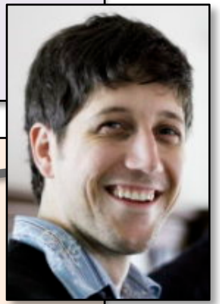
Lectures

Genetic Programming
Evolving programs with evolutionary algorithms

Cellular Automata
Programs that look like biological systems

Gene Regulatory Models
Programs that look even more like biological systems

Evolvable Hardware
Evolving at the physical level on electronic devices



Assessment

Learning Outcomes: Subject Mastery	*Understanding, Knowledge and Cognitive Skills; Scholarship, Enquiry and Research (Research-Informed Learning)* ♦ Understanding of limitations of traditional computation. ♦ A critical understanding of a range of biologically inspired computation methods, their limitations and areas of applicability. ♦ Ability to apply one or more biologically inspired techniques in solving a practical problem.
Learning Outcomes:: Personal Abilities:	*Industrial, Commercial & Professional Practice; Autonomy, Accountability & Working with Others; Communication, Numeracy & ICT* ♦ Identify and define approaches that can be used to apply bio-inspired methods to existing problems in optimisation and machine learning. ♦ Exercise substantial autonomy and initiative (both courseworks) (PDP) ♦ Demonstrate critical reflection (both courseworks) (PDP).

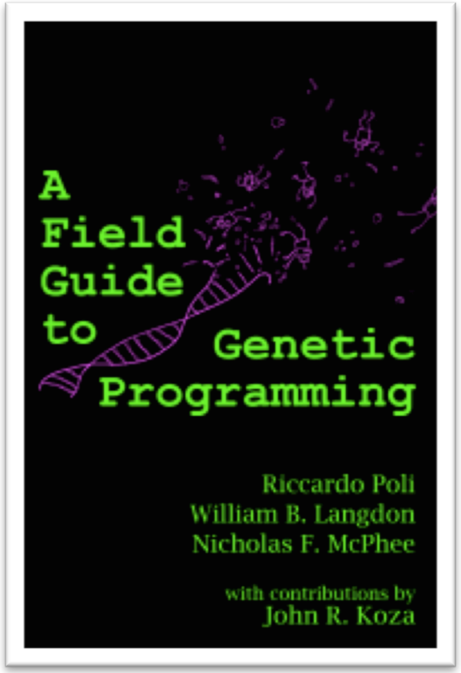
Fundamentals of Genetic Programming

Dr. Michael Lones
Room EM.G31
M.Lones@hw.ac.uk

Books – free to download

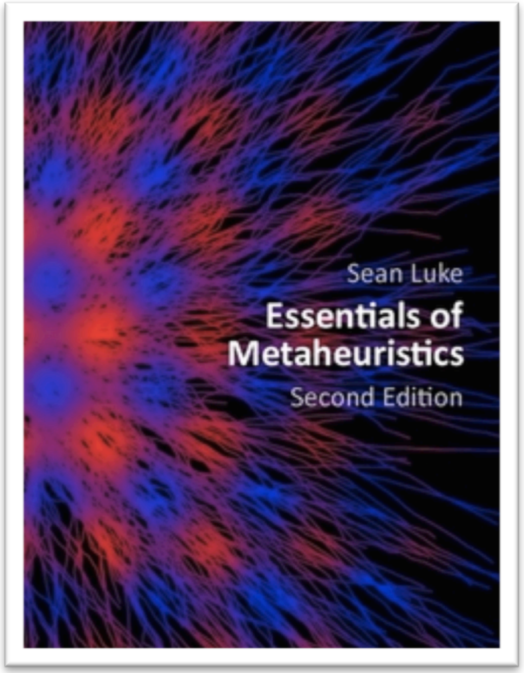
- Written by leading researchers in the field…

R. Poli et al, A Field Guide to Genetic Programming



www.gp-field-guide.org.uk

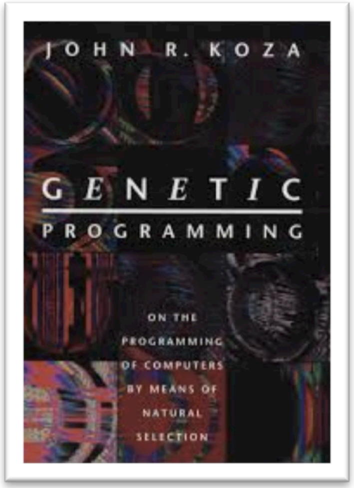
S. Luke, Essentials of Metaheuristics

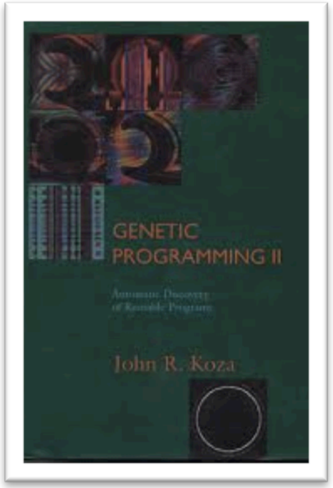


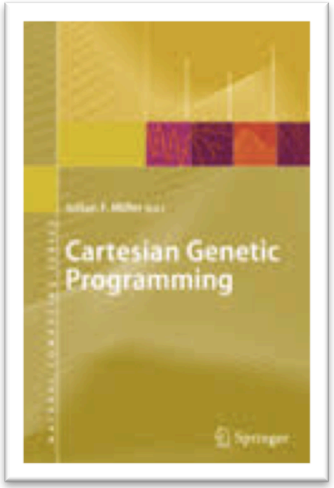
http://cs.gmu.edu/~sean/book/metaheuristics/

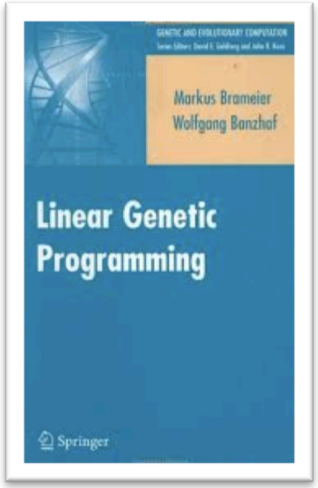
Books

- Not essential, though may be of interest…









John Koza, Genetic Programming & Genetic Programming II

Both in the library

Julian Miller Cartesian Genetic Programming

http://link.springer.com/book/10.1007/978-3-642-17310-3

Brameier&Banzhaf Linear Genetic Programming

http://link.springer.com/book/10.1007%2F978-0-387-31030-5

Software – free to download

- Download from http://cs.gmu.edu/~eclab/projects/ecj/

ECJ 21

A Java-based Evolutionary Computation Research System

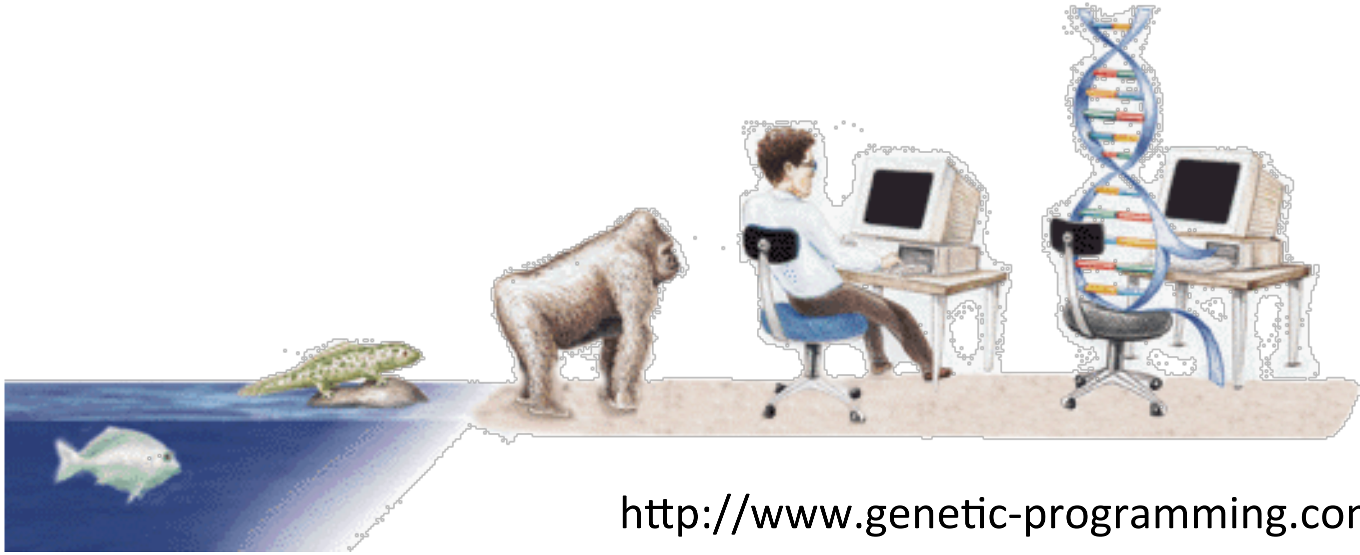
By Sean Luke, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Rafal Kicinger, Elena Popovici, Keith Sullivan, Joseph Harrison, Jeff Bassett, Robert Hubley, Ankur Desai, Alexander Chircop, Jack Compton, William Haddon, Stephen Donnelly, Beenish Jamil, Joseph Zelibor, Eric Kangas, Faisal Abidi, Houston Mooers, James O'Beirne, Khaled Ahsan Talukder, and James McDermott

ECJ

- A Java-based framework for evolutionary computing
  - Supports common evolutionary algorithms
    - GAs, evolution strategies, GP, PSO, …
    - You just need to implement a **Problem** subclass
  - Individual components are configurable
    - Using parameter files
    - Representations, operators, selection mechanisms
  - Relatively easy to evolve non-standard things
    - New representations subclass **Individual** and **Species**
    - New variation operators subclass **BreedingPipeline**

Genetic Programming (GP)

- In a nutshell, using evolutionary algorithms to design computer programs
  - Or other ‘executable structures’, e.g. circuits, equations
  - Generally small programs that do specific things
  - So we wouldn’t expect to evolve Microsoft Office



http://www.genetic-programming.com

Genetic Programming (GP)

- In a nutshell…

> - Create a population of random programs
> - Then repeat:
>   - Evaluate them
>   - Kill off the (really) bad ones
>   - Keep the (relatively) good ones
>   - Use them to breed the next generation (by using mutation and recombination operators)
> - Until the problem is (hopefully!) solved

