Propagators

CP course, lecture 4

Recapitulation

- Formal model: CSP (X,C)

  with $X = \{x_1 : D_1, \ldots, x_n : D_n\}$
- Domain reduction + Search

Recapitulation

check failure

$$\frac{\langle X \cup \{x : \emptyset\} \; ; \mathfrak{C} \rangle}{\textbf{fail}}$$

split

$$\frac{\langle X \cup \{x : D\} \; ; \mathfrak{C} \rangle \qquad |D| > 1 \qquad D = D_1 \uplus \cdots \uplus D_k}{\langle X \cup \{x : D_1\} \; ; \mathfrak{C} \rangle \mid \cdots \mid \langle X \cup \{x : D_k\} \; ; \mathfrak{C} \rangle}$$

narrowing

$$\frac{\langle X \cup \{x : D\} \; ; \mathfrak{C} \cup \{C\} \rangle \qquad d \in D \qquad \mathsf{sat}(C) \cap \{\, \alpha \in \mathsf{ass}(X) \mid \alpha x = d \,\} = \emptyset}{\langle X \cup \{x : D - \{d\}\} \; ; \mathfrak{C} \cup \{C\} \rangle}$$

Recapitulation

- Algorithm, specified by inference rules:

  check failure

  split domains

  reduce domains

This makes it efficient!

Propagators

- Are the *workhorses* of a CP system
- *Implement* constraints through domain reduction + checking
- Compute on a *Constraint Store*

Constraint Store

- Mapping from variables to current domain

  Store s corresponds to domain dom

- Store s_1 stronger t

  $\forall x \in X$: $s_1(x) \subseteq s_2($

written: $\{x_1 : D_1, \ldots, x_n : D_n\}$

Constraint Store

- Mapping from variables to current domain
  Store s corresponds to domain dom
- Store s_1 stronger than s_2 ($s_1 \leq s_2$):
  $\forall x \in X$: $s_1(x) \subseteq s_2(x)$
- Store s_1 strictly stronger than s_2 ($s_1 < s_1$):
  $s_1 \leq s_2$ and $s_1 \neq s_2$
- $(S, <)$ is a well-founded order (all is finite)

Propagators

$$\frac{\langle X \cup \{x : D\}\, ;\, \mathfrak{C} \cup \{C\}\rangle \qquad d \in D \qquad \mathsf{sat}(C) \cap \{\, \alpha \in \mathsf{ass}(X) \mid \alpha x = d \,\} = \emptyset}{\langle X \cup \{x : D - \{d\}\}\, ;\, \mathfrak{C} \cup \{C\}\rangle}$$

- Functions S→S (mapping stores to stores)
- Properties:
  - contracting ($p(s) \leq s$)
  - correct (maintain solutions)
  - checking (decisive for assignments)

