CSE583: Programming Languages

David Notkin
22 February 2000
notkin@cs.washington.edu
http://www.cs.washington.edu/education/courses/583

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

Constraint 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

Importance of Constraint Logic Programming

"Were you to ask me which programming paradigm is likely to gain most in commercial significance over the next 5 years I'd have to pick Constraint Logic Programming..."
— Dick Pountain

Tonight

- 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

```
solution(X,Y,Z) :- p(X),p(Y),p(Z),test(X,Y,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)?
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

```
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
    - "here'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(15).
p(14).
test(X,Y,Z) :- Y = 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(N,55)?
  // 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(N,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 place holder for values

$X, Y, Z, L_3, U_{21}, List$

Function Symbol: mapping of values to values

$+, -, \times, \div, \sin, \cos, ||$

Relation Symbol: relation between values

$=, \leq, \neq$

Constraints

Primitive Constraint: constraint relation with arguments

$$X \geq 4$$

$$X + 2Y = 9$$

Constraint: conjunction of primitive constraints

$$X \leq 3 \land X = Y \land Y \geq 4$$

Satisfiability

Very similar to unification

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

$$\theta(X \geq 3 \land Y = X + 1)$$

$$= (3 \geq 3 \land 4 = 3 + 1) = true$$

Satisfiability

Satisfiable: constraint has a solution

Unsatisfiable: constraint does not have a solution

$X \leq 3 \land Y = X + 1$ *satisfiable*

$X \leq 3 \land Y = X + 1 \land Y \geq 6$ *unsatisfiable*

Constraints: syntactic issues

- **Constraints are strings of symbols**
- **Parentheses don't matter**

  $(X = 0 \land Y = 1) \land Z = 2 \;\equiv\; X = 0 \land (Y = 1 \land Z = 2)$
- **Order does matter**

  $X = 0 \land Y = 1 \land Z = 2 \;\not\equiv\; Y = 1 \land Z = 2 \land X = 0$
- **Some algorithms will depend on order**

Equivalent Constraints

Two different constraints can represent the same information

$$X > 0 \leftrightarrow 0 < X$$

$$X = 1 \land Y = 2 \leftrightarrow Y = 2 \land X = 1$$

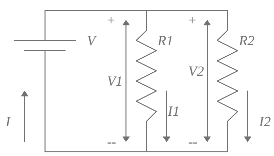
$$X = Y + 1 \land Y \geq 2 \leftrightarrow X = Y + 1 \land X \geq 3$$

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

Modeling with constraints

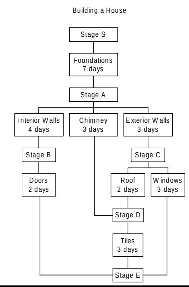
- **Constraints describe idealized behavior of objects in the real world**

$$V1 = I1 \times R1$$
$$V2 = I2 \times R2$$
$$V - V1 = 0$$
$$V - V2 = 0$$
$$V1 - V2 = 0$$
$$I - I1 - I2 = 0$$
$$-I + I1 + I2 = 0$$



Modelling with constraints

start	$T_S \geq 0$
foundations	$T_A \geq T_S + 7$
interior walls	$T_B \geq T_A + 4$
exterior walls	$T_C \geq T_A + 3$
chimney	$T_D \geq T_A + 3$
roof	$T_D \geq T_C + 2$
doors	$T_E \geq T_B + 2$
tiles	$T_E \geq T_D + 3$
windows	$T_E \geq T_C + 3$



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

- **How do we answer the question?**
- **Simple approach: try all valuations.**

$X > Y$

$\{X \mapsto 1, Y \mapsto 1\}$ *false*
$\{X \mapsto 1, Y \mapsto 2\}$ *false*
$\{X \mapsto 1, Y \mapsto 3\}$ *false*
•
•
•

$X > Y$

$\{X \mapsto 1, Y \mapsto 1\}$ *false*
$\{X \mapsto 2, Y \mapsto 1\}$ *true*
$\{X \mapsto 2, Y \mapsto 2\}$ *false*
$\{X \mapsto 3, Y \mapsto 1\}$ *true*
$\{X \mapsto 3, Y \mapsto 2\}$ *true*
•
•

