System Architecture and Implementation of a Prototyping Tool for SAT-based Constraint Programming Systems

Takehide Soh\textsuperscript{1}, Naoyuki Tamura\textsuperscript{1}, Mutsunori Banbara\textsuperscript{1}, Daniel Le Berre\textsuperscript{2} and Stéphanie Roussel\textsuperscript{2}

1) Kobe University
2) CRIL-CNRS, UMR 8188

Pragmatics of SAT 2013
(July 8th, 2013 at University of Helsinki)
Modern fast SAT solvers have promoted the development of **SAT-based systems** for various problems.

For an intended problem, we usually need to develop a dedicated program that encodes it into SAT.

It sometimes bothers focusing on **problem modeling** which plays an important role in the system development process.

**In this talk**

- We introduce the **Scarab** system, which is a prototyping tool for developing SAT-based systems.
- Its features are also introduced through examples of **Graph Coloring** and **Pandiagonal Latin Square**.
Contents of Talk

1. **Getting Started: Overview of Scarab**
   - Features
   - Architecture
   - Example: Graph Coloring Problem

2. **Designing Constraint Models in Scarab**
   - Pandiagonal Latin Square
   - alldiff Model
   - Boolean Cardinality Model

3. **Advanced Solving Techniques using Sat4j**
   - Incremental SAT Solving
   - CSP Solving under Assumption
Contents of Talk

1 Getting Started: Overview of Scarab
   - Features
   - Architecture
   - Example: Graph Coloring Problem

2 Designing Constraint Models in Scarab
   - Pandiagonal Latin Square
   - alldiff Model
   - Boolean Cardinality Model

3 Advanced Solving Techniques using Sat4j
   - Incremental SAT Solving
   - CSP Solving under Assumption
Features

1. Expressiveness
   - Scarab provides its CP domain-specific language (Scarab DSL) embedded in Scala. We can program using both features.

2. Efficiency
   - Scarab is efficient in the sense that it uses an optimized version of the order encoding for encoding CSP into SAT.

3. Customizability
   - Scarab is 500 lines long without comments. It allows programmers to customize their own constraints.

4. Portability
   - The combination of Scarab and Sat4j enables the development of portable applications on JVM (Java Virtual Machine).

5. Availability of Advanced SAT Techniques
   - Thanks to the tight integration to Sat4j, it is available to use several SAT techniques, e.g., incremental SAT solving.
A CSP object is defined in Scarab program.

When the program calls **Scarab solver**, the CSP is encoded to a SAT object.

**Sat4j** is then called from Scarab solver to find a SAT solution.

A CSP solution (if exists) is returned back to the Scarab program by decoding the SAT solution.
Graph coloring problem (GCP) is a problem of finding a coloring of the nodes such that colors of adjacent nodes are different.

```scala
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: for (i <- nodes) int('n(i),1,colors)
9: for ((i,j) <- edges) add('n(i) !== 'n(j))
10:
11: if (find) println(solution)
```

T. Soh, N, Tamura, M. Banbara, D. Le Berre, and S. Roussel

Scarab: a Prototyping Tool for SAT-based CP Systems
Imports

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

- First 2 lines import classes of CSP and CSP solver.
- Third line imports the default CSP, Encoder, SAT Solver, and CSP Solver objects.
- It also imports DSL methods provided by Scarab.
  - `int(x, lb, ub)` method defines an integer variable.
  - `add(c)` method defines a constraint.
  - `find` method searches a solution.
  - `solution` method returns the solution.
  - etc.
val nodes = Seq(1,2,3,4,5)
val edges = Seq((1,2),(1,5),(2,3),(2,4),(3,4),(4,5))
val colors = 3

- It defines the given set of nodes and edges as the sequence object in Scala.
- Available number of colors are defined as 3.
Defining CSP

It adds an integer variable to the default CSP object by the `int` method.

’n is a notation of symbols in Scala.

They are automatically converted integer variable (Var) objects by an implicit conversion defined in Scarab.

It adds constraints to the default CSP object.