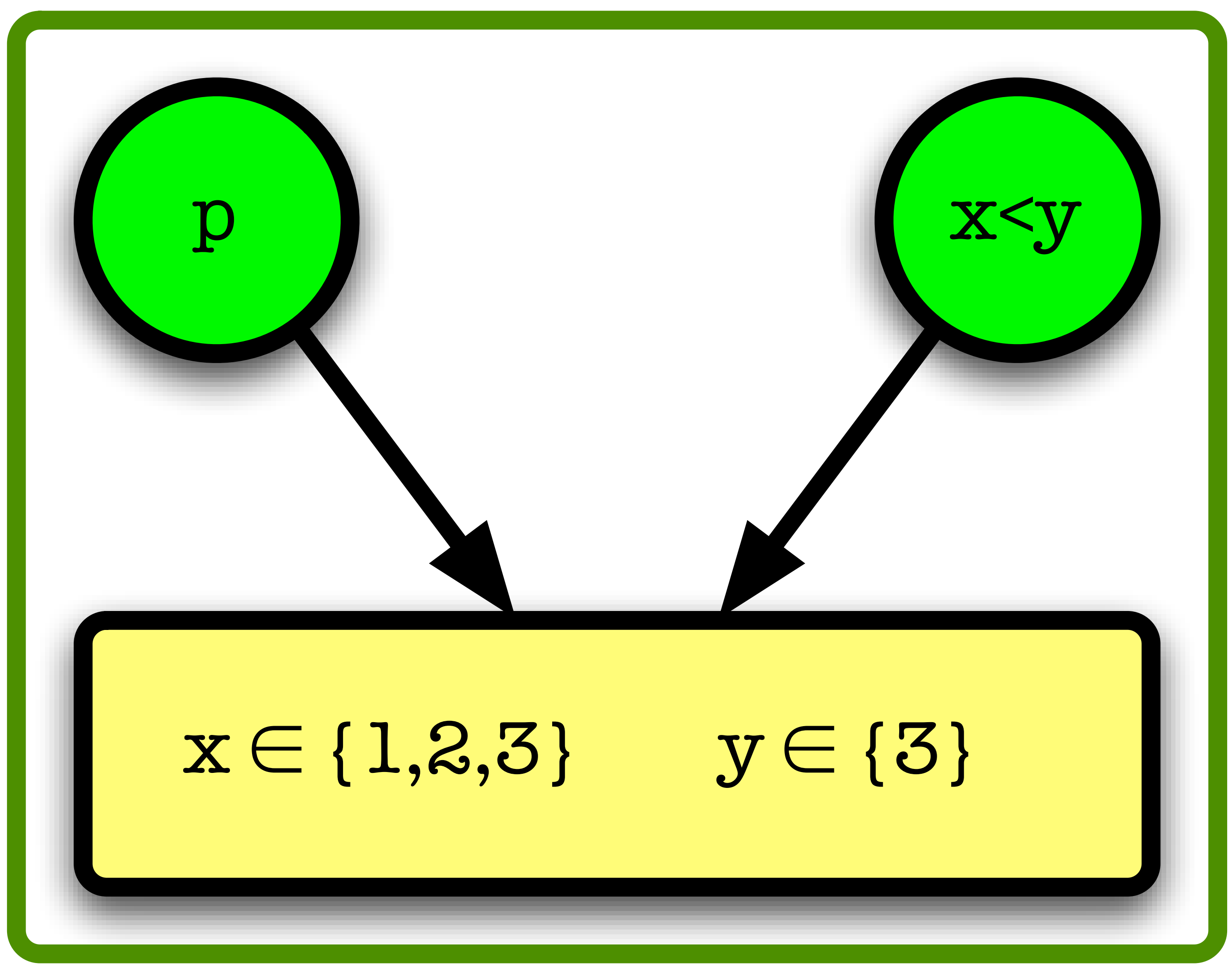
Domain Reduction

