Propagation Algorithms

CP course, lecture 5

Recapitulation

- Propagators: S→S

  (mapping constraint stores to constraint stores)

- Implement constraints
- Must be contracting, monotonic, correct, checking
- Can be idempotent, subsumed
- Can be bounds, domain consistent

Recapitulation

- Global constraints: exploit global view on variables

  $a+b=c$, $c+d=e$ is weaker than $a+b+d=e$

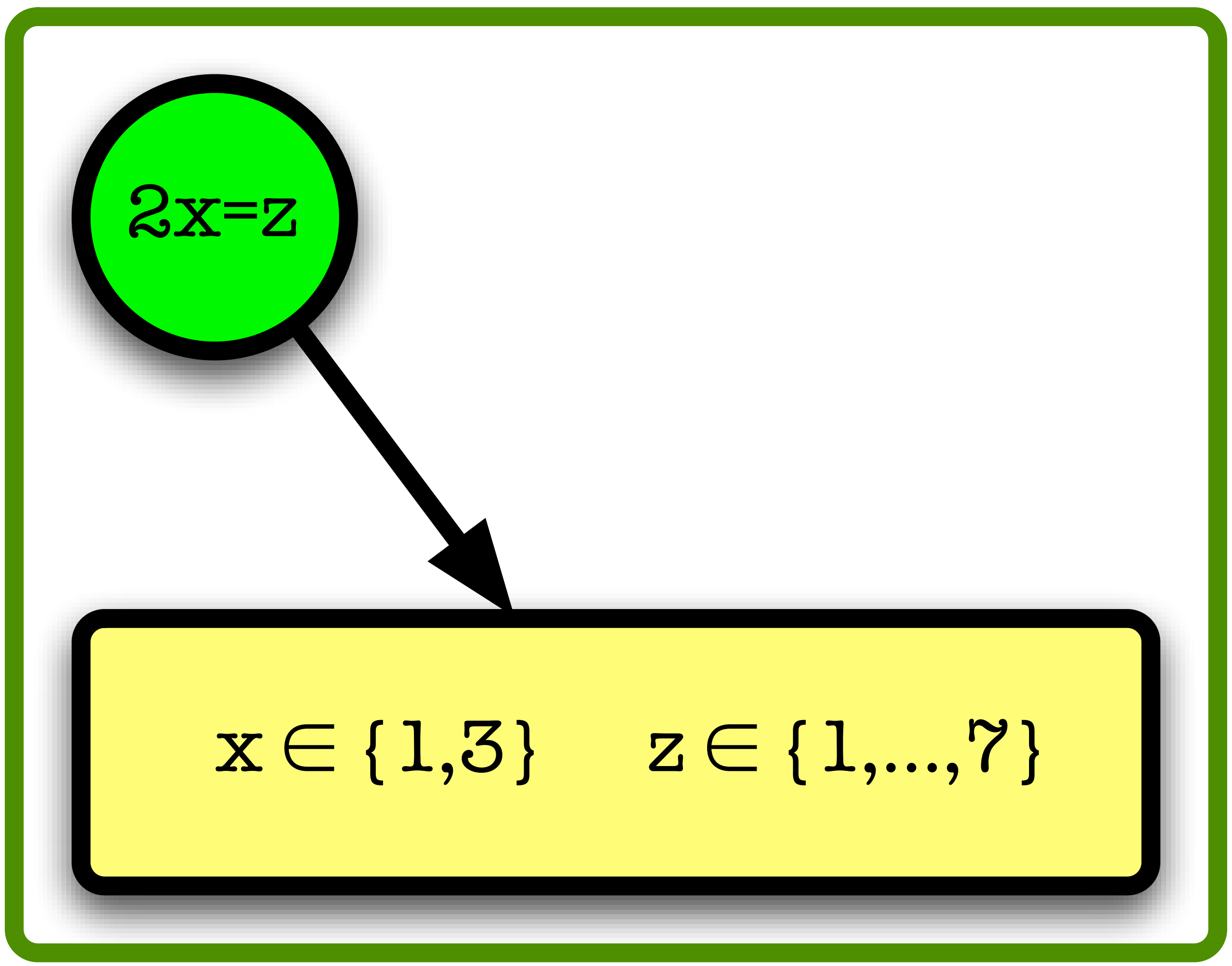
  $x \neq y$, $y \neq z$, $x \neq z$ is weaker than distinct(x,y,z)

