Constraint Programming Systems

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

Focus

- What are the key concepts in a
  - constraint-propagation based
  - tree-search based

  constraint programming system

- Focus: constraint propagation
  - basic model
  - properties and guarantees

- No complete story, see background material

Material

- Slides
  - will be available online (?)

- Background material
  - Efficient Constraint Propagation Engines.

    Christian Schulte, Peter J. Stuckey.

    Transactions on Programming Languages and Systems, pages 2:1-2:43. ACM Press, December, 2008.
  - Finite Domain Constraint Programming Systems.
    Christian Schulte, Mats Carlsson.

    Handbook of Constraint Programming, Foundations of Artificial Intelligence, pages 495-526. Elsevier Science Publishers, 2006.

Outline

- Model for propagator-based propagation
  - propagators and propagation loops
  - dependency directed propagation
  - what is computed
- Efficient propagation: a menu to choose from
  - fixpoint reasoning
  - event sets: static, monotonic, fully dynamic
  - which propagator to run next
  - combining propagation
  - variable-centered propagation

Constraint Satisfaction Problems

Specifications versus Implementations

- Specification
  - constraint satisfaction problem (CSP)
  - variables, values, constraints
  - semantics defined by its set of solutions

- Implementation
  - constraint model
  - variables, values, propagators
  - also defines set of solutions
  - constraint propagation and search for computing solutions

Essential Questions

- When does model implement CSP?
  - same set of solutions
- What are properties of propagators?
  - contract variable domains
  - can identify solutions
  - are monotonic

Constraint Propagation

- Given propagators with right properties
  - how to perform constraint propagation
  - what is computed
  - solutions are maintained
  - important invariant: order of execution irrelevant

Constraint Satisfaction Problems

- Here: constraint satisfaction problem (CSP) as problem specification
  - variables
  - which values do variables take
  - which constraints
- Specification: ***what*** are the solutions, ***not how*** to compute them
  - declarative specification

Parts of CSP

- Variables V

  finite set of variables $V=\{x_0, x_1, \ldots\}$

- Universe U

  finite set of values U

  - simplicity: all variables take values from same set

- Constraints C
  - which variables involved
  - what are the solutions

Constraints

- A constraint c is defined by

  its variables

  $$\mathrm{var}(c) = (x_1, \ldots, x_n) \in V^n$$

  its solutions

  $$\mathrm{sol}(c) \subseteq U^n = \underbrace{U \times \ldots \times U}_{\text{n times}}$$

Assignments

- ***Assignment*** *a* defines which values variables take

  $$a \in V \rightarrow U$$

- Assignment *a* ***solution of constraint*** *c* (written $a \in c$), iff

  $\mathrm{var}(c) = (x_1, \ldots, x_n)$ and

  $(a(x_1), \ldots, a(x_n)) \in \mathrm{sol}(c)$

Example: Assignments

- Suppose $V=\{x, y, z\}$ and $U=\{1, 2, 3\}$
- Then $a \in V \to U$ defined by

  $$a(x) = 2,\ a(y) = 3,\ a(z) = 1$$

  is assignment
- We will write

  $$a=\{x \to 2,\ y \to 3,\ z \to 1\}$$

Solutions of a CSP

- Assignment $a \in V \rightarrow U$ ***solution of CSP*** $P=(V,U,C)$ if

  $a \in c \qquad$ for all $c \in C$

- Solutions sol(P) of P defined

  $\{a \in V \rightarrow U \mid a \text{ solution of } P\}$

Example: CSP

- $PWD := (V, U, C)$ with
  - $V := \{x, y, z\}$
  - $U := \{1, 2, 3\}$
  - $C := \{c_1, c_2, c_3\}$ where

    $\mathrm{var}(c_1)=(x,y)$

    $\mathrm{sol}(c_1)=\{(1,2),(1,3),(2,1),(2,3),(3,1),(3,2)\}$

    $\mathrm{var}(c_2)=(x,z)$, $\mathrm{sol}(c_2):=\mathrm{sol}(c_1)$

    $\mathrm{var}(c_3)=(y,z)$, $\mathrm{sol}(c_3):=\mathrm{sol}(c_1)$

Example: CSP Solutions

- sol(PWD) = {

  {$x \rightarrow 1, y \rightarrow 2, z \rightarrow 3$},

  {$x \rightarrow 1, y \rightarrow 3, z \rightarrow 2$},

  {$x \rightarrow 2, y \rightarrow 1, z \rightarrow 3$},

  {$x \rightarrow 2, y \rightarrow 3, z \rightarrow 1$},

  {$x \rightarrow 3, y \rightarrow 1, z \rightarrow 2$},

  {$x \rightarrow 3, y \rightarrow 2, z \rightarrow 1$}}

Constraint Models

Constraint Model

- Gives an implementation of a CSP *P*
  - when is it really an implementation?
- Instead of constraints, we have propagators
  - what is a propagator?
  - propagators compute over a constraint store
  - what is a constraint store?

