Two weeks: logic and constraint logic programming paradigms

- Use logic and theorem proving as the underlying computational model
- From a set of axioms and rules, a program executes by trying to prove a given hypothesis
- In constraint logic programming, more information is provided about the domain, which can increase the efficiency of the programs significantly

重要意义

Constraints Logic Programming

- CLP(R) --- built on top of Prolog’s foundations
- Developed by Jaffar and Lassez at Monash University in Melbourne, Australia
- Includes domain-specific constraint solvers to augment the logical deduction algorithm
- Different domains are targeted with different specialized solvers
  - CLP(FD), for finite domains
  - CLP(R), for real number

今晚

- Overview of CLP(R)
  - With examples
- Stepping back to look more carefully at CLP in general
  - Based on slides from Marriott and Stuckey

Prolog example

```prolog
solution(X, Y, Z) :- p(X), p(Y), p(Z), test(X, Y, Z).
p(11).
p(3).
p(7).
p(14).
p(15).
test(X, Y, Z) :- Y is X+1, Z is Y+1.
```

```
solution(X, Y, Z) ?
X=14; Y=15; Z=16 ?
no
```
How many search steps?

- In small groups, determine how many search steps are needed to find the one (and only) solution to the previous Prolog program.
- In the form of: “This takes X steps to find the solution and a total of Y steps to exhaust the search space.”

The problem is...

- ...that Prolog has an extremely limited knowledge of mathematics.
  - It leads to a big search space over only six possible integer values!
- It checks to see if the formulae hold, but it doesn’t think about them as mathematical formulae nor does it manipulate them as math.

Speeding up the earlier example: reordering conjuncts

```prolog
solution(X, Y, Z) :- test(X, Y, Z), p(X), p(Y), p(Z).
    p(11).
    p(3).
    p(7).
    p(16).
    p(15).
    p(14).
test(X, Y, Z) :- Y is X+1, Z is Y+1.
solution(X, Y, Z)?
```

This fails, since X is uninstantiated in test.

CLP

- CLP essentially merges logic programming with constraint solving.
- Constraint solving is much in the spirit of logic programming, allowing a two-way flow of computation.
  - But the domains are not limited to relations.
  - Borning’s Thinglab is a classic example of a system based on constraint solving.
    - “There’s a polygon in which I always want the opposite sides to be parallel to each other.”
    - “Keep point M as the midpoint of the line defined by points A and B.”

Solvers

- Underneath any constraint-based system is a constraint solver that takes equations and solves them (preferably quickly).
- The constraint satisfaction algorithms used depend on the domain over which the constraints are defined.
  - For reals, common algorithms include gauss and simplex methods.
  - A little more later.
- To become truly facile at CLP for a given domain one has to become knowledgeable about the solvers.

CLP does “more”

- The reason CLP can do “more” than logic programming is that the elements have semantic meaning.
  - In CLP(R), they are real numbers.
  - In logic programming they were just strings to which you associated some meaning.
- That is, CLP can, in general, manipulate symbolic expressions, too.
- To do this, CLPR has to understand numbers, equations, arithmetic, etc.
A CLP(R) example

\[ p(X, Y, Z) :- Z = X + Y. \]
\[ p(3, 4, Z)? \]
\[ Z=7 \]
\[ p(X, 4, 7)? \]
\[ X=3 \]
\[ p(X, Y, 7), X = -Y + 7 \] // instead of returning //multiple answers

The example in CLP(R):

replace is with =

\[ solution(X, Y, Z) :- test(X, Y, Z), p(X), p(Y), p(Z). \]
\[ p(11). \]
\[ p(3). \]
\[ p(7). \]
\[ p(16). \]
\[ p(14). \]
\[ test(X, Y, Z) :- X = X+1, Z = Y+1. \]
\[ solution(X, Y, Z)? \]
\[ X=14;Y=15;Z=16; \]
\[ NO \]

How many steps to find the solution?