The following operators can be used to construct constraints:

- logical operator: `&&`, `||`
- comparison operator: `===`, `!==`, `<`, `<=`, `>=`, `>`
- arithmetic operator: `+`, `-`
if (find) println(solution)

- The **find** method encodes the CSP to SAT by order encoding, and call Sat4j to compute a solution.
- **solution** returns satisfiable assignment of the CSP.
Contents of Talk

1 Getting Started: Overview of Scarab
   - Features
   - Architecture
   - Example: Graph Coloring Problem

2 Designing Constraint Models in Scarab
   - Pandiagonal Latin Square
   - alldiff Model
   - Boolean Cardinality Model

3 Advanced Solving Techniques using Sat4j
   - Incremental SAT Solving
   - CSP Solving under Assumption
Pandigonal 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 $\text{alldiff}(a_1, \ldots, a_n)$ 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.

T. Soh, N. Tamura, M. Banbara, D. Le Berre, and S. Roussel

Scarab: a Prototyping Tool for SAT-based CP Systems
Pandigional Latin Square \( PLS(5) \)

- \( x_{ij} \in \{1, 2, 3, 4, 5\} \)
### Pandiagonal Latin Square $PLS(5)$

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
<th>$x_{14}$</th>
<th>$x_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td>$x_{24}$</td>
<td>$x_{25}$</td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
<td>$x_{34}$</td>
<td>$x_{35}$</td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
<td>$x_{44}$</td>
<td>$x_{45}$</td>
</tr>
<tr>
<td>$x_{51}$</td>
<td>$x_{52}$</td>
<td>$x_{53}$</td>
<td>$x_{54}$</td>
<td>$x_{55}$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>( x_{11} )</th>
<th>( x_{12} )</th>
<th>( x_{13} )</th>
<th>( x_{14} )</th>
<th>( x_{15} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{21} )</td>
<td>( x_{22} )</td>
<td>( x_{23} )</td>
<td>( x_{24} )</td>
<td>( x_{25} )</td>
<td></td>
</tr>
<tr>
<td>( x_{31} )</td>
<td>( x_{32} )</td>
<td>( x_{33} )</td>
<td>( x_{34} )</td>
<td>( x_{35} )</td>
<td></td>
</tr>
<tr>
<td>( x_{41} )</td>
<td>( x_{42} )</td>
<td>( x_{43} )</td>
<td>( x_{44} )</td>
<td>( x_{45} )</td>
<td></td>
</tr>
<tr>
<td>( x_{51} )</td>
<td>( x_{52} )</td>
<td>( x_{53} )</td>
<td>( x_{54} )</td>
<td>( x_{55} )</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
<th>$x_{14}$</th>
<th>$x_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td>$x_{24}$</td>
<td>$x_{25}$</td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
<td>$x_{34}$</td>
<td>$x_{35}$</td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
<td>$x_{44}$</td>
<td>$x_{45}$</td>
</tr>
<tr>
<td>$x_{51}$</td>
<td>$x_{52}$</td>
<td>$x_{53}$</td>
<td>$x_{54}$</td>
<td>$x_{55}$</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th></th>
<th>( x_{11} )</th>
<th>( x_{12} )</th>
<th>( x_{13} )</th>
<th>( x_{14} )</th>
<th>( x_{15} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{21} )</td>
<td></td>
<td>( x_{22} )</td>
<td>( x_{23} )</td>
<td>( x_{24} )</td>
<td>( x_{25} )</td>
</tr>
<tr>
<td>( x_{31} )</td>
<td>( x_{32} )</td>
<td>( x_{33} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_{41} )</td>
<td>( x_{42} )</td>
<td>( x_{43} )</td>
<td>( x_{44} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_{51} )</td>
<td>( x_{52} )</td>
<td>( x_{53} )</td>
<td>( x_{54} )</td>
<td>( x_{55} )</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>( x_{11} )</th>
<th>( x_{12} )</th>
<th>( x_{13} )</th>
<th>( x_{14} )</th>
<th>( x_{15} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{21} )</td>
<td>( x_{22} )</td>
<td>( x_{23} )</td>
<td>( x_{24} )</td>
<td>( x_{25} )</td>
<td></td>
</tr>
<tr>
<td>( x_{31} )</td>
<td>( x_{32} )</td>
<td>( x_{33} )</td>
<td>( x_{34} )</td>
<td>( x_{35} )</td>
<td></td>
</tr>
<tr>
<td>( x_{41} )</td>
<td>( x_{42} )</td>
<td>( x_{43} )</td>
<td>( x_{44} )</td>
<td>( x_{45} )</td>
<td></td>
</tr>
<tr>
<td>( x_{51} )</td>
<td>( x_{52} )</td>
<td>( x_{53} )</td>
<td>( x_{54} )</td>
<td>( x_{55} )</td>
<td></td>
</tr>
</tbody>
</table>

