Generation of Implied Constraints for Automaton-Induced Decompositions

M. Andreína Francisco, Pierre Flener, Justin Pearson

ASTRA Research Group on CP
Uppsala University

Sweden

November 6, 2013

Background

Although modern constraint solvers have many global constraints, often a constraint that one is looking for is not there.

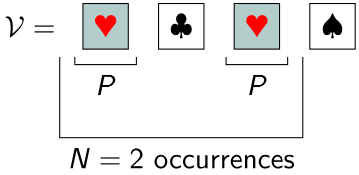
In [Beldiceanu, Carlsson, and Petit, CP2004], a framework is given for defining global constraints by an automaton, possibly with counters, which corresponds to a constraint checker. Using the automaton, the new constraint is decomposed into a conjunction of already implemented constraints.

Currently there are about 120 constraints in the Global Constraint Catalog defined by one or more automata.

Example: The Exactly Constraint

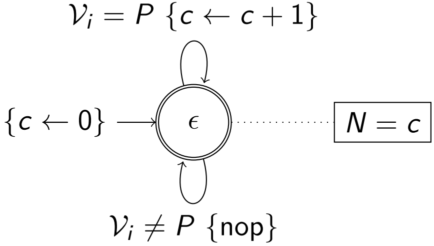
The $\textsc{Exactly}(N, \mathcal{V}, P)$ constraint holds if and only if the sequence $\mathcal{V}$ of decision variables contains exactly N elements taking the given value P.

$$\textsc{Exactly}(2, [\heartsuit, \clubsuit, \heartsuit, \spadesuit], \heartsuit)$$



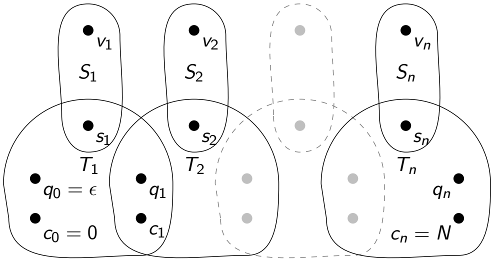
Example: The EXACTLY Constraint

```
function EXACTLY($N$,$\mathcal{V}$,$P$)
    $i \leftarrow 0$
    $c \leftarrow 0$
    while $i < |\mathcal{V}|$ do
        if $\mathcal{V}[i] = P$ then
            $c \leftarrow c + 1$
        $i \leftarrow i + 1$
    return $N = c$
```



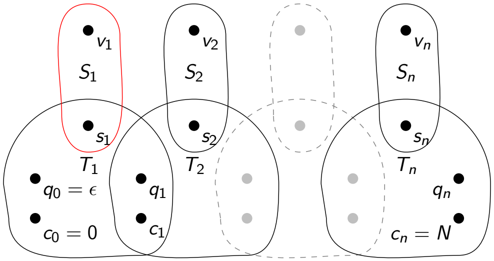
$\mathcal{V}_i = P$ $\{c \leftarrow c + 1\}$
$\{c \leftarrow 0\}$
ϵ
$N = c$
$\mathcal{V}_i \neq P$ $\{\text{nop}\}$

Example: The EXACTLY Constraint



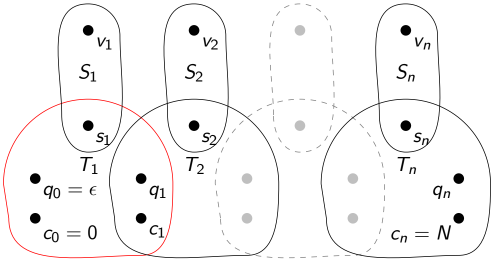
$$\bigwedge_{i=1}^{n}(\mathcal{V}_i = P \Leftrightarrow s_i = \text{true}) \wedge (\mathcal{V}_i \neq P \Leftrightarrow s_i = \text{false})$$
$$\wedge\ \textsc{Trans}(q_0, c_0, s_1, q_1, c_1) \wedge\ \cdots \wedge\ \textsc{Trans}(q_{n-1}, c_{n-1}, s_n, q_n, c_n) \wedge$$
$$q_0 = \epsilon \wedge c_0 = 0 \wedge\ c_n = N$$

