CS5126: Logic Programming and Constraints

Joxan Jaffar

March 3 - April 7, 2008

Readings

Textbook:

Programming with Constraints, K. Marriott and P. Stuckey, MIT Press, 2000

Additional reading:

Constraint Logic Programming using *ECL^{i}PS^e*, K.R. Apt and M. Wallace,
Cambridge Univ Press, 2007.

P. Van Hentenryck, Constraint Satisfaction in Logic Programming, MIT Press, 1989.

A CLP Survey, by J. Jaffar and M. Maher, *Journal of Logic Programming*, 1994. (*Advanced*)

Why CLP?

- Constraints support relationships among programmer-defined entities
- The CLP Scheme applies to various constraint domains
- Logic Programming supports declarative reasoning
- Constraints + LP support search
- CLP is not just a programming language; its a methodology

Modern CLP Systems:

- CLP($\mathscr{R}$)
- CHIP
- *ECL*^{i}*PS*^{e}
- SicsTus Prolog
- B-Prolog

Constraint Logic Programming

- ***The CLP Scheme***
  - various constraint domains
- ***CLP Evaluation***
  - partial constraint solving: *true*, *false* and *unknown* solutions
- ***Advanced Programming Techniques***
  - literal and rule orderings, and coroutining
  - redundant constraints
- ***Finite Domains***
  - complex constraints
  - ordering variables and values
  - domain splitting
  - FD modelling techniques
- ***Algorithms for Solvers***
  - finite and infinite trees
  - boolean
  - linear arithmetic
  - finite domains
- ***Mathematical Foundations*** (*may be nonexaminable*)

The CLP Scheme

- a ***first-order*** language $\mathscr{L}$ of
  - variables (X, Y, $\cdots$)
  - function symbols Σ $\quad$ ($eg:\ +,\ -,\ fib(.),\ \cdots$)
  - constraint symbols Π $\quad$ ($eg:\ <,\ is\_prime(.),\ \cdots$)
  - predicate symbols (user defined in programs)
  - a *term* is either a variable, or of the form $f(t_1, t_2, \cdots, t_n)$ where $f \in \Sigma$ and t_i, $0 \leq i \leq n$, are terms. Eg $X + 2 * Y - 1$).
  - a *constraint* is of the form $c(t_1, t_2, \cdots, t_n)$ where c is a constraint symbol, and the t_i, $0 \leq i \leq n$, are terms
  - an *atom* is of the form $p(t_1, t_2, \cdots, t_n)$ where p is a predicate symbol, and the t_i, $0 \leq i \leq n$, are terms

- a ***structure*** $\mathscr{D}$
  is an algebra with an underlying *domain of discourse* (eg integers), and a number of basic operations (eg: $+$, $-$)

- A ***constraint domain*** is defined by a language of constraints, and an associated structure.

Example Constraint Domains

- ***Term Structures (Trees)***
  Structure $\mathscr{D}$: the set of terms constructible from Σ
  Function symbols Σ: any collection of n-ary function symbols, for all $n \geq 0$
  Constraint symbols Π: $\quad = \neq$

- ***Integers***
  Structure $\mathscr{D}$: the integers with addition, multiplication, order
  Function symbols Σ: $\quad -1 \;\; 0 \;\; +1 \;\; + \;\; *$
  Constraint symbols Π: $\quad = < \leq$

  Examples:
  $X = 1, \quad X * Y < 5, \quad 2 * X = 1$ (*unsatisfiable*)
  $4 * X^3 + 5 * X^2 - 7 * X = 17$ (*unsatisfiable*), $\quad \cdots$

- ***Real numbers***
  As above, except that the structure is the real number algebra.
  Examples:
  $X = 1, \quad X * Y < 5, \quad 2 * X = 1$ (*satisfiable*)
  $4 * X^3 + 5 * X^2 - 7 * X = 17$ (*satisfiable*), $\quad \cdots$

Basic Operations on Constraints

- Testing for ***consistency*** or ***satisfiability***: $\mathscr{D} \models \tilde{\exists}\, c$.

  A function *solve*() maps a constraint into $\{true, false, maybe\}$.
  It is ***complete*** if it only returns $\{true, false\}$.

- Obtaining the ***projection*** of a constraint c_0 onto variables $\tilde{x}$ to obtain a constraint c_1 such that $\mathscr{D} \models c_1 \leftrightarrow \exists_{-\tilde{x}}\, c_0$. (It is always possible to take c_1 to be $\exists_{-\tilde{x}}\, c_0$, but the aim is to compute the simplest c_1 with fewest quantifiers. In general it is not possible to eliminate all uses of the existential quantifier.)

- Testing for ***entailment*** of one constraint by another: $\mathscr{D} \models c_0 \rightarrow c_1$. (More generally, we may ask whether a disjunction of constraints is implied by a constraint: $\mathscr{D} \models c_0 \rightarrow \bigvee_{i=1}^{n} c_i$.)

- Detecting that, given a constraint c, there is only ***one value*** that a variable x can take that is consistent with c. ($\mathscr{D} \models c(x, \tilde{z}) \wedge c(y, \tilde{w}) \rightarrow x = y$ or, equivalently, $\mathscr{D} \models \exists z \forall x, \tilde{y}\; c(x, \tilde{y}) \rightarrow x = z$.)

The CLP Scheme

A constraint domain $\mathscr{D}$ defines a programming language $clp(\mathscr{D})$ which is an *instance* of the CLP Scheme.