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

- \( x_{ij} \in \{1, 2, 3, 4, 5\} \)
- \( \text{alldiff in each row (5 rows)} \)
- \( \text{alldiff in each column (5 columns)} \)
- \( \text{alldiff in each pandiagonal (10 pandiagonals)} \)
The alldiff model is satisfied for a Pandiagonal Latin Square PLS(5), with constraints:

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

#### Pandiagonal Latin Square $PLS(5)$

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{11}$</td>
<td>$x_{12}$</td>
<td>$x_{13}$</td>
<td>$x_{14}$</td>
<td>$x_{15}$</td>
<td></td>
</tr>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td>$x_{24}$</td>
<td>$x_{25}$</td>
<td></td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
<td>$x_{34}$</td>
<td>$x_{35}$</td>
<td></td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
<td>$x_{44}$</td>
<td>$x_{45}$</td>
<td></td>
</tr>
<tr>
<td>$x_{51}$</td>
<td>$x_{52}$</td>
<td>$x_{53}$</td>
<td>$x_{54}$</td>
<td>$x_{55}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- 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)
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 \( \text{alldiff}(a_1, \ldots, a_n) \)

- In Pandiagonal Latin Square \( \text{PLS}(n) \), 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:**
  \[
  \neg \bigwedge_{i=1}^{n} (a_i < n) \land \neg \bigwedge_{i=1}^{n} (a_i > 1)
  \]
  It represents that mutually different \( n \) variables cannot be assigned within the interval of the size \( n - 1 \).
def alldiff(xs: Seq[Var]) =
  And(for (Seq(x, y) <- xs.combinations(2))
      yield x !== y)
def \textbf{alldiff}(xs: \text{Seq}[\text{Var}]) = \{ \\
\text{val } lb = \text{for } (x \leftarrow xs) \text{ yield } \text{csp.dom}(x).\text{lb} \\
\text{val } ub = \text{for } (x \leftarrow xs) \text{ yield } \text{csp.dom}(x).\text{ub} \\
// \text{ pigeon hole} \\
\text{val } ph = \\
\quad \text{And}(\text{Or}(\text{for } (x \leftarrow xs) \text{ yield } \neg(x < lb.\text{min}+xs.\text{size}-1)), \\
\quad \text{Or}(\text{for } (x \leftarrow xs) \text{ yield } \neg(x > ub.\text{max}-xs.\text{size}+1))) \\
// \text{ permutation} \\
\text{def } perm = \\
\quad \text{And}(\text{for } (\text{num} \leftarrow lb.\text{min} \text{ to } ub.\text{max}) \\
\quad \text{yield } \text{Or}(\text{for } (x \leftarrow xs) \text{ yield } x === \text{num})) \\
\text{val } extra = \text{if } (ub.\text{max}-lb.\text{min}+1 == xs.\text{size}) \text{ And}(ph,perm) \\
\text{else } ph \\
\quad \text{And}(\text{And}(\text{for } (\text{Seq}(x, y) \leftarrow xs.\text{combinations}(2)) \\
\quad \text{yield } x !== y),\text{extra}) \\
}
### Boolean Cardinality Model

<table>
<thead>
<tr>
<th>$y_{11k}$</th>
<th>$y_{12k}$</th>
<th>$y_{13k}$</th>
<th>$y_{14k}$</th>
<th>$y_{15k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{21k}$</td>
<td>$y_{22k}$</td>
<td>$y_{23k}$</td>
<td>$y_{24k}$</td>
<td>$y_{25k}$</td>
</tr>
<tr>
<td>$y_{31k}$</td>
<td>$y_{32k}$</td>
<td>$y_{33k}$</td>
<td>$y_{34k}$</td>
<td>$y_{35k}$</td>
</tr>
<tr>
<td>$y_{41k}$</td>
<td>$y_{42k}$</td>
<td>$y_{43k}$</td>
<td>$y_{44k}$</td>
<td>$y_{45k}$</td>
</tr>
<tr>
<td>$y_{51k}$</td>
<td>$y_{52k}$</td>
<td>$y_{53k}$</td>
<td>$y_{54k}$</td>
<td>$y_{55k}$</td>
</tr>
</tbody>
</table>

