Final Review

See “Midterm Review” for the first half of the semester

Prolog

- Basic Prolog syntax:
  - All the various kinds of “=“.
  - :-
  - ;
  - !
  - Uppercase vs. lowercase
  - _SomeGibberish
  - _
- Prolog superpowers: Depth-first search and unification.
- Left recursion: What will make Prolog dive off the DFS cliff?

Remember:

Prolog tries very hard to be declarative. So:

- There are no functions: nothing is *returned.*
- There are no procedures: you can’t explicitly say “do this, then this, then this.”

• All Prolog can *do* is depth-first search (so, the order of the clauses DOES matter) and unification.

Equality in Prolog

See http://www.swi-prolog.org/pldoc/man?section=arith for exact definitions, but roughly:

- A = B. means: Can they be unified, symbolically?
- A == B. means: Are they exactly the same (barring operator syntax shifts), before evaluation?
- A =:= B. means: Do they evaluate to the same thing? (i.e. arithmetic equal)
- A is B. means: As close as you are going to come in Prolog to assignment of a value to a variable. A is not evaluated, B must evaluate, and they must be unifiable.

Test cases

Write down what you think will happen in each of these cases. Then, test them in Prolog.

```
a(b,c) = a(C,B).
A(b,c) = a(b,c).
a(b(c),C) = a(C,B).
a(b(C),C) = a(C,B).
a(b,c) == a(b,c).
a(B,c) == a(b,c).
a(b,c) =:= a(b,c).
4+5 =:= 10-1.
4+5 is 10-1.
X is 10-1.
a(X,c) is a(10,c).
```

Lists

- [a, b, c(d), 4, 4+5, [], [a, b]]
- [H|R] form is equivalent to CAR and CDR in LISP
    - [a, b, c, d] = [H|R].
    - Unifies to: H = a, R = [b, c, d]
- Can put multiple items in the head (but not the tail).
    - [A, B, C | T] = [1, 2, 3, 4, 5].
    - Unifies to: A = 1, B = 2, C = 3, T = [4,5]
- [H|R]=[a,b,c,d]. also works.
- Can embed in a predicate. Try these:
    - a([b,C], d) = a([D, 3], d).
    - a([b,C], D) = a(D, b).

In-class exercise: Let’s recurse in Prolog!

- ismember/2. Checks if its first argument (an atom) is a member of its second item (a list).
- sum_up/2. Takes a list of numbers as its first argument, and returns the sum of the numbers as its second argument.
- myappend/3. The third argument is the first two arguments, appended.
- reverselist/2. Reverses the order of its first argument and returns the reversed list as its second argument. [Hint: might want to create a second predicate with *3* arguments to do all the work. See discussion here: http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse25]

Note that member/2, append/3, and reverse/2 are built-in functions. Don’t use in assignments.

Natural Language

Levels of analysis

- Prosody: the ‘tune’ of the language.
- Phonology: the sound of the language.
- Morphology: parts of words, and how they fit together.
- Syntax/grammar: how meaningful strings of language are constructed.
- Semantics: the meaning of each valid string.
- Pragmatics: meaning in conversational context.
- Dialogue: understanding an entire exchange.

Stages of analysis

ANALYSIS	PRODUCT
Speech analysis	Text, annotated text…
Syntactic analysis	Parse tree…
Semantic analysis	Predicate logic, semantic network…
Pragmatic/dialogue analysis	Database query language, specialized translation representation…

Ambiguity

Ambiguity is when two or more interpretations are produced at some stage of processing.

e.g. syntactic ambiguity:

- I watched Mary with a telescope.

e.g. pragmatic ambiguity:

- Do you have the time?

e.g. phonological ambiguity:

- Bear/bare
- Etc.

Parsing strategies

- Parsing is like any other kind of search, with rules used to generate nodes
- Can be done top-down or bottom-up, breadth-first or depth-first
- Unlike some searches, we want to find ALL solutions (parses), so that we can use other strategies to determine which one is correct.
- Problem: How do we know when to terminate the search? cf. left-recursive rules

Left-recursive rules

S → NP VP
VP → V NP
NP → AP NP
NP → N
AP → AP A
AP → A

N → cats
V → love
A → fat

Try a top down and bottom up search, and parse: “Cats love fat cats.” Do you have a problem? Describe it. How would Prolog handle this?

Agreement in natural language

What’s wrong with these sentences?

- The cat love dogs.*
- The cats loves a dogs.*
- Cat love some dog.*
- The cat ate and drink.*

Most natural languages require some kind of *agreement* between the words in the sentence.