A $clp(\mathscr{D})$ program consists of a finite number of rules:

$$A_0 \longleftarrow c_1, \cdots, c_n, A_1, \cdots, A_m.$$

where

- $n \geq 0, m \geq 0$,
- the c_i are *constraints* over $\mathscr{X}$
- the A_i are *atoms* over $\mathscr{X}$

clp($\mathcal{FT}$) — PROLOG

Structure: $\langle$Finite trees (Σ), {f, g, ...}, $=\rangle$
Solver: standard unifier

```
add(0, B, C) :- B = C.
add(s(A), B, s(C)) :- add(A, B, C).
fib(0, s(0)).
fib(s(0), s(0)).
fib(s(s(N)), X) :-
    fib(s(N), X1),
    fib(N, X2),
    add(X1, X2, X).
```

Goal: `?- fib(s^5(0), X).`
Answer: X = $s^8(0)$.

Goal: `?- fib(X, s^8(0)).`
Answer: X = $s^5(0)$.

clp($\mathcal{N}$)

Structure: $\langle$Natural numbers, $+, -, <, \leq, =\rangle$
Solver: integer linear inequalities

```
fib(0, 1).
fib(1, 1).
fib(N, X + Y) :-
        N >= 2,
        fib(N - 2, X),
        fib(N - 1, Y).
```

Goal: `?- fib(14, Z).`
Answer: `Z = 610.`

Goal: `?- fib(Z, 610).`
Answer: `Z = 14.`

clp($\mathcal{B}$)

Structure: $\langle \{0,1\}, +, \times, \oplus, \neg, = \rangle$
Solver: boolean unifier

```
adder(In1, In2, In3, Out1, Out2) :-
        In1 ⊕ In2 = X1,
        In1 × In2 = A1,
        X1 ⊕ In3 = Out1,
        In3 × X1 = A2,
        A1 + A2 = Out2.
```

Goal: `?- adder(1, 1, 1, Out1, Out2).`
Answer: `Out1 = 1, Out2 = 1.`

Goal: `?- adder(1, 1, In3, Out1, 1).`
Answer: `Out1 = In3.`

clp($\mathscr{S}$)

Structure: $\langle \text{Strings}(\Sigma),\ .,\ = \rangle$
Solver: equations on strings

```
unit(a).
unit(b).
palindrome(ε).
palindrome(X) :- unit(X).
palindrome(X.Y.X) :-
    unit(X),
    palindrome(Y).
```

Goal: `?- palindrome(a.b.a.b.a).`
Answer: *true*

Goal: `?- palindrome(X.b.a).`
Answers: X = a, X = a.b, X = a.b.a, $\cdots$

clp($\mathscr{R}$)

Structure: $\langle$Real numbers, $+, -, <, \leq, =\rangle$
Solver: real inequalities

```
mortgage(P, T, I, B, M) :-
        T <= 1,
        B = P * T * (P*I/1200 - M).
mortgage(P, T, I, B, M) :-
        T > 1,
        mortgage(P * (1 + I/1200) - M, T - 1, I, B, M).
```

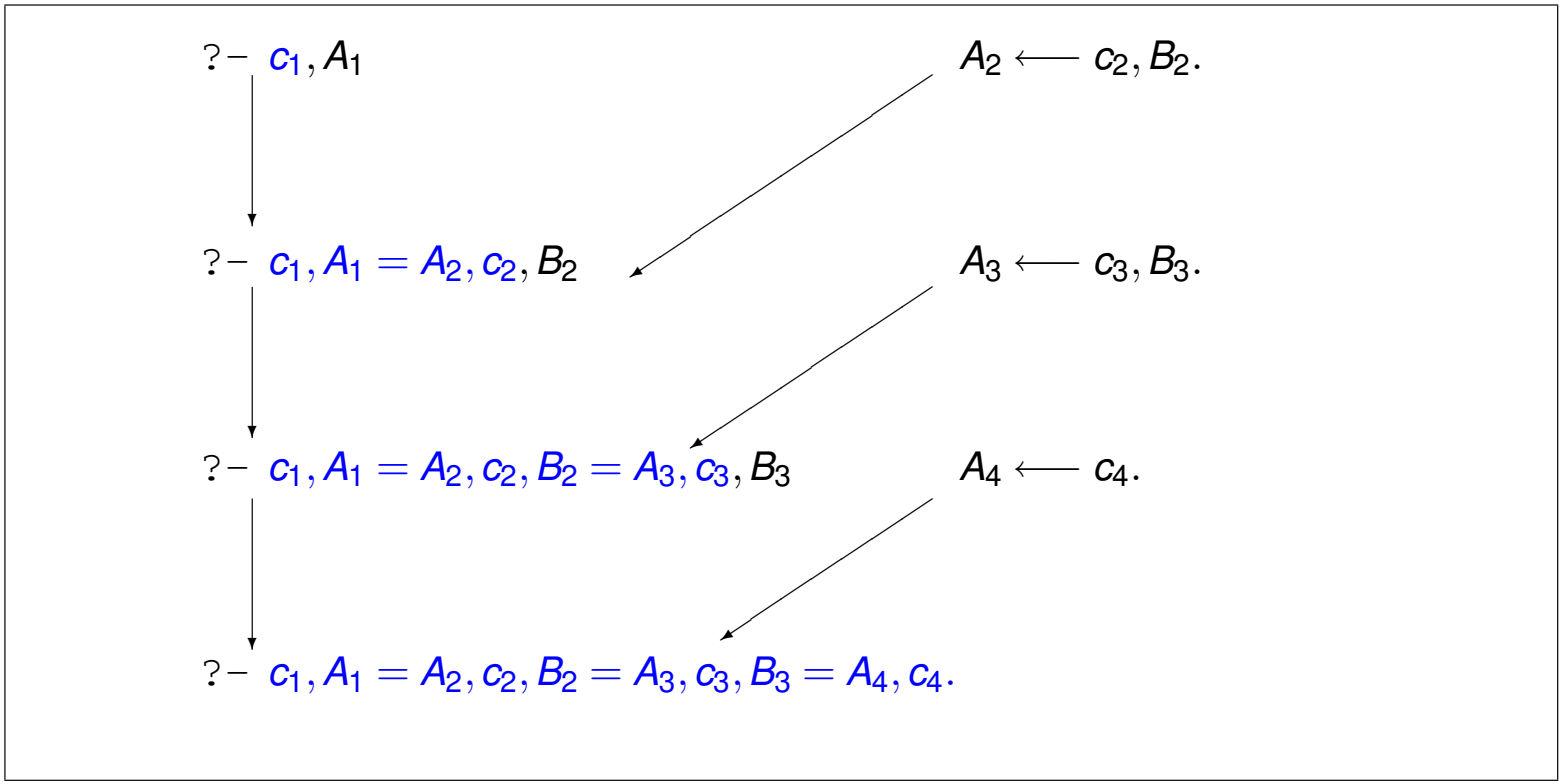
Goal: `?- mortgage(100000, 360, 7.25, 0, M).`
Answer: M = 682.17