Constraint Stores

- ***Constraint store*** *s* maps variables to sets of values, that is

  $$s \in V \rightarrow 2^U$$

  - also store instead of constraint store
  - also known as domain
  - we refer to set of stores by $S = V \rightarrow 2^U$

Strength of Stores

- Store s_1 ***stronger*** than store s_2, iff

  $s_1(x) \subseteq s_2(x)$ for all $x \in V$
  - written $s_1 \leq s_2$

- Store s_1 ***strictly stronger*** than s_2, iff

  $s_1 \leq s_2$ and $s_1 \neq s_2$
  - written $s_1 < s_2$
  - equivalent: $s_1 \leq s_2$ and there exists $x \in V$ such that $s_1(x) \subset s_2(x)$

Example: Stores

- Suppose $V=\{x, y\}$ and $U=\{1, 2, 3\}$
- Consider

$$s_1 = \{x \rightarrow \{1,2\}, y \rightarrow \{2,3\}\}$$
$$s_2 = \{x \rightarrow \{2\}, y \rightarrow \{2,3\}\}$$
$$s_3 = \{x \rightarrow \{2,3\}, y \rightarrow \{1,2,3\}\}$$

Then

$$s_2 < s_1 \quad \text{and} \quad s_2 < s_3$$

but neither

$$s_3 \leq s_1 \quad \text{nor} \quad s_1 \leq s_3$$

Stores

- (S,<) is well-founded order!
  - only finitely many variables
  - only finitely many values

Propagator Properties

- Clearly a propagator must compute stronger stores
  - sometimes will fail to make it strictly stronger
- Propagator p is function from stores to stores ($p \in S \rightarrow S$) which is contracting
  - $p(s) \leq s$ for all stores s

Intuition: Propagators Implement Constraints

- Assume constraint *c* and propagator *p*

- Require: if *p* "implements" *c, p* never removes solution of *c*
  - this is not sufficient as we will see
  - we need connection between assignments and stores
    - propagators compute with stores
    - solutions are assignments

Assignments and Stores

- We write $a \in s$ for an assignment a and a store s, if

  $$a(x) \in s(x) \qquad \text{for all } x \in V$$

- Propagators are defined on stores, for assignment a, define

  $$\text{store}(a)(x) = \{a(x)\} \qquad \text{for all } x \in V$$

  - store(a) is a store
  - $a \in s \Leftrightarrow \text{store}(a) \leq s$

Example: Assignments and Stores

- Suppose $V=\{x, y, z\}$ and $U=\{1, 2, 3\}$

  and

  $$a = \{x \rightarrow 2, y \rightarrow 3, z \rightarrow 1\}$$

  Then

  $$\text{store}(a) = \{x \rightarrow \{2\}, y \rightarrow \{3\}, z \rightarrow \{1\}\}$$

Example: Propagator

- Assume $V=\{x,y\}$ and $U=\{0, \ldots, 5\}$

- Propagator $p_{\leq}$ for $x \leq y$

  $p_{\leq}(s) =$

  $\{ x \rightarrow \{ n \in s(x) \mid n \leq \max(s(y)) \},$

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

Example: Propagator

- For store

  $$s = \{\, x \rightarrow \{3,4,5\},\ y \rightarrow \{0,1,2,3\}\}$$

  propagator $p_{\leq}$ returns

  $$p_{\leq}\,(s) = \{\, x \rightarrow \{\, n \in s(x) \mid n \leq 3 \,\},$$

  $$y \rightarrow \{\, n \in s(y) \mid n \geq 3 \,\}\}$$

  $$= \{\, x \rightarrow \{3\},\ y \rightarrow \{3\}\}$$

Implementing a Constraint

- *p* implements *c*, if

  $a \in c$, then $p(\mathrm{store}(a))=\mathrm{store}(a)$

  - *p* respects the solutions of *c*
  - with other words: solutions are fixpoints

- Is this sufficient?

