Constraint (Logic) Programming languages

The Holy Grail

> *Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.*

Eugene C. Freuder, Inaugural issue of the *Constraints Journal*, 1997.

Constraint Reasoning

The Idea

- *Example – Combination Lock:*
  0 1 2 3 4 5 6 7 8 9
  Greater or equal 5.
  Prime number.
- *Declarative problem* representation by variables and constraints:
  $x \in \{0, 1, \ldots, 9\} \ \wedge \ x \geq 5 \ \wedge \ \text{prime}(x)$
- *Constraint propagation and simplification* reduce search space:
  $x \in \{0, 1, \ldots, 9\} \ \wedge \ x \geq 5 \ \rightarrow \ x \in \{5, 6, 7, 8, 9\}$

Constraint Programming

Robust, flexible, maintainable software faster.

- *Declarative modeling by constraints:*
  Description of properties and relationships between partially known objects.
  Correct handling of precise and imprecise, finite and infinite, partial and full information.
- *Automatic constraint reasoning:*
  Propagation of the effects of new information (as constraints).
  Simplification makes implicit information explicit.
- *Solving combinatorial problems efficiently:*
  Easy Combination of constraint solving with search and optimization.

Terminology

Language is first-order logic with equality.

- *Constraint:*
  Conjunction of atomic constraints (predicates)
  E.g., $4X + 3Y = 10 \ \wedge \ 2X - Y = 0$
- *Constraint Problem (Query):*
  A given, initial constraint
- *Constraint Solution (Answer):*
  A valuation for the variables in a given constraint problem that satisfies all constraints of the problem. E.g., $X = 1 \wedge Y = 2$

In general, a normal/solved form of, e.g., the problem
$4X + 3Y + Z = 10 \ \wedge \ 2X - Y = 0$ simplifies into
$Y + Z = 10 \ \wedge \ 2X - Y = 0$

Constraint Reasoning and Constraint Programming

A generic framework for

- Modeling
  - with partial information
  - with infinite information
- Reasoning
  - with new information
- Solving
  - combinatorial problems

Two main classes of problems

Constraint Satisfaction Problems (CSP)

- studied in artificial intelligence since 1970s
- A CSP is defined by:

  a finite set of variables $\{X_1, \ldots, X_n\}$
  a set of values (Domain) for each variable $D(X_1), \ldots D(X_n)$
  A set of constraints $\{C1, \ldots, C_m\}$
- A solution to a CSP is a complete assignment to all the variables that satisfies the constraints.
- e.g. $X \in \{1,2\} \wedge Y \in \{1,2\} \wedge Z \in \{1,2\} \wedge X = Y \wedge X \neq Z \wedge Y > Z$

Constraint Optimization Problems (COP)

- A COP is a CSP defined on the variables $\{X_1, \ldots, X_n\}$ and domains $D(X_1), \ldots D(X_n)$, together with an objective function $f : D(X_1), \ldots D(X_n) \rightarrow D$. A solution to a COP is a solution of the CSP that optimize the value of f.

Two main family of constraint languages

Constraint Logic Programming (CLP)

- developed in the mid-1980s. Simple idea: add constraints (and the related solvers) to logic programming
- two declarative paradigms together: constraint solving and logic programming
- more expressive, flexible, and in general more efficient than logic programs
- e.g. $X - Y = 3 \ \wedge \ X + Y = 7$ leads to $X = 5 \ \wedge \ Y = 2$
- e.g. $X < Y \ \wedge \ Y < X$ fails without the need to know values
- modern Prolog implementations are CLP

Imperative languages with constraints

- Integration of constraints (and the related solvers) to imperative languages
- Commercial distributions and applications
- One example (free): MinZinc

Constraint Solving

The *solver* is an essential part of constraint languages: it solves the constraints by adapting and combining existing efficient algorithms from

- Mathematics
  - Operations research
  - Graph theory
  - Algebra
- Computer science
  - Finite automata
  - Automatic proving
- Economics
- Linguistics

There are many different constraint domains, with different solvers.

Early History of Constraint Programming

60s, 70s Constraint networks in artificial intelligence.
70s Logic programming (Prolog).
80s Constraint logic programming.
80s Concurrent logic programming.
90s Concurrent constraint programming.
90s Commercial applications.