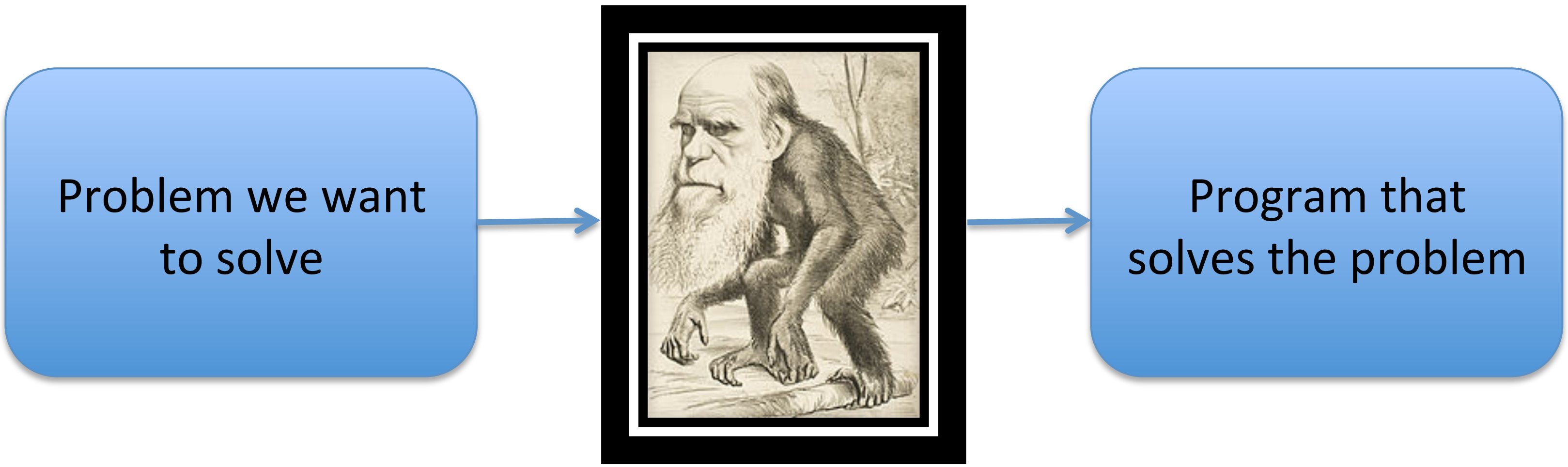
Genetic Programming (GP)

- Why use evolutionary algorithms?
  - Good at solving global optimisation problems
  - Flexible in how solutions are represented
  - However, focus on EAs is in part historical
  - Other optimisers may, in principle, be used

- Also a slightly iffy bio-inspired argument
  - Biological systems are evolved
  - Biological systems are, in a sense, complex computers
  - Therefore complex computations can be evolved ☐

Genetic Programming (GP)

- Why do we want to evolve programs?
  - Sometimes because we're lazy!
  - More often because we don't know how to write a program to solve a particular problem
  - Or we want to do better than an existing solution

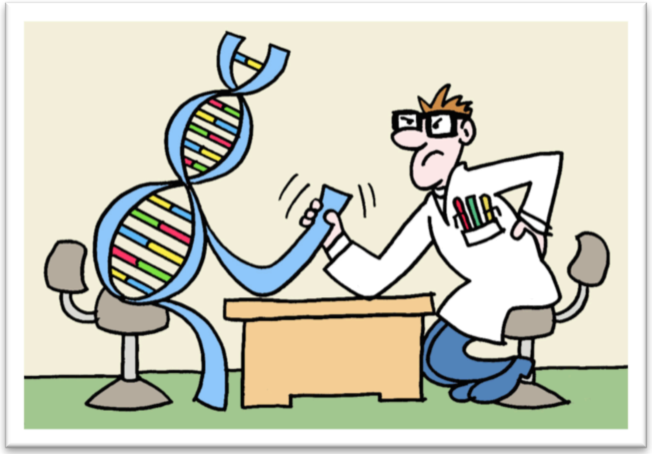


Evolutionary 'black box'

Genetic Programming (GP)

- Often portrayed as a form of automatic innovation
  - http://www.human-competitive.org/
  - “Humies” is an annual contest for human-beating results
  - $10,000 in prizes every year

- Previous Humies winners include:
  - Games controllers
  - Circuit designs/designers
  - Image analysis algorithms
  - Software engineering tools
  - Medical diagnostics tools



Genetic Programming (GP)

- There are a number of varieties of GP
  - You'll see lots of these over the coming lectures
- They differ in how they represent programs
  - Syntax: control structures, modules, language
  - Also their degree of bio-inspiration
- Representation is important
  - The programs we write are fragile
  - Imagine "mutating" one ➔
  - Can we remove this fragility?? (this is a big research question)

http://en.wikipedia.org/wiki/File:Windows_NT_BSOD_at_GVA_baggage_claim,_1999-10-03.jpg

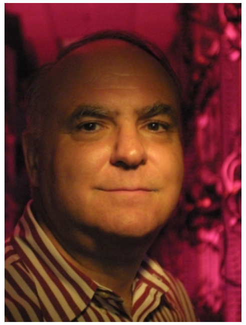
Evolvability

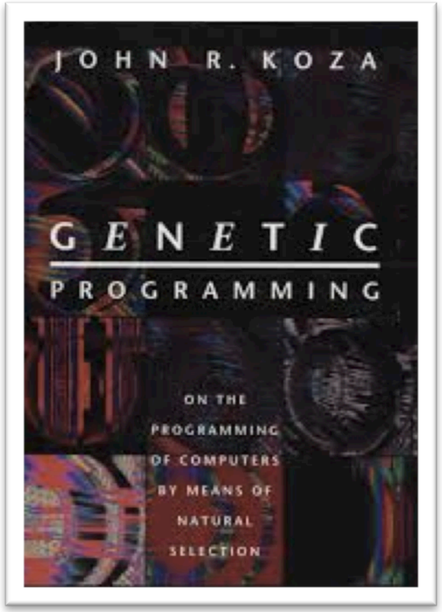
This is the capacity for a program to improve its fitness as a result of an evolutionary process (i.e. mutation and recombination).

For genetic programming, there's little value in being theoretically able to express a program if it can not be discovered by evolution.

Koza Tree-Based GP

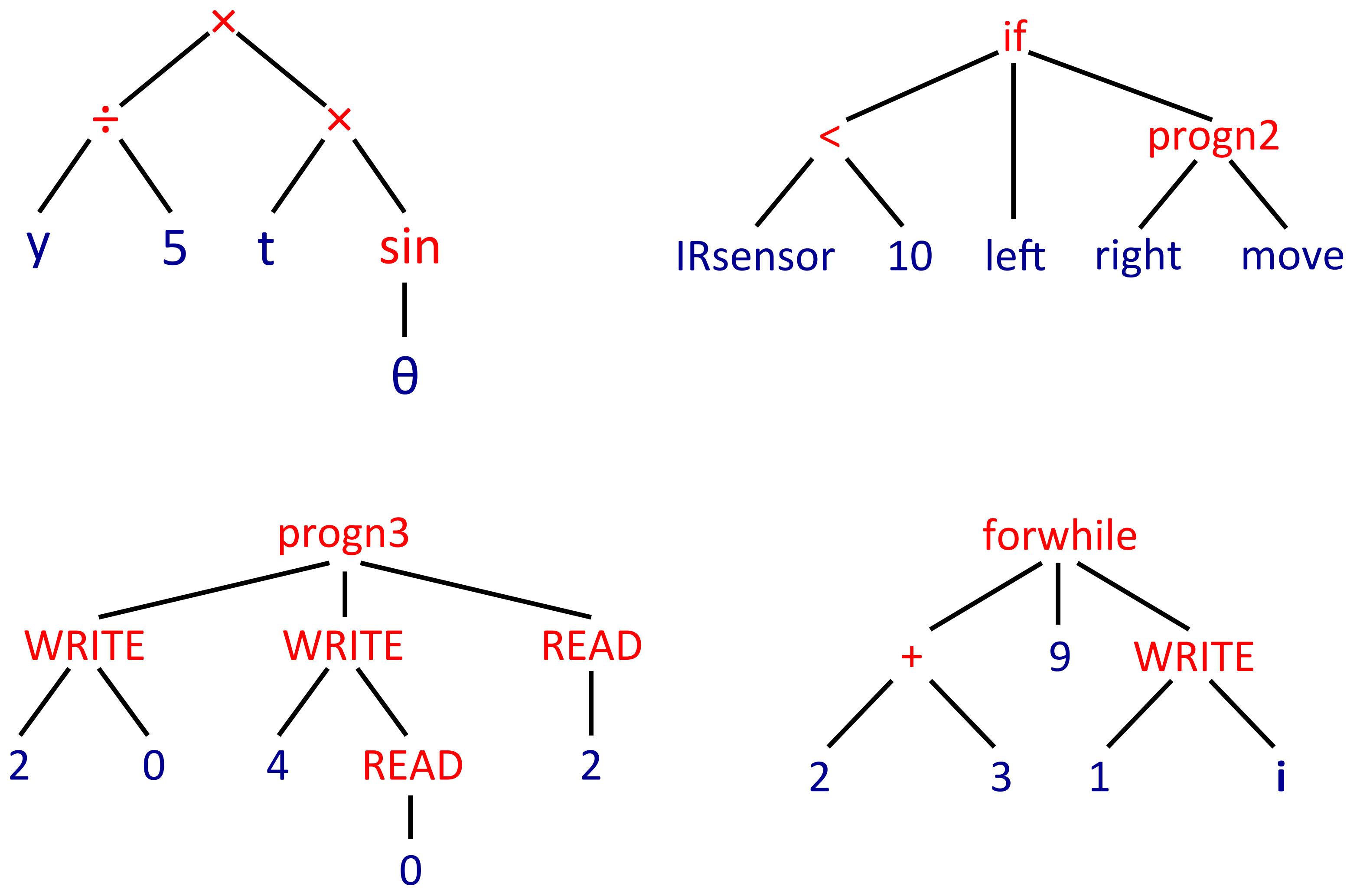
- Invented by John Koza
  - Also invented the scratch card
  - Earliest successful form of GP
  - (Though arguably not the first)
  - Still the most widely used form





- Programs are represented by trees
  - Also known as *syntax trees* or *parse trees*
  - Internal nodes are sampled from a **function set**
  - Leaves are sampled from a **terminal set**

Parse Trees


×
÷
×
y
5
t
sin
θ
if
<
progn2
IRsensor
10
left
right
move
progn3
WRITE
WRITE
READ
2
0
4
READ
0
2
forwhile
+
9
WRITE
2
3
1
i

Initialisation

- To create a mathematical expression
  - Function set = { +, -, ×, ÷, sin, cos }
  - Terminal set = { y, t, θ }

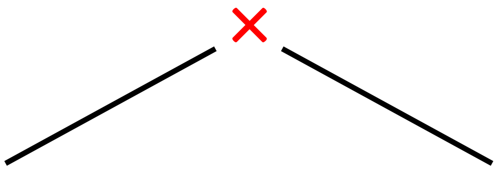
Initialisation

- To create a mathematical expression
  - Function set = { +, -, ×, ÷, sin, cos }
  - Terminal set = { y, t, θ }