Constraint Satisfaction

- **The enumeration method won't work for reals**
- **A smarter version will be used for finite domain constraints**
- **How do we solve constraints on the reals?**
- **⇒ Gauss-Jordan elimination**

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 ^* (*d != 0*)**
- **Return *true* in the first case else *false***

Gauss-Jordan Example 1

$$1+X=2Y+Z \wedge \quad Z-X=3 \wedge \quad X+Y=5+Z \qquad 1+X=2Y+Z$$

Replace X by $2Y+Z-1$

$$X=2Y+Z-1 \wedge \quad Z-2Y-Z+1=3 \wedge \quad 2Y+Z-1+Y=5+Z \qquad -2Y=2$$

Replace Y by -1

$$X=-2+Z-1 \wedge \quad Y=-1 \wedge \quad -2+Z-1-1=5+Z \qquad -4=5$$

Return *false*

Gauss-Jordan Example 2

$$1+X=2Y+Z \wedge \quad Z-X=3 \qquad 1+X=2Y+Z$$

Replace X by $2Y+Z-1$

$$X=2Y+Z-1 \wedge \quad Z-2Y-Z+1=3 \qquad -2Y=2$$

Replace Y by -1

$$X=Z-3 \wedge \quad Y=-1$$

Solved form: constraints in this form are satisfiable

Solved Form

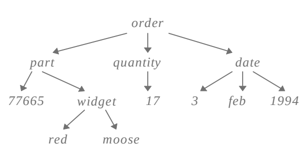
- Non-parametric variable: **appears on the left of one equation.**
- Parametric variable: **appears on the right of any number of equations.**
- Solution: **choose parameter values and determine non-parameters**

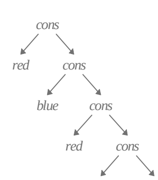
$$X=Z-3 \wedge \; Y=-1 \longrightarrow Z=4 \longrightarrow X=4-3=1 \; Y=-1$$

Tree Constraints

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

Tree Examples





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

cons(red,cons(blue,cons(red,cons(…))))

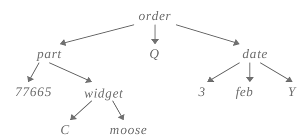
Tree Constraints

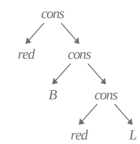
- Height of a tree:
  - **a constant has height 1**
  - **a tree with children *t1, …, tn* has height one more than the maximum of trees *t1,…,tn***

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





order(part(77665, widget(C, moose)), Q, date(3, feb, Y))

cons(red,cons(B,cons (red,L)))

Tree Constraint Solving

- Assign trees to variables so that the terms are identical
  - ***cons(R, cons(B, nil)) = cons(red, L)***

    $\{R \mapsto red, L \mapsto cons(blue, nil), B \mapsto blue\}$
- 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(s1,..,sn)=g(t1,..,tn):*** return ***false***
  - case ***f(s1,..,sn)=f(t1,..,tn):***
    - add ***s1=t1, .., sn=tn*** 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	*S*
$\underline{cons(Y, nil) = cons(X, Z)} \wedge Y = cons(a, T)$	*true*
$\underline{Y = X} \wedge nil = Z \wedge Y = cons(a, T)$	*true*
$\underline{nil = Z} \wedge X = cons(a, T)$	$Y = X$
$\underline{Z = nil} \wedge X = cons(a, T)$	$Y = X$
$\underline{X = cons(a, T)}$	$Y = X \wedge Z = nil$
true	$Y = cons(a, T) \wedge Z = nil \wedge X = cons(a, T)$

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

One extra case

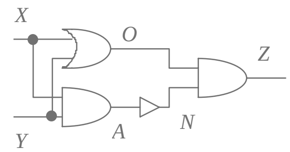
- Is there a solution to X = f(X) ?
- NO!
  - if the height of X in the solution is n
  - then f(X) has height n+1
- Occurs check:
  - before substituting t for x
  - check that x does not occur in t

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.