Furthermore

\[ solution(X, Y, Z) :- \]
\[ test(X, Y, Z), p(X), p(Y), p(Z). \]
\[ test(X, Y, Z) :- Y = X+1, Z = Y+1. \]

\[ solution(A, B, C)? \]
\[ B = C - 1 \]
\[ A = C - 2 \]

Fibonacci: Prolog vs. CLP(R)

\[ fib(0,0). \]
\[ fib(1,1). \]
\[ fib(N, F) :- N > 1, N1 is N-1, N2 is N-2, fib(N1,F1), fib(N2,F2), F is F1 + F2. \]
\[ fib(10,L)? \]
\[ fib(55)? \]
\[ fib(X,X)? // instantiation error \]

\[ fib(0,0). \]
\[ fib(1,1). \]
\[ fib(N,F1 + F2) :- N > 1, fib(N-1,F1), fib(N-2,F2). \]
\[ fib(10,L)? \]
\[ fib(55)? \]
\[ fib(X,X)? //0,1,5 \]

Slides

- Most of tonight’s slides are taken (with implicit permission) from slides produced by Marriott and Stuckey as support material for their text book Programming with Constraints: An Introduction
- This is a great place to look for more material, if you’re interested

Constraints

- What are constraints?
- Modeling problems
- Constraint solving
- Tree constraints
- Other constraint domains
- Properties of constraint solving
Constraints

Variable: a placeholder for values
\(X, Y, Z, L_3, U_{23}, List\)

Function Symbol: mapping of values to values
\(+, -, \times, +, \sin, \cos, \|\)

Relation Symbol: relation between values
\(=, \leq, \neq\)

Satisfiability

Valuation: an assignment of values to variables
\(\theta = \{ X \mapsto 3, Y \mapsto 4, Z \mapsto 2\}\)
\(\theta(X + 2Y) = (3 + 2 \times 4) = 11\)

Solution: valuation which satisfies constraint
\(\theta(X \geq 3 \land Y = X + 1)\)
\((3 \geq 3 \land 4 = 3 + 1) = \text{true}\)

Equivalent Constraints

Two different constraints can represent the same information

\(X > 0 \iff 0 < X\)
\(X = 1 \land Y = 2 \iff Y = 2 \land X = 1\)
\(X = Y + 1 \land Y \geq 2 \iff X = Y + 1 \land X \geq 3\)

Two constraints are equivalent if they have the same set of solutions

Satisfiability

Satisfiable: constraint has a solution

Unsatisfiable: constraint does not have a solution

Constraints: syntactic issues

- Constraints are strings of symbols
- Parentheses don’t matter
\((X = 0 \land Y = 1) \land Z = 2 \iff X = 0 \land (Y = 1 \land Z = 2)\)
- Order does matter
\(X = 0 \land Y = 1 \land Z = 2 \neq Y = 1 \land Z = 2 \land X = 0\)
- Some algorithms will depend on order

Constraints

Primitive Constraint: constraint relation with arguments
\(X \geq 4\)
\(X + 2Y = 9\)

Constraint: conjunction of primitive constraints
\(X \leq 3 \land X = Y \land Y \geq 4\)
Modeling with constraints

- Constraints describe idealized behavior of objects in the real world

\[ V1 = I1 \times R1 \]
\[ V2 = I2 \times R2 \]
\[ V - V1 = 0 \]
\[ V - V2 = 0 \]
\[ I1 - I2 = 0 \]
\[ I - I1 - I2 = 0 \]

Modeling with constraints

- Start
  - Foundations: \( T_f \geq 0 \)
  - Walls: \( T_b \geq T_f + 7 \)
  - Chimney: \( T_c \geq T_f + 4 \)
  - Chimney: \( T_c \geq T_b + 3 \)
  - Roof: \( T_r \geq T_f + 2 \)
  - Doors: \( T_d \geq T_f + 2 \)
  - Windows: \( T_w \geq T_f + 3 \)

Constraint Satisfaction

- Given a constraint \( C \), two questions
  - Satisfaction: does it have a solution?
  - Solution: give me a solution, if it has one?
- The first is more basic
- A constraint solver answers the satisfaction problem

Constraint Satisfaction

- Simple approach: try all valuations.

\[
\begin{align*}
X > Y & \quad \text{false} \\
(X \mapsto 1, Y \mapsto 1) & \quad \text{false} \\
(X \mapsto 1, Y \mapsto 2) & \quad \text{false} \\
(X \mapsto 1, Y \mapsto 3) & \quad \text{false} \\
* & \quad * \\
* & \quad * \\
X > Y & \quad \text{true} \\
(X \mapsto 1, Y \mapsto 2) & \quad \text{true} \\
* & \quad * \\
\end{align*}
\]

Gauss-Jordan elimination

- Choose an equation \( c \) from \( C \)
- Rewrite \( c \) into the form \( x = e \)
- Replace \( x \) everywhere else in \( C \) by \( e \)
- Continue until
  - all equations are in the form \( x = e \)
  - or an equation is equivalent to \( d = 0 \) \( \land \) \( d \neq 0 \)
- Return true in the first case else false
Gauss-Jordan Example 1