Application Domains

- Modeling
- Executable Specifications
- Solving combinatorial problems
  - Scheduling, Planning, Timetabling
  - Configuration, Layout, Placement, Design
  - Analysis: Simulation, Verification, Diagnosis

  of software, hardware and industrial processes.

Application Domains (cont)

- Artificial Intelligence
  - Machine Vision
  - Natural Language Understanding
  - Temporal and Spatial Reasoning
  - Theorem Proving
  - Qualitative Reasoning
  - Robotics
  - Agents
  - Bio-informatics

Applications in Research

- *Computer Science:* Program Analysis, Robotics, Agents
- *Molecular Biology, Biochemistry, Bio-informatics:* Protein Folding, Genomic Sequencing
- *Economics:* Scheduling
- *Linguistics:* Parsing
- *Medicine:* Diagnosis Support
- *Physics:* System Modeling
- *Geography:* Geo-Information-Systems

Early Commercial Applications

- *Lufthansa:* Short-term staff planning.
- *Hong Kong Container Harbor:* Resource planning.
- *Renault:* Short-term production planning.
- *Nokia:* Software configuration for mobile phones.
- *Airbus:* Cabin layout.
- *Siemens:* Circuit verification.
- *Caisse d'epargne:* Portfolio management.

In *Decision Support Systems* for Planning, Configuration, for Design, Analysis.

CLP

CLP Syntax vs. LP Syntax

- signature augmented with *constraint symbols*
- consistent first-order constraint theory (CT)
- at least constraint symbols $\top$ and $\bot$
- syntactic equality $\doteq$ as constraints (by including CET into CT)
- constraints handled (solved) by predefined, given constraint solver

CLP Syntax (cont)

- *atom*: expression $p(t_1, \ldots, t_n)$, with predicate symbol p/n
- *atomic constraint*: expression $c(t_1, \ldots, t_n)$, with n-ary constraint symbol c/n
- *constraint*:
  - atomic constraint, or
  - conjunction of constraints
- *goal*:
  - $\top$ (top), or $\bot$ (bottom), or
  - atom, or an atomic constraint, or
  - conjunction of goals
- *(CL) clause*: $A \leftarrow G$, with atom A and goal G
- *CL program*: finite set of CL clauses

CLP Syntax – Summary

Atom:	A, B	$::=$	$p(t_1, \ldots, t_n),\ n \geq 0$
Constraint:	C, D	$::=$	$c(t_1, \ldots, t_n) \mid C \wedge D,\ n \geq 0$
Goal:	G, H	$::=$	$\top \mid \bot \mid A \mid C \mid G \wedge H$
CL Clause:	K	$::=$	$A \leftarrow G$
CL Program:	P	$::=$	$K_1 \ldots K_m,\ m \geq 0$

CLP Semantics via Transition System

- *state* $\langle G, C \rangle$: G goal (store), C constraint (store)
- *initial state*: $\langle G, \top \rangle$
- *successful final state*: $\langle \top, C \rangle$ and C is different from $\bot$
- *failed final state*: $\langle G, \bot \rangle$
- *successful and failed derivations and goals*: as in LP calculus

CLP Operational Semantics

Unfold

If $(B \leftarrow H)$ is a fresh variant of a clause in P

and $CT \models \exists ((B \doteq A) \wedge C)$

then $\langle A \wedge G, C \rangle \mapsto \langle H \wedge G, (B \doteq A) \wedge C \rangle$

Failure

If there is no clause $(B \leftarrow H)$ in P

with $CT \models \exists ((B \doteq A) \wedge C)$

then $\langle A \wedge G, C \rangle \mapsto \langle \bot, \bot \rangle$

Solve

If $CT \models \forall ((C \wedge D_1) \leftrightarrow D_2)$

then $\langle C \wedge G, D_1 \rangle \mapsto \langle G, D_2 \rangle$

CLP Unfold – Comparison with LP

Unfold	
If	$(B \leftarrow H)$ is a fresh variant of a clause in P
and	$CT \models \exists ((B \doteq A) \wedge C)$
then	$\langle A \wedge G, C \rangle \quad \mapsto \langle H \wedge G, (B \doteq A) \wedge C \rangle$

