Chapter 4:Constraint Logic Programs

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

1

Constraint Logic Programs

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

2

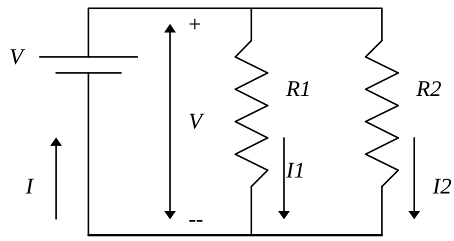
User-Defined Constraints

- 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

3

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.
```

4

Using Rules

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

Behaviour with resistors of 10 and 5 Ohms

$$\texttt{parallel\_resistors}(V, I, R1, R2) \wedge R1 = 10 \wedge R2 = 5$$

Behaviour with 10V battery where resistors are the same

$$\texttt{parallel\_resistors}(10, I, R, R)$$

It represents the constraint (macro replacement)

$$10 = I1 \times R \wedge 10 = I2 \times R \wedge I1 + I2 = I$$

5

User-Defined Constraints

- **user-defined constraint**: *p(t1,...,tn)* where *p* is an *n*-ary **predicate** and *t1,...,tn* are expressions
- **literal**: a prim. or user-defined constraint
- **goal**: a sequence of literals *L1,...,Lm*
- **rule**: *A* :- *B* where *A* is a user-defined constraint and *B* a goal
- **program**: a sequence of rules

6

Its not macro replacement!

Imagine two uses of parallel resistors

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

After macro replacement (converting comma to conj)

$$VA = I1 \times 10 \wedge VA = I2 \times 5 \wedge I1 + I2 = IA \wedge$$
$$VB = I1 \times 8 \wedge VB = I2 \times 3 \wedge I1 + I2 = IB \wedge$$
$$VA + VB = V \wedge I = IB \wedge I = IA$$

Confused the two sets of local variables *I1*, *I2*

7

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*

8

Rewriting User-Defined Cons.

- goal *G* of the form (or empty *m*=0 [])
  - *L1, ..., Li-1, Li, Li+1, ..., Lm*
- *Li* is of the form *p(t1,...,tn)*
- *R* is of the form *p(s1,...,sn) :- B*
- *r* is a renaming s.t. vars in *r(R)* not in *G*
- The rewriting of *G* at *Li* by *R* using renaming *r* is
  - *L1,...,Li-1,t1=r(s1),...,tn=r(sn),r(B),Li+1,...,Lm*

9

Rewriting Example

```
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

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

Renaming: $\{V \mapsto V', I \mapsto I', R1 \mapsto R1', R2 \mapsto R2', I1 \mapsto I1', I2 \mapsto I2'\}$

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

10

Rewriting Example

```
VA=V’, IA=I’, 10=R1’, 5=R2’,
V’ = I1’*R1’, V’ = I2’*R2’, I1’+I2’ = I’,
parallel_resistors(VB,IB,8,3),
VA + VB = V, I = IB, I = IA
```

Rewrite the 8th literal

Renaming: $\{V \mapsto V'', I \mapsto I'', R1 \mapsto R1'', R2 \mapsto R2'', I1 \mapsto I1'', I2 \mapsto I2''\}$

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

11

Rewriting Example

```
VA=V’, IA=I’, 10=R1’, 5=R2’,
V’ = I1’*R1’, V’ = I2’*R2’, I1’+I2’ = I’,
VB=V’’, IB=I’’, 8=R1’’, 3=R2’’,
V’’=I1’’*R1’’, V’’=I2’’*R2’’, I1’’+I2’’=I’’
VA + VB = V, I = IB, I = IA
```