1 + X = 2Y + Z ∧ 1 + X = 2Y + Z
Z = Y = 1 ∧
X + Y = 5 ∧

Replace X by 2Y + Z - 1

X = 2Y + Z - 1 ∧
Z = 2Y - Z + 1 = 3 ∧
2Y + Z - 1 + Y = 5 + Z

Replace Y by -1

X = -2 + Z - 1 ∧
Y = -1 ∧
Z = 4 ∧
X = -2 + Z - 1 + Y = 5 + Z

Return false

Gauss-Jordan Example 2

1 + X = 2Y + Z ∧ 1 + X = 2Y + Z
Z = X = 3

Replace X by 2Y + Z - 1

X = 2Y + Z - 1 ∧
Z = 2Y + Z - 1 + 3 ∧
2Y + Z - 1 + Y = 5 + Z

Replace Y by -1

X = Z = 3 ∧
Y = -1

Solved form: constraints in this form are satisfiable

Solved Form

• Non-parametric variable: appears on the left of one equation.
• Parametric variable: appears on the right of any number of equations.
• Solution: choose parameter values and determine non-parameters

X = Z = 3 ∧
Y = -1
Z = 4
X = 4 - 3 = 1

Tree Constraints

• Tree constraints represent structured data
• Tree constructor: character string
  – cons, node, null, widget, f
• Constant: constructor or number
• Tree:
  – A constant has height 1
  – A constructor with a list of > 0 trees is a tree
  – Drawn with constructor above children

Tree Constraints

• Height of a tree:
  – a constant has height 1
  – a tree with children t1, ..., tn has height one more than the maximum of trees t1,...,tn

Tree Examples

```plaintext
order(part(77665, widget(red, moose)), quantity(17), date(3, feb, 1994))
```

```plaintext
cons(red, cons(blue, cons(red, cons(...))))
```
Terms

- A term is a tree with variables replacing subtrees
- Term:
  - A constant is a term
  - A variable is a term
  - A constructor with a list of > 0 terms is a term
  - Drawn with constructor above children
- Term equation: s = t (s,t terms)

Term Examples

- \(\text{order}(\text{part}(77665, \text{widget}(C, \text{moose})), \text{Q}, \text{date}(3, \text{feb}, Y))\)
- \(\text{cons}(\text{red}, \text{cons}(\text{B}, \text{cons}(\text{red}, L)))\)

Tree Constraint Solving

- Assign trees to variables so that the terms are identical
  - \(\text{cons}(R, \text{cons}(B, \text{nil})) = \text{cons}(\text{red}, L)\)
    \(\{ R \mapsto \text{red}, L \mapsto \text{cons(blue, nil)}, B \mapsto \text{blue} \}\)
- Similar to Gauss-Jordan
- Starts with a set of term equations \(C\) and an empty set of term equations \(S\)
- Continues until \(C\) is empty or it returns false

Tree Constraint Solving

- unify(C)
  - Remove equation \(c\) from \(C\)
  - case \(x = x\): do nothing
  - case \(f(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)\): return false
  - case \(f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)\):
    - add \(s_1 = t_1, \ldots, s_n = t_n\) to \(C\)
    - case \(t = x\) (\(x\) variable): add \(x = t\) to \(C\)
    - case \(x = t\) (\(x\) variable): add \(x = t\) to \(S\)
  - substitute \(t\) for \(x\) everywhere else in \(C\) and \(S\)

Tree Solving Example

- \(C\)
  - \(\text{cons}(Y, \text{nil}) = \text{cons}(X, Z) \land Y = \text{cons}(a, T)\)
  - \(Y = X \land \text{nil} = Z \land Y = \text{cons}(a, T)\)
  - \(\text{nil} = Z \land X = \text{cons}(a, T)\)
  - \(Z = \text{cons}(a, T)\)
  - \(X = \text{cons}(a, T)\)

- \(S\)
  - \(Y = X \land Z = \text{nil} \implies Y = \text{cons}(a, T)\)

Like Gauss-Jordan, variables are parameters or non-parameters. A solution results from setting parameters (i.e., \(T\)) to any value.

