Chapter 5: Simple Modelling

Where we examine various modelling abilities of CLP languages

Simple Modelling

- Modelling
- Modelling Choice
- Iteration
- Optimization
Modelling

- Choose the variables that will be used to represent the parameters of the problem (this may be straightforward or difficult)
- Model the idealized relationships between these variables using the primitive constraints available in the domain

Modelling Example

A traveller wishes to cross a shark infested river as quickly as possible. Reasoning the fastest route is to row straight across and drift downstream, where should she set off

width of river: $W$
speed of river: $S$
set of position: $P$
rowing speed: $R$
Modelling Example

Reason: in the time the rower rows the width of the river, she floats downstream distance given by river speed by time. Hence model

\[
\]

Suppose she rows at 1.5m/s, river speed is 1m/s and width is 24m.

\[
\text{river}(24, 1, 1.5, P).
\]

Has unique answer \( P = 16 \)

Modelling Example Cont.

If her rowing speed is between 1 and 1.3 m/s and she cannot set out more than 20 m upstream can she make it?

\[
1 <= R, R <= 1.3, P <= 20, \quad \text{river}(24, 1, R, P).
\]

Flexibility of constraint based modelling!
Modelling Choice

- Multiple rules allow modelling relationships that involve choice
- E.g. tables of data using multiple facts.

father(jim, edward).
mother(maggy, fi).
father(jim, maggy).
mother(fi, lillian).
father(edward, peter).
father(edward, helen).
father(edward, kitty).
father(bill, fi).

Choice Examples

The goal father(edward, X) finds children of Edward. Answers:

- X = peter,
- X = helen,
- X = kitty

The goal mother(X, fi) finds the mother of Fi. Answers:

- X = maggy
**Choice Examples**

We can define other predicates in terms of these:

- `parent(X,Y) :- father(X,Y).`
- `parent(X,Y) :- mother(X,Y).`
- `sibling(X,Y) :- parent(Z,X), parent(Z,Y), X != Y.`
- `cousin(X,Y) :- parent(Z,X), sibling(Z,T), parent(T,Y).`

The goal `cousin(peter, X)` has a single answer: \( X = fi \)

**More Complicated Choice**

- A **call option** gives the holder the right to buy 100 shares at a fixed price \( E \).
- A **put option** gives the holder the right to sell 100 shares at a fixed price \( E \).
- A **pay off** of an option is determined by cost \( C \) and current share price \( S \).
- e.g. call cost $200 exercise $300
  - stock price $2, don’t exercise payoff = -$200
  - stock price $7, exercise payoff = $200
Options Trading

- Call: $C = 200$, $E = 300$
- Put: $C = 100$, $E = 300$

Butterfly strike:
- Buy call at 500
- And 100 sell 2 puts at 300

Modelling Functions

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

Model a function with $n$ arguments as a predicate with $n+1$ arguments. Tests are constraints, and result is an equation.

```prolog
buy_call_payoff(S,C,E,P) :-
  0 <= S, S <= E/100, P = -C.
buy_call_payoff(S,C,E,P) :-
  S >= E/100, P = 100*S - E - C.
```
Modelling Options

Add an extra argument \(B=1\) (buy), \(B = -1\) (sell)

\[
\text{call\_option}(B, S, C, E, P) :-
0 <= S, S <= E/100, P = -C \times B.
\]

\[
\text{call\_option}(B, S, C, E, P) :-
S >= E/100, P = (100*S - E - C)*B.
\]

The goal (the original call option question)

\[
\text{call\_option}(1, 7, 200, 300, P)
\]

has answer \(P = 200\)

Using the Model

\[
\text{butterfly}(S, P_1 + 2*P_2 + P_3) :-
\]

\[
\quad \text{Buy = 1, Sell = -1,}
\quad \text{call\_option(Buy, S, 100, 500, P1),}
\quad \text{put\_option(Sell, S, 200, 300, P2),}
\quad \text{call\_option(Buy, S, 400, 100, P3).}
\]