Goal: `?- mortgage(P, 360, 7.25, 0, 682.17).`
Answer: P = 100000.

Goal: `?- mortgage(P, 360, 7.25, B, M).`
Answer: P = 0.114*B + 146.59*M

CLP Operational Model

Repeatedly reduce atoms in subgoals, ensuring that all constraints are ***not known to be unsatisfiable***, until the subgoal contains only constraints.



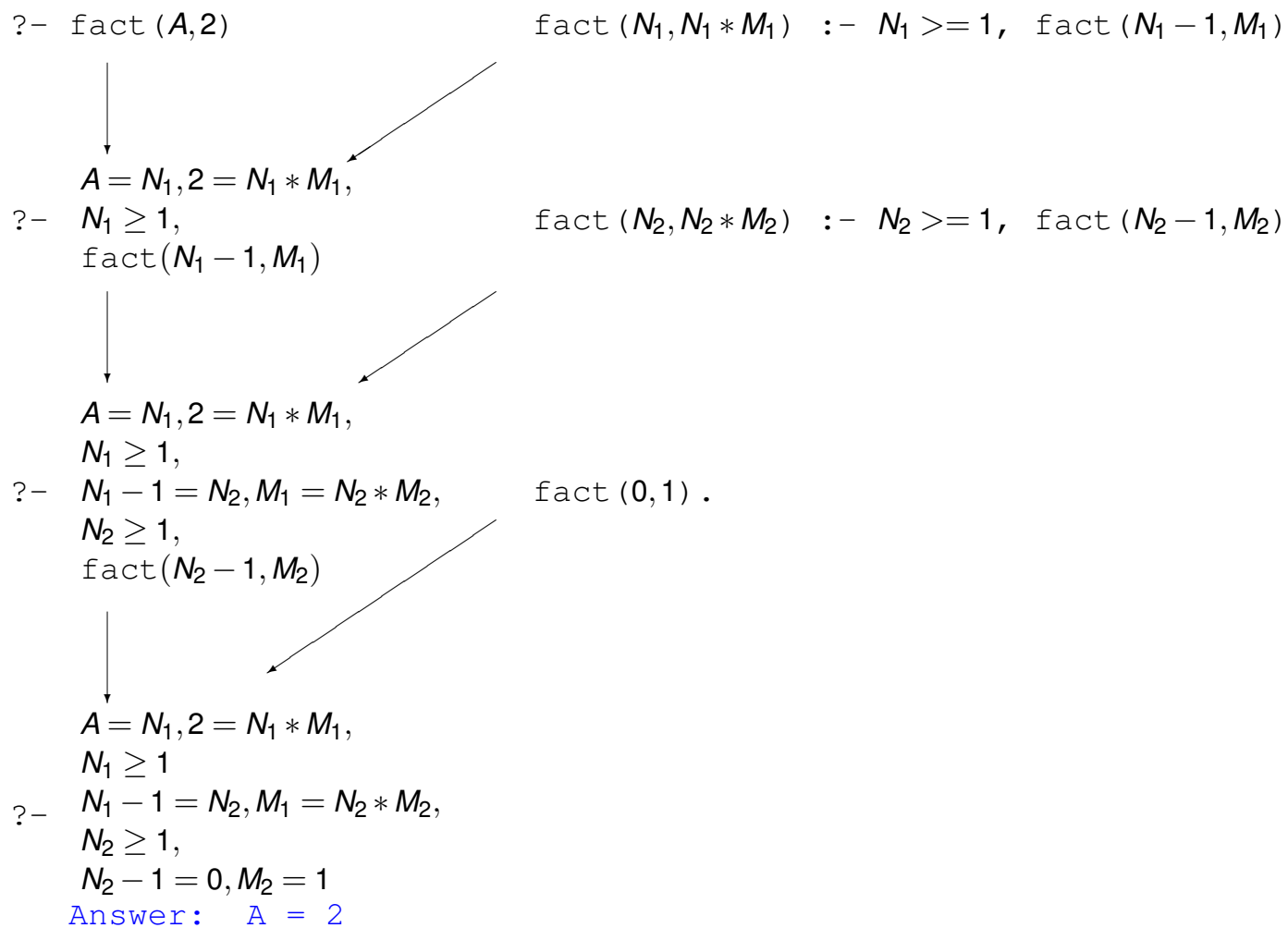
The ***answer constraint*** is obtained by projecting the final constraint onto the variables in the initial goal.

