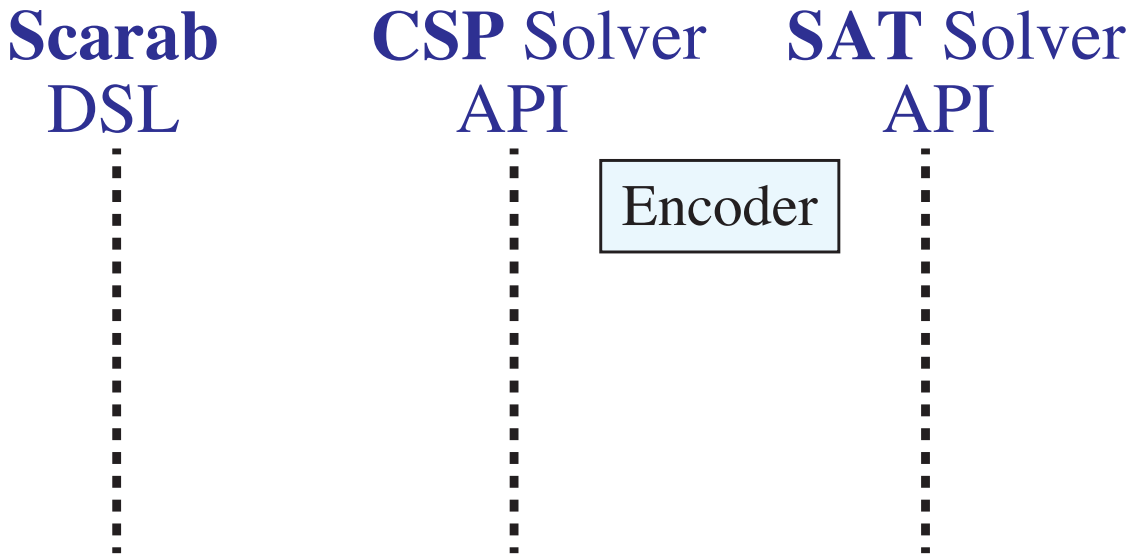
Scarab

Scarab は 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，変更の容易性，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

Scarab は 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，変更の容易性，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．

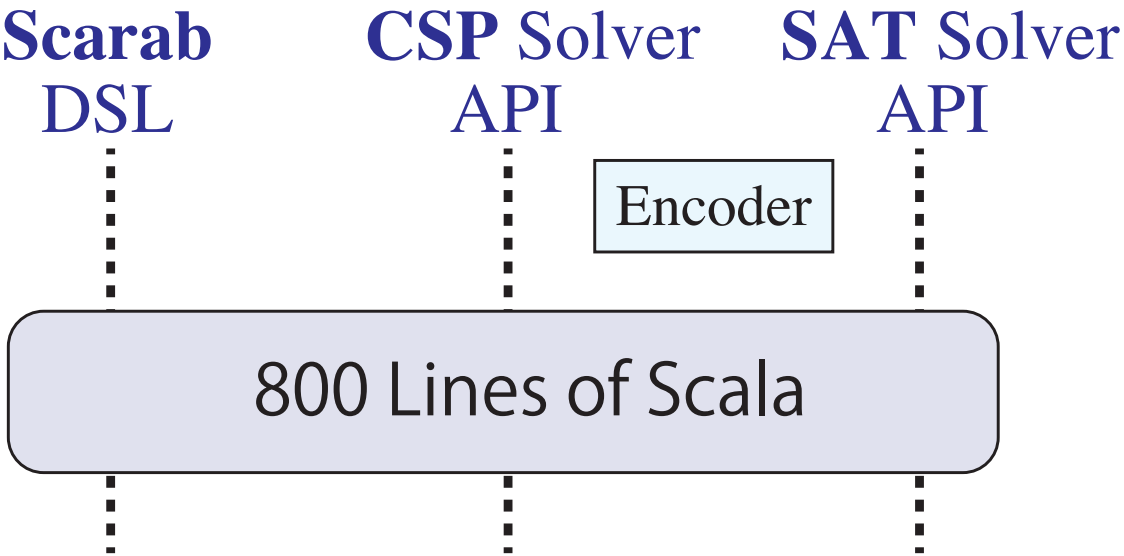
- **以下から構成される**:
  1. Scarab DSL: **制約プログラミングのためのドメイン特化言語**
  2. CSP **ソルバー** API
  3. SAT **符号化モジュール**
  4. SAT **ソルバー** API



Scarab

Scarab は 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，変更の容易性，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．

