

Introduction to constraint programming

Lecture V

Per Kreuger piak@sics.se &
Markus Bohlin bola@sics.se

December 2002

Constraint programming — CP & CLP

Constraint programming integrates general CSP solving and specialised techniques into a practically useful framework

- CP was first developed in the logic programming community as extensions of existing logic programming (Prolog) systems
- Distinction between constraint programming (CP) and constraint logic programming (CLP)
 - CLP is constraints integrated in a logic programming framework e.g: CHIP, SICStus Prolog, Eclipse Prolog, GNU Prolog
 - CP takes main ideas of CLP and implements them as library to some specialised or existing language and development system e.g: ILOG Solver & CHIP Library (C++), JSolver (Java), Choco (Claire)

Difference between general CSP solving and CP

- CP systems are heterogeneous
 - No single algorithm is used to achieve some particular form of *global consistency* (but *arc-B* is often *locally* maintained)
 - Each constraint is implemented as a specialised propagation algorithm (*propagator*) for the variables of *that* constraint
 - Propagators are co-routines interacting through shared variables
- CP systems are *programming systems*
 - Individual constraints are used as modelling primitives
 - CP integrate specialised techniques as constraint abstractions
 - Other programming constructs also influence overall model

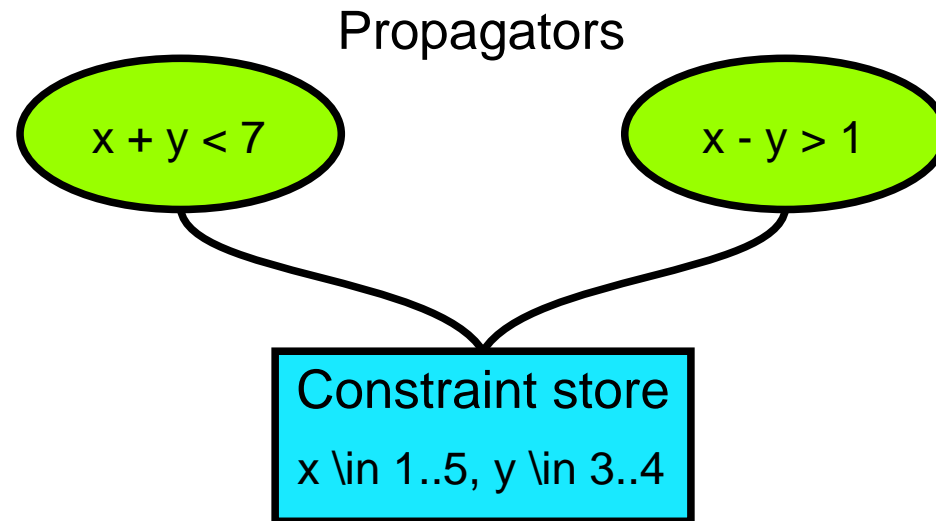
Bounds (interval) consistency

An arc $\{x_1, \dots, x_k\} \subseteq Z$ in a constraint graph of a CSP $\langle Z, D, C \rangle$ and its corresponding constraint $C_{\langle x_1, \dots, x_k \rangle}$ is *arc-B-consistent* if and only if, for each variable x_i and the bounds \underline{x}_i and \overline{x}_i on its domain $D_{x_i} = [\underline{x}_i.. \overline{x}_i]$ there exists values $v_{x_j}, v'_{x_j} \in D_{x_j}$ for each $j \leq k$ such that both $\{\langle x_1, v_{x_1} \rangle, \dots, \langle x_i, \underline{x}_i \rangle, \dots, \langle x_k, v_{x_k} \rangle\}$ and $\{\langle x_1, v'_{x_1} \rangle, \dots, \langle x_i, \overline{x}_i \rangle, \dots, \langle x_k, v'_{x_k} \rangle\}$ satisfy $C_{\langle x_1, \dots, x_k \rangle}$.

A CSP is *arc-B-consistent* if all its constraints are arc-consistent.

- Arc-B-consistency can be defined on CSPs that have domains that can be represented as numerical intervals $[a..b]$, e.g. finite domains

Constraint store and propagators



- What can be said about this constraint system?