- generalization of LP
- most general unifier in LP
- equality constraint between B and A in context of constraint store C, add equality constraint to store C

$(B \doteq A)$ is shorthand for equating arguments of B and A pairwise

CLP Solve

Solve

If $CT \models \forall\,((C \wedge D_1) \leftrightarrow D_2)$

then $\langle C \wedge G, D_1 \rangle \mapsto \langle G, D_2 \rangle$

- form of simplification depends on constraint system and its constraint solver
- trying to simplify inconsistent constraints to $\bot$
- a failed final state can be reached using **Solve**

CLP State Transition System (vs. LP)

- like in LP, two degrees of non-determinism in the calculus (selecting the goal and selecting the clause)
- like in LP, search trees (mostly SLD resolution)
- *LP*: accumulate and compose substitutions
  CLP: accumulate and simplify constraints
- like substitutions, constraints never removed from constraint store (information increases monotonically during derivations)

CLP as Extension to LP

- derivation in LP can be expressed as CLP derivations:
  - *LP*: substitution $\{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$
  - *CLP*: equality constraints: $X_1 \doteq t_1 \wedge \ldots \wedge X_n \doteq t_n$
- CLP generalizes form of answers.
  - *LP answer*: substitution
  - *CLP answer*: constraint
- Constraints summarize several (even infinitely many) LP answers into one (intensional) answer, e.g.,
  - $X+Y \geq 3 \ \wedge \ X+Y \leq 3$ simplified to
  - $X+Y \doteq 3$
  - variables do not need to have a value

CLP Example – Min

$\min(X,Y,Z) \leftarrow X \leq Y \wedge X \doteq Z$ $(c1)$
$\min(X,Y,Z) \leftarrow Y \leq X \wedge Y \doteq Z$ $(c2)$
Constraints with usual meaning

- $\leq$ total order
- $\doteq$ syntactic equality

CLP Example – Search Tree for `min(1,2,C)`

Goal `min(1,2,C)`:

$$
\begin{array}{ll}
 & \langle \texttt{min}(1,2,\texttt{C}),\ \texttt{true} \rangle \\
\mapsto_{\textbf{Unfold}\ (c1)} & \langle \texttt{X} \leq \texttt{Y} \wedge \texttt{X} \doteq \texttt{Z},\quad 1 \doteq \texttt{X} \wedge 2 \doteq \texttt{Y} \wedge \texttt{C} \doteq \texttt{Z} \rangle \\
\mapsto_{\textbf{Solve}} & \langle \top,\ \texttt{C} \doteq 1 \rangle
\end{array}
$$

Using $(c2)$ leads to inconsistent constraint store
$2 \leq 1 \wedge 2 \doteq \texttt{C}$ – derivation fails

CLP Example – Min (More Derivations)

$\mathsf{min(X,Y,Z)} \leftarrow \mathsf{X} \leq \mathsf{Y} \wedge \mathsf{X} \doteq \mathsf{Z}$ $(c1)$
$\mathsf{min(X,Y,Z)} \leftarrow \mathsf{Y} \leq \mathsf{X} \wedge \mathsf{Y} \doteq \mathsf{Z}$ $(c2)$

- Goal `min(A,2,1)`:

$$\begin{array}{ll} & \langle \mathtt{min(A,2,1)},\ \mathtt{true} \rangle \\ \mapsto_{\textbf{Unfold (c1)}} & \langle \mathtt{X} \leq \mathtt{Y} \wedge \mathtt{X} \doteq \mathtt{Z},\quad \mathtt{A} \doteq \mathtt{X} \wedge \mathtt{2} \doteq \mathtt{Y} \wedge \mathtt{1} \doteq \mathtt{Z} \rangle \\ \mapsto_{\textbf{Solve}} & \langle \top,\ \mathtt{A} \doteq \mathtt{1} \rangle \end{array}$$

  but fails with $(c2)$.

- `min(A,2,2)` has answer $\mathtt{A} \doteq \mathtt{2}$ for $(c1)$, and $2 \leq A$ for $(c2)$
- `min(A,2,3)` fails

(In Prolog, these transitions would lead to error messages.)