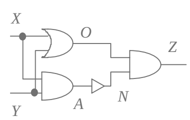
$$O \leftrightarrow (X \vee Y) \wedge$$
$$A \leftrightarrow (X \& Y) \wedge$$
$$N \leftrightarrow \neg A \wedge$$
$$Z \leftrightarrow (O \& N)$$

An exclusive or gate

Boolean constraint describing the xor circuit

Boolean Constraints

$$\neg FO \leftrightarrow (O \leftrightarrow (X \vee Y)) \wedge$$
$$\neg FA \leftrightarrow (A \leftrightarrow (X \& Y)) \wedge$$
$$\neg FN \leftrightarrow (N \leftrightarrow \neg A) \wedge$$
$$\neg FG \leftrightarrow (Z \leftrightarrow (N \& O))$$



Constraint modeling the circuit with faulty variables

$$\neg(FO \& FA) \wedge \neg(FO \& FN) \wedge \neg(FO \& FG) \wedge$$
$$\neg(FA \& FN) \wedge \neg(FA \& FG) \wedge \neg(FN \& FG)$$

Boolean Solver

let m be the number of primitive constraints in C

$$n := \left\lceil \frac{\ln(\varepsilon)}{\ln(1-(1-\frac{1}{m})^m)} \right\rceil$$

epsilon is between 0 and 1 and determines the degree of incompleteness

for i := 1 to n **do**

  generate a random valuation over the variables in C

  if the valuation satisfies C **then return** *true* **endif**

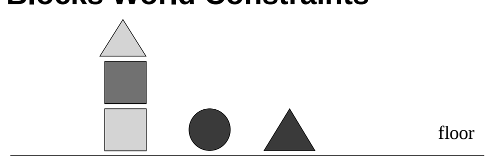
endfor

return *unknown*

Boolean Constraints

- Something new?
- 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



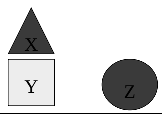
Constraints don't have to be mathematical

Objects in the blocks world can be on the floor or on another object. Physics restricts which positions are stable. Primitive constraints are e.g. *red(X), on(X,Y), not_sphere(Y).*

Blocks World Constraints

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

$yellow(Y) \wedge$
$red(X) \wedge$
$on(X,Y) \wedge$
$floor(Z) \wedge$
$red(Z)$



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: **is solver fails for *C1* it also fails for *C1 /\ C2***
  - variable name independent: **the solver gives the same answer regardless of names of vars**

Properties of Solvers

- **The most restrictive property we can ask:**
  - complete: **A solver is complete if it always answers *true* or *false***
    - **(never *unknown*)**

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**

$$X \geq 1 \wedge X \geq 3 \wedge 2 = Y + X$$
$$\leftrightarrow X \geq 3 \wedge 2 = Y + X$$
$$\leftrightarrow 3 \leq X \wedge X = 2 - Y$$
$$\leftrightarrow X = 2 - Y \wedge 3 \leq X$$
$$\leftrightarrow X = 2 - Y \wedge 3 \leq 2 - Y$$
$$\leftrightarrow X = 2 - Y \wedge Y \leq -1$$

Removing redundant constraints, rewriting a primitive constraint, changing order, substituting using an equation all preserve equivalence

Redundant Constraints

- **One constraint *C1*** implies **another *C2* if the solutions of *C1* are a subset of those of *C2***
- ***C2* is said to be** redundant **with respect to *C1***

$$X \geq 3 \rightarrow X \geq 1$$
$$Y \leq X + 2 \wedge Y \geq 4 \rightarrow X \geq 1$$
$$cons(X, X) = cons(Z, nil) \rightarrow Z = nil$$

