Constraint Programming

Maurizio Gabbrielli
Universita' di Bologna

Including slides by , K. Marriott

Overview

- Constraints (what they are)
- Constraint solvers
- Finite constraints domains: CSP

- Constraint Logic Programming
- Modeling with Constraints
- Practical problems

Overview: part 2

- Concurrent constraint programming (ccp)
- Timed extensions of ccp (tccp)
- A proof system for (timed) ccp
- Expressive power
- Constraint Handling Rules (CHR)

Constraints

- What are constraints?
- Modelling 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 (full FOL formula in some cases)

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

Satisfiability

Valuation: an assignment of values (of a suitable domain) 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

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

$$= (3 \geq 3 \wedge 4 = 3 + 1) = \mathit{true}$$

Satisfiability

Satisfiable: constraint has a solution

Unsatisfiable: constraint does not have a solution

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

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

Constraints as Syntax

- Constraints are strings of symbols
- Brackets don't matter (don't use them)

$$(X = 0 \wedge Y = 1) \wedge Z = 2 \;\equiv\; X = 0 \wedge (Y = 1 \wedge Z = 2)$$

- Order does matter (!, so not really FOL)

$$X = 0 \wedge Y = 1 \wedge Z = 2 \;\not\equiv\; Y = 1 \wedge Z = 2 \wedge X = 0$$