×

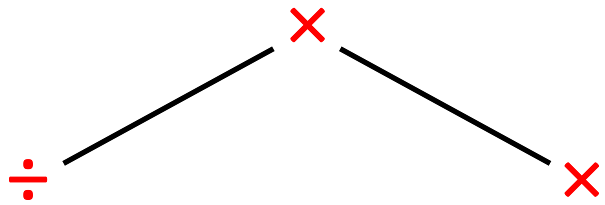
Initialisation

- To create a mathematical expression
  - Function set = { +, -, ×, ÷, sin, cos }
  - Terminal set = { y, t, θ }



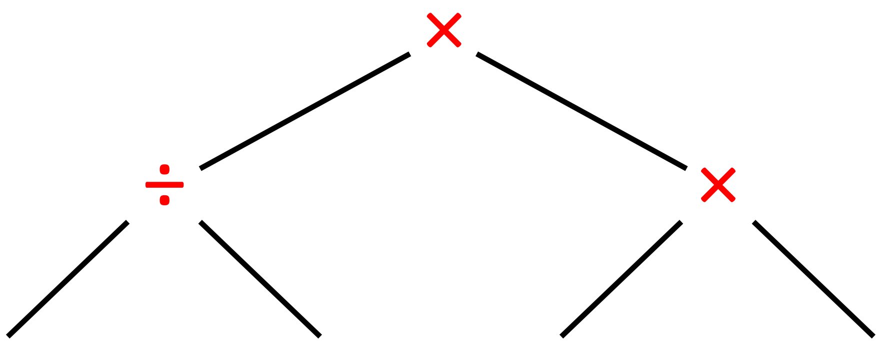
Initialisation

- To create a mathematical expression
  - Function set = { +, -, ×, ÷, sin, cos }
  - Terminal set = { y, t, θ }



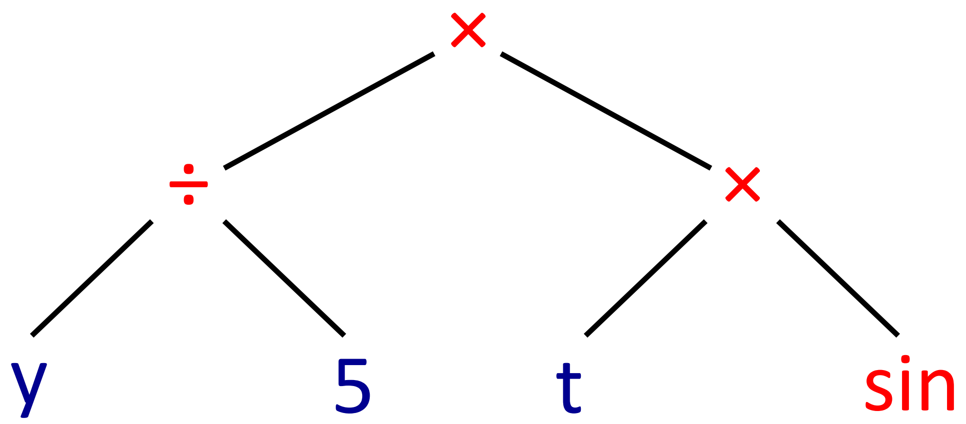
Initialisation

- To create a mathematical expression
  - Function set = { +, -, ×, ÷, sin, cos }
  - Terminal set = { y, t, θ }



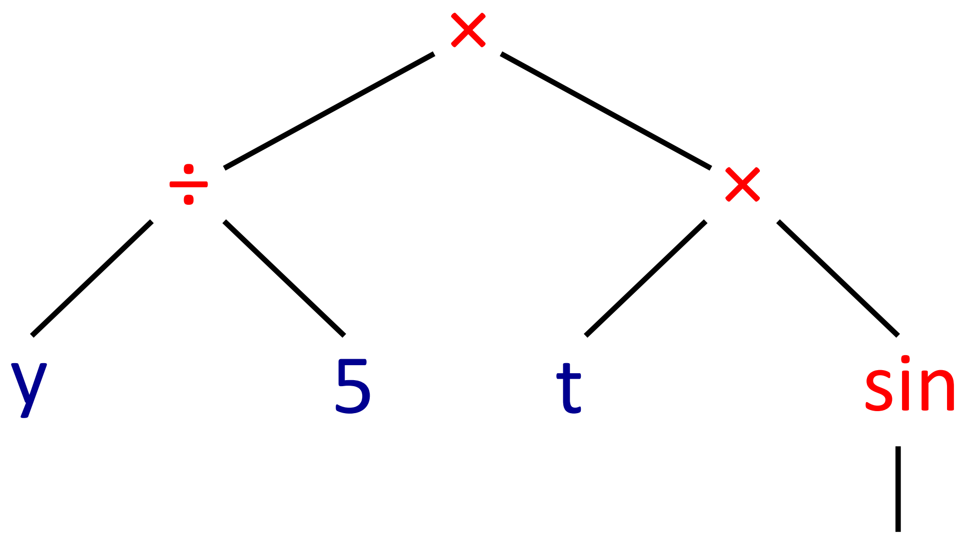
Initialisation

- To create a mathematical expression
  - Function set = { +, -, ×, ÷, sin, cos }
  - Terminal set = { y, t, θ, constant }



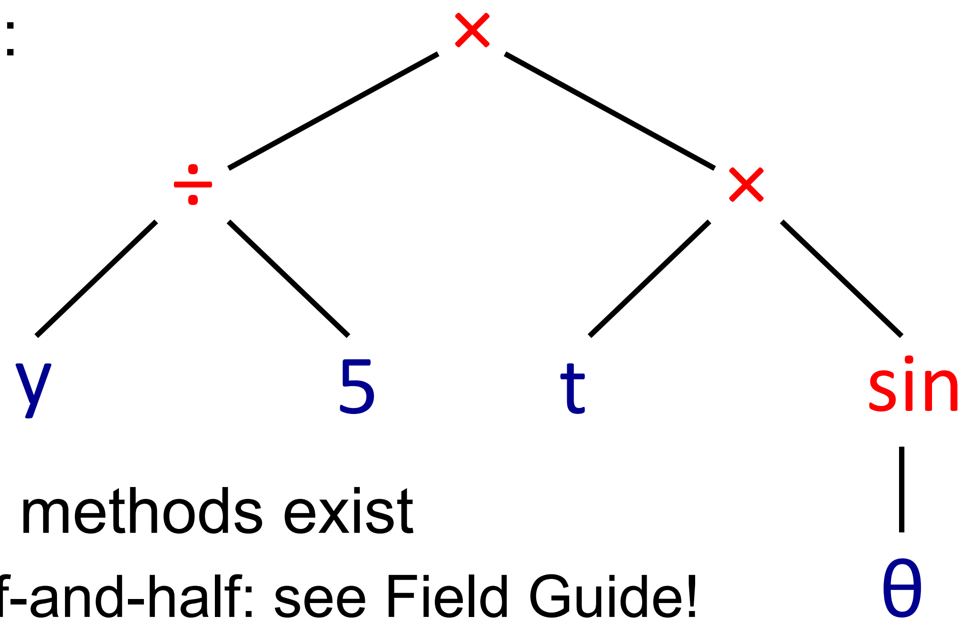
Initialisation

- To create a mathematical expression
  - Function set = { +, -, ×, ÷, sin, cos }
  - Terminal set = { y, t, θ, constant }



Initialisation

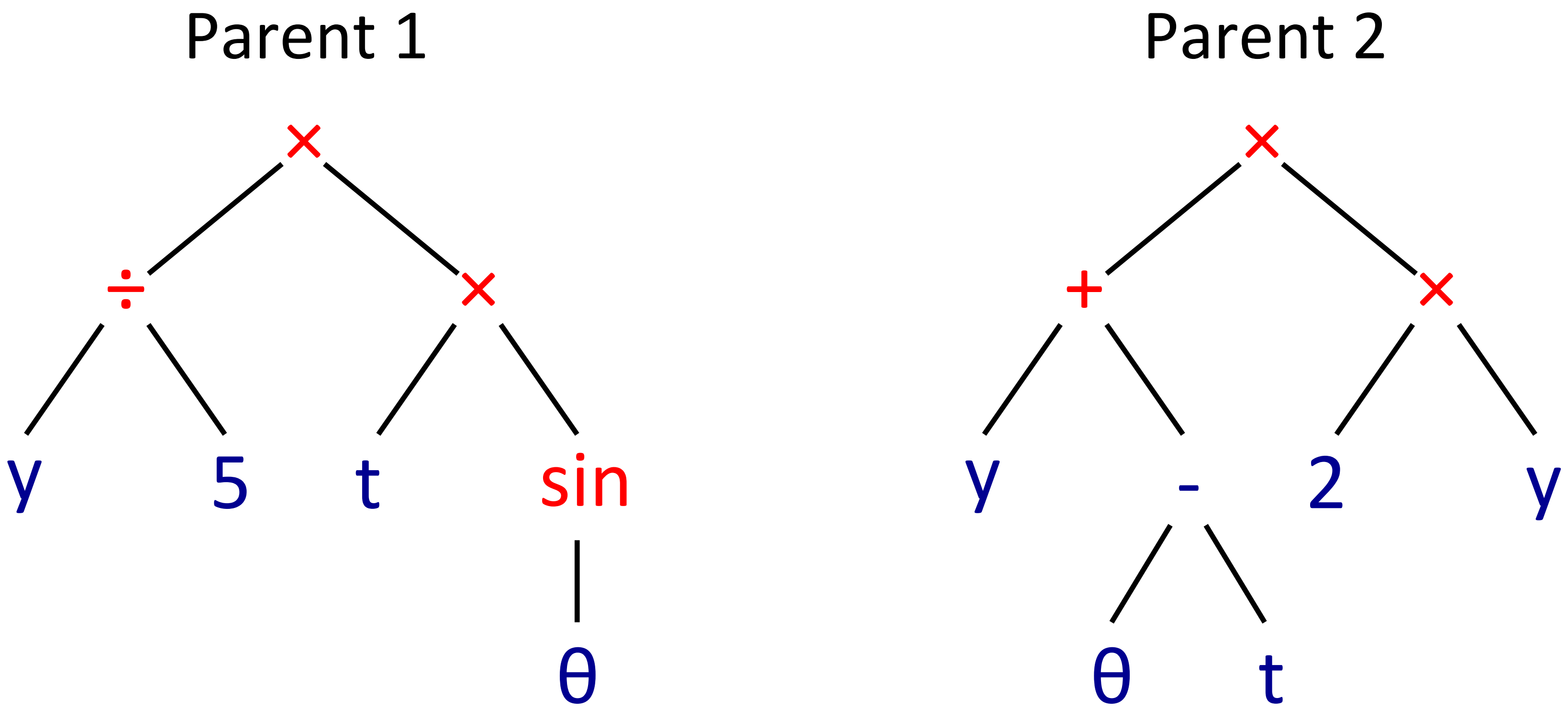
- To create a mathematical expression
  - Function set = { +, -, ×, ÷, sin, cos }
  - Terminal set = { y, t, θ, constant }
  - e.g. (y/5)*(t sin θ) :



  - Other initialisation methods exist
    - E.g. ramped half-and-half: see Field Guide!

