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:*


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(\(\mathcal{R}\))
- CHIP
- \(ECL^iPS^e\)
- 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 \( \mathcal{L} \) of
  - variables (\( X, Y, \cdots \))
  - function symbols \( \Sigma \) (\( eg: +, -, \text{fib}(.), \cdots \))
  - constraint symbols \( \Pi \) (\( eg: <, \text{is\_prime}(.), \cdots \))
  - 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 \( X + 2 \times Y - 1 \).
  - 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** \( \mathcal{D} \)
  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 $\mathcal{D}$: the set of terms constructible from $\Sigma$
  - Function symbols $\Sigma$: any collection of $n$-ary function symbols, for all $n \geq 0$
  - Constraint symbols $\Pi$: $= \neq$

- **Integers**
  - Structure $\mathcal{D}$: the integers with addition, multiplication, order
  - Function symbols $\Sigma$: $- 1 \ 0 \ + 1 \ + \ *$
  - Constraint symbols $\Pi$: $= < \leq$

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

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

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

  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 $\mathcal{D} \models c_1 \leftrightarrow \exists_{\tilde{x}} c_0$. (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: $\mathcal{D} \models c_0 \rightarrow c_1$. (More generally, we may ask whether a disjunction of constraints is implied by a constraint: $\mathcal{D} \models c_0 \rightarrow \bigvee_{i=1}^{n} c_i$.)

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

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

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

\[
A_0 \leftarrow c_1, \cdots, c_n, A_1, \cdots, A_m.
\]

where

- \( n \geq 0, m \geq 0 \),
- the \( c_i \) are *constraints* over \( \mathcal{X} \)
- the \( A_i \) are *atoms* over \( \mathcal{X} \)
Structure: \langle \text{Finite trees } (\Sigma), \{f, g, \ldots\}, = \rangle \\
Solver: standard unifier

\begin{verbatim}
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).
\end{verbatim}

Goal: \( ?- \text{fib}(s^5(0), X). \) \\
Answer: \( X = s^8(0). \)

Goal: \( ?- \text{fib}(X, s^8(0)). \) \\
Answer: \( X = s^5(0). \)
Structure: ⟨Natural numbers, +, −, <, ≤, =⟩
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.
Structure: $\langle\{0, 1\}, +, \times, \oplus, \neg, =\rangle$
Solver: boolean unifier

adder(In1, In2, In3, Out1, Out2) :-
    In1 $\oplus$ In2 $=$ X1,
    In1 $\times$ In2 $=$ A1,
    X1 $\oplus$ In3 $=$ Out1,
    In3 $\times$ 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.
Structure: \( \langle \text{Strings}(\Sigma), \cdot, = \rangle \)
Solver: equations on strings

\begin{verbatim}
unit(a).
unit(b).
palindrome(\epsilon).
palindrome(X) :- unit(X).
palindrome(X.Y.X) :-
    unit(X),
    palindrome(Y).
\end{verbatim}

