Chapter 4: Constraint Logic Programs

Where we learn about the only programming concept rules, and how programs execute

Constraint Logic Programs

- User-Defined Constraints
- Programming with Rules
- Evaluation
- Derivation Trees and Finite Failure
- Goal Evaluation
- Simplified Derivation Trees
- The CLP Scheme
User-Defined Constraints

- Many examples of modelling can be partitioned into two parts
  - a general description of the object or process
  - and specific information about the situation at hand
- The programmer should be able to define their own problem specific constraints
- Rules enable this

Rules

A user defined constraint to define the model of the simple circuit:

```
parallel_resistors(V,I,R1,R2)
```

And the rule defining it

```
parallel_resistors(V,I,R1,R2) :-
    V = I1 * R1, V = I2 * R2, I1 + I2 = I.
```
**Using Rules**

parallel_resistors(V,I,R1,R2) :-

\[ V = I1 \times R1, \quad V = I2 \times R2, \quad I1 + I2 = I. \]

Behaviour with resistors of 10 and 5 Ohms

\[ \text{parallel_resistors}(V,I,R1,R2) \land R1 = 10 \land R2 = 5 \]

Behaviour with 10V battery where resistors are the same

\[ \text{parallel_resistors}(10,I,R,R) \]

It represents the constraint (macro replacement)

\[ 10 = I1 \times R \land 10 = I2 \times R \land I1 + I2 = I \]

**User-Defined Constraints**

- **user-defined constraint**: \( p(t1,\ldots,tn) \) where \( p \) is an \( n \)-ary **predicate** and \( t1,\ldots,tn \) are expressions
- **literal**: a prim. or user-defined constraint
- **goal**: a sequence of literals \( L1,\ldots,Lm \)
- **rule**: \( A :- B \) where \( A \) is a user-defined constraint and \( B \) a goal
- **program**: a sequence of rules
Its not macro replacement!

Imagine two uses of parallel resistors

\[ \text{parallel_resistors}(VA, IA, 10, 5), \]
\[ \text{parallel_resistors}(VB, IB, 8, 3), \]
\[ VA + VB = V, \quad I = IB, \quad I = IA \]

After macro replacement (converting comma to conj)
\[ VA = I_1 \times 10 \land VA = I_2 \times 5 \land I_1 + I_2 = IA \land \]
\[ VB = I_1 \times 8 \land VB = I_2 \times 3 \land I_1 + I_2 = IB \land \]
\[ VA + VB = V \land I = IB \land I = IA \]

Confused the two sets of local variables \( I_1, I_2 \)

Renamings

- A **renaming** \( r \) is a bijective (invertable) mapping of variables to variables
- A **syntactic object** is a constraint, user-defined constraint, goal or rule
- Applying a renaming to a syntactic object gives the object with each variable \( x \) replaced by \( r(x) \)
- **variant** \( o' \) of object \( o \) has renaming \( r(o') = o \)
Rewriting User-Defined Cons.

- goal $G$ of the form (or empty $m=0$ [])
  - $L_1, ..., L_{i-1}, L_i, L_{i+1}, ..., L_m$
  - $L_i$ is of the form $p(t_1, ..., t_n)$
  - $R$ is of the form $p(s_1, ..., s_n) : -$ $B$
  - $r$ is a renaming s.t. vars in $r(R)$ not in $G$
- The rewriting of $G$ at $L_i$ by $R$ using renaming $r$ is
  - $L_1, ..., L_{i-1}, t_1 = r(s_1), ..., t_n = r(s_n), r(B), L_{i+1}, ..., L_m$

Rewriting Example

```haskell
parallel_resistors(VA, IA, 10, 5),
parallel_resistors(VB, IB, 8, 3),
VA + VB = V, I = IB, I = IA
```

Rewrite the first literal with rule

```haskell
parallel_resistors(V, I, R1, R2) :-
   V = I1 * R1, V = I2 * R2, I1 + I2 = I.
```

