# Implied Constraints for CP solvers

**M. Andreína Francisco** Pierre Flener Justin Pearson

Department of Information Technology

Uppsala University

Sweden

October 18, 2015

# Background: Combinatorial optimisation

**Combinatorial optimisation** consists of finding an object from a finite set of objects:

- The set of feasible solutions is discrete or can be discretised.
- The goal is to find a solution, or all solutions, or a best solution.
- Examples:
  - the travelling tournament problem.
  - the nurse scheduling problem.

**Constraint programming** (CP) is a set of techniques and tools for effectively modelling and efficiently solving hard combinatorial problems.

CP solving = inference + search

# Background: CP Modelling

**Constraints** form the vocabulary of a CP modelling language: they allow a modeller to express commonly occurring substructures.

**Example**

The $\textsc{Distinct}(x, y, z)$ constraint, over the variables $x$,$y$, and $z$ with domains $x \in \{1, 2\}, y \in \{1, 2\}$, and $z \in \{2, 3, 4, 5\}$

A **constraint problem** is a conjunction of constraints.

**Example**

$$\textsc{Distinct}(x, y, z) \wedge x + y < z$$

# Background: CP Inference and Search

A constraint comes with a **propagator**, which removes impossible values from the domains of its variables.

## Example

$\textsc{Distinct}(x, y, z)$ with $x = \{1, 2\}, y = \{1, 2\}$, and $z = \{\cancel{2}, 3, 4, 5\}$

After the propagators have removed the values they can, the solver will begin a systematic search if need be:

- Select a variable
- Select a value (or a range of values)
- Propagate again on the domain of the variables

## Example

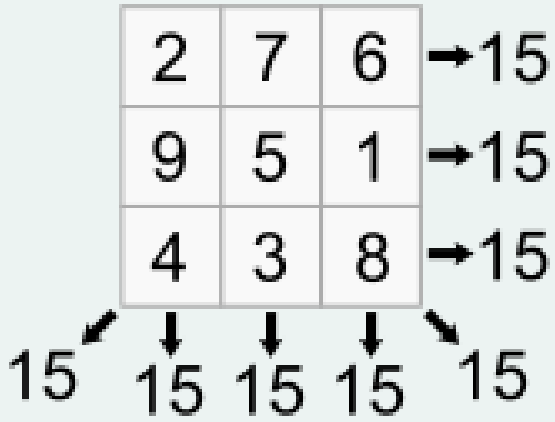
$x = 1$: $\textsc{Distinct}(x, y, z)$, $x = \{1, \cancel{2}\}, y = \{\cancel{1}, 2\}$, and $z = \{3, 4, 5\}$
$x \neq 1$: $\textsc{Distinct}(x, y, z)$, $x = \{\cancel{1}, 2\}, y = \{1, \cancel{2}\}$, and $z = \{3, 4, 5\}$

# Background: Implied Constraints

An **implied constraint** is a constraint that logically follows from other constraints.

Implied constraints may improve propagation or running time, without losing any solutions.

## Example (Magic Square)



# Background

Although modern CP solvers have many built-in constraints, often a constraint that one is looking for is not there. In such cases, the choices are:

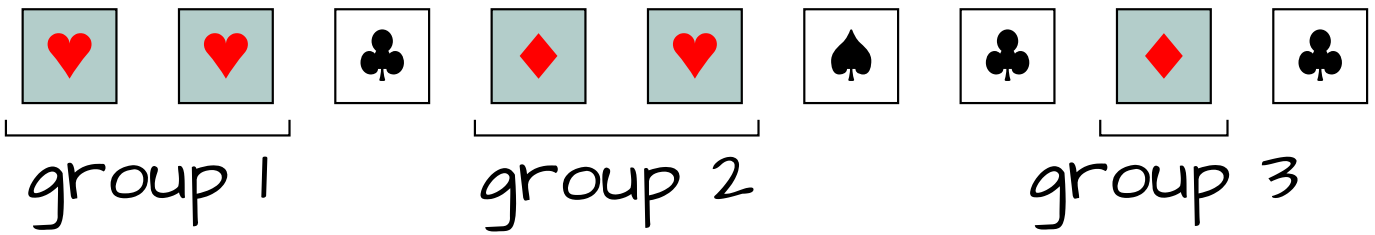
- to reformulate the model without the needed constraint.
- to write one's own propagator.

For example, deterministic finite automata (DFA) augmented with accumulators can encode a constraint on a sequence $S$ of variables [BCP04].

# Example: The Group Constraint

The $\textsc{Group}(S, N, W)$ constraint holds if and only if there are $N$ contiguous subsequences of values from the given set $W$ in the sequence $S$ of variables.