Components of a CP system

Variables with an associated domain (usually *finite domains*)

Propagators implements the propagation of individual constraints

Propagation loop schedules the propagators of each constraint until a fixpoint (no further propagation) is reached

Search_procedure(s) Interleaves speculative domain reduction with propagation

A host language API, data structures and iterators

Propagators

- A propagator p is a procedure that
 - implements domain reduction for the variables $var(p)$ associated with its constraint $con(p)$
- Execution of a propagator p
 - reduces the domains of $var(p)$ if possible
 - signals failure if some domain becomes empty
 - may signal entailment
 - * i.e. that further reduction cannot not influence validity of $con(p)$

Classes of constraints

Basic, primitive and global constraints plus metaconstraints

- Basic constraints
 - For which the solver is complete
 - Stored in “constraint store”
 - $x \in D$, $x = v$ and in some cases $x = y$ (variable aliasing)
- Primitive constraints
 - Cannot be meaningfully decomposed
 - Implemented as propagators
 - $x < y$, $x \neq y$, $x + y = z$, ...

- Global constraints
 - Subsumes a set of basic and or primitive constraints
 - Encode complex or high level modeling concepts
 - Implement semantics of corresponding set of primitive constraints efficiently
 - May give (significantly) stronger propagation than corresponding set of primitive constraints
- Metaconstraints
 - Reifies the truth of a constraint into the value of a boolean value
$$x + y \leq 15 \Leftrightarrow a = 1, x + y \geq 25 \Leftrightarrow b = 1$$
 - Used to do boolean reasoning over constraints, e.g:
$$a + b = 1 \text{ imply that } x + y \notin 16..24$$

Propagator properties

- A propagator p must be:

Correct: No solution of $con(p)$ is removed by domain reduction of variables $ivar(p)$

Assignment complete: guaranteed to signal failure for an assignment of all variables in $var(p)$ that is invalid w.r.t. $con(p)$

Contracting: $p(D_{\bar{x}}) \leq D_{\bar{x}}$

Monotonic: $D_{\bar{x}} \leq D_{\bar{y}} \rightarrow p(D_{\bar{x}}) \leq p(D_{\bar{y}})$

where \leq represents pairwise \subseteq over vectors $\bar{\cdot}$

- and may be

Idempotent: $p(p(D_{\bar{x}})) = p(D_{\bar{x}})$

Propagation loop

- The propagation loop terminates when a *fixpoint* is reached
 - I.e. when no propagators can narrow domains further
 - Termination is guaranteed since domains are finite and propagators are contracting
- If all propagators are correct and monotonic this will be the *largest* fixpoint
 - I.e. no solutions are excluded

States of propagators and the propagation loop

- A *propagator* is either:

Fix: Cannot reduce domains further

Runnable: Can possibly reduce domains further

- The *propagation loop* maintains the following propagator sets

Prop: all propagators

Run: all runnable propagators

– Initially $Run = Prop$

Propagation loop

Algorithm 1 Propagation loop

procedure *LOOP*(*Prop*, *D*)

Run \leftarrow *Prop*

while *Run* $\neq \emptyset$ **do**

Run \leftarrow *Run* \setminus {*p*}

D' \leftarrow *p*(*D*)

if *D'* = *FAIL* **then**

return *D'*

else

Run \leftarrow *Run* \cup {*q* | *x* \in *var*(*q*) \wedge *x* \in {*y* | *D'*_{*y*} < *D*_{*y*}}

D \leftarrow *D'*

end if

end while

return *D*

Notes on the propagation loop

- p gets rescheduled until it produces no change
 - if p is idempotent that is unnecessary
- Level of consistency is *heterogeneous* and *local* since it depends on the (operational) semantics of individual propagators
 - Several propagators may exist for a given constraint e.g:
 - Domain consistency** AC or stronger
 - Bounds consistency** Check only bounds of each (numerical) domain
 - Different levels of consistency may be enforced for different variables of a single constraint

