# Constraint Programming Systems

Third International Summer School of the Association of Constraint Programming


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


## Constraint Programming Systems

- Offer *reusable* software components for
  - constraint propagation
  - combining constraints (combinators)
  - search
    - branching (labeling)
    - exploration (for example: depth-first, LDS, ...)
- Services provided
  - abstractions to implement new components
  - environment for integrating components
  - libraries of commonly used components

[Henz & Müller 00]



## Focus

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

  constraint programming system
- Stress constraint propagation
  - basic model
  - properties and guarantees
- No complete story, see background material



## Learning Outcomes

- Be able to name and explain the key concepts
  - describe what is computed
  - describe how it is computed
  - describe how it is computed efficiently
- Optional: how can concepts be used to implement a constraint



## Material

- Slides
  - will be available online
- Background
  - Finite Domain Constraint Programming Systems. Christian Schulte, Mats Carlsson.

    Handbook of Constraint Programming, Foundations of Artificial Intelligence, pages 495-526. Elsevier Science Publishers, 2006.
  - Efficient Constraint Propagation Engines. Christian Schulte, Peter J. Stuckey. (experiments taken from this paper)

    CoRR entry, arXiv:cs/0611009v1 [cs.AI], 2006.
  - Gecode/J software [as sent by email]



## Voluntary Assignment

- Few questions
- Tasks to implement
  - a model as warm up
  - a constraint

## Outline: Parts

- Part I: Model for propagator-based propagation
  - propagators and propagation loops
  - dependency directed propagation
  - what is computed
- Part II: Implementation overview
  - propagation and search
- Part III: Efficient propagation
  - fixpoint reasoning
  - event sets: static, monotonic, fully dynamic
  - which propagator to run next
  - combining propagation
  - variable-centered propagation



# Part I
## A Model for Propagator-based 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 $a \in V \rightarrow U$ defined by
  $a(x) = 2, a(y) = 3, a(z) = 1$
  is assignment
- We will write
  $a=\{x \rightarrow 2, y \rightarrow 3, z \rightarrow 1\}$



## Solutions of a CSP

- Assignment $a \in V \rightarrow U$ ***solution of CSP*** $P=(V,U,C)$ if
  $a \in c$ for all $c \in C$
- Solutions $\mathrm{sol}(P)$ 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

- $\mathrm{sol}(PWD) = \{$
  - $\{x\to1,y\to2,z\to3\}$,
  - $\{x\to1,y\to3,z\to2\}$,
  - $\{x\to2,y\to1,z\to3\}$,
  - $\{x\to2,y\to3,z\to1\}$,
  - $\{x\to3,y\to1,z\to2\}$,
  - $\{x\to3,y\to2,z\to1\}\}$



# 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

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



## 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

  $s_2 < s_1$ and $s_2 < s_3$

  but neither

  $s_3 \leq s_1$ nor $s_1 \leq s_3$



## 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

  $a(x) \in s(x)$ for all $x \in V$
- Propagators are defined on stores, for assignment $a$, define

  $\text{store}(a)(x) = \{a(x)\}$ 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 $p(\text{store}(a))=\text{store}(a)$
  - $p$ 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 $a \in p(s)$

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

  $\Rightarrow$ ????

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

  $\Leftrightarrow a \in p(s)$



## Example: No Propagator

- Assume propagator

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

  **else** $s$

  and

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

  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 $a \in p(s)$

  $a \in s \;\Leftrightarrow\; \text{store}(a) \leq s$
  $\Rightarrow\; p(\text{store}(a)) \leq p(s)$
  ***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
  $p(s)$ failed



## Non-Solution Assignments

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

  $a \notin c \quad \Leftrightarrow\; p(s) \neq s$
  $\Rightarrow\; p(s) < s$
  - Remember: $p$ is contracting!