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

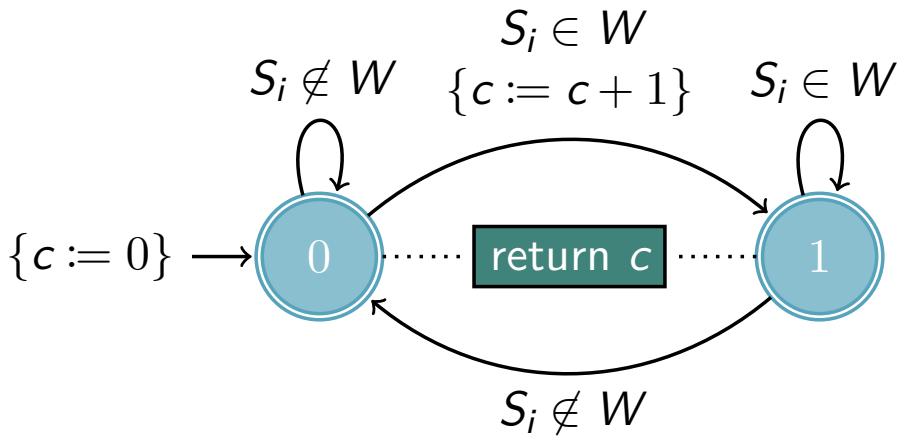


# Example: The Group Constraint

The $\textsc{Group}(S, N, W)$ constraint holds if and only if there are $N$ contiguous subsequences of values from the given set $W$ in the sequence $S$ of variables.

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

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 |

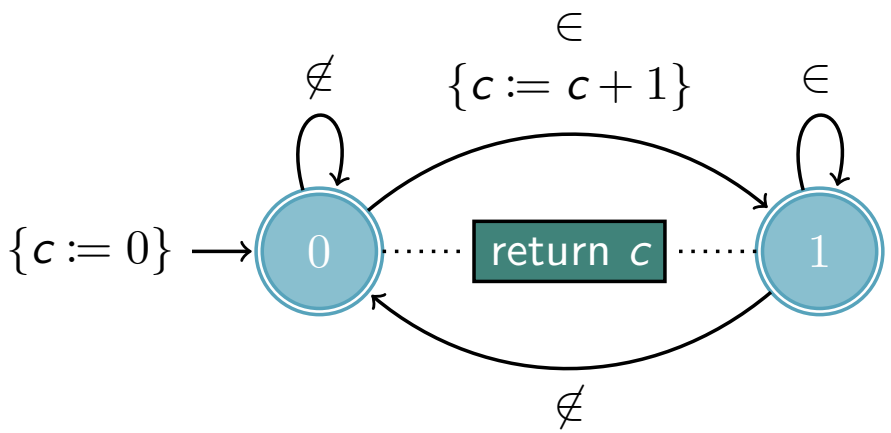


# Example: The Group Constraint

The $\textsc{Group}(S, N, W)$ constraint holds if and only if there are $N$ contiguous subsequences of values from the given set $W$ in the sequence $S$ of variables.

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

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 |



# Motivation

In general, it is hard to do perfect propagation efficiently for constraints encoded via automata [BCP04].

**Implied constraints on accumulators** are a way to improve propagation, as these may trigger extra propagation at each node of the search tree [BCRT05, FFP13].

# Our Objective

Generate implied constraints that improve propagation for constraints encoded via automata with at least one accumulator.

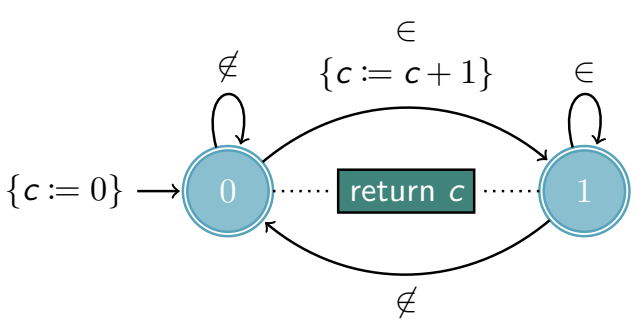
- The generation is specific to the given automaton $\mathcal{A}$
- The implied constraints are generated offline.
- The implied constraints are of the form:

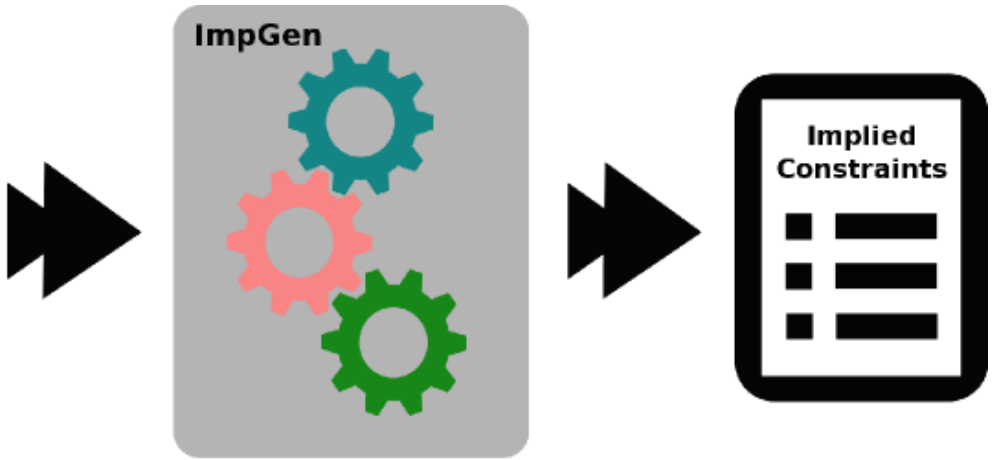
$$\alpha_1 y_1 + \cdots + \alpha_k y_k + \beta \geq 0$$

where the $y_i$ are the accumulators of $\mathcal{A}$ and the weights $\alpha_i$ and $\beta$ are to be found.

# Our Work

We developed a tool based on Farkas' Lemma.