Implementation choices

Run as stack or (priority) queue

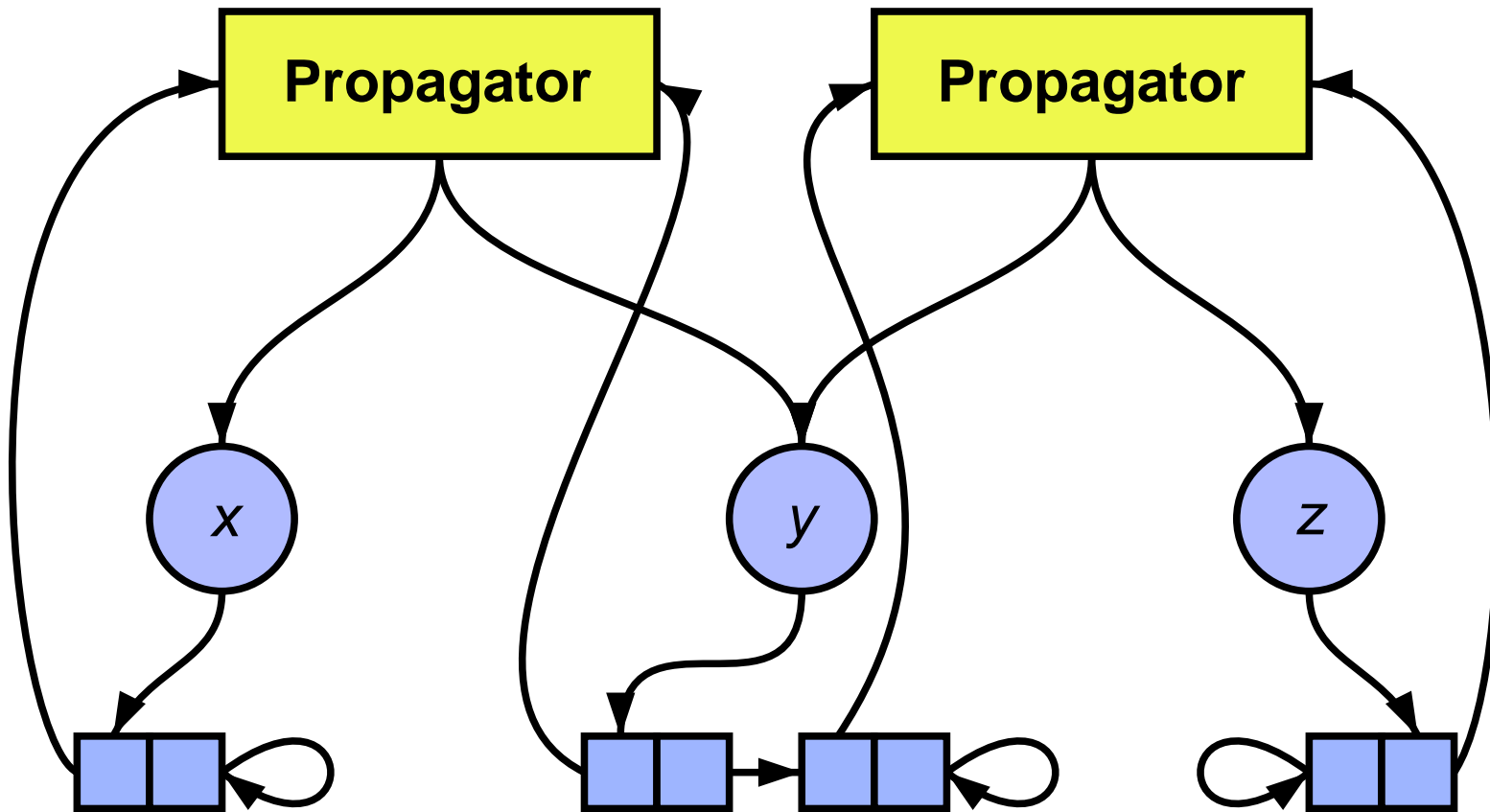
- Stack can cause starvation and thrashing

Variable-Propagator referencing How to find the propagators depending on a variable

- Each variable x is associated with a suspension-list $prop(x)$ of propagators that depend on it

$$p \in prop(x) \Leftrightarrow x \in var(p)$$

Propagator & variable datastructure



Search

- Search in CP is similar to that of CSP i.e.
 - Variable order (first fail, most constrained)
 - Value order
 - Depth or Breadth first or various hybrids, e.g. Recomputation, LDS
 - Parallel search also possible
- Labelling generalised to speculatively adding constraints, e.g:
 - Domain splitting

Optimisation — branch and bound

- Not identical to method used OR!
- Reflect value of cost function into a *cost variable*
- Important that constraints on this value really reduce search tree
- Easiest if cost function is linear
- Depends heavily on the propagation behaviour of constraints on variables that are parameters to cost function

Outline of algorithm

1. Find an arbitrary solution and make it current
2. Record the cost or value of current solution by constraining the domain of the cost variable to be “better” than current solution
3. Search for new solution under this constraint
4. If such a solution exists, make it current and go to 2
if not, terminate with current solution

Alternatives to optimisation

- Enforce hard constraints on (local) parameters to cost function
- Use cost for heuristic rather than as optimisation tool e.g:
 - Use search order that tend to give solutions with low cost early in search tree
 - Use limited discrepancy search
- Use hybrid methods combining methods from local search or integer programming with propagation — “Cost based filtering”