  $\Rightarrow$ ex. $x \in V \;\; p(s)(x) \subset s(x)$
  $\Rightarrow$ ex. $x \in V \;\; p(s)(x) = \varnothing$
  $\Rightarrow$ $p$ 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 $s_2 = s_4$
  - $s_0 \leq p(s_0)=s_1 \;\Rightarrow\; s_3=q(s_0) \leq q(s_1)=s_2$
    $\Rightarrow\; s_4=p(s_3) \leq p(s_2)=s_2 \quad \Rightarrow\; s_4 \leq s_2$
  - $s_0 \leq q(s_0)=s_3 \;\Rightarrow\; s_1=p(s_0) \leq p(s_3)=s_4$
    $\Rightarrow\; s_2=q(s_1) \leq q(s_4)=s_4 \quad \Rightarrow\; s_2 \leq s_4$



## Implementation Granularity

- No one-to-one correspondence between propagator and constraint
  - one propagator implements many constraints
    - example: alldifferent for $O(n^2)$ disequalities
  - many propagators implement one constraint
    - example: $O(n^2)$ disequalities for alldifferent
- Solution: state when propagators implement a CSP via set of solutions



## 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

  sol($M$) = sol($C$)



## Solutions Refined

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

  sol($M$,$s$) for model $M=(V,U,P)$, store $s$

  defined as

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



## Soundness of Propagation

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

  for all $p \in P$

  sol($M$,$s$) = 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)`=$s'$

  sol((V,U,P),s) = 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 $s_0$ 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
  $s_i := p_i(s_{i-1})$ $i > 0$
  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 $i \Rightarrow i + 1$

  $t \le s_i$

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

  $p_{i+1}$ monotonic

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

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

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

  definition of $s_i$

  $\Rightarrow t \le s_{i+1}$



## 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), s0)
   s := s0; 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 $p \in P\text{-}N \Rightarrow p(s) = s$
  after termination ($N = \emptyset$):
  for all $p \in P \Rightarrow p(s) = s$
- Obligations
  - holds initially
  - is actually invariant



## Invariant Obligations

- Holds initially
  - trivially, as $P\text{-}N=\emptyset$ ($N$ initialized to $P$)
- Is invariant
  $I$ := for all $p \in P\text{-}N \Rightarrow p(s) = s$
  - if $s' = p(s) \Rightarrow$ okay to remove from $N$
  - otherwise
    - no guarantee that $s$ is fixpoint for $p \in DP \Rightarrow$ 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 $(2^P,\subset)$

## Connections

## Extensional versus Intentional

- Constraints from CSP: extensional
  - all assignments as extension
- Propagators from model: intensional
  - characterize solutions
  - can profit from structure in constraints
  - "global constraints": dedicated algorithms for propagators, alldifferent, etc…
- Similar to logics
  - formulae are intensional
  - models are extensional



## Extensional Propagation

- Assume AC for binary constraints
- Consider $c$ with var($c$)=($x$,$y$)
  - keep
    
    $n \in s(x)$
    
    only if
    
    $(n,m) \in$ sol(c) and $m \in s(y)$
  - similar for $y$



## Arc Consistency Propagators

- Define ac-propagators for $x$ and $y$
  - for each constraint $c$ with var($c$)=($x$,$y$)
  - taken from [Apt 2003]
- ac($c$,$x$)($s$)($z$) :=
  **if** $z \neq x$ **then** $s(z)$ **else**
  { $n \in s(x)$ | exists $(n,m) \in$ sol($c$): $m \in s(y)$ }
  **end**
- ac($c$,$y$)($s$)($z$) :=
  **if** $z \neq y$ **then** $s(z)$ **else**
  { $m \in s(y)$ | exists $(n,m) \in$ sol($c$): $n \in s(x)$ }
  **end**



## Arc Consistency Model

- Assume a CSP **C**=($V$,$U$,$C$). Then the AC-model ($V$,$U$,$P$) for **C** is defined as
  
  $P$={ ac($c$,$x$),ac($c$,$y$) | $c \in C$, var($c$)=($x$,$y$) }
- AC-model **AC** for a CSP **M** implements **M**:
  
  sol(**AC**) = sol(**M**)
  - proof is straightforward, just applications of definitions, rests on how ac-propagators work on assignments



## Arc Consistency

- Given constraint $c$ with var($c$)={$x$,$y$} and store $s$
  
  $s$ arc consistent for $c$ ⇔
  
  for all $n \in s(x)$ exists $m \in s(y)$
  
  $(n,m) \in$ sol($c$)
  
  and
  
  for all $m \in s(y)$ exists $n \in s(x)$
  
  $(n,m) \in$ sol($c$)