Recap: Consistency

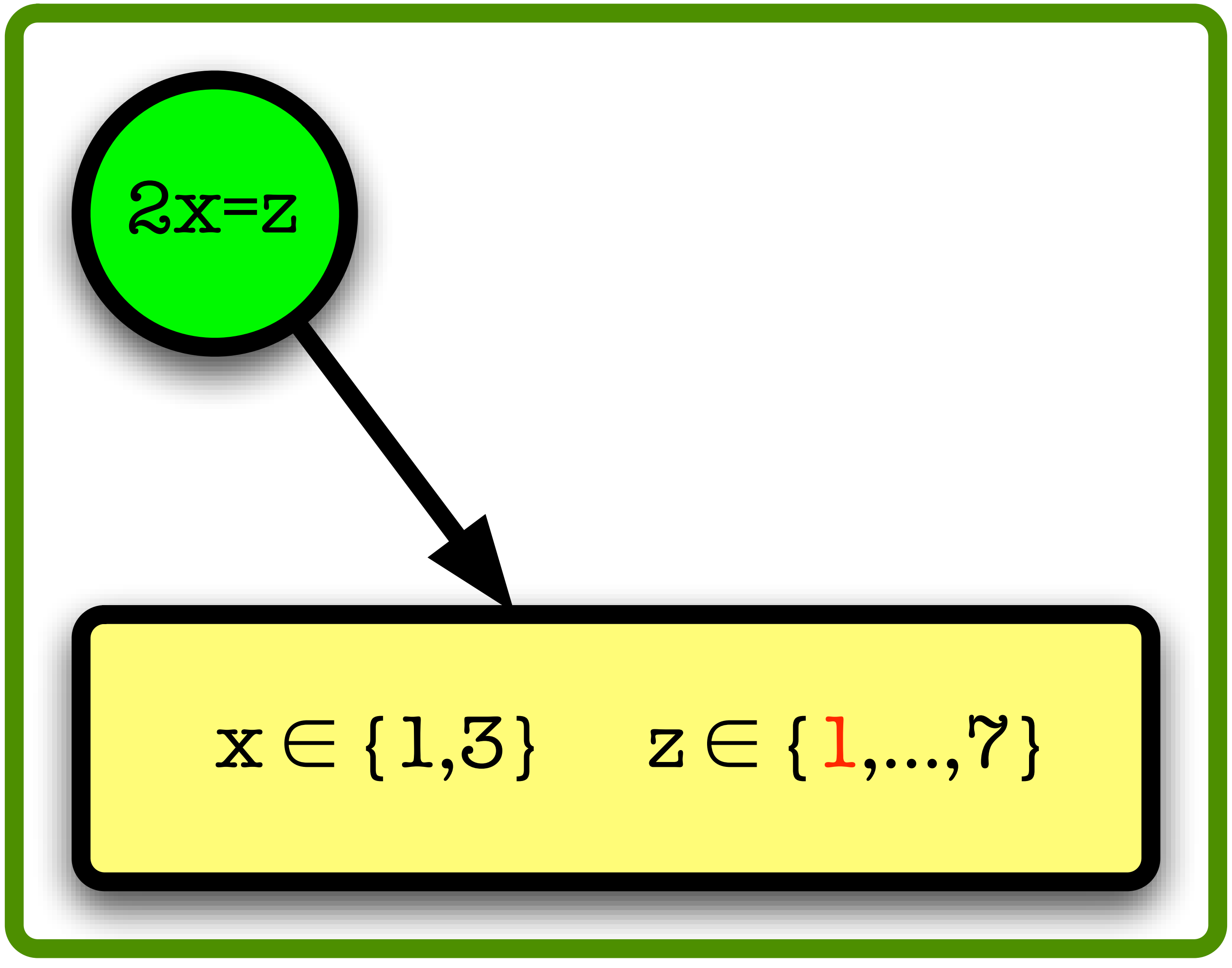
- Consider $2x=z$

  with $x \in \{1,3\}$, $z \in \{1,...,7\}$

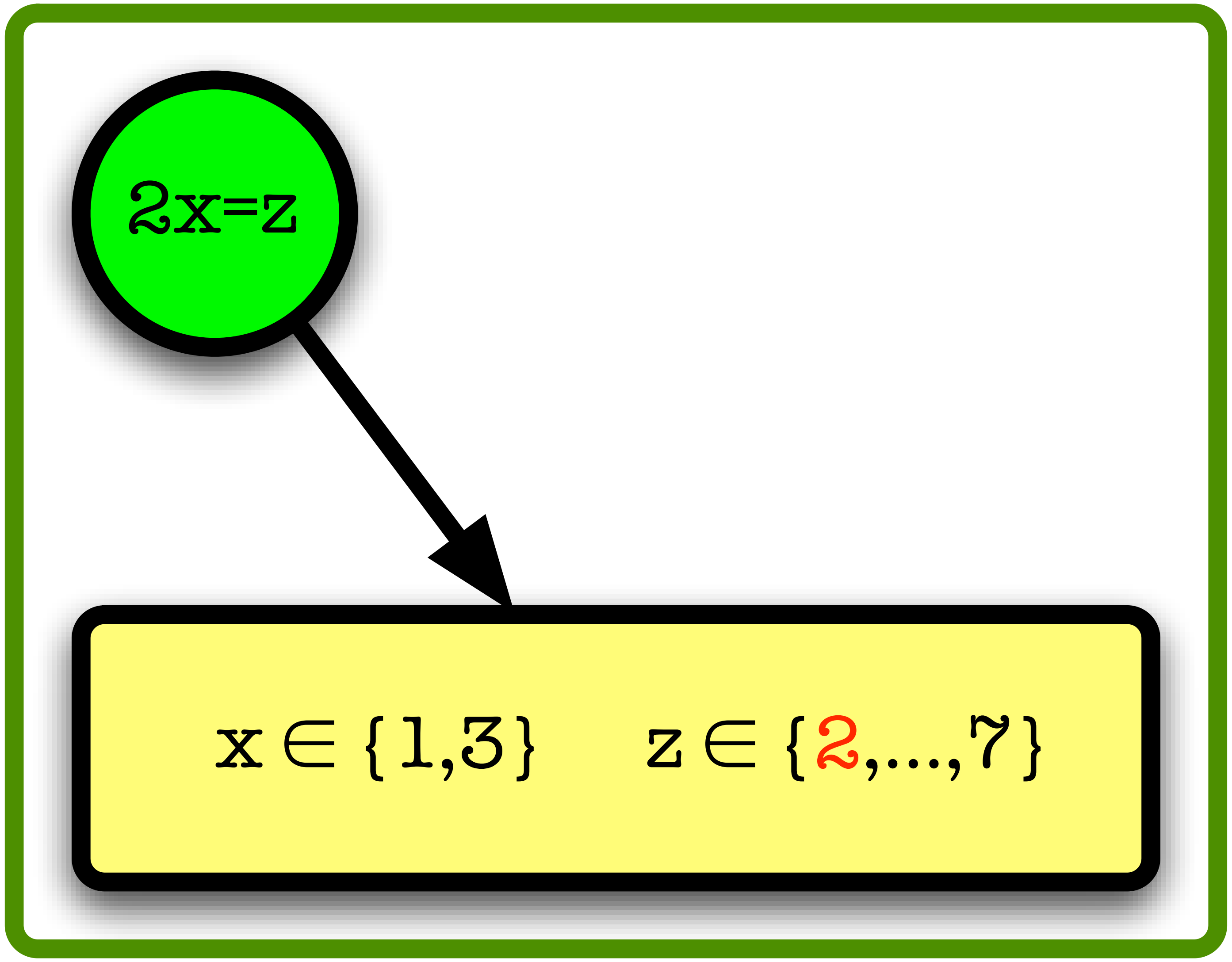
Bounds consistency



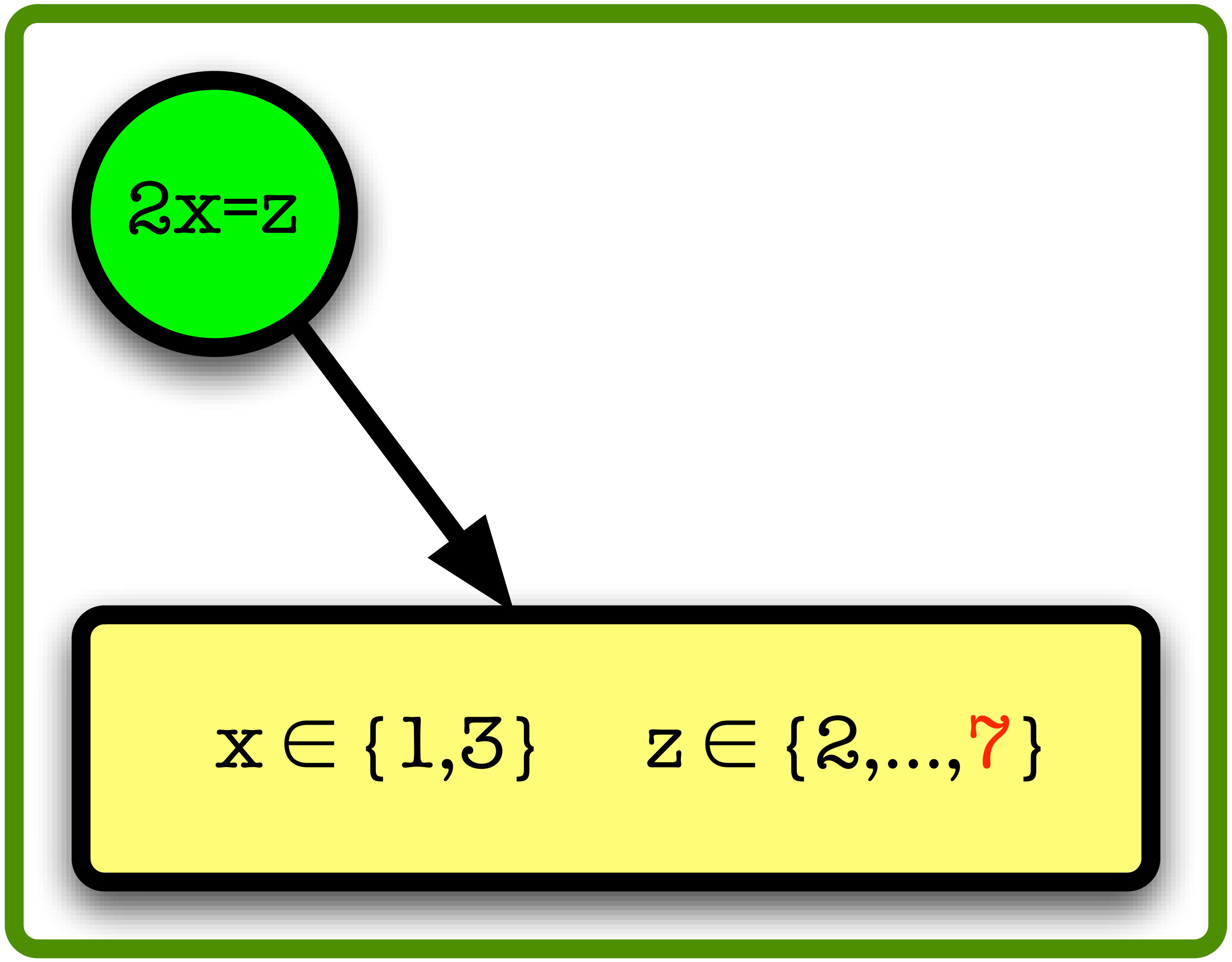
Bounds consistency



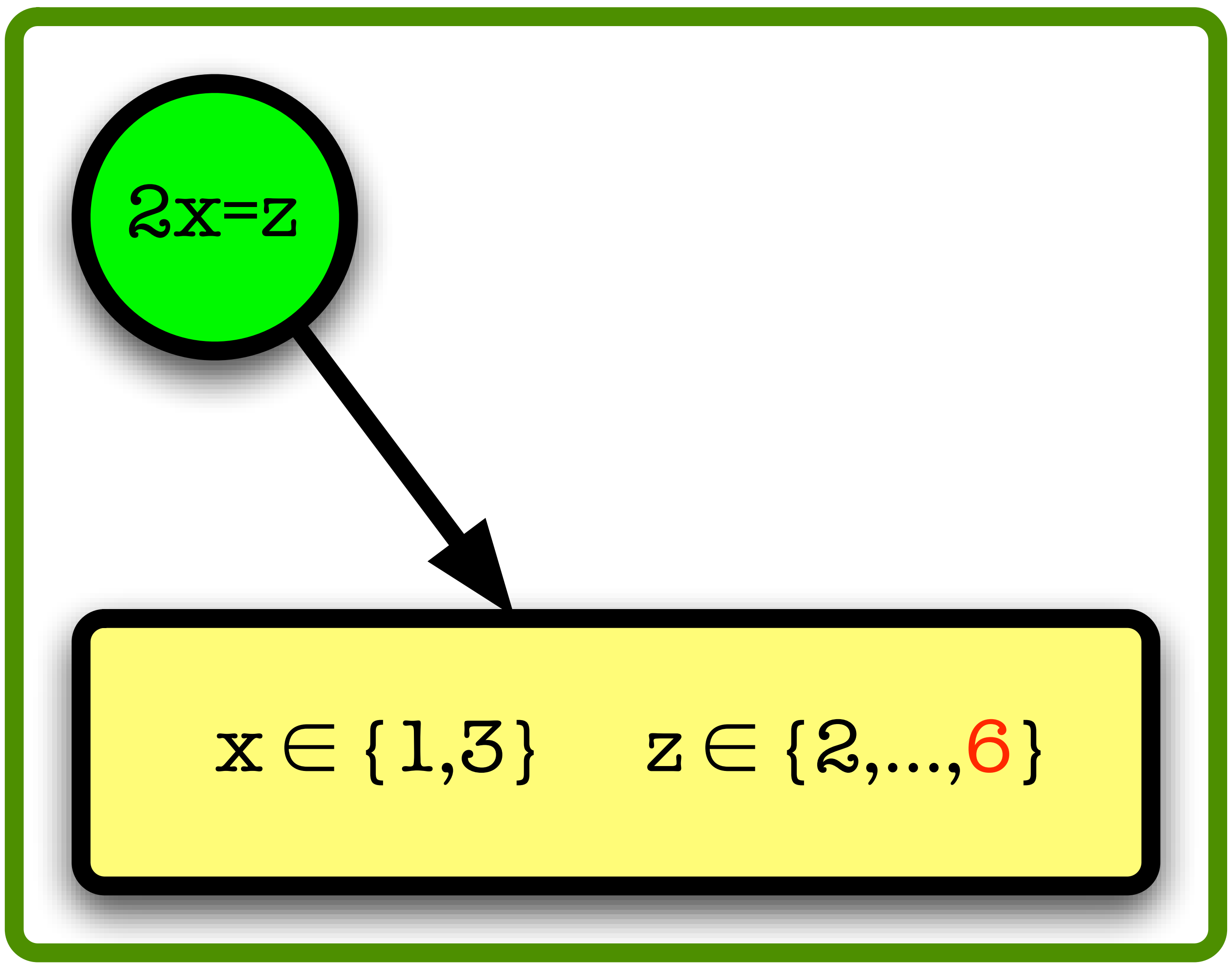
Bounds consistency



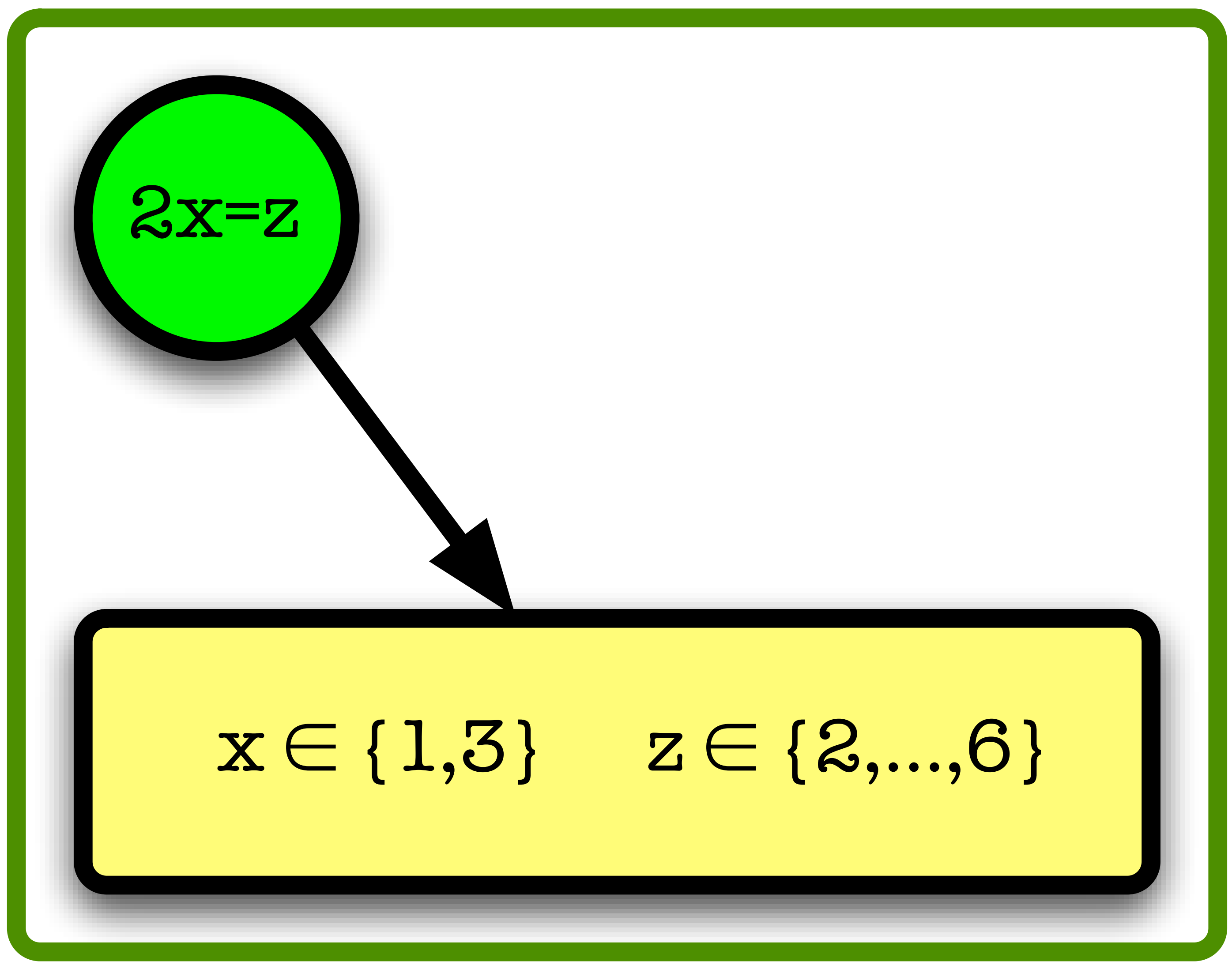
Bounds consistency



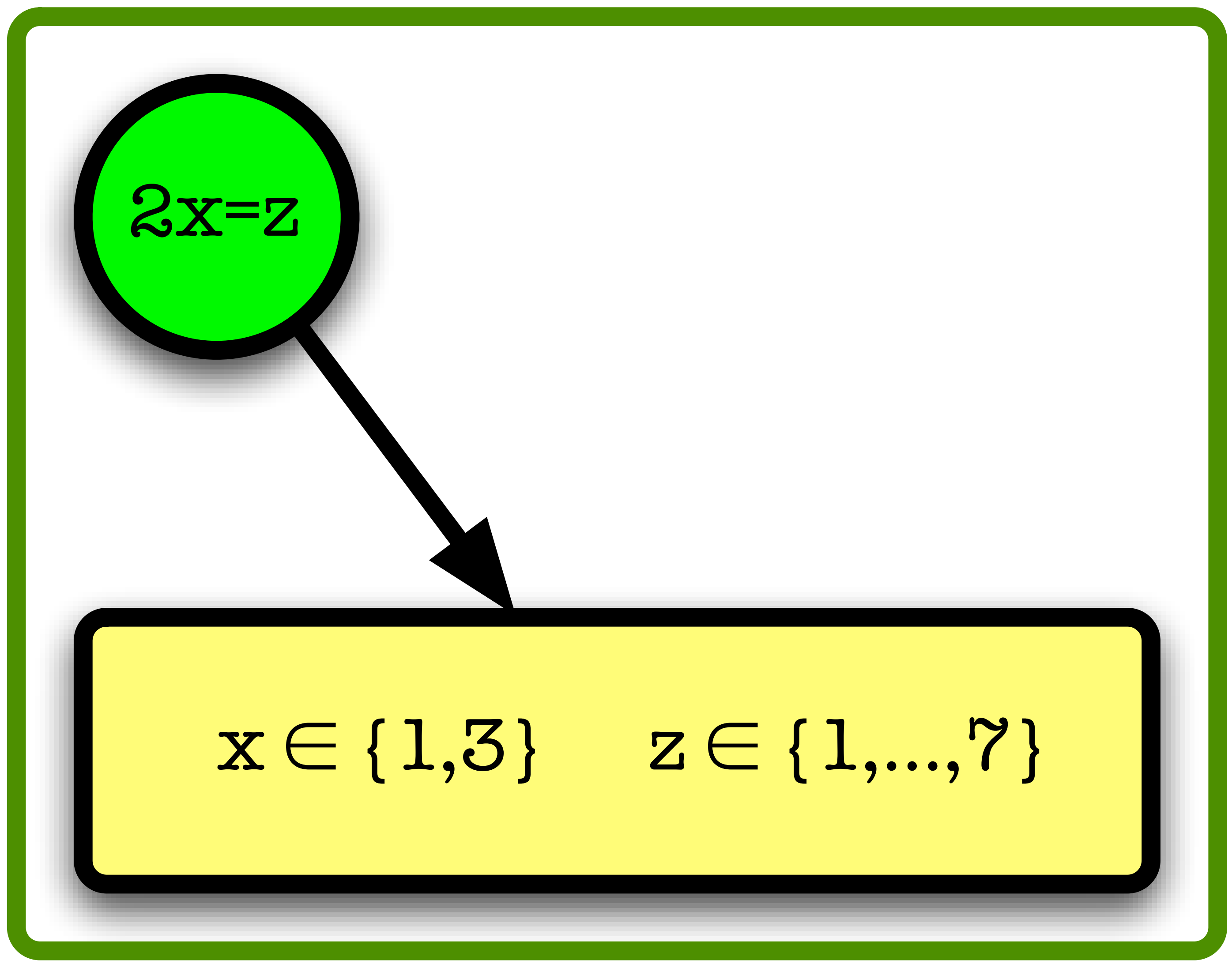
Bounds consistency



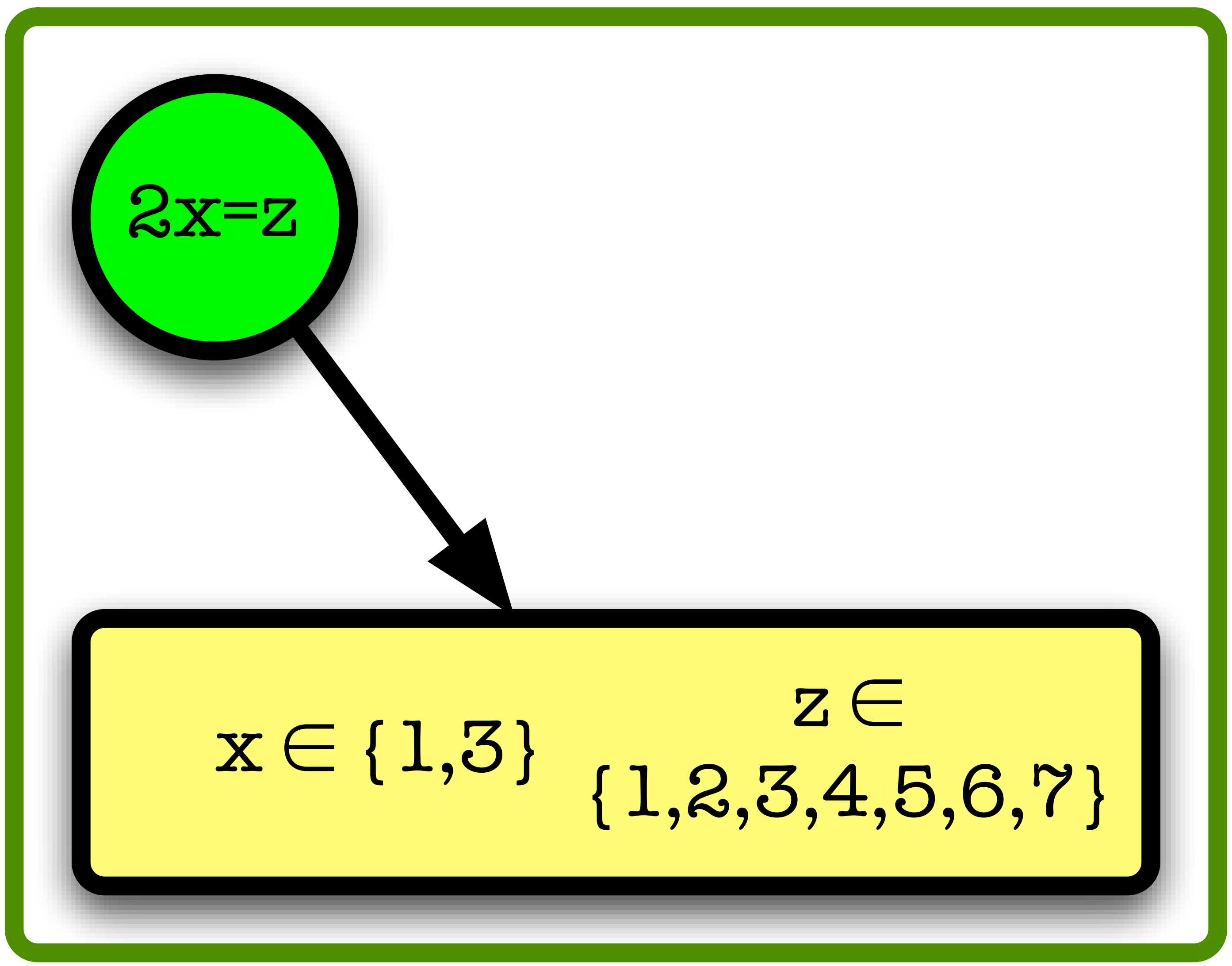
Bounds consistency



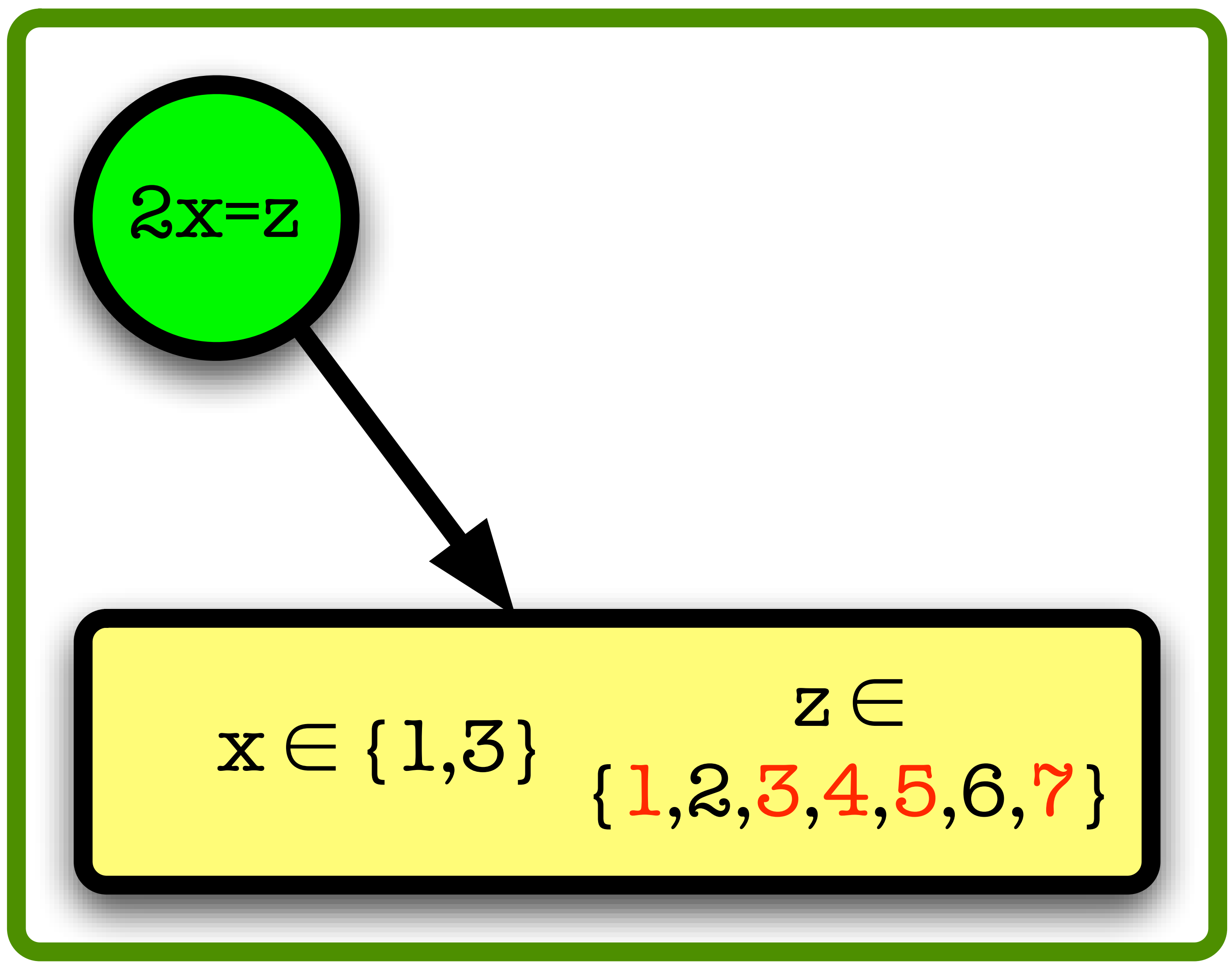
Domain consistency



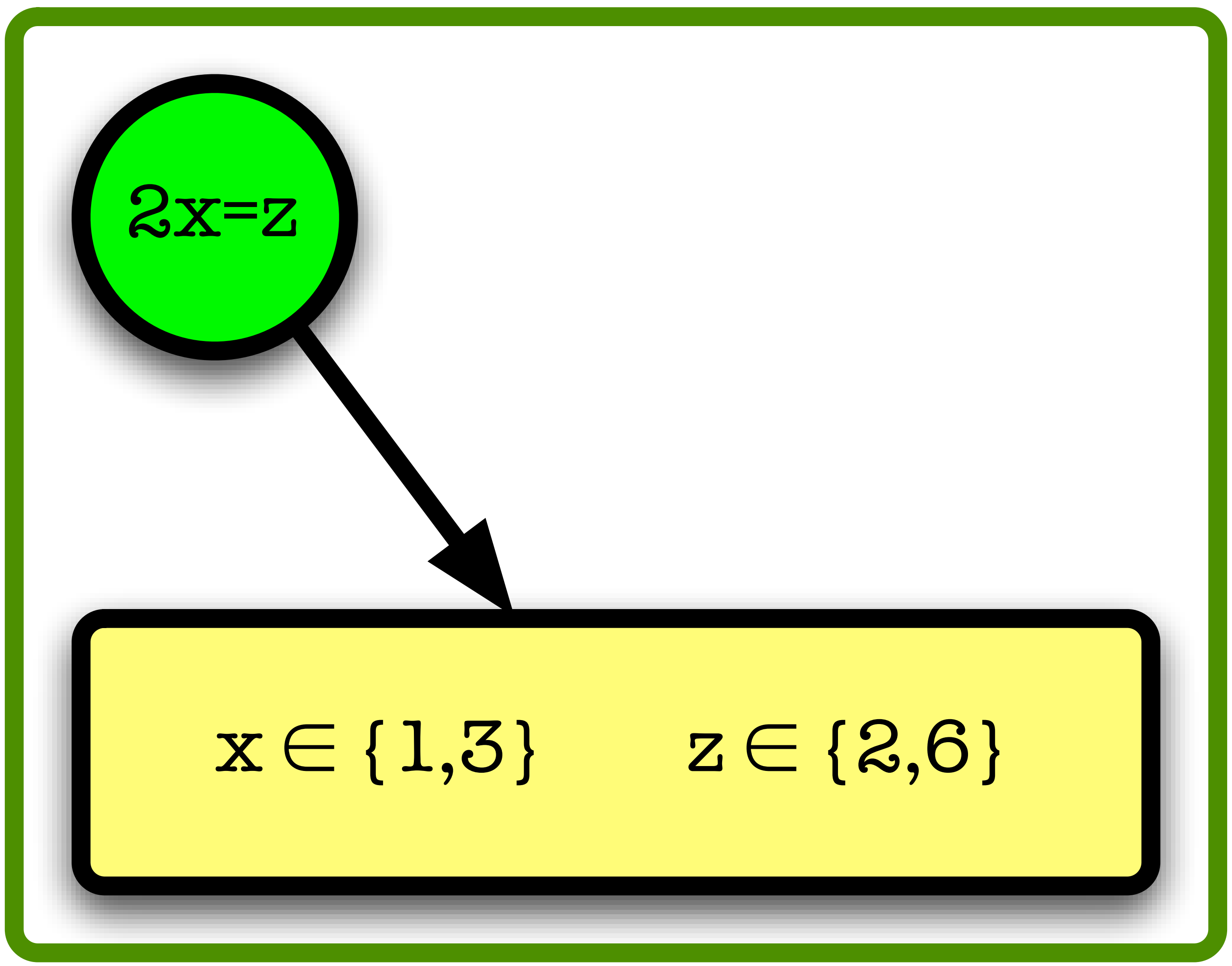
Domain consistency



Domain consistency



Domain consistency



Recap: Consistency

- Consider $2x=z$
  with $x \in \{1,3\}$, $z \in \{1,...,7\}$
- Domain consistency:
  Stronger propagation, more complex algorithms
- Bounds consistency:
  Weaker propagation, simpler algorithms

Linear equations

- Propagator for

$$\sum a_i x_i = c$$

- How can bounds information be propagated efficiently?
- Example:

$$ax + by = c$$

Propagating bounds

- Rewrite:

$ax + by = c$ $\qquad\qquad$ $ax = c - by$

$x = (c-by)/a$

- Propagate

$x \leq \lfloor \max\{ (c-bn)/a) \mid n \in s(y) \} \rfloor$

$x \geq \lceil \min\{ (c-bn)/a \mid n \in s(y) \} \rceil$

Propagating bounds

- $m = \max\{ (c-bn)/a) \mid n \in s(y) \}$
- $a > 0$:

  $m = \max\{ (c-bn) \mid n \in s(y) \} / a$
- $a < 0$:

  $m = \min\{ (c-bn) \mid n \in s(y) \} / a$

Propagating bounds

- For $a>0$:

  $m = \max\{ (c-bn) \mid n \in s(y) \} / a$

  $= (c - \min \{bn \mid n \in s(y)\}) / a$

  - For $b>0$:

    $m = (c - b \cdot \min s(y)) / a$

  - For $b<0$:

    $m = (c - b \cdot \max s(y)) / a$

General Case

- Repeat until fixpoint, for each variable x_i
- Improvement: Compute

$$u = \max\left\{ d - \sum_{i=1}^{n} a_i n_i \mid n_i \in s(x_i) \right\}$$

$$l = \min\left\{ d - \sum_{i=1}^{n} a_i n_i \mid n_i \in s(x_i) \right\}$$

- Reuse by removing term for x_i in each iteration

Questions

- Is it necessary to iterate?

  Yes, otherwise not idempotent

- What level of consistency does the propagator achieve?

Consistency

- This propagator is not bounds consistent:

  $x = 3y + 5z$ with

  $x \in \{2,\ldots,7\}$, $y \in \{0,1,2\}$, $z \in \{-1,0,1,2\}$

- Propagator will compute

  $x \in \{2,\ldots,7\}$, $y \in \{0,1,2\}$, $z \in \{0,1\}$

  should be 6

Consistency

- Algorithm considers real-valued solutions:

  $x=7, y=2/3, z=1 \quad \Rightarrow \quad 7=3 \cdot 2/3 + 5 \cdot 1$

- New notion: R-bounds consistency

  (allow solutions over the reals)

- Even bounds consistency cannot be achieved efficiently for some propagators!

Propagator Properties

- A domain consistent propagator is idempotent
- A bounds consistent propagator is idempotent
- Proof: Exercise

All-distinct

- Naive:
  - check that no two determined variables have the same value
  - remove values of determined variables from domains of undetermined variables
- Advantage: simple implementation, avoid $O(n^2)$ propagators
- Disadvantage: not very strong

All-distinct

- Is there an efficient bounds or domain consistent propagator?
- Puget: bounds consistent, O(n log n)