One way to handle agreement

S → SingNP SingVP.

SingVP → SingV SingNP.

S → PlurNP PlurVP.

PlurVP → PlurV SingNP.

SingNP → SingDet SingNoun.

PlurNP → PlurDet PlurNoun.

SingVP → SingV PlurNP.

...and so on! Problem: You end up with a *lot* of rules, especially when you consider all the kinds of agreement that matter in English. Also, it doesn’t capture the generalities of the language.

A better way to handle agreement

Use ***features***. Features keep track of important linguistic attributes like number, tense, person and so on. This way, the grammar need only say that the features must be the same, without saying what they should be. For example:

S → NP(number=X) VP(number=X).

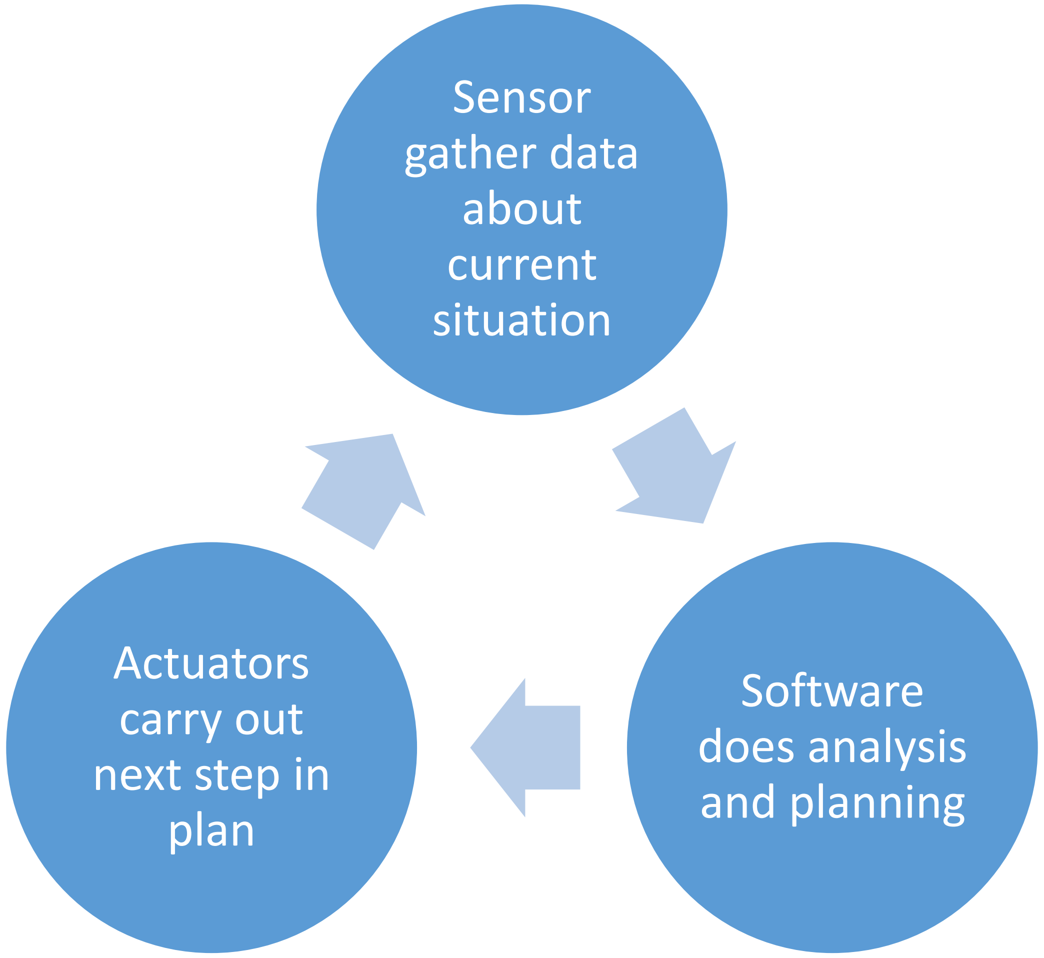
Covers both cases (number=plural and number=singular).

Robotics

Pros and Cons of Robots

So, what are they? Especially compared with humans!