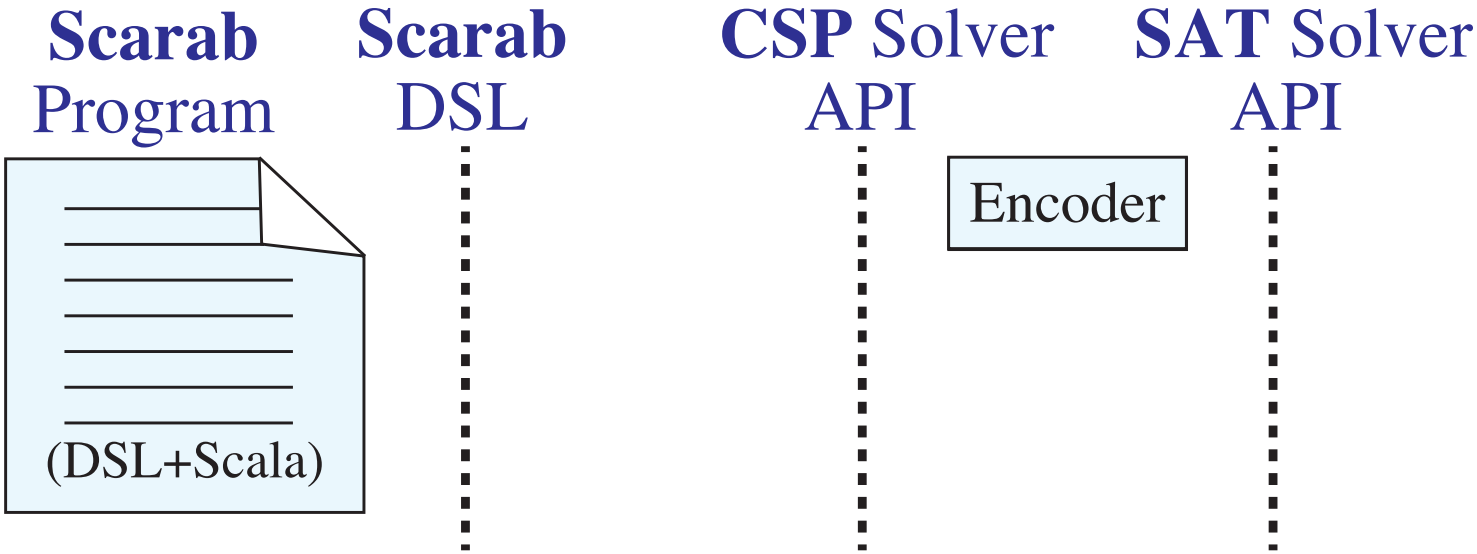
- 以下から構成される:
  1. Scarab DSL: 制約プログラミングのためのドメイン特化言語
  2. CSP ソルバー API
  3. SAT 符号化モジュール
  4. SAT ソルバー API



Scarab

Scarab は 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，変更の容易性，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．

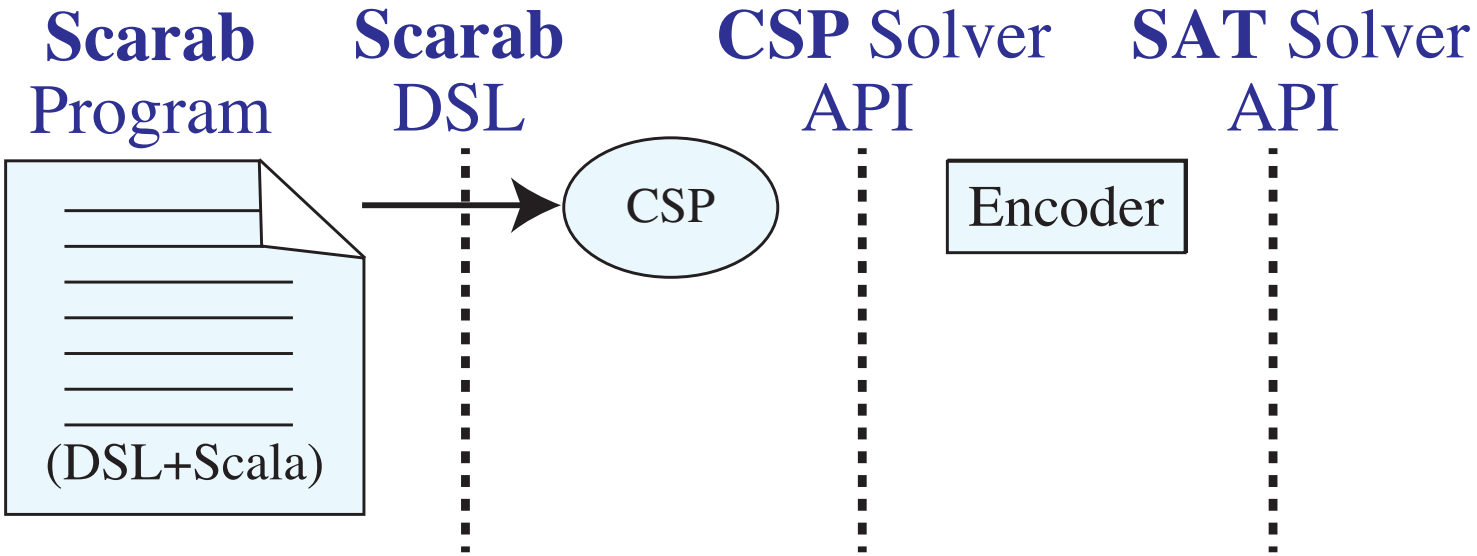
- **以下から構成される**:
  1. Scarab DSL: **制約プログラミングのためのドメイン特化言語**
  2. CSP **ソルバー** API
  3. SAT **符号化モジュール**
  4. SAT **ソルバー** API



Scarab

Scarab は 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，変更の容易性，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．