## Arc Consistent CSP

- Store $s$ is arc consistent for a CSP $(V,U,C)$ ⇔

  for each $c \in C$: $s$ arc consistent for $c$



## Propagating **AC**

- Assume AC-model **AC** for a CSP **M**
- Then for store $s$

  if propagate(**AC**,$s$)=$s'$

  then $s'$ arc consistent for **M**
  - Proof: straightforward, if not arc consistent, propagator not at fixpoint



## Propagating GAC

- Algorithms for propagating extensionally defined constraints also available for the general $n$-ary case
  - propagate generalized arc consistency
  - see for example: [Bessière & Régin, 1997], [Bessière ea, 2001], [Lecoutre & Szymanek, 2006]



## Consistency Level Computed

- Model is generic
- Consistency level defined by each individual propagator
  - accurate way of characterization [Maher 02]
- Supports many different consistency levels
  - propagator for domain-consistent alldifferent
  - propagator for bound-consistent alldifferent
  - propagator for value-consistent alldifferent



## Store Approximations

- Here: all elements from $2^U$
- Might be unrealistic:
  - finite sets, graphs, real intervals, …
- Store (domain) approximation:
  - only from some subset of $2^U$
  - must be closed under intersection, must contain singletons (unit approximations), …
  - [Benhamou, 1996]



## Generic Iteration Algorithms

- Propagation as presented here is an instance of a generic iteration algorithm
- For a more general treatment, see [Apt, 2003]

## Part II
## Implementation

Propagation and search

## Implementing Propagation

## Stores and Propagators

- Propagators perform destructive update on single store
  - no functions returning new stores
  - interaction with search: restore on backtrack
- Updates must comply to properties in model
  - contracting
  - monotonic
- Propagators have state
  - store var(*p*) and data structures for propagation
  - for example: domain-consistent alldifferent stores variable-value graph



## Major Design Decisions

- Implementing *N*
  - queue: first in – first out
  - stack: last in – first out
  - priority queue
  - to be discussed later
- Implementing *MV* and *DP*
  - variable-centered representation

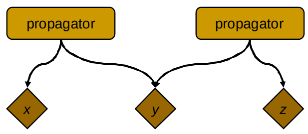


## Implementing *MV* and *DP*

- Variable-centered approach
  - each variable *x* knows dependent propagators
  - typically organized as list (*suspension list*)
  - propagator *p* included in list of *x* ⇔ $x \in$ var(*p*)
- Upon propagator creation
  - propagator subscribes to its variables
  - becomes runnable

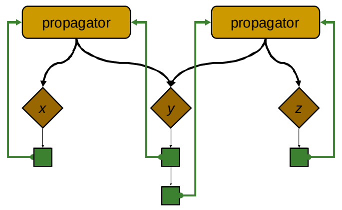


## Propagators ⇨ Variables




- **Propagators** know their **variables** (that is, var(*p*))
  - to perform store modifications
  - passed as parameters to propagator creation

## Variables ⇨ Propagators




- **Variables** know dependent **propagators**
  - to perform efficient computation of dependent propagators
  - implemented by **suspension lists**



## Modifying a Variable

- Traverse suspension list
  - add propagators to *N*
- Optimization
  - mark runnable propagators
  - that is: propagators already in *N*
- Multiple variable modification by propagator
  - explicitly maintain *MV* (as in model)
  - only after propagator execution: process *MV*
  - suspension list traversed only once per variable



# Search

## Branching and Exploration

- Branching: defines shape of search tree
  - labeling, branching, distribution, …
  - often based on heuristics
- Exploration: explore nodes of search tree
  - often fixed to be depth-first
  - many aspects
    - optimization (branch-and-bound)
    - development tools
    - parallelism



## Branching

- Requires synchronization on fixpoint
  - for implementing dynamic variable orderings
  - by construction: Prolog, ILOG Solver, …
  - explicit synchronization in concurrent setup: Oz
- Programmed
  - from builtin-search: Prolog-based
  - special (language) constructs: ILOG Solver, Oz
- Typically, rich library available



## Exploration

- All systems support
  - search for first solution
  - search for some/all solutions
  - search for best solution
- Most systems support
  - LDS and some variants

