Rule-based Approach to Constraint Programming

Krzysztof R. Apt

CWI, Amsterdam
University of Amsterdam
National University of Singapore

Underlying Thesis

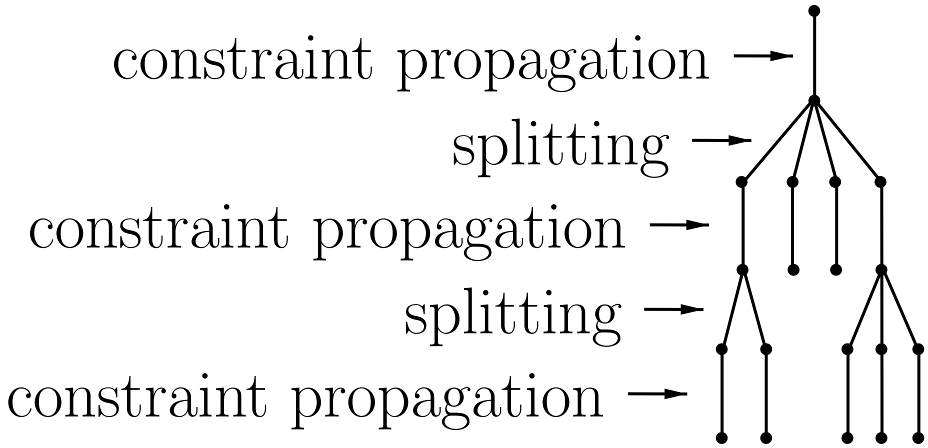
- At various levels of abstraction **constraint programming** (**CP**) can be viewed as an instance of **rule-based programming**.
- At each level this view **sheds light** on the essence of CP.
- At the highest level it allows us to bring CP closer to the **computation as deduction** paradigm.
- At the lowest level it allows us to address the issues of **efficiency**.

High Level

Intuition

- CP is a mix of top-down search and constraint propagation.
- This yields specific **search trees**.
- We separate the issue of the **tree generation** from the **search algorithm** used. (remember LP and Prolog?)

Search Trees



- The nodes in the tree are CSP's.
- The root (level 0) is the original CSP.
- At the even levels the **constraint propagation** is applied to the current CSP.
- At the odd levels **splitting** is applied to the current CSP.
- the '**union**' of the direct descendants of a node is '**equivalent**' to it.

Search Algorithms: an Example

```
MODULE abstract_branch_and_bound;
PROCEDURE abstract_b_and_b(children: searchtree;
                           VAR sol: CSP; VAR bound: REAL);
BEGIN
  WHILE children[P] <> {} DO
    choose R from children[P];    % ‘splitting’
    children[P] := children[P] - {R};
    IF NOT failed(R) THEN
      P := R;
      IF solved(P) THEN
        IF obj(P) > bound THEN
          bound := obj(P);
          sol := P
        END
      ELSE
        P := next(P); % constraint propagation
        IF NOT failed(P) THEN
          IF h(P) > bound THEN
            abstract_b_and_b(children,sol,bound)
          END
        END
      END
    END
  END
END abstract_b_and_b;
```

```
BEGIN
  sol := NIL;
  bound := -infinity;
  P := next(Pinit);   % constraint propagation
  IF NOT failed(P) THEN
    abstract_b_and_b(children,sol,bound)
  END
END abstract_branch_and_bound;
```

Back to Rule-based Approach

Proof Theoretic Framework (Apt '98, '03).

- Two types of **rules** that transform CSP's.
- **Deterministic rules**:

$$\frac{\phi}{\psi}$$

- **Splitting rules**:

$$\frac{\phi}{\psi_1 \mid \ldots \mid \psi_n}$$

- A rule

$$\frac{\phi}{\psi}$$

  is **equivalence preserving** if ϕ and ψ are equivalent (have the same set of solutions).
- A rule

$$\frac{\phi}{\psi_1 \mid \ldots \mid \psi_n}$$

  is **equivalence preserving** if the 'union' of ψ_i's is equivalent to ϕ.

Rule Applications

- **Application** of a deterministic rule:
  replace in a CSP the part that matches the premise by the conclusion.
- **Relevant application** of a deterministic rule:
  the outcome is a different CSP.
- A CSP $\mathcal{P}$ is **closed under the applications of deterministic rule R** if

  $-$ R cannot be applied to $\mathcal{P}$

  or

  $-$ no application of it to $\mathcal{P}$ is relevant.

Derivations

Assumed: notions of **failed** and **solved** CSP's.

Given: a finite set of proof deterministic rules.

- **Derivation**: a sequence of CSP's s.t. each is obtained from the previous one by an application of a deterministic proof rule.
- A finite derivation is called
  - **successful**: last element is a first solved CSP in this derivation,
  - **failed**: last element is a first failed CSP in this derivation,
  - **stabilising**: last element is a first CSP closed under the applications of the proof rules.

Computation Trees

- **Application** of a splitting rule (informally): given

$$\frac{\phi}{\psi_1 \mid \ldots \mid \psi_n}$$

  replace ϕ by some ψ_i.

- By allowing splitting rules in the derivations we obtain **computation trees**.

Medium Level

Intuition

- When deterministic rules are of a **known form**, derivations can be generated more efficiently.
- These derivations correspond then to specific **constraint propagation algorithms**.

Example 1: Domain Reduction Rules

$$\frac{\langle \mathcal{C} \ ; \ x_1 \in D_1, \ldots, x_n \in D_n \rangle}{\langle \mathcal{C}' \ ; \ x_1 \in D'_1, \ldots, x_n \in D'_n \rangle}$$

where $D'_i \subseteq D_i$ and $\mathcal{C}'$ is the restriction of $\mathcal{C}$ to $D'_1, \ldots, D'_n$.