- Some algorithms will depend on order

Equivalent Constraints

Two different constraints can represent the same information

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

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

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

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

Modelling with constraints

- Constraints describe idealized behaviour of objects in the real world

$$V1 = I1 \times R1$$

$$V2 = I2 \times R2$$

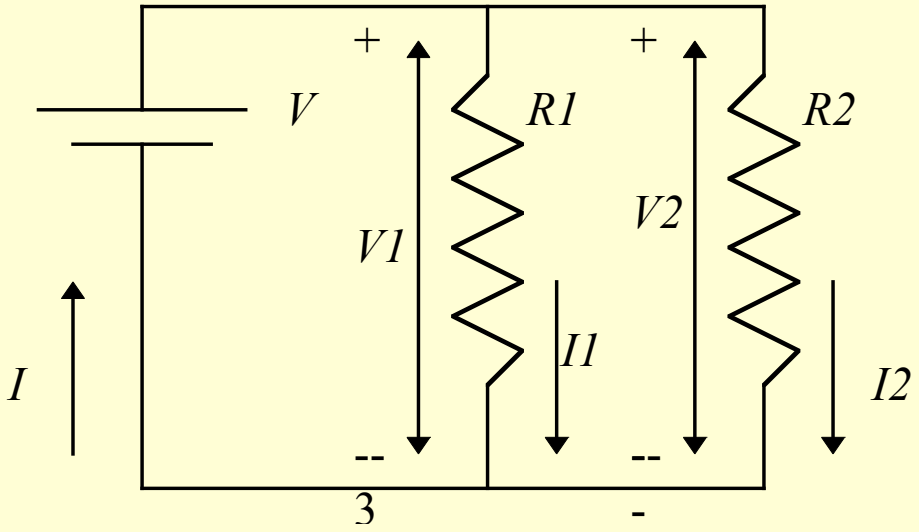
$$V - V1 = 0$$

$$V - V2 = 0$$

$$V1 - V2 = 0$$

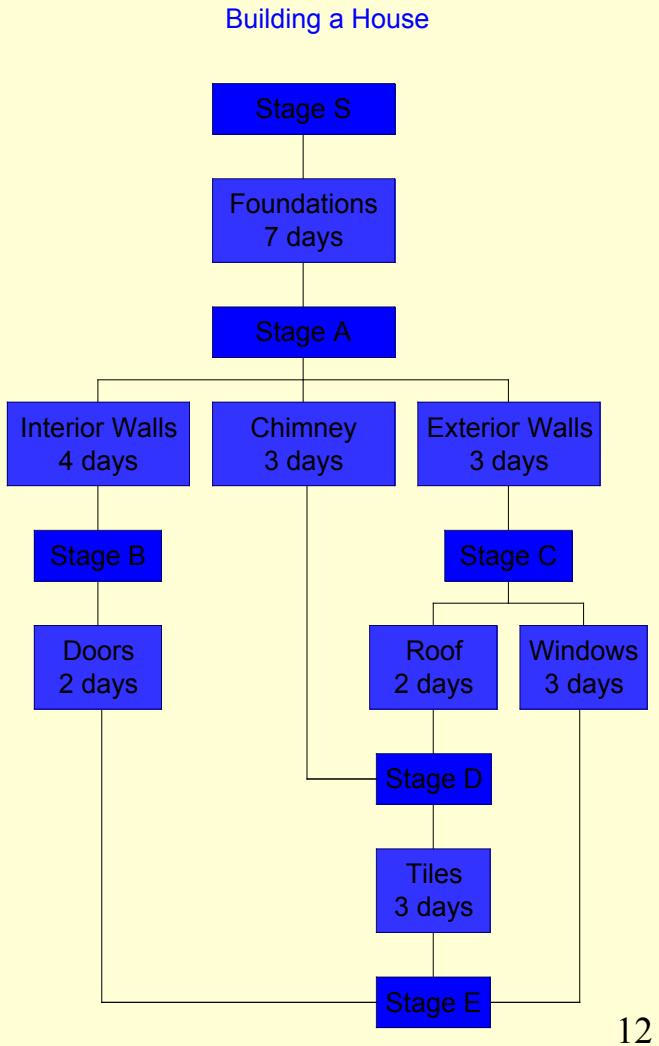
$$I - I1 - I2 = 0$$

$$-I + I1 + I2 = 0$$



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.

$$X > Y$$

$$\{X \mapsto 1, Y \mapsto 1\} \quad \mathit{false}$$
$$\{X \mapsto 1, Y \mapsto 2\} \quad \mathit{false}$$
$$\{X \mapsto 1, Y \mapsto 3\} \quad \mathit{false}$$
$$\bullet$$
$$\bullet$$
$$\bullet$$

$$X > Y$$

$$\{X \mapsto 1, Y \mapsto 1\} \quad \mathit{false}$$
$$\{X \mapsto 2, Y \mapsto 1\} \quad \mathit{true}$$
$$\{X \mapsto 2, Y \mapsto 2\} \quad \mathit{false}$$
$$\{X \mapsto 3, Y \mapsto 1\} \quad \mathit{true}$$
$$\{X \mapsto 3, Y \mapsto 2\} \quad \mathit{true}$$
$$\bullet$$
$$\bullet$$

Constraint Satisfaction

- The enumeration method wont work for Reals (why not?)
- A smarter version will be used for finite domain constraints
- How do we solve Real constraints
- Remember Gauss-Jordan elimination from high school

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 $d = 0$ ($d \mathrel{!}= 0$)
- Return *true* in the first case else *false*

Gauss-Jordan Example 1

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

Replace X by $2Y+Z-1$

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

Replace Y by -1

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

Return *false*

Gauss-Jordan Example 2

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

Replace X by $2Y+Z-1$

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

Replace Y by -1

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

Solved form: constraints in this form are satisfiable

Solved Form

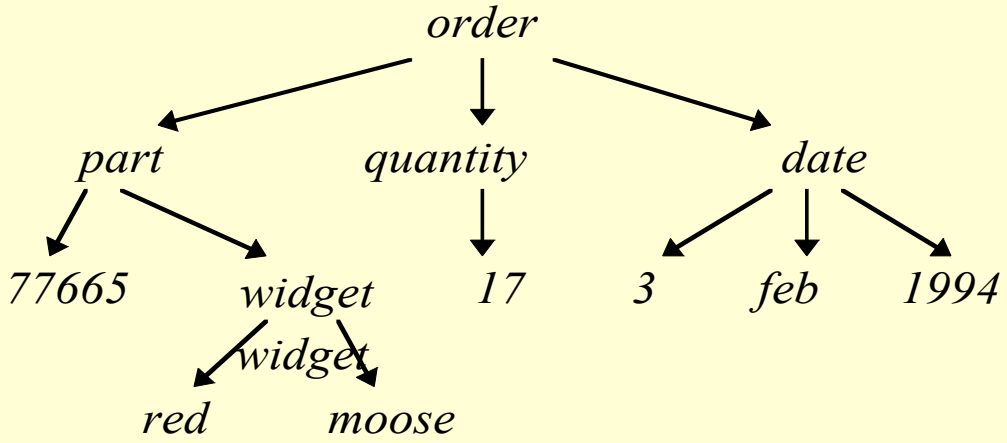
- **Non-parametric variable:** appears on the left of one equation.
- **Parametric variable:** appears on the right of any number of equations.
- **Solution:** choose parameter values and determine non-parameters