The ***atom selection strategy*** and ***search strategy*** are left unspecified.

Example

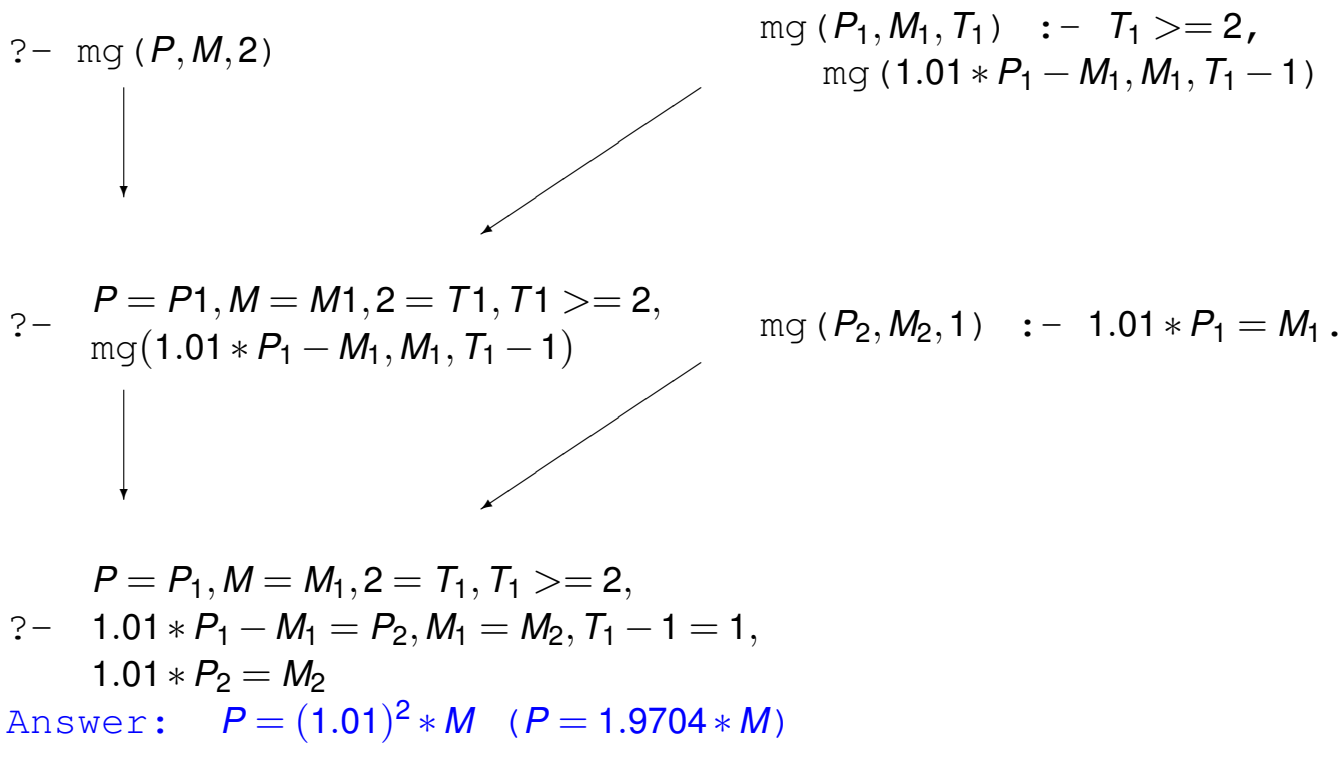
(Rule 1) `fact(0, 1).`

(Rule 2) `fact(N, N * M) :- N >= 1, fact(N - 1, M).`



Example

```
mg(P,M,1) :- 1.01 * P = M.
mg(P,M,T) :- T >= 2, mg(1.01*P - M, M, T-1).
```



Derivations

A ***literal*** is either an atom or a constraint.

A ***goal*** $\mathscr{G}$ is a sequence of atoms or constraints.

A ***state*** is of the form $(\mathscr{G} \mid \mathscr{C})$ where $\mathscr{G}$ is a goal and $\mathscr{C}$ a sequence of constraints.

Suppose $\mathscr{G}_1$ is of the form $L_1, \cdots, L_i, \cdots, L_m$ where L_i is an atom $p(t_1, \cdots, t_n)$. Let R be a rule of the form $p(s_1, \cdots, s_n) :- B$. Then a ***rewriting*** of G using R is the goal
$L_1, \cdots, L_{i-1}, \theta(s_1) = t_1, \cdots, \theta(s_n) = t_n), L_{i+1}, \cdots, L_m$
where θ is a renaming of $s_1, \cdots, s_n$ away from $\mathscr{G}$.

A ***derivation step*** from a state $(\mathscr{G}_1 \mid \mathscr{C}_1)$ to a state $(\mathscr{G}_2 \mid \mathscr{C}_2)$, written $(\mathscr{G}_1 \mid \mathscr{C}_1) \Longrightarrow (\mathscr{G}_2 \mid \mathscr{C}_2)$, is defined as follows. Suppose $\mathscr{G}_1$ is of the form $L_1, \cdots, L_n$.