Such a rule is **monotonic** if

smaller variable domains $\Rightarrow$ smaller reductions.

Lemma Suppose each D'_i is obtained from D_i using a combination of

- union and intersection operations,
- transposition and composition operations,
- join operation $\bowtie$,
- projection functions, and
- removal of an element.

Then the rule is monotonic.

Note This covers typical constraint solvers (Boolean constraints, linear constraints over integers, arithmetic constraints over reals, ...).

Generic iteration algorithm

Monotonic domain reduction rules can be scheduled using a **generic iteration algorithm** that computes the lcf of a set of functions F.

(Benhamou '96, Tellerman, Ushakov '96, Apt '97, Fages et al. '98)

$d := \perp;$
$G := F;$
WHILE $G \neq \emptyset$ DO
  choose $g \in G$;
  IF $d \neq g(d)$ THEN
    $G := G \cup update(G, g, d);$
    $d := g(d)$
  ELSE
    $G := G - \{g\}$
  END
END

where for all G, g, d

$$\{f \in F - G \mid f(d) = d \wedge f(g(d)) \neq g(d)\} \subseteq update(G, g, d).$$

Example 2 : Arc Consistency

C: a constraint on x and y.

- *ARC CONSISTENCY 1*

$$\frac{\langle C \ ; \ x \in D_x, y \in D_y \rangle}{\langle C \ ; \ x \in D'_x, y \in D_y \rangle}$$

where $D'_x := \{a \in D_x \mid \exists\, b \in D_y \ (a, b) \in C\}$

- *ARC CONSISTENCY 2*

$$\frac{\langle C \ ; \ x \in D_x, y \in D_y \rangle}{\langle C \ ; \ x \in D_x, y \in D'_y \rangle}$$

where $D'_y := \{b \in D_y \mid \exists\, a \in D_x \ (a, b) \in C\}$.

- **Note** These rules can be scheduled using an **improved** generic iteration algorithm (Apt '00) of which AC-3 is an instance.
- Crucial properties: **commutativity** and **idempotence**.

Example 3 : Propagation Rules

(Apt, Brand '03, '05)

A general class of rules that includes

$$\frac{B}{C}$$

where $B, C \subseteq \mathcal{A}$ with $\mathcal{A}$ a set of given primitive constraints.

Interpretation: if all constraints in B are in the constraint store, then **add** to it all constraints in C.

Scheduling of Propagation Rules

- Propagation rules can be scheduled using a more **fine-tuned** scheduler than the generic iteration.

  Crucial property: **stability** (generalization of idempotence).
- During the computation this scheduler **permanently removes** some rules from the initial set.

  If after splitting we relaunch this scheduler, we can disregard the removed functions.

  This leads to an additional gain.
- Special case of propagation rules: **membership rules** (Apt, Monfroy '99, '01).
- On membership rules this scheduler performs substantially better than the CHR scheduler.

Low Level

Membership rules:

$$\frac{\langle \mathcal{C}\ ;\ y_1 \in S_1, \ldots, y_k \in S_k, z_1 \in D_{z_1}, \ldots, z_m \in D_{z_m}\rangle}{\langle \mathcal{C}\ ;\ z_1 \in D_{z_1} - \{a_1\}, \ldots, z_m \in D_{z_m} - \{a_m\}\rangle}$$

Shorthand:

$$y_1 \in S_1, \ldots, y_k \in S_k \ \rightarrow\ z_1 \neq a_1, \ldots, z_m \neq a_m.$$

Example: Three valued logic of Kleene '52.
Consider `and3`(x, y, z):

	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

- $y \in \{u, f\} \rightarrow z \neq t$ is a **valid** (equivalence preserving) membership rule.
- There are 18 minimal valid membership rules.

Generating Valid Membership Rules

- (Apt, Monfroy '99, '01):

  Given a finite constraint all minimal valid membership rules can be generated.
- These rules define **hyper-arc consistency** (GAC).
- So the fine-tuned scheduler applied to these rules is a **hyper-arc consistency algorithm**.
- (Brand '03, Brand, Apt '05):

  A rule r is **redundant** if the least common fixpoint is the same with r removed.

  One can remove from the set of generated valid rules the redundant ones.

Scheduling of Membership Rules: a Summary

- For finite domain constraints all valid membership rules can be automatically generated (implemented in ECL^{i}PS^e by E. Monfroy).
- Redundant rules can be removed (implemented in ECL^{i}PS^e by S. Brand).
- A fine-tuned scheduler can be used to schedule the rules.
- This scheduler allows us to remove permanently some rules: useful during the top-down search.

Example

Consider `and11`(x, y, z) (used in ATPG) with a randomized labeling.

- Generated minimal valid membership rules: 4656.

  After removing redundant rules: 393.

- Computation times:

Fine-tuned	Generic	CHR
1874	3321	7615

- Computation times after removing redundant rules:

Fine-tuned	Generic	CHR
157	316	543

Conclusions

- Rule-based programming provides useful insights into CP.
- At the **high level** it allows us to stress relations between CP and the computation as deduction paradigm.
- At the **medium level** we can focus on efficient scheduling of specific rules.
- At the **low level** specific rules can be automatically generated, optimized and scheduled in a customized way.
- **A challenge**: use this framework to describe syntax-based complete constraint solvers:
  - Gaussian elimination,
  - Gauss–Jordan Elimination,
  - Martelli-Montanari unification algorithm,
  - optimized versions of Fourier elimination.
- **Needed**: a language to describe the order of rule applications.