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
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 **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 $P$.

$$\text{EXACTLY}(2, [\heartsuit, \clubsuit, \heartsuit, \spadesuit], \heartsuit)$$

$$\mathcal{V} = \begin{array}{cccc}
\heartsuit & \clubsuit & \heartsuit & \spadesuit \\
\hline
P & & P \\
\end{array}$$

$N = 2$ occurrences
Example: The **EXACTLY** Constraint

```
function EXACTLY(N, V, P)
  i ← 0
  c ← 0
  while i < |V| do
    if V[i] = P then
      c ← c + 1
    i ← i + 1
  return N = c
```

\[
\begin{align*}
\forall i &= \text{ } P \{c \leftarrow c + 1\} \\
\{c \leftarrow 0\} &\quad \text{ } \epsilon \quad \text{ } \quad N = c \\
\forall i &\neq P \{\text{nop}\}
\end{align*}
\]
Example: The **Exactly** Constraint

\[\bigwedge_{i=1}^{n} (V_i = P \iff s_i = \text{true}) \land (V_i \neq P \iff s_i = \text{false}) \land \text{TRANS}(q_0, c_0, s_1, q_1, c_1) \land \ldots \land \text{TRANS}(q_{n-1}, c_{n-1}, s_n, q_n, c_n) \land q_0 = \epsilon \land c_0 = 0 \land c_n = N\]
Example: The \textbf{E}XACTLY Constraint

\[ \bigwedge_{i=1}^{n} (v_i = P \Leftrightarrow s_i = \text{true}) \land (v_i \neq P \Leftrightarrow s_i = \text{false}) \land \text{TRANS}(q_0, c_0, s_1, q_1, c_1) \land \cdots \land \text{TRANS}(q_{n-1}, c_{n-1}, s_n, q_n, c_n) \land q_0 = \epsilon \land c_0 = 0 \land c_n = N \]
Example: The **EXACTLY** Constraint

\[ \bigwedge_{i=1}^{n} (V_i = P \iff s_i = \text{true}) \land (V_i \neq P \iff s_i = \text{false}) \land \text{TRANS}(q_0, c_0, s_1, q_1, c_1) \land \cdots \land \text{TRANS}(q_{n-1}, c_{n-1}, s_n, q_n, c_n) \land q_0 = \epsilon \land c_0 = 0 \land c_n = N \]
Example: The **EXACTLY** Constraint

\[ \bigwedge_{i=1}^{n}(\forall_i = P \iff s_i = \text{true}) \land (\forall_i \neq P \iff s_i = \text{false}) \]
\[ \land \text{TRANS}(q_0, c_0, s_1, q_1, c_1) \land \cdots \land \text{TRANS}(q_{n-1}, c_{n-1}, s_n, q_n, c_n) \land \]
\[ q_0 = \epsilon \land c_0 = 0 \land c_n = N \]
Example: The EXACTLY Constraint

\[ \bigwedge_{i=1}^{n} (\forall_i = P \iff s_i = \text{true}) \land (\forall_i \neq P \iff s_i = \text{false}) \land \text{TRANS}(q_0, c_0, s_1, q_1, c_1) \land \cdots \land \text{TRANS}(q_{n-1}, c_{n-1}, s_n, q_n, c_n) \land q_0 = \epsilon \land c_0 = 0 \land c_n = N \]

5 of 19
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.

\[
D(x) = \{1, 2\}, \quad D(y) = \{1, 2\}, \quad D(z) = \{1, 2, 3\}
\]

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

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

- group 1

- group 2

- group 3
The \texttt{nGroup} Constraint

\begin{verbatim}
function nGroup(N, V, W)
    q ← ε; i ← 0
    while i < |V| do
        if V[i] ∈ W then
            if q = ε then
                c ← c + 1
                q ← γ
            else
                q ← ε
        else
            q ← ε
        i ← i + 1
    return N = c
\end{verbatim}