Goal: \(?-\) palindrome(a.b.a.b.a).
Answer: \text{true}

Goal: \(?-\) palindrome(X.b.a).
Answers: \( X = a, X = a.b, X = a.b.a, \cdots \)
Structure: \( \langle \text{Real numbers, } +, -, <, \leq, = \rangle \)
Solver: real inequalities

```prolog
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) \( \text{fact}(0, 1) \).
(Rule 2) \( \text{fact}(N, N \times M) \) :- \( N \geq 1 \), \( \text{fact}(N - 1, M) \).

?- \( \text{fact}(A, 2) \)

\( \text{fact}(N_1, N_1 \times M_1) \) :- \( N_1 \geq 1 \), \( \text{fact}(N_1 - 1, M_1) \)

?- \( N_1 \geq 1 \),

\( \text{fact}(N_1 - 1, M_1) \)

\( A = N_1, 2 = N_1 \times M_1 \),

\( A = N_1, 2 = N_1 \times M_1 \),

\( N_1 \geq 1 \),

?- \( N_1 - 1 = N_2, M_1 = N_2 \times M_2 \),

\( N_2 \geq 1 \),

\( \text{fact}(N_2 - 1, M_2) \)

\( \text{fact}(0, 1) \).

\( A = N_1, 2 = N_1 \times M_1 \),

\( N_1 \geq 1 \)

?- \( N_1 - 1 = N_2, M_1 = N_2 \times M_2 \),

\( N_2 \geq 1 \),

\( N_2 - 1 = 0, M_2 = 1 \)

Answer: \( A = 2 \)
Example

\[ mg(P, M, 1) :\ - 1.01 \times P = M. \]
\[ mg(P, M, T) :\ - T \geq 2, \ mg(1.01 \times P - M, M, T-1). \]

?- \( mg(P, M, 2) \)

\[ P = P_1, M = M_1, 2 = T_1, T_1 \geq 2, \]
\[ mg(1.01 \times P_1 - M_1, M_1, T_1 - 1) \]

?- \( P_1 = P_2, M_1 = M_2, T_1 = T_1 - 1 = 1, \)
\[ 1.01 \times P_2 = M_2 \]

Answer: \( P = (1.01)^2 \times M \) (\( P = 1.9704 \times M \))
Derivations

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

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

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

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

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

- **L1 is a constraint:**
  Then $\mathcal{G}_2$ is $L_2, \ldots, L_n$ and $\mathcal{C}_2$ is $\mathcal{C}_1 \land L_1$.
  If $\text{solve}(\mathcal{C}_2) \equiv \text{false}$, then $(\mathcal{G}_2 \mid \mathcal{C}_2)$ is a **false** state.

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

Note: we have described a **left-to-right** selection strategy
Example Derivation

\[
\begin{align*}
\text{fact}(2, X) & \mid \text{true} \\
\downarrow (\text{rule2}) \quad & \\
2 = N, X = N \cdot F, N \geq 1, \text{fact}(N - 1, F) & \mid \text{true} \\
\downarrow & \\
X = N \cdot F, N \geq 1, \text{fact}(N - 1, F) & \mid 2 = N \\
\downarrow & \\
N \geq 1, \text{fact}(N - 1, F) & \mid 2 = N, X = N \cdot F \\
\downarrow & \\
\text{fact}(N - 1, F) & \mid 2 = N, X = N \cdot F, N \geq 1 \\
\downarrow (\text{rule2}) \quad & \\
N - 1 = N', F = N' \cdot F', N' \geq 1, \text{fact}(N' - 1, F') & \mid 2 = N, X = N \cdot F, N \geq 1 \\
\downarrow & \\
\ldots & \\
\downarrow & \\
\text{fact}(N' - 1, F') & \mid 2 = N, X = N \cdot F, N \geq 1, N - 1 = N', F = N' \cdot F', N' \geq 1 \\
\downarrow (\text{rule1}) \quad & \\
N' - 1 = 0, F' = 1 & \mid 2 = N, X = N \cdot F, N \geq 1, N - 1 = N', F = N' \cdot F', N' \geq 1 \\
\downarrow & \\
\ldots & \\
\downarrow & \\
\lozenge & \mid 2 = N, X = N \cdot F, N \geq 1, N - 1 = N', F = N' \cdot F', N' \geq 1, N' - 1 = 0, F' = 1
\end{align*}
\]
Successful and Failed Derivations, Derivation Trees

A *success state* \((G \mid C)\) is such that \(G \equiv \Box\) and \(solve(C) \not\equiv false\).

A *failed state* \((G \mid C)\) 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 \(G\) and a program \(P\) is a tree with states as nodes. The root is \(G \mid true\). 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) \quad \text{fact}(0, 1).

(Rule 2) \quad \text{fact}(N, N \times F) :- N \geq 1, \text{fact}(N - 1, F).

Using rule 1:

Using rule 2:

\[
\begin{align*}
\text{fact}(0, 2) & \mid \text{true} \\
\downarrow & \\
0 = 0, 2 = 1 & \mid \text{true} \\
\downarrow & \\
2 = 1 & \mid 0 = 0 \\
\downarrow & \\
\Box & \mid 0 = 0, 2 = 1
\end{align*}
\]
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* $(G_1 \mid C_1) \rightarrow (G_2 \mid C_2)$, is defined as follows. Suppose $G_1$ is of the form $L_1, \cdots, L_i, \cdots, L_n$ where $L_i$ is *selected*.

- $L_i$ is a constraint:
  $G_2$ is $L_2, \cdots, L_n$ and $C_2$ is $C_1 \land L_i$.
  If $\text{solve}(C_2) \equiv \text{false}$, then $(G_2 \mid C_2)$ is a *false* state.

- $L_i$ is an atom:
  $C_2$ is $C_1$, and $G_2$ is a rewriting of $G_1$ at $L_i$ using some rule $R$.
  The variables in $G_2$ are renamed away from $(G_1 \mid C_1)$. If there is no such $R$, then then $(G_2 \mid C_2)$ is a *false* state.