Régin: domain consistent, $O(n^{2.5})$

Régin's algorithm

- Construct a variable-value graph

  bipartite, variable node → value node

- Characterize solutions in the graph

  maximal matchings

- Use matching theory

  one matching describes all matchings

- Remove edges not taking part in any solution

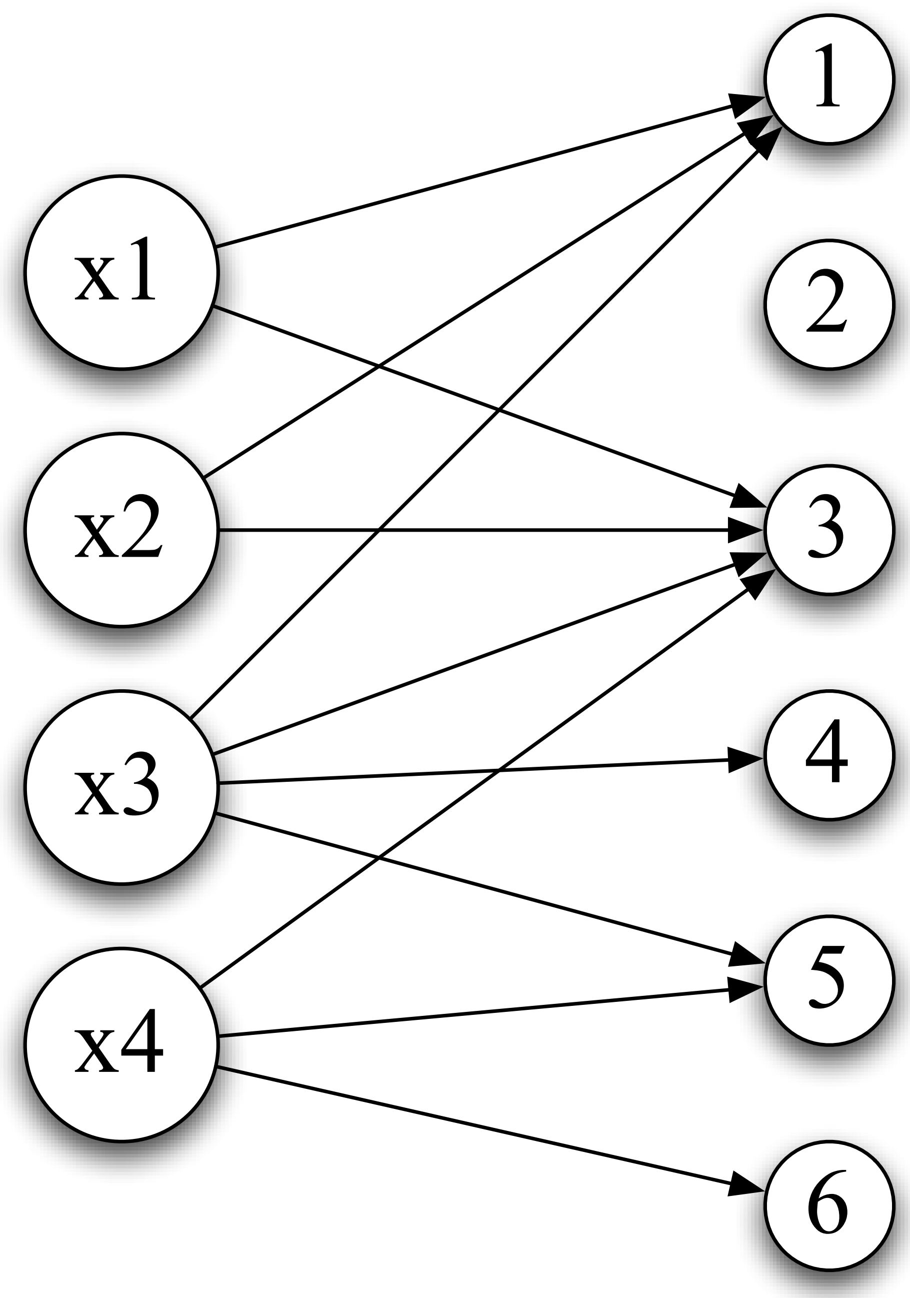
Variable-value Graph

x1∈{1,3}

x2∈{1,3}

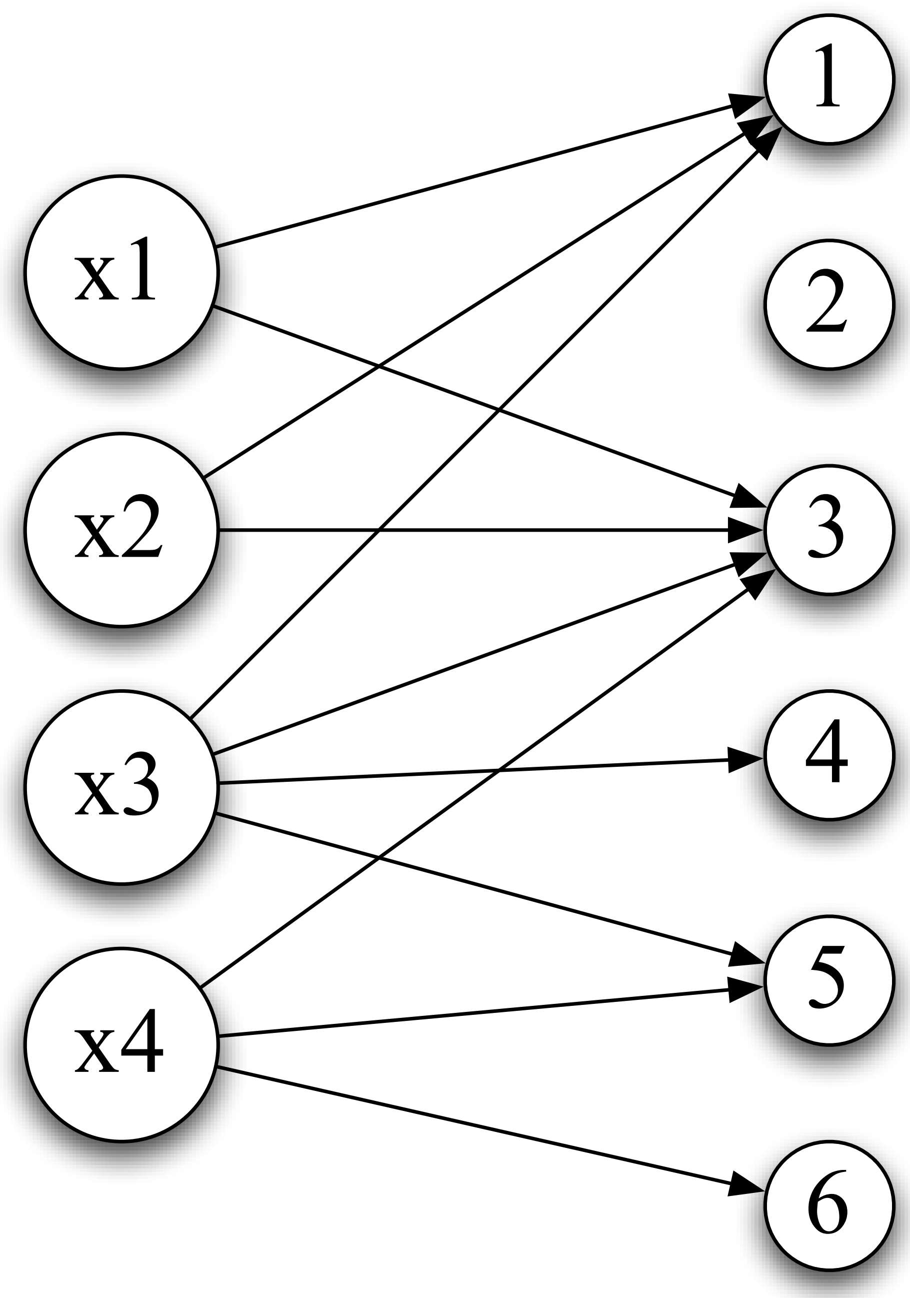
x3∈{1,3,4,5}

x4∈{3,5,6}



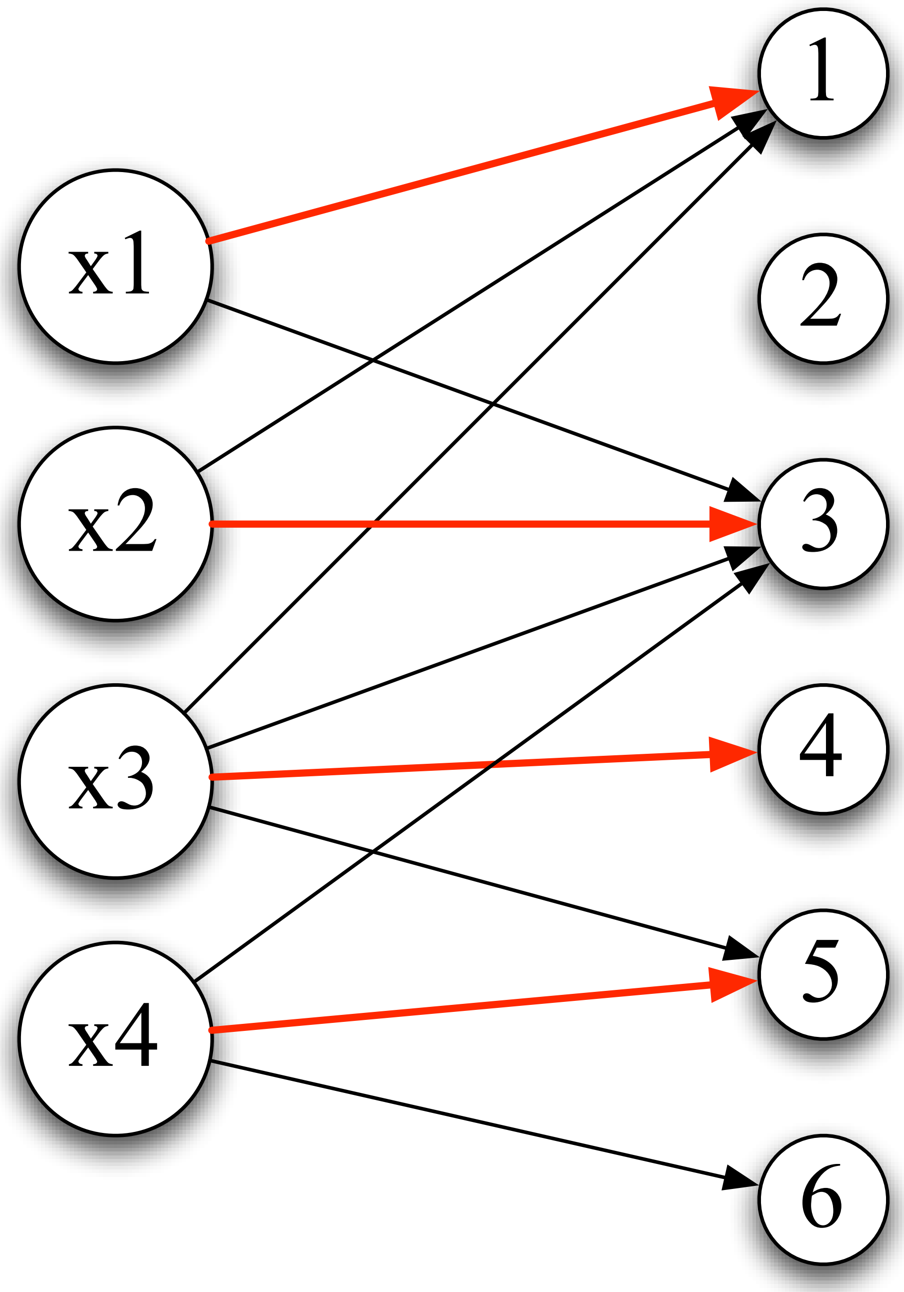
Matching

- subset of edges s.th. no two edges share a vertex
- maximal: maximum cardinality



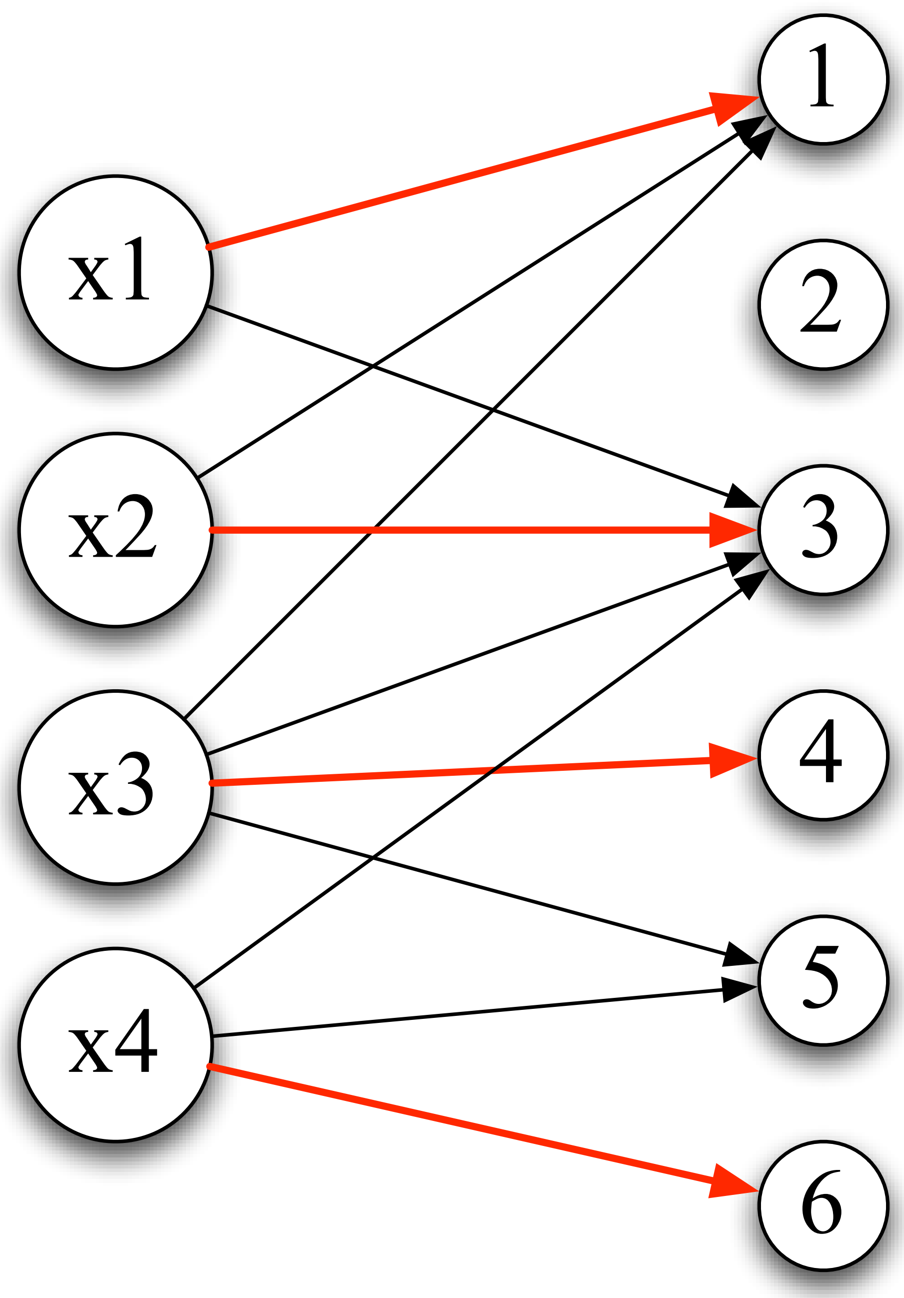
Matching

- subset of edges s.th. no two edges share a vertex
- maximal: maximum cardinality



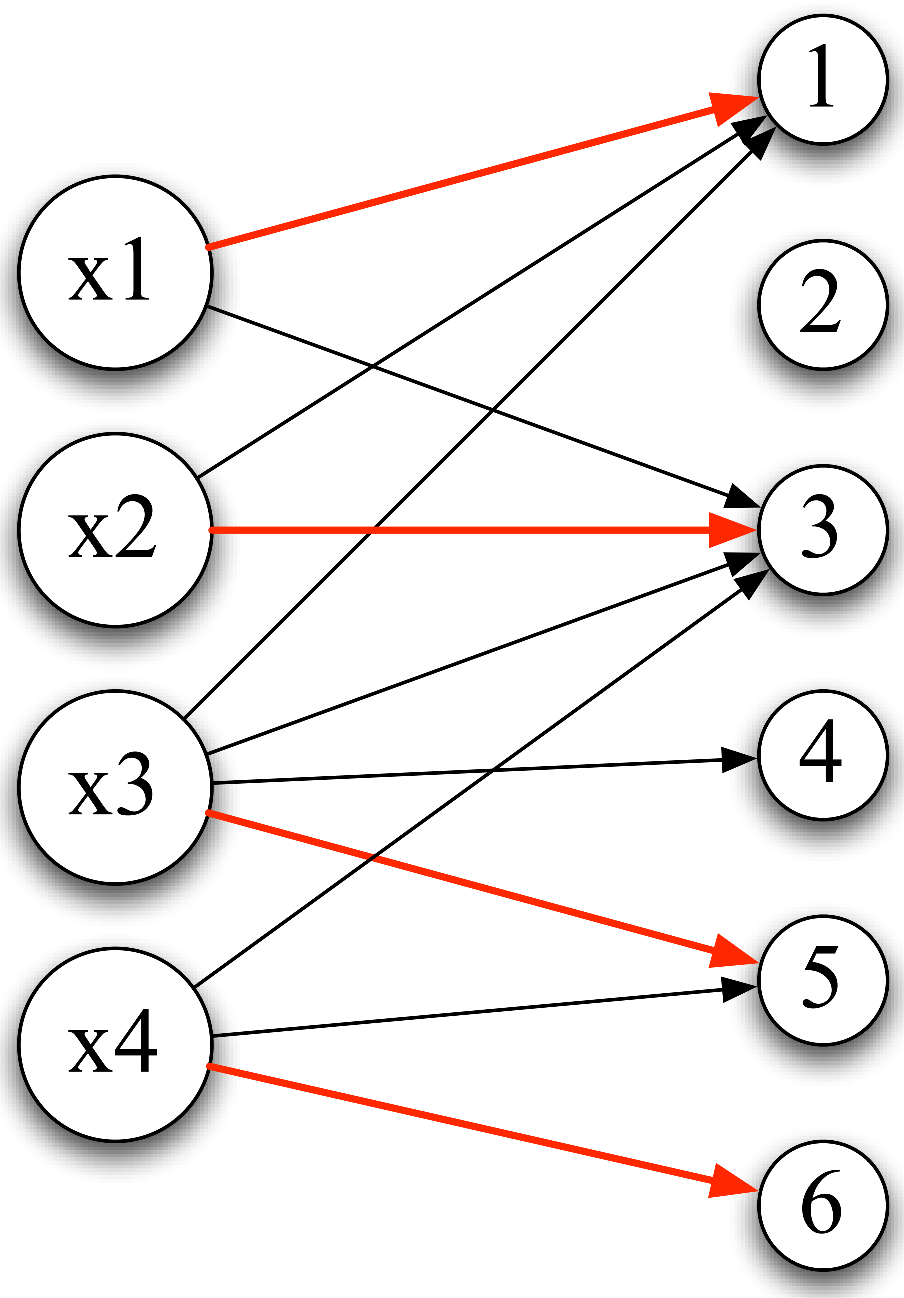
Matching

- subset of edges s.th. no two edges share a vertex
- maximal: maximum cardinality



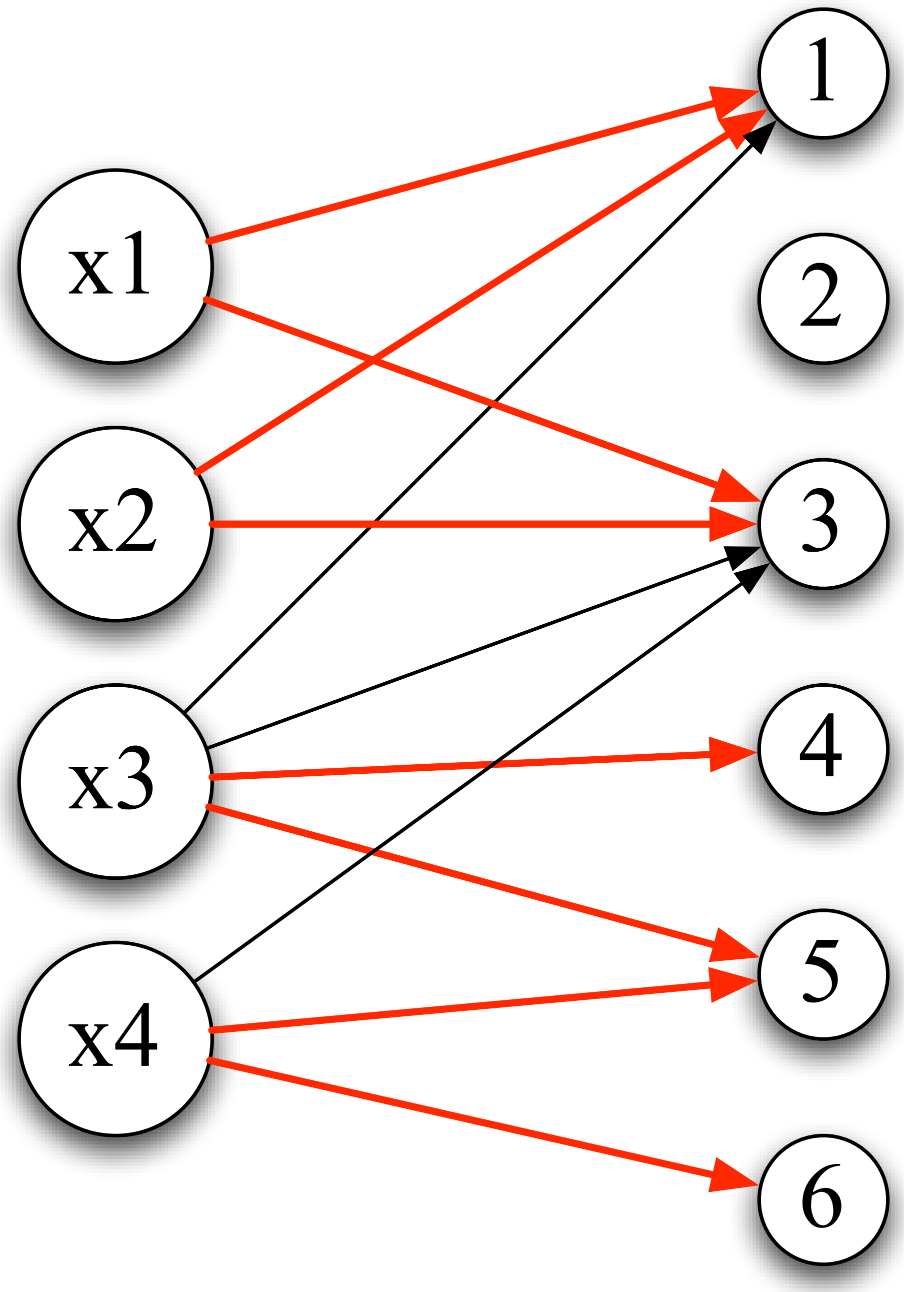
Matching

- subset of edges s.th. no two edges share a vertex
- maximal: maximum cardinality