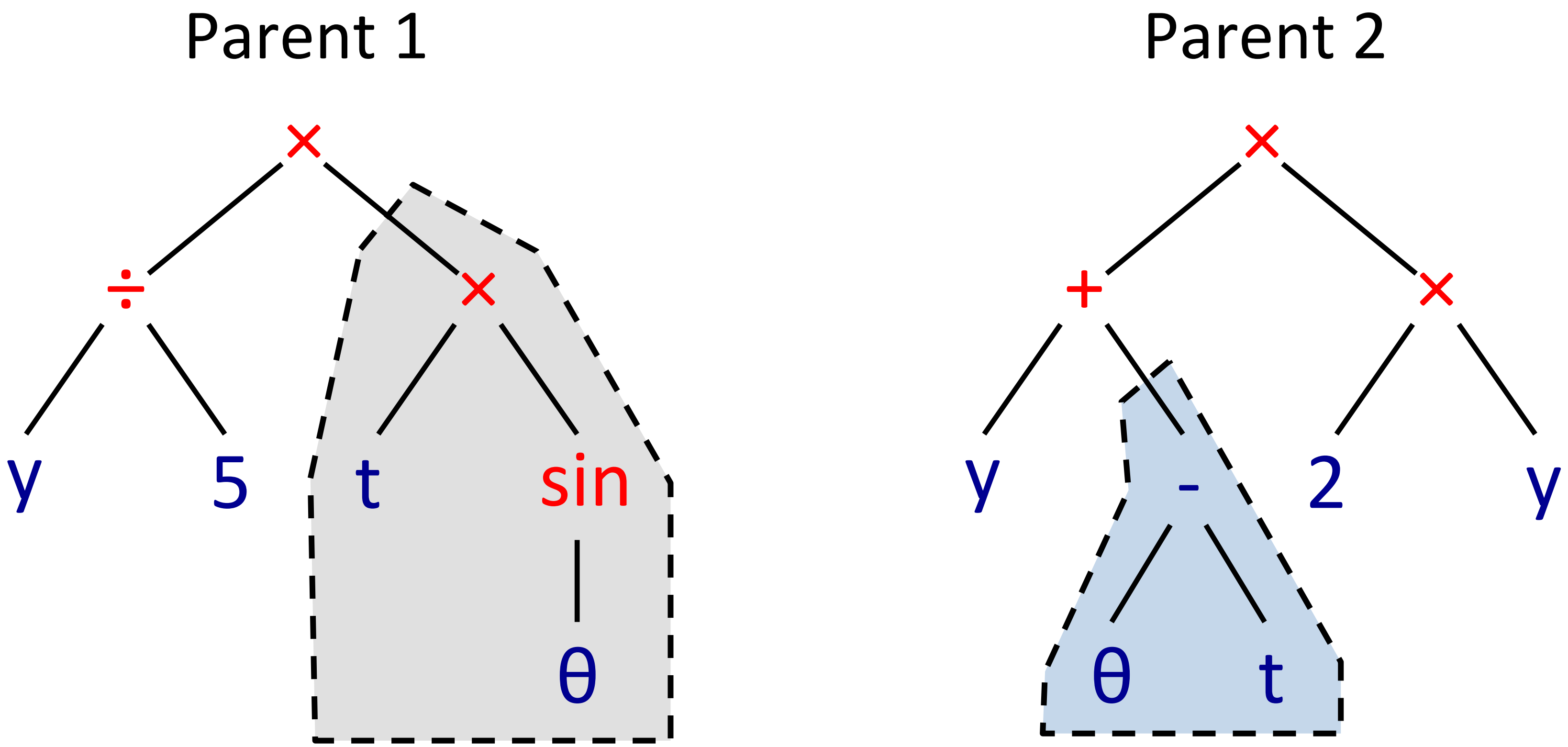
Recombination

- Sub-tree crossover:



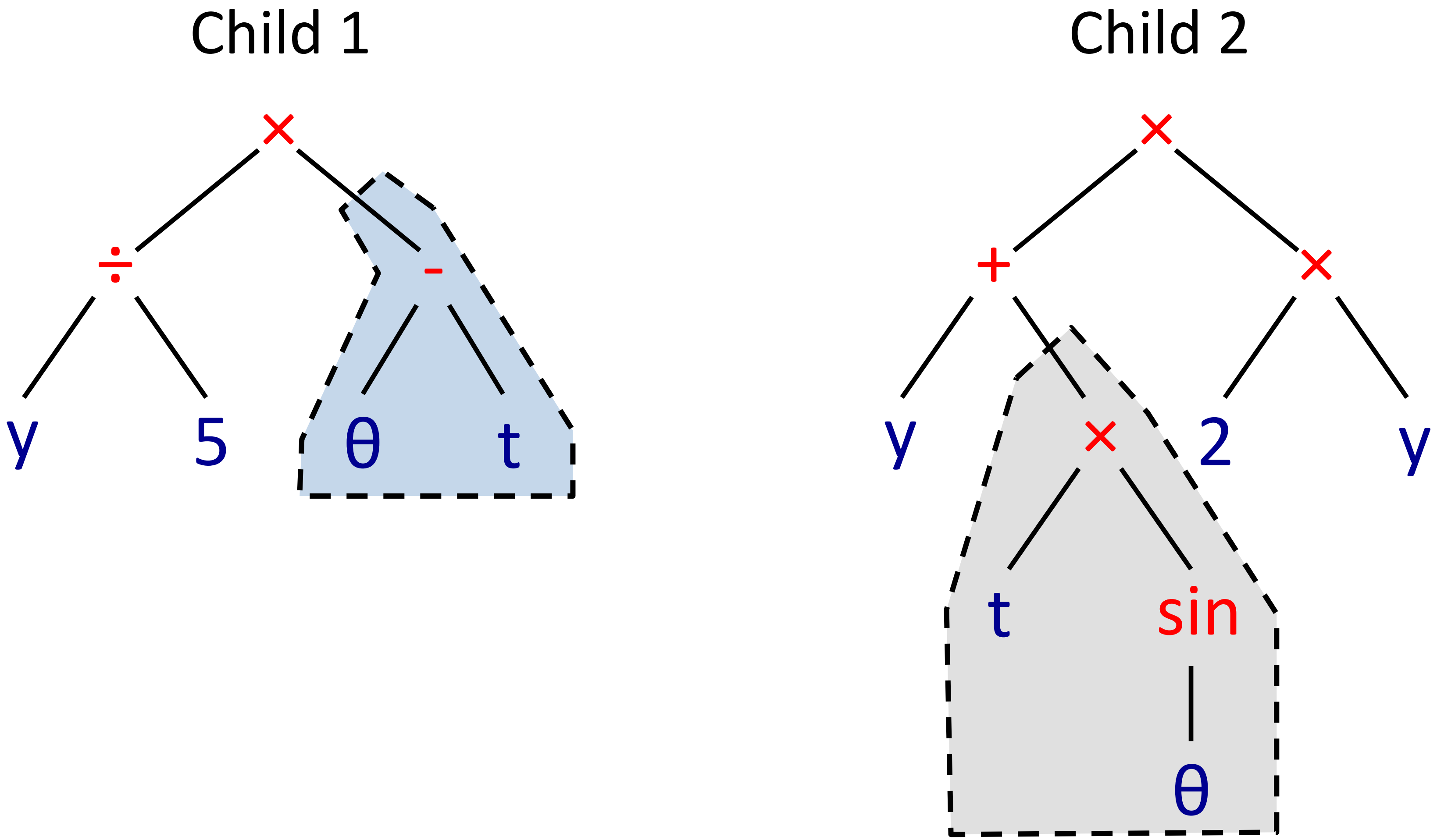
Recombination

- Sub-tree crossover:



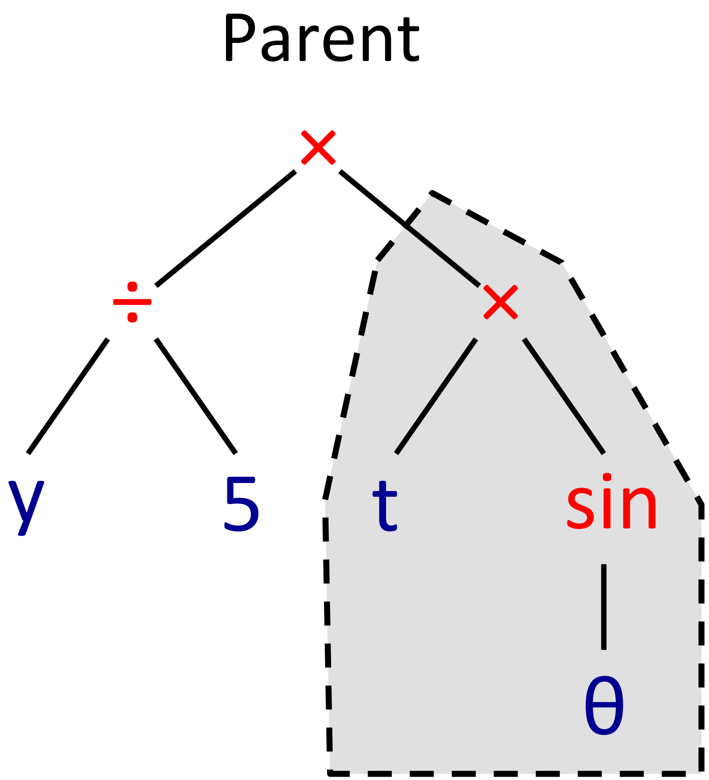
Recombination

- Sub-tree crossover:



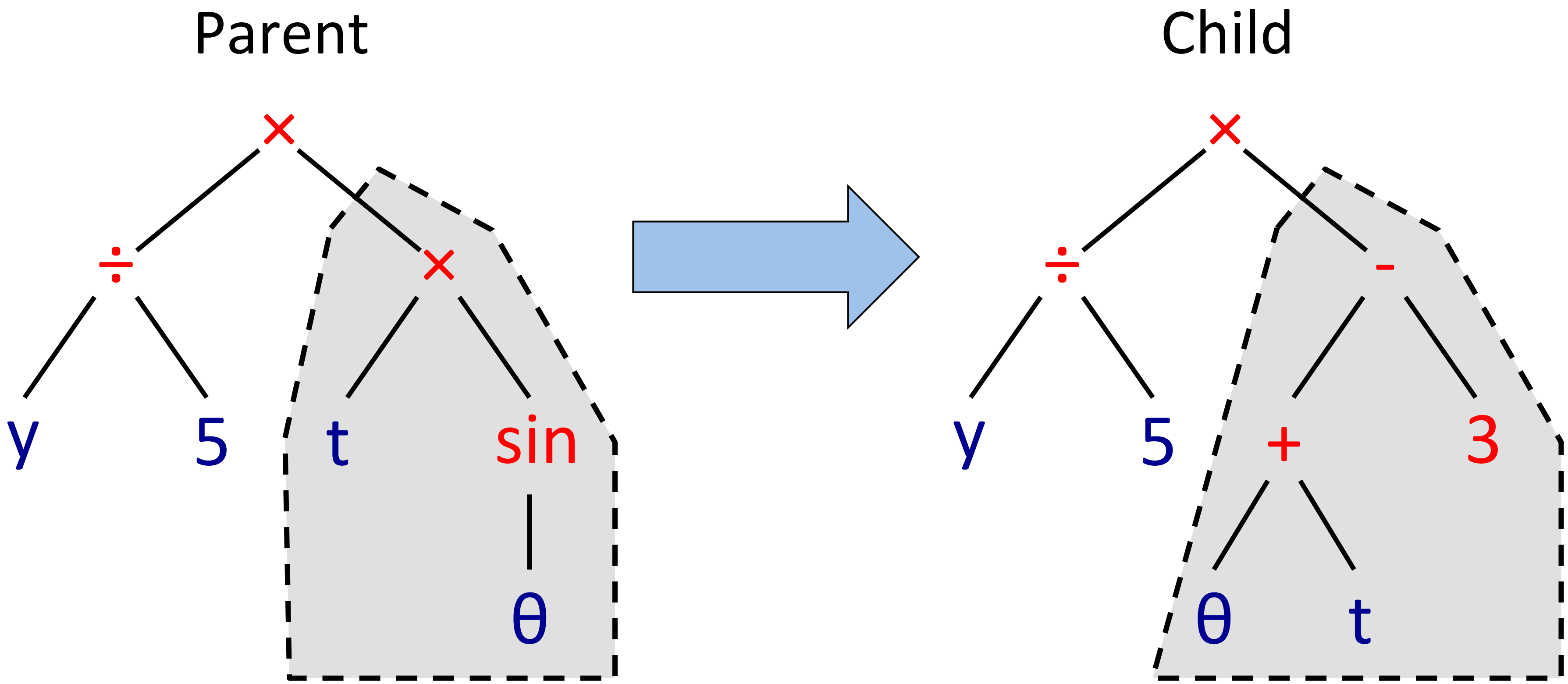
Mutation

- Sub-tree mutation:



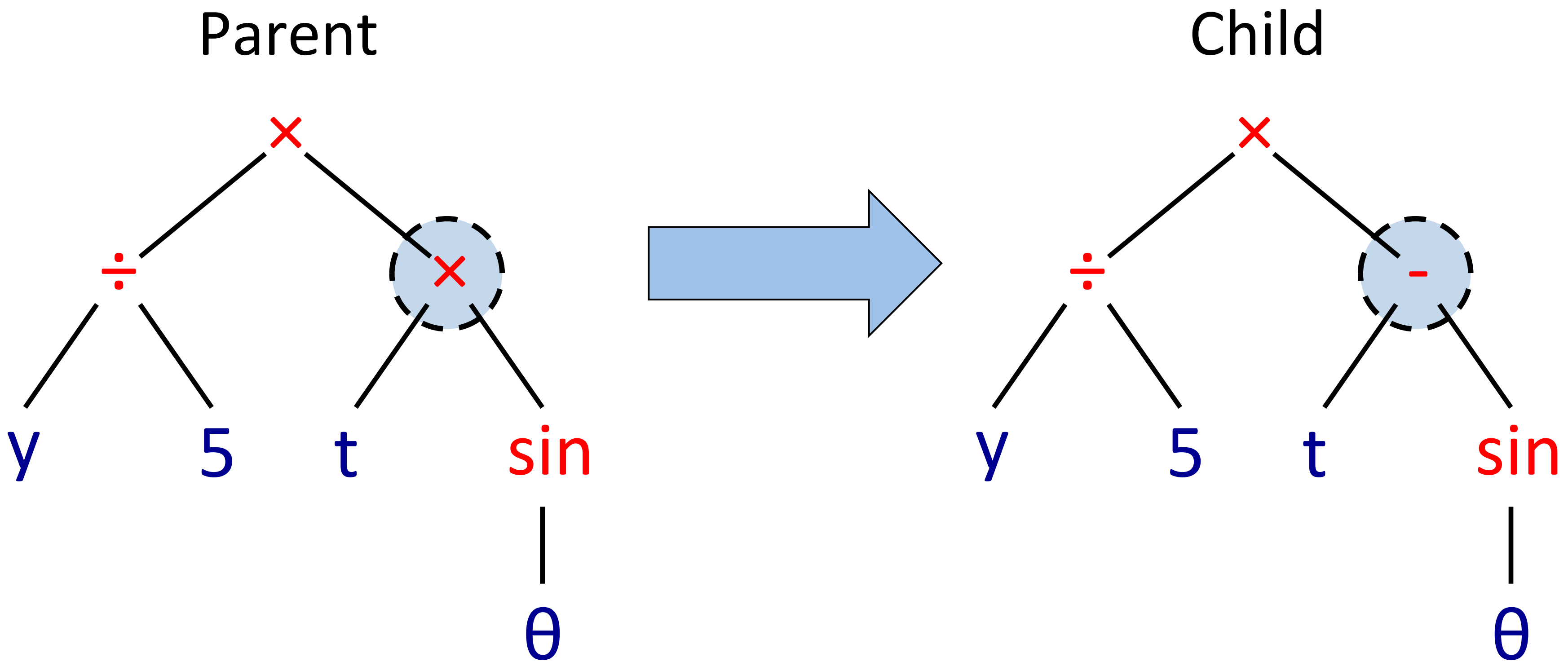
Mutation

- Sub-tree mutation:



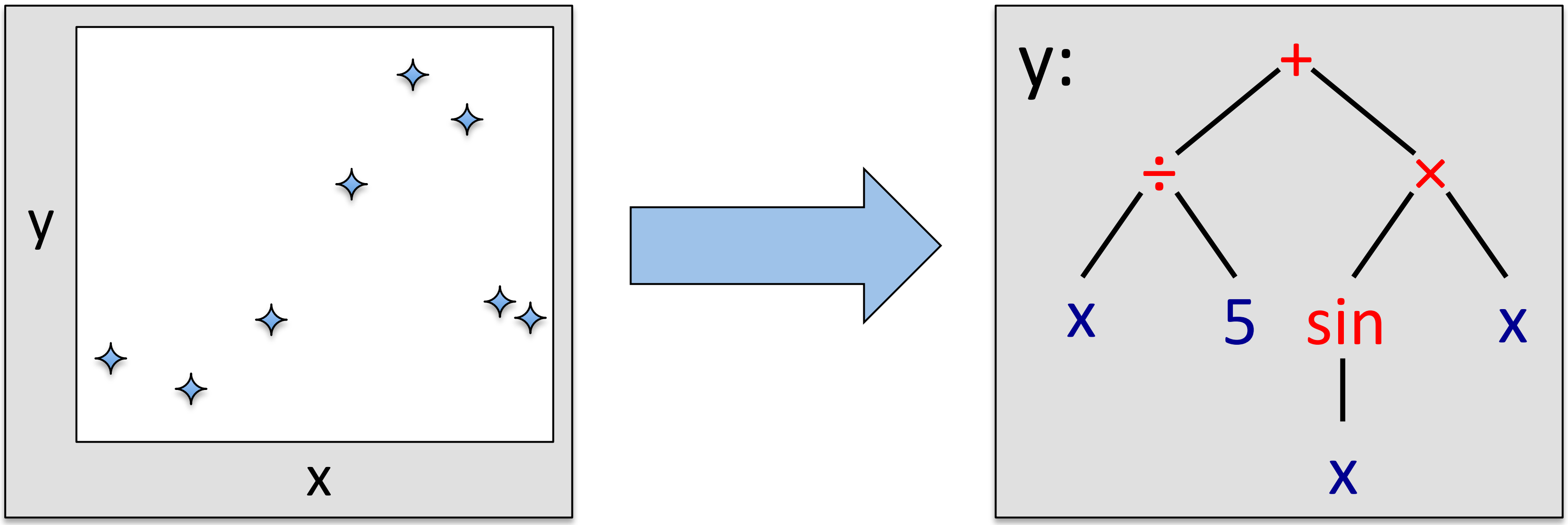
Mutation

- Point mutation (less disruptive):