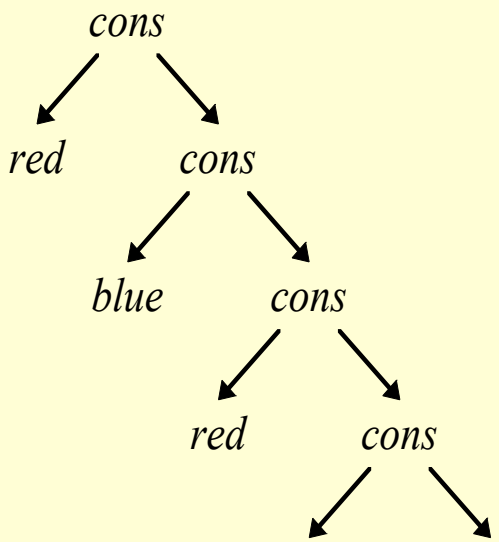
$$\begin{array}{l} X = Z - 3 \\ Y = -1 \end{array} \wedge \longrightarrow Z = 4 \longrightarrow \begin{array}{l} X = 4 - 3 = 1 \\ Y = -1 \end{array}$$

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(...))))

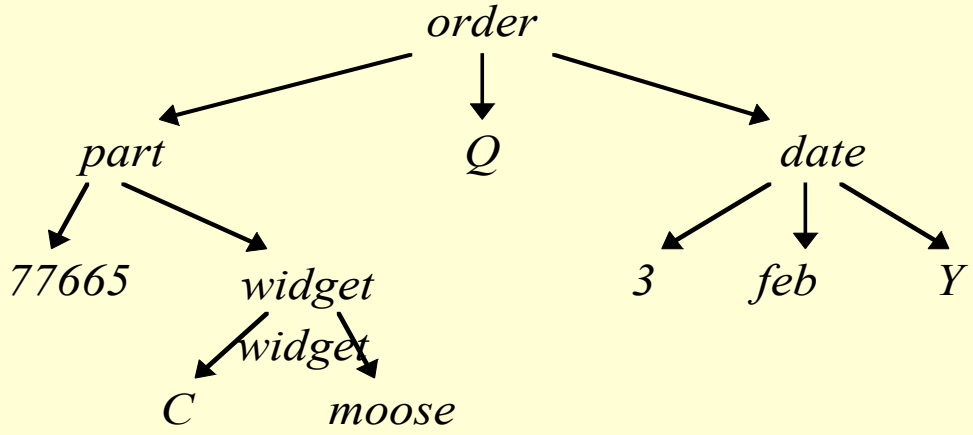
Tree Constraints

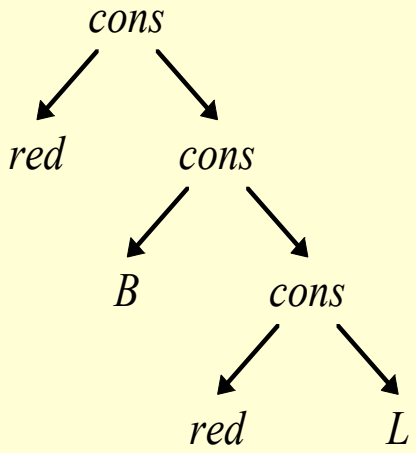
- **Height of a tree:**
  - a constant has height 1
  - a tree with children *t1, ..., tn* has height one more than the maximum of trees *t1,...,tn*
- **Finite tree:** has finite height
- Examples: height 4 and height ∞

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.

$$\{T \mapsto nil, X \mapsto cons(a,nil), Y \mapsto cons(a,nil), Z \mapsto nil\}$$