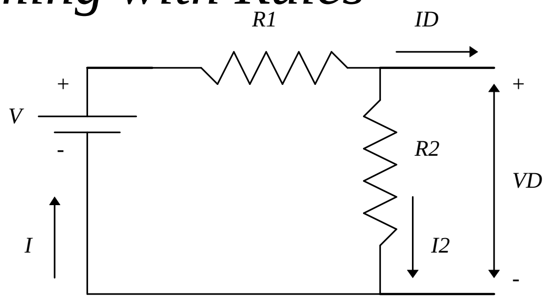
Simplifying onto the variables of interest V and I

$$V = 26/3 \times I$$

12

Programming with Rules

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.
cell(9).  (shorthand for cell(9) :- [].)
cell(12).
resistor(5). resistor(9). resistor(14).
```

13

Programming with Rules

Aim: find component values such that the divider voltage *VD* 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,
cell(V), resistor(R1), 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

14

Programming with Rules

$$N! = \begin{cases} 1 & \text{if } N = 0 \\ N \times (N-1)! & \text{if } N \geq 1 \end{cases}$$

Consider the factorial function, how do we write rules for a predicate `fac(N,F)` where $F = N!$

```
(R1) fac(0,1).
(R2) fac(N,N*F) :- N >= 1, fac(N-1, F).
```

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

15

Programming with Rules