Test

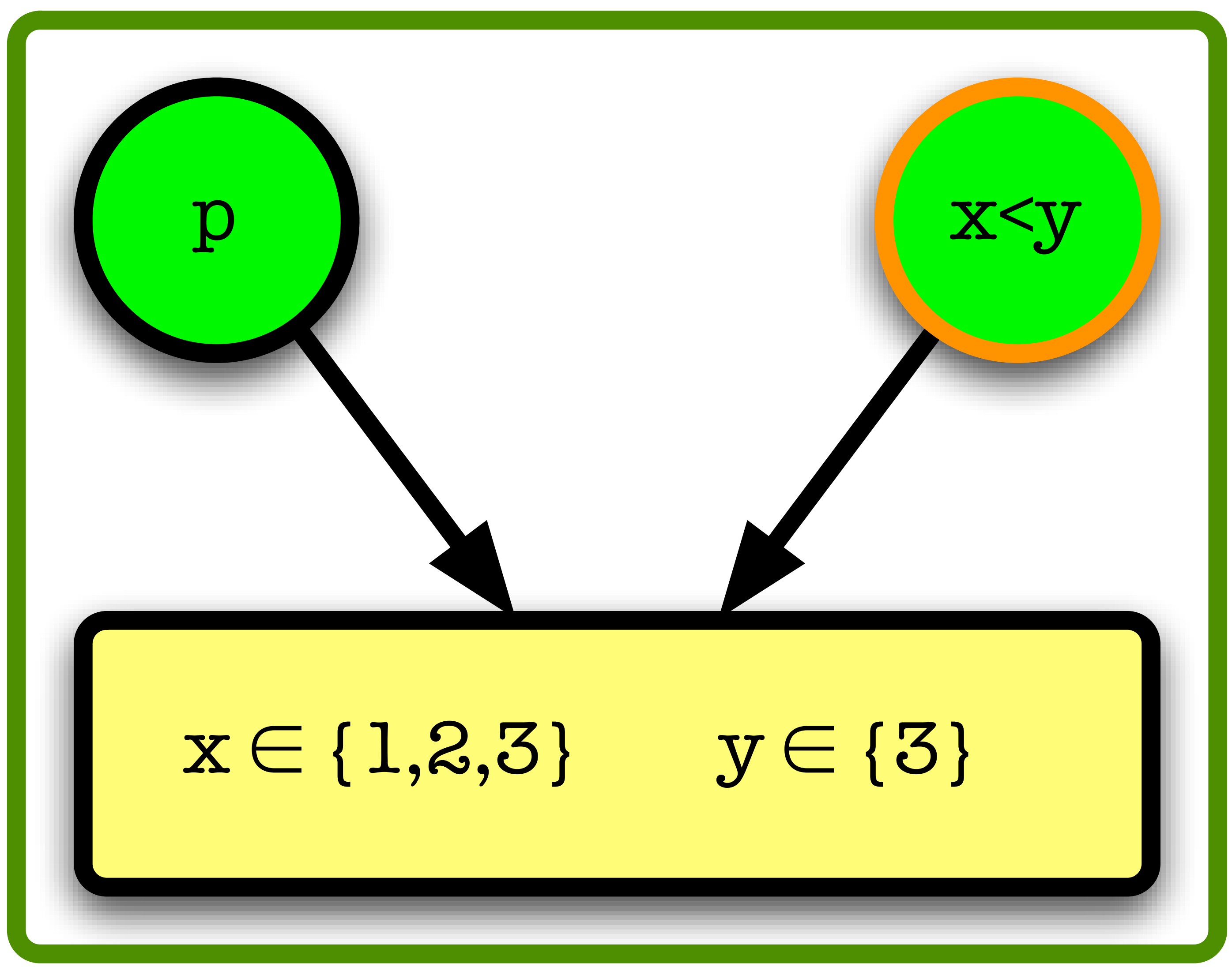
Is that enough?