  No!

Keeping Solutions: Sketch…

- Assume p implements c, and $a \in c$
- Required: if $a \in s$, then $a \in p(s)$

$$
\begin{aligned}
a \in s \quad & \Leftrightarrow \text{store}(a) \leq s \\
& \Rightarrow \text{????} \\
& \Leftrightarrow \text{store}(a) \leq p(s) \\
& \Leftrightarrow a \in p(s)
\end{aligned}
$$

Example: No Propagator

- Assume propagator

  $p_{?}(s) = \textbf{if } s(x)=\{1,2,3\} \textbf{ then } \{x\rightarrow\{1\}\}$
  $\qquad\qquad \textbf{else } s$

  and

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

  Then

  $s_1 > s_2$ but $p_{?}(s_1) < p_{?}(s_2)$

  - makes propagation order dependent
  - must be ruled out!

Propagators Are Monotonic!

- Propagator $p \in S \rightarrow S$ is
  - contracting $p(s) \leq s$
  - monotonic $s_1 \leq s_2 \Rightarrow p(s_1) \leq p(s_2)$

Keeping Solutions: Again…

- Assume p implements c, and $a \in c$

  $p(\text{store}(a)) = \text{store}(a)$
- Required: if $a \in s$, then $a \in p(s)$

  $a \in s \quad \Leftrightarrow \text{store}(a) \leq s$

  $\Rightarrow p(\text{store}(a)) \leq p(s)$

  monotonicity

  $\Leftrightarrow \text{store}(a) \leq p(s)$

  $\Leftrightarrow a \in p(s)$

Handling Failure…

- Store s is ***failed***, if exists $x \in V$

  $$s(x) = \varnothing$$

- Propagator p ***fails on store*** s, if

  $$p(s) \text{ failed}$$

Non-Solution Assignments

- Assume p implements c, $a \notin c$ then p fails on store(a)=s

  a∉c $\Leftrightarrow$ $p(s) \neq s$

  $\Rightarrow$ $p(s) < s$

  - Remember: p is contracting!

  $\Rightarrow$ ex. $x \in V$ $p(s)(x) \subset s(x)$

  $\Rightarrow$ ex. $x \in V$ $p(s)(x) = \emptyset$

  $\Rightarrow$ p fails on s

Order Does Not Matter

$$\begin{tikzcd} & s_1 \arrow[r, "q"] & s_2 \\ s_0 \arrow[ur, "p"] \arrow[dr, "q"'] & & \\ & s_3 \arrow[r, "p"] & s_4 \end{tikzcd}$$

- Assume propagation done
  - $p(s_2) = q(s_2) = s_2$ and $p(s_4) = q(s_4) = s_4$
- Then $s_2 = s_4$
  - $s_0 \le p(s_0)=s_1 \Rightarrow s_3=q(s_0) \le q(s_1)=s_2$
    $\Rightarrow s_4=p(s_3) \le p(s_2)=s_2 \Rightarrow s_4 \le s_2$
  - $s_0 \le q(s_0)=s_3 \Rightarrow s_1=p(s_0) \le p(s_3)=s_4$
    $\Rightarrow s_2=q(s_1) \le \mathrm{q}(s_4)=s_4 \Rightarrow s_2 \le s_4$

Constraint Model

- A constraint model $M=(V,U,P)$ is defined by
  - set of variables V
  - set of values U
  - set of propagators P

Solutions

- Solutions sol(p) of propagator p is defined as

  $$\{\, a \in V \rightarrow U \mid \text{store}(a) = p(\text{store}(a)) \,\}$$

- Solutions sol(M) of constraint model $M=(P,V,U)$ is defined as

  $$\{\, a \in V \rightarrow U \mid a \in \text{sol}(p) \text{ for all } p \in P \,\}$$

Model Implementation

- A constraint model $M=(V,U,P)$ ***implements*** the CSP C, if

  $$\mathrm{sol}(M) = \mathrm{sol}(C)$$

Solutions Refined

- We will be interested in solutions starting propagation from some store

  $\mathrm{sol}(M,s)$ for model $M=(V,U,P)$

  store s

  defined as

  $$\{\, a \in \mathrm{sol}(M) \mid \mathrm{store}(a) \leq s \,\}$$

Soundness of Propagation

- Given model M=(V,U,P) and store s

  for all $p \in P$

  $$\text{sol}(M,s) = \text{sol}(M,p(s))$$

  - follows from previous discussion of monotonicity as propagator property

- Slogan: propagation is solution preserving

Naïve Constraint Propagation

Naïve Constraint Propagation

- Looking for

  propagate : $M \times S \rightarrow S$

  performing constraint propagation
  - start from some initial store
  - return store on which all propagation has been performed
  - ignore efficiency, focus on principle idea

Naïve Propagation Function

```
propagate((V,U,P), s)
    while p∈P and p(s) ≠ s do
        s := p(s);
    return s;
```

- What is returned as result?
- Does it terminate?