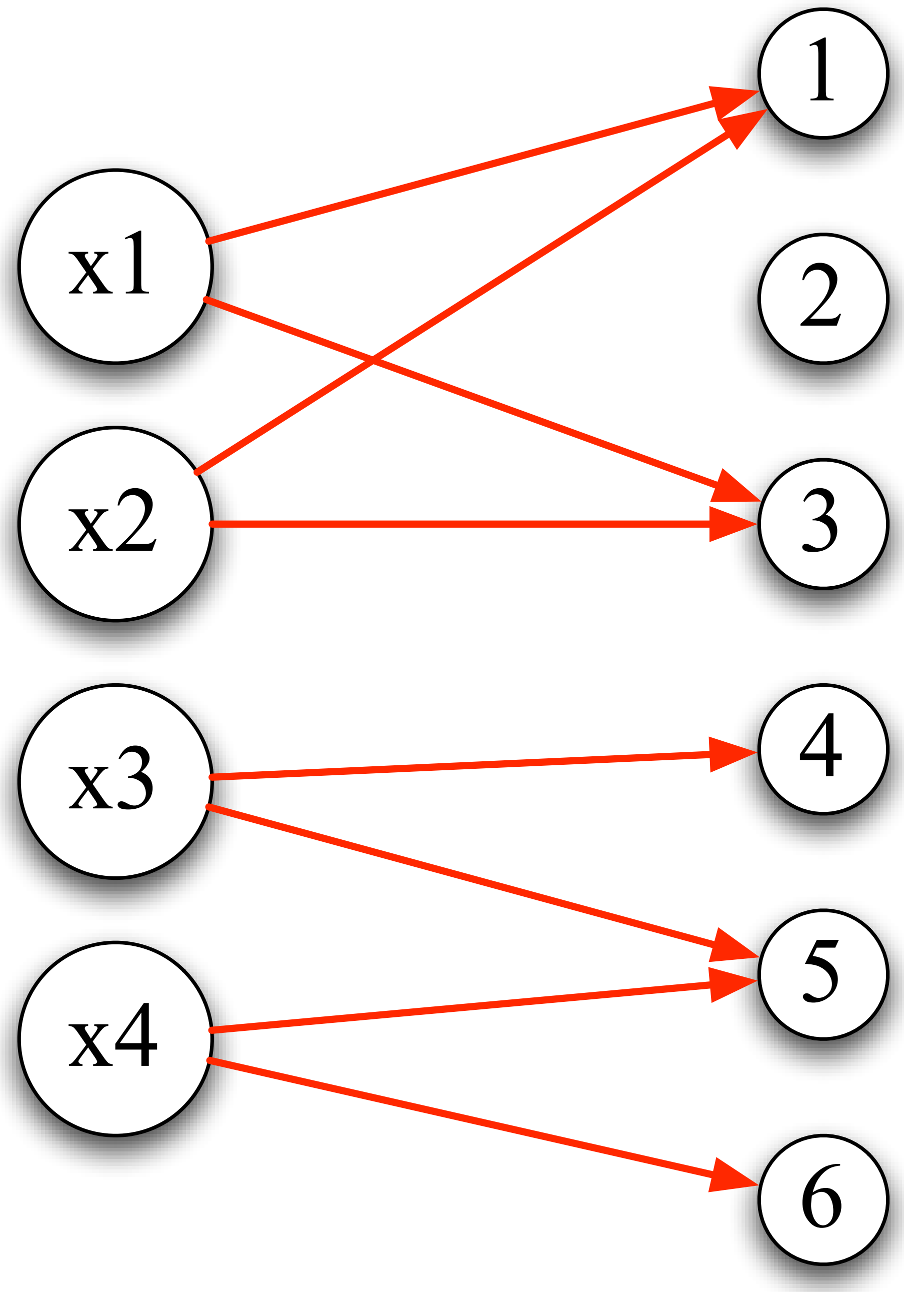
Matching

- Compute union of all maximal matchings



Matching

- Compute union of all maximal matchings
- Delete unmatched edges



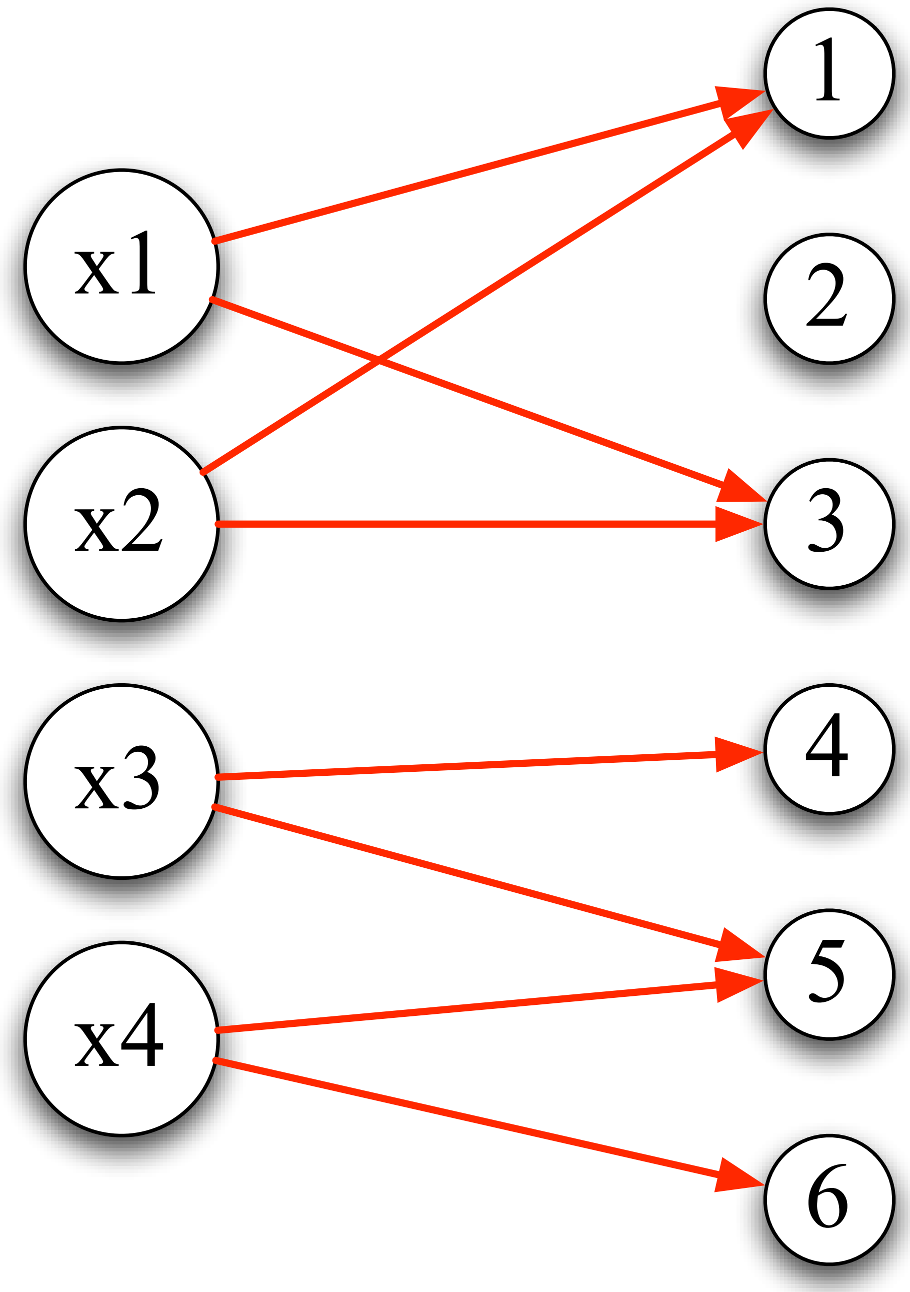
Compute new domains

$x1 \in \{1,3\}$

$x2 \in \{1,3\}$

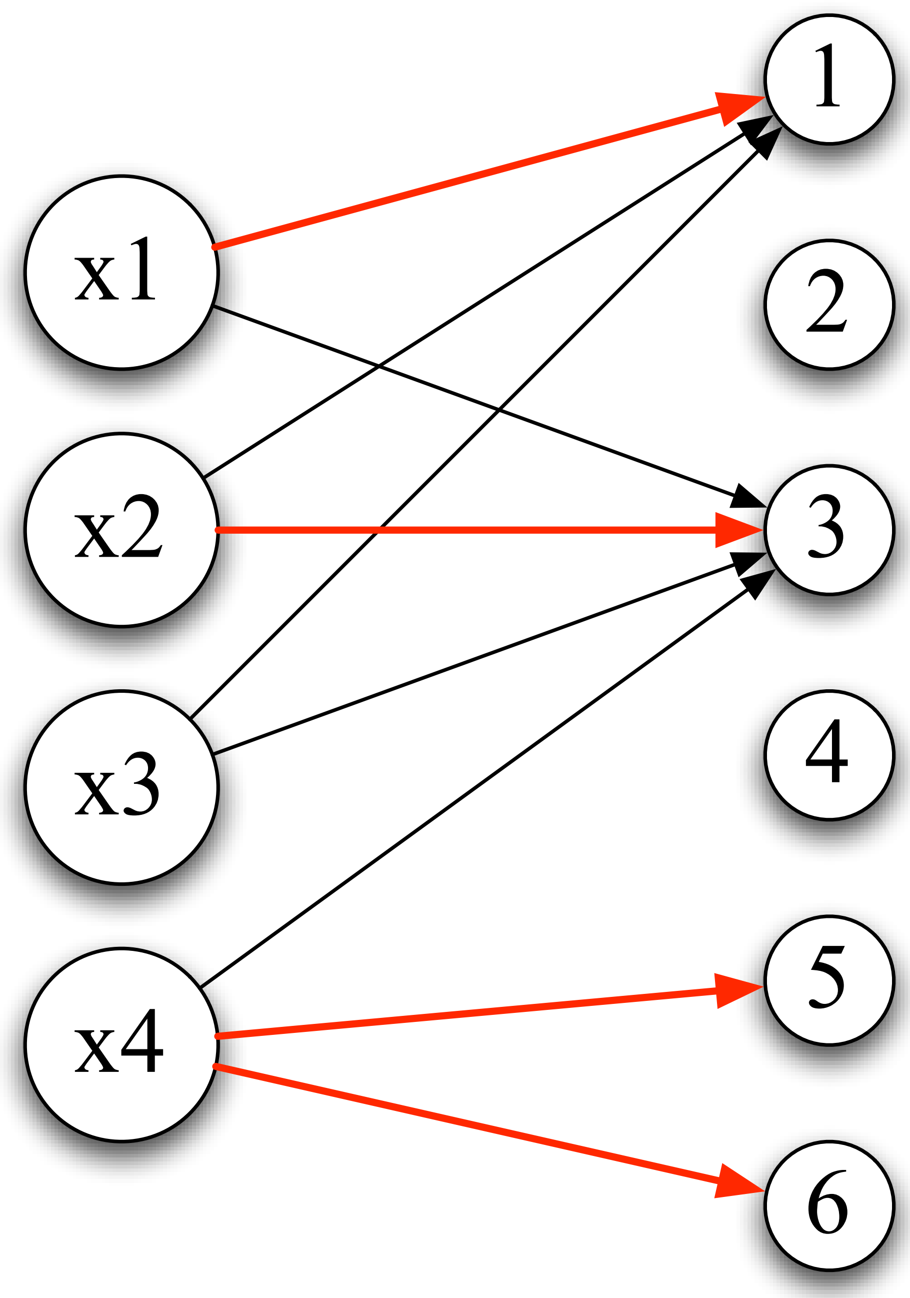
$x3 \in \{4,5\}$

$x4 \in \{5,6\}$



Failure

- If no maximal matching covering all variable nodes exists, we have detected failure



Notions

- For a given matching, we say that
  - an edge is matching if it belongs to the matching, otherwise it is free
  - a node is matched if incident to a matching edge, otherwise free

Maximal matching

- Can be computed in time $O(mn^{0.5})$, where m is the size of the union of the domains

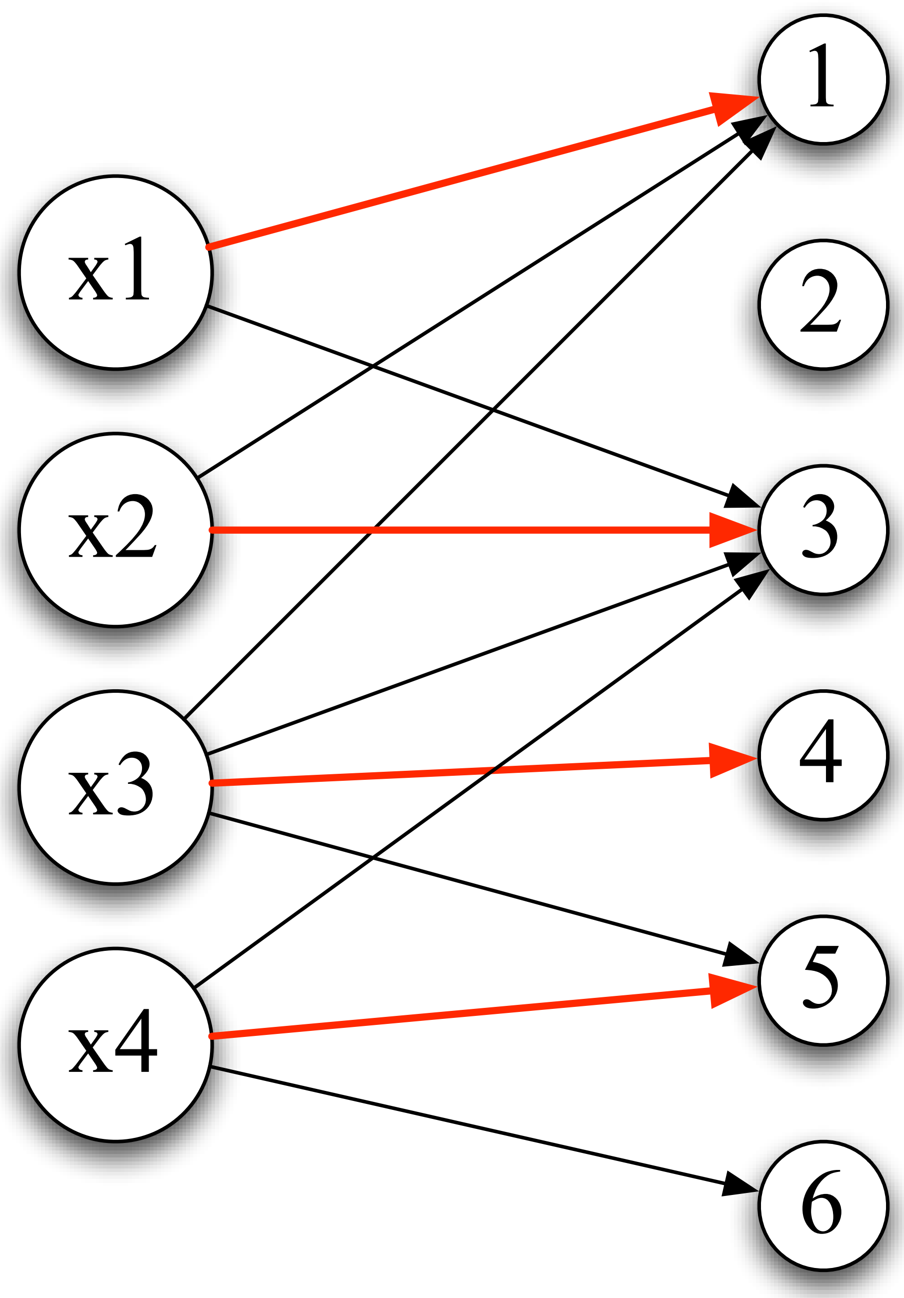
  (Hopcroft & Karp, 1973)

- Theorem:

  If M is some maximal matching in G, an edge belongs to any maximal matching in G iff it belongs to M, or to an *M-alternating cycle*, or to an *even M-alternating path* starting at an *M-free node*.