Defines the relationship in previous graph

\[
P >= 0, \text{butterfly}(S,P).
\]

has two answers

\[
P = 100S - 200 \land 2 \leq S \land S \leq 3
\]

\[
P = -100S + 400 \land 3 \leq S \land S \leq 4
\]
Modelling Iteration

- Natural model may be iterating over some parameter
- CLP languages have no direct iteration constructs (for, while) instead recursion

 Iteration Example

Mortgage: principal $P$, interest rate $I$, repayment $R$ and balance $B$ over $T$ periods

Simple Interest: $B = P + P \times I - R$

Relationship over 3 periods:

$P_1 = P + P \times I - R \land$
$P_2 = P_1 + P_1 \times I - R \land$
$P_3 = P_2 + P_2 \times I - R \land$
$B = P_3$

Number of constraints depend on the variable $T$
Reason Recursively

Zero time periods then \( B = P \)
else new princ. \( P + P^*I - R \) and new time \( T-1 \)

\[
\text{mortgage}(P,T,I,R,B) :\quad T = 0, \quad B = P. \quad (M1)
\]
\[
\text{mortgage}(P,T,I,R,B) :\quad T >= 1,
\quad NP = P + P^*I - R, \quad NT = T - 1, \quad (M2)
\quad \text{mortgage}(NP,NT,I,R,B).
\]

Example Derivation
\[
\langle \text{mortgage}(P,3,I,R,B)|true\rangle
\]
\[
\Downarrow M2
\]
\[
\langle \text{mortgage}(P_1,2,I,R,B)|P_1 = P + P^*I - R\rangle
\]
\[
\Downarrow M2
\]
\[
\langle \text{mortgage}(P_1,1,I,R,B)|P_1 = P + P^*I - R \land P_2 = P_1 + P_1^*I - R \rangle
\]
\[
\Downarrow M2
\]
\[
\langle \text{mortgage}(P_2,0,I,R,B)|P_1 = P + P^*I - R \land P_2 = P_1 + P_1^*I - R \land \\
\quad P_3 = P_2 + P_2^*I - R \land B = P_1\rangle
\]
\[
\Downarrow M1
\]
\[
\langle ||P_1 = P + P^*I - R \land P_2 = P_1 + P_1^*I - R \land \\
\quad P_3 = P_2 + P_2^*I - R \land B = P_1\rangle
\]
Translating Iteration

- Novice CLP programmers may have difficulty defining recursive relationships
- Give a procedural definition
- translate iteration to recursion
- translate tests and assignments to constraints

Translation Example

Pseudo C code for the mortgage problem

```c
float mg1(float P, int T, float I, float R) {
    while (T >= 1) {
        P = P + P * I - R;
        T = T - 1;
    }
    return P;
}
```

Remove the while loop using recursion
Translation Example

Pseudo C code for the mortgage problem

```c
float mg2(float P, int T, float I, float R)
{
    if (T >= 1) {
        P = P + P * I - R;
        T = T - 1;
        return mg2(P, T, I, R); }
    else
        return P;
}
```

Make each variable only take one value

Translation Example

Pseudo C code for the mortgage problem

```c
float mg3(float P, int T, float I, float R)
{
    if (T >= 1) {
        NP = P + P * I - R;
        NT = T - 1;
        return mg3(NP, NT, I, R); }
    else
        return P;
}
```

Replace the function with a procedure answer by ref.