- Assume a propagator
  $p(s) = \textbf{if } s(x) = \{1,2,3\} \textbf{ then } \{x{:}\{1\}\}$
  $\qquad \textbf{else } s$

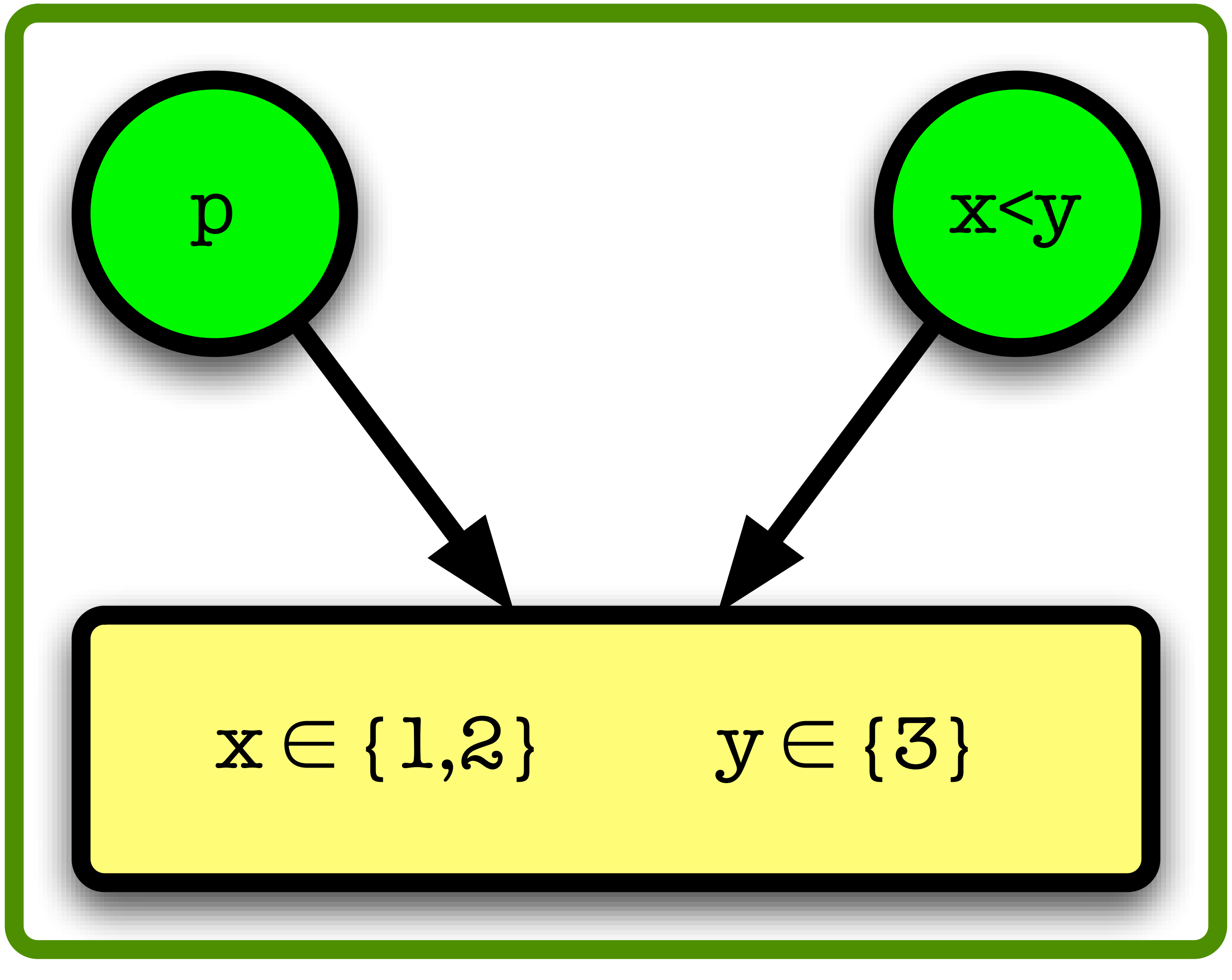