Options

# Farkas' Lemma

A set of $e$ linear inequalities over real-valued variables $y_i$ has another linear inequality over the same variables as a logical consequence if the latter is equal to a linear combination of the former.

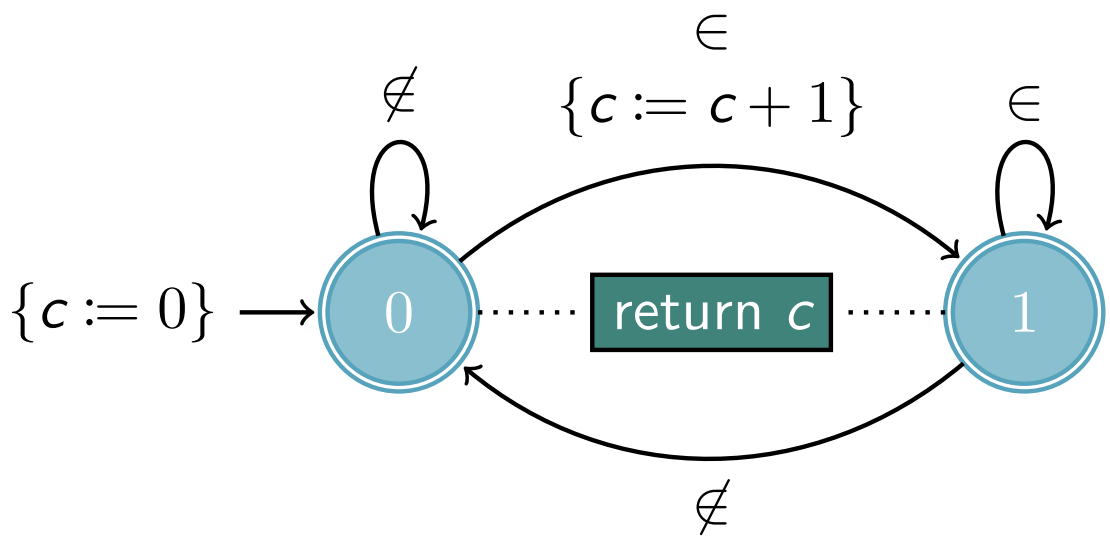
$$\begin{array}{c|c} \lambda_1 & a_{11}y_1 + \cdots + a_{1k}y_k + b_1 \geq 0 \\ \vdots & \vdots \quad \vdots \quad \vdots \quad \vdots \\ \lambda_e & a_{e1}y_1 + \cdots + a_{ek}y_k + b_e \geq 0 \\ \hline & \alpha_1 y_1 + \cdots + \alpha_k y_k + \beta \geq 0 \end{array}$$

That is, we need to find values to the $\alpha_j$, $\beta$, and $\lambda_i$ such that:

- $\alpha_j = \sum_{i=1}^{e} \lambda_i a_{ij}$ for $1 \leq j \leq k$
- $\beta \geq \sum_{i=1}^{e} \lambda_i b_i$
- $\lambda_i \geq 0$, except if the $i$-th linear constraint is an equality

# Using Farkas' Lemma (example)

Consider implied constraints of the template $\alpha c + \beta \geq 0$ and the GROUP DFA:

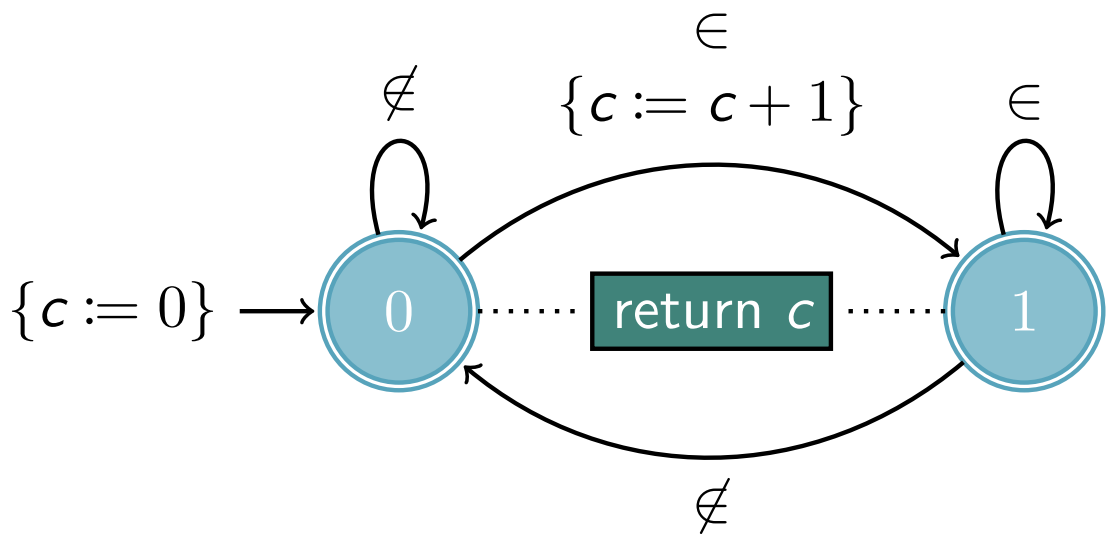


# Using Farkas' Lemma (example)

Initialisation:

| $\lambda_0$ | $c \qquad = 0$ |
|---|---|
| | $\alpha c + \beta \geq 0$ |

$$\lambda_0 = \alpha$$
$$\beta \geq 0$$



# Using Farkas' Lemma (example)

Transition $0 \to 1$:

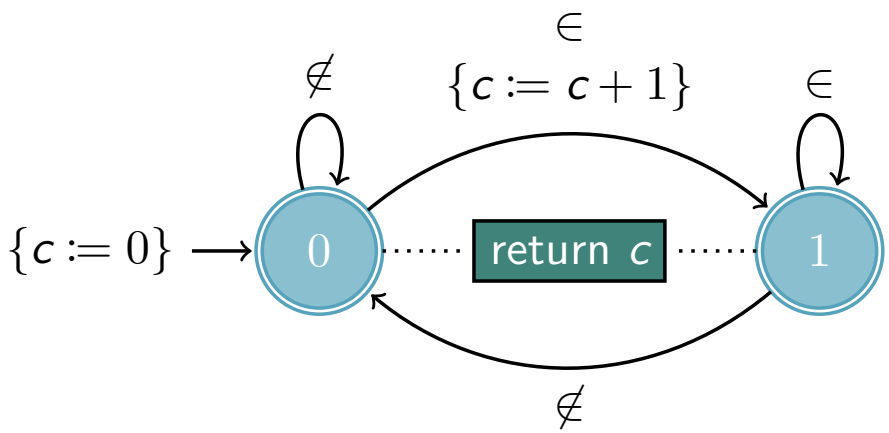
$$\frac{\lambda_1 \mid \alpha c + \beta \geq 0}{\mid \alpha(c+1) + \beta \geq 0}$$

All the other transitions:

$$\frac{\lambda_2 \mid \alpha c + \beta \geq 0}{\mid \alpha c + \beta \geq 0}$$

subject to $\lambda_1 \geq 0$ and $\lambda_2 \geq 0$.

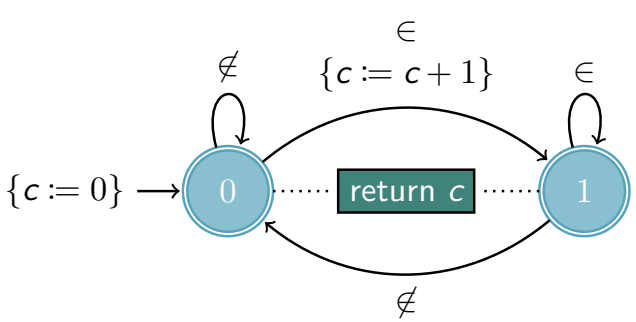
For example, the solution $\alpha = 1 \wedge \beta = 0$, corresponds to the implied constraint $c \geq 0$.

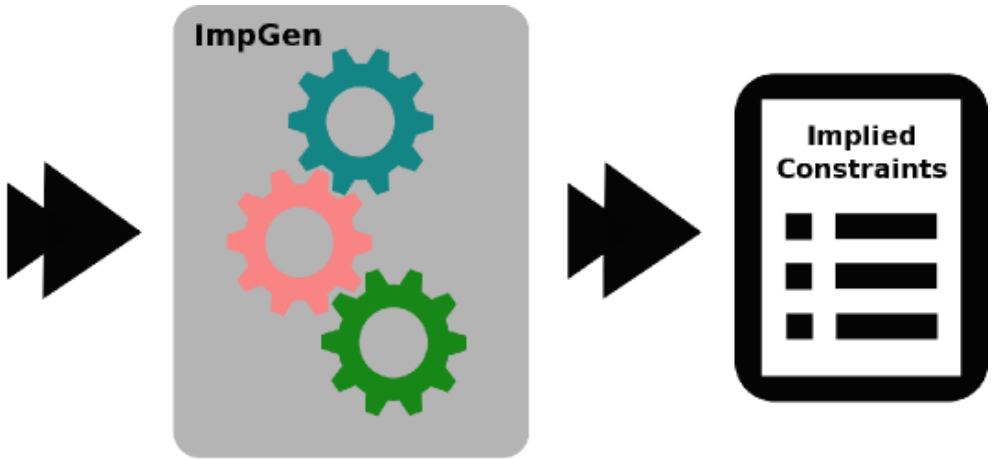



The implied constraints can then be added to the model in order to improve propagation.

# Our Work

We developed a tool based on Farkas' Lemma.

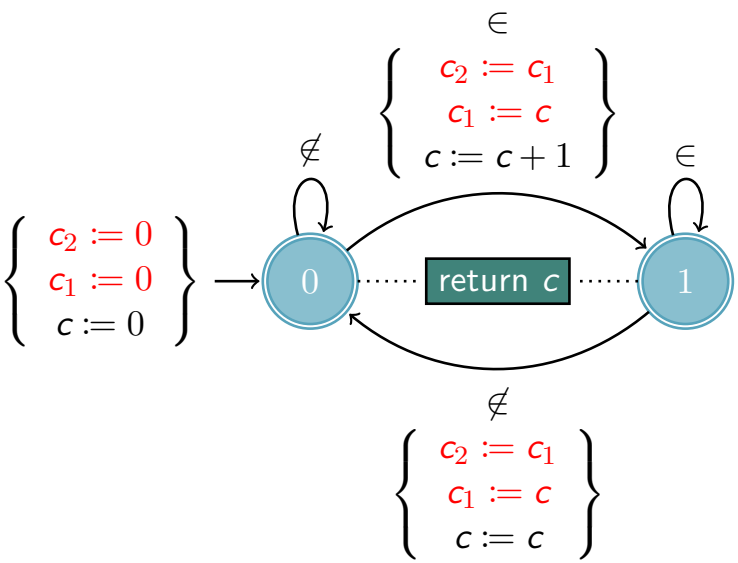



Options

# Option 1: History Accumulators

The tool can add $h$ accumulators $c_i$ to the DFA in order to store the previous $h$ values of each accumulator $c$.