One extra case

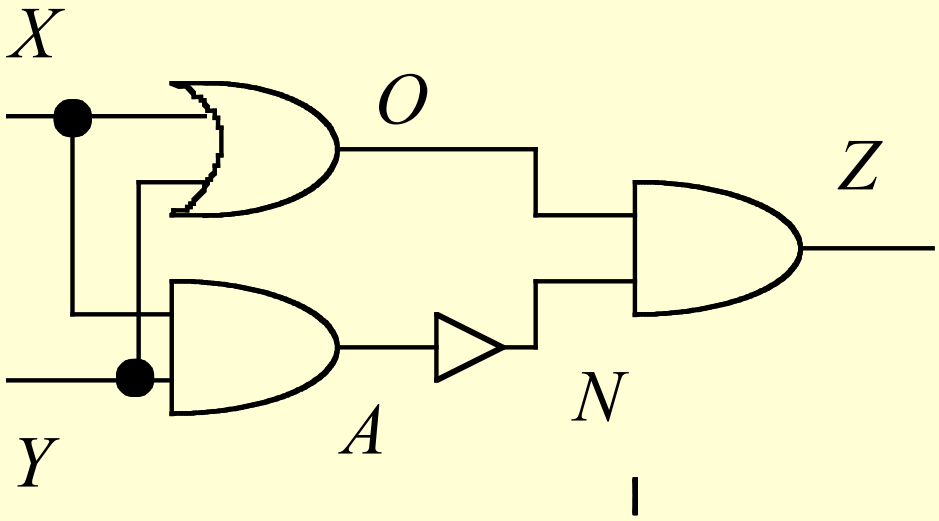
- Is there a solution to $X = f(X)$?
- NO!
  - if the height of X in the solution is n
  - then $f(X)$ has height $n+1$
- **Occurs check:**
  - before substituting t for x
  - check that x does not occur in t

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.



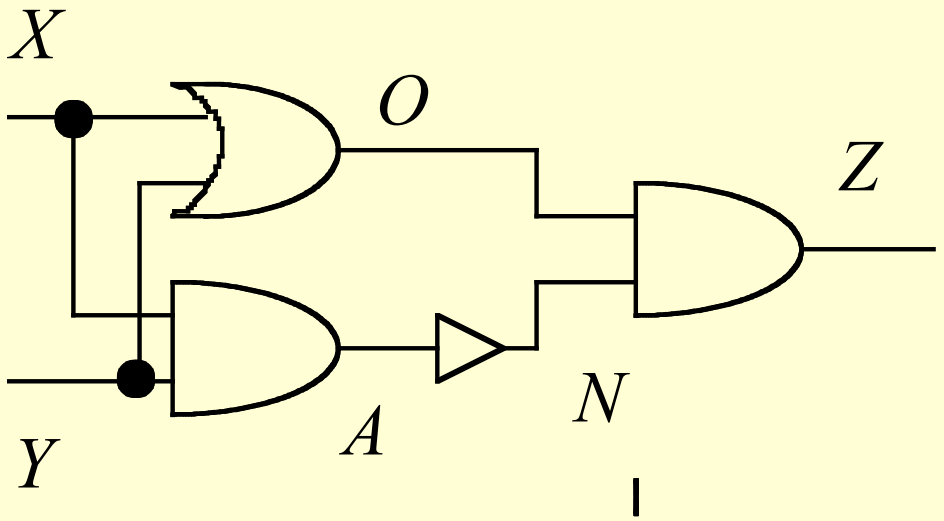
An exclusive or gate

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

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 modelling the circuit with faulty variables

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

Constraint modelling that only one gate is faulty

Observed behaviour: $\{X \mapsto 0, Y \mapsto 0, Z \mapsto 1\}$

Solution: $\{FO \mapsto 1, FA \mapsto 0, FN \mapsto 0, FG \mapsto 0,$
$X \mapsto 0, Y \mapsto 0, O \mapsto 1, A \mapsto 0, N \mapsto 1, Z \mapsto 1\}$