The sense-think-act cycle


Sensor gather data about current situation
Software does analysis and planning
Actuators carry out next step in plan

Pros and Cons

Pro:

- Easy to understand
- Modular at a high level
- Cautious (always working with as much information as possible)

Con:

- Can be very slow
- Stop-and-start
- Doesn’t handle surprises well
- Poor for dynamic reactive processes (e.g. walking)
- Not what people/animals really do

Behavior-based/reactive

- Tight coupling between sensors and actuators, with little “thinking” in between
- Each part is responsible for its own task (e.g. foot must find stable position, leg must balance and push etc.)
- Minimal central control
- Overall behavior is emergent

Pros and Cons

Pro:

- Handles dynamic situations well
- Modular at the level of components
- No micro-management
- Relatively fast
- Reactive

Con:

- Sometimes unpredictable
- Very hardware specific
- Hard to program?
- High-level control needed at some point

Planning

What is planning?

Planning is an AI approach to control

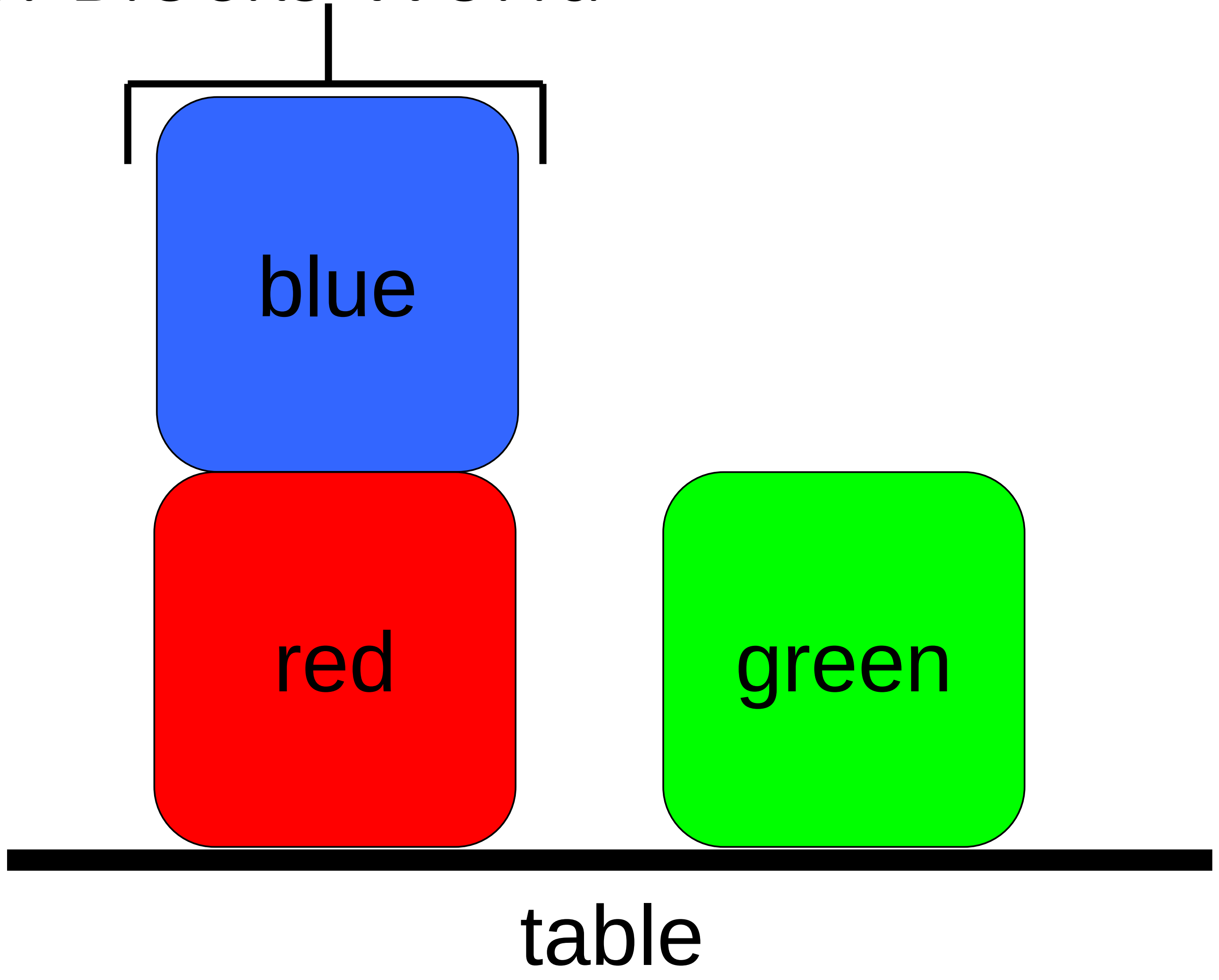
Deliberation about action

Key ideas

- We have a **model** of the world
- Model describes **states** and **actions**
- Give the planner a **goal** and it outputs a **plan**
- Aim for **domain independence**

Planning is search

Example: Blocks World



Representing States

States are described by conjunctions of ground predicates (possibly negated).