Renaming: [$V \mapsto V', I \mapsto I', R1 \mapsto R1', R2 \mapsto R2', I1 \mapsto I1', I2 \mapsto I2'$]

```haskell
parallel_resistors(V', I', R1', R2') :-
   V' = I1' * R1', V' = I2' * R2', I1' + I2' = I'.
```
**Rewriting Example**

\[
\begin{align*}
VA &= V', \quad IA = I', \quad 10 = R1', \quad 5 = R2', \\
V' &= I_1' \times R1', \quad V' &= I_2' \times R2', \quad I_1' + I_2' = I', \\
\text{parallel_resistors}(V_B, I_B, 8, 3), \\
VA + VB &= V, \quad I = IB, \quad I = IA
\end{align*}
\]

Rewrite the 8th literal

Renaming:\{(V \mapsto V'', I \mapsto I'', R_1 \mapsto R_1'', R_2 \mapsto R_2'', I_1 \mapsto I_1'', I_2 \mapsto I_2'')\}

\[
\text{parallel_resistors}(V'', I'', R_1'', R_2'') \implies
V'' = I_1'' \times R_1'', \quad V' = I_2'' \times R_2'', \quad I_1'' + I_2'' = I''
\]

**Rewriting Example**

\[
\begin{align*}
VA &= V', \quad IA = I', \quad 10 = R1', \quad 5 = R2', \\
V' &= I_1' \times R1', \quad V' &= I_2' \times R2', \quad I_1' + I_2' = I', \\
VB &= V'', \quad IB = I'', \quad 8 = R1'', \quad 3 = R2'', \\
V'' &= I_1'' \times R1'', \quad V'' = I_2'' \times R2'', \quad I_1'' + I_2'' = I'' \\
VA + VB &= V, \quad I = IB, \quad I = IA
\end{align*}
\]

Simplifying onto the variables of interest \( V \) and \( I \)

\[ V = \frac{26}{3} \times I \]
A voltage divider circuit, where cell must be 9 or 12V resistors 5,9 or 14

\[
voltage\_divider(V,I,R1,R2,VD,ID) :-
V1 = I*R1, VD = I2*R2, V = V1+VD, I = I2+ID.
\]

\[\text{cell}(9). \quad \text{(shorthand for cell}(9) :- []).\]
\[\text{cell}(12).\]
\[\text{resistor}(5). \quad \text{resistor}(9). \quad \text{resistor}(14).\]

Aim: find component values such that the divider voltage \( V_D \) is between 5.4 and 5.5 V when the divider current \( ID \) is 0.1A

\[
voltage\_divider(V,I,R1,R2,VD,ID),
5.4 <= VD, VD <= 5.5, ID = 0.1,
\]

\[\text{cell}(V), \quad \text{resistor}(R1), \quad \text{resistor}(R2).\]

Note: when rewriting cell and resistor literals there is a choice of which rule to use

\((V=9,R1=5,R2=5)\) unsatisfiable constraint
\((V=9,R1=5,R2=9)\) satisfiable constraint
Consider the factorial function, how do we write rules for a predicate \( \text{fac}(N, F) \) where \( F = N! \)

(R1) \( \text{fac}(0,1) \).
(R2) \( \text{fac}(N,N*F) :- N >= 1, \text{fac}(N-1, F). \)

Note how the definition is recursive (in terms of itself) and mimics the mathematical definition

Rewriting the goal \( \text{fac}(2, X) \) (i.e. what is 2!)

\[
\text{fac}(2, X) \\
\Downarrow\text{R2} \\
2 = N, X = N \times F, N \geq 1, \text{fac}(N-1, F) \\
\Downarrow\text{R2} \\
2 = N, X = N \times F, N \geq 1, N - 1 = N', F = N' \times F', N' \geq 1, \text{fac}(N'-1, F') \\
\Downarrow\text{R1} \\
2 = N, X = N \times F, N \geq 1, N - 1 = N', F = N' \times F', N' \geq 1, N' - 1 = 0, F' = 1
\]

Simplified onto variable \( X \), then answer \( X = 2 \)
Evaluation

- In each rewriting step we should check that the conjunction of primitive constraints is satisfiable
- derivation does this
- in each step a literal is handled
  - primitive constraints: added to constraint store
  - user-defined constraints: rewritten