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**

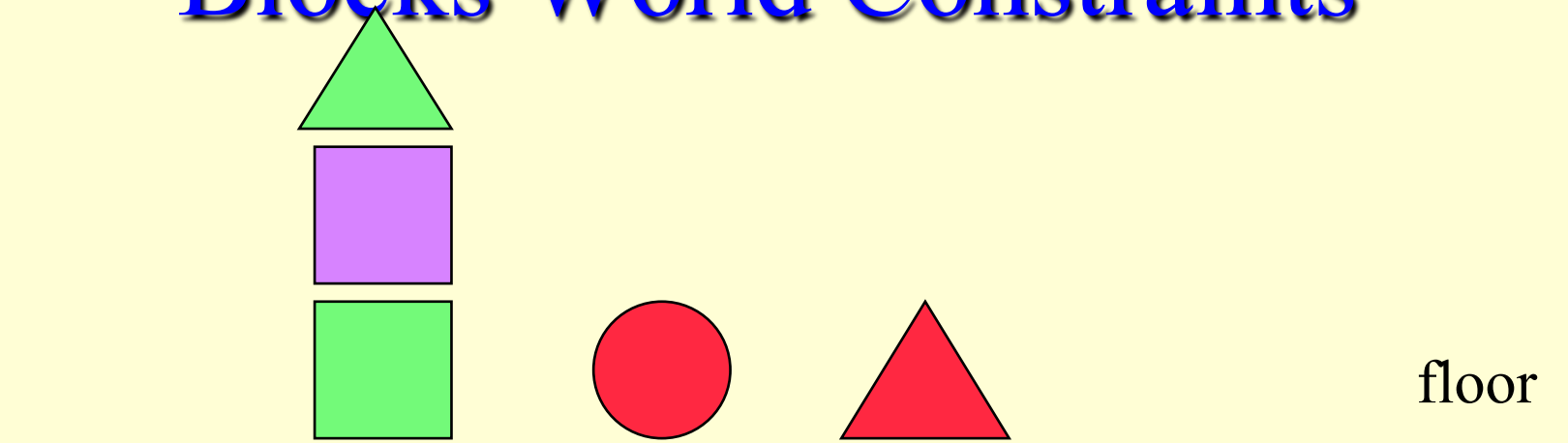
endfor

return *unknown*

Boolean Constraints

- **Something new?**
- The Boolean solver can return *unknown*
- It is **incomplete** (doesnt 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



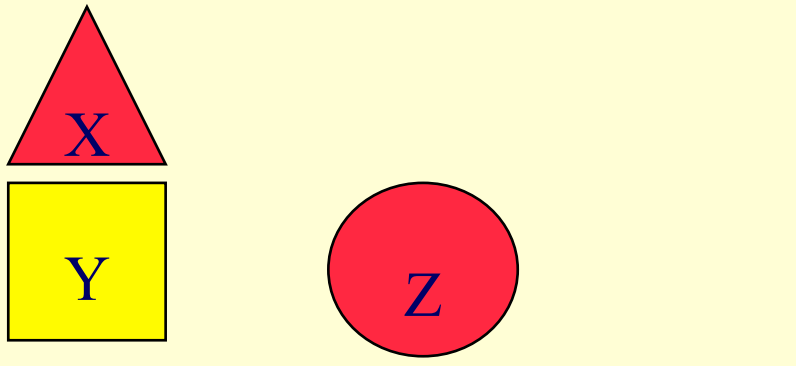
Constraints don't have to be mathematical

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* which 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

$$solv(X > Y \wedge Y > Z) = solv(T > U_1 \wedge U_1 > Z)$$

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 behaviour
- A constraint solver determines if a constraint has a solution
- Real arithmetic and tree constraints
- Properties of solver we expect (well-behavedness)