Translation Example

Pseudo C code for the mortgage problem

```c
float mg2(float P, int T, float I, float R)
{
    if (T >= 1) {
        P = P + P * I - R;
        T = T - 1;
        return mg2(P, T, I, R); }
    else
        return P;
}
```

Make each variable only take one value

Translation Example

Pseudo C code for the mortgage problem

```c
float mg3(float P, int T, float I, float R)
{
    if (T >= 1) {
        NP = P + P * I - R;
        NT = T - 1;
        return mg3(NP, NT, I, R); }
    else
        return P;
}
```

Replace the function with a procedure answer by ref.
Translation Example

Pseudo C code for the mortgage problem

```c
mg4(float P, int T, float I, float R, float *B)
{
  if (T >= 1) {
    NP = P + P * I - R;
    NT = T - 1;
    mg4(NP, NT, I, R, B);
  } else
    *B = P;
}
```

Replace tests and assignments by constraints

Translation Example

Pseudo C code for the mortgage problem

```prolog
mg(P, T, I, R, B) :-
  T >= 1,
  NP = P + P * I - R,
  NT = T - 1,
  mg(NP, NT, I, R, B).
mg(P, T, I, R, B) :- T = 0, (note extra)
  B = P.
```
**Why Constraints and not C**

- Both programs can answer the goal
  - `mortgage(500, 3, 10/100, 150, B).`
- But the CLP program can answer
  - `mortgage(P, 3, 10/100, 150, 0).`
  
  \[ P = 373.028 \]

- an even the goal
  - `mortgage(P, 3, 10/100, R, B).
  \[ P = 0.38553B + 6.14457R \]

---

**Optimization**

- Many problems require a “best” solution
  - minimization literal: `minimize(G,E)`
- answers are the answers of goal G which minimize expression E (in context of state)
**Optimization Examples**

\[
p(X,Y) := X = 1. \\
p(X,Y) :- Y = 1.
\]
\[
x >= 0, y >= 0, \text{minimize}(p(X,Y), X+Y)
\]
Answers: \(X = 1 \land Y = 0\) and \(X = 0 \land Y = 1\)
\[
x >= 0, x >= y, \text{minimize}(\text{true}, X-Y)
\]
Answer: \(X >= 0 \land X = Y\)
minimize(\text{butterfly}(S,P), -P)
Answer: \(S = 3 \land P = 100\)

**Optimization Evaluation**

\- A valuation \(v\) is a **solution** to a state if it is a solution of some answer to the state

\- **minimization derivation step:** \(<G1 \mid C1>\) to \(<G2 \mid C2>\) where \(G1 = L1, L2, ..., Lm\)
  \- \(L1\) is **minimize**(\(G, E\))
  \- exists solution \(v\) of \(<G \mid C1>\) with \(v(E) = m\)
  and for all other sols \(w\), \(m <= w(E)\)
  \- \(G2\) is \(G, L2, ..., Lm\) and \(C2\) is \(C1 \land E = m\)
  \- else \(G2\) is [] and \(C2\) is false
**Optimization Example**

\[ X \geq 0, \min \{ X \geq Y, X - Y \} \]

\[
\begin{align*}
&\{ X \geq 0, \min \{ X \geq Y, X - Y \} \mid true \} \\
&\downarrow \\
&\{ \min \{ X \geq Y, X - Y \} \mid X \geq 0 \} \\
&\downarrow \\
&\{ X \geq Y \mid X \geq 0 \\wedge X - Y = 0 \} \\
&\downarrow \\
&\{ \exists \mid X \geq 0 \wedge X \geq Y \}
\end{align*}
\]

Simplified: \( X \geq 0 \wedge X = Y \)

Minimum value of \( X - Y \) is 0 e.g. \( \{ X \mapsto 3, Y \mapsto 3 \} \)

**Optimization**

Optimization doesn't only have to be at the goal

\[
\text{straddle}(S, C_1 + C_2, E, P_1 + P_2) :- \\
\text{Buy} = 1, \\
\text{call_option}(\text{Buy}, S, C_1, E, P_1), \\
\text{put_option}(\text{Buy}, S, C_2, E, P_2).
\]

\[
\text{best_straddle}(C, E, P) :- \\
\text{minimize}(\text{straddle}(S, C, E, P), -P).
\]
Simple Modelling Summary

- Converting problem constraints to constraints of the domain
- Choice is modelled with multiple rules
- Functions are modelled as predicates with an extra argument
- Iteration is modelled using recursion
- Optimization requires a new kind of literal