Consider $h = 2$.

The template now is
$\alpha c + \alpha_1 c_1 + \alpha_2 c_2 + \beta \geq 0$.

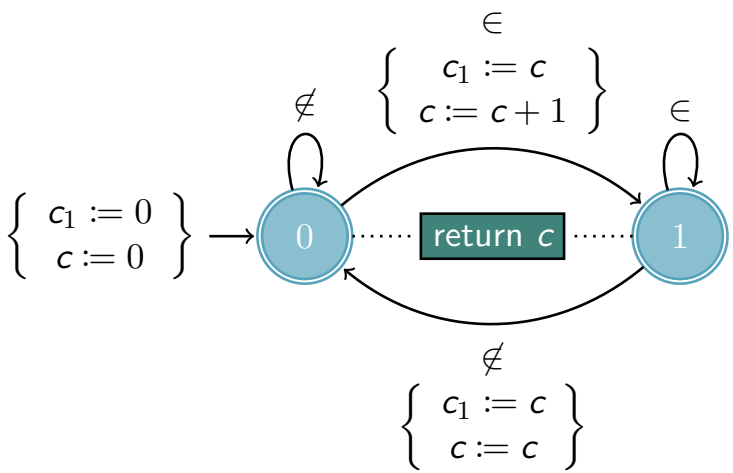
For example, we now find the implied constraints $c_1 \leq c$ and $c \leq c_2 + 1$ (the latter also requires Options 2 and 3).

# Option 2: State Variable

The tool can add a term $\rho q$ for the current state $q$ to the template $\alpha_1 y_1 + ... + \alpha_k y_k + \rho q + \beta \geq 0$.

For example, for a history of length $h \geq 1$, the tool now finds the implied constraint:

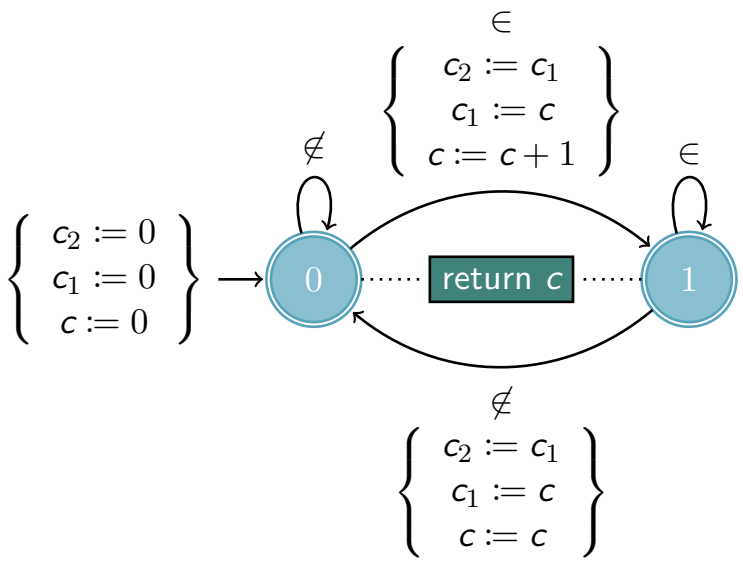
$$c - c_1 \leq q$$



# Option 3: Number of linear systems (or turns)

The tool cannot generate the previously mentioned implied constraint $c \leq c_2 + 1$ after solving the first linear system.

Generating the implied constraint $c \leq c_2 + 1$ requires the prior implied constraints $c - c_1 \leq q$ and $c_2 \leq c_1$. Inferring the latter also requires the prior implied constraint $c_1 \leq c$.



# Experiments

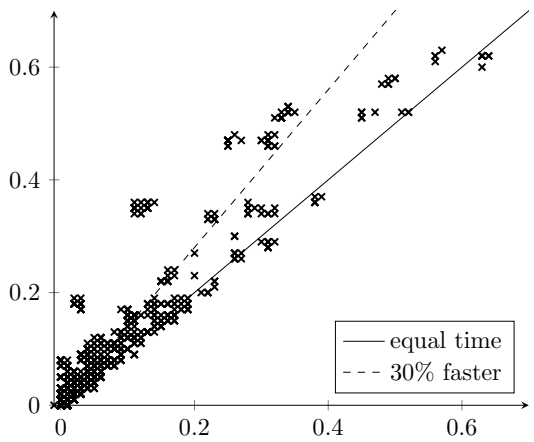
- Constraints in isolation:
  - Randomly generated instances.
  - Instances are divided into satisfiable and unsatisfiable ones.