Chapter 5: Simple Modelling

Simple Modelling

- Modelling
- Modelling Choice
- Iteration
- Optimization

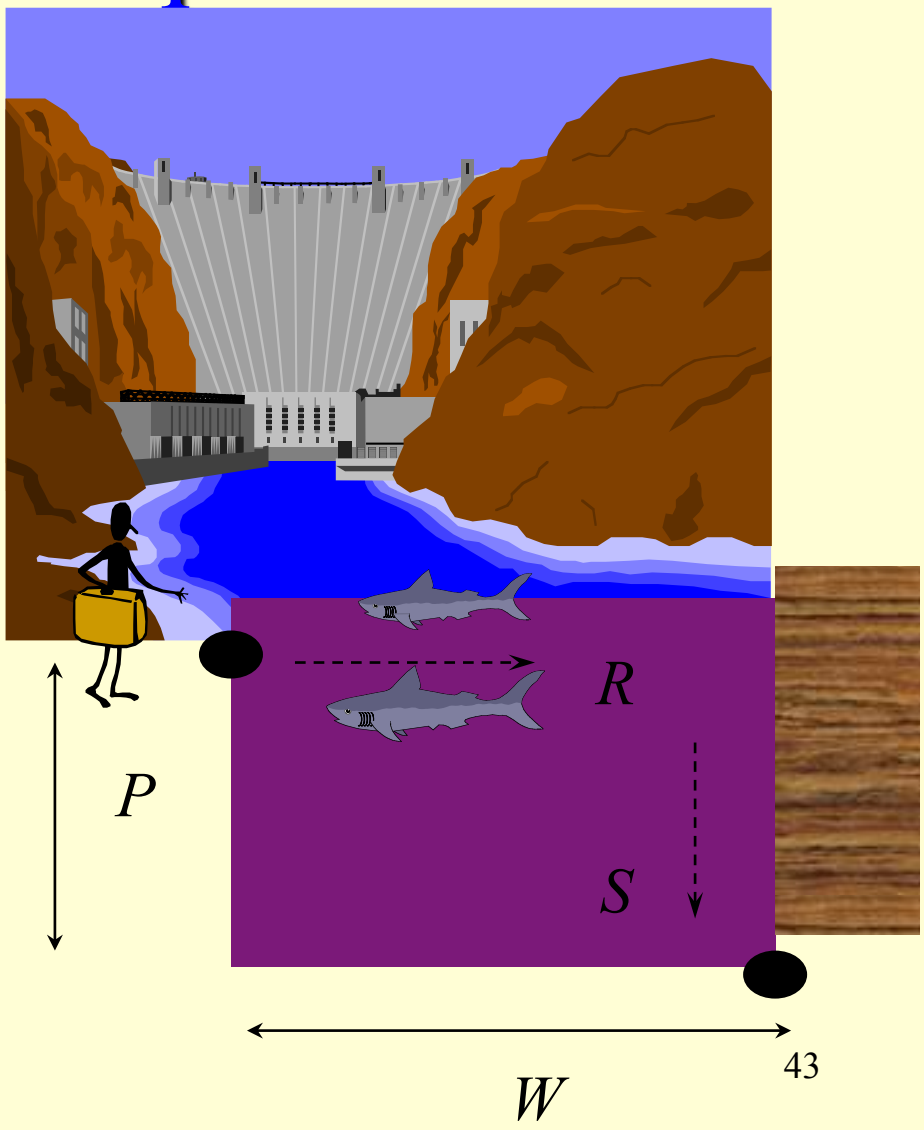
Modelling

- 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

```
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*

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

```
mother(maggy,fi).
mother(fi,lillian).
```

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:

$$X = maggy$$

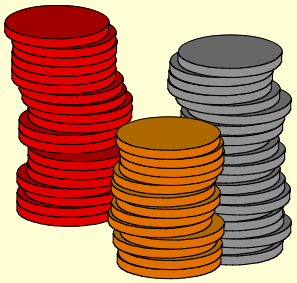
Choice Examples

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).
```

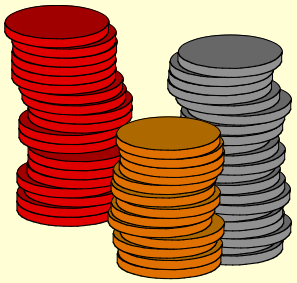
The goal cousin(peter, X) has a single answer $X = fi$