Global constraints

- Global Constraints (GC) is one of the main trends in today's CP-research and practice
- Algorithms that can be implemented as incremental filtering mechanisms integrate well into the general CP framework
- Many such algorithms from matching theory, flow optimisation and graph algorithms have proved to be most effective as specialised modelling components in such frameworks
- Modern CP systems are more or less complete collections of such algorithms made available as high level constraint abstractions

The following is a selection of constraints available in SICStus Prolog

Global arithmetic constraints¹

sum(+Xs,+RelOp,?Value) where X_s is a list of integers or domain variables, $RelOp$ is relational symbol, and $Value$ is an integer or a domain variable. True iff

$$X_1 + \cdots + X_n \text{ RelOp } Value$$

scalar_product(+Coeffs,+Xa,+RelOp,?Value) where $Coeffs$ is a list of length n of integers, X_s is a list of length n of integers or domain variables, $RelOp$ is a relational symbol as above, and $Value$ is an integer or a domain variable. True if

$$Coeff_1 X_1 + \cdots + Coeff_n X_n \text{ RelOp } Value$$

¹Useful for e.g. cost functions

Counting constraints²

global_cardinality(+Vars,+Vals) where $Vars$ is a list of integers or domain variables, and $Vals$ is a list of terms $V - K$, where V is an integer and K is a domain variable or an integer. Each V must be unique. *True* iff every element of $Vars$ is equal to some V and for each pair $V - K$, exactly K elements of $Vars$ are equal to V .

If either $Vars$ or $Vals$ is ground, and in many other special cases, *global_cardinality/2* maintains domain-consistency, but generally, interval-consistency cannot be guaranteed.

²Useful in complex assignment problems

Selection & assignment

element(?X,+List,?Y) where X and Y are integers or domain variables and $List$ is a list of integers or domain variables. *True* if the X :th element of $List$ is Y . *element/3* maintains domain-consistency in X and interval-consistency in $List$ and Y .

all_distinct(+Variables) where $Variables$ is a list of domain variables with bounded domains or integers. Each variable is constrained to take a value that is *unique* among the variables. Several variants.

assignment(+Xs,+Ys) where Xs and Ys are lists of domain variables or integers, both of length n . True iff

$$(\forall ij) X_i, X_j \in [1..n] \wedge X_i = j \Leftrightarrow Y_j = i$$

Scheduling constraints

serialized(+Starts,+Durations) where $Starts = [S_1, \dots, S_n]$ and $Durations = [D_1, \dots, D_n]$ are lists of domain variables with finite bounds or integers. Durations must be non-negative. *True* if $Starts$ and $Durations$ denote a set of non-overlapping tasks, i.e.:

$$(\forall ij) (S_i + D_i \leq S_j) \vee (S_j + D_j \leq S_i)$$

cumulative(+Starts,+Durations,+Resources,?Limit) where $Starts = [S_1, \dots, S_n]$, $Durations = [D_1, \dots, D_n]$, $Resource = [R_1, \dots, R_n]$ are lists of domain variables with finite bounds or integers, and $Limit$ is a domain variable with finite bounds or an integer.

The elements of *Durations*, *Resources* and *Limit* must all be non-negative. Let:

$$a = \min (S_1, \dots, S_n)$$

$$b = \max (S_1 + D_1, \dots, S_n + D_n)$$

and

$$R_{ij} = \begin{cases} R_j & \text{if } S_j \leq i < S_j + D_j \\ 0 & \text{otherwise} \end{cases}$$

The constraint is *True* iff:

$$(\forall a \leq i < b) R_{i1} + R_{in} \leq Limit$$

Geometric constraints

disjoint2(+Rectangles) where *Rectangles* is a list of terms $F(S_{i1}, D_{i1}, S_{i2}, D_{i2})$. S_{j1} and D_{i1} are domain variables with finite bounds or integers denoting the origin and size of rectangle i in the X dimension, S_{i2} and D_{i2} are the values for the Y dimension, F is any functor. *True* iff no rectangles overlap i.e:

$$(\forall ij) \left(\begin{array}{c} S_{i1} + D_{i1} \leq S_{j1} \\ \vee \\ S_{j1} + D_{j1} \leq S_{i1} \\ \vee \\ S_{i2} + D_{i2} \leq S_{j2} \\ \vee \\ S_{j2} + D_{j2} \leq S_{i2} \end{array} \right)$$