Result Computed

- Assume `propagate(`(V,U,P)`,`s`)`$=s'$

  $\mathrm{sol}((V,U,P),s) = \mathrm{sol}((V,U,P),s')$

  no solutions removed

  for all $p \in P$: $p(s') = s'$

  no further propagation possible

  largest simultaneous fixpoint

Termination

- Consider store s_i at i-th iteration of loop with s_o initial store

  $$s_{i+1} < s_i$$

- That is, s_i form strictly decreasing sequence: cannot be infinite
  - remember: (S,<) is well-founded!
- Loop terminates!

Weakest Simultaneous Fixpoint

- Assume `propagate(`(V,U,P)`,`s`)`$=s'$
  Then
  s' weakest sim. fixpoint with $s' \leq s$
  that is
  for all $p \in P$ $p(s') = s'$
  - clear, follows from termination of loop

  weakest fixpoint?
  - any other fixpoint is stronger

Weakest Fixpoint

- Let p_i be propagator of i-th iteration

  $s_i := p_i(s_{i-1}) \qquad i > 0$

  where $s_0 := s$
- Termination: there is n such that

  $s' = s_n$
- Assume t is ssim. fp. with $t \leq s$, show

  $t \leq s'$
  - that is, t is indeed stronger and hence s' is weakest

Proof: Base Case

- Show by induction over i

  $t \leq s_i$ for all $i \geq 0$

  from this: $t \leq s_n = s'$
- Base case $i = 0$

  holds, as we assume $t \leq s_0$
- Induction step $i \Rightarrow i + 1$

  …

Proof: Induction Step

- Induction step $\quad i \Rightarrow i + 1$

  $t \leq s_i$

  $\Rightarrow \quad p_{i+1}(t) \leq p_{i+1}(s_i)$

  $\qquad\qquad p_{i+1}$ monotonic

  $\Rightarrow \quad t = p_{i+1}(t) \leq p_{i+1}(s_i)$

  $\qquad\qquad t$ is fixpoint of p_{i+1}

  $\Rightarrow \quad t \leq p_{i+1}(s_i) = s_{i+1}$

  $\qquad\qquad$ definition of s_i

  $\Rightarrow \quad t \leq s_{i+1}$

Why Naïve?

- Always searches all propagators for propagator which can contract
  - maintain propagators which are known to have fixpoint computed
  - might have to find out by having propagators which do no contraction
  - take variables into account which connect two propagators

Realistic Propagation

Improving Propagation

- Idea: propagator narrows domain of some (few) variables
  - re-propagate only propagators sharing variables
- Maintain a set of "dirty" propagators
  - not known whether fixpoint
  - all other propagators have fixpoint computed

Propagator Variables

- Variables var(*p*) of propagator *p*
  - variables of interest
- No input considered on other variables
- No output computed on other variables

Variable Dependencies

- No output on other variables
  for all $s \in S$, for all $x \in (V\text{-var}(p))$
  $$p(s)(x)=s(x)$$
- No input from other variables
  for all $s_1, s_2 \in S$
  if (for all $x \in \text{var}(p)$: $s_1(x)=s_2(x)$),
  then (for all $x \in \text{var}(p)$:
  $$p(s_1)(x)=p(s_2)(x))$$

Propagation Loop

```
propagate((V,U,P), s_0)
    s := s_0; N := P;
    while N ≠ ∅ do
        choose p ∈ N;
        s' := p(s); N := N – {p};
        MV := { x ∈ V | s(x) ≠ s'(x) };
        DP := { q ∈ P | exists x ∈ var(q): x ∈ MV };
        N := N ∪ DP;
        s := s';
    return s;
```

Questions

- What does it compute
  - does it compute simultaneous fixpoint?
  - the largest?
  - important: loop invariant
- Termination?
  - stores are not any longer strictly stronger