Example



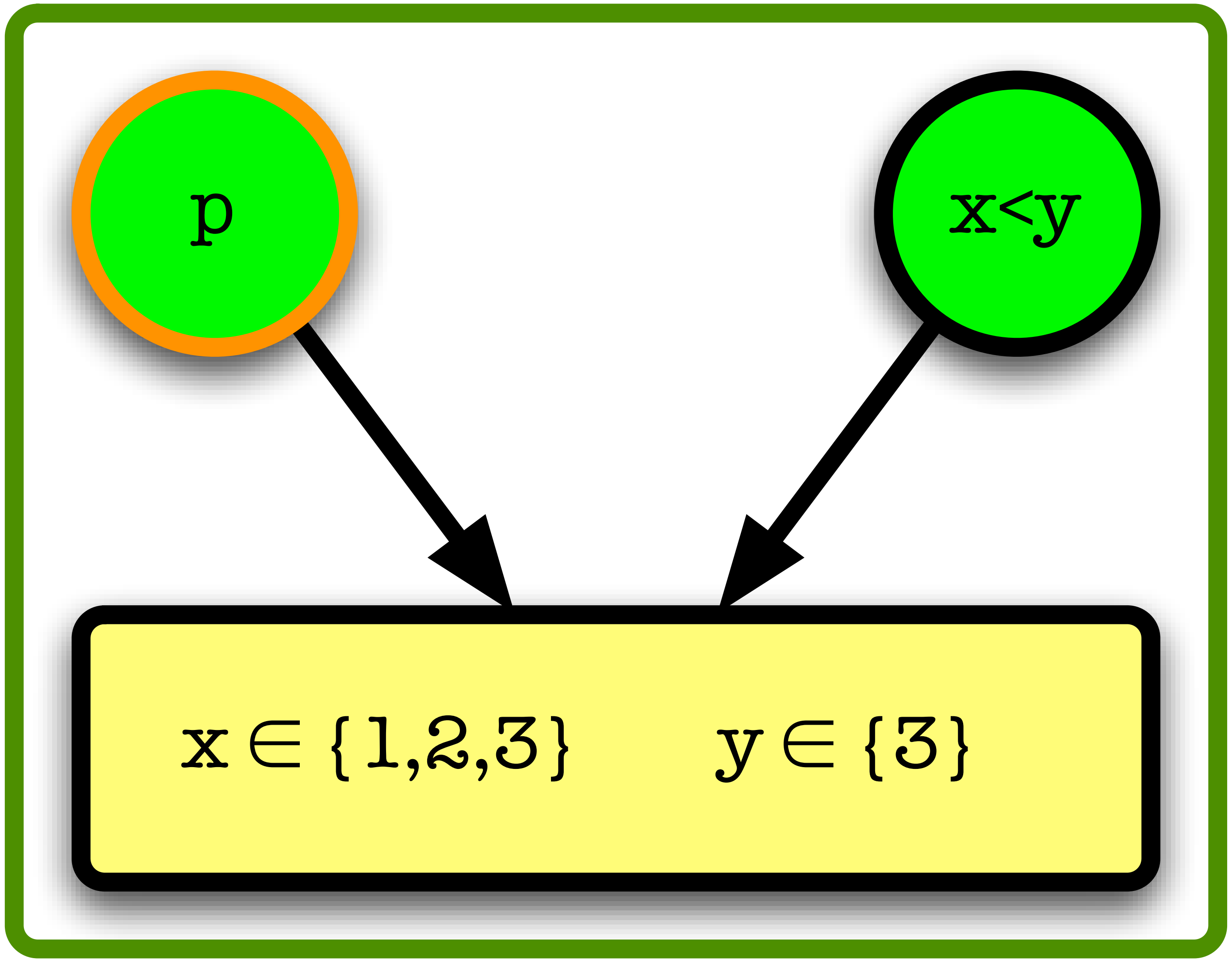
Example



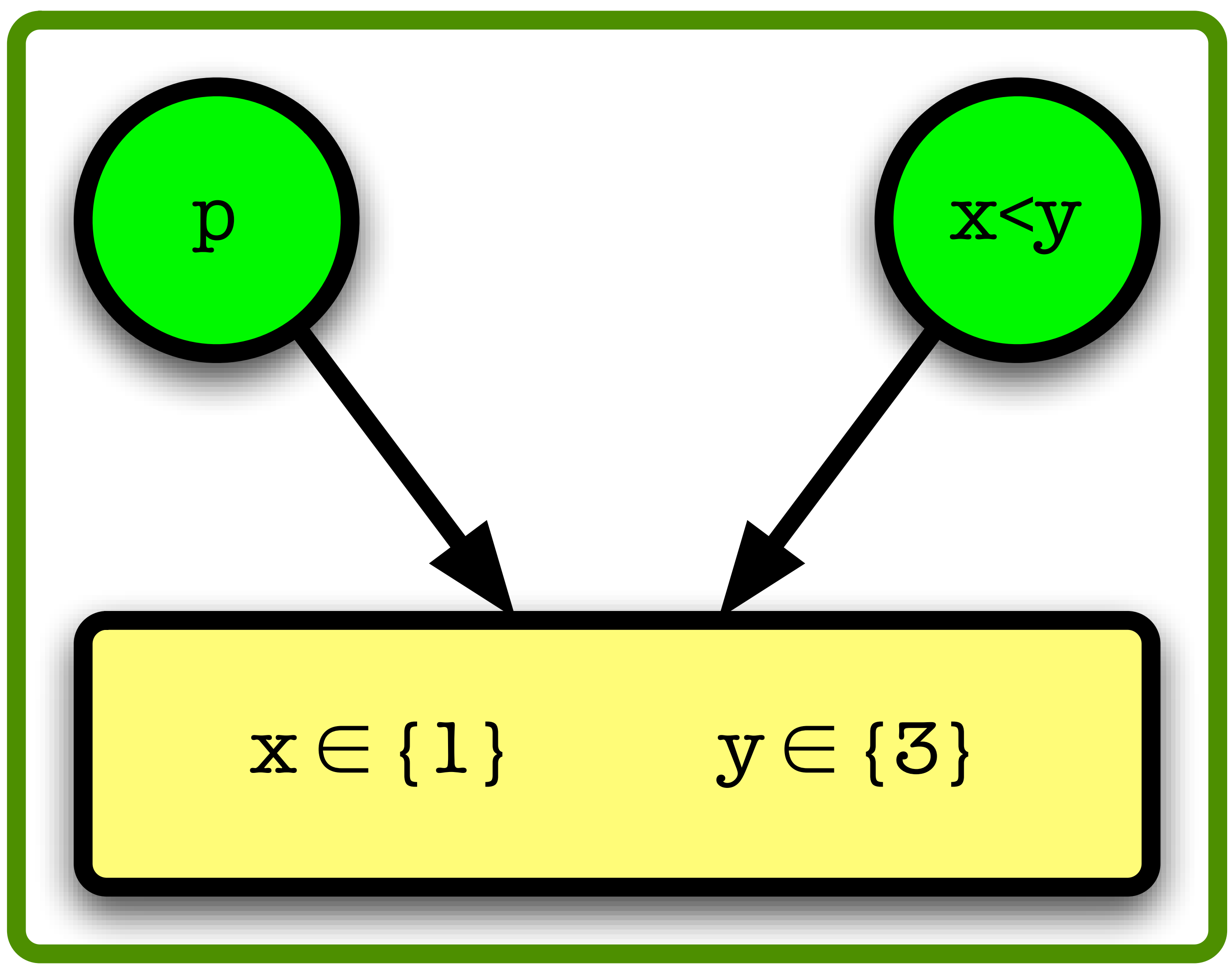
Example



Example



Example



That's not enough!