Example: The EXACTLY Constraint



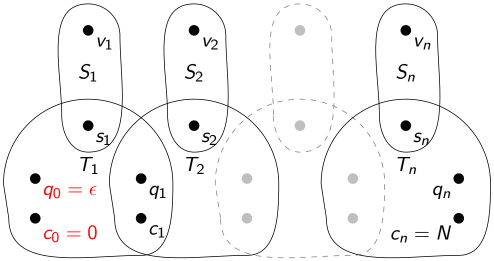
$$\bigwedge_{i=1}^{n} (\mathcal{V}_i = P \Leftrightarrow s_i = \text{true}) \wedge (\mathcal{V}_i \neq P \Leftrightarrow s_i = \text{false})$$
$$\wedge \ \textsc{Trans}(q_0, c_0, s_1, q_1, c_1) \wedge \ \cdots \wedge \ \textsc{Trans}(q_{n-1}, c_{n-1}, s_n, q_n, c_n) \ \wedge$$
$$q_0 = \epsilon \wedge c_0 = 0 \wedge \ c_n = N$$

Example: The Exactly Constraint



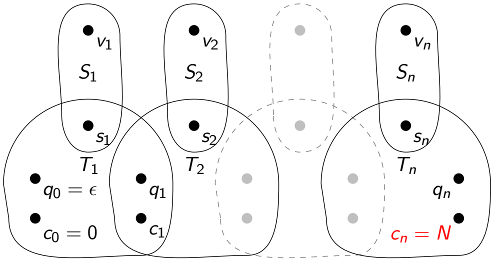
$$\bigwedge_{i=1}^{n} (\mathcal{V}_i = P \Leftrightarrow s_i = \text{true}) \wedge (\mathcal{V}_i \neq P \Leftrightarrow s_i = \text{false})$$
$$\wedge\ \textsc{Trans}(q_0, c_0, s_1, q_1, c_1) \wedge\ \cdots \wedge\ \textsc{Trans}(q_{n-1}, c_{n-1}, s_n, q_n, c_n) \wedge$$
$$q_0 = \epsilon \wedge c_0 = 0 \wedge\ c_n = N$$

Example: The EXACTLY Constraint



$$\bigwedge_{i=1}^{n}(\mathcal{V}_i = P \Leftrightarrow s_i = \text{true}) \wedge (\mathcal{V}_i \neq P \Leftrightarrow s_i = \text{false})$$
$$\wedge\ \textsc{Trans}(q_0, c_0, s_1, q_1, c_1) \wedge\ \cdots \wedge\ \textsc{Trans}(q_{n-1}, c_{n-1}, s_n, q_n, c_n) \wedge$$
$$q_0 = \epsilon \wedge c_0 = 0 \wedge\ c_n = N$$

Example: The EXACTLY Constraint



$$\bigwedge_{i=1}^{n} (\mathcal{V}_i = P \Leftrightarrow s_i = \text{true}) \wedge (\mathcal{V}_i \neq P \Leftrightarrow s_i = \text{false})$$
$$\wedge\ \textsc{Trans}(q_0, c_0, s_1, q_1, c_1) \wedge\ \cdots \wedge\ \textsc{Trans}(q_{n-1}, c_{n-1}, s_n, q_n, c_n) \wedge$$
$$q_0 = \epsilon \wedge c_0 = 0 \wedge\ c_n = N$$

Motivation

Unfortunately, generalised arc consistency (GAC) is in general not maintained on decompositions induced by automata with counters, even if GAC is maintained individually on the constraints.

For example, when decomposing the $\textsc{Distinct}(x, y, z)$ constraint into $x \neq y$, $y \neq z$, $z \neq x$, some propagation is lost.

$$D(x) = \{1, 2\},\ D(y) = \{1, 2\},\ D(z) = \{1, 2, 3\}$$

We propose a method to improve propagation for automaton-induced decompositions!

Our Work

We added the following steps to the design process:

1. Derive loop invariants
   - It is possible to use an automatic invariant generator like InvGen [CAV2009]
   - Normally requires code manipulation in order to:
     - derive disjunctive invariants
     - consider previous values of variables

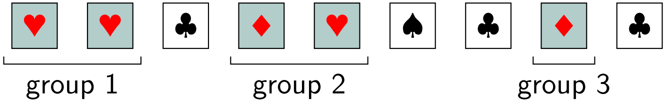
2. Rewrite invariants as implied constraints and add them to the decomposition, yielding the *extended* decomposition.