- L1 is a constraint:
  Then $\mathscr{G}_2$ is $L_2, \cdots, L_n$ and $\mathscr{C}_2$ is $\mathscr{C}_1 \wedge L_1$.
  If $solve(\mathscr{C}_2) \equiv false$, then $(\mathscr{G}_2 \mid \mathscr{C}_2)$ is a *false* state.
- L_1 is an atom:
  Then $\mathscr{C}_2$ is $\mathscr{C}_1$, and $\mathscr{G}_2$ is a rewriting of $\mathscr{G}_1$ at L_1 using some rule R.
  The variables in $\mathscr{G}_2$ are renamed away from $(\mathscr{G}_1 \mid \mathscr{C}_1)$.
  If there is no such R, then then $(\mathscr{G}_2 \mid \mathscr{C}_2)$ is a *false* state.

Note: we have described a ***left-to-right*** selection stategy

Example Derivation

$$fact(2,X) \mid true$$
$$\Downarrow (rule2)$$
$$2 = N, X = N * F, N \geq 1, fact(N-1,F) \mid true$$
$$\Downarrow$$
$$X = N * F, N \geq 1, fact(N-1,F) \mid 2 = N$$
$$\Downarrow$$
$$N \geq 1, fact(N-1,F) \mid 2 = N, X = N * F$$
$$\Downarrow$$
$$fact(N-1,F) \mid 2 = N, X = N * F, N \geq 1$$
$$\Downarrow (rule2)$$
$$N-1 = N', F = N' * F', N' \geq 1, fact(N'-1,F') \mid 2 = N, X = N * F, N \geq 1$$
$$\Downarrow$$
$$\cdots$$
$$\Downarrow$$
$$fact(N'-1,F') \mid 2 = N, X = N * F, N \geq 1, N-1 = N', F = N' * F', N' \geq 1$$
$$\Downarrow (rule1)$$
$$N'-1 = 0, F' = 1 \mid 2 = N, X = N * F, N \geq 1, N-1 = N', F = N' * F', N' \geq 1$$
$$\Downarrow$$
$$\cdots$$
$$\Downarrow$$
$$\square \mid 2 = N, X = N * F, N \geq 1, N-1 = N', F = N' * F', N' \geq 1, N'-1 = 0, F' = 1$$

Successful and Failed Derivations, Derivation Trees

A ***success state*** $(\mathscr{G} \mid \mathscr{C})$ is such that $\mathscr{G} \equiv \square$ and $solve(C) \not\equiv false$.

A ***failed state*** $(\mathscr{G} \mid \mathscr{C})$ is such that $solve(C) equiv false$.

A derivation sequence is ***successful*** if its last state is a success state. The ***answer constraint*** of this sequence is obtained by a projection of constraints in the success state onto the variables in the original goal.

A derivation sequence is ***failed*** if its last state is a failed state.

A ***derivation tree*** for a goal $\mathscr{G}$ and a program P is a tree with states as nodes. The root is $\mathscr{G} \mid true$. Each descendant node is a state that can be reached in a derivation step from its parent. A node which has two or more descendants is called a *choicepoint*.

A goal is ***finitely failed*** if its derivation tree is finite and ***all*** its derivations are failed.

Example of Finite Failure

(Rule 1) `fact(0, 1).`

(Rule 2) `fact(N, N * F) :- N >= 1, fact(N - 1, F).`

Using rule 1:

$$fact(0,2) \mid true$$
$$\Downarrow$$
$$0 = 0, 2 = 1 \mid true$$
$$\Downarrow$$
$$2 = 1 \mid 0 = 0$$
$$\Downarrow$$
$$\square \mid 0 = 0, 2 = 1$$

Using rule 2:

$$fact(0,2) \mid true$$
$$\Downarrow$$
$$0 = N, 2 = N * F, N \geq 1, fact(N-1,F) \mid true$$
$$\Downarrow$$
$$2 = N * F, N \geq 1, fact(N-1,F) \mid 0 = N$$
$$\Downarrow$$
$$N \geq 1, fact(N-1,F) \mid 0 = N, 2 = N * F$$
$$\Downarrow$$
$$\square \mid 0 = N, 2 = N * F, N \geq 1$$

Searching a Derivation Tree

Rule Order

Does not affect the answers, only in the sequence they are discovered. However, it can

- affect how ***quickly*** an answer is found,
- determine if an answer is ***ever*** found

Literal Order

A ***selection derivation step*** $(\mathscr{G}_1 \mid \mathscr{C}_1) \Longrightarrow (\mathscr{G}_2 \mid \mathscr{C}_2)$, is defined as follows. Suppose $\mathscr{G}_1$ is of the form $L_1, \cdots, L_i, \cdots, L_n$ where L_i is ***selected***.

- L_i is a constraint:
  $\mathscr{G}_2$ is $L_2, \cdots, L_n$ and $\mathscr{C}_2$ is $\mathscr{C}_1 \wedge L_i$.
  If $solve(\mathscr{C}_2) \equiv false$, then $(\mathscr{G}_2 \mid \mathscr{C}_2)$ is a *false* state.
- L_i is an atom:
  $\mathscr{C}_2$ is $\mathscr{C}_1$, and $\mathscr{G}_2$ is a rewriting of $\mathscr{G}_1$ at L_i using some rule R.
  The variables in $\mathscr{G}_2$ are renamed away from $(\mathscr{G}_1 \mid \mathscr{C}_1)$. If there is no such R, then then $(\mathscr{G}_2 \mid \mathscr{C}_2)$ is a *false* state.