`on(blue, red)` $\wedge \neg$ `on(green, red)`

The **closed world assumption** (CWA) is employed to remove negative literals:

`on(blue, red)`

That is, everything that is not explicitly stated to be true is assumed to be false.

The state description is **complete**.

Representing Goals

The goal is the specification of the task

A goal is a usually conjunction of predicates:

`on(red, green)` ∧ **`on_table(green)`**

The CWA does **not** apply.

So the above goal could be satisfied by:

`on(red, green)` ∧ **`on_table(green)`** ∧ **`on(blue, red)`** ∧ **`clear(blue)`** ∧ …

Representing Actions

Actions are described in terms of **preconditions** and **effects**.

Preconditions are predicates that must be true **before** the action can be applied.

Effects are predicates that are made true (or false) **after** the action has executed.

Sets of similar actions can be expressed as a **schema** (roughly, actions with variables).

STRIPS operators

An early but still widely used form of action description is as “STRIPS operators”.

Three parts:

Precondition	A conjunction of predicates
Add-list	The set of predicates made **true**
Delete-list	The set of predicates made **false**

Blocks World Action Schema

`move(`*`block`*`, `*`from`*`, `*`to`*`)`

Pre:

`on(`*`block`*`, `*`from`*`), clear(block),clear(`*`to`*`)`

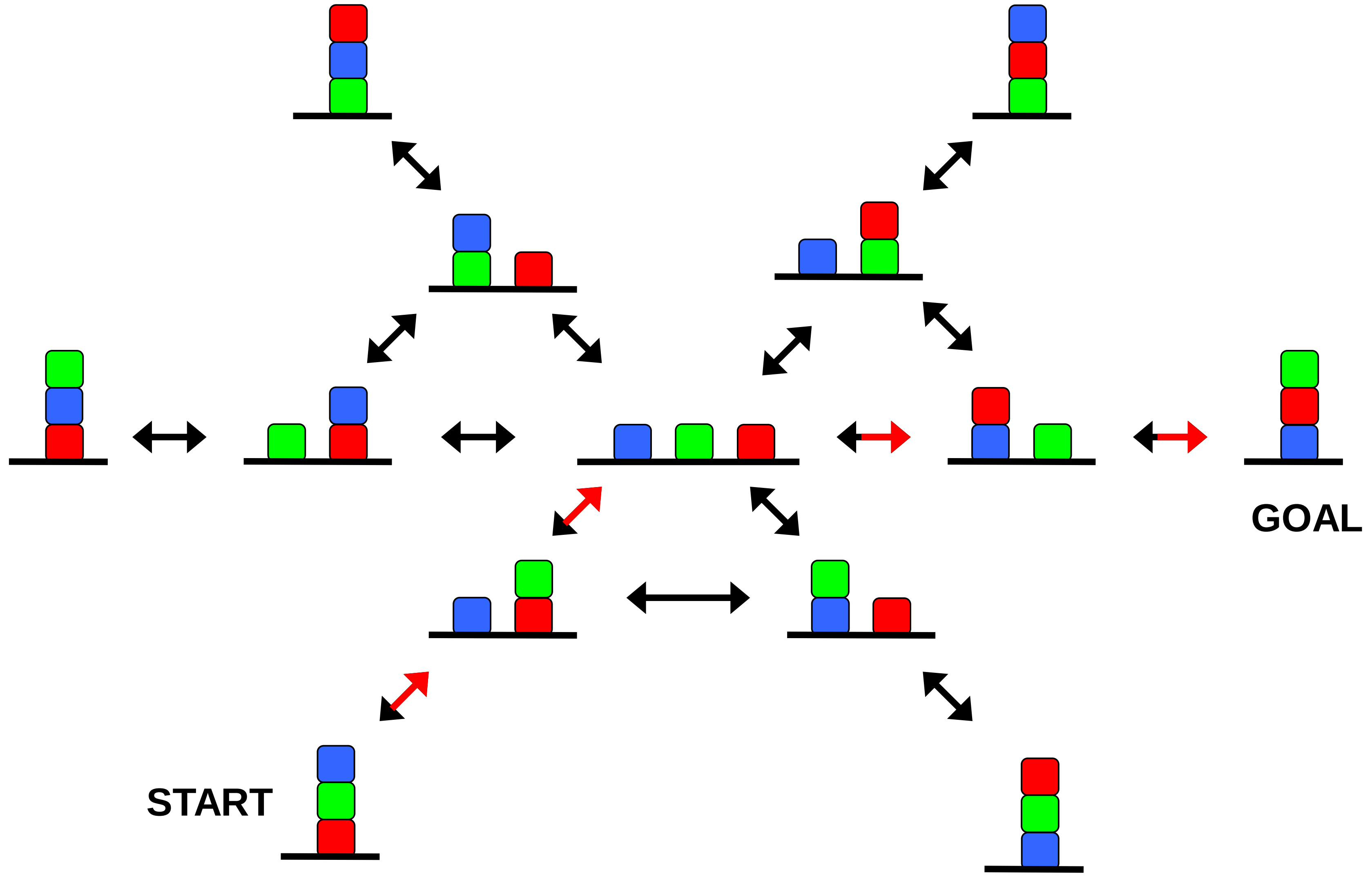
Add:

`on(`*`block`*`, `*`to`*`), `*`clear(from`*`)`

Del:

`on(`*`block`*`, `*`from`*`), clear(`*`to`*`)`

Graph of state space



Heuristics for planning

Remember A*: we want an (under) estimate of the distance to a goal. One approach: do a quick plan (for the node to be evaluated) under relaxed constraints. For example:

- ***Ignore preconditions:*** How many steps would it take if actions had no preconditions?
- ***Ignore delete lists:*** How many steps would it take if actions had no delete lists?

Are these admissible? Do you see any issues?

Evolutionary Computation

Natural evolution

Adaptation based on a combination of *competition* (e.g. for food), *selection* (i.e. surviving long enough to be able to reproduce in a given enviroment), *mutation* (i.e. spontaneous change in the gene itself), and *reproduction* (i.e. producing a new individual with the same or similar genetic makeup).

What are GAs?

- A class of stochastic (i.e. probabilistic) search algorithms based on biological evolution.
- Rely on well-defined termination criteria and fitness functions.
- Nodes in the search space (i.e. potential solutions) are represented as *chromosomes* (typically a binary series). Each element in a chromosome is called a *gene*.

A GA run

1. Initialization. Represent the problem domain as a chromosome of length L. Choose population size (N), crossover probability (p_c) and mutation probability (p_m).
2. Define a *fitness function*, which quantitatively measures the success of an individual chromosome.
3. Randomly generate an initial population of size N.
4. Calculate the fitness of each chromosome. If the termination condition is satisfied, stop. Otherwise, continue.

A GA run (2)

5. Select parent chromosomes probabilistically, based on their fitness.
6. Apply the genetic operators (crossover and mutation)
7. Place the generated offspring in the new population. Repeat from 5 until population size is N.
8. Repeat from 4, with the new population.

Representing candidate solutions as chromosomes

- A chromosome is typically a binary string (although other representations exist).
- If potential solutions are numbers, or series of numbers, then encoding in this representation is straightforward.
- If potential solutions are more complex entities, **encoding can be the hardest part of the problem**!

The fitness function

- Should be a relatively simple (i.e. quick to calculate) measure of an individual’s fitness (i.e. how close it is to a solution).
- A domain-specific, heuristic measure
- In a GA, N individuals are evaluated in each generation, and there are typically many (hundreds or thousands) of generations – so a complex fitness function can slow things down a lot.
- Coming up with a good fitness function can also be very difficult! Try a weighted sum of desirable features…

The crossover operator

Corresponds to bisexual reproduction in natural selection (typically, p_c=.7). If it fires:

1. A point in the length of the parent chromosomes is randomly selected.
2. The two offspring are:
    1. The first part of Parent A plus the second part of Parent B, and
    2. The first part of Parent B plus the second part of Parent A.

If it doesn’t fire, the offspring are (typically) simply reproductions of the parents.

The mutation operator

Represents random mutation in nature. Typically very low probability (e.g. p_m=.001). If it fires:

1. A gene in the chromosome is randomly chosen.
2. It is flipped to the opposite value (or, if the chromosome is not binary, some other change function is applied).

If this operator doesn’t fire, the chromosome is unaffected.

Why have mutation?

- Helps knock the population out of local maxima.
- Introduces diversity, just in case the best solution can’t be reached from the initial population.

Elitism

If the GA is set to use *elitism*, the most fit members of the current population are simply copied over to the new population, before the other operators are applied. What are the pros and cons?

What is genetic programming?

- Like GAs, GPs use evolutionary techniques to solve problems.
- Unlike GAs, the solutions are *programs.*
- The goal is to produce programs which can solve problems, without explicitly programming them.
- Chromosomes are both *data* (i.e. can be manipulated) and *programs* (i.e. can be run to determine fitness).