Redundant Constraints

- **We can remove a primitive constraint that is redundant with respect to the rest of the constraint**

$$\underline{X \geq 1} \wedge X \geq 3 \leftrightarrow X \geq 3$$
$$Y \leq X + 2 \wedge \underline{X \geq 1} \wedge Y \geq 4 \leftrightarrow Y \leq X + 2 \wedge Y \geq 4$$
$$cons(X, X) = cons(Z, nil) \wedge \underline{Z = nil} \leftrightarrow cons(X, X) = cons(Z, nil)$$

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

$$cons(X, X) = cons(Z, nil) \wedge Y = succ(X) \wedge succ(Z) = Y \wedge Z = nil$$
$$\leftrightarrow X = nil \wedge Z = nil \wedge Y = succ(nil)$$

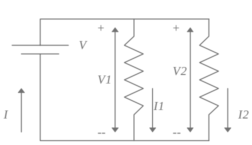
Or using the Gauss-Jordan solver

$$X = 2 + Y \wedge 2Y + X - T = Z \wedge X + Y = 4 \wedge Z + T = 5$$
$$\leftrightarrow X = 3 \wedge Y = 1 \wedge Z = 5 - T$$

Projection

It becomes even more important to simplify when we are only interested in some variables in the constraint

$$V1 = I1 \times R1$$
$$V2 = I2 \times R2$$
$$V - V1 = 0$$
$$V - V2 = 0$$
$$V1 - V2 = 0$$
$$I - I1 - I2 = 0$$
$$-I + I1 + I2 = 0$$
$$R1 = 5$$



Simplified w.r.t. to *V* and *I*

$$V = \frac{10}{} I$$

Constraint Simplifiers

- **constraints *C1* and *C2* 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=succ(Y)* and *X=succ(Z)* wrt *{X}***

$X = succ(Y)$	$\{X\}$	$X = succ(Z)$
$\{X \mapsto succ(a), Y \mapsto a\}$	$\{X \mapsto succ(a)\}$	$\{X \mapsto succ(a), Z \mapsto a\}$

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

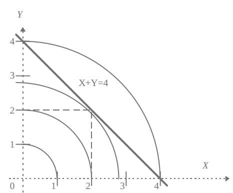
Optimization Example

An optimization problem

$(C \equiv X + Y \geq 4, \quad f \equiv X^2 + Y^2)$

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

    $(X \geq 2 \wedge X \leq 0, \quad X^2)$
  - Problem has no optimum — for any solution there is more preferable one

    $(X \leq 0, \quad X)$

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

$$\text{minimize } 3X+2Y-Z+1 \text{ subject to}$$
$$X + Y = 3 \wedge$$
$$-X - 3Y + 2Z + T = 1 \wedge$$
$$X \geq 0 \wedge Y \geq 0 \wedge Z \geq 0 \wedge T \geq 0$$

- An arbitrary problem can be put in simplex form by
  - replacing unconstrained var X by new vars $X^{+} - X^{-}$
  - replacing ineq $e \leq r$ by new var s and $e + s = r$

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 Example

An equivalent problem to that before in bfs form

$$\text{minimize } 10 - Y - Z \text{ subject to}$$
$$X = 3 - Y \wedge$$
$$T = 4 + 2Y - 2Z \wedge$$
$$X \geq 0 \wedge Y \geq 0 \wedge Z \geq 0 \wedge T \geq 0$$

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 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 x + ...*

Substitute *-b/c + 1/c 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

$$\text{minimize } 10 - Y - Z \text{ subject to}$$
$$X = 3 - Y \wedge$$
$$T = 4 + 2Y - 2Z \wedge$$
$$X \geq 0 \wedge Y \geq 0 \wedge Z \geq 0 \wedge T \geq 0$$

Choose variable *Y*, the first eqn is only one with neg. coeff

Choose variable *Z*, the 2nd eqn is only one with neg. coeff $Z = 5 - X - 0.5T$

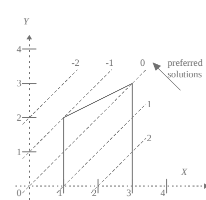
$$\text{minimize } 2 + 2X + 0.5T \text{ subject to}$$
$$Y = 3 - X \wedge$$
$$Z = 5 - X - 0.5T \wedge$$

$$\text{minimize } 7 + X - Z \text{ subject to}$$
$$Y = 3 - X \wedge$$
$$T = 10 - 2X - 2Z \wedge$$