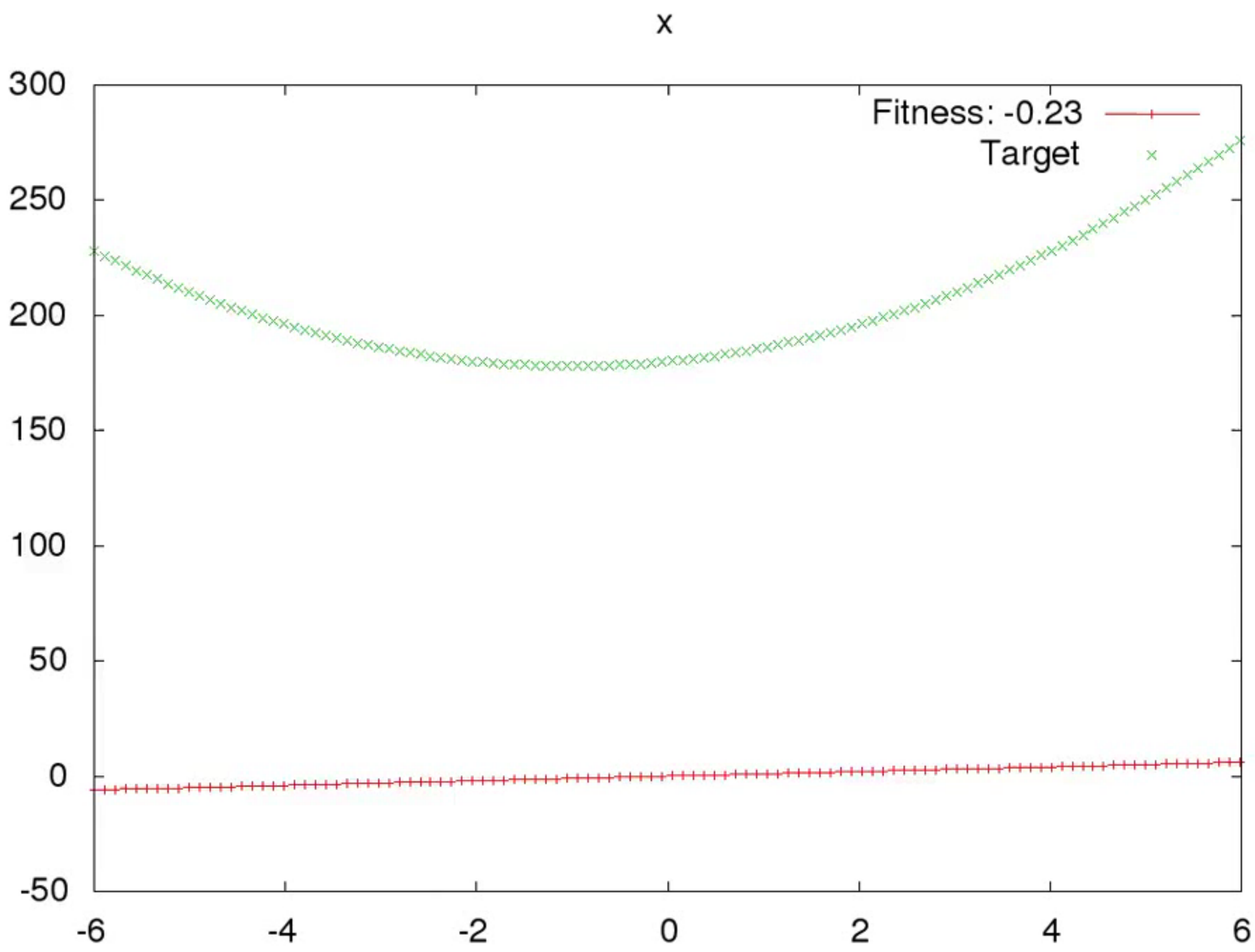


Symbolic Regression

- Fitting a mathematical expression to data
  - A common use of genetic programming
  - Useful when little is known about the generating function



Curve Fitting Example



https://www.youtube.com/watch?v=37D3QpFvrgs

Symbolic Regression

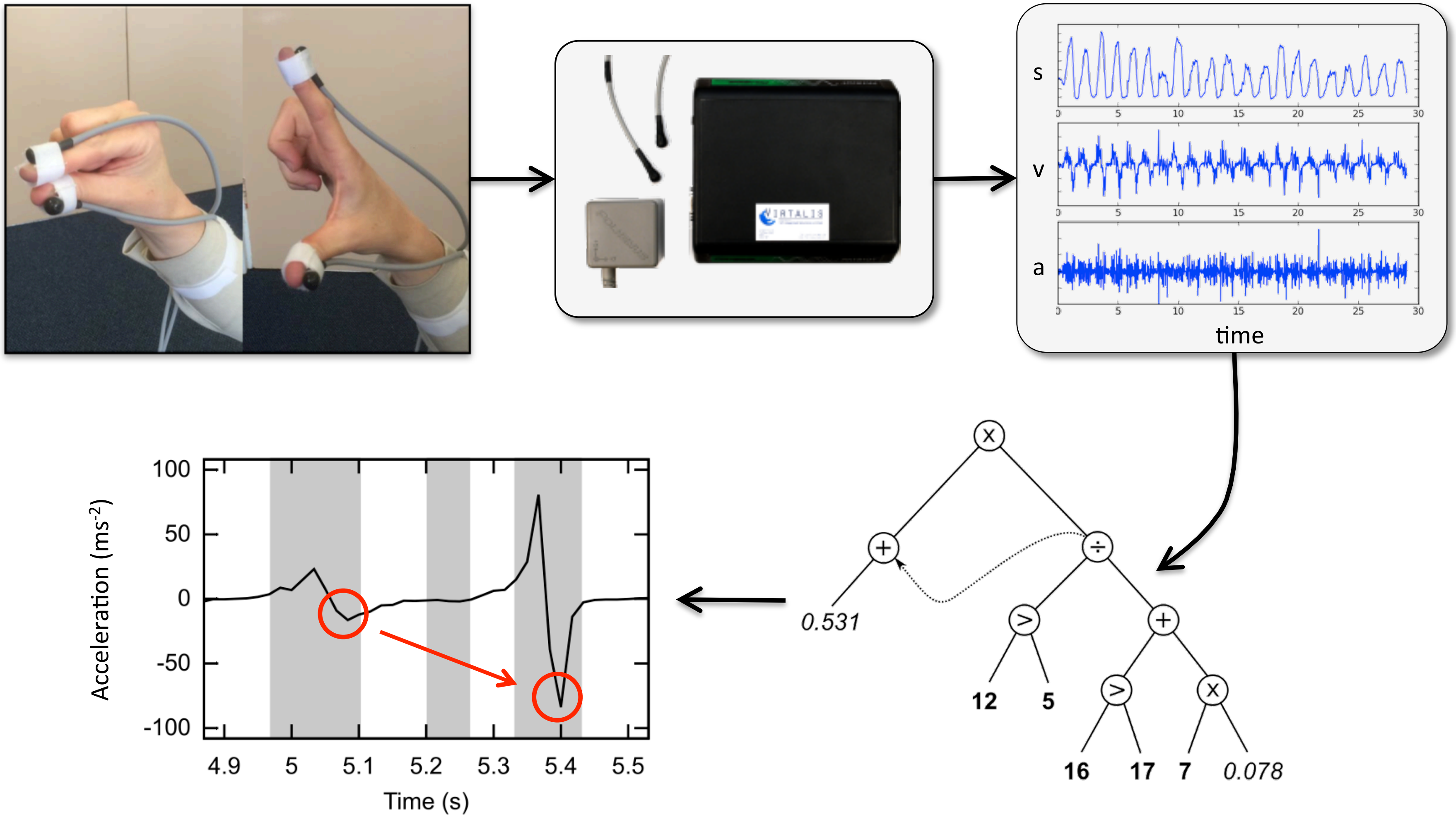
- Regression using ECJ
  - Target expression is the quintic polynomial $x^4+x^3+x^2+x$

```
▹ java ec.Evolve -from app/regression/erc.params
▹ Generation: 1
  Fitness: Adjusted=0.25664273 Hits=1
  Tree 0:
   (- (* (* x x) (+ (cos -0.315)
       (- x -0.870))) (* (rlog -0.707)
       (+ x x)))
  ...
  Generation: 10
  Fitness: Adjusted=1.0 Hits=20
  Tree 0:
   (+ x (* (+ (* (+ x (* x x)) x) x) x))
```

"erc" = ephemeral random constant, i.e. expressions can contain random numbers

Note the prefix notation commonly used by GP systems

Real World Example



- **MA Lones, SL Smith, J Alty, S Lacy, K Possin, S Jamieson, AM Tyrrell**, Evolving Classifiers to Recognise the Movement Characteristics of Parkinson's Disease Patients, *IEEE Trans. Evolutionary Computation,* 2014.

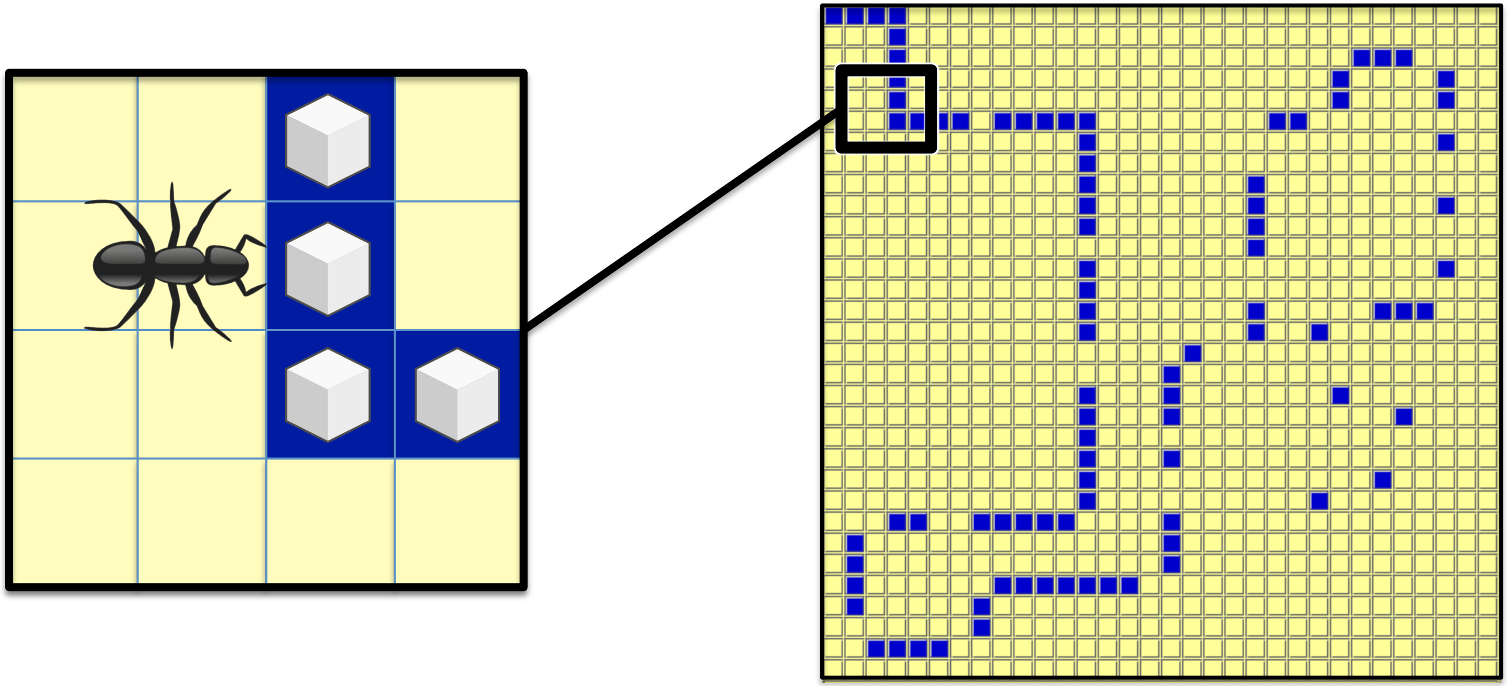
Programmatic Expressions

- Symbolic regression is a popular application of GP
  - But mathematical expressions aren't programs
  - Or, at least, not very exciting programs!