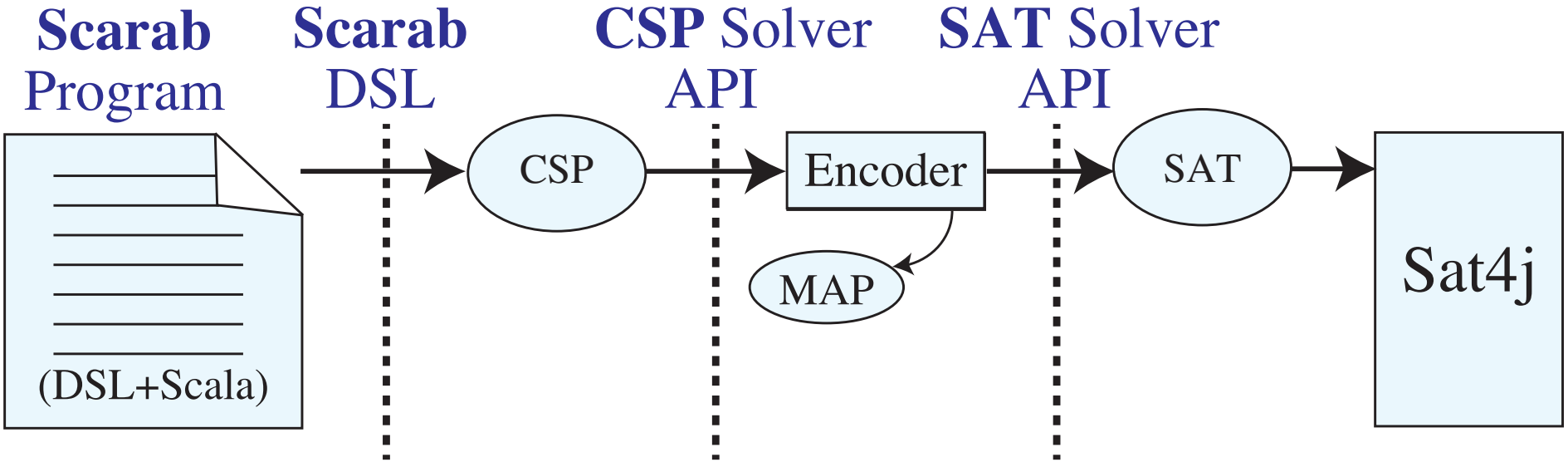
- **以下から構成される**:
  1. Scarab DSL: **制約プログラミングのためのドメイン特化言語**
  2. CSP **ソルバー** API
  3. SAT **符号化モジュール**
  4. SAT **ソルバー** API



Scarab

Scarab は 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，変更の容易性，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．

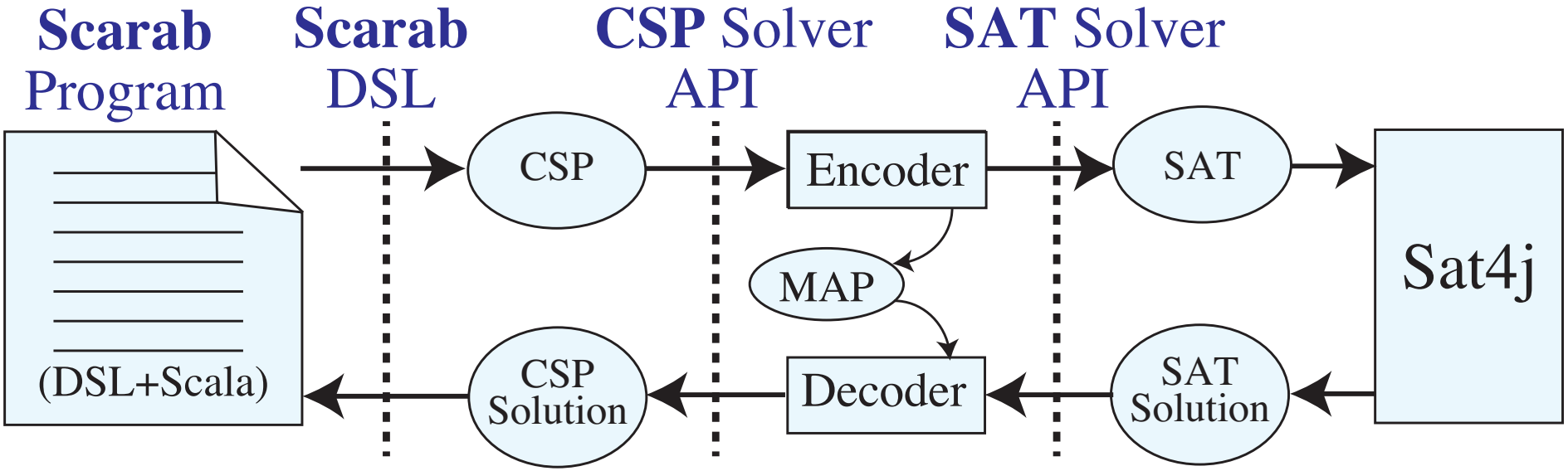
- **以下から構成される**:
  1. Scarab DSL: **制約プログラミングのためのドメイン特化言語**
  2. CSP **ソルバー** API
  3. SAT **符号化モジュール**
  4. SAT **ソルバー** API



Scarab

Scarab は 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，変更の容易性，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．

- **以下から構成される**:
  1. Scarab DSL: **制約プログラミングのためのドメイン特化言語**
  2. CSP **ソルバー** API
  3. SAT **符号化モジュール**
  4. SAT **ソルバー** API



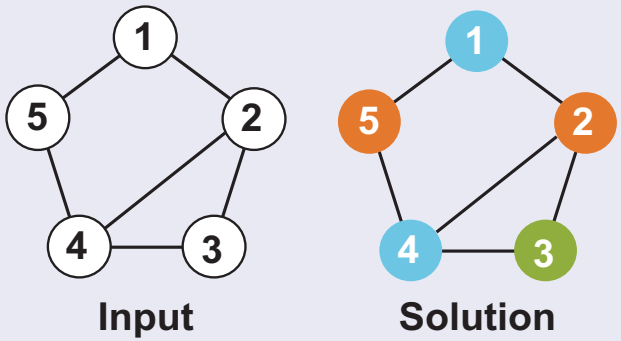
その他の Scarab の特長

- (効率性，変更の容易性) Scarab では，CSP ソルバー競技会の優勝ソルバーである Sugar が採用している順序符号化を 25 行の Scala プログラムで実装している．
- (可搬性) デフォルトの SAT ソルバーである Sat4j を使う場合，開発した SAT 型システム全体が Java Virtual Machine 上で動作可能．
- Sat4j との連携により，インクリメンタル解法や仮説を用いた CSP 解法などの高度な解法を利用可能．