If the solver were ***complete***, then computing answers is is ***independent*** of literal order. Otherwise, we can get infinite derivations when in fact the constraints are unsatisfiable.

Any answer constraint is ***always correct*** (it never describes an error state) regardless of the solver.

Efficiency

Critial aspects:

- Completeness of the Solver
- Choosing rule order
- Choosing literals

Most CLP systems allow the use of different solvers, dynamic consideration of rule order, and dynamic literal selection.

(More on this later ...)

Modeling Techniques (Arithmetic Examples)

- choice
- iteration
- data structures
- hierarchical modelling

More on CLP($\mathscr{R}$)

The Constraint Domain $\mathscr{R}$:

- structure: finite trees of real numbers
- Constraint symbols: $+ \quad - \quad * \quad / \quad pow \quad sin \quad cos$
- Tree symbols (**functors**): `f/2, cons/2, is_prime/1, ...`
- Constraints:
  (a) aritmetic relations using $= \quad < \quad > \quad \leq \quad \geq$
  (b) term equations, eg: `X = a, f(X, a) = Y, ...`

Solver (incomplete) of $\mathscr{R}$:

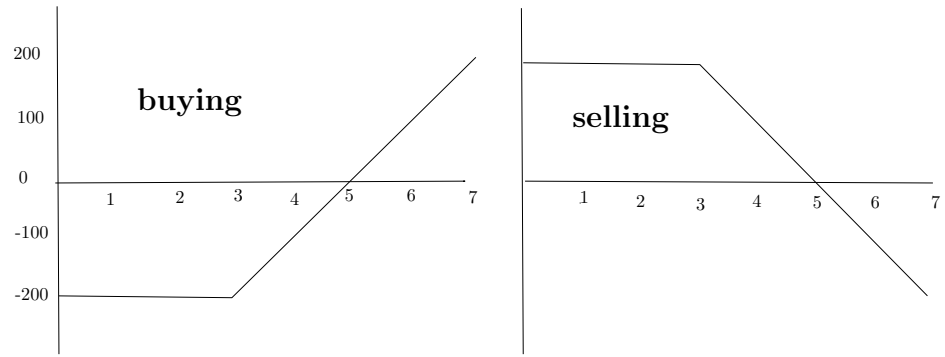
- linear constraints: *interpret in the usual way*
- nonlinear constraints: ***delay*** consideration until it becomes linear (Eg. delay $X * Y = Z$ until one of X or Y becomes known.)
- term equations:
  `f(`t_1`, ... , `t_N`) = f(`u_1`, ... , `u_N`)` is true only if
  $t_1 = u_1, t_2 = u_2, \cdots, t_N = u_N$.
  All other cases are *false*.
- Example: $f(X, X) = f(Y, Z)$ would be equivalent to $Y = Z$.

Modelling Choice - Options Trading

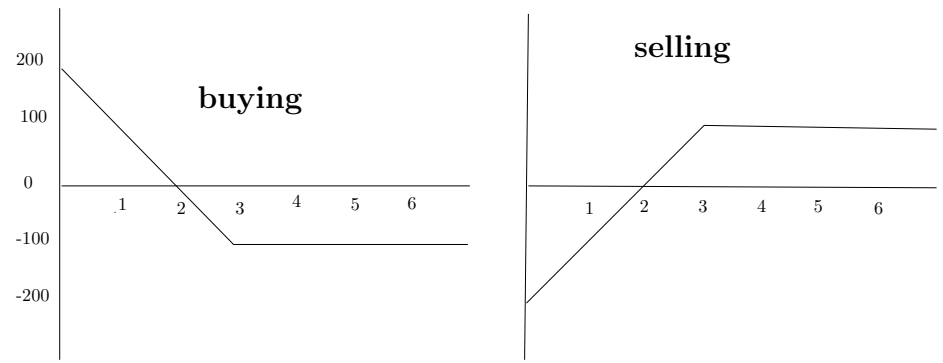
An ***option*** is a contract allowing one to buy (a *call* option) or sell (a *put* option) something (eg: 100 stock shares whose unit price is S) a at a particular price (the exercise price E) at a particular time. The option itself costs C. Then:

$$payoff(S, C, E) = \begin{cases} -C, & \text{if } 0 \leq S \leq E/100 \\ 100 * S - E - C, & \text{if } S \geq E/100 \end{cases}$$

Example: C = 200, E = 300: Payoff for Call Option



Example: C = 100, E = 500: Payoff for Put Option



Options Trading

```
call_option(B, S, C, E, P) :- 0 ≤ S, S ≤ E/100, P = -C*B.
call_option(B, S, C, E, P) :- S ≥ E/100, P = (100*S - E - C) * B.
put_option(B, S, C, E, P) :- 0 ≤ S, S ≤ E/100, P = (E-100*S-C) * B.
put_option(B, S, C, E, P) :- S ≥ E/100, P = -C * B.
```