- $y_{ijk} \in \{0, 1\}$
- $y_{ijk} = 1 \iff k$ is placed at $(i, j)$
### Boolean Cardinality Model

<table>
<thead>
<tr>
<th>$y_{11k}$</th>
<th>$y_{12k}$</th>
<th>$y_{13k}$</th>
<th>$y_{14k}$</th>
<th>$y_{15k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{21k}$</td>
<td>$y_{22k}$</td>
<td>$y_{23k}$</td>
<td>$y_{24k}$</td>
<td>$y_{25k}$</td>
</tr>
<tr>
<td>$y_{31k}$</td>
<td>$y_{32k}$</td>
<td>$y_{33k}$</td>
<td>$y_{34k}$</td>
<td>$y_{35k}$</td>
</tr>
<tr>
<td>$y_{41k}$</td>
<td>$y_{42k}$</td>
<td>$y_{43k}$</td>
<td>$y_{44k}$</td>
<td>$y_{45k}$</td>
</tr>
<tr>
<td>$y_{51k}$</td>
<td>$y_{52k}$</td>
<td>$y_{53k}$</td>
<td>$y_{54k}$</td>
<td>$y_{55k}$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$y_{11k}$</th>
<th>$y_{12k}$</th>
<th>$y_{13k}$</th>
<th>$y_{14k}$</th>
<th>$y_{15k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{21k}$</td>
<td>$y_{22k}$</td>
<td>$y_{23k}$</td>
<td>$y_{24k}$</td>
<td>$y_{25k}$</td>
</tr>
<tr>
<td>$y_{31k}$</td>
<td>$y_{32k}$</td>
<td>$y_{33k}$</td>
<td>$y_{34k}$</td>
<td>$y_{35k}$</td>
</tr>
<tr>
<td>$y_{41k}$</td>
<td>$y_{42k}$</td>
<td>$y_{43k}$</td>
<td>$y_{44k}$</td>
<td>$y_{45k}$</td>
</tr>
<tr>
<td>$y_{51k}$</td>
<td>$y_{52k}$</td>
<td>$y_{53k}$</td>
<td>$y_{54k}$</td>
<td>$y_{55k}$</td>
</tr>
</tbody>
</table>

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

- for each value (5 values)
  - for each row (5 rows)
    - $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
### Boolean Cardinality Model

<table>
<thead>
<tr>
<th>$y_{11k}$</th>
<th>$y_{12k}$</th>
<th>$y_{13k}$</th>
<th>$y_{14k}$</th>
<th>$y_{15k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{21k}$</td>
<td>$y_{22k}$</td>
<td>$y_{23k}$</td>
<td>$y_{24k}$</td>
<td>$y_{25k}$</td>
</tr>
<tr>
<td>$y_{31k}$</td>
<td>$y_{32k}$</td>
<td>$y_{33k}$</td>
<td>$y_{34k}$</td>
<td>$y_{35k}$</td>
</tr>
<tr>
<td>$y_{41k}$</td>
<td>$y_{42k}$</td>
<td>$y_{43k}$</td>
<td>$y_{44k}$</td>
<td>$y_{45k}$</td>
</tr>
<tr>
<td>$y_{51k}$</td>
<td>$y_{52k}$</td>
<td>$y_{53k}$</td>
<td>$y_{54k}$</td>
<td>$y_{55k}$</td>
</tr>
</tbody>
</table>

- $y_{ijk} \in \{0, 1\}$
- $y_{ijk} = 1 \iff k$ is placed at $(i, j)$
- for each value (5 values)
  - for each row (5 rows)
    - $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns)
    - $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
### Boolean Cardinality Model