これらの特長はこれまでに提案されている制約プログラミング DSL である Copris と比べた際の利点もなっている．

Scarab プログラムの例: GCP.scala

グラフ彩色問題は与えられたグラフの全ての頂点をそれぞれの隣接頂点の色と異なるように彩色する問題である.




```
 1:  import jp.kobe_u.scarab.csp._
 2:  import jp.kobe_u.scarab.solver._
 3:  import jp.kobe_u.scarab.sapp._
 4:
 5:  val nodes = Seq(1,2,3,4,5)
 6:  val edges = Seq((1,2),(1,5),(2,3),(2,4),(3,4),(4,5))
 7:  val colors = 3
 8:
 9:  for (i <- nodes)      int('n(i),1,colors)
10:  for ((i,j) <- edges) add('n(i) !== 'n(j))
11:  if (find)             println(solution)
```

Scarab DSL: インポートと問題の定義

```
1:  import jp.kobe_u.scarab.csp._
2:  import jp.kobe_u.scarab.solver._
3:  import jp.kobe_u.scarab.sapp._
```

- 1，2 行目は CSP およびソルバーのクラスをインポートする.
- 3 行目は CSP, 符号化モジュール, SAT ソルバー, CSP ソルバーのデフォルトオブジェクトとメソッドをインポートする.

```
5: val nodes = Seq(1,2,3,4,5)
6: val edges = Seq((1,2),(1,5),(2,3),(2,4),(3,4),(4,5))
7: val colors = 3
```

- 5，6 行目は与えられた頂点と辺の集合を Scala の Seq オブジェクトとして定義する.
- 7 行目は彩色数を 3 として定義する.

Scarab DSL: 整数変数と制約の定義

```
9: for (i <- nodes) int('n(i),1,colors)
```

- 9 行目は整数変数をデフォルトの CSP へと追加している.
- 'n は Scala の Symbol オブジェクトを表す.
- Symbol オブジェクトは暗黙変換により Scarab の整数変数を表す Var オブジェクトに自動的に変換される.

```
10: for ((i,j) <- edges)  add('n(i) !== 'n(j))
```

- 10 行目は制約を CSP へと追加している.
- 制約を表現するために以下の演算子が利用可能である:
  - 論理演算子: &&, ||
  - 比較演算子: ===, !==, <, <=, >=, >
  - 算術演算子: +, -

Scarab DSL: 解の探索

```
11: if (find) println(solution)
```

- **find** は順序符号化を用いて CSP を SAT に符号化し，SAT ソルバー Sat4j を呼んで解を求める.
- **solution** は CSP の充足解を返す.

目次

汎対角線ラテン方陣: $PLS(n)$

汎対角線ラテン方陣 $PLS(n)$ **は異なる** n **個の数字を** n **行** n **列の行列にそれぞれの数字が各行，各列，各汎対角線にちょうど一度表れるように配置する問題である．**2009 **年の** CSP **ソルバー競技会では**, Sugar **を除いて** $n > 8$ **の** PLS(n) **を解けた** CSP **ソルバーは無い．**

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

汎対角線ラテン方陣: $PLS(n)$

汎対角線ラテン方陣 $PLS(n)$ **は異なる** n **個の数字を** n **行** n **列の行列にそれぞれの数字が各行，各列，各汎対角線にちょうど一度表れるように配置する問題である．**2009 **年の** CSP **ソルバー競技会では**, Sugar **を除いて** $n > 8$ **の** PLS(n) **を解けた** CSP **ソルバーは無い．**

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

汎対角線ラテン方陣: $PLS(n)$

汎対角線ラテン方陣 $PLS(n)$ **は異なる** n **個の数字を** n **行** n **列の行列にそれぞれの数字が各行，各列，各汎対角線にちょうど一度表れるように配置する問題である．**2009 **年の** CSP **ソルバー競技会では，**Sugar **を除いて** $n > 8$ **の** PLS(n) **を解けた** CSP **ソルバーは無い．**

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

Scarab **を使うことで** PLS **に対する** 5 **種類の** SAT **型システムを** 35 **行以内で書くことができる．**

システム名	モデリング	符号化	行数
AD1	alldiff	naive	17
AD2		with Perm. & P. H. Const.	31
BC1	**基数制約**	Pairwise	22
BC2		Totalizer [Bailleux '03]	35
BC3		Seq. Counter [Sinz '05]	27

alldiff モデル

汎対角線ラテン方陣 $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

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

- **整数変数** $x_{ij} \in \{1, 2, 3, 4, 5\}$
- **それぞれの行に対する** alldiff (5 **行**)
- **それぞれの列に対する** alldiff (5 **列**)
- **それぞれの汎対角線に対する** alldiff (10 **汎対角線**)

alldiff モデルの Scarab プログラム

```
 1:  import jp.kobe_u.scarab.csp._
 2:  import jp.kobe_u.scarab.solver._
 3:  import jp.kobe_u.scarab.sapp._
 4:
 5:  val n = args(0).toInt
 6:
 7:  for (i <- 1 to n; j <- 1 to n)  int('x(i,j),1,n)
 8:  for (i <- 1 to n) {
 9:   add(alldiff((1 to n).map(j => 'x(i,j))))
10:   add(alldiff((1 to n).map(j => 'x(j,i))))
11:   add(alldiff((1 to n).map(j => 'x(j,(i+j-1)%n+1))))
12:   add(alldiff((1 to n).map(j => 'x(j,(i+(j-1)*(n-1))%n+1))))
13:  }
14:
15:  if (find)  println(solution)
```