- Programmatic expressions also typically have:
  - Command sequences: `command; command; …`
  - Conditional execution: `if … then … else`
  - Iteration: `for …, do … while`
  - Memory, variables: `int i = 0 …`
  - Functions, modules: `foo = bar(x, y)`

Programmatic Expressions

- Santa Fe Trail Problem
  - A control problem commonly used to benchmark GP
  - Guide an 'ant' to 'eat' all the 'food' in minimum time

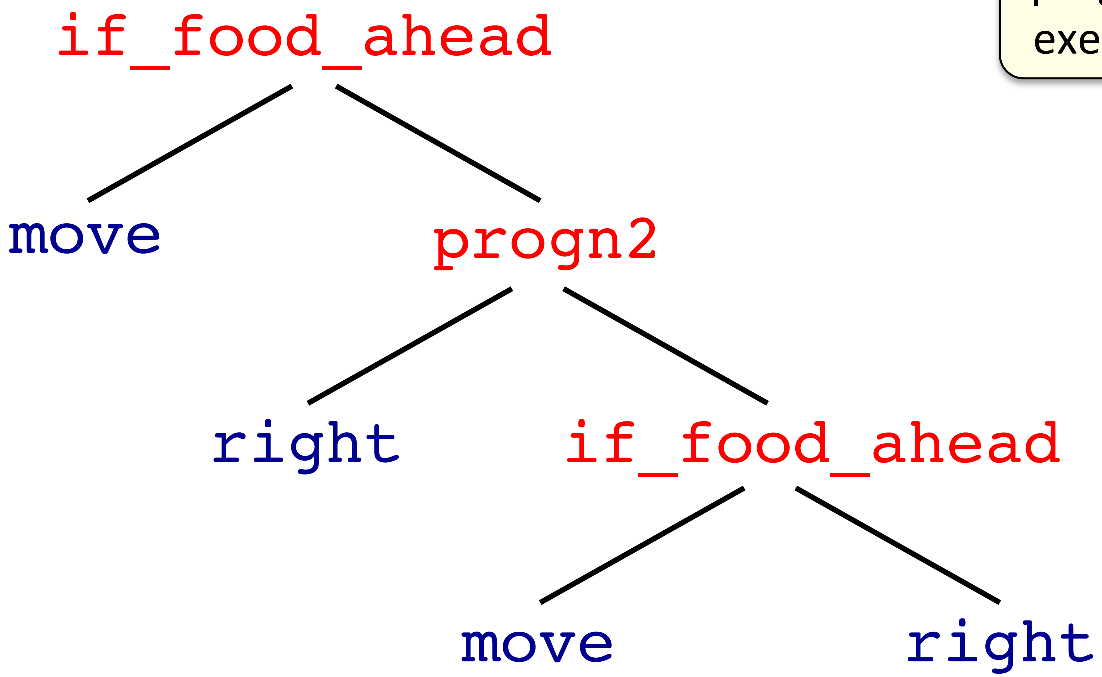


http://http://en.wikipedia.org/wiki/Santa_Fe_Trail_problem

Programmatic Expressions

- Function and terminal sets
  - Functions: { `if-food-ahead`, `progn2`, `progn3` }
  - Terminals: { `left`, `right`, `move` }

progn* are sequential execution statements



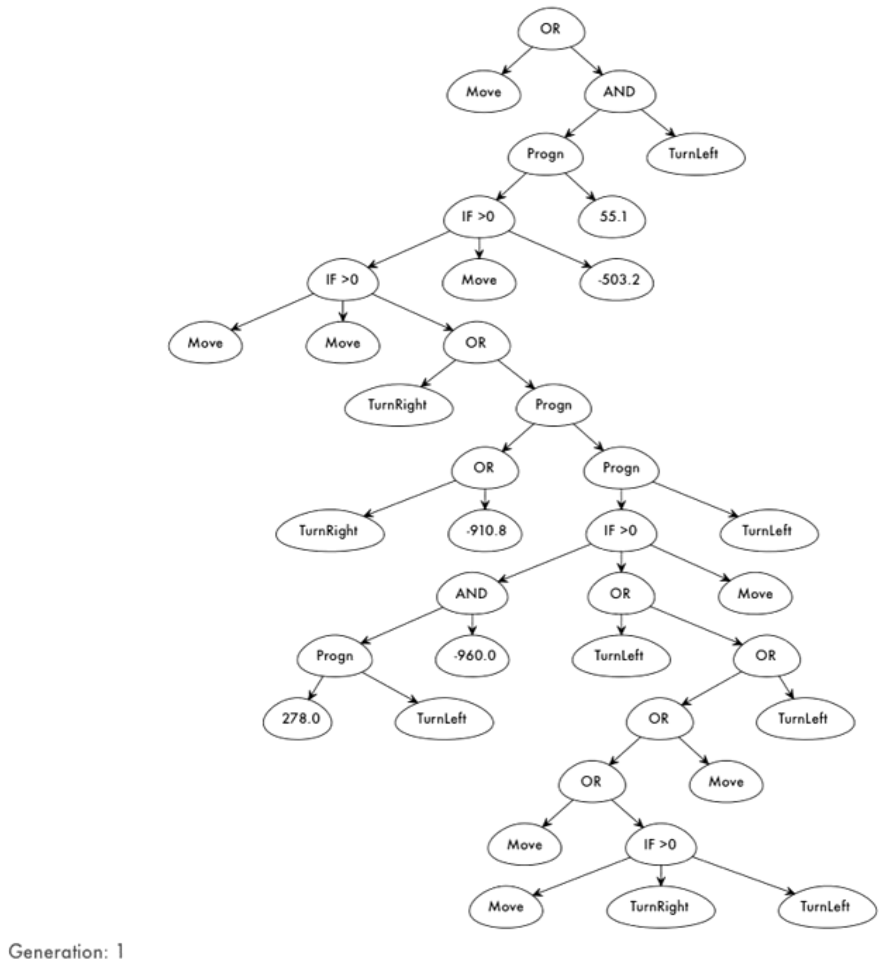
Programmatic Expressions

- Santa Fe Trail using ECJ

```
▹ java ec.Evolve -from app/ant/ant.params
▹ Generation: 50
  Fitness: Standardized=2 Hits=87
  Tree 0:
  (if-food-ahead (if-food-ahead (progn2 (if-food-ahead
          move left) move) left) (progn3 (if-food-ahead
          move left) (if-food-ahead (if-food-ahead
          (progn3 (if-food-ahead move (if-food-ahead
              left left)) (if-food-ahead move (progn3 left
              (if-food-ahead move left) (progn2 (progn3
              left move move) move)) (if-food-ahead (if-
              move move) (progn3 left move move))))
              move right)) (if-food-ahead move (progn3
  ...
```

This one ‘ate’ 87/89 pieces of ‘food’ – pretty good!

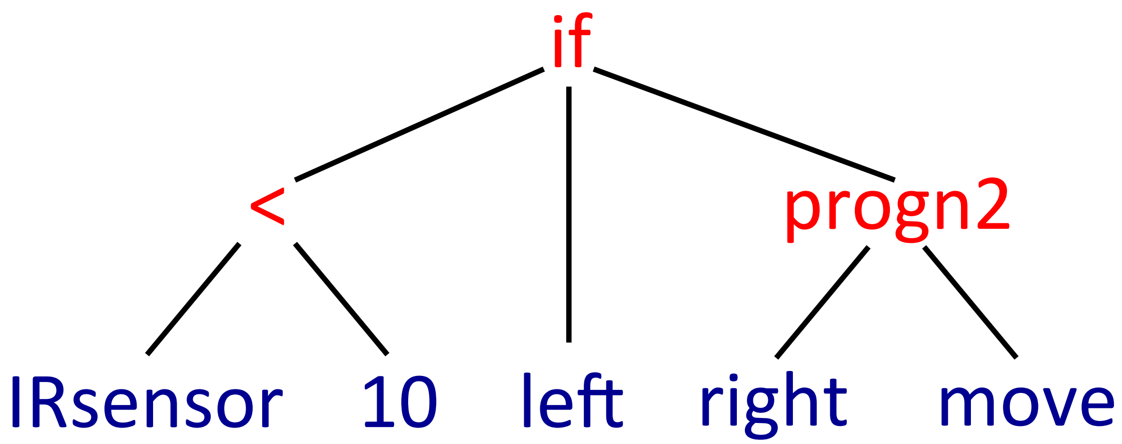
Santa Fe Solution Evolution



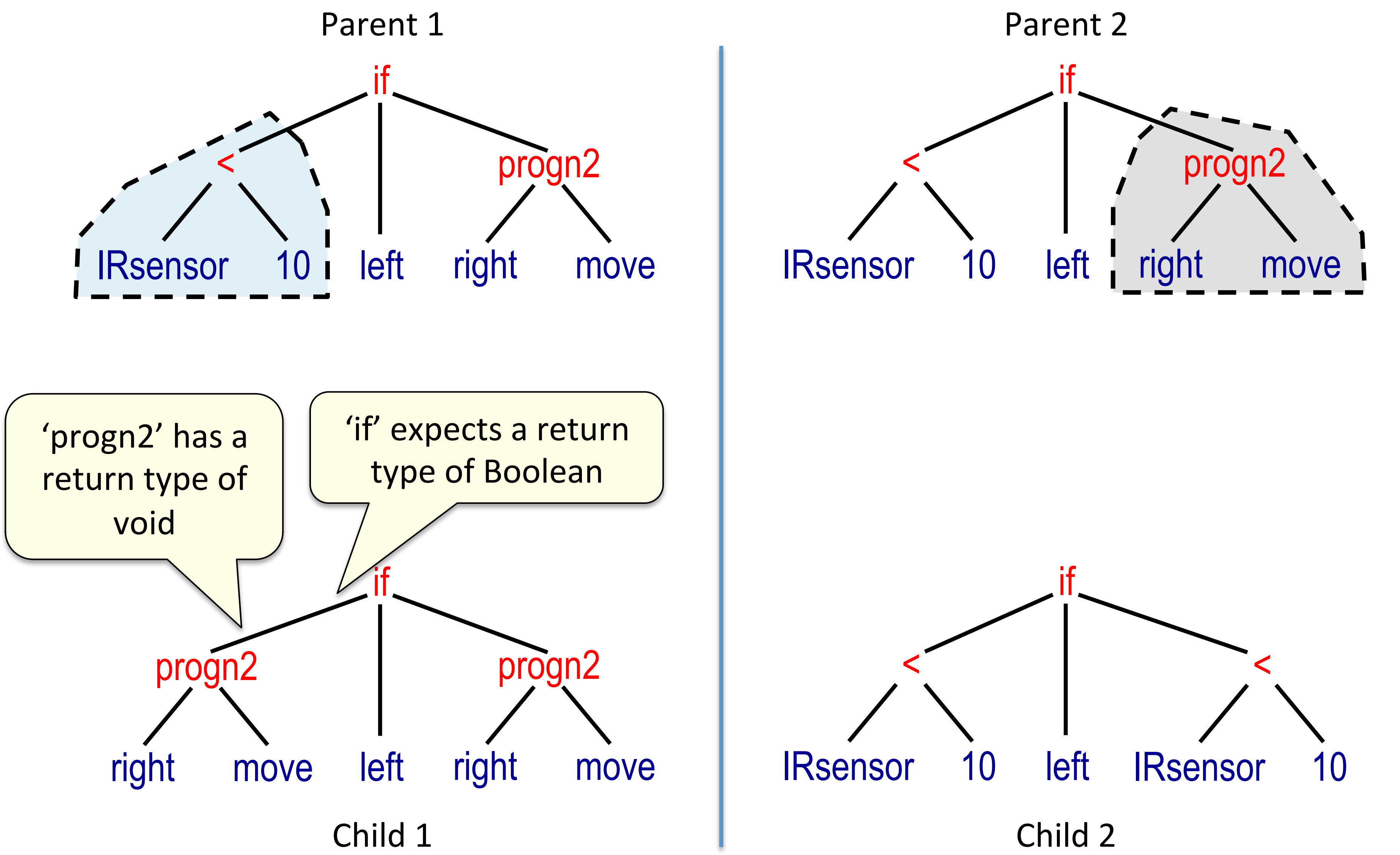
https://www.youtube.com/watch?v=6cMXN5rGLCs