Example: The NGROUP Constraint

In a sequence, a **group** is a contiguous subsequence with values from a given set.

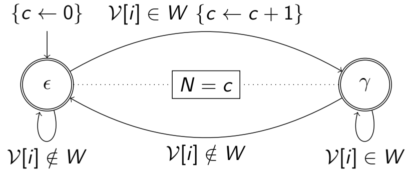
The NGROUP$(N, \mathcal{V}, W)$ constraint holds if and only if there are N groups of values from the set W in the sequence $\mathcal{V}$ of decision variables.

$$\textsc{nGroup}(3, [\heartsuit, \heartsuit, \clubsuit, \diamondsuit, \heartsuit, \spadesuit, \clubsuit, \diamondsuit, \clubsuit], \{\heartsuit, \diamondsuit\})$$



The nGroup Constraint

```
function nGroup($N$,$\mathcal{V}$,$W$)
    $q \leftarrow \epsilon$;  $i \leftarrow 0$
    while $i < |\mathcal{V}|$ do
        if $\mathcal{V}[i] \in W$ then
            if $q = \epsilon$ then
                $c \leftarrow c + 1$
                $q \leftarrow \gamma$
        else
            $q \leftarrow \epsilon$
        $i \leftarrow i + 1$
    return $N = c$
```


$\{c \leftarrow 0\}$
$\mathcal{V}[i] \in W$ $\{c \leftarrow c+1\}$
ϵ
$N = c$
γ
$\mathcal{V}[i] \notin W$
$\mathcal{V}[i] \notin W$
$\mathcal{V}[i] \in W$

This checker does not have interesting invariants

The nGroup Constraint

```
function nGroup($N$,$\mathcal{V}$,$W$)
    $c \leftarrow 0$;  $c_1 \leftarrow 0$;  $c_2 \leftarrow 0$
    $q \leftarrow \epsilon$;  $i \leftarrow 0$
    while $i < |\mathcal{V}|$ do
        $c_2 \leftarrow c_1$;  $c_1 \leftarrow c$
        if $\mathcal{V}[i] \in W$ then
            if $q = \epsilon$ then
                $c \leftarrow c + 1$
                $q \leftarrow \gamma$
        else
            $q \leftarrow \epsilon$
        $i \leftarrow i + 1$
    return $N = c$
```

We manipulate the checker to keep track of previous values of the variable c.
We obtain the invariants

$$c_2 \leq c \leq c_2 + 1$$

Rewrite invariants as constraints.

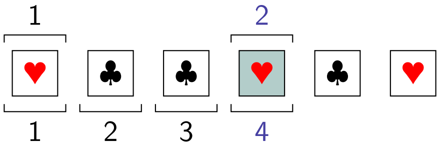
$$c_{i-2} \leq c_i \leq c_{i-2} + 1$$

for all $1 < i \leq |\mathcal{V}|$

Example: The JthNonZeroPos Constraint

The JthNonZeroPos$(J, P, \mathcal{V})$ constraint holds if and only if the J^{th} non-zero element is at position P (counting from 1) of the sequence $\mathcal{V}$ of decision variables. Parameter J must be a constant.

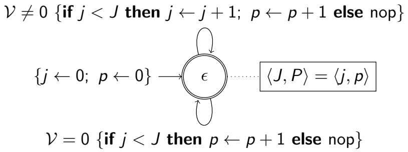
JthNonZeroPos(2, 4, [♥, ♣, ♣, ♥, ♣, ♥])



♥ symbols are interpreted as NonZero

The JTHNONZEROPOS Constraint

```
function JTHNONZEROPOS($J$,$P$,$\mathcal{V}$)
    $i \leftarrow 0$
    $j \leftarrow 0$
    $p \leftarrow 0$
    while $i < |\mathcal{V}|$ do
        if $j < J$ then
            if $\mathcal{V}[i] \neq 0$ then
                $j \leftarrow j + 1$
            $p \leftarrow p + 1$
        $i \leftarrow i + 1$
    return $j = J \wedge p = P$
```