One extra case

- Is there a solution to \(X = f(X)\) ?
- NO!
  - if the height of \(X\) in the solution is \(n\)
  - then \(f(X)\) has height \(n+1\)
- Occurs check:
  - before substituting \(t\) for \(x\)
  - check that \(x\) does not occur in \(t\)
Other Constraint Domains

- There are many
  - Boolean constraints
  - Sequence constraints
  - Blocks world
- Many more, usually related to some well understood mathematical structure

Boolean Constraints

Used to model circuits, register allocation problems, etc.

\[
\begin{align*}
X &\rightarrow (O \leftrightarrow (X \lor Y)) \\
A &\rightarrow (A \leftrightarrow (X \land Y)) \\
N &\rightarrow (N \leftrightarrow \neg A) \\
Z &\rightarrow (N \leftrightarrow (N \land O))
\end{align*}
\]

An exclusive or gate

The Boolean solver can return unknown
It is incomplete (doesn’t answer all questions)
It is polynomial time, where a complete solver is exponential (unless P \(=\) NP)
Still such solvers can be useful!
Blocks World Constraints

A solution to a Blocks World constraint is a picture with an annotation of which variable is which block:

\[
yellow(Y) \land red(X) \land on(X,Y) \land floor(Z) \land red(Z)
\]

Solver Definition

- A constraint solver is a function \( solv \) that takes a constraint \( C \) and returns \( true, false \) or \( unknown \) depending on whether the constraint is satisfiable:
  - if \( solv(C) = true \) then \( C \) is satisfiable
  - if \( solv(C) = false \) then \( C \) is unsatisfiable

Properties of Solvers

- We desire solvers to have certain properties:
  - well-behaved:
    - set based: answer depends only on set of primitive constraints
    - monotonic: if solver fails for \( C_1 \) it also fails for \( C_1 \land C_2 \)
    - variable name independent: the solver gives the same answer regardless of names of vars

Constraints Summary

- Constraints are pieces of syntax used to model real world behavior
- A constraint solver determines if a constraint has a solution
- Real arithmetic and tree constraints
- Properties of solver we expect (well-behavedness)

Simplification, Optimization and Implication

- Constraint Simplification
- Projection
- Constraint Simplifiers
- Optimization
- Implication and Equivalence
Constraint Simplification

- Two equivalent constraints represent the same information
- But one may be simpler than the other

\[ X \geq 1 \land X \geq 3 \land 2 = Y + X \]
Removing redundant constraints, rewriting a primitive constraint, changing order, substituting using an equation all preserve equivalence

\[ X = 2 - Y \land 3 \leq X \]

\[ X = 2 - Y \land Y \leq -1 \]

Redundant Constraints

- One constraint \( C_1 \) implies another \( C_2 \) if the solutions of \( C_1 \) are a subset of those of \( C_2 \)
- \( C_2 \) is said to be redundant with respect to \( C_1 \)

\[ X \geq 3 \rightarrow X \geq 1 \]

\[ Y \leq X \land 2 \land Y \geq 4 \rightarrow X \geq 1 \]

\[ \text{cost}(X, X) = \text{cost}(Z, \text{nil}) \rightarrow Z = \text{nil} \]

Definitely produces a simpler constraint

Solved Form Solvers

- Since a solved form solver creates equivalent constraints, it can be a simplifier

For example, using the term constraint solver

\[ \text{cost}(X, X) = \text{cost}(Z, \text{nil}) \land Y = \text{succ}(X) \land \text{succ}(Z) = Y \land Z = \text{nil} \]

\[ \leftrightarrow X = \text{nil} \land Z = \text{nil} \land Y = \text{succ}(\text{nil}) \]

Or using the Gauss-Jordan solver

\[ X = 2 + Y \land 2 \land X - T = Z \land X + Y = 4 \land Z + T = 5 \]

\[ \leftrightarrow X = 3 \land Y = 1 \land Z = 5 - T \]

Constraint Simplifiers

- \( C_1 \) and \( C_2 \) are equivalent wrt variables \( V \) if
  - taking any solution of one and restricting it to the variables \( V \), this restricted solution can be extended to be a solution of the other
- Example \( X = \text{succ}(Y) \) and \( X = \text{succ}(Z) \) wrt \( \{X\} \)

\[ X = \text{succ}(Y) \]
\[ (X \rightarrow \text{succ}(a), Y \rightarrow a) \]
\[ \{X\} \]
\[ X = \text{succ}(Z) \]
\[ (X \rightarrow \text{succ}(a), Z \rightarrow a) \]
Optimization

