Knowledge Representation and Reasoning

Constraint Handling Rules

(Thom Frühwirth)

Prof. Dr. Hans Jürgen Ohlbach
Ideas

- Constraint Processing not as special implementation, but specified by a **rule language** (CHR).
- Integration in a 'Host Language', typically Prolog.
- Choice, **Constraint Processing**, Backtracking

  by CHR

- Interpreter/Compiler for CHR, in Host-Language integrated.
Constraints are represented as Prolog like Facts (Constraint Store)

Constraints may be

- replaced
- removed
- Extended by derived facts.
Euclidian Algorithm:
\[
gcd(a,b):
\text{while}(a \neq b)\
\quad \text{if}(a > b) \text{ replace } a \text{ by } a-b.\
\quad \text{if}(b > a) \text{ replace } b \text{ by } b-a.
\]

Example: \( \gcd(15,12) \):
\[
(15,12) \rightarrow (3,12) \rightarrow (3,9) \rightarrow (3,6) \rightarrow (3,3)
\]

CHR-Version:
Constraint Store: \( \gcd(15), \gcd(12), ... \)
Initial Constraints:

gcd(15), gcd(12), ...

CHR-Rule:

• choose gcd(N), gcd(M) with N < M
• replace gcd(M) with gcd(M-N)

Example:

gcd(12), gcd(15)
gcd(12), gcd(3)
gcd(9), gcd(3)
gcd(6), gcd(3)
gcd(3)              (doubles automatically removed)
Three Types of Rules:

First Type: Simplification Rules:

\[ h_1, \ldots, h_n \iff g_1, \ldots, g_m \mid b_1, \ldots, b_k \]

Meaning:

**If** the head literals are found in the constraint store,
**and** the guards are true
(either found in the constraint store, or computed by build-ins)

**Replace** the head literals by the body literals.
Simplification Rule: \[ \text{gcd}(N), \text{gcd}(M) \iff N \leq M \mid L \text{ is } M - N, \text{gcd}(N), \text{gcd}(L). \]

Example:
\[ \text{gcd}(12), \text{gcd}(15) \]
head literals \textit{match}: \( N = 12, M = 15 \)
guard: \( N \leq M \) is true (12 \( \leq \) 15)
body: build-in assignment \( L \) is \( M - N \) yields \( L = 3 \)
Remove head literals: \( \text{gcd}(12), \text{gcd}(15) \)
add body literals: \( \text{gcd}(12), \text{gcd}(3) \)
Continue rule application until nothing changes.
Another Example for gcd

**Simplification Rule:**
\[ \text{gcd}(N), \text{gcd}(M) \iff N \leq M \mid L \text{ is } M - N, \text{gcd}(N), \text{gcd}(L). \]

**Example:**
\[ \text{gcd}(12), \text{gcd}(15), \text{gcd}(9) \]
\[ \rightarrow \text{gcd}(12), \text{gcd}(3), \text{gcd}(9) \] (ordering of literals is irrelevant)
\[ \rightarrow \text{gcd}(9), \text{gcd}(3) \] (double literals removed)
\[ \rightarrow \text{gcd}(6), \text{gcd}(3) \]
\[ \rightarrow \text{gcd}(3) \]
**Propagation Rules**

\[
h_1, \ldots, h_n \implies g_1, \ldots, g_m \mid b_1, \ldots, b_k
\]

**Meaning:**
- **if** the head literals are found in the constraint store
- **and** the guards are true (either found, or computed)
- **then add** the body literals to the constraint store.
Propagation Rule:
\[ \mathsf{r}(X,Y), \mathsf{r}(Y,Z) \implies X \neq Y, Y \neq Z \mid \mathsf{r}(X,Z). \]

Example:
\[ \mathsf{r}(a,b), \mathsf{r}(b,c), \mathsf{r}(c,d) \]
\[ \to \mathsf{r}(a,b), \mathsf{r}(b,c), \mathsf{r}(c,d), \mathsf{r}(a,c) \]
\[ \to \mathsf{r}(a,b), \mathsf{r}(b,c), \mathsf{r}(c,d), \mathsf{r}(a,c), \mathsf{r}(a,d) \]
\[ \to \mathsf{r}(a,b), \mathsf{r}(b,c), \mathsf{r}(c,d), \mathsf{r}(a,c), \mathsf{r}(a,d), \mathsf{r}(b,d) \]
A Combination of Simplification and Propagation Rules

\[ h_1, \ldots, h_l \ \\backslash \ \ h_l+1, \ldots, h_n \ \iff \ g_1, \ldots, g_m \ | \ b_1, \ldots, b_k \]