$\mathcal{V} \neq 0$ {**if** $j < J$ **then** $j \leftarrow j + 1$; $p \leftarrow p + 1$ **else** nop}
$\{j \leftarrow 0;\ p \leftarrow 0\}$
ϵ
$\langle J, P\rangle = \langle j, p\rangle$
$\mathcal{V} = 0$ {**if** $j < J$ **then** $p \leftarrow p + 1$ **else** nop}

The $\textsc{JthNonZeroPos}$ Constraint

```
function $\textsc{JthNonZeroPos}(J,P,\mathcal{V})$
    $i \leftarrow 0$
    $j \leftarrow 0$
    $p \leftarrow 0$
    while $i < |\mathcal{V}|$ do
        if $j < J$ then
            if $\mathcal{V}[i] \neq 0$ then
                $j \leftarrow j + 1$
            $p \leftarrow p + 1$
        $i \leftarrow i + 1$
    return $j = J \land p = P$
```

This checker has the following invariants:

$$j \leq J$$

$$0 \leq j \leq p \leq i$$

Which can be rewriten as the following constraints:

$$j_i \leq J$$

$$0 \leq j_i \leq p_i \leq i$$

The JthNonZeroPos Constraint

```
function JthNonZeroPos($J$,$P$,$\mathcal{V}$)
    $i \leftarrow 0$; $j \leftarrow 0$
    $p \leftarrow 0$
    while $i < |\mathcal{V}| \wedge j < J$ do
        if $\mathcal{V}[i] \neq 0$ then
            $j \leftarrow j + 1$
        $p \leftarrow p + 1$
        $i \leftarrow i + 1$
    while $i < |\mathcal{V}| \wedge j \geq J$ do
        $i \leftarrow i + 1$
    return $j = J \wedge p = P$
```

(Technique by Sharma et al CAV2011)
This checker has the following invariants:

$$(j \leq J \wedge p = i) \vee (j = J \wedge p \leq i)$$

These invariants allow us to derive the following implied constraint:

$$j_i < J \Rightarrow p_i = i$$

The JthNonZeroPos Constraint

```
function JthNonZeroPos($J$,$P$,$\mathcal{V}$)
    $i \leftarrow 0$;  $j \leftarrow 0$;  $p \leftarrow 0$
    while $i < |\mathcal{V}| \wedge j < J - 1$ do
        if $\mathcal{V}[i] \neq 0$ then
            $j \leftarrow j + 1$
        $p \leftarrow p + 1$
        $i \leftarrow i + 1$
    while $i < |\mathcal{V}| \wedge j \geq J - 1 \wedge \mathcal{V}[i] = 0$ do
        $p \leftarrow p + 1$
        $i \leftarrow i + 1$
    return $j = J - 1 \wedge p = P - 1 \wedge \mathcal{V}[i] \neq 0$
```

This checker has the following invariants:

$$(j \leq J1 \wedge p = i) \\ \vee \\ (j = J1 \wedge s = z \wedge p = i)$$

These invariants allow us to derive the following implied constraint:

$$(s_i = z \vee j_i1 \neq J1) \\ \Rightarrow p_{i+1} \neq i$$

Experiments

1. We compared in SICStus Prolog version 4.2.1 the original and extended decompositions of $\textsc{nGroup}(N, \mathcal{V}, W)$ and the $\textsc{JthNonZeroPos}(J, P, \mathcal{V})$ constraints.

2. We generated instances with random amounts of variables ($|\mathcal{V}| \leq 50$), as well as random initial domains of N, P and $\mathcal{V}_i$ (one value, two values, and intervals of length 2 or 3).

3. We ran millions of instances.

Results

- The extended decomposition of the $\textsc{JthNonZeroPos}(J, P, \mathcal{V})$ maintains generalized arc consistency.

- The extended decomposition of the $\textsc{nGroup}(N, \mathcal{V}, W)$ is on average 2% faster, prunes 105% more and detects 8% more failures when calculating the fixpoint of the constraint.

- Note that these implied constraints do not increase the time or space complexity of computing the common fixpoint of the decomposition (formal proof in the paper).

Future Work

The creative process on how to modify the checker in order to find the right invariants is still manual.

We will now automate as many steps as possible of this methodology.

In particular, we want to automate the process of modifying the checker and testing whether or not an implied constraint is useful or not.

Questions?