- Often given some problem that is modeled by constraints we don’t want just any solution, but a “best” solution
- This is an optimization problem
- We need an objective function so that we can rank solutions
  – That is, a mapping from solutions to a real value

Optimization Problem

- An optimization problem \((C, f)\) consists of a constraint \(C\) and objective function \(f\)
- A valuation \(v1\) is preferred to valuation \(v2\) if \(f(v1) < f(v2)\)
- An optimal solution is a solution of \(C\) such that no other solution of \(C\) is preferred to it

Optimization Example

An optimization problem \((C = X + Y \geq 4, \ f = X^2 + Y^2)\)
Find the closest point to the origin satisfying the \(C\).
Some solutions and \(f\) value

| \((X \mapsto 0, Y \mapsto 4)\) | 16  |
| \((X \mapsto 3, Y \mapsto 3)\) | 18  |
| \((X \mapsto 2, Y \mapsto 2)\)  | 8   |

Optimal solution \((X \mapsto 2, Y \mapsto 2)\)

Optimization

- Some optimization problems have no solution
  – Constraint has no solution
    \((X \geq 2 \land X \leq 0, \ X^2)\)
  – Problem has no optimum — for any solution there is more preferable one
    \((X \leq 0, \ X)\)

Simplex Algorithm

- The most widely used optimization algorithm
- Optimizes a linear function wrt to linear constraints
- Related to Gauss-Jordan elimination

Simplex Algorithm

- A optimization problem \((C, f)\) is in simplex form:
  – \(C\) is the conjunction of \(CE\) and \(CI\)
  – \(CE\) is a conjunction of linear equations
  – \(CI\) constrains all variables in \(C\) to be non-negative
  – \(f\) is a linear expression over variables in \(C\)
Simplex Example

An optimization problem in simplex form

\[
\begin{align*}
\text{minimize} & \quad 3X + 2Y - Z + 1 \\
\text{subject to} & \quad X + Y = 3 \\
& \quad -X - 3Y + 2Z + T = 1 \\
& \quad X \geq 0, Y \geq 0, Z \geq 0, T \geq 0
\end{align*}
\]

- An arbitrary problem can be put in simplex form by
  - replacing unconstrained var \( X \) by new vars
  - replacing ineq \( \leq \) by new vars

Simplex Solved Form

- A simplex optimization problem is in basic feasible solved (bfs) form if:
  - The equations are in solved form
  - Each constant on the right hand side is non-negative
  - Only parameters occur in the objective

- A basic feasible solution is obtained by
  - setting each parameter to 0 and each non-parameter to the constant in its equation

Simplex Algorithm

starting from a problem in bfs form

repeat

- Choose a variable \( y \) with negative coefficient in the obj. func.
- Find the equation \( x = b + cy + ... \) where \( c < 0 \) and \( -b/c \) is minimal
- Rewrite this equation with \( y \) the subject \( y = -b/c + 1/c \times x + ... \)
- Substitute \( -b/c + 1/c \times x + ... \) for \( y \) in all other eqns and obj. func.
- until no such variable \( y \) exists or no such equation exists

if no such \( y \) exists optimum is found
else there is no optimum solution

Simplex Example

An equivalent problem to that before in bfs form

\[
\begin{align*}
\text{minimize} & \quad 10 - Y - Z \\
\text{subject to} & \quad X = 3 - Y \\
& \quad T = 4 + 2Y - 2Z \\
& \quad X \geq 0, Y \geq 0, Z \geq 0, T \geq 0
\end{align*}
\]

We can read off a solution and its objective value:

\( (X \mapsto 3, T \mapsto 4, Y \mapsto 0, Z \mapsto 0) \)

\( f = 10 \)

Simplex Example

Choose variable \( Y \), the first eqn is only one with neg. coeff

Choose variable \( Z \), the 2nd eqn is only one with neg. coeff

Choose variable \( Z \) the 2nd eqn is only one with neg. coeff

\[
\begin{align*}
\text{minimize} & \quad 2 + 2X + 0.5T \\
\text{subject to} & \quad Y = 3 - X \\
& \quad Z = 5 - X - 0.5T
\end{align*}
\]

No variable can be chosen, optimal value 2 is found

Another example

\[
\begin{align*}
\text{minimize} & \quad X - Y \\
\text{subject to} & \quad Y \geq 0 \\
& \quad X \geq 1 \\
& \quad X \leq 3
\end{align*}
\]

An equivalent simplex form is:

\[
\begin{align*}
X - S0 &= 1 \\
X + S1 &= 3 \\
- X + 2F + S1 &= 3
\end{align*}
\]