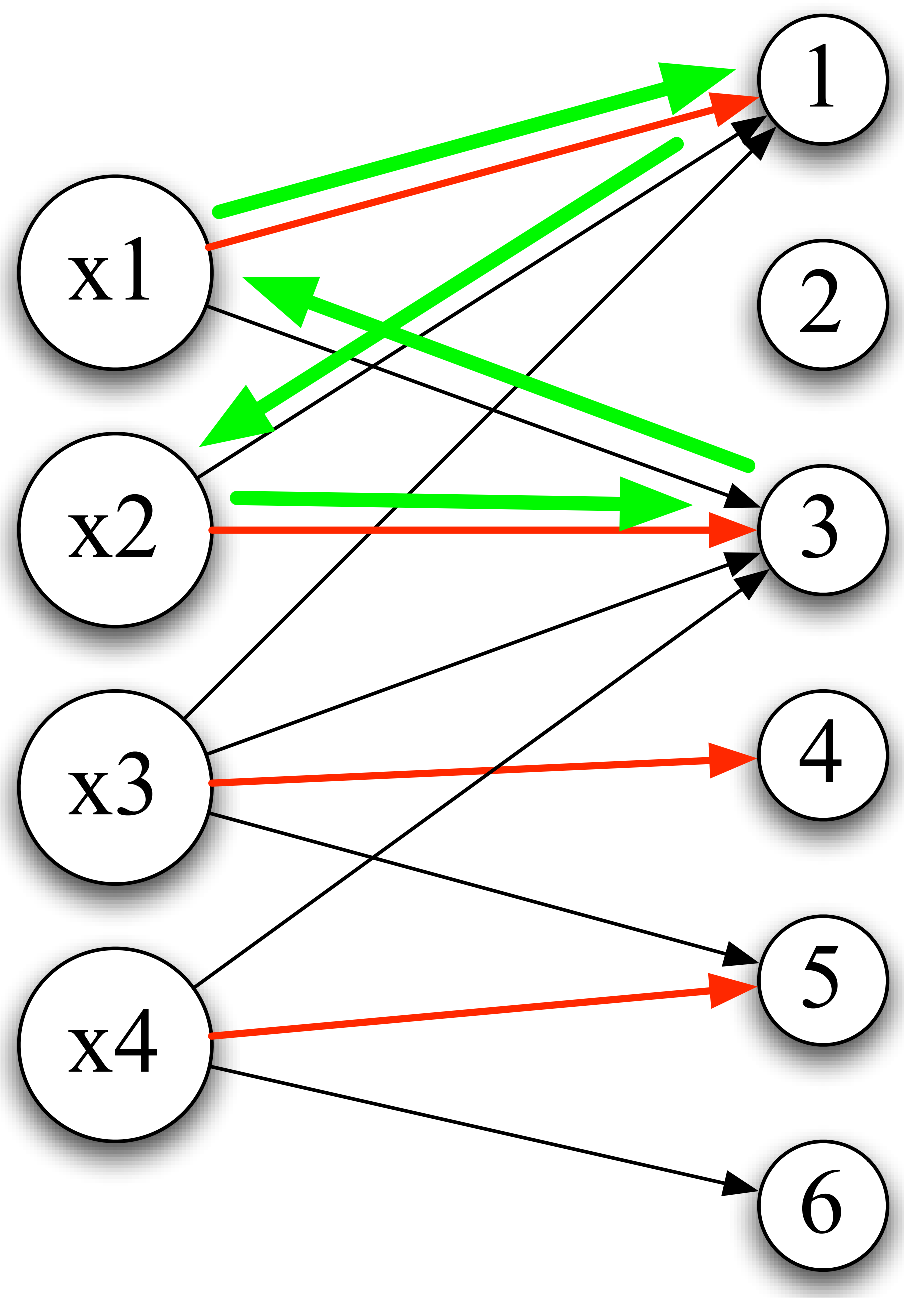
Maximal matching

- An M-alternating cycle



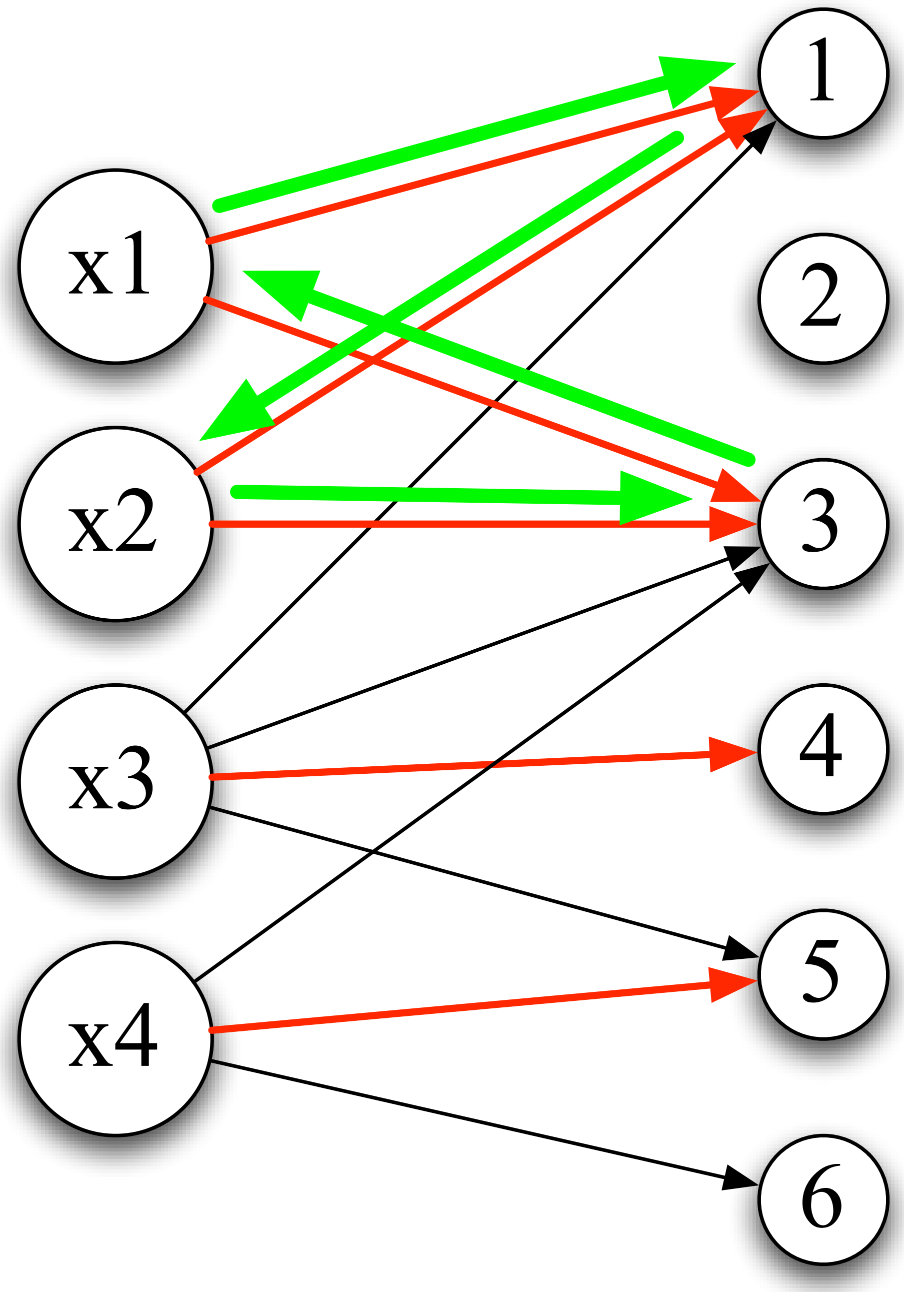
Maximal matching

- An M-alternating cycle



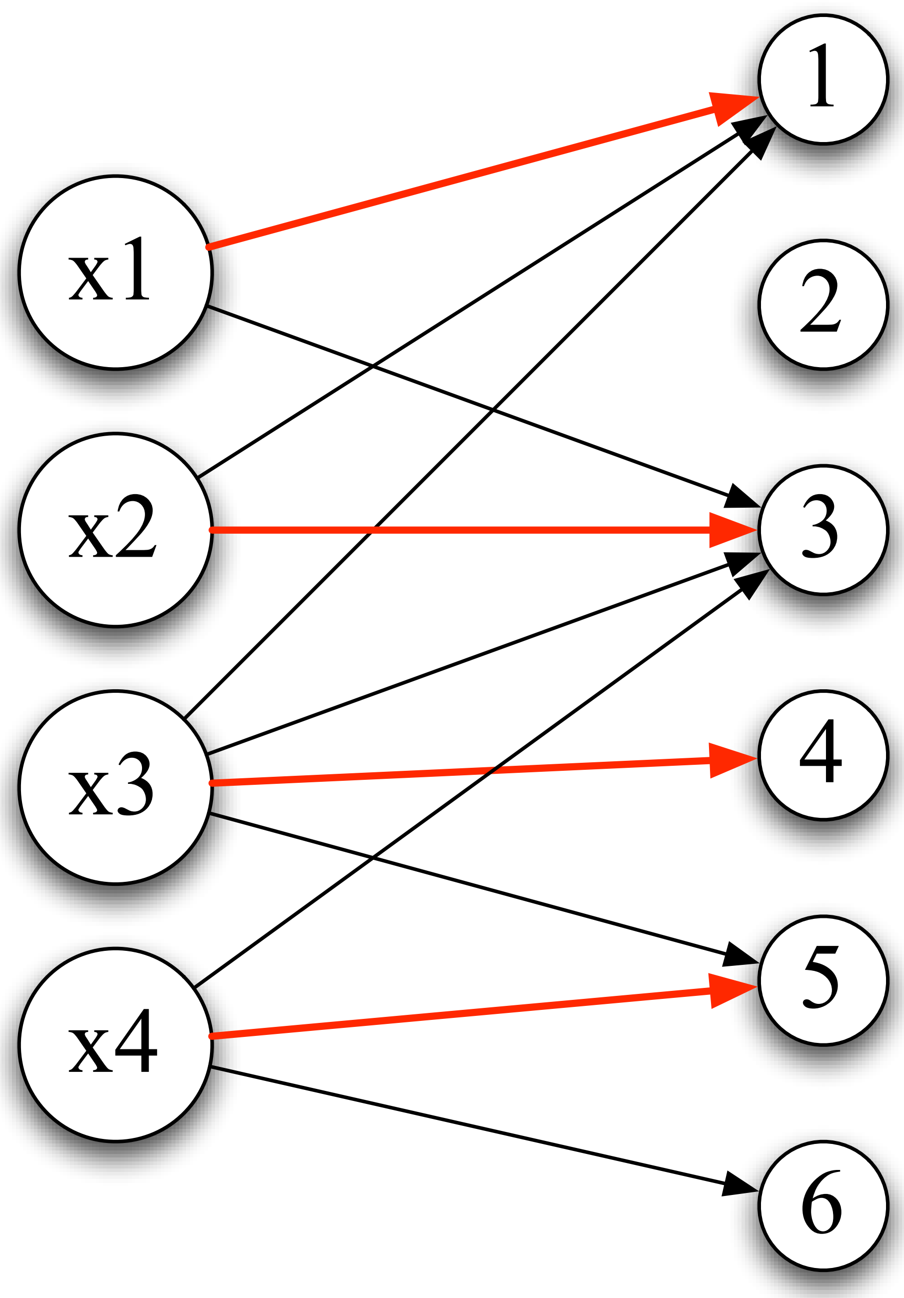
Maximal matching

- An M-alternating cycle



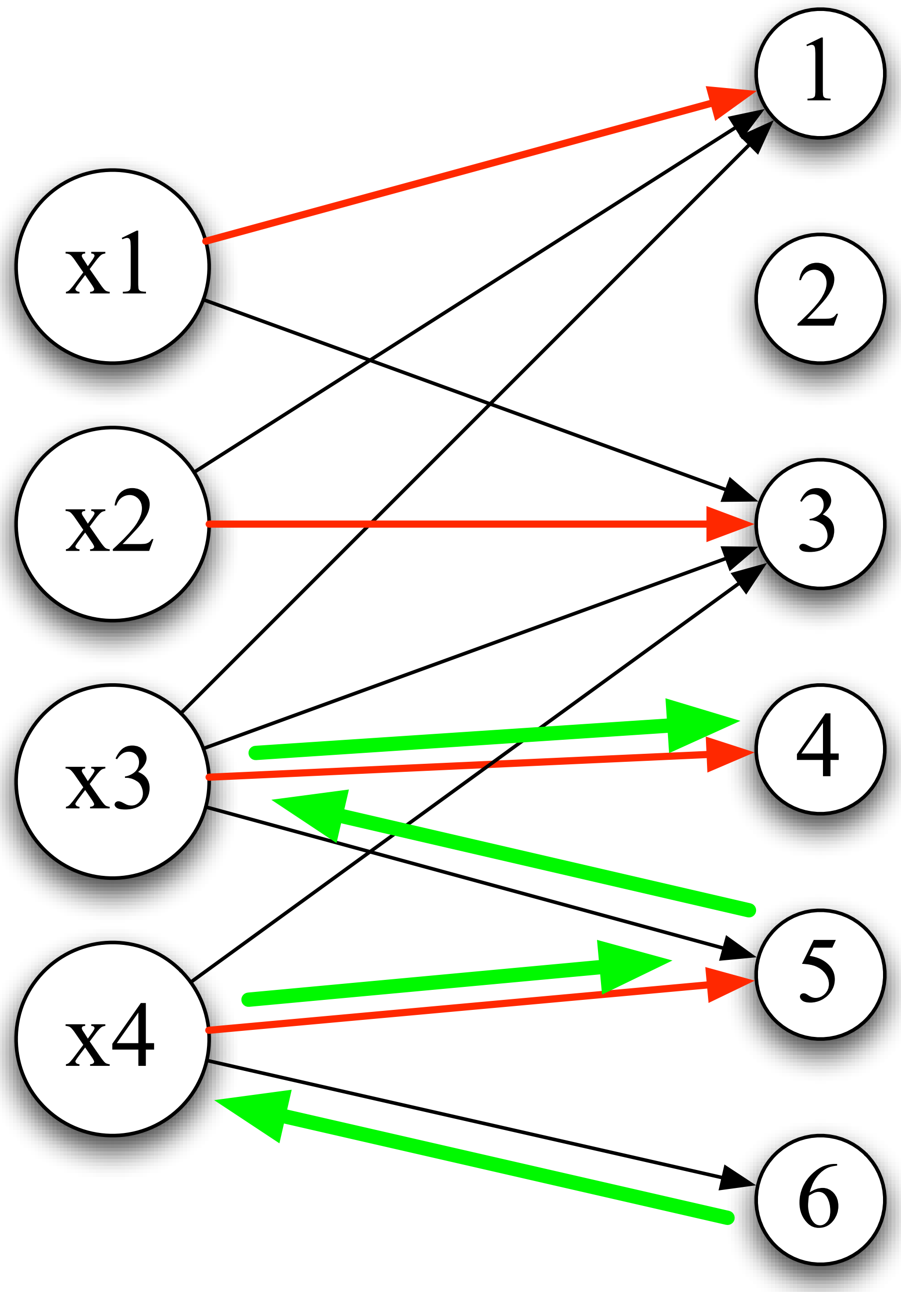
Maximal matching

- An even M-alternating path



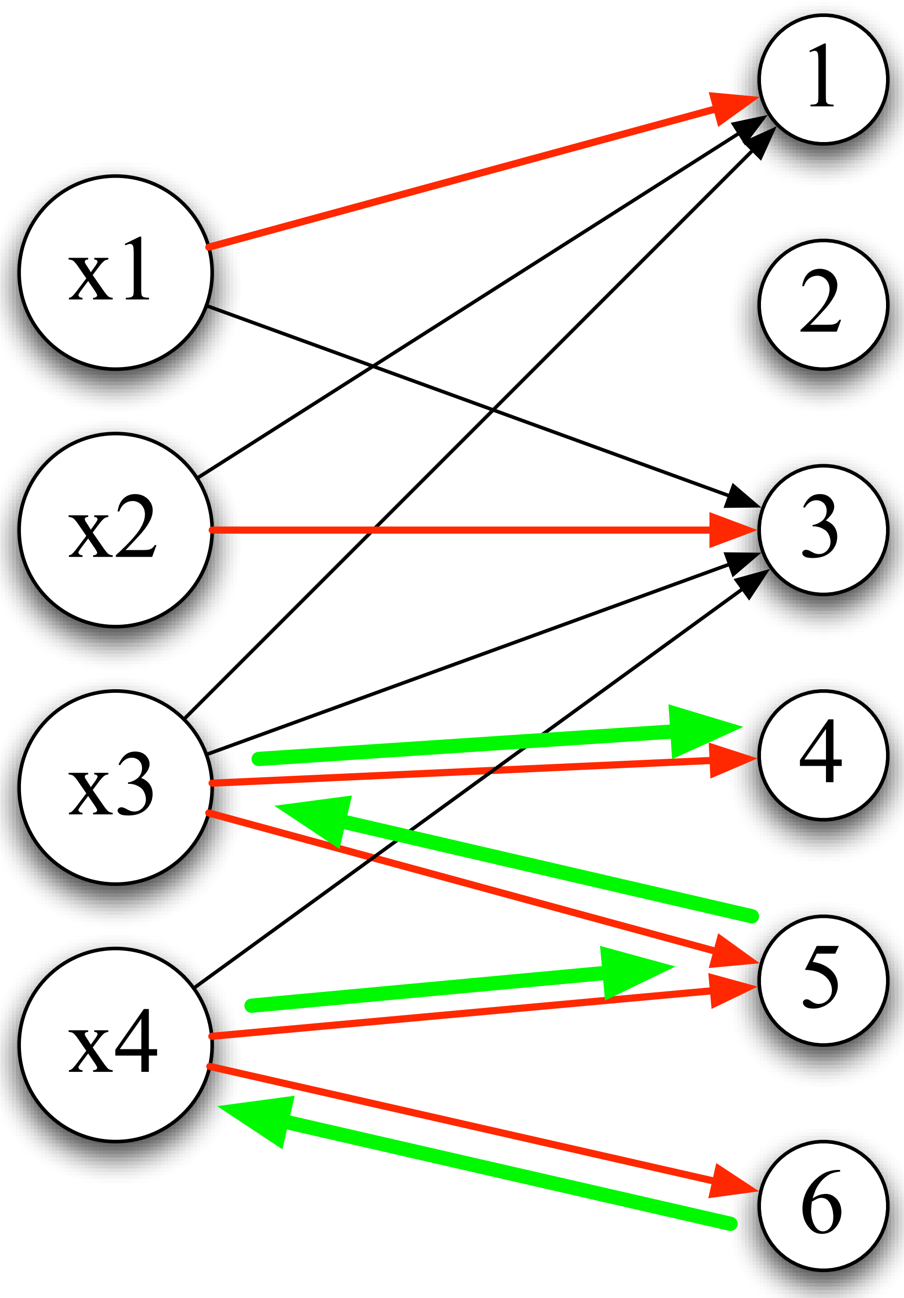
Maximal matching

- An even M-alternating path



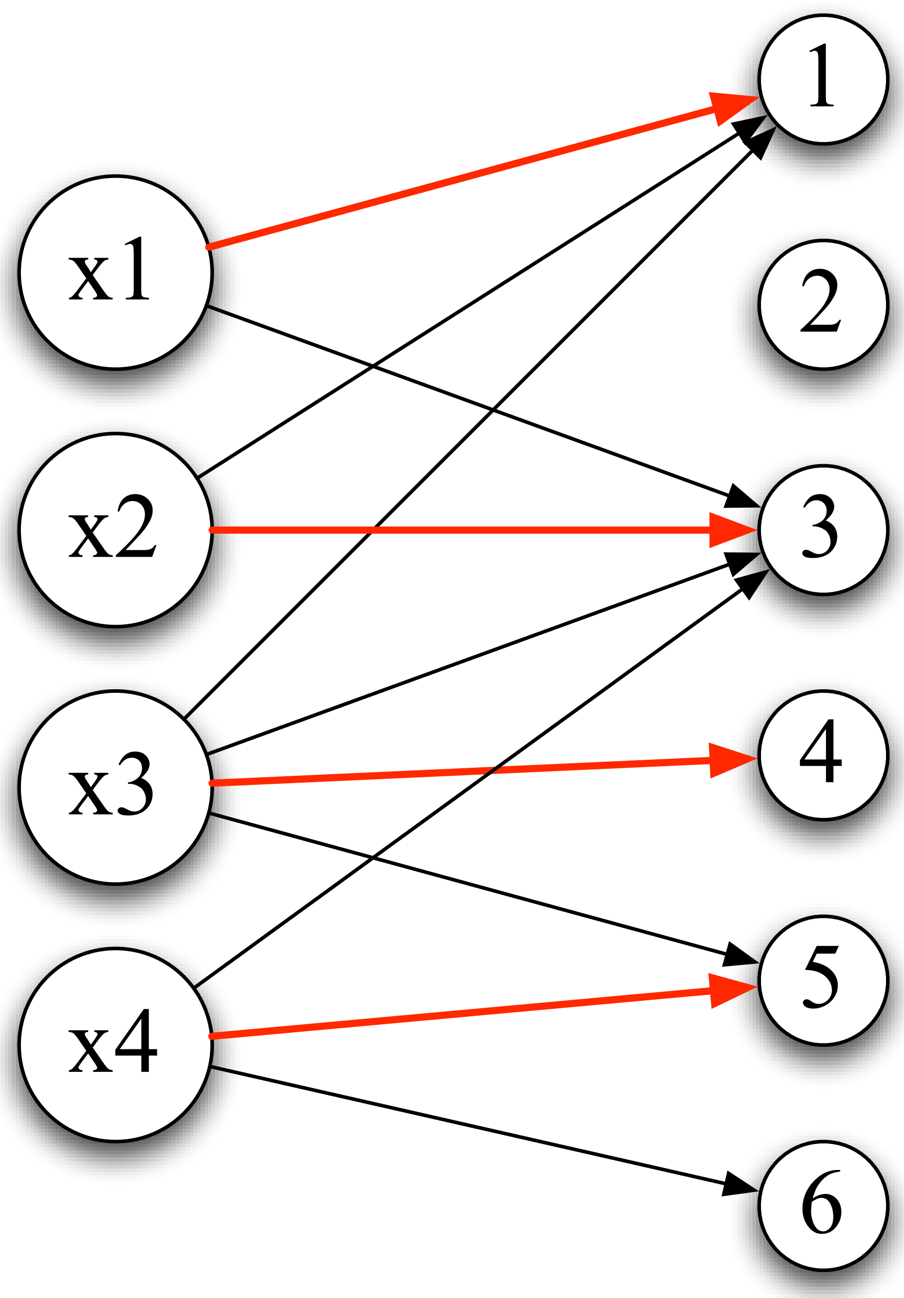
Maximal matching

- An even M-alternating path