CLP vs. LP – Overview

- *generate-and-test in LP*: impractical, facts used in passive manner only
- *constrain-and-generate in CLP*: use facts in active manner to reduce the search space (constraints)
- combination of
  - *LP languages*: declarative, for arbitrary predicates, non-deterministic
  - *constraint solvers*: declarative, efficient for special predicates, deterministic
- combination of search with constraints solving particularly useful

CLP Constrain/Generate vs. LP Generate/Test

Crypto-arithmetic Puzzle – Send More Money

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

Replace distinct letters by distinct digits, numbers have no leading zeros.

CLP Example – Send More Money (Solution)

```
[S,E,N,D,M,O,R,Y] = [9,5,6,7,1,0,8,2]
```

		S	E	N	D
		9	5	6	7
		M	O	R	E
+		1	0	8	5
	M	O	N	E	Y
=	1	0	6	5	2

CLP Example – Send More Money (Constrain/Generate)

```
:- use_module(library(clpfd)).

send([S,E,N,D,M,O,R,Y]) :-
      gen_domains([S,E,N,D,M,O,R,Y],0..9),
      S #\= 0, M #\= 0,
      all_distinct([S,E,N,D,M,O,R,Y]),
                   1000*S + 100*E + 10*N + D
      +            1000*M + 100*O + 10*R + E
      #= 10000*M + 1000*O + 100*N + 10*E + Y,
      labeling([],[S,E,N,D,M,O,R,Y]).

gen_domains([],_).
gen_domains([H|T],D) :- H in D, gen_domains(T,D).
```

CLP Example – Send More Money (Constrain/Generate 2)

`send` **Without labeling**

```
:- send([S,E,N,D,M,O,R,Y]).
M = 1, O = 0, S = 9,
E in 4..7,
N in 5..8,
D in 2..8,
R in 2..8,
Y in 2..8 ?
```

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

CLP Example – Send More Money (Constrain/Generate 3)

`send` **Without labeling**

```
:- send([9,4,N,D,M,O,R,Y]).
no
```

Propagation determines `N = 5`, `R = 8`, but fails as `D` has no possible value. But

```
:- send([9,5,N,D,M,O,R,Y]).
D = 7, M = 1, N = 6,
O = 0, R = 8, Y = 2
yes
```

already computes solution.

		S	E	N	D
+		M	O	R	E
=	M	O	N	E	Y

LP Example – Send More Money (Generate/Test)

```
send([S,E,N,D,M,O,R,Y]) :-
      gen_domains([S,E,N,D,M,O,R,Y],0..9),
      labeling([],[S,E,N,D,M,O,R,Y]),
      S #\= 0, M #\= 0,
      all_distinct([S,E,N,D,M,O,R,Y]),
                   1000*S + 100*E + 10*N + D
      +            1000*M + 100*O + 10*R + E
      #= 10000*M + 1000*O + 100*N + 10*E + Y.
```

95,671,082 choices to find the solution

Houses logical puzzle. Folklore attributes this puzzle to Einstein

- Five colored houses in a row, each with an owner, a pet, cigarettes, and a drink.
- Each house has a different color
- Each owner has a different nationality
- Each owner has a different pet
- Each owner smoke a different brand of cigarette
- Each owner has a different drink

Plus the following contsraints:

Houses logical puzzle: constraints

1. The English lives in the red house.
2. The Spanish has a dog.
3. They drink coffee in the green house.
4. The Ukrainian drinks tea.
5. The green house is next to the white house.
6. The Winston smoker has a serpent.
7. In the yellow house they smoke Kool.
8. In the middle house they drink milk.
9. The Norwegian lives in the first house from the left.
10. The Chesterfield smoker lives near the man with the fox.
11. In the house near the house with the horse they smoke Kool. Lucky Strike smoker drinks juice. Japanese smokes Kent.
12. The Norwegian lives near the blue house.

Who owns the zebra and who drinks water?