An optimization problem showing contours of the objective function
Implication and Equivalence

- Other important operations involving constraints are:
  - implication: test if \( C_1 \) implies \( C_2 \)
    - \( \text{impl}(C_1, C_2) \) answers true, false or unknown
  - equivalence: test if \( C_1 \) and \( C_2 \) are equivalent
    - \( \text{equiv}(C_1, C_2) \) answers true, false or unknown

Implication Example

For the house constraints \( CH \), will stage B have to be reached after stage C?

\[ CH \rightarrow T_B \geq T_C \]

For this question the answer if false, but if we require the house to be finished in 15 days the answer is true

\[ CH \land T_B = 15 \rightarrow T_B \geq T_C \]

Simplication, Optimization and Implication Summary

- Equivalent constraints can be written in many forms, hence we desire simplification
- Particularly if we are only interested in the interaction of some of the variables
- Many problems desire a optimal solution, there are algorithms (simplex) to find them

Some more CLP(R) examples

- To try to tie this all together

Rules

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

\[ \text{parallel_resistors}(V, I, R_1, R_2) \]

And the rule defining it

\[ \text{parallel_resistors}(V, I, R_1, R_2) :- \]

\[ V = I_1 \cdot R_1, \quad V = I_2 \cdot R_2, \quad I_1 + I_2 = I. \]

Using Rules

\[ \text{parallel_resistors}(V, I, R_1, R_2) :- \]

\[ V = I_1 \cdot R_1, \quad V = I_2 \cdot R_2, \quad I_1 + I_2 = I. \]

Behavior with resistors of 10 and 5 Ohms

\[ \text{parallel_resistors}(V, I, R_1, R_2) \land R_1 = 10 \land R_2 = 5 \]

Behavior with 10V battery where resistors are the same

\[ \text{parallel_resistors}(V, I, R_1, R_2) \land V = 10 \land R_1 = R_2 \]

It represents the constraint (macro replacement)
Modeling

- 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 traveler 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: $R$

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

\[ \text{river}(W, S, R, P) : - T = W/R, P = S*T. \]

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?

\[ 1 \leq R, R \leq 1.3, P \leq 20, \]

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

Flexibility of constraint based modeling!

More Complicated Model

- 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 $E$
- 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
- 100 sell 2 puts at 300
Modeling Functions

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

\[
\begin{align*}
\text{call\_payoff}(S,C,E,P) &= \begin{cases} 
-C & \text{if } 0 \leq S \leq E/100 \\
100S - E - C & \text{if } S \geq E/100 
\end{cases} 
\end{align*}
\]

\[
\begin{align*}
\text{buy\_call\_payoff}(S,C,E,P) &:\quad 0 \leq S, S \leq E/100, P = -C. \\
\text{buy\_call\_payoff}(S,C,E,P) &:\quad S \geq E/100, P = 100S - E - C.
\end{align*}
\]

Modeling Options

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

\[
\begin{align*}
\text{call\_option}(B,S,C,E,P) &:\quad 0 \leq S, S \leq E/100, P = -C \times B. \\
\text{call\_option}(B,S,C,E,P) &:\quad S \geq E/100, P = (100S - E - C)B.
\end{align*}
\]

The goal (the original call option question):

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

has answer \( P = 200 \)

Using the Model

\[
\begin{align*}
\text{butterfly}(S, P1 + 2*P2 + P3) &:\quad \\
&\text{Buy} = 1, \text{Sell} = -1, \\
&\text{call\_option}(\text{Buy}, S, 100, 500, P1), \\
&\text{call\_option}(\text{Sell}, S, 200, 300, P2), \\
&\text{call\_option}(\text{Buy}, S, 400, 100, P3). \\
P &\geq 0, \text{butterfly}(S,P).
\end{align*}
\]

has two answers:

\[
\begin{align*}
P &= 100S - 200 \land 2 \leq S \land S \leq 3 \\
P &= -100S + 400 \land 3 \leq S \land S \leq 4
\end{align*}
\]

Wrap up

- LP and CLP are not general purpose computing paradigms
  - Even though they are Turing equivalent, there is no way you’d do most general purpose programs in them
- However, there are a number of important problems for which this is a good match

Domains

- But the expense of building a solver, simplifier, etc. for a given domain is not small
  - So the narrow domain must provide enough benefit to justify this effort

Next week

- Visual programming and program visualization
- Final week: domain specific languages