Evaluation

- state: \(<G1| C1>\) where \(G1\) is a goal and \(C1\) is a constraint
- derivation step: \(G1\) is \(L1, L2, ..., Lm\)
  - \(L1\) is a primitive constraint, \(C2\) is \(C1 \land L1\)
    - if \(\text{solv}(C \land L1) = \text{false}\) then \(G2 = []\)
    - else \(G2 = L2, ..., Lm\)
  - \(L1\) is a user-defined constraint, \(C2\) is \(C1\) and \(G2\) is the rewriting of \(G1\) at \(L1\) using some rule and renaming
Evaluation

- derivation for \( <G0 \mid C0> \):
  \[
  \langle G0|C0 \rangle \Rightarrow \langle G1|C1 \rangle \Rightarrow \langle G2|C2 \rangle \Rightarrow \ldots
  \]

- where each \( <Gi \mid Ci> \) to \( <Gi+1 \mid Ci+1> \) is a derivation step

- derivation for \( G \) is a derivation for the state \( <G \mid true> \)

Derivation for \( \text{fac}(1, Y) \)

\[
\begin{align*}
\langle \text{fac}(1, Y)|true \rangle \\
\downarrow_{R2} \\
\langle 1 = N, Y = N \times F, N \geq 1, \text{fac}(N-1, F)|true \rangle \\
\downarrow \\
\langle Y = N \times F, N \geq 1, \text{fac}(N-1, F)|1 = N \rangle \\
\downarrow \\
\langle N \geq 1, \text{fac}(N-1, F)|1 = N \wedge Y = N \times F \rangle \\
\downarrow \\
\langle \text{fac}(N-1, F)|1 = N \wedge Y = N \times F \wedge N \geq 1 \rangle \\
\downarrow_{R1} \\
\langle N - 1 = 0, F = 1|1 = N \wedge Y = N \times F \wedge N \geq 1 \rangle \\
\downarrow \\
\langle F = 1|1 = N \wedge Y = N \times F \wedge N \geq 1 \wedge N - 1 = 0 \rangle \\
\downarrow \\
\langle |1| = N \wedge Y = N \times F \wedge N \geq 1 \wedge N - 1 = 0 \wedge F = 1 \rangle
\end{align*}
\]

Corresponding answer simplified to \( Y = 1 \)
Derivation for \texttt{fac}(1, Y)

\[
\{ \text{fac}(1, Y)|true \} \\
\downarrow R_1 \\
\{ l = 0, Y = 1 | true \} \\
\downarrow \\
\{ |l| = 0 \}
\]

A failed derivation for \texttt{fac}(1, Y)

Derivations

\begin{itemize}
\item For derivation beginning at \texttt{<G0 | C0>}
\item \textbf{success state:} \texttt{<I | C>} where \texttt{solv(C)} \neq false
\item \textbf{successful} derivation: last state is success
\item \textbf{answer:} simpl(C, vars(\texttt{<G0 | C0>})
\item \textbf{fail state:} \texttt{<I | C>} where \texttt{solv(C)} = false
\item \textbf{failed} derivation: last state is fail state
\end{itemize}
Derivation Trees

- **derivation tree** for goal G
  - root is $< G \mid true >$
  - the children of each state are the states reachable in one derivation step
- Encodes all possible derivations
- when leftmost literal is prim. constraint only one child
- otherwise children ordered like rule order

Derivation Tree Example

\[ \langle \text{fac}(1,Y)\mid \text{true} \rangle \]

- $\downarrow R_1$
  - $\{ 1 = 0, Y = 1 \mid \text{true} \}$
  - $\downarrow$
  - $\{ | | | = 0 \}$
- failed derivation
  - $\downarrow R_1$
    - $\{ N - 1 = 0, F = \|$C3$\}$
    - $\downarrow$
    - $\{ F = \|$C4$ = C3 \land N - 1 = 0 \}$
    - $\downarrow$
    - $\{ | | | C5 \land F = 1 \}$