alldiff (naive)

```
def alldiff(xs: Seq[Var]) =
  And(for (Seq(x, y) <- xs.combinations(2))
      yield x !== y)
```

alldiff (optimized)

```
def alldiff(xs: Seq[Var]) = {
  val lb = for (x <- xs) yield csp.dom(x).lb
  val ub = for (x <- xs) yield csp.dom(x).ub
  // pigeon hole
  val ph =
    And(Or(for (x <- xs) yield !(x < lb.min+xs.size-1)),
        Or(for (x <- xs) yield !(x > ub.max-xs.size+1)))
  // permutation
  def perm =
    And(for (num <- lb.min to ub.max)
        yield Or(for (x <- xs) yield x === num))
  val extra = if (ub.max-lb.min+1 == xs.size) And(ph,perm)
              else ph

  And(And(for (Seq(x, y) <- xs.combinations(2))
          yield x !== y),extra)
}
```

性能評価: CPU 時間 (秒) の比較. (1 時間打切り)

n	SAT/UNSAT	AD1	AD2	BC1	BC2	BC3
7	SAT	0.2	0.2	0.2	0.3	0.3
8	UNSAT	T.O.	0.5	0.3	0.3	0.3
9	UNSAT	T.O.	0.3	0.5	0.3	0.2
10	UNSAT	T.O.	0.4	1.0	0.3	0.3
11	SAT	0.3	0.3	2.3	0.5	0.4
12	UNSAT	T.O.	1.0	5.3	0.8	0.8
13	SAT	T.O.	0.5	T.O.	T.O.	T.O.
14	UNSAT	T.O.	9.7	32.4	8.2	6.8
15	UNSAT	T.O.	388.9	322.7	194.6	155.8
16	UNSAT	T.O.	457.1	546.6	300.7	414.8

- **同じソルバーを使っても各モデリング/符号化によってシステムの性能が異なることが分かる** (AD2 **が最も良い**)**.**
- Scarab **は，これらのシステムの円滑な開発を支援する.**

目次

1. **命題論理の充足可能性判定** (SAT) **問題について**
2. Scarab: Scala **上に実現された** SAT **型制約プログラミングシステム**
3. Scarab **における制約モデリング**: **事例紹介**
4. Sat4j **との連携**

Sat4j との連携

- SAT ソルバー Sat4j は JVM 上で動作し，Scarab から直接呼び出すことができる．
- この Scarab と Sat4j の連携により，インクリメンタル解法や仮説を用いた CSP 解法を実現することができる．
- 以下の例を用いて説明する．

```
1:  int('x, 1, 3)      // 整数変数 x ∈ {1,2,3}
2:  int('y, 1, 3)      // 整数変数 y ∈ {1,2,3}
3:  add('x === 'y)     // 制約 x = y
4:  find               // 1 回目の find 呼び出し
5:  add('x !== 3)
6:  find               // 2 回目の find 呼び出し
7:
8:  find('y === 3)     // 仮説 y = 3
9:  find('x === 1)     // 仮説 x = 1
```

インクリメンタル解法

```
int('x, 1, 3)     // 整数変数 x ∈ {1,2,3}
int('y, 1, 3)     // 整数変数 y ∈ {1,2,3}
add('x === 'y)    // 制約 x = y
find              // 1 回目の find 呼び出し: x = 3; y = 3
add('x !== 3)
find              // 2 回目の find 呼び出し: x = 2; y = 2
```

- 1 回目の `find` の呼び出しでは CSP 全体が SAT 符号化され解が計算される．
- 2 回目の `find` の呼び出しでは追加制約 $x \neq 3$ のみが SAT 符号化され，1 回目の符号化分と併せて解が計算される．
- その際に学習節などの前回得られた探索情報も保持・再利用されるため効果的な解探索が可能となる．

仮説を用いたCSP解法

```
find('y === 3)    // 仮説 y = 3: UNSAT
find('x === 1)    // 仮説 x = 1: x = 1; y = 1
```

- `find(assumption: Constraint)` は与えられた仮説の基でCSPの解を計算する (リテラルの連言にSAT符号化される制約を仮説として与えることが可能).
- インクリメンタル解法および仮説を用いたCSP解法は制約最適化問題や解列挙に利用できる.

おわりに

- **Scarab** は 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，変更の容易性，効率性，可搬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** を用いることで，様々な制約モデリング/符号化を用いた SAT 型システムを少ない行数で簡潔に記述・開発できる．
- **Sat4j** との連携により，インクリメンタル解法や仮説を用いた CSP 解法などの高度な解法を利用可能になる．
- **今後の課題**
  - Sat4j が持つ SAT 技術の利用
  - バックエンドのソルバー追加
- **Scarab の URL** `http://kix.istc.kobe-u.ac.jp/~soh/scarab/`

補助スライド

Encoded Variables in Order Encoding

Table: Truth table of $p(x \leq a)$

x の値	$p(x \leq 0)$	$p(x \leq 1)$	$p(x \leq 2)$
0	1	1	1
1	0	1	1
2	0	0	1
3	0	0	0

Scala

- Scala is a relatively new programming language receiving an increasing interest for developing real-world applications.
- Scala is an integration of both functional and object-oriented programming paradigms.
- The main features of Scala are:
  - type inferences,
  - higher order functions,
  - immutable collections, and
  - concurrent computation.
- It is also suitable for implementing Domain-Specific Language (DSL)˜[Mernik et al., 2005] embedded in Scala.
- The Scala compiler generates Java Virtual Machine (JVM) bytecode, and Java class libraries can be used in Scala.