Loop Invariant

- Loop maintains

  for all $p \in P\text{-}N \quad \Rightarrow \quad p(s) = s$

  after termination ($N = \emptyset$):

  for all $p \in P \quad \Rightarrow \quad p(s) = s$

- Obligations
  - holds initially
  - is actually invariant

Invariant Obligations

- Holds initially
  - trivially, as $P\text{-}N=\emptyset$ (N initialized to P)
- Is invariant

$$I := \text{for all } p \in P\text{-}N \quad \Rightarrow \quad p(s) = s$$

  - if $s' = p(s)$ ⇨ okay to remove from N
  - otherwise
    - no guarantee that s is fixpoint for $p \in DP$ ⇨ move them to N
    - if $p \in P\text{-}DP$, no need move to N (def of var(p))

What Is Computed

- Fixpoint follows from loop invariant
- Largest simultaneous fixpoint as for naïve propagation
  - proofs works exactly as before
  - sequence of stores not strictly decreasing
  - sufficient: store sequence and decreasing and finite (to prove next)

Termination

- Insight:
  - if $MV=\emptyset$, then p removed from N
  - if $MV\neq\emptyset$, then $p(s)<s$

- Consider pairs (s_i,N_i) with
  - s_i the value of s at i-th iteration
  - N_i the value of N at i-th iteration

  strictly decreasing wrt well-founded lexicographic order of $(S,<)$ and $(2^P,\subset)$

Fixpoint Reasoning

General Idea

- Essential: knowledge on fixpoint for a propagator
- So far: only implicit knowledge
- Here: make knowledge explicit
  - propagators provide information

We Are Done! What Now?

- Suppose the following propagator

  $$p(s) = \{x \rightarrow (s(x) \cap \{1,2,3\})\}$$

  - implements domain constraint $x \in \{1,2,3\}$
- After executing p once, no further execution needed:

  if $s' \leq p(s)$ then $p(s')=s'$
- We can safely delete p from model
  - otherwise, pointless re-execution!

Subsumed Propagators

- Propagator *p* ***subsumed*** by store *s*, iff

  for all $s' \leq s : p(s')=s'$

  - all stronger stores are fixpoints
  - *p* entailed by *s*
  - *s* subsumes *p* (*s* entails *p*)

Reminder: Propagator for $\leq$

- Propagator $p_{\leq}$ for $x \leq y$

  $p_{\leq}(s) =$

  $\{ x \rightarrow \{ n \in s(x) \mid n \leq \max(s(y)) \},$

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

We Are Done! What Next?

- After executing $p_\leq$ on store s we have

  $$p_\leq(p_\leq(s)) = p_\leq(s)$$

  - max(s(y)) does not change!
  - min(s(x)) does not change!
- What happens: as var($p_\leq$)={x,y}, $p_\leq$ is added to *DP*
  - but: s' is fixpoint for $p_\leq$
  - no need to include in *DP*

First Attempt: Idempotent Functions

- A function $f \in X \rightarrow X$ is ***idempotent***, if

  for all $x \in X$: $f(f(x)) = f(x)$

- Very strong property for a propagator: required for all stores!

Falling Into Domain Holes

- Consider propagator p for $x = y + 1$

  $p(s) =$

  $\{ x \rightarrow \{ n \in s(x) \mid \min s(y)+1 \leq n \leq \max s(y)+1\},$

  $\;\; y \rightarrow \{ n \in s(y) \mid \min s(x)-1 \leq n \leq \max s(x)-1\}\}$

- Not idempotent, consider

  $$s = \{ x \rightarrow \{0,4,5,6\}, y \rightarrow \{2,3,4,5\}\}$$

- But idempotent if $s(x)$ and $s(y)$ are ranges (have no holes)

Second Attempt: Dynamic Idempotence

- A function $f \in X \rightarrow X$ is ***idempotent on*** $x \in X$ if

  $$f(f(x)) = f(x)$$

  - statement on just one element

- For a propagator: if p is idempotent on s, it does not mean that p is idempotent on s' with $s' \leq s$

How to Find Out?