<table>
<thead>
<tr>
<th></th>
<th>( y_{11k} )</th>
<th>( y_{12k} )</th>
<th>( y_{13k} )</th>
<th>( y_{14k} )</th>
<th>( y_{15k} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_{11k} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( y_{21k} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( y_{31k} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( y_{41k} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( y_{51k} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

- For each value (5 values)
  - For each row (5 rows)
    - \( y_{11k} + y_{21k} + y_{31k} + y_{41k} + y_{51k} = 1 \)
  - For each column (5 columns)
    - \( y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1 \)
**Boolean Cardinality Model**

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

- for each value (5 values)
  - for each row (5 rows) \( y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1 \)
  - for each column (5 columns) \( y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1 \)
  - for each pandiagonal (10 pandiagonals) \( y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1 \)
## Boolean Cardinality Model

<table>
<thead>
<tr>
<th>$y_{11k}$</th>
<th>$y_{12k}$</th>
<th>$y_{13k}$</th>
<th>$y_{14k}$</th>
<th>$y_{15k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{21k}$</td>
<td>$y_{22k}$</td>
<td>$y_{23k}$</td>
<td>$y_{24k}$</td>
<td>$y_{25k}$</td>
</tr>
<tr>
<td>$y_{31k}$</td>
<td>$y_{32k}$</td>
<td>$y_{33k}$</td>
<td>$y_{34k}$</td>
<td>$y_{35k}$</td>
</tr>
<tr>
<td>$y_{41k}$</td>
<td>$y_{42k}$</td>
<td>$y_{43k}$</td>
<td>$y_{44k}$</td>
<td>$y_{45k}$</td>
</tr>
<tr>
<td>$y_{51k}$</td>
<td>$y_{52k}$</td>
<td>$y_{53k}$</td>
<td>$y_{54k}$</td>
<td>$y_{55k}$</td>
</tr>
</tbody>
</table>

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

**for each value (5 values)**
- **for each row (5 rows)**
  - $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
- **for each column (5 columns)**
  - $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
- **for each pandiagonal (10 pandiagonals)**
  - $y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$
**Boolean Cardinality Model**

<table>
<thead>
<tr>
<th>$y_{11k}$</th>
<th>$y_{12k}$</th>
<th>$y_{13k}$</th>
<th>$y_{14k}$</th>
<th>$y_{15k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{21k}$</td>
<td>$y_{22k}$</td>
<td>$y_{23k}$</td>
<td>$y_{24k}$</td>
<td>$y_{25k}$</td>
</tr>
<tr>
<td>$y_{31k}$</td>
<td>$y_{32k}$</td>
<td>$y_{33k}$</td>
<td>$y_{34k}$</td>
<td>$y_{35k}$</td>
</tr>
<tr>
<td>$y_{41k}$</td>
<td>$y_{42k}$</td>
<td>$y_{43k}$</td>
<td>$y_{44k}$</td>
<td>$y_{45k}$</td>
</tr>
<tr>
<td>$y_{51k}$</td>
<td>$y_{52k}$</td>
<td>$y_{53k}$</td>
<td>$y_{54k}$</td>
<td>$y_{55k}$</td>
</tr>
</tbody>
</table>

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

$y_{ijk} = 1 \iff k$ is placed at $(i, j)$

- for each value (5 values)
  - for each row (5 rows) $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals) $y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$
Boolean Cardinality Model

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

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

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

- 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 ;)

T. Soh, N. Tamura, M. Banbara, D. Le Berre, and S. Roussel

**Scarab: a Prototyping Tool for SAT-based CP Systems**
Contents of Talk

1 Getting Started: Overview of Scarab
   - Features
   - Architecture
   - Example: Graph Coloring Problem

2 Designing Constraint Models in Scarab
   - Pandiagonal Latin Square
   - alldiff Model
   - Boolean Cardinality Model

3 Advanced Solving Techniques using Sat4j
   - Incremental SAT Solving
   - CSP Solving under Assumption
Thanks to the **tight integration to Sat4j**, Scarab provides the functions: Incremental solving and CSP solving with assumptions.

We explain it using the following program.

```plaintext
1: int('x, 1, 3)  
2: int('y, 1, 3)  
3: add('x === 'y)  
4: find  // first call of find  
5: add('x !== 3)  
6: find  // second call of find  
7:  
8: find('y === 3)  // with assumption y = 3  
9: find('x === 1)  // with assumption x = 1
```
In the first call of `find` method, the whole CSP is encoded and generated SAT clauses are added to Sat4j, then it computes a solution.