This checker does not have interesting invariants
The \texttt{nGroup} Constraint

\begin{function} \texttt{nGroup}(N, V, W) \end{function}
\begin{align*}
&c \leftarrow 0; \quad c_1 \leftarrow 0; \quad c_2 \leftarrow 0 \\
&q \leftarrow \epsilon; \quad i \leftarrow 0 \\
&\textbf{while } i < |V| \textbf{ do} \\
&\quad c_2 \leftarrow c_1; \quad c_1 \leftarrow c \\
&\quad \textbf{if } V[i] \in W \textbf{ then} \\
&\quad \quad \textbf{if } q = \epsilon \textbf{ then} \\
&\quad \quad \quad c \leftarrow c + 1 \\
&\quad \quad \quad q \leftarrow \gamma \\
&\quad \textbf{else} \\
&\quad \quad q \leftarrow \epsilon \\
&\quad i \leftarrow i + 1 \\
&\textbf{return } N = c
\end{align*}

We manipulate the checker to keep track of previous values of the variable $c$.
We obtain the invariants
\begin{align*}
&c_2 \leq c \leq c_2 + 1 \\
&\text{Rewrite invariants as constraints.}
\end{align*}
\begin{align*}
&c_{i-2} \leq c_i \leq c_{i-2} + 1 \\
&\text{for all } 1 < i \leq |V|
\end{align*}
Example: The $\text{JthNonZeroPos}$ Constraint

The $\text{JthNonZeroPos}(J, P, \mathcal{V})$ 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.

$\text{JthNonZeroPos}(2, 4, [\heartsuit, \spadesuit, \spadesuit, \heartsuit, \spadesuit, \heartsuit])$

♥ symbols are interpreted as NonZero
The JthNonZeroPos Constraint

function JthNonZeroPos(J, P, V)
    i ← 0
    j ← 0
    p ← 0
    while i < |V| do
        if j < J then
            if V[i] ≠ 0 then
                j ← j + 1
            p ← p + 1
        i ← i + 1
    return j = J ∧ p = P
The \texttt{JthNonZeroPos} Constraint

\begin{verbatim}
function \texttt{JthNonZeroPos}(J, P, V)  
i \leftarrow 0 
j \leftarrow 0 
p \leftarrow 0 
while i \leq \lvert V \rvert do 
    if j < J then 
        if V[i] \neq 0 then 
            j \leftarrow j + 1
            p \leftarrow p + 1
        i \leftarrow i + 1
return j = J \land p = P
\end{verbatim}

This checker has the following invariants:

\[ j \leq J \]
\[ 0 \leq j \leq p \leq i \]

Which can be rewritten as the following constraints:

\[ j_i \leq J \]
\[ 0 \leq j_i \leq p_i \leq i \]
The JthNonZeroPos Constraint

function JthNonZeroPos(J, P, V) {
    i ← 0; j ← 0
    p ← 0
    while i < |V| ∧ j < J do
        if V[i] ≠ 0 then
            j ← j + 1
            p ← p + 1
            i ← i + 1
        end
    end
    while i < |V| ∧ j ≥ J do
        i ← i + 1
    end
    return j = J ∧ p = P
}

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

(j ≤ J ∧ p = i) ∨ (j = J ∧ p ≤ i)

These invariants allow us to derive the following implied constraint:

j_i < J ⇒ p_i = i
The \texttt{JthNonZeroPos} Constraint

\begin{verbatim}
function \texttt{JthNonZeroPos}(J,P,V)
    i ← 0; j ← 0; p ← 0
    while i < |V| ∧ j < J - 1 do
        if V[i] ≠ 0 then
            j ← j + 1
            p ← p + 1
            i ← i + 1
        \end{verbatim}
        \begin{verbatim}
        while i < |V| ∧ j ≥ J - 1 ∧ V[i] = 0 do
            p ← p + 1
            i ← i + 1
        \end{verbatim}
    \end{verbatim}
    \begin{verbatim}
    return j = J - 1 ∧ p = P - 1 ∧ V[i] ≠ 0
\end{verbatim}

This checker has the following invariants:

\begin{align*}
(j ≤ J1 ∧ p = i) \\
∨ (j = J1 ∧ s = z ∧ p = i)
\end{align*}

These invariants allow us to derive the following implied constraint:

\begin{align*}
(s_i = z ∨ j_{i+1} = J1) \\
⇒ p_{i+1} ≠ i
\end{align*}
Experiments

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

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

3. We ran millions of instances.
Results

- The extended decomposition of the $J^{thNonZeroPos}(J, P, V)$ maintains generalized arc consistency.

- The extended decomposition of the $nGroup(N, V, W)$ 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?