No variable can be chosen, optimal value 2 is found

Another example

$$\text{minimize } X - Y \text{ subject to}$$
$$Y \geq 0 \wedge$$
$$X \geq 1 \wedge$$
$$X \leq 3 \wedge$$



An equivalent simplex form is:

$$X - S_2 = 1 \wedge$$
$$X + S_3 = 3 \wedge$$
$$-X + 2Y + S_1 = 3 \wedge$$

An optimization problem showing contours of the objective function

Implication and Equivalence

- **Other important operations involving constraints are:**
- implication**: test if *C1* implies *C2***
  - ***impl(C1, C2)* answers *true, false* or *unknown***
- equivalence**: test if *C1* and *C2* are equivalent**
  - ***equiv(C1, C2)* answers *true, false* or *unknown***

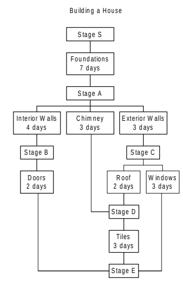
Implication Example

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*

$$CH \wedge T_E = 15 \rightarrow T_B \geq T_C$$



Simplication, Optimization and Implication Summary

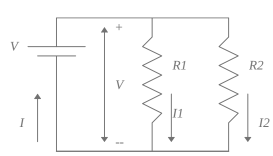
- **Equivalent constraints can be written in many forms, hence we desire simplification**
- **Particularly if we are only interested in the interaction of some of the variables**
- **Many problems desire a optimal solution, there are algorithmms (simplex) to find them**

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:




```
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.
```


Using Rules

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

Behavior with resistors of 10 and 5 Ohms

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

Behavior with 10V battery where resistors are the same

parallel_resistors

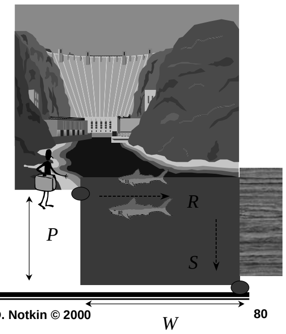
It represents the constraint (macro replacement)

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*

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

```
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.

```
river(24, 1, 1.5, P).
```

Has unique answer *P = 16*

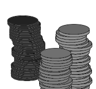
Modeling Example Cont.

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,
    river(24,1,R,P).
```

Flexibility of constraint based modeling!

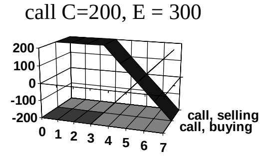
More Complicated Model



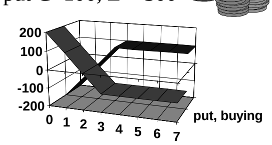
- **A** call option **gives the holder the right to buy 100 shares at a fixed price *E***
- **A** put option **gives the holder the right to sell 100 shares at a fixed price *E***
- pay off **of an option is determined by cost *C* and current share price *S***
- **e.g. call cost $200 exercise $300**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

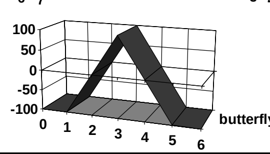
Options Trading

call C=200, E = 300



put C=100, E = 300





Butterfly strike: buy call at 500 and 100 sell 2 puts at 300

Modeling Functions

$$call\_payoff(S,C,E) = \begin{cases} -C & \text{if } 0 \le S \le E/100 \\ 100S - E - C & \text{if } S \ge E/100 \end{cases}$$

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

```
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.
```


Modeling Options

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

```
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.
```

The goal (the original call option question)

```
call_option(1, 7, 200, 300, P)
```

has answer *P = 200*

Using the Model

```
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).
P >= 0, butterfly(S,P).
```

has two answers

$$P = 100S - 200 \wedge 2 \le S \wedge S \le 3$$
$$P = -100S + 400 \wedge 3 \le S \wedge S \le 4$$

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**