- Given store *s* and propagator *p*
- Does *s* subsume *p*?
  - try all *s*’ < *s*: way to costly
- Is *p* idempotent on *s*?
  - apply *p* to *s*: that is what we tried
    to avoid in the 1st place

Status Messages

- Solution: propagator returns status and tells result

propagator p is function

$$p \in S \rightarrow SM \times S$$

with

$$SM := \{\text{fix}, \text{nofix}, \text{subsumed}\}$$

Propagator with Status

- Assume propagator *p* and store *s*

  if *p*(*s*) = (fix, *s*'), then

  s' is fixpoint for *p*

  if *p*(*s*) = (subsumed, *s*'), then

  s' subsumes *p*

  if *p*(*s*) = (nofix, *s*'), then

  no further knowledge

  always safe (as before)

Propagator for ≤ with Subsumption

- Propagator $p_\leq$ for $x \leq y$

  $p_\leq(s) =$

  if $\max(s(x)) \leq \min(s(y))$ **then**

  (subsumed, s)

  else

  (fix,

  $\{\ x \rightarrow \{\ n \in s(x) \mid n \leq \max(s(y))\ \},$

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

What to Return?

- Propagation function now also needs to return the set of propagators
  - in case of subsumption, propagators are removed

Improved Propagation

```
propagate((V,U,P), s0)
    s := s0; N := P;
    while N ≠ ∅ do
        choose p ∈ N;
        (ms,s'):= p(s); N := N – {p};
        if ms=subsumed then P := P –{p}; end
        MV := { x ∈V | s(x) ≠ s'(x) };
        DP := { q ∈ P | exists x ∈ var(q): x ∈ MV };
        if ms=fix then DP := DP –{p}; end
        N := N ∪ DP;
        s := s';
    return (P,s);
```

Correctness

- Are the optimizations correct?
- How to prove:
  - invariant is still invariant
  - solutions remain the same
  - still computes the same

argument: fixpoints!

Fixpoint Reasoning Experiments

- Relative to no fixpoint reasoning

	time	steps
static	-2.9%	-12.7%
dynamic	-6.1%	-15.9%

- Reduction in steps does not directly translate to time:
  - steps avoided are cheap (perform no propagation)

Propagation Events

Propagation Events

- Many propagators
  - simple to decide whether still at fixpoint for changed domain
  - based on ***how*** domain has changed
- How domain changes described by

  propagation event

  or just event

Propagator for $\leq$

- Propagator $p_{\leq}$ for $x \leq y$

  $p_{\leq}(s) =$

  $\{\, x \rightarrow \{\, n \in s(x) \mid n \leq \max(s(y)) \,\},$

  $\;\; y \rightarrow \{\, n \in s(y) \mid n \geq \min(s(x)) \,\}\}$

  - must be propagated only if max(s(y)) or min(s(x)) changes

Propagator for $\neq$

- Propagator $p_{\neq}$ for $x \neq y$

  $p_{\neq}(s) =$

  $\{ x \rightarrow s(x) - \mathrm{single}(s(y)),$

  $y \rightarrow s(y) - \mathrm{single}(s(x))\}$

  - where: $\mathrm{single}(\{n\}) = \{n\}$
    $\mathrm{single}(N) = \varnothing$ (otherwise)

  - must be propagated only if x or y become assigned

Events

- Typical events
  - fix(*x*) *x* becomes assigned
  - min(*x*) minimum of *x* changes
  - max(*x*) maximum of *x* changes
  - any(*x*) domain of *x* changes
- Clearly overlap
  - fix(*x*) occurs: min(*x*) or max(*x*) occur
    any(*x*) occurs
  - min(*x*) or max(*x*) occur: any(*x*) occurs

Events on Store Change

events(s,s') =

$\{\ \mathrm{any}(x)\ \mid s'(x) \subset s(x)\ \} \cup$

$\{\ \min(x)\ \mid \min s'(x) > \min s(x)\ \} \cup$

$\{\ \max(x) \mid \max s'(x) < \max s(x)\ \} \cup$

$\{\ \mathrm{fix}(x)\ \mid |s'(x)|=1 \text{ and } |s(x)|>1\ \}$

- where $s' \leq s$

Events: Example

- Given stores
  - $s = \{\ x_1 \rightarrow \{1,2,3\},\quad x_2 \rightarrow \{3,4,5,6\},$
    $x_3 \rightarrow \{0,1\},\quad x_4 \rightarrow \{7,8,10\}\}$
  - $s' = \{\ x_1 \rightarrow \{1,2\},\quad x_2 \rightarrow \{3,5,6\},$
    $x_3 \rightarrow \{1\},\quad x_4 \rightarrow \{7,8,10\}\}$
- Then events(s,s') =