## Exploration Strategy

- Often fixed to be depth-first
- Sometimes can be programmed
  - Gecode: spaces ("nodes") as ADT for exploration
    - exploration programmed from operations
    - for example: copy node in search tree; access solution
  - ILOG Solver: control exploration by limits and priorities
    - limit: cut-off branches
    - priorities: which node to explore next

[Schulte 97] [Perron 99]



## Infrastructure for Exploration

- State restauration
  - backtrack to a *previous* state
- Approaches
  - trailing: recording and undoing changes
  - copying: put complete state aside
  - recomputation: recompute state on need
- By far dominating approach: trailing



## Trailing

- Trailing stores undo and redo information
  - interleaved with constraint propagation
  - uses trail data structure
  - update: put ⟨location,content⟩
  - undo: write location ← content
  - every choice point: put mark or record top of trail
- Requires
  - all updates trail-aware
  - for example: domain change, change of suspension list, …



## Time Stamping

- Problem: multiple change of same location
  - for example: multiple narrowing of domain
  - only original value needs restauration
  - intermediate values not needed
- Solution: local time stamp on modified entity
  - new choice point: increase global time stamp
  - upon modification: trail, if local stamp earlier; update local stamp

[Aggoun & Beldiceanu 90] [Aggoun & Beldiceanu 91]



## Multiple Value Trail

- Modifying *n* successive locations
  - record start, number (*n*) and *n* locations on trail
  - instead of 2*n* individual entries

[Aggoun & Beldiceanu 90] [Aggoun & Beldiceanu 91]



## Copying And Recomputation

- Copying
  - operations ignorant of state restauration
  - support for concurrency and parallelism
  - alone infeasible: excessive memory requirements
- Hybrid strategies: copying and recomputation
  - adaptive: create copy on demand to speed up future recomputation
  - batch: speed up recomputation by avoiding repeated fixpoint computation, related idea: decomposition-based search (entirely based on efficient recomputation)
  - competitive with trailing

[Schulte 1999] [Choi & Henz & Ng 2001] [Schulte 2002] [Michel & Van Hentenryck 2004]

## Part III
### Efficient Constraint Propagation

## 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 ≤

- 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 $x = y + 1$
  $p(s)$ =
  { $x \rightarrow$ { $n \in s(x)$ | min $s(y)+1 \leq n \leq$ max $s(y)+1$},
  $y \rightarrow$ { $n \in s(y)$ | 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 1st place



## Status Messages

- Solution: propagator returns status and tells result
  propagator $p$ is function
  $p \in S \rightarrow SM \times S$
  with
  $SM$ := {fix, nofix, 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($s(x)$) ≤ 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 ≤

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

  $p_{\leq}(s) =$

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

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

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



## Propagator for ≠

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

  $p_{\neq}(s) =$

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

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

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

    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($s$,$s'$) =

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

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

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

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

- where $s' \leq s$

## Events: Example

- Given stores
  - $s$ = { $x_1 \to \{1,2,3\}$, $x_2 \to \{3,4,5,6\}$, $x_3 \to \{0,1\}$, $x_4 \to \{7,8,10\}$}
  - $s'$ = { $x_1 \to \{1,2\}$, $x_2 \to \{3,5,6\}$, $x_3 \to \{1\}$, $x_4 \to \{7,8,10\}$}
- 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 $s'' \leq s'$ and $s' \leq s$ then

  events($s$,$s''$) =

  events($s$,$s'$) $\cup$ 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 $p(p(s)) \neq p(s)$:

    es($p$) $\cap$ 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

    es($p$) $\cap$ 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 ≤

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

  $p_{\leq}(s)$ =

  { $x \to \{ n \in s(x) \mid n \leq \max(s(y)) \}$,
  $y \to \{ n \in s(y) \mid n \geq \min(s(x)) \}$}
  - good one: es($p_{\leq}$) = { max($y$), min($x$) }
  - but also: es($p_{\leq}$) = { any($y$), any($x$)}



## Propagator for ≠

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

  $p_{\neq}(s)$ =

  { $x \to s(x)$ - single($s(y)$),
  $y \to s(y)$ - 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