If the solver were *complete*, then computing answers 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

Critical 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($\mathbb{R}$)

The Constraint Domain $\mathbb{R}$:

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

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

- linear constraints: *interpret in the usual way*
- nonlinear constraints: *delay* consideration until it becomes linear (Eg. delay $X \ast Y = Z$ until one of $X$ or $Y$ becomes known.)
- term equations:
  \[ f(t_1, \ldots, t_N) = f(u_1, \ldots, u_N) \] is true only if
  \[ t_1 = u_1, t_2 = u_2, \ldots, t_N = u_N. \]
  All other cases are false.
- Example: $f(X, X) = f(Y, Z)$ would be equivalent to $Y = Z$. 
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 $S$) at a particular price (the exercise price $E$) at a particular time. The option itself costs $C$. Then:

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

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 \leq S, S \leq E/100, P = -C \times B.
call_option(\(B, S, C, E, P\)) :- S \geq E/100, P = (100 \times S - E - C) \times B.
put_option(\(B, S, C, E, P\)) :- 0 \leq S, S \leq E/100, P = (E - 100 \times S - C) \times B.
put_option(\(B, S, C, E, P\)) :- S \geq E/100, P = -C \times 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.

\[\text{butterfly}(S, P_1 + 2 \times P_2 + P_3) :-\]
  \[\text{Buy} = 1, \text{Sell} = -1,\]
  \[\text{call_option}(\text{Buy}, S, 100, 500, P_1),\]
  \[\text{call_option}(\text{Sell}, S, 200, 300, P_2),\]
  \[\text{call_option}(\text{Buy}, S, 400, 100, P_3).\]
Running $\mathcal{P} \geq 0$, \textit{butterfly}(S, \mathcal{P}) returns \textit{exactly} two answers:

- $\mathcal{P} = 100 \times S - 200, 2 \leq S, S \leq 3$
- $\mathcal{P} = -100 \times S + 400, 3 \leq S, S \leq 4$
Iteration - Mortgage Example

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

We have seen:

- How much can I borrow?
  mortgage(P, 3, 0.1, 150, 0) $$\Rightarrow$$ P = 373

- What is the relationship between P, B, and R?
  mortgage(P, 10, 0.1, R, B) $$\Rightarrow$$ P = 0.38 * B + 6.14 * R

How about:

- How much interest?
  mortgage(120, 2, IR, 0, 80) $$\Rightarrow$$ 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?
  mortgage(373, T, 0.1, 150, 0) $$\Rightarrow$$ …?
  This wouldn’t terminate. Why?
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],[100,100,100,100,100,100,100,100]], rows(X).

⇒

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 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]]
Laplace Example

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

⇒

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).

?- 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

\[
\begin{eqnarray*}
c_{\text{equal}}(c(Re, Im), c(Re, Im)) & \cdot \\
c_{\text{add}}(c(Re1, Im1), c(Re2, Im2), c(Re1 + Re2, Im1 + Im2)) & \cdot \\
c_{\text{mult}}(c(Re1, Im1), c(Re2, Im2), c(Re3, Im3)) & :\neg \\
& \text{Re3} = \text{Re1} \times \text{Re2} - \text{Im1} \times \text{Im2}, \\
& \text{Im3} = \text{Re1} \times \text{Im2} + \text{Re2} \times \text{Im1}. \\
\end{eqnarray*}
\]

\[
\begin{eqnarray*}
circuit\left(\text{resistor}(R), V, I, W\right) & :\neg c_{\text{mult}}(V, I, c(R, 0)). \\
circuit\left(\text{inductor}(L), V, I, W\right) & :\neg c_{\text{mult}}(V, I, c(0, W \times L)). \\
circuit\left(\text{capacitor}(C), V, I, W\right) & :\neg \\
& c_{\text{mult}}(V, I, c(0, -1 / (W \times C))). \\
circuit\left(\text{series}(N1, N2), V, I, W\right) & :\neg \\
& c_{\text{equal}}(I, I1), c_{\text{equal}}(I, I2), \\
& c_{\text{add}}(V, V1, V2), \\
& \text{circuit}(N1, V1, I1, W), \\
& \text{circuit}(N2, V2, I2, W). \\
circuit\left(\text{parallel}(N1, N2), V, I, W\right) & :\neg \\
& c_{\text{equal}}(V, V1), c_{\text{equal}}(V, V2), \\
& c_{\text{add}}(I, I1, I2), \\
& V = V1, V = V2, \\
& I = I1 + I2, \\
& \text{circuit}(N1, V1, I1, W), \\
& \text{circuit}(N2, V2, I2, W). \\
\end{eqnarray*}
\]
Can we program LP to be CLP?

CLP:
\[
p(X, Y, Z) :- \cdots X + Y = Z \cdots
\]

LP
\[
p(X, Y, Z) :- \cdots \text{add}(X, Y, Z) \cdots
\]
\[
\text{add}(0, N, N).
\]
\[
\text{add}(s(N), M, s(K)) :- \text{add}(N, M, K).
\]

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

- fails in CLP
- runs forever in LP

The essential difference:
\(?- \text{add}(N, M, K)\) 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