Pandiagonal Latin Square: $PLS(n)$

Place different n numbers into $n \times n$ matrix such that each number appears exactly once
for each row, column, diagonally down right, and diagonally up right.

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

Pandiagonal Latin Square: $PLS(n)$

Place different n numbers into $n \times n$ matrix such that each number appears exactly once
for each row, column, diagonally down right, and diagonally up right.

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

We can write five SAT-based PLS Solvers **within 35 lines**.

Name	Modeling	Encoding	Lines
AD1	alldiff	naive	17
AD2		with Perm. & P. H. Const.	31
BC1	Boolean	Pairwise	22
BC2	Cardinality	Totalizer [Bailleux '03]	35
BC3		Seq. Counter [Sinz '05]	27

Let's have a look their performance. Note that, in CSP Solver Comp. 2009, **NO CSP solver** (except Sugar) could solve $n > 8$.

Designing Constraint Models in Scarab

Pandiagonal Latin Square $PLS(n)$ is a problem of placing different n numbers into $n \times n$ matrix such that each number is occurring exactly once for each row, column, diagonally down right, and diagonally up right.

- **alldiff Model**
  - One uses alldiff constraint, which is one of the best known and most studied global constraints in constraint programming.
  - The constraint alldiff$(a_1, \ldots, a_n)$ ensures that the values assigned to the variable $a_1, \ldots, a_n$ must be pairwise distinct.
- **Boolean Cardinality Model**
  - One uses Boolean cardinality constraint.

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

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

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)
- $PLS(5)$ is satisfiable.

Scarab Program for alldiff Model

```
 1:  import jp.kobe_u.scarab.csp._
 2:  import jp.kobe_u.scarab.solver._
 3:  import jp.kobe_u.scarab.sapp._
 4:
 5:  val n = args(0).toInt
 6:
 7:  for (i <- 1 to n; j <- 1 to n)  int('x(i,j),1,n)
 8:  for (i <- 1 to n) {
 9:   add(alldiff((1 to n).map(j => 'x(i,j))))
10:   add(alldiff((1 to n).map(j => 'x(j,i))))
11:   add(alldiff((1 to n).map(j => 'x(j,(i+j-1)%n+1))))
12:   add(alldiff((1 to n).map(j => 'x(j,(i+(j-1)*(n-1))%n+1))))
13:  }
14:
15:  if (find)  println(solution)
```

Encoding alldiff

- In Scarab, all we have to do for implementing global constraints is just decomposing them into simple arithmetic constraints [Bessiere et al. '09].

In the case of alldiff($a_1, \ldots, a_n$),

It is decomposed into pairwise not-equal constraints

$$\bigwedge_{1 \leq i < j \leq n} (a_i \neq a_j)$$

.

- This (naive) alldiff is enough to just have a feasible constraint model for $PLS(n)$.
- But, one probably want to improve this :)

Extra Constraints for alldiff$(a_1, \ldots, a_n)$

- In Pandiagonal Latin Square $PLS(n)$, all integer variables $a_1, \ldots, a_n$ have the same domain $\{1, \ldots, n\}$.
- Then, we can add the following extra constraints.
- **Permutation constraints**:

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} (a_j = i)$$

  - It represents that one of $a_1, \ldots, a_n$ must be assigned to i.
- **Pigeon hole constraint**:

$$\neg \bigwedge_{i=1}^{n} (a_i < n) \wedge \neg \bigwedge_{i=1}^{n} (a_i > 1)$$

  - It represents that mutually different n variables cannot be assigned within the interval of the size $n - 1$.

alldiff (naive)

```
def alldiff(xs: Seq[Var]) =
  And(for (Seq(x, y) <- xs.combinations(2))
      yield x !== y)
```

alldiff (optimized)

```
def alldiff(xs: Seq[Var]) = {
  val lb = for (x <- xs) yield csp.dom(x).lb
  val ub = for (x <- xs) yield csp.dom(x).ub
  // pigeon hole
  val ph =
    And(Or(for (x <- xs) yield !(x < lb.min+xs.size-1)),
        Or(for (x <- xs) yield !(x > ub.max-xs.size+1)))
  // permutation
  def perm =
    And(for (num <- lb.min to ub.max)
        yield Or(for (x <- xs) yield x === num))
  val extra = if (ub.max-lb.min+1 == xs.size) And(ph,perm)
              else ph

  And(And(for (Seq(x, y) <- xs.combinations(2))
          yield x !== y),extra)
}
```

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad$ $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad$ $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows)
  - for each column (5 columns)

$$y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$$
$$y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows)
  - for each column (5 columns)

$$y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$$
$$y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals) $\quad y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals) $\quad y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals) $\quad y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$

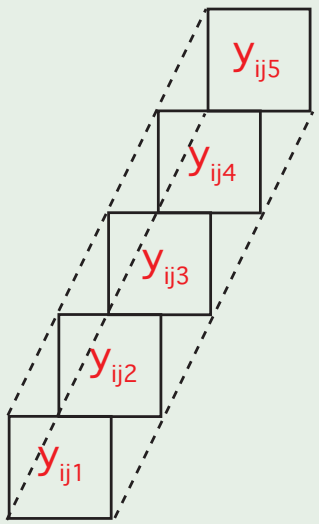
Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals) $\quad y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}



- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals) $\quad y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$
- for each (i, j) position (25 positions) $\quad y_{ij1} + y_{ij2} + y_{ij3} + y_{ij4} + y_{ij5} = 1$

