SAT ソルバーとそのアプリケーション開発について (後半: SAT 型制約ソルバー)

宋剛秀 (神戸大学) web　鍋島英知 (山梨大学) web

共同研究者
田村直之 (神戸大学) web　番原睦則 (神戸大学) web
井上克巳 (国立情報学研究所) web

人工知能学会 第 9 回 AI ツール入門講座
(2015.12.14 @ 国立情報学研究所)

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)
}
```

BC1: Pairwise

BC1 の定義

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

BC1: Pairwise (cont.)

$x + y + z = 1$ に対する Scarab プログラム

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

CNF Generated by Scarab

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

$$\left.\neg p(x \leq 0) \ \vee \ \neg p(y \leq 0) \ \vee \ \neg p(z \leq 0)\right\} x + y + z \geq 1$$

BC2: [?]

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: [?] (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)

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

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