  { max(x_1), any(x_1),
  any(x_2),
  fix(x_3), min(x_3), any(x_3)}

Events are Monotonic

- If $s'' \le s'$ and $s' \le s$ then

  $$\text{events}(s,s'') = \text{events}(s,s') \cup \text{events}(s',s'')$$

- Event occurs on change from s to s''
  - occurs on change from s to s', or
  - occurs on change from s' to s''

Event Sets: First Requirement

- Event set for propagator p: es(p)
  - for all stores s with $p(p(s)) \neq p(s)$:

$$\mathrm{es}(p) \cap \mathrm{events}(s, p(s)) \neq \emptyset$$

  - captures propagation by p
  - if propagator does not compute fixpoint on store s, then events from s to $p(s)$ must be included in es(p)

  - does not occur for idempotent propagators

Event Sets: Second Requirement

- Event set for propagator *p*: es(*p*)
  - for all stores s_1 and s_2 with $s_2 \leq s_1$
    if $p(s_1)=s_1$ and $p(s_2)\neq s_2$ then
    $\mathrm{es}(p) \cap \mathrm{events}(s_1,s_2) \neq \emptyset$
  - captures propagation by other propagators
  - if store s_1 is fixpoint and changes to non-fixpoint s_2, then events from s_1 to s_2 must be included in es(*p*)

Propagator for $\leq$

- Propagator $p_{\leq}$ for $x \leq y$

  $p_{\leq}(s) =$

  $\{\, x \rightarrow \{\, n \in s(x) \mid n \leq \max(s(y)) \,\},$

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

  - good one: $es(p_{\leq}) = \{ \max(y), \min(x) \}$
  - but also: $es(p_{\leq}) = \{ \text{any}(y), \text{any}(x) \}$

Propagator for $\neq$

- Propagator $p_{\neq}$ for $x \neq y$

  $p_{\neq}(s) =$

  $\{ x \rightarrow s(x) - \text{single}(s(y)),$

  $y \rightarrow s(y) - \text{single}(s(x))\}$

  - where: single($\{n\}$) $= \{n\}$
    single(N) $= \emptyset$ (otherwise)

  - good one: es($p_{\neq}$) = { fix(y), fix(x) }
  - but also: es($p_{\neq}$) = { any(y), any(x)}

Taking Advantage from Event Sets

- Base decision of propagators to re-propagate on event sets rather than on modified variables

$$DP := \{\, q \in P \mid \text{events}(s,s') \cap \text{es}(q) \neq \emptyset \,\};$$

Event Granularity

- Not all event types must be supported
- Many systems collapse min and max to bnd
- Tradeoff between time and space
    - per event type: memory for each variable needed

Event Set Experiments: Time & Steps

- Relative to no events

	time	steps
fix, any	-7.8%	-24.1%
with bnd	-7.8%	-27.8%
with min, max	-6.3%	-27.7%

- Depends on overhead of propagator execution

Event Set Experiments: Memory

- Relative to no events

	memory
fix, any	+3.9%
with bnd	+9.9%
with min, max	+15.5%

Monotonic and Dynamic Event Sets

Changing Event Sets

- Like dynamic fixpoint reasoning, also have changing event sets
  - monotonic: event sets become smaller for stronger stores
  - fully dynamic: event sets change arbitrarily
- How to guarantee that propagation still works?

Minimum Propagator

- Propagate such that

  $\{ x \rightarrow \{ n \in s(x) \mid \min(\min s(y), \min s(z)) \leq n \leq \max(\max s(y), \max s(z)) \},$

  $y \rightarrow \{ n \in s(y) \mid \min s(x) \leq n \},$

  $z \rightarrow \{ n \in s(z) \mid \min s(x) \leq n \}\}$

- Static event set

  $\{ \min(x), \min(y), \max(y), \min(z), \max(z)\}$

Minimum Propagator

- Assume store s with
  $s(x) = \{1,2,3\}$ and $s(z)=\{5,6,7\}$
- Idea: make es dependent on the store
- For minimum:
  es(min,s) = { min(x), min(y), max(y)}

Monotonic Event Sets: First Requirement

- Event set for propagator p in context of store s: es(p,s)
  - for all stores s' **with $s' \leq s$** and $p(p(s')) \neq p(s')$:

    $$\text{es}(p,s) \cap \text{events}(s', p(s')) \neq \emptyset$$

  - if propagator does not compute fixpoint on store s' (stronger than s), then events from s' to $p(s')$ must be included in es(p,s)