**Meaning:**
- if the keep and remove literals are found in the constraint store
- and the guards are true (either found, or computed)
- then **remove** the 'remove literals' and
  - **add** the body literals to the constraint store.
  (thus untouching the keep literals)
Simpagation Rule for gcd

**Simplification Rule:**
\[ \text{gcd}(N) \setminus \text{gcd}(M) \leftrightarrow N \leq M \mid L \text{ is } M - N, \text{gcd}(L). \]

**Example:**
\[ \text{gcd}(12), \text{gcd}(15) \]
head literals match: \( N = 12, M = 15 \)
guard: \( N \leq M \) is true (12 \( \leq \) 15)
body: build-in assignment \( L \) is \( M - N \) yields \( L = 3 \)
Remove 'remove literal': \( \text{gcd}(15) \)
add the body literal: \( \text{gcd}(3) \)

Continue rule application until nothing changes.
Dijkstra's Shortest-Path Algorithm for Graphs

```java
public final class DijkstraShortestPath<
    V, E> {
    private List<E> edgeList;
    private double pathLength;
    public DijkstraShortestPath(Graph<
        V, E> graph,
        V startVertex, V endVertex) {
        this.graph = graph;
        this.startVertex = startVertex;
        this.endVertex = endVertex;
        this.pathLength = Double.POSITIVE_INFINITY;
    }
    public DijkstraShortestPath(Graph<
        V, E> graph, V startVertex, V endVertex, double radius) {
        this.graph = graph;
        this.startVertex = startVertex;
        this.endVertex = endVertex;
        this.radius = radius;
        this.pathLength = Double.POSITIVE_INFINITY;
    }
    public double getShortestPathLength(V vertex) {
        return pathLength;
    }
    public void clear() {
        edgeList = null;
        pathLength = Double.POSITIVE_INFINITY;
    }
    public List<E> getEdgeList() {
        return edgeList;
    }
}

public class FHNode<T> {
    T data;
    FHNode<T> child;
    FHNode<T> left;
    FHNode<T> parent;
    FHNode<T> right;
    boolean mark;
    double key;
    int degree;
    public FHNode(T data, double key) {
        this.data = data;
        this.key = key;
    }
    public final double getKey() {
        return key;
    }
    public final T getData() {
        return data;
    }
    public static final double oneOverLogPhi = 1.0 / Math.log(1.0 + Math.sqrt(5.0)) / 2.0;
    private FHNode<T> minNode;
    private int nNodes;
    public FHNode<T> minNode;
Dijkstra's Shortest-Path Algorithm in CHR

Start Node: S
source(S) ==> distance(S,0).

Keep the shorter distance from S to V:
distance(V,D1) \ distance(V,D2) <=> D1 <= D2 | true.

Extend the distance to V by the label of the edge from V to W:
distance(V,D), edge(V,W,C) ==> distance(W,D+C).
Dijkstra's Shortest-Path Example

edge(1,2,3), edge(1,3,5), edge(2,3,1)
edge(2,4,8), edge(3,4,2), source(1).

Distance Calculations:
distance(1,0)
distance(2,3)
distance(3,5)
distance(3,4) → delete distance(3,5)
distance(4,11)
distance(4,6) → delete distance(4,11)

Final constraint store:
edge(1,2,3), edge(1,3,5), edge(2,3,1)
edge(2,4,8), edge(3,4,2), source(1),
distance(1,0), distance(2,3), distance(3,4), distance(4,6)
Useful Links

- CHR in SWI-Prolog: http://www.swi-prolog.org/man/chr.html
- CHR in Java: https://dtai.cs.kuleuven.be/CHR/JCHR/
- CHR online: http://chr.informatik.uni-ulm.de/~webchr/