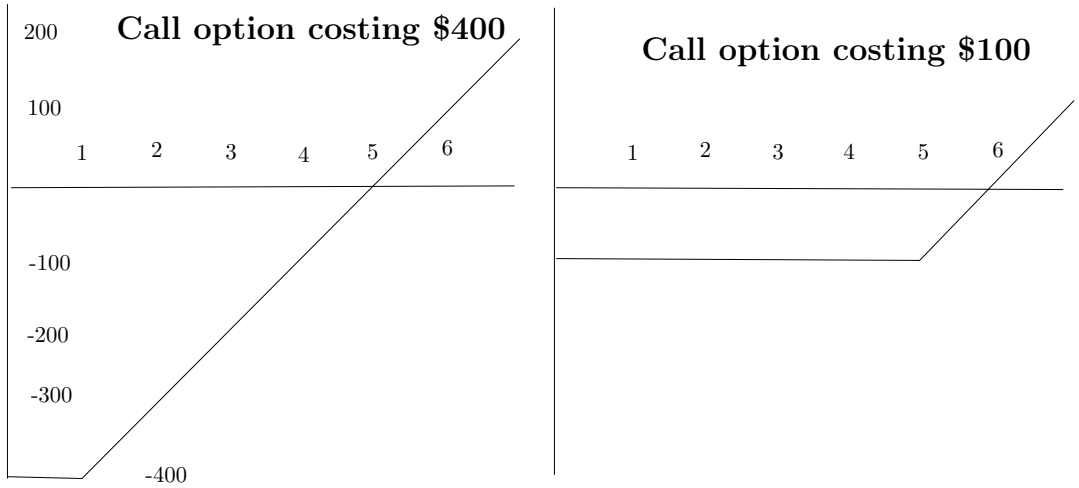
Options trading involves buying and selling complex ***combinations*** of options, in order to satisfy a certain risk profile. Example: a ***butterfly*** combination bets that a stock price remains in a certain range (ex: between $2 and $4) and bounds the loss (ex: never lose more than $100).

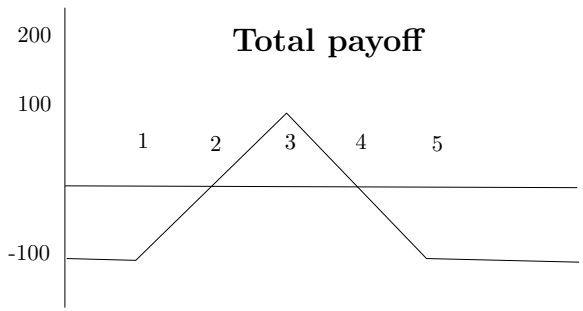
Example:

- buy a call of exercise $500 for $100,
- buy another call of exercise $100 for $400, and
- sell two calls of exercise $300 at $200 each.

```
butterfly(S, P1 + 2*P2 + P3) :-
   Buy = 1, Sell = -1,
   call_option(Buy, S, 100, 500, P1),
   call_option(Sell, S, 200, 300, P2),
   call_option(Buy, S, 400, 100, P3).
```

Butterfly Option





Running `?- P ≥ 0, butterfly(S, P)` returns ***exactly*** two answers:

- $P = 100 * S - 200, 2 \leq S, S \leq 3$
- $P = -100 * S + 400, 3 \leq S, S \leq 4$

Iteration - Mortgage Example

```
mortgage(P, T, I, R, B) :-
      T ≥ 1
      mortgage(P + P*I - R, T - 1, I, R, B).
mortgage(P, T, I, R, B) :- T = 0, B = P.
```

We have seen:

- How much can I borrow?
  $\texttt{mortgage}(P, 3, 0.1, 150, 0) \implies P = 373$
- What is the relationship between P, B, and R?
  $\texttt{mortgage}(P, 10, 0.1, R, B) \implies P = 0.38 * B + 6.14 * R$

How about:

- How much interest?
  $\texttt{mortgage}(120, 2, IR, 0, 80) \implies 80 = (0.1 * IR + 40) * (0.000833 * IR + 1)$

  Note that the CLP system will return this constraint and "maybe" because it cannot determine if this constraint is satisfiable. However, this constraint is ***correct***.
- How much time?
  $\texttt{mortgage}(373, T, 0.1, 150, 0) \implies \cdots ?$

  This wouldn't terminate. Why?

Data Structures - Laplace Example

Finite element modelling is used to approximate a continuous object by a grid of discrete points. Eg. to model temperature on a metal sheet we can use a grid of variables to capture temperature values:

$$\begin{bmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \end{bmatrix}$$

We then require the constraints

$$T_{22} = \frac{T_{12}+T_{21}+T_{23}+T_{32}}{4}$$

$$T_{23} = \frac{T_{13}+T_{22}+T_{24}+T_{33}}{4}$$

In general:

```
rows([_, _]).
rows([H1, H2, H3 | T]):-
   cols(H1, H2, H3),
   rows([H2, H3 | T]).

cols([TL, T, TR | T1], [ML, M, MR | T2], [BL, B, BR | T3]):-
   B + T + ML + MR - 4 * M = 0,
   cols([T,TR|T1],[M,MR|T2],[B,BR|T3]).
cols([_, _], [_, _], [_, _]).
```

Laplace Example

```
?- X = [[0,0,0,0,0,0,0,0,0,0,0],
   [100,_,_,_,_,_,_,_,_,_,100],
   [100,_,_,_,_,_,_,_,_,_,100],
   [100,_,_,_,_,_,_,_,_,_,100],
   [100,_,_,_,_,_,_,_,_,_,100],
   [100,_,_,_,_,_,_,_,_,_,100],
   [100,_,_,_,_,_,_,_,_,_,100],
   [100,_,_,_,_,_,_,_,_,_,100],
   [100,_,_,_,_,_,_,_,_,_,100],
   [100,_,_,_,_,_,_,_,_,_,100],
   [100,100,100,100,100,100,100,100,100,100,100]], rows(X).
```