Monotonic Event Sets: Second Requirement

- Event set for propagator p in context of store s: es(p,s)
  - for all stores s_1 and s_2 with $s_2 \leq s_1$ **and $s_1 \leq s$**

    if $p(s_1)=s_1$ and $p(s_2)\neq s_2$ then

    $$\text{es}(p,s) \cap \text{events}(s_1,s_2) \neq \emptyset$$

  - captures propagation by other propagators
  - if store s_1 is fixpoint and changes to non-fixpoint s_2, then events from s_1 to s_2 must be included in es(p)

Full Dynamic Event Sets

- Event set can be made fully dynamic
    - prevents any form of idempotence

Fully Dynamic Event Sets for Minimum

- Assume store s_1
  - $s_1(x) = \{0 .. 10\}$, $s_1(y)=\{0 .. 15\}$, $s_1(z)=\{5 .. 10\}$
  - is fixpoint of minimum propagator
  - es(min,s_1) = { min(x), min(y), max(y), max(z)}
- Assume store s_2 with $s_2 \leq s_1$
  - $s_2(x) = \{5 .. 9\}$, $s_2(y)=\{6 .. 9\}$, $s_1(z)=\{5 .. 10\}$
  - also fixpoint
  - es(min,s_2) = { min(x), max(y), min(z), max(z)}

Watched Literals

- Fully dynamic event sets are related to watched literals
- Watched literals for unit propagation in SAT
  - consider clause for propagation only if one of two watched literals becomes false
- Introduced to CP by Minion [Gent ea, 2006]

Dynamic Event Set Experiments

- Relative to static event sets for examples using dynamic event sets

	time	steps	memory
monotonic	-39.5%	-31.8%	-19.1%
fully dynamic	-41.1%	-39.3%	-19.7%

Propagator Selection

Which one to run next

Pending Propagators

- How to implement choose
- Possibilities
  - immediately: stack
  - as late as possible: queue
  - decide on cost: priorities (cost)

Queue ⇆ Stack

- Stack shows pathological behavior in some cases
  - can increase runtime by 3 orders
  - in average: almost twice the runtime

- Pathological behavior
  - cheap, expensive global, cheap, expensive global, …
  - can that fixed by cost: no (jumping ahead)

Propagator Costs/Priorities

- Define cost metric
  - unary, binary, ternary, linear, quadratic, cubic, crazy
  - fine metric: low and high variants
  - coarse metric: collapse some cost values
- Organize according to cost
  - one queue for each cost category
  - pick always from cheapest queue first

Using Costs

- Impact of cost metric on runtime
  - fine -7.4%
  - medium -6.3%
  - coarse -5.6%

- Variations
  - fine + stack +23.9%
  - inverted +107.5%

Using Costs

- Number of propagators executed increases
  - from 3.8% to 7.0%
- Reason: iterated fixpoints
  - cheap fixpoint
  - single more expensive propagator
  - cheap fixpoint
  - …

Importance

- Protection from pathological behavior
- Efficiency (to a lesser extent)
- Quite complicated: minimize number of cost computations (due to dynamic cost)
- Enables more optimizations (later)

Combining Propagators

Combining Filter Algorithms

- Consider alldifferent(x)
  - naïve		variable becomes assigned
    remove value from other variables
    cheap
  - domain		find and prune Hall sets [Régin, 1994]
    expensive

- Common approach
  - first naïve, then domain
  - applicable to many global constraints
  - but how?

Possible Decisions

- Nothing
  - only do expensive but strong
- Immediate
  - do cheap immediately followed by expensive
- Multiple propagators
  - create propagators for cheap and expensive
  - with according costs
  - other cheap propagators before expensive

Better: Staging

- Single propagator [Schulte & Stuckey, 2004, 2008]
  - idle and must be run: set stage one
  - stage one: do cheap
set stage two
  - stage two: do expensive
set idle
- Optimizations
  - stage one finds stage two not needed: idle
  - more stages (possibly)

Comparison

- Relative to nothing: time memory
  - immediate -1.6% 0.0%
  - multiple -4.8% +3.0%
  - staging -6.5% 0.0%
- Examples with costly global constraints
  - immediate just around -2%
  - staging often -16% up to -40% time

Summary

Constraint Programming Systems

- It is not about how they are implemented in detail…
  - varies with each individual system
- It is about what model they implement
  - models capture most common aspects in systems
- Focus on model here
  - efficient, propagator-centered constraint propagation