A Prolog solution

```
houses(Hs) :-
% each house in the list Hs of houses is represented as:
%  h(Nationality, Pet, Cigarette, Drink, Color)}
        length(Hs, 5), % 1
        member(h(english,_,_,_,red), Hs),  % 2
        member(h(spanish,dog,_,_,_), Hs), % 3
        member(h(_,_,_,coffee,green), Hs), % 4
        member(h(ukrainian,_,_,tea,_), Hs), % 5
        next(h(_,_,_,_,green), h(_,_,_,_,white), Hs), % 6
        member(h(_,snake,winston,_,_), Hs),  % 7
        member(h(_,_,kool,_,yellow), Hs),  % 8
        Hs = [_,_,h(_,_,_,milk,_),_,_],  % 9
        Hs = [h(norwegian,_,_,_,_)|_],  %10
        next(h(_,fox,_,_,_), h(_,_,chesterfield,_,_), Hs), %
        next(h(_,_,kool,_,_), h(_,horse,_,_,_), Hs),  %12
        member(h(_,_,lucky,juice,_), Hs),  %13
        member(h(japanese,_,kent,_,_), Hs),  %14
        next(h(norwegian,_,_,_,_), h(_,_,_,_,blue), Hs), %15
        member(h(_,_,_,water,_), Hs), %one of them drinks
                                                %water
        member(h(_,zebra,_,_,_), Hs). %one of them owns
                                                % a zebra
```

Solution (ctnd)

```
zebra_owner(Owner) :-
        houses(Hs),
        member(h(Owner,zebra,_,_,_), Hs).

water_drinker(Drinker) :-
        houses(Hs),
        member(h(Drinker,_,_,water,_), Hs).

next(A, B, Ls) :- append(_, [A,B|_], Ls).
next(A, B, Ls) :- append(_, [B,A|_], Ls).
```

Examples of goals (queries:)

```
?- zebra_owner(Owner).
?- water_drinker(Drinker).
?- houses(Houses).
```

A CLP(FD) Solution

1. 25 Variables:
   - nationality: english, spaniard, japanese, italian, norwegian,
   - pet: dog, snails, fox, horse, zebra,
   - profession: painter, sculptor, diplomat, violinist, doctor,
   - drink: tea, coffee, milk, juice, water,
   - colour: red, green, white, yellow, blue.
2. Domain: [1...5].
3. Constraints:

```
alldifferent(red,green,white,yellow,blue),
alldifferent(english,spaniard,japanese,italian,norwegian
alldifferent(dog,snails,fox,horse,zebra),
alldifferent(painter,sculptor,diplomat,violinist,doctor)
alldifferent(tea,coffee,milk,juice,water).
```

Constrains (ctnd)

```
% The Englishman lives in the red house:
english = red,
% The Spaniard has a dog:
spaniard = dog,
% The Japanese is a painter:
japanese = painter,
% The Italian drinks tea:
italian = tea,
% The Norwegian lives in the first house on
the left:
norwegian = 1,
& The owner of the green house drinks coffee:
green = coffee,
% The green house is on the right of the white
house:
green = white + 1,
% The sculptor breeds snails:
sculptor = snails,
%  The diplomat lives in the yellow house:
diplomat = yellow,
```

Constrains (ctnd)

```
%  They drink milk in the middle house:
milk = 3,
%  The Norwegian lives next door to the blue house:
| norwegian ? blue | = 1,
% The violinist drinks fruit juice:
violinist = juice,
% The fox is in the house next to the doctor's:
|fox ? doctor| = 1,
% The horse is in the house next to the diplomat's:
| horse ? diplomat | = 1.
```

Exercise: complete the program by using these constraints.

Mortgage example

```
mg(P,T,I,B,R):-
    { T = 1,
      R + B = P * (1 + I)
    }.
mg(P,T,I,B,R):-
    { T > 1,
      P1 = P * (1 + I) - B,
      T1 = T - 1
    },
    mg(P1, T1, I, B, R).
```

P: Amount of Loan, Debt, Principal

T: Duration of loan in months

I: Interest rate per month

B: Balance of debt after T months

R: Rate of payments per month

Mortgage example

- We can use the previous program to compute the balance if we get an interest of 5% over a period of 30 years.

```
?- mg(1000, 30, 5/100, B, 0)
```

- We can however also compute the linear relation between the initial balance, the final balance and the withdrawal:

```
?- mg(B0, 30, 5/100, B, MP)
```