- $\downarrow R_2$
  - $\{ Y = N \times F, N \geq 1, \text{fac}(N - 1, F)\mid \text{true} \}$
  - $\downarrow$
  - $\{ Y = N \times F, N \geq 1, \text{fac}(N - 1, F)\mid C1 \equiv 1 = N \}$
  - $\downarrow$
  - $\{ N \geq 1, \text{fac}(N - 1, F)\mid C2 = C1 \land Y = N \times F \}$
  - $\downarrow$
  - $(\text{fac}(N - 1, F)\mid C3 = C2 \land N \geq 1)\}$
  - $\downarrow R_2$
    - $\{ N - 1 = N', F = N' \times F', N' \geq 1, \text{fac}(N' - 1, F')\mid C3 \}$
    - $\downarrow$
    - $\{ F = \|$C4$ = C3 \land N - 1 = 0 \}$
    - $\downarrow$
    - $\{ | | | C5 \land C4 \land F = 1 \}$
    - $\downarrow$
    - $\{ N' \geq 1, \text{fac}(N' - 1, F')\mid C7 = C6 \land F = N' \times F' \}$

answer: $Y = 1$
Derivation Trees

- The previous example shows three derivations, 2 failed and one successful
- **finitely failed**: if a derivation tree is finite and all derivations are failed
- next slide a finitely failed derivation tree
- **infinite** derivation tree: some derivations are infinite

\[
\begin{align*}
\text{Finitely Failed Example} & \quad \langle \text{fac}(1,0)[\text{true}] \rangle \\
\Downarrow R_1 \quad \langle 1 = 0.0 = \top[\text{true}] \rangle & \quad \langle 1 = N, 0 = N \times F, N \geq 1, \text{fac}(N - 1, F)[\text{true}] \rangle \\
\Downarrow \quad \langle \top \rangle & \quad \langle 0 = N \times F, N \geq 1, \text{fac}(N - 1, F)[C1 \equiv 1 = N] \rangle \\
\Downarrow & \quad \langle N \geq 1, \text{fac}(N - 1, F)[C2 = C1 \land 0 = N \times F] \rangle \\
\Downarrow & \quad \langle \text{fac}(N - 1, F)[C3 \equiv C2 \land N \geq 1] \rangle \\
\Downarrow R_2 & \quad \langle N - 1 = 0, F = \top[C3] \rangle \\
\Downarrow & \quad \langle N - 1 = N', F = N' \times F', N' \geq 1, \text{fac}(N' - 1, F')\rangle\rangle[C3] \\
\Downarrow & \quad \langle F = \top[C4 \equiv C3 \land N - 1 = 0] \rangle \\
\Downarrow & \quad \langle F = N' \times F', N' \geq 1, \text{fac}(N' - 1, F')\rangle[C6 \equiv C3 \land N - 1 = N'] \\
\Downarrow & \quad \langle \top[C5 \equiv C4 \land F = 1] \rangle \\
\Downarrow & \quad \langle N' \geq 1, \text{fac}(N' - 1, F')\rangle[C7 \equiv C6 \land F = N' \times F'] \\
\Downarrow & \quad \langle \top[C8 \equiv C7 \land N' \geq 1] \rangle
\end{align*}
\]
Infinite Derivation Tree

(S1) stupid(X) :- stupid(X).
(S2) stupid(1).

\[ \langle \text{stupid}(X)|\text{true} \rangle \]
\[ \Downarrow S1 \]
\[ \langle X = X', \text{stupid}(X')|\text{true} \rangle \]
\[ \Downarrow \]
\[ \langle \text{stupid}(X')|X = X' \rangle \]
\[ \Downarrow S1 \]
\[ \langle X' = X'', \text{stupid}(X'')|X = X' \rangle \]
\[ \Downarrow \]
\[ \langle \text{stupid}(X'')|X = X' \wedge X' = X'' \rangle \]
\[ \Downarrow S1 \]
\[ \langle X = 1|\text{true} \rangle \]
\[ \Downarrow S2 \]
\[ \langle \langle X = 1 \rangle \rangle \]
Answer: \( X = 1 \)

\[ \Downarrow S2 \]
\[ \langle \langle X = 1 \rangle \rangle \]
Answer: \( X = 1 \)