Conditional Execution

- An ‘if’ command for every condition?
  - `if_x_equals_1, if_x_is_greater_than_2 …`
  - Not a very flexible or effective approach
- We would prefer something like this:



Crossover Problem



Closure

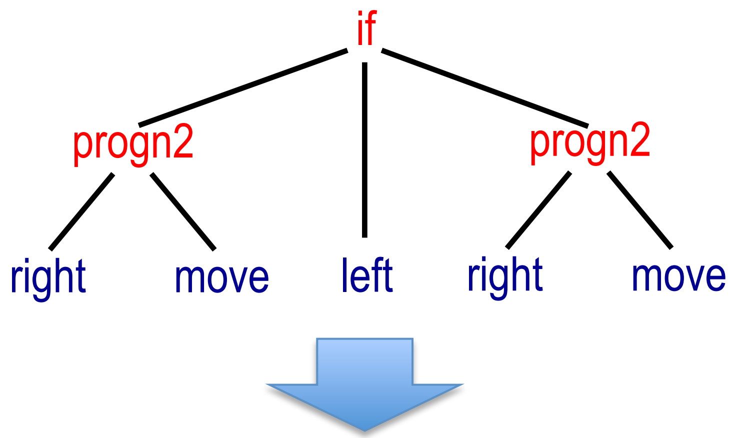
- Traditional tree-based GP requires **closure**
  - All functions must be able to do something with whatever input they may receive
  - i.e. their input types must be more general than any other function or terminal's output type

- Function set with closure – good ☺
  - { AND, OR, NAND, NOR, NOT }

- Function set without closure – bad ☹
  - { +, -, AND, OR, progn2, sin, cos }

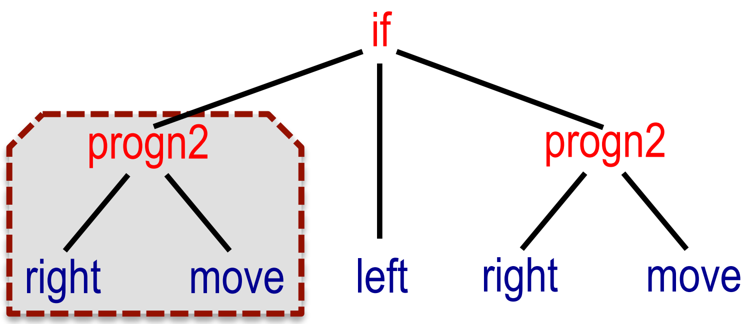
Can we avoid closure?

- Penalise invalid solutions
  - A common approach in EAs
  - Easy to implement
  - Can lead to search space bias
  - Inefficient use of population if invalidity occurs often



Fitness = 0 ☹

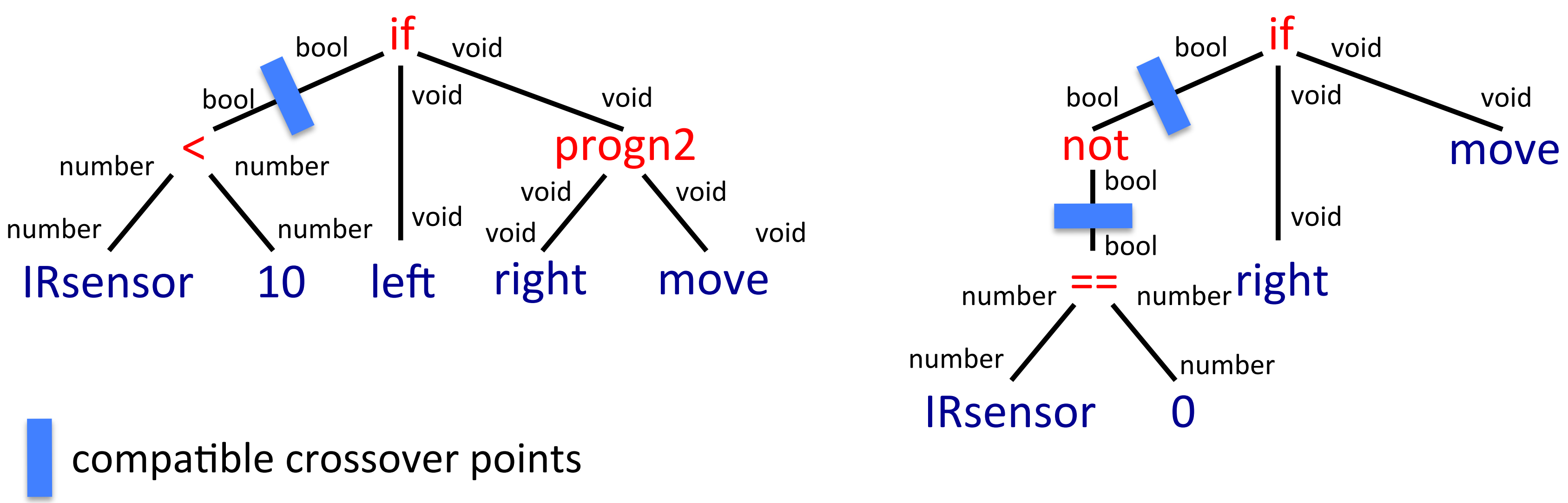
- Repair invalid solutions
  - Another common EA approach
  - Maintains population efficiency
  - Can be time consuming



Mutate sub-tree until valid

Type-Constrained Operators

- Constrain initialisation and variation operators
  - By taking into account the return types of branches
  - e.g. only allow crossover points at type-compatible points
  - The preferred approach to handling mixed types in GP

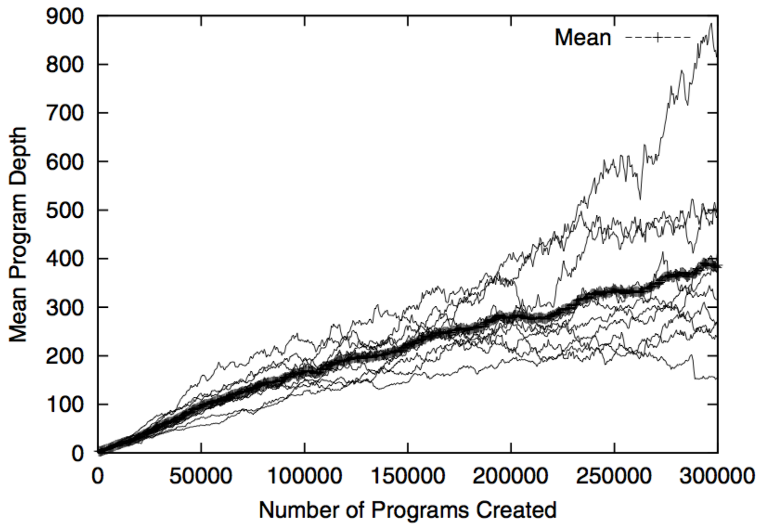


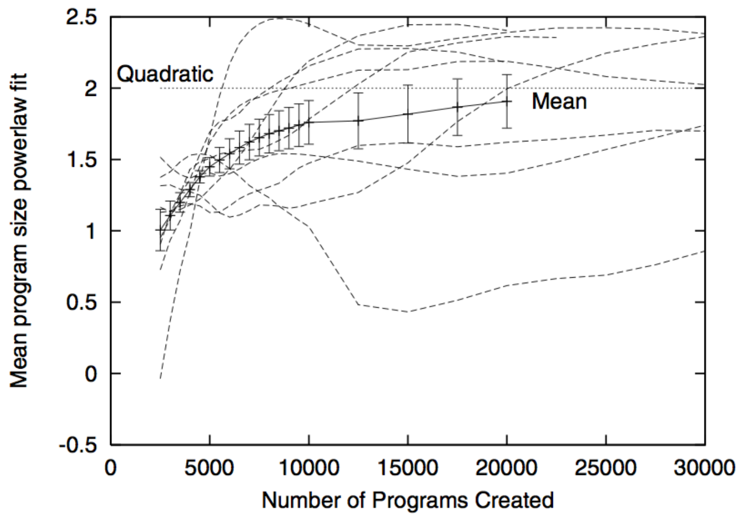
Strongly-Typed GP

- A variant of GP [Montana, 1995]
  - Builds upon the idea of type constraints
  - Every terminal and function is assigned a type
  - Provides scope for type hierarchies
  - Also supports generic functions with flexible types
  - Paper discusses mixing scalars, vectors and matrices:
    - http://davidmontana.net/papers/stgp.pdf

Bloat

- Bloat is a big problem for genetic programming
  - Tendency for trees to grow large during evolution
  - In standard GP, growth has quadratic complexity
  - Leads to inefficient uninterpretable programs

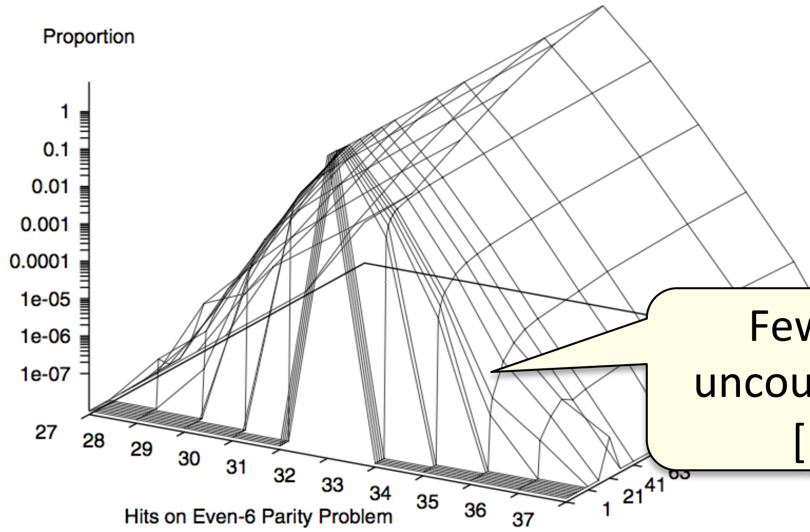




From Langdon, 2000, Quadratic Bloat in Genetic Programming