- Assume a propagator

  $p(s) = \textbf{if}\ s(x) = \{1,2,3\}\ \textbf{then}\ \{x:\{1\}\}$

  $\qquad \textbf{else}\ s$

  and

  $s_1 = \{x:\{1,2,3\}\} \quad s_2 = \{x:\{1,2\}\}$

- Then $s_2 \leq s_1$ but $p(s_1) \leq p(s_2)$
- Propagators must be monotonic:

  $s_1 \leq s_1 \Rightarrow p(s_1) \leq p(s_2)$

Maintaining solutions

- Assume p implements C, and $\alpha \in C$

  then $p(\alpha) = \alpha$
- To show: $\alpha \in s$ implies $\alpha \in p(s)$

  $\alpha \in s \Leftrightarrow \alpha \leq s$

  $\Leftrightarrow p(\alpha) \leq p(s)$ (monotonicity)

  $\Leftrightarrow \alpha \leq p(s)$

  $\Leftrightarrow \alpha \in p(s)$

How much can we propagate?

- Idea: Propagate as much as possible, only then resort to branching
- This means:

Compute the largest simultaneous fixpoint of all propagators!

(exists + is unique)

Naive propagation

```
propagate(s,P) =
  while p in P and p(s) != s do
    s := p(s);
  return s;
```

- Preserves solutions
- Computes largest simultaneous fixpoint

Termination

- Store s_i at *i*-th iteration:

  $s_i < s_{i-1}$

- $(S,<)$ is well-founded
- Loop terminates!

Correctness

- Assume propagate(s,P) = s'
- Then for all p in P: p(s') = s'

  (follows from termination of the loop)
- s' is largest simultaneous fixpoint smaller than s (proof left as an exercise)

Why is that naive?

- We always apply all propagators
- Probably a lot of them cannot contract
- We don't apply them in any particular order

Improvements

- Idea: only *some* variables' domains are narrowed during propagation
- Only run propagators that have a narrowed variable *in their scope*
- Maintain a set of "dirty" propagators
  - not known whether at fixpoint
  - all other propagators are at fixpoint

A bit more clever

```
propagate(s0,P) =
  s := s0; DP:=P;
  while DP not empty do
    choose p∈DP;
    s' := p(s); DP := DP-{p};
    MV := {x∈s | s(x) ≠ s'(x)};
    NP := {q∈P | ∃x∈scope(q): x∈MV};
    DP := DP ∪ NP;
    s := s';
  return s;
```

Dirty Propagators

Modified variables

New Dirty Propagators

May contain p!

Important properties

- It *still* computes the largest simultaneous fixpoint
- It *still* terminates

Loop invariant

- For all p in P-DP: p(s) = s
- After termination: for all p in P: p(s) = s
- Holds initially
- Is invariant

A bit more clever

```
propagate(s0,P) =
   s := s0; DP:=P;
   while DP not empty do
      choose p∈DP;

      s':= p(s); DP := DP-{p};
      MV := {x∈s | s(x) ≠ s'(x)};

      NP := {q∈P | ∃x∈scope(q): x∈MV};

      DP := DP ∪ NP;

      s := s';
   return s;
```

