ID2204: Constraint Programming

Constraints: Modeling & Propagation

Lecture 07, 2009-04-21

Christian Schulte

cschulte@kth.se

Electronic, Computer and Software Systems
School of Information and Communication Technology
KTH – Royal Institute of Technology
Stockholm, Sweden

KTH Information and Communication Technology

Linear Equality

Linear Equality

- Propagator for

$$\sum_{i=1}^{n} a_i x_i = d$$

where a_i, d integers, $a_i \neq 0$

- How to propagate cheaply bounds information?
  - for each variable x_i consider how small and how large it possibly can be
  - restrict us here to $ax + by = d$

Floor and Ceiling

- $\lfloor x \rfloor$ (read: floor of *x*) is greatest integer *k* such that: $k \leq x$
  - example: $\lfloor 3.5 \rfloor = 3$
- $\lceil x \rceil$ (read: ceiling of x) is smallest integer *k* such that: $k \geq x$
  - example: $\lceil 3.5 \rceil = 4$

Propagating Linear Equality

- Rewrite for x

$$ax + by = d \quad \Leftrightarrow \quad ax = d - by$$
$$\Leftrightarrow \quad x = (d - by)/a$$

- Propagate

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

and

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

Propagating Linear Equality

- Computing

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

- If $a>0$ then

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

- If $a<0$ then

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

Propagating Linear Equality

- Computing ($a > 0$)

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

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

- If $b>0$, then

$$m = (d - b \times \min s(y))/a$$

- If $b<0$, then

$$m = (d - b \times \max s(y))/a$$

General Setup

- Repeat until fixpoint
  - propagate for each variable x_i
- Speed up: compute once

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

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

- Reuse by removing term for x_i
- Refer to propagator by $p_=$

Questions

- Is it necessary to perform several iterations?
  - yes, otherwise it is not idempotent
  - reason: non-unit coefficients
- What does $p_=$ compute?
  - is it bounds-consistent?

Example

- Example: $x = 3y + 5z$

  $s(x)=\{2..7\}$ $s(y)=\{0..2\}$ $s(z)=\{-1..2\}$

  propagator:

  $p_=(s)(\mathrm{x}) = \{\min_s(3y + 5z) \ldots \max_s(3y + 5z)\}$

- Resulting domain

  $s'(x)=\{\mathbf{2}..\mathbf{7}\}$ $s'(y)=\{0..2\}$ $s'(z)= \{0..1\}$

  - different from bounds propagation!
  - should be 3 and 6!

What Is Computed?

- Algorithm just considers existence of real solutions
  - bounds-consistency defined for integer solutions only
- Possible: introduce new notion
  - **R**-bounds consistency
  - Allow constraints to be defined by solutions over the reals

More Details

- Apt's book: Section 6.4
- Paper

  Schulte, Stuckey. *When Do Bounds and Domain Propagation Lead to the Same Search Space*. Transactions of Programming Languages and Systems, 2005.

Summary: Propagation Strength

- Propagators can have different propagation strengths
- Interesting classes
  - domain-consistent propagators
  - bounds-consistent propagators
- Typical propagators for linear arithmetic are ***not*** bounds-consistent
  - but close: not for integers, but for reals

Element Constraint

Constraints defined by extension

Modeling Price

- Suppose variable modeling location in warehouse
  - values model good to be stored at location
  - different goods have different prices
- How to propagate the price while the variable is not yet assigned a good?
- Very common: map variable to variable according to given values

Example

- Assume goods represented by numbers 0, 1, 2, 3
- Prices
  - good 0 price 10
  - good 1 price 15
  - good 2 price 5
  - good 3 price 12

Model by Reification

```
BoolVar b0(*this,0,1);
…
rel(*this, g, IRT_EQ,  0, b0);
rel(*this, p, IRT_EQ, 10, b0);
rel(*this, g, IRT_EQ,  1, b1);
rel(*this, p, IRT_EQ, 15, b1);
rel(*this, g, IRT_EQ,  2, b2);
rel(*this, p, IRT_EQ,  5, b2);
rel(*this, g, IRT_EQ,  3, b3);
rel(*this, p, IRT_EQ, 12, b3);
"b0 + b1 + b2 + b3 = 1";
```

- Tedious: several goods can have same price…
- Inefficient: too many propagators…
- Propagation: if propagators run to fixpoint, domain-consistency!