$\Longrightarrow$

```
X =
   [[0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]
   [100.00 51.11 32.52 24.56 21.11 20.12 21.11 24.56 32.52 51.11 100.00]
   [100.00 71.91 54.41 44.63 39.74 38.26 39.74 44.63 54.41 71.91 100.00]
   [100.00 82.12 68.59 59.80 54.97 53.44 54.97 59.80 68.59 82.12 100.00]
   [100.00 87.97 78.03 71.00 66.90 65.56 66.90 71.00 78.03 87.97 100.00]
   [100.00 91.71 84.58 79.28 76.07 75.00 76.07 79.28 84.58 91.71 100.00]
   [100.00 94.30 89.29 85.47 83.10 82.30 83.10 85.47 89.29 94.30 100.00]
   [100.00 96.20 92.82 90.20 88.56 88.00 88.56 90.20 92.82 96.20 100.00]
   [100.00 97.67 95.59 93.96 92.93 92.58 92.93 93.96 95.59 97.67 100.00]
   [100.00 98.89 97.90 97.12 96.63 96.46 96.63 97.12 97.90 98.89 100.00]
   [100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00]]
```

Laplace Example

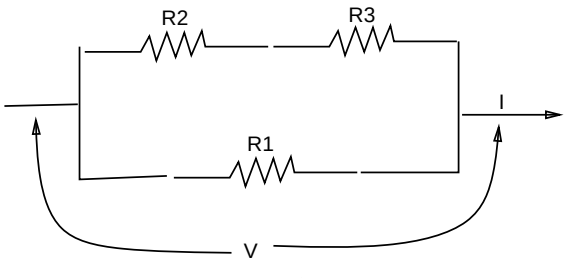
```
?- rows([
   [B11, B12, B13, B14],
   [B21, M22, M23, B24],
   [B31, M32, M33, B34],
   [B41, B42, B43, B44]
]).
```

$\Longrightarrow$

```
B12 = -B21 - 4*B31 + 16*M32 - 8*M33 + B34 - 4*B42 + B43
B13 = -B24 + B31 - 8*M32 + 16*M33 - 4*B34 + B42 - 4*B43
M22 = -B31 + 4*M32 - M33 - B42
M23 = -M32 + 4*M33 - B34 - B43
```

Hierarchical Modelling - Circuits

```
circuit(resistor(R), V, I) :- V = I * R.
circuit(series(N1, N2), V, I) :-
   V = V1 + V2,
   circuit(N1, V1, I), circuit(N2, V2, I).
circuit(parallel(N1, N2), V, I) :-
   I = I1 + I2,
   circuit(N1, V, I1), circuit(N2, V, I2).
```



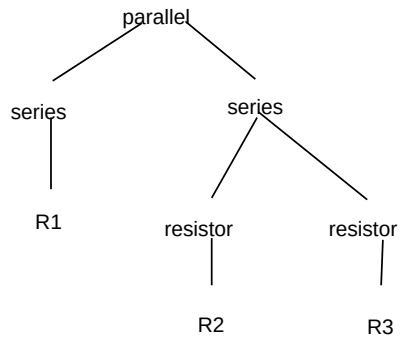


Figure: Circuit

```
?- circuit(
     parallel(resistor(R1), series(resistor(R2), resistor(R3))),
     V, I).
```

Answer: $V = I2 * (R2 + R3), V = (I - I2) * R1$

```
?- R1 = 4, R2 = 5, R3 = 6,
   circuit(
     parallel(resistor(R1), series(resistor(R2), resistor(R3))),
     V, I).
```

Answer: $V = 2.9333 * I$

Circuit with Complex Numbers

```
c_equal(c(Re, Im), c(Re, Im)).
c_add(c(Re1, Im1), c(Re2, Im2), c(Re1 + Re2, Im1 + Im2)).
c_mult(c(Re1, Im1), c(Re2, Im2), c(Re3, Im3)) :-
   Re3 = Re1 * Re2 - Im1 * Im2,
   Im3 = Re1 * Im2 + Re2 * Im1.

circuit(resistor(R), V, I, W) :- c_mult(V, I, c(R, 0)).
circuit(inductor(L), V, I, W) :- c_mult(V, I, c(0, W * L)).
circuit(capacitor(C), V, I, W) :-
   c_mult(V, I, c(0, -1 / (W * C))).
circuit(series(N1, N2), V, I, W) :-
   c_equal(I, I1), c_equal(I, I2),
   c_add(V, V1, V2),
   circuit(N1, V1, I1, W),
   circuit(N2, V2, I2, W).
circuit(parallel(N1, N2), V, I, W) :-
   c_equal(V, V1), c_equal(V, V2),
   c_add(I, I1, I2),
   V = V1, V = V2,
   I = I1 + I2,
   circuit(N1, V1, I1, W),
   circuit(N2, V2, I2, W).
```

Can we program LP to be CLP?

CLP: `p(X, Y, Z) :- ··· X + Y = Z ···`

LP
```
p(X, Y, Z) :- ··· add(X, Y, Z) ···
add(0, N, N).
add(s(N), M, s(K)) :- add(N, M, K).
```

Goal: `?- p(N, M, K), p(N, M, K+1)`

- fails in CLP
- runs forever in LP

The essential difference:

`?- add(N, M, K)` does not return a ***complete*** representation of the set of solutions.

Summary of CLP Introduction

- The CLP Scheme
- Constraint Solving, complete and incomplete
- CLP evaluation
- Modelling with arithmetic constraints
- ***NEXT:*** controlling search