P-DP is empty

Move all propagators that might be not at FP to DP

Termination

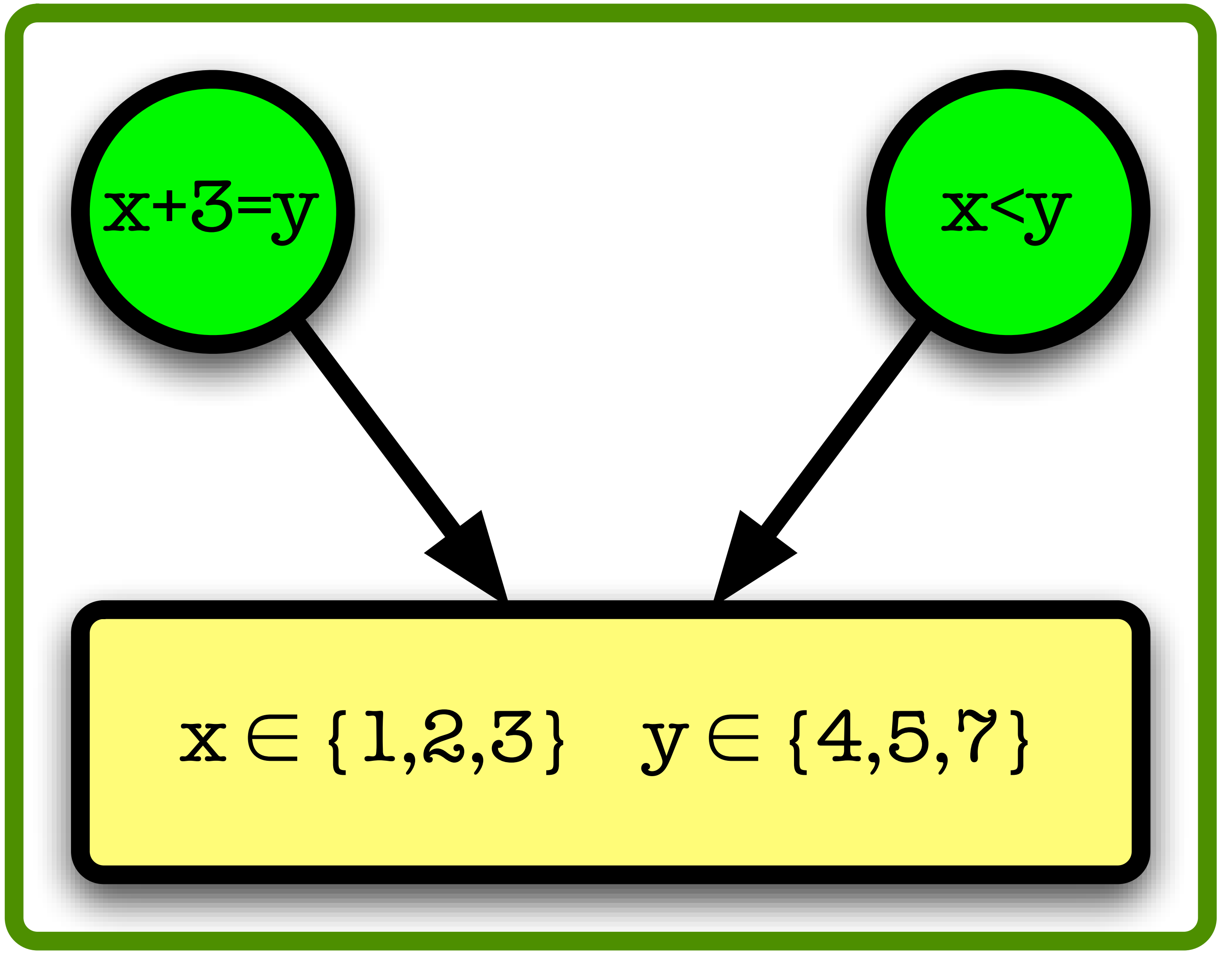
- Stores do not strictly decrease!
- But:
  MV empty $\Rightarrow$ p removed from DP

  MV not empty $\Rightarrow$ $p(s) < s$
- pairs (s_i, DP_i) are strictly decreasing wrt lexicographic order of $(S,<)$ and $(2^P, \subset)$