Infinite derivation

Goal Evaluation

- **Evaluation** of a goal performs an in-order depth-first search of the derivation tree
- when a success state in encountered the system returns an answer
- the user can ask for more answers in which case the search continues
- execution halts when the users requests no more answers or the entire tree is explored
Goal Evaluation Example

\[
\begin{align*}
\downarrow R_1 & \quad \{x = 0, y = \text{true}\} & \downarrow R_2 & \quad \{x = N, y = N \times \text{factor}(N - 1), \text{factor}(N - 1), \text{true}\} \\
\downarrow R_1 & \quad \{y = N \times \text{factor}(N - 1), N \geq 1, y = \text{true}\} & \downarrow R_2 & \quad \{y = N \times \text{factor}(N - 1), N \geq 1, y = \text{true}\} \\
\downarrow R_1 & \quad \{N \geq 1, \text{factor}(N - 1), C_1 \land y = N \times \text{factor}(N - 1), \text{true}\} & \downarrow R_2 & \quad \{N \geq 1, \text{factor}(N - 1), C_2 \land y = N \times \text{factor}(N - 1), \text{true}\} \\
\end{align*}
\]

Return answer: Y = 1 more? \quad \{\text{true} \} \\
Return no more

Goal Evaluation Example 2

\[
\begin{align*}
\downarrow S_1 & \quad \{\text{stupid}(X) \land \text{true}\} & \downarrow S_2 & \quad \{X = X' \land \text{true}\} \\
\downarrow S_1 & \quad \{X = X' \land \text{true}\} & \downarrow S_2 & \quad \{X = X' \land \text{true}\} \\
\downarrow S_1 & \quad \{X = X' \land \text{true}\} & \downarrow S_2 & \quad \{X = X' \land \text{true}\} \\
\end{align*}
\]

The evaluation never finds an answer, even though infinitely many exist
Simplified Derivation Trees

- Derivation trees are very large
- A simplified form which has the most useful information
  - constraints in simplified form (variables in the initial goal and goal part of state)
  - uninteresting states removed

Simplified State

- **simplified state:** \( < G_0 | C_0 > \) in derivation for \( G \)
  - replace \( C_0 \) with \( C_1 = simpl(C_0, \text{vars}(G,G_0)) \)
  - if \( x = t \) in \( C_1 \) replace \( x \) by \( t \) in \( G_0 \) giving \( G_1 \)
  - replace \( C_1 \) with \( C_2 = simpl(C_1, \text{vars}(G,G_1)) \)

\[
\frac{\text{fac}(N'-1,F')}{|l = N \land Y = N \times F \land N \geq 1 \land N' - 1 = N' \land F = F'}
\]
  vars = \{Y, N', F'\}

\[
\frac{\text{fac}(N'-1,F')}{|N' = 0 \land Y = F'}
\]
replace \( N' \) by 0 and simplify again

\[
\frac{\text{fac}(-1,F')}{|Y = F'}
\]
Simplified Derivation

- A state is **critical** if it is the first or last state of a derivation or the first literal is a user-defined constraint
- A **simplified derivation** for goal G contains all the critical states in simplified form
- similarly for a **simplified derivation tree**

Example Simplified Tree

\[
\begin{array}{c}
\langle \text{fac}(1,Y) | \text{true} \rangle \\
\downarrow R1 \quad \downarrow R2 \\
\langle [] | \text{false} \rangle \quad \langle \text{fac}(0,F) | Y = F \rangle \\
\downarrow R1 \quad \downarrow R2 \\
\langle [] | Y = 1 \rangle \\
\end{array}
\]

Note: fail states are <[] | false> and success states contain answers
The CLP Scheme

- The scheme defines a family of programming languages
- A language $CLP(X)$ is defined by
  - constraint domain $X$
  - solver for the constraint domain $X$
  - simplifier for the constraint domain $X$
- Example we have used $CLP(Real)$
- Another example $CLP(Tree)$

$CLP(R)$

- Example domain for chapters 5, 6, 7
- Elements are trees containing real constants
- Constraints are $\{=, \neq\}$ for trees
- and $\{=, \leq, <, \geq, >\}$ for arithmetic
Summary

- rules: for user-defined constraints
  - multiple rules for one predicate
  - can be recursive
- derivation: evaluates a goal
  - successful: gives an answer (constraint)
  - failed: can go no further
  - infinite
- scheme: defines a CLP language