In the second call of `find` method, only the extra constraint $x \neq 3$ is encoded and added to Sat4j, then it computes a solution.

The learned clauses obtained by the first `find` are kept at the second call.
CSP Solving under Assumption

```javascript
find('y === 3') // with assumption y = 3
find('x === 1') // with assumption x = 1
```

- `find(assumption: Constraint)` method provides CSP solving under assumption given by the specified constraint.
- The constraint of assumption should be encoded to a conjunction of literals (otherwise an exception is raised).
- Then, the literals are passed to Sat4j, then it computes a solution under assumption.
- We can utilize those techniques for optimization and enumeration problems.
Introduction

Using Scarab, we can write various constraint models without developing dedicated encoders, which allows us to focus on problem modeling and encoding.

Future Work

- Introducing more features from Sat4j
- Sat4j has various functions of finding MUS, optimization, solution enumeration, handling natively cardinality and pseudo-Boolean constraints.

URL of Scarab

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

Scarab also appear on tool demo and poster session on Friday (10:05-11:30)
Supplemental Slides
BC1: Pairwise

Definition of BC1

```
def BC1(xs: Seq[Var]): Term = Sum(xs)
```
**BC1: Pairwise (cont.)**

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

\[
\begin{align*}
\text{int('x,0,1)} \\
\text{int('y,0,1)} \\
\text{int('z,0,1)} \\
\text{add(BC1(Seq('x, 'y, 'z)) === 1)}
\end{align*}
\]

**CNF Generated by Scarab**

\[
\begin{align*}
p(x \leq 0) \lor p(y \leq 0) \\
p(x \leq 0) \lor p(z \leq 0) \\
p(y \leq 0) \lor p(z \leq 0) \\
\neg p(x \leq 0) \lor \neg p(y \leq 0) \lor \neg p(z \leq 0)
\end{align*}
\]

\[
\begin{align*}
x + y + z \leq 1 \\
x + y + z \geq 1
\end{align*}
\]
**Definition of BC2**

```scala
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
  }
}
```
### Scarab Program for $x + y + z = 1$

```plaintext
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)

\[
\begin{align*}
q & \lor \neg p(y \leq 0) \lor \neg p(z \leq 0) \\
\neg q & \lor p(z \leq 0) \\
\neg q & \lor p(y \leq 0) \\
p(y \leq 0) & \lor p(z \leq 0) \\
q & \lor p(x \leq 0) \\
\neg q & \lor \neg p(x \leq 0)
\end{align*}
\]

\[
\begin{align*}
y + z &= S \\
x + S &= 1
\end{align*}
\]
Definition of BC3

```scala
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** $x + y + z = 1$

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

**CNF Generated by Scarab (q₁ and q₂ are auxiliary variables)**

\[
\begin{align*}
q₁ &\lor \neg p(y \leq 0) &\lor \neg p(x \leq 0) \\
\neg q₁ &\lor p(x \leq 0) \\
\neg q₁ &\lor p(y \leq 0) \\
&\quad \lor p(x \leq 0) &\lor p(y \leq 0) \\
q₂ &\lor \neg q₁ &\lor \neg p(z \leq 0) \\
\neg q₂ &\lor q₁ \\
\neg q₂ &\lor p(z \leq 0) \\
q₁ &\lor p(z \leq 0) \\
\neg q₂
\end{align*}
\]

\[
\begin{align*}
\left\{ \begin{array}{l}
x + y = S₁ \\
S₁ + z = S₂ \\
\end{array} \right. \\
\} \quad S₂ = 1
\]
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)

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

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

Example of $SP(15, 36)$

- **Optimum solution 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 ($1 \leq i < j \leq n$)**

$$(x_i + i \leq x_j) \lor (x_j + j \leq x_i) \lor (y_i + i \leq y_j) \lor (y_j + j \leq y_i)$$
Decremental Search

**Scarab Program for $SP(n, s)$**

```plaintext
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**

```plaintext
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)) {
  add('m <= ub)
  ub = solution.intMap('m) - 1
}
```
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
    }
}
```