```
(R1) fac(0,1).
(R2) fac(N,N*F) :- N >= 1, fac(N-1, F).
```

Rewriting the goal `fac(2,X)` (i.e. what is 2!)

$$fac(2, X)$$
$$\Downarrow R2$$
$$2 = N, X = N \times F, N \geq 1, fac(N-1, F)$$
$$\Downarrow R2$$
$$2 = N, X = N \times F, N \geq 1, N-1 = N', F = N' \times F', N' \geq 1, fac(N'-1, F')$$
$$\Downarrow R1$$
$$2 = N, X = N \times F, N \geq 1, N-1 = N', F = N' \times F', N' \geq 1, N'-1 = 0, F' = 1$$

Simplified onto variable X, then answer $X = 2$

16

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

17

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 /\ L1*
    - **if** *solv(C /\ L1) = 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

18

Evaluation

- **derivation** for *<G0 | C0>:*

$$\langle G0 | C0 \rangle \Rightarrow \langle G1 | C1 \rangle \Rightarrow \langle G2 | C2 \rangle \Rightarrow \cdots$$

  - where each *<Gi | Ci>* to *<Gi+1 | Ci+1>* is a derivation step
- **derivation** for *G* is a derivation for the state *<G | true>*

19

Derivation for `fac(1,Y)`

$$\begin{array}{c}
\langle fac(1,Y) | true \rangle \\
\Downarrow R2 \\
\langle 1 = N, Y = N \times F, N \geq 1, fac(N-1,F) | true \rangle \\
\Downarrow \\
\langle Y = N \times F, N \geq 1, fac(N-1,F) | 1 = N \rangle \\
\Downarrow \\
\langle N \geq 1, fac(N-1,F) | 1 = N \wedge Y = N \times F \rangle \\
\Downarrow \\
\langle fac(N-1,F) | 1 = N \wedge Y = N \times F \wedge N \geq 1 \rangle \\
\Downarrow R1 \\
\langle N - 1 = 0, F = 1 | 1 = N \wedge Y = N \times F \wedge N \geq 1 \rangle \\
\Downarrow \\
\langle F = 1 | 1 = N \wedge Y = N \times F \wedge N \geq 1 \wedge N - 1 = 0 \rangle \\
\Downarrow \\
\langle [] | 1 = N \wedge Y = N \times F \wedge N \geq 1 \wedge N - 1 = 0 \wedge F = 1 \rangle
\end{array}$$

Corresponding answer simplified to *Y* is *Y = 1*

20

Derivation for `fac(1,Y)`

$$\langle fac(1,Y) | true \rangle$$
$$\Downarrow R1$$
$$\langle 1 = 0, Y = 1 | true \rangle$$
$$\Downarrow$$
$$\langle [] | 1 = 0 \rangle$$

A failed derivation for `fac(1,Y)`

21

Derivations

- For derivation beginning at *<G0 | C0>*
- **success state:** *<[] | C>* where *solv(C) != false*
- **successful** derivation: last state is success
- **answer**: simpl(C, vars(<G0 | C0>))
- **fail state:** *<[] | C>* where *solv(C) = false*
- **failed** derivation: last state is fail state

22

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

23

Derivation Tree Example

$$\langle fac(1,Y) \mid true \rangle$$

$$\Downarrow R1$$

$$\langle 1 = 0, Y = 1 \mid true \rangle$$

$$\Downarrow$$

$$\langle [] \mid 1 = 0 \rangle$$

failed derivation

$$\Downarrow R2$$

$$\langle 1 = N, Y = N \times F, N \ge 1, fac(N-1,F) \mid true \rangle$$

$$\Downarrow$$

$$\langle Y = N \times F, N \ge 1, fac(N-1,F) \mid C1 \equiv 1 = N \rangle$$

$$\Downarrow$$

$$\langle N \ge 1, fac(N-1,F) \mid C2 \equiv C1 \wedge Y = N \times F \rangle$$

$$\Downarrow$$

$$\langle fac(N-1,F) \mid C3 \equiv C2 \wedge N \ge 1 \rangle$$

$$\Downarrow R1$$

$$\langle N - 1 = 0, F = 1 \mid C3 \rangle$$

$$\Downarrow$$

$$\langle F = 1 \mid C4 \equiv C3 \wedge N - 1 = 0 \rangle$$

$$\Downarrow$$

$$\langle [] \mid C5 \equiv C4 \wedge F = 1 \rangle$$

answer: *Y = 1*

$$\Downarrow R2$$

$$\langle N - 1 = N', F = N' \times F', N' \ge 1, fac(N'-1,F') \mid C3 \rangle$$

$$\Downarrow$$

$$\langle F = N' \times F', N' \ge 1, fac(N'-1,F') \mid C6 \equiv C3 \wedge N - 1 = N' \rangle$$

$$\Downarrow$$

$$\langle N' \ge 1, fac(N'-1,F') \mid C7 \equiv C6 \wedge F = N' \times F' \rangle$$

$$\Downarrow$$

$$\langle [] \mid C8 \equiv C7 \wedge N' \ge 1 \rangle$$

failed derivation

24

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

25

Finitely Failed Example

$$\begin{array}{cc}
 & \langle fac(1,0) | true \rangle \\
\Downarrow R1 & \Downarrow R2 \\
\langle 1 = 0, 0 = 1 | true \rangle & \langle 1 = N, 0 = N \times F, N \geq 1, fac(N-1, F) | true \rangle \\
\Downarrow & \Downarrow \\
\langle [] | 1 = 0 \rangle & \langle 0 = N \times F, N \geq 1, fac(N-1, F) | C1 \equiv 1 = N \rangle \\
 & \Downarrow \\
 & \langle N \geq 1, fac(N-1, F) | C2 \equiv C1 \wedge 0 = N \times F \rangle \\
 & \Downarrow \\
 & \langle fac(N-1, F) | C3 \equiv C2 \wedge N \geq 1 \rangle \\
\Downarrow R1 & \Downarrow R2 \\