Scarab Program for Boolean Cardinality Model

```
 1:  import jp.kobe_u.scarab.csp._
 2:  import jp.kobe_u.scarab.solver._
 3:  import jp.kobe_u.scarab.sapp._
 4:
 5:  for (i <- 1 to n; j <- 1 to n; num <- 1 to n)
 6:    int('y(i,j,num),0,1)
 7:
 8:  for (num <- 1 to n) {
 9:    for (i <- 1 to n) {
10:     add(BC((1 to n).map(j => 'y(i,j,num)))===1)
11:     add(BC((1 to n).map(j => 'y(j,i,num)))===1)
12:     add(BC((1 to n).map(j => 'y(j,(i+j-1)%n+1,num))) === 1)
13:     add(BC((1 to n).map(j => 'y(j,(i+(j-1)*(n-1))%n+1,num))) === 1)
14:   }
15: }
16:
17: for (i <- 1 to n; j <- 1 to n)
18:   add(BC((1 to n).map(k => 'y(i,j,k))) === 1)
19:
20: if (find)  println(solution)
```

SAT Encoding of Boolean Cardinality in Scarab

- There are several ways for encoding Boolean cardinality.
- In Scarab, we can easily write the following encoding methods by defining your own BC methods.
  - Pairwise
  - Totalizer [Bailleux '03]
  - Sequential Counter [Sinz '05]
- In total, **3 variants of Boolean cardinality model** are obtained.
  - BC1: Pairwise (implemented by 2 lines)
  - BC2: Totalizer [Bailleux '03] (implemented by 15 lines)
  - BC3: Sequential Counter [Sinz '05] (implemented by 7 lines)
- Good point to use Scarab is that we can test those models **without writing dedicated programs**.

Experiments

Comparison on Solving Pandiagonal Latin Square

To show the differences in performance, we compared the following 5 models.

1. AD1: naive alldiff
2. AD2: optimized alldiff
3. BC1: Pairwise
4. BC2: [Bailleux '03]
5. BC3: [Sinz '05]

Benchmark and Experimental Environment

- Benchmark: Pandiagonal Latin Square ($n = 7$ to $n = 16$)
- CPU: 2.93GHz, Mem: 2GB, Time Limit: 3600 seconds

Results (CPU Time in Seconds)

n	SAT/UNSAT	AD1	AD2	BC1	BC2	BC3
7	SAT	0.2	0.2	0.2	0.3	0.3
8	UNSAT	T.O.	0.5	0.3	0.3	0.3
9	UNSAT	T.O.	0.3	0.5	0.3	0.2
10	UNSAT	T.O.	0.4	1.0	0.3	0.3
11	SAT	0.3	0.3	2.3	0.5	0.4
12	UNSAT	T.O.	1.0	5.3	0.8	0.8
13	SAT	T.O.	0.5	T.O.	T.O.	T.O.
14	UNSAT	T.O.	9.7	32.4	8.2	6.8
15	UNSAT	T.O.	388.9	322.7	194.6	155.8
16	UNSAT	T.O.	457.1	546.6	300.7	414.8

- Only optimized version of alldiff model (AD2) solved all instances.
- Modeling and encoding have an important role in developing SAT-based systems.
- Scarab helps users to focus on them ;)

BC1: Pairwise

Definition of BC1

```
def BC1(xs: Seq[Var]): Term = Sum(xs)
```

BC1: Pairwise (cont.)

Scarab Program for $x + y + z = 1$

```
int('x,0,1)
int('y,0,1)
int('z,0,1)
add(BC1(Seq('x, 'y, 'z)) === 1)
```

CNF Generated by Scarab

$$\left.\begin{array}{l} p(x \leq 0) \ \vee \ p(y \leq 0) \\ p(x \leq 0) \ \vee \ p(z \leq 0) \\ p(y \leq 0) \ \vee \ p(z \leq 0) \end{array}\right\} x + y + z \leq 1$$

$$\neg p(x \leq 0) \ \vee \ \neg p(y \leq 0) \ \vee \ \neg p(z \leq 0) \quad \} \, x + y + z \geq 1$$

BC2: [Bailleux '03]

Definition of BC2

```
def BC2(xs: Seq[Var]): Term = {
  if (xs.size == 2) xs(0) + xs(1)
  else if (xs.size == 3) {
    val v = int(Var(), 0, 1)
    add(v === BC2(xs.drop(1)))
    xs(0) + v
  } else {
    val (xs1, xs2) =
         xs.splitAt(xs.size / 2)
    val v1 = int(Var(), 0, 1)
    val v2 = int(Var(), 0, 1)
    add(v1 === BC2(xs1))
    add(v2 === BC2(xs2))
    v1 + v2
  }
}
```

BC2: [Bailleux '03] (cont.)

Scarab Program for $x + y + z = 1$

```
int('x,0,1)
int('y,0,1)
int('z,0,1)
add(BC2(Seq('x, 'y, 'z)) === 1)
```

CNF Generated by Scarab (q is auxiliary variable)

$$\left.\begin{array}{rcccc} q & \vee & \neg p(y \leq 0) & \vee & \neg p(z \leq 0) \\ \neg q & \vee & p(z \leq 0) & & \\ \neg q & \vee & p(y \leq 0) & & \\ p(y \leq 0) & \vee & p(z \leq 0) & & \end{array}\right\} y + z = S$$

$$\left.\begin{array}{rcl} q & \vee & p(x \leq 0) \\ \neg q & \vee & \neg p(x \leq 0) \end{array}\right\} x + S = 1$$

BC3: [Sinz '05]

Definition of BC3

```
def BC3(xs: Seq[Var]): Term = {
  val ss =
    for (i <- 1 until xs.size) yield int(Var(), 0, 1)
  add(ss(0) === xs(1) + xs(0))
  for (i <- 2 until xs.size)
    add(ss(i-1) === (xs(i) + ss(i-2)))
  ss(xs.size-2)
}
```

BC3: [Sinz '05] (cont.)