More Complicated Choice

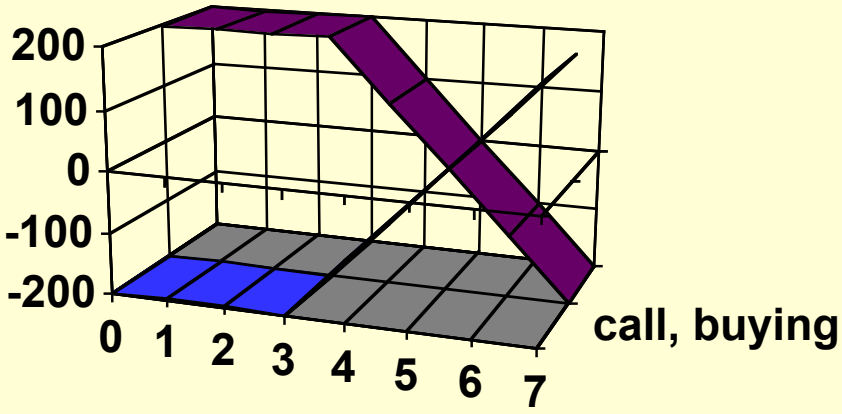


- A **call option** gives the holder the right to buy 100 shares at a fixed price *E*.
- A **put option** gives the holder the right to sell 100 shares at a fixed price *E*
- **pay off** of an option is determined by cost *C* and current share price *S*
- e.g. call cost $200 exercise $300
  - stock price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

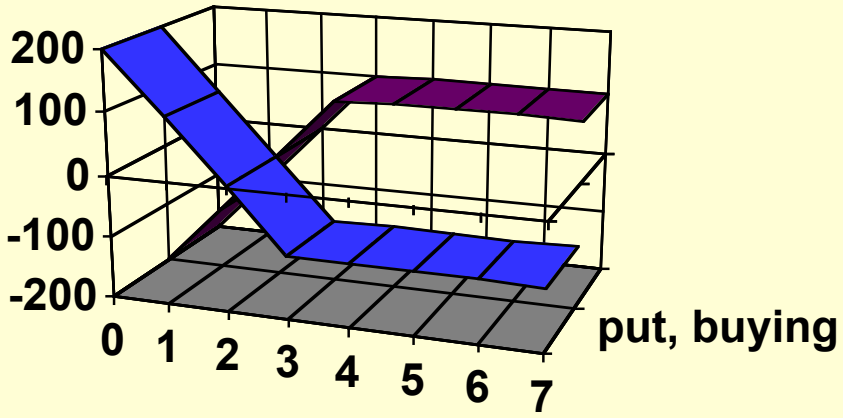
Options Trading

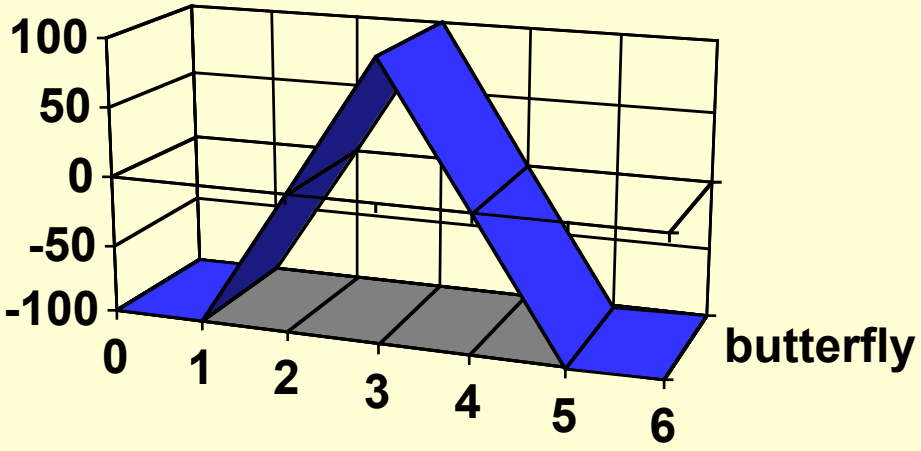


call C=200, E = 300



put C=100, E = 300





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

Modelling 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.
```

Modelling 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),
        put_option(Sell, S, 200, 300, P2),
        call_option(Buy, S, 400, 100, P3).
```