The Element Constraint

- Element constraint $a[[x]] = y$
  - array of integers *a*
  - variables *x* and *y*
  - value of *y* is value of *a* at *x*-th position
  - in particular: $0 \leq x <$ elements in *a*
- In Gecode
  - `element(*this, a, x, y);`
  - also for arrays of variables

Model with Element

```
IntArgs prices(4, 10,15,5,12);
element(*this, prices, g, p);
```

- Just single propagator!
- Okay, if same integer occurs multiply in array

Propagating Element

- We insist on domain-consistency
  - bounds-consistency too weak
- For $a[[x]] = y$ and store s propagate
  - if $j \in s(y)$ then keep all k from $s(x)$ with $j=a[k]$
  - if $k \in s(x)$ then keep all j from $s(y)$ with $j=a[k]$
  - remove all other values

Implementing Element…

- Fundamental requirement: new domains must be computed in order!
- Iterate over all elements $k \in s(x)$

  $$\{ a[k] \mid k \in s(x) \} \cap s(y)$$

- Iterate k from 0 to n-1:= width of a

  construct new domain for x

  - if $a[k] \in s(y)$ then keep k
  - requires intersection and sorting

Problems…

- Array in element constraints can be very large
  - always iterate over entire array
  - always sort (or maintain sorted data structure)
  - always compute intersection
- We can do better!

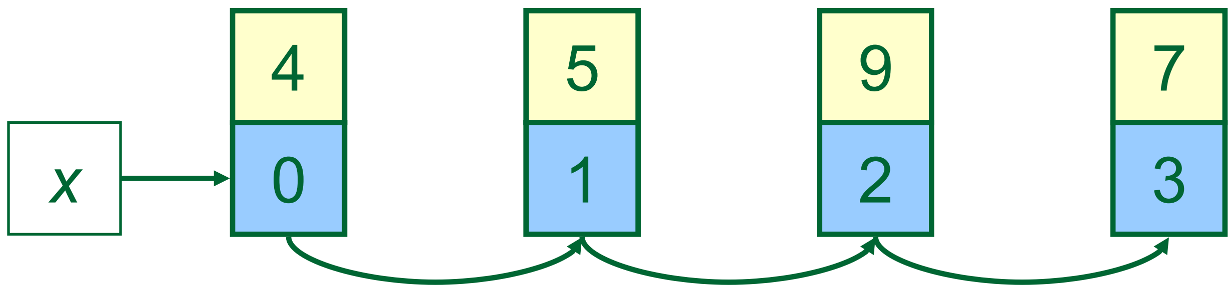
Running Example

- Consider $a[[x]] = y$ with
  - $a = (4,5,9,7)$
  - $s(x) = \{1,2,3\}$
  - $s(y) = \{2\ldots8\}$
- Propagation yields
  - $s(x) = \{1,3\}$
  - $s(y) = \{5,7\}$

Approach

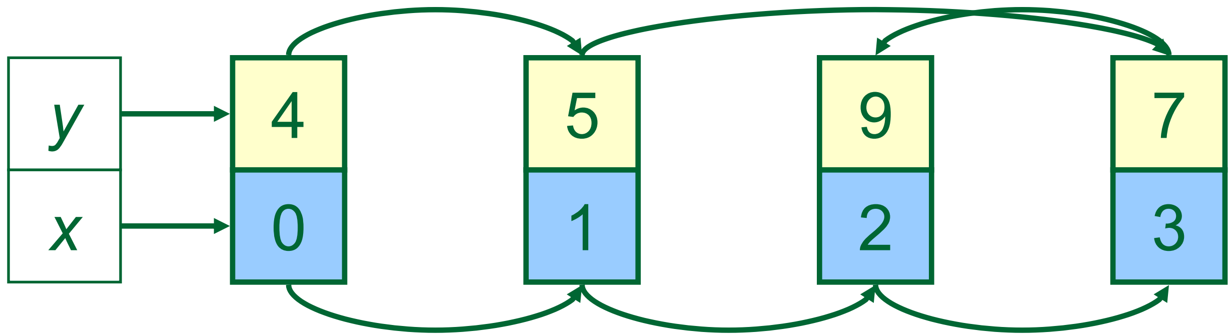
- Construct data structure
  - contains pairs of (*i*,*a*[*i*])
  - allows traversal for increasing *i*
  - allows traversal for increasing *a*[*i*]
  - allows removal of pairs (later)
- Data structure constructed initially