  $DP$ := { $q \in P$ | events($s$,$s'$) $\cap$ 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 $p(p(s')) \neq p(s')$:
    es$(p,s) \cap$ events$(s',p(s')) \neq \varnothing$
  - 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
    es$(p,s) \cap$ events$(s_1,s_2) \neq \varnothing$
  - 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]
  - idle and must be run: set stage one
  - stage one: do cheap<br>set stage two
  - stage two: do expensive<br>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



# Variable-centered Propagation

## Variable- and Propagator-centered Propagation

- Model discussed here: propagator-centered
  - propagation loop controlled by set of propagators
  - used in, for example: B-Prolog, CHIP, Eclipse, Mozart, SICStus, Gecode
- Alternative: variable-centered
  - propagation loop controlled by set of modified variables
  - set not empty: still modification to be propagated
  - used in, for example: Choco, ILOG Solver



## Variable-centered Propagation

```
propagate((V,U,P), s)
   X := V;
   while X ≠ ∅ do
      choose x ∈ X;
      X := X - {x};
      foreach p with x ∈ var(p) do
         s' := p(s);
         MV := { x ∈ V | s(x) ≠ s'(x) };
         X := X ∪ MV;
         s := s';
   return s;
```



## Variable-centered Propagation

- Fixpoint reasoning
  - subsumption: easy and similar
  - idempotence: not directly possible…
- Using events: as for propagator-centered
- Using priorities/cost
  - possible but not straightforward
  - only per variable change

## Variable-centered Propagation

- Advantage: knowledge about changed variables available
  - maintain not only variables but also which values have been removed
- Finding which variable has changed
  - important: propagation with sublinear time
  - when propagator *p* is run
  - variable-centered O(1)
  - propagator-centered O(*n*) (|var(*p*)| = *n*)



## Propagator-centered Propagation

- Folklore approach: have demons attached to variables (similar to variables)
  - execute demon each time variable changes
  - demon has access to propagator's state
  - changes can be recorded in propagator's state



## 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



## References

Partial, for full references see additional material.

## References

[Aggoun & Beldiceanu 90]
A. Aggoun, N. Beldiceanu. *Time Stamps Techniques for the Trailed Data in Constraint Logic Programming Systems*, Actes du Séminaire 1990 de programmation en Logique, Trégastel, France, 1990.

[Aggoun & Beldiceanu 91]
A. Aggoun, N. Beldiceanu. *Overview of the CHIP Compiler System*, ICLP 1991.

[Apt, 2003]
Krzysztof R. Apt, Principles of Constraint Programming, Cambridge University Press, 2003.

[Benhamou, 1996]
Frédéric Benhamou. Heterogeneous Constraint Solving, ALP '96, LNCS 1139, Springer.

## References

[Bessière & Régin, 1997]
Christian Bessière and Jean-Charles Régin. Arc Consistency for General Constraint Networks: Preliminary Results, IJCAI 1997.

[Bessière ea, 2001]
Christian Bessière, Jean-Charles Régin, Roland H. C. Yap, and Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm, Artificial Intelligence, 165(2), 2005.

[Choi & Henz & Ng 01]
C.W. Choi, M. Henz, K.B. Ng. *Components for State Restoration in Tree Search*, CP 2001.

[Gent ea, 2006]
Ian P. Gent, Chris Jefferson, and Ian Miguel. Watched Literals for Constraint Propagation in Minion, CP 2006.





## References

[Henz & Müller, 2000]
M. Henz, T. Müller. An Overview of Finite Domain Constraint Programming. Proc. APORS, 2000.

[Lecoutre & Szymanek, 2006]
Christophe Lecoutre and Radoslaw Szymanek. Generalized Arc Consistency for Positive Table Constraints, CP 2006.

[Maher, 2002]
M. Maher. *Propagation Completeness of Reactive Constraints*. Proc. ICLP, 2002.

[Michel & Van Hentenryck, 2004]
Laurent Michel and Pascal Van Hentenryck. A decomposition-based implementation of search strategies, ACM Transactions on Computational Logic, 5(2), 2004.





## References

[Perron, 1999]
L. Perron. Search Procedures and Parallelism in Constraint Programming, CP 1999.

[Schulte, 1999]
C. Schulte. Comparing Copying and Trailing, ICLP 1999.