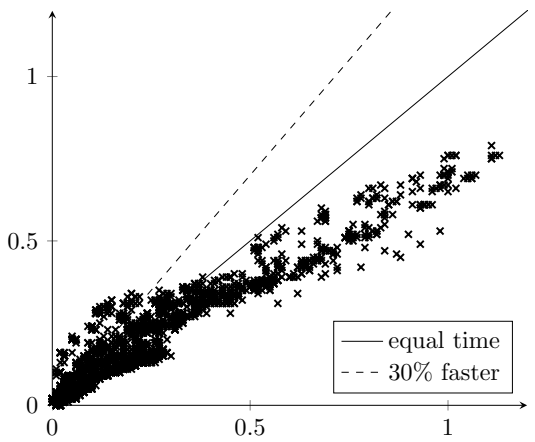
- Entire constraint problems:
  - Randomly generated instances.
  - Designed to be hard problem instances.
  - Half of the constraints in the problem instance are all of the same kind, and it corresponds to the constraint being tested.

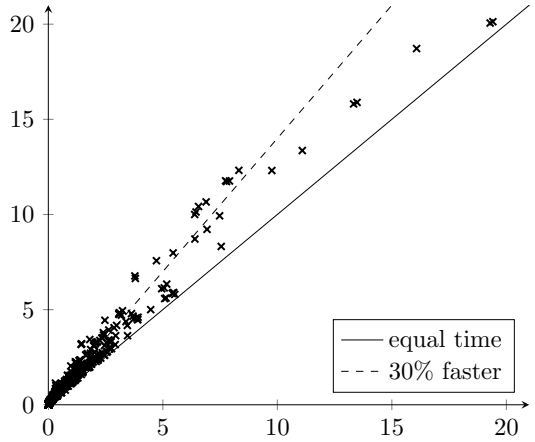
All experiments were run in SICStus Prolog 4.2.

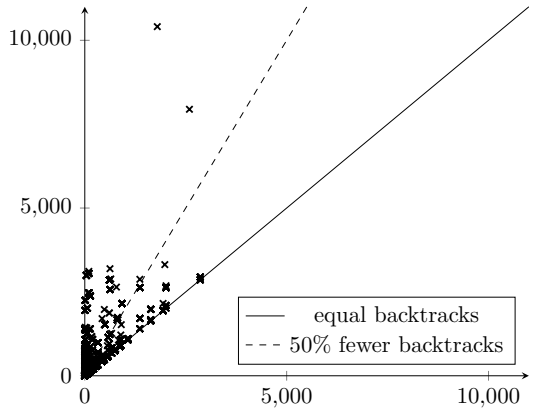
# Constraints in Isolation (satisfiable)

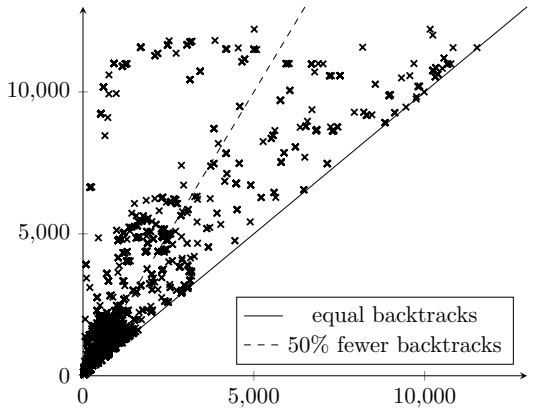
Seconds (top) and backtracks (bottom) to find all solutions to satisfiable instances of Group, FullGroupNval, and Inflexion:

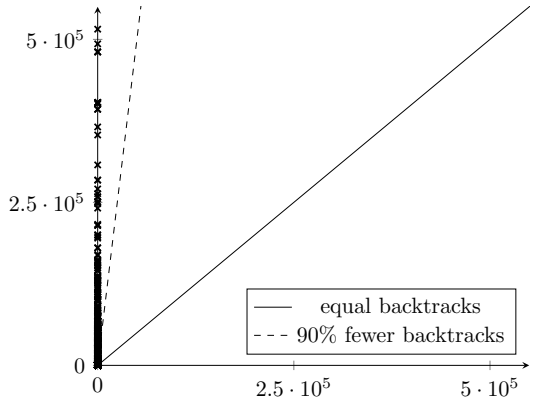


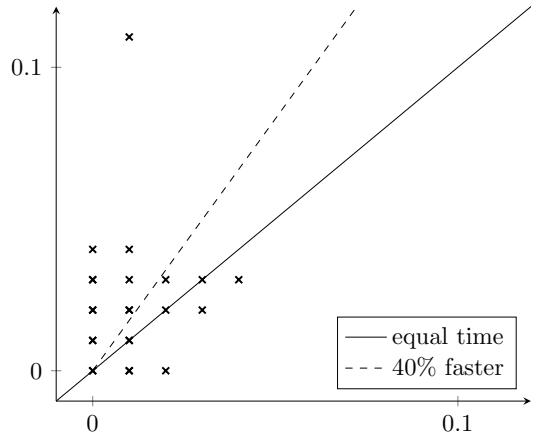



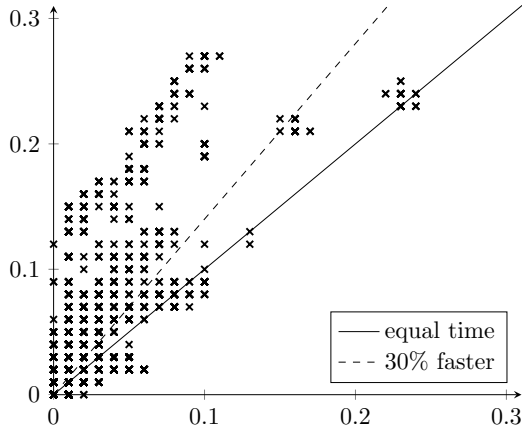



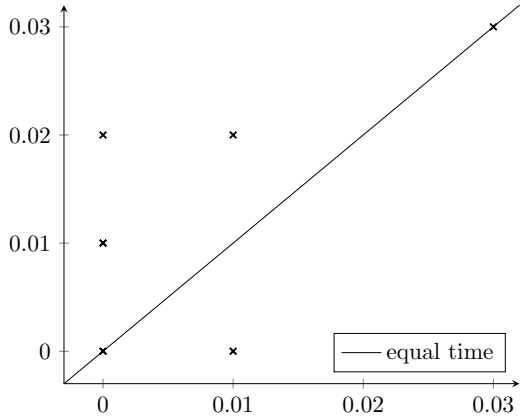



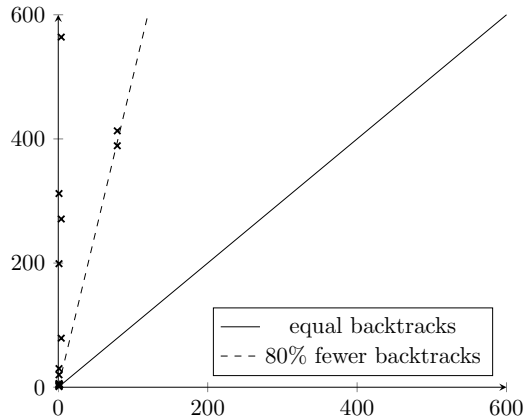
# Constraints in Isolation (unsatisfiable)

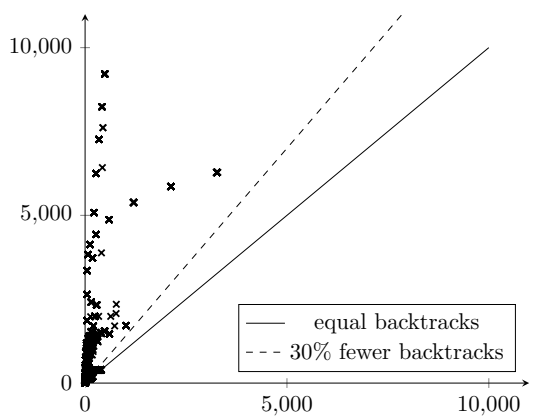
Seconds (top) and backtracks (bottom) to show unsatisfiability on instances of Group, FullGroupNval, and Inflexion:

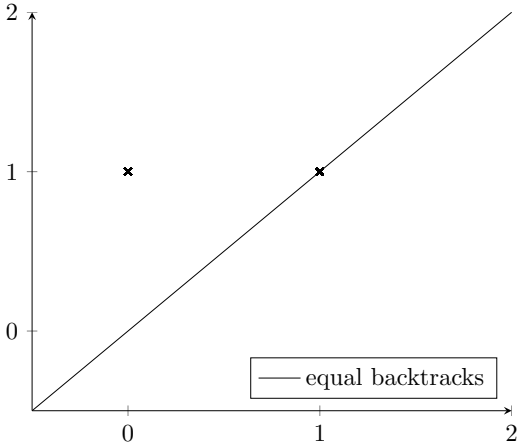


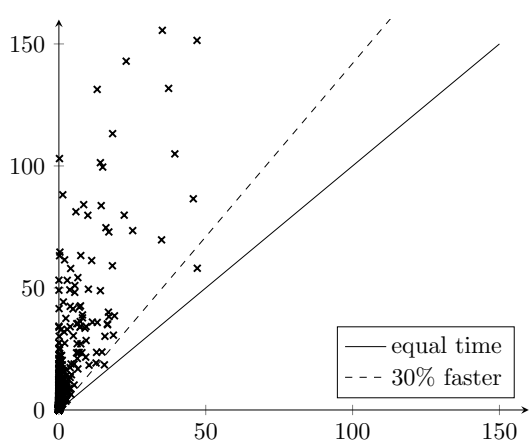



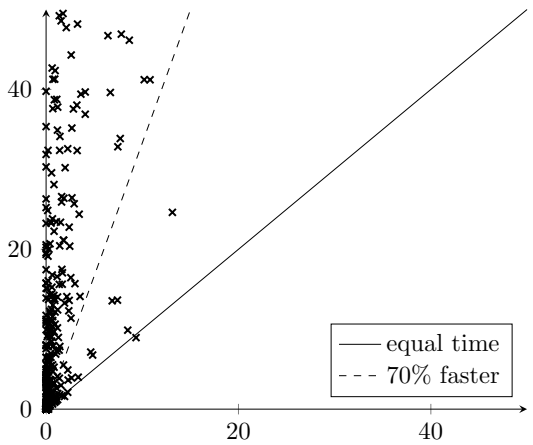



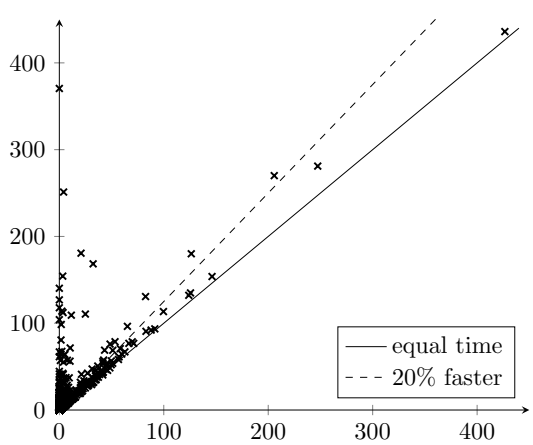



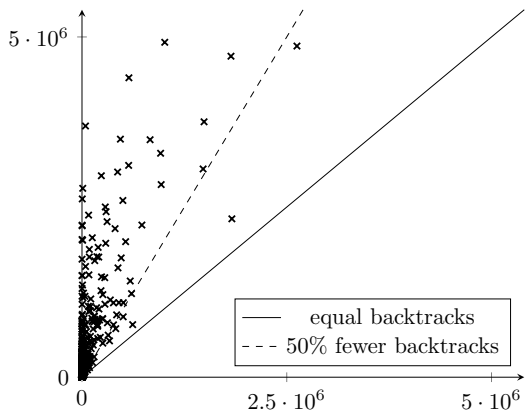
# Entire Constraint Problems

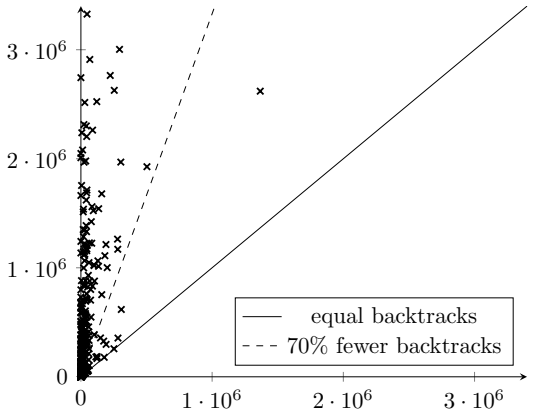
Seconds (top) and backtracks (bottom) to maximise a sum in problem instances involving Group, FullGroupNval, and Inflexion:

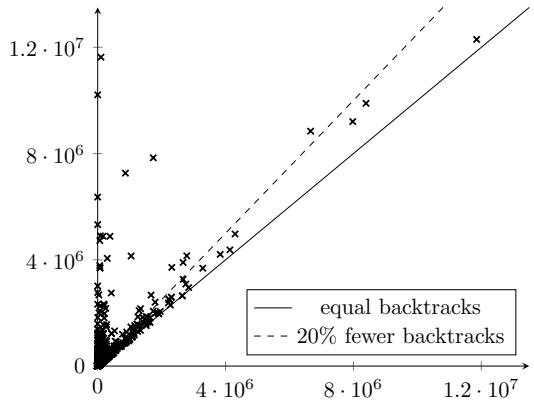


# Conclusion

- We developed a fully automated parametric tool that suggests, in an offline process, a set of linear constraints that are implied by an automaton with accumulators.
- We showed that a suitable choice, by the user, among the suggested implied constraints can considerably improve solving time.
- The generated implied constraints have also been very successfully used in the context of integer programming [Ara15].

# Related Work

- In [BCF+14], we generate implied constraints for automata where the returned value is the same whether the automaton consumes the sequence of variables or its reverse.
  - Like here, the implied constraints are on the accumulator variables and state variables
  - Unlike here, the generation is limited to the indicated particular case and is manual in most sub-cases.

- In [BCRT05], graph invariants are used to generate implied constraints.
  - Our approach does not require a database of precomputed invariants.

# Future Work

- Use a richer template for implied constraints:
  - Disjunction;
    a motivating example is in [FFP13].
  - Non-linear templates, for instance, multiplication of accumulators;
    a motivating example is in [BCRT05].

- Add other options to the tool:
  - Adding a term $\rho i$ for the index variable $i$ to the template $\alpha_1 y_1 + \cdots + \alpha_k y_k + \rho i + \beta \geq 0$.


Ekaterina Arafailova.
Reformulation of automata for time series constraints as linear programs.
Master's thesis, Mines Nantes, France, 2015.


Nicolas Beldiceanu, Mats Carlsson, Pierre Flener, María Andreína Francisco Rodríguez, and Justin Pearson.
Linking prefixes and suffixes for constraints encoded using automata with accumulators.
In Barry O'Sullivan, editor, *CP 2014*, volume 8656 of *LNCS*, pages 142–157. Springer, 2014.


Nicolas Beldiceanu, Mats Carlsson, and Thierry Petit.
Deriving filtering algorithms from constraint checkers.
In Mark Wallace, editor, *CP 2004*, volume 3258 of *LNCS*, pages 107–122. Springer, 2004.


Nicolas Beldiceanu, Mats Carlsson, Jean-Xavier Rampon, and Charlotte Truchet.
Graph invariants as necessary conditions for global constraints.

In Peter van Beek, editor, *CP 2005*, volume 3709 of *LNCS*, pages 92–106. Springer, 2005.

María Andreína Francisco, Pierre Flener, and Justin Pearson. Generation of implied constraints for automaton-induced decompositions.
In Alexander Brodsky, Éric Grégoire, and Bertrand Mazure, editors, *ICTAI 2013*, pages 1076–1083. IEEE Computer Society, 2013.