Datastructure Construction



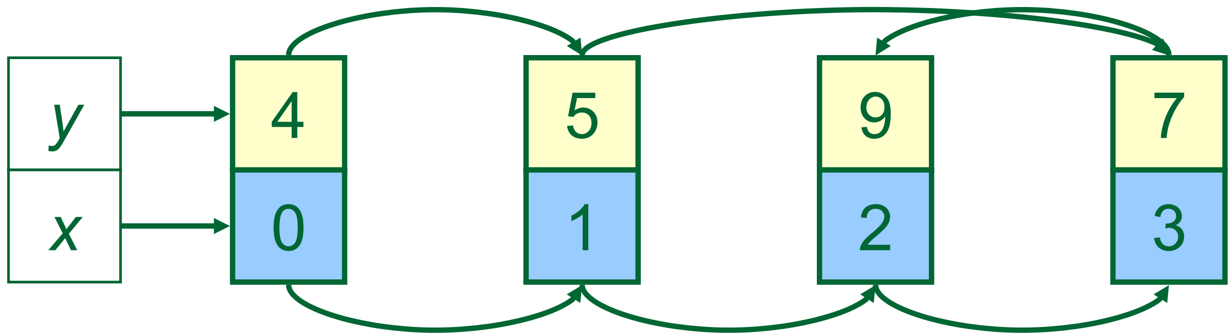
- Iterate over all *i* between 0 and 3
  - create node (*i*, *a*[*i*])
  - create links in order of creation (*x*-links)

Datastructure Construction



- Create links for *a*[*i*] values in increasing order (*y*-links)
  - sort and create links

Datastructure Invariant

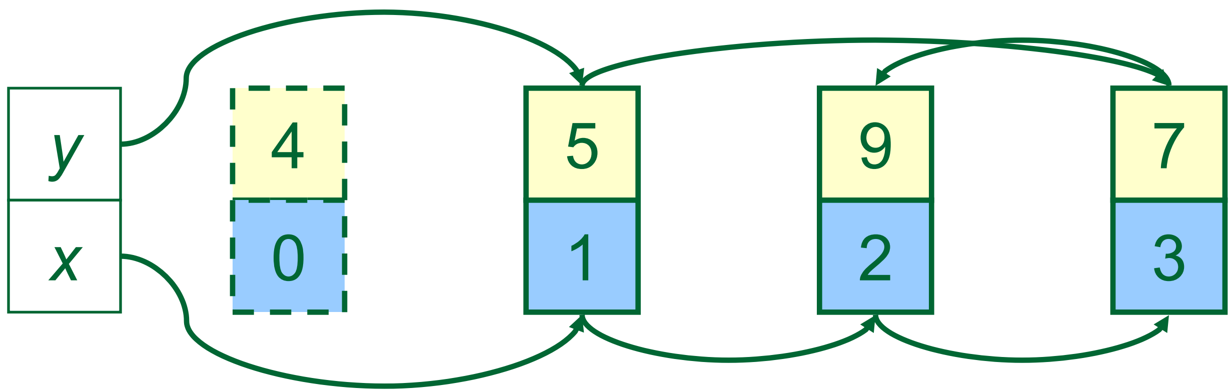


- Datastructure allows iteration
  - *i* values in order: follow *x*-links
  - *a*[*i*] values in order: follow *y*-links