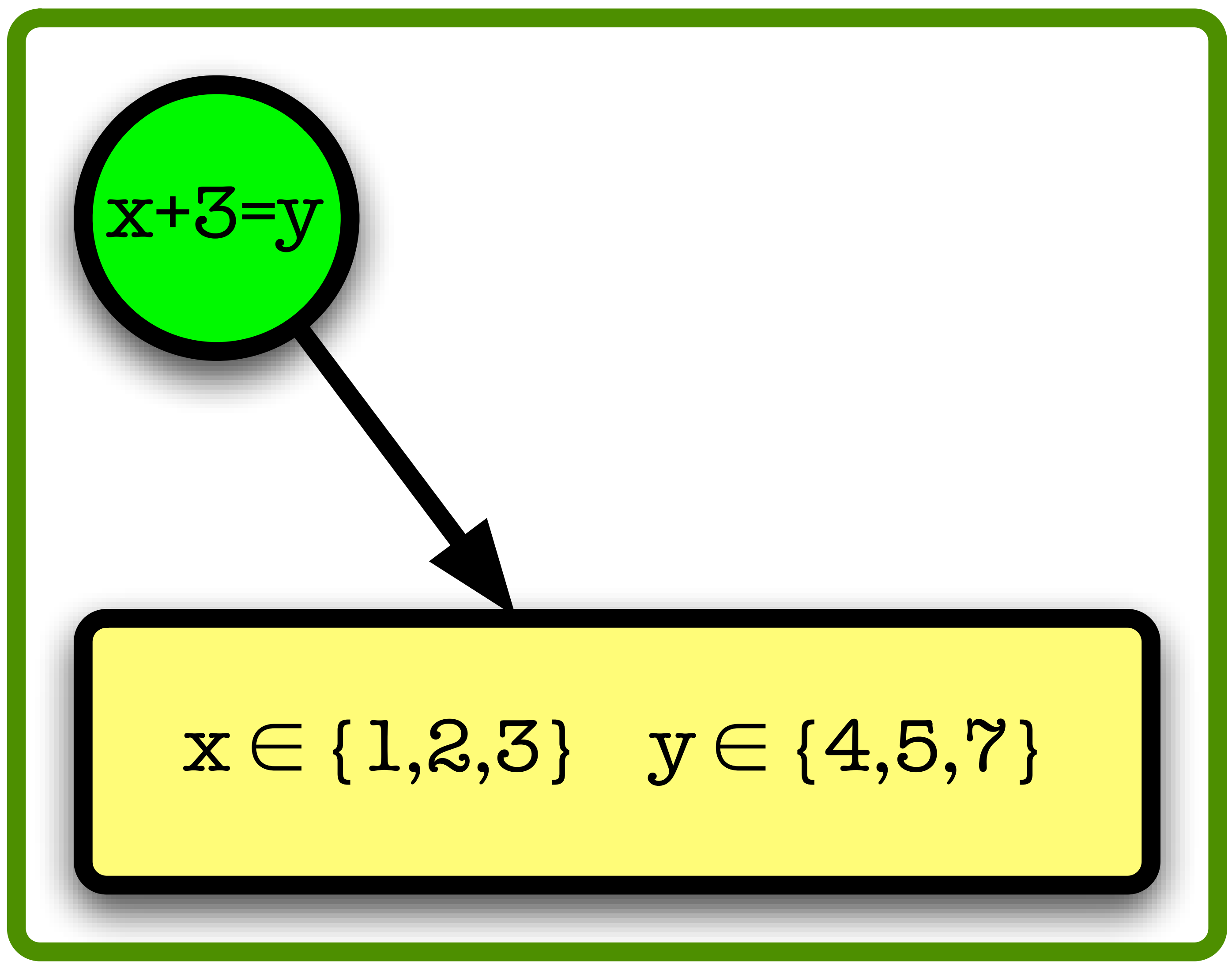
Optimisation 1: Subsumed Propagators

- We call a propagator p *subsumed* in a store s iff for all $s' \leq s$: $p(s') = s'$
- Do not add subsumed propagators to DP!

Example



Example



Optimisation 2: Idempotency

- A propagator p is *idempotent* iff for all stores s: p(p(s)) = p(s)
- If p is idempotent, do not add it to NP, even if it changed variables in its own scope

Idempotency

```
propagate(s0,P) =
  s := s0; DP:=P;
  while DP not empty do
    choose p∈DP;

    s′:= p(s); DP := DP-{p};
    MV := {x∈s | s(x) ≠ s′(x)};

    NP := {q∈P | ∃x∈scope(q): x∈MV};

    if p idempotent
      NP := NP \ {p};
    DP := DP ∪ NP;

    s := s′;
  return s;
```

Optimisation 2: Idempotency

- A propagator p is *idempotent* iff for all stores s: p(p(s)) = p(s)
- If p is idempotent, do not add it to NP, even if it changed variables in its own scope

Consistency Notions

- How strong can a propagator be?
- At most: remove all values that are not consistent with the constraint it implements:

$$\forall x \in X \ \forall d \in \mathrm{dom}(x) \ \exists \alpha \in \mathrm{ass}(X) : \ \alpha \in C \wedge \alpha(x) = d$$

- This is called *domain consistency*

Consistency Notions

- Often algorithmically difficult
- Weaker notion: bounds consistency

$$\forall x \in X \ \forall d \in \{\min \operatorname{dom}(x), \max \operatorname{dom}(x)\} \ \exists \alpha \in \text{ass}_{\text{bnd}}(X) :$$

$$\alpha \in C \wedge \alpha(x) = d$$

- $\text{ass}_{\text{bnd}}(X)$ is is the set of valuations α for X s.th. ∀ $(x{:}\text{D})$ in X, $\alpha(x) \in$ [min D, max D]

Consistency Notions

- Example: domain consistent propagator

$$eq_{x,y}(s) = \{\ x : s(x) \cap s(y),$$
$$y : s(x) \cap s(y)\ \}$$

Consistency Notions

- Example: bounds consistent propagator

$$leq_{x \leq y}(s) = \{\ x : \{n \in s(x) \mid n \leq max(s(y))\},$$

$$y : \{n \in s(y) \mid n \geq min(s(x))\}\ \}$$

Summary

- Propagators implement constraints
- Constraint store implements domains
- Propagators must be contracting, monotonic, correct, checking
- Propagators can be idempotent, subsumed
- Propagators can be bounds or domain consistent

This week's exercises

- Prove properties of the propagation loop
- Prove properties of some propagators
- Define bounds- and domain-consistent propagators

What’s up next week?

- Algorithmic aspects of propagation
- Global constraints (all distinct)