Defines the relationship in previous graph

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

has two answers

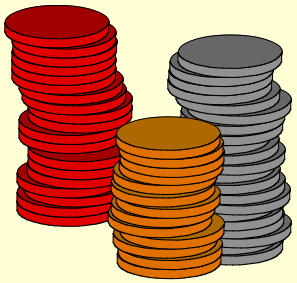
$$P = 100S - 200 \wedge 2 \leq S \wedge S \leq 3$$

$$P = -100S + 400 \wedge 3 \leq S \wedge S \leq 4$$

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:

$$P_1 = P + P \times I - R \wedge$$

$$P_2 = P_1 + P_1 \times I - R \wedge$$

$$P_3 = P_2 + P_2 \times I - R \wedge$$

$$B = P_3$$

Number of constraints depend on the variable T

Reason Recursively

Zero time periods then $B = P$

else new principal $P + P*I - R$ and new time $T-1$

```
mortgage(P,T,I,R,B) :- T = 0, B = P. (M1)
mortgage(P,T,I,R,B) :- T >= 1,
        NP = P + P * I - R, NT = T - 1, (M2)
        mortgage(NP,NT,I,R,B).
```

Example Derivation

$$\langle mortgage(P,3,I,R,B)|true\rangle$$

$$\Downarrow M2$$

$$\langle mortgage(P_1,2,I,R,B)|P_1 = P + P \times I - R\rangle$$

$$\Downarrow M2$$

$$\langle mortgage(P_2,1,I,R,B)|P_1 = P + P \times I - R \wedge P_2 = P_1 + P_1 \times I - R\rangle$$

$$\Downarrow M2$$

$$\langle mortgage(P_3,0,I,R,B)|\ P_1 = P + P \times I - R \wedge P_2 = P_1 + P_1 \times I - R \wedge P_3 = P_2 + P_2 \times I - R\rangle$$

$$\Downarrow M1$$

$$\langle []|P_1 = P + P \times I - R \wedge P_2 = P_1 + P_1 \times I - R \wedge P_3 = P_2 + P_2 \times I - R \wedge B = P_3\rangle$$

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

```
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.
```

Replace tests and assignments by constraints

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, minimize(p(X,Y), X+Y)

Answers: $X = 1 \wedge Y = 0$ and $X = 0 \wedge Y = 1$

X >= 0, X >= Y, minimize(true, X-Y)

Answer: $X >= 0 \wedge X = Y$

minimize(butterfly(S,P), -P)

Answer: $S = 3 \wedge P = 100$

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* | *C1*> to <*G2* | *C2*> where *G1* = *L1,L2,...,Lm*
  - *L1* is *minimize(G,E)*
  - exists solution *v* of <*G* | *C1*> with *v(E)* = *m* and for all other sols *w*, *m* <= *w(E)*
    - *G2* is *G,L2,...,Lm* and *C2* is *C1* /\ *E* = *m*
  - else *G2* is [] and *C2* is *false*

Optimization Example

X >= 0, minimize(X >= Y, X-Y)

$$\langle X \geq 0, \text{minimize}(X \geq Y, X - Y) | true \rangle$$

$$\Downarrow$$

$$\langle \text{minimize}(X \geq Y, X - Y) | X \geq 0 \rangle$$

$$\Downarrow$$

$$\langle X \geq Y | X \geq 0 \wedge X - Y = 0 \rangle$$

$$\Downarrow$$

$$\langle [] | X \geq 0 \wedge X - Y = 0 \wedge X \geq Y \rangle$$

Simplified $X \geq 0 \wedge X = Y$

$$\langle X \geq Y | X \geq 0 \rangle$$

$$\Downarrow$$

$$\langle [] | X \geq 0 \wedge X \geq Y \rangle$$

Minimum value of X-Y is 0 e.g. $\{X \text{ a } 3, Y \text{ a } 3\}$

Optimization

Optimization doesn't only have to be at the goal

```
straddle(S,C1+C2,E,P1+P2) :-
        Buy = 1,
        call_option(Buy, S, C1, E, P1),
        put_option(Buy, S, Cs, E, P2).

best_straddle(C,E,P) :-
        minimize(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