\langle N - 1 = 0, F = 1 | C3 \rangle & \langle N - 1 = N', F = N' \times F', N' \geq 1, fac(N'-1, F') | C3 \rangle \\
\Downarrow & \Downarrow \\
\langle F = 1 | C4 \equiv C3 \wedge N - 1 = 0 \rangle & \langle F = N' \times F', N' \geq 1, fac(N'-1, F') | C6 \equiv C3 \wedge N - 1 = N' \rangle \\
\Downarrow & \Downarrow \\
\langle [] | C5 \equiv C4 \wedge F = 1 \rangle & \langle N' \geq 1, fac(N'-1, F') | C7 \equiv C6 \wedge F = N' \times F' \rangle \\
 & \Downarrow \\
 & \langle [] | C8 \equiv C7 \wedge N' \geq 1 \rangle
\end{array}$$

26

Infinite Derivation Tree

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

$$
\begin{array}{ccl}
\langle stupid(X) \mid true \rangle & & \\
\Downarrow S1 & \Downarrow S2 & \\
\langle X = X', stupid(X') \mid true \rangle & \langle X = 1 \mid true \rangle & \\
\Downarrow & \Downarrow & \\
\langle stupid(X') \mid X = X' \rangle & \langle [] \mid X = 1 \rangle & \text{Answer: } X=1 \\
\Downarrow S1 & \Downarrow S2 & \\
\langle X' = X'', stupid(X'') \mid X = X' \rangle & \langle X' = 1 \mid X = X' \rangle & \\
\Downarrow & \Downarrow & \\
\langle stupid(X'') \mid X = X' \wedge X' = X'' \rangle & \langle [] \mid X = X' \wedge X' = 1 \rangle & \text{Answer: } X=1 \\
\Downarrow S1 & \Downarrow S2 &
\end{array}
$$

Infinite derivation

27

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

28

Goal Evaluation Example

$\langle fac(1,Y) \mid true \rangle$

$\Downarrow R1$

$\langle 1=0, Y=1 \mid true \rangle$

$\Downarrow$

$\langle [] \mid 1=0 \rangle$

$\Downarrow R2$

$\langle 1=N, Y=N\times F, N\geq 1, fac(N-1,F) \mid true \rangle$

$\Downarrow$

$\langle Y=N\times F, N\geq 1, fac(N-1,F) \mid C1 \equiv 1=N \rangle$

$\Downarrow$

$\langle N\geq 1, fac(N-1,F) \mid C2 \equiv C1 \wedge Y=N\times F \rangle$

$\Downarrow$

$\langle fac(N-1,F) \mid C3 \equiv C2 \wedge N\geq 1 \rangle$

$\Downarrow R1$

$\langle N-1=0, F=1 \mid C3 \rangle$

$\Downarrow$

$\langle F=1 \mid C4 \equiv C3 \wedge N-1=0 \rangle$

$\Downarrow$

$\langle [] \mid C5 \equiv C4 \wedge F=1 \rangle$

Return answer: *Y = 1* more?

$\Downarrow R2$

$\langle N-1=N', F=N'\times F', N'\geq 1, fac(N'-1,F') \mid C3 \rangle$

$\Downarrow$

$\langle F=N'\times F', N'\geq 1, fac(N'-1,F') \mid C6 \equiv C3 \wedge N-1=N' \rangle$

$\Downarrow$

$\langle N'\geq 1, fac(N'-1,F') \mid C7 \equiv C6 \wedge F=N'\times F' \rangle$

$\Downarrow$

$\langle [] \mid C8 \equiv C7 \wedge N'\geq 1 \rangle$

Return no more

29

Goal Evaluation Example 2

$\langle stupid(X) \mid true \rangle$

$\Downarrow S1$

$\langle X=X', stupid(X') \mid true \rangle$

$\Downarrow$

$\langle stupid(X') \mid X=X' \rangle$

$\Downarrow S1$

$\langle X'=X'', stupid(X'') \mid X=X' \rangle$

$\Downarrow$

$\langle stupid(X'') \mid X=X' \wedge X'=X'' \rangle$

$\Downarrow S1$

$\boxed{}$

$\Downarrow S2$

$\langle X=1 \mid true \rangle$

$\Downarrow$

$\langle [] \mid X=1 \rangle$

$\Downarrow S2$

$\langle X'=1 \mid X=X' \rangle$

$\Downarrow$

$\langle [] \mid X=X' \wedge X'=1 \rangle$

$\Downarrow S2$

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

30

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

31

Simplified State

- **simplified state:** <*G0* | *C0*> in derivation for *G*
  - replace *C0* with *C1=simpl(C0, vars(G,G0))*
  - if *x=t* in *C1* replace *x* by *t* in *G0* giving *G1*
  - replace *C1* with *C2=simpl(C1, vars(G,G1))*
- Example

$$\langle fac(N'-1,F') | 1 = N \wedge Y = N \times F \wedge N \geq 1 \wedge N - 1 = N' \wedge F = F' \rangle$$

$$\text{vars} = \{Y, N', F'\}$$

$$\langle fac(N'-1,F') | N' = 0 \wedge Y = F' \rangle$$

replace N' by 0 and simplify again

$$\langle fac(-1,F') | Y = F' \rangle$$

32

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**

33

Example Simplified Tree

$$
\begin{array}{c|c}
 & \langle fac(1,Y) | true \rangle \\
\Downarrow R1 & \Downarrow R2 \\
\langle [] | \mathit{false} \rangle & \langle fac(0,F) | Y = F \rangle \\
\hline
\Downarrow R1 & \Downarrow R2 \\
\langle [] | Y = 1 \rangle & \langle [] | \mathit{false} \rangle \\
\hline
\end{array}
$$

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

34

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)*

35

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

36

Constraint Logic Programs 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

37