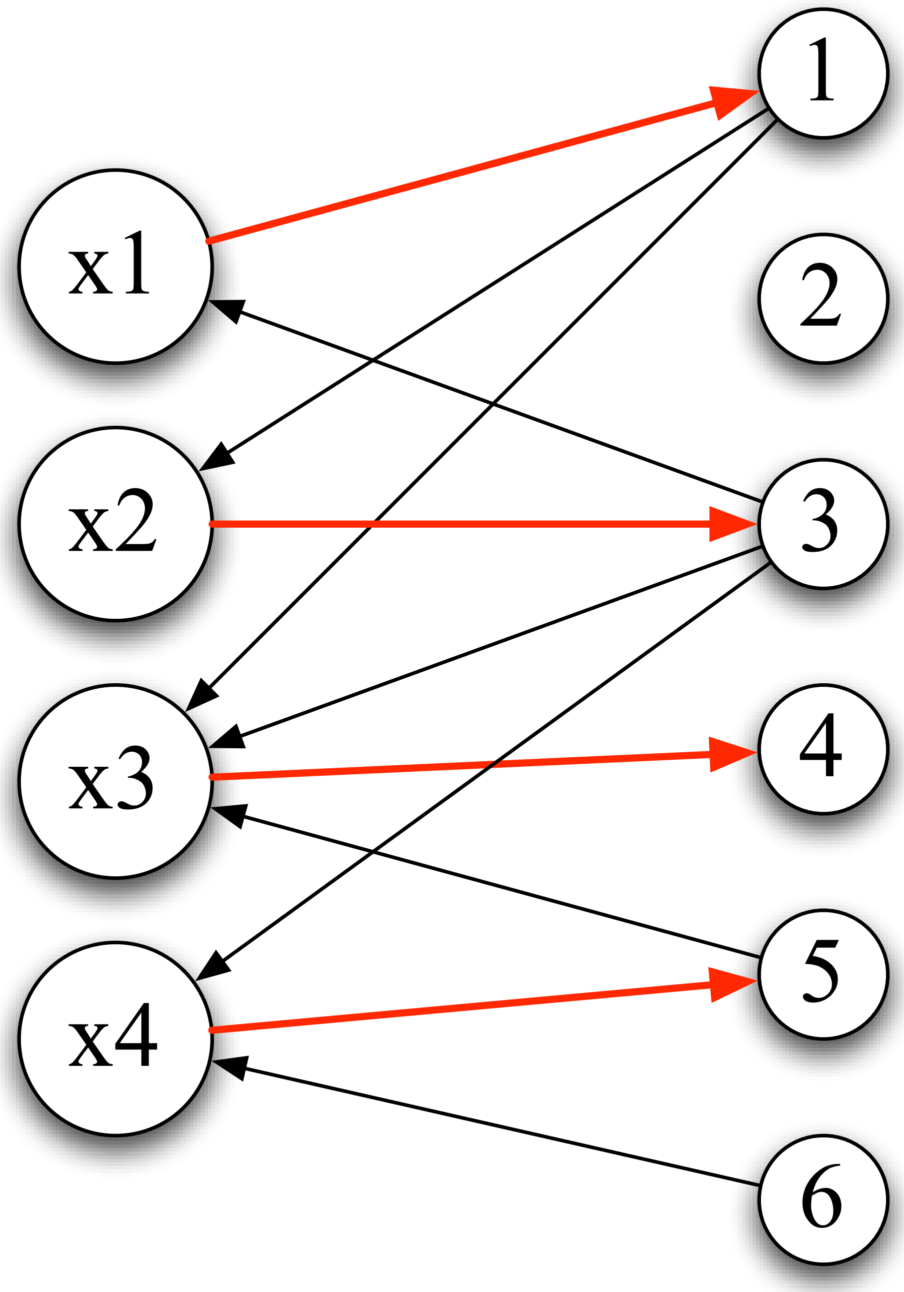
Maximal matching

- Reverse unmatched edges



Maximal matching

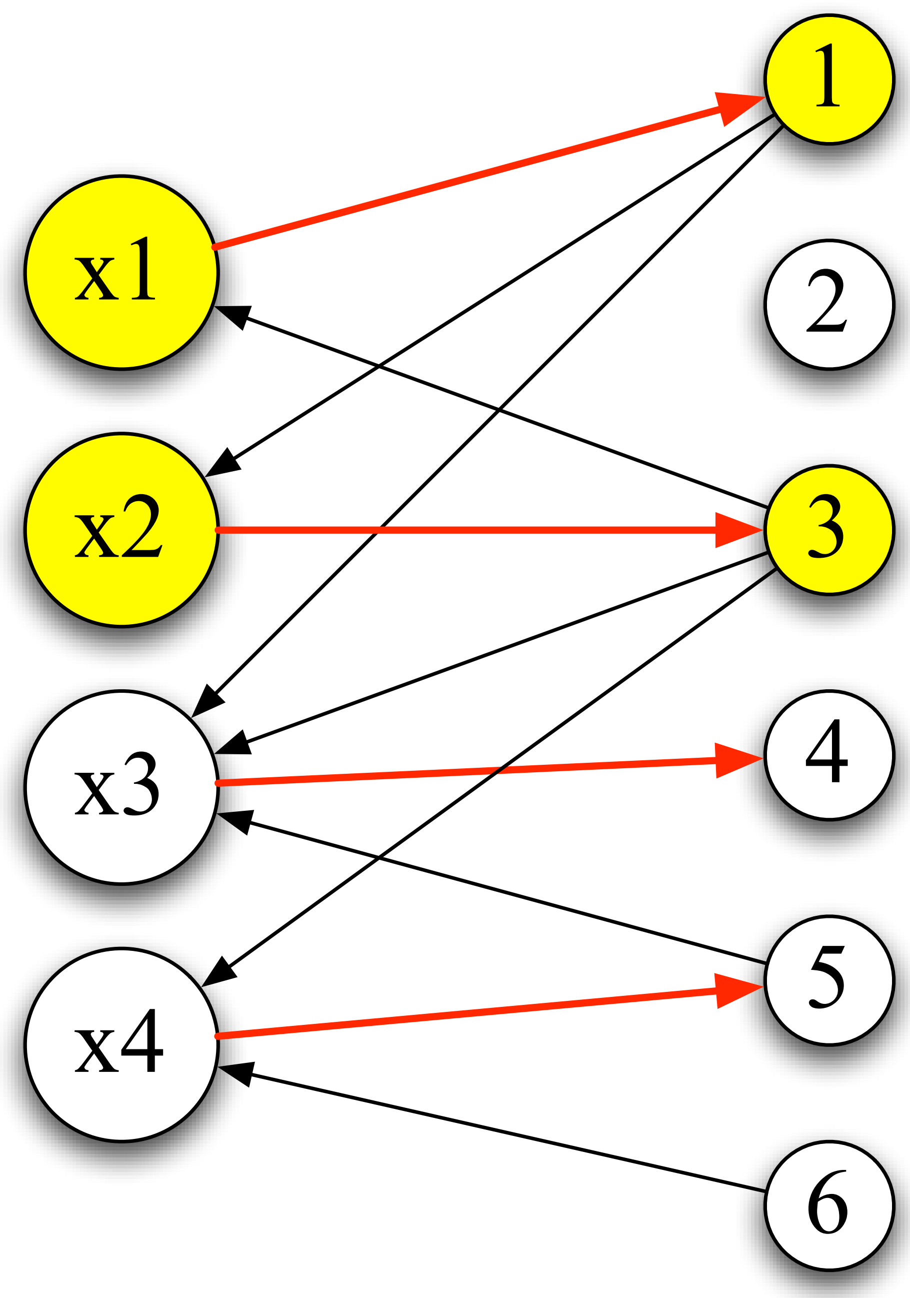
- Reverse unmatched edges



Maximal matching

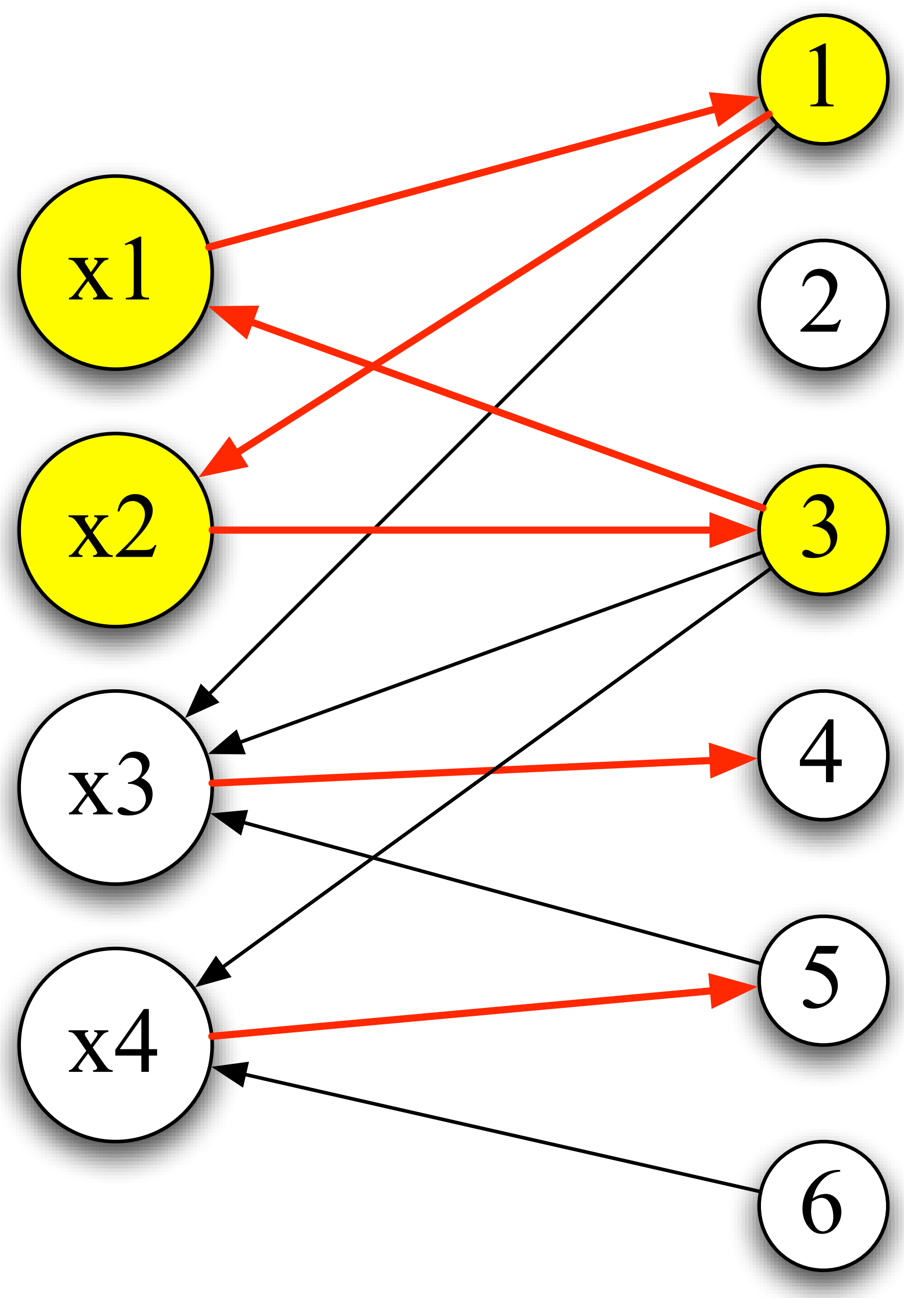
- Reverse unmatched edges
- Compute strongly connected components (SCCs)

maximal set of nodes where each node is reachable from any other node



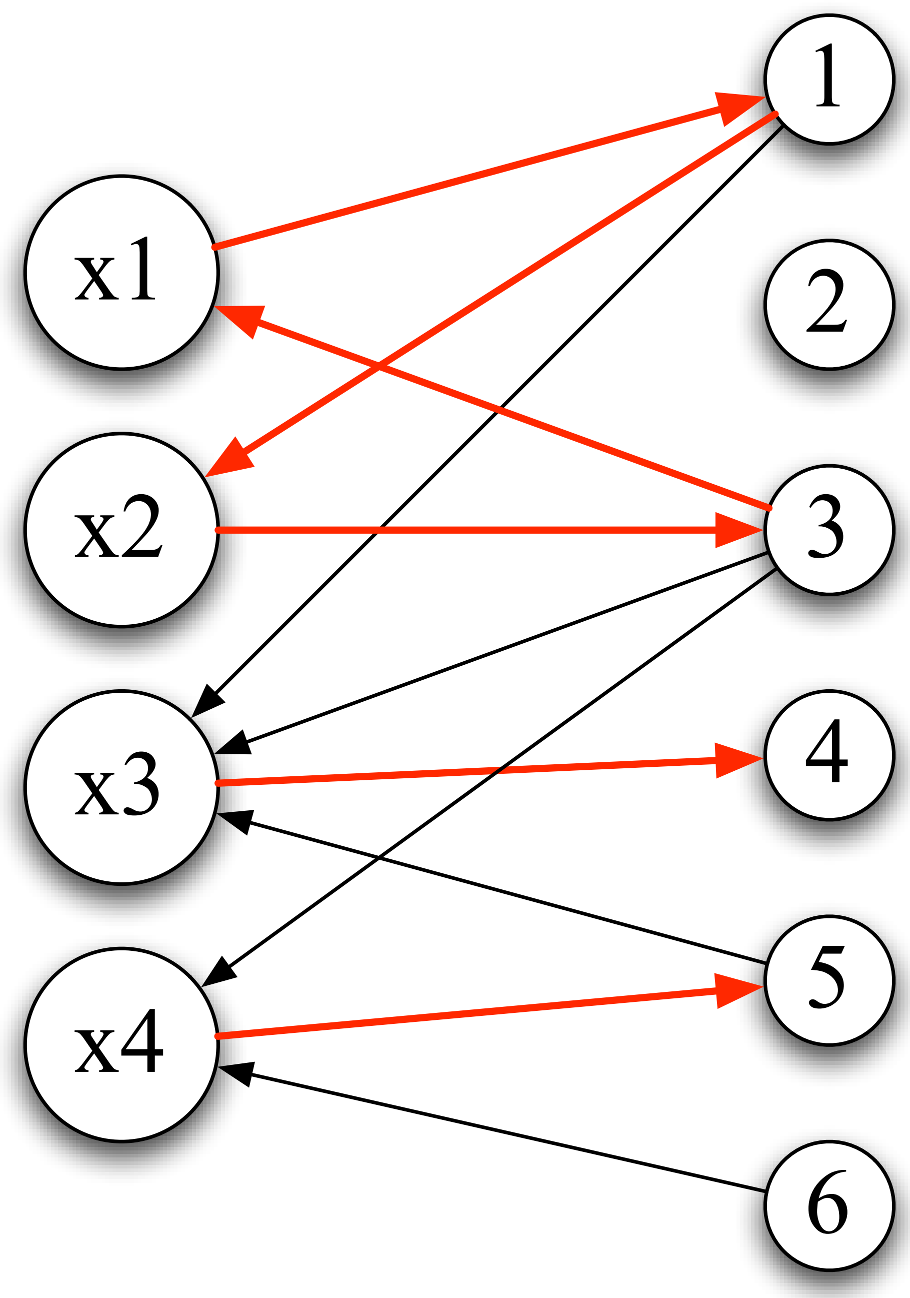
Maximal matching

- Reverse unmatched edges
- Compute strongly connected components
- Edges in one SCC are on an M-alt. circuit