Propagation

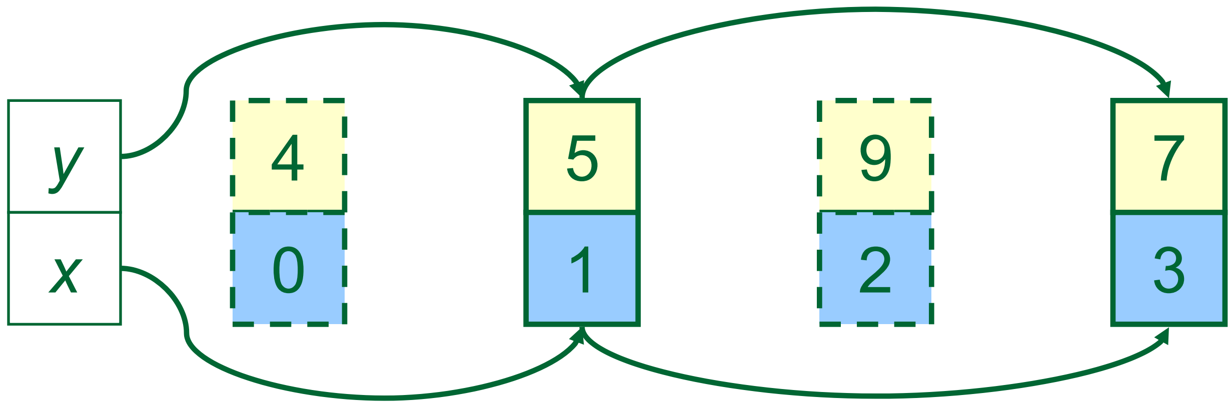
- Follow *x*-links and iterate values in *s*(*x*)
  - if value not in *s*(*x*), remove node
- Follow *y*-links and iterate values in *s*(*y*)
  - if value not in *s*(*y*), remove node
- Result: nodes with correct values remain for both *x* and *y*
  - in increasing order!

Follow *x*-links…



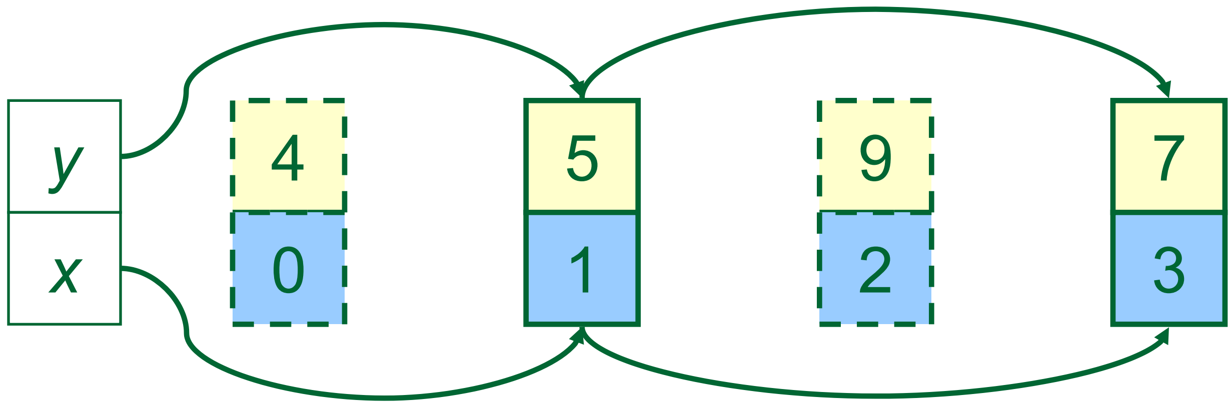
- Store $s(x) = \{1,2,3\}$
- Remove node for 0
  - by relinking
  - constant time: doubly-linked lists

Follow *y*-links…



- Store $s(y) = \{2,\dots,8\}$
- Remove node for 9
  - by relinking
  - constant time: doubly-linked lists

Read-off Variable Domains



- New store: $s(x) = \{1,3\}$, $s(y) = \{5,7\}$
- By just following respective links
  - are sorted
  - are smaller than original domains

Incremental Propagation

- One option: destroy data structure
- Better: keep data structure for next propagator invocation
  - propagators with state
  - our model: propagator rewriting
- Incremental propagation
  - construction only initially
  - sorting only initially
  - traversing never for full number of array elements

Summary: Element

- Element important constraint for mapping variables to values
  - cost functions
  - arbitrary constraints defined extensionally
- Important for propagation
  - maintain clever data structure
  - make propagation incremental