Program for $x + y + z = 1$

```
int('x,0,1)
int('y,0,1)
int('z,0,1)
add(BC3(Seq('x, 'y, 'z))===1)
```

CNF Generated by Scarab (q_1 and q_2 are auxiliary variables)

$$
\left.\begin{array}{llll}
q_1 & \vee & \neg p(y \leq 0) & \vee \quad \neg p(x \leq 0) \\
\neg q_1 & \vee & p(x \leq 0) & \\
\neg q_1 & \vee & p(y \leq 0) & \\
 & & p(x \leq 0) & \vee \quad p(y \leq 0)
\end{array}\right\} x + y = S_1
$$

$$
\left.\begin{array}{llll}
q_2 & \vee & \neg q_1 & \vee \quad \neg p(z \leq 0) \\
\neg q_2 & \vee & q_1 & \\
\neg q_2 & \vee & p(z \leq 0) & \\
q_1 & \vee & p(z \leq 0) &
\end{array}\right\} S_1 + z = S_2
$$

$$
\left.\begin{array}{l}
\neg q_2
\end{array}\right\} S_2 = 1
$$

BC Native Encoder (work in progress)

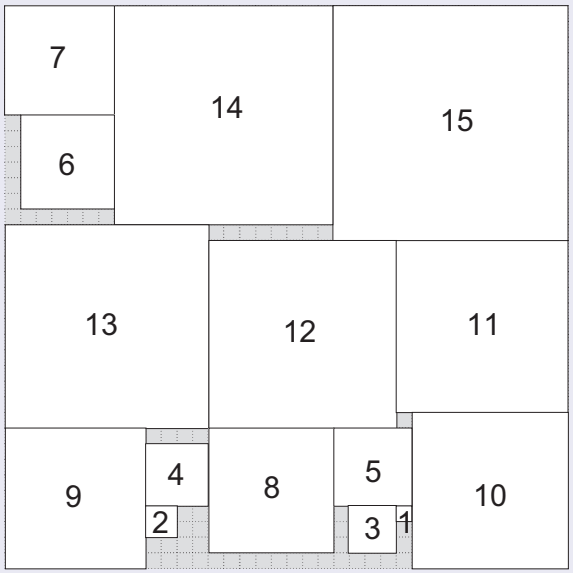
- We have tested Boolean Cardinality Encoder (BC Native Encoder), which natively encodes Boolean cardinality constraints by using `addAtMost` or `addAtLeast` methods of Sat4j
- Preliminary Results (CPU time in seconds)

n	SAT/UNSAT	#Clauses (BC1)	#Constraints (BC Enc.)	time (sec) (BC1)	time (sec) (BC Enc.)
7	SAT	5341	441	0.1	0.1
8	UNSAT	9216	576	0.3	0.1
9	UNSAT	14904	729	0.5	0.1
10	UNSAT	22900	900	1.0	0.1
11	SAT	33759	1089	2.2	0.1
12	UNSAT	48096	1296	5.3	0.3
13	-	66586	1521	T.O.	T.O.
14	UNSAT	89964	1764	32.3	6.7
15	UNSAT	119025	2025	322.6	672.5
16	UNSAT	154624	2304	546.5	1321.4

Example: Square Packing

- **Square Packing** $SP(n, s)$ is a problem of packing a set of squares of sizes 1×1 to $n \times n$ into an enclosing square of size $s \times s$ without overlapping.

Example of $SP(15, 36)$



- **Optimum soluiton of** $SP(n, s)$ is the smallest size of the enclosing square having a feasible packing.

Non-overlapping Constraint Model for $SP(n, s)$

Integer variables

- $x_i \in \{0, \ldots, s - i\}$ and $y_i \in \{0, \ldots, s - i\}$
- Each pair $(x_i,\ y_i)$ represents the lower left coordinates of the square i.

Non-overlapping Constraint ($1 \leq i < j \leq n$)

$$(x_i + i \leq x_j) \vee (x_j + j \leq x_i) \vee (y_i + i \leq y_j) \vee (y_j + j \leq y_i)$$

Decremental Seach

Scarab Program for $SP(n, s)$

```
for (i <- 1 to n) { int('x(i),0,s-i) ; int('y(i),0,s-i) }
for (i <- 1 to n; j <- i+1 to n)
  add(('x(i) + i <= 'x(j)) || ('x(j) + j <= 'x(i)) ||
      ('y(i) + i <= 'y(j)) || ('y(j) + j <= 'y(i)))
```

Searching an Optimum Solution

```
val lb = n; var ub = s; int('m, lb, ub)
for (i <- 1 to n)
  add(('x(i)+i <= 'm) && ('y(i)+i <= 'm))

// Incremental solving
while (lb <= ub && find('m <= ub)) { // using an assumption.
  add('m <= ub)
  ub = solution.intMap('m) - 1
}
```

Bisection Search

Bisection Search

```
var lb = n; var ub = s; commit

while (lb < ub) {
  var size = (lb + ub) / 2
  for (i <- 1 to n)
    add(('x(i)+i<=size)&&('y(i)+i<=size))
  if (find) {
    ub = size
    commit // commit current constraints
  } else {
    lb = size + 1
    rollback // rollback to the last commit point
  }
}
```

References I


Cook, S. A. (1971).
The complexity of theorem-proving procedures.
In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pages 151–158.


Davis, M., Logemann, G., and Loveland, D. W. (1962).
A machine program for theorem-proving.
Communications of the ACM, 5(7):394–397.


Mernik, M., Heering, J., and Sloane, A. M. (2005).
When and how to develop domain-specific languages.
ACM Comput. Surv., 37(4):316–344.