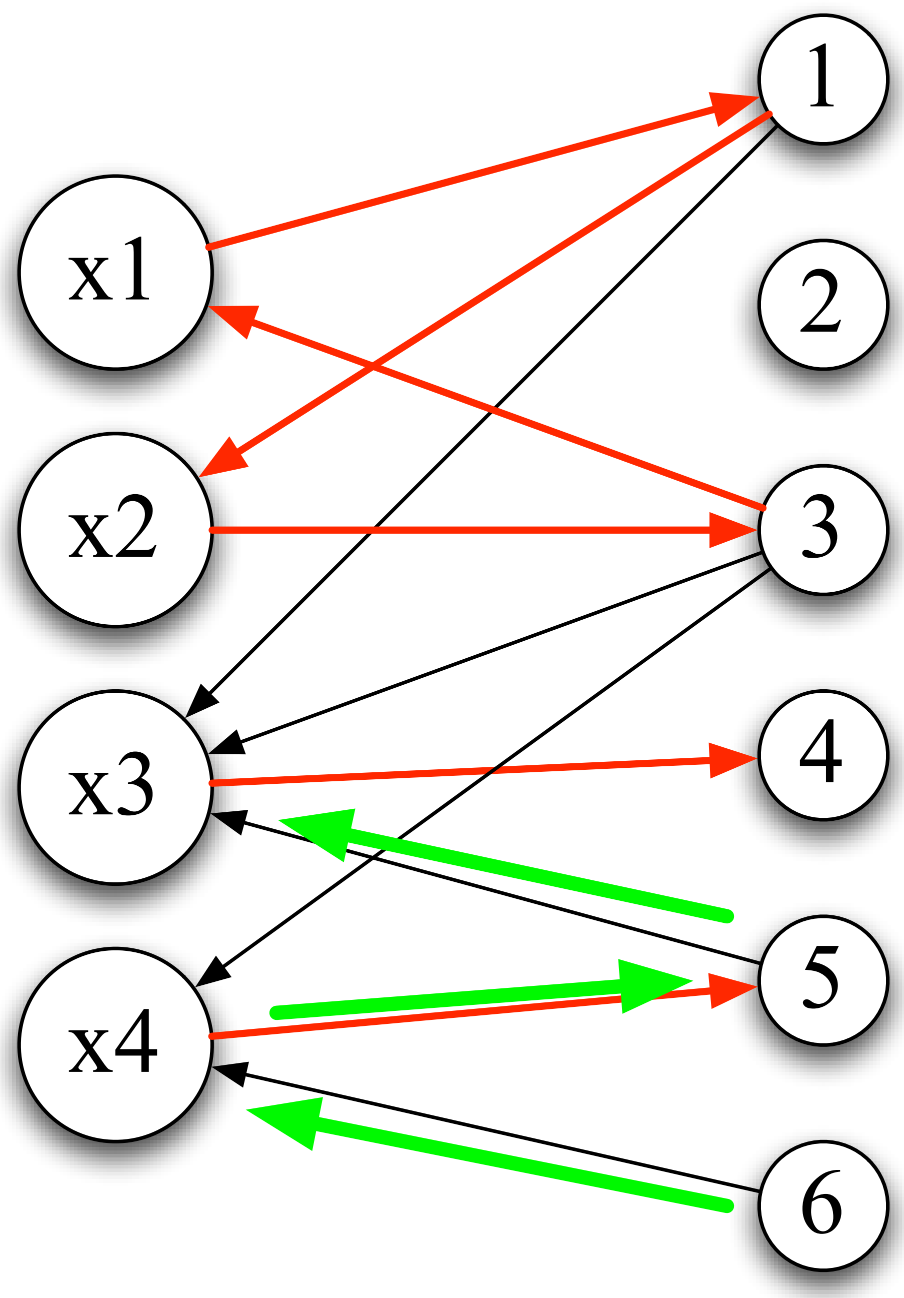
Maximal matching

- Edges on a directed path starting at a free vertex



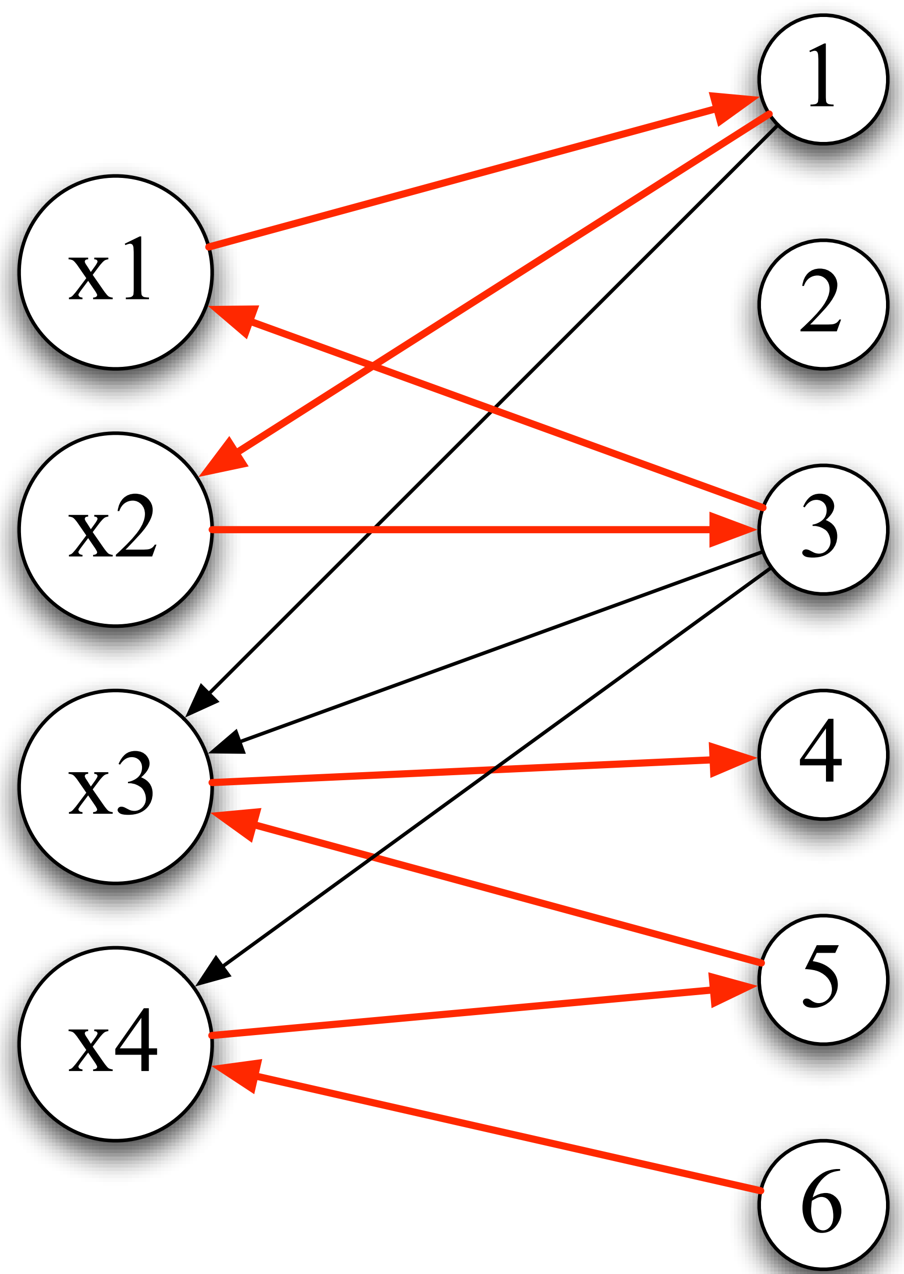
Maximal matching

- Edges on a directed path starting at a free vertex
- Breadth-first search



Maximal matching

- Edges on a directed path starting at a free vertex
- Breadth-first search



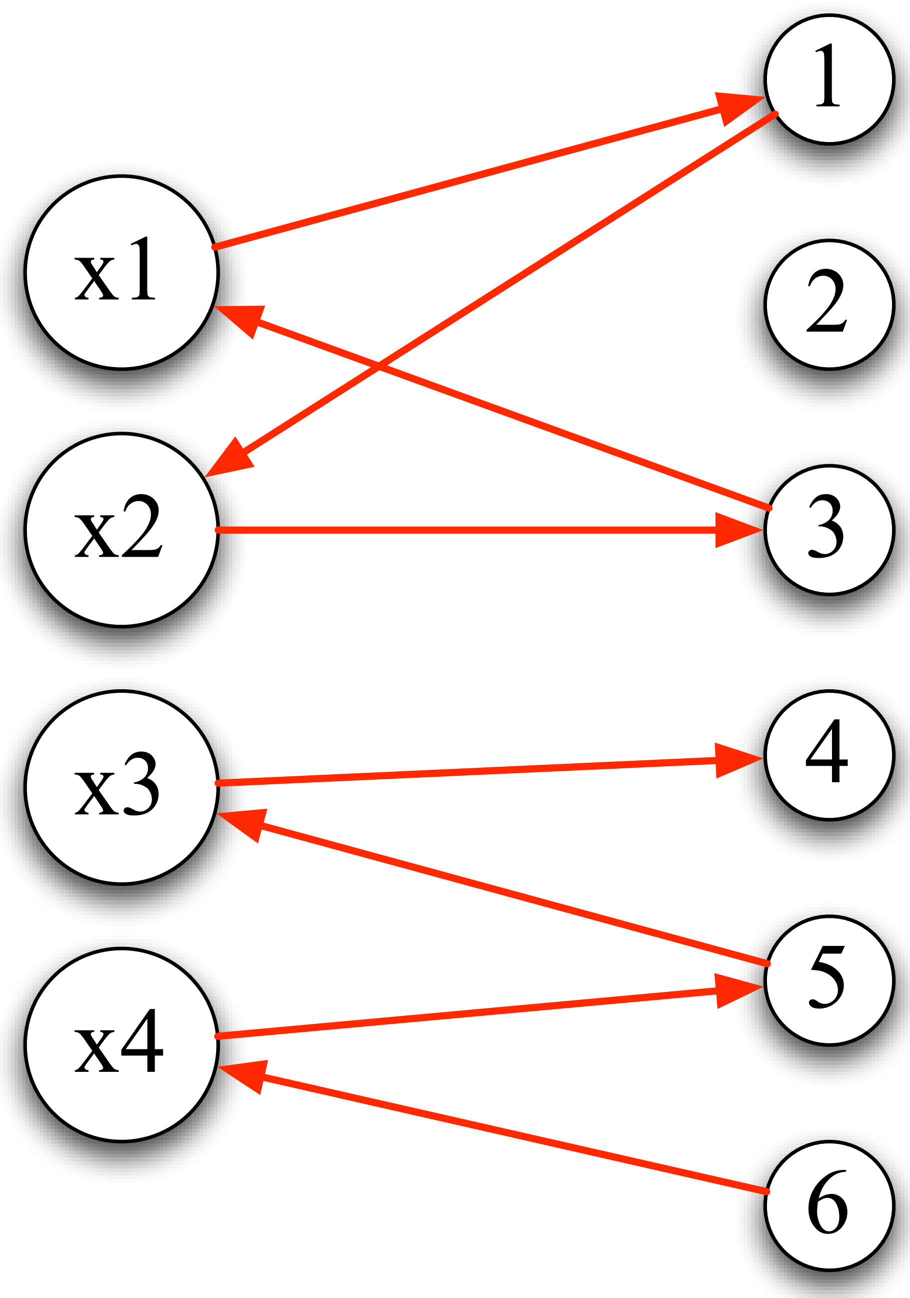
Compute new domains

$x1 \in \{1,3\}$

$x2 \in \{1,3\}$

$x3 \in \{4,5\}$

$x4 \in \{5,6\}$



Complete algorithm

- Construct the variable-value graph
- Compute maximal matching
- Orient the graph
- Find M-alternating cycles (SCCs)
- Find even M-alternating paths (graph search)
- Remove edges + narrow domains

Runtime

- Construction: $O(n+m)$
- Matching: $O(mn^{0.5})$
- SCC: $O(n+m)$ (Tarjan, 1972)
- Directed path: $O(m)$

- This gives overall complexity $O(mn^{0.5}) = O(n^{2.5})$

Optimizations

- Consider not only consistent and inconsistent edges, but also *vital* edges
- A vital edge is one that is contained in *all* matchings
- Vital edge between x and j means x must be assigned to j

Optimization: Incrementality

- Keep the variable-value graph between invocations
- When the propagator is run again, update the matching accordingly

Bounds consistency

- Efficient algorithms
  - based on Hall intervals O(n log n)

    (Puget, 1998) (Lopez-Ortiz & Quimper & al., 2003)
  - based on graphs & matchings O(n)

    (Mehlhorn & Thiel, 2000)

Bounds vs. domain consistency

- Bounds: only consider endpoints
- Domain: consider whole domains

Often a difference of O(m) if m is the size of the domains!

Extension: Global Cardinality

- For each value, give lower and upper bound on how often it may be taken by the variables.
- distinct(x1,...,xn) = gcc(x1,...,xn,0,...,0,1,...,1)

  (all values at least 0 times and at most once)
- Algorithm by Régin (very similar to distinct)

Does it pay off?

- In most cases, domain consistent distinct leads to considerably smaller search trees than naive version
- In some cases, bounds consistent distinct is "just as strong"

  (Schulte, Stuckey, 2001)
- Try it out! (exercise)

Summary

- Hard problems require strong propagators
- Domain consistency is feasible for some constraints
- Global propagation algorithms require insight into structure of the constraint

This week's exercises

- Implement propagators in Alice!
- You will use ECoDE, the *educational constraint development environment*

ECoDE

- Implemented in Alice
- You can look at the main loop, branchings, and propagators
- 500 loc
- Same interface as Gecode, so you can use the explorer

ECoDE: propagators

```
fun less(s, x, y) =
    let
        fun f s = if adjmin(s, y, min(s,x)+1) andalso
                     adjmax(s, x, max(s,y)-1) then
                      if max(s,x)<min(s,y)
                      then PS_SUBSUMED [x,y]
                      else PS_NOFIX
                  else PS_FAILED
    in
        Space.addPropagator(s, [x,y], "less", f)
    end
```


less: space * var * var → unit

Exercise

- Implement linear equations
- Implement distinct (naive and domain consistent)
- Graded exercises, submit by June 2

Have fun!

Outlook

We know how to propagate, so how does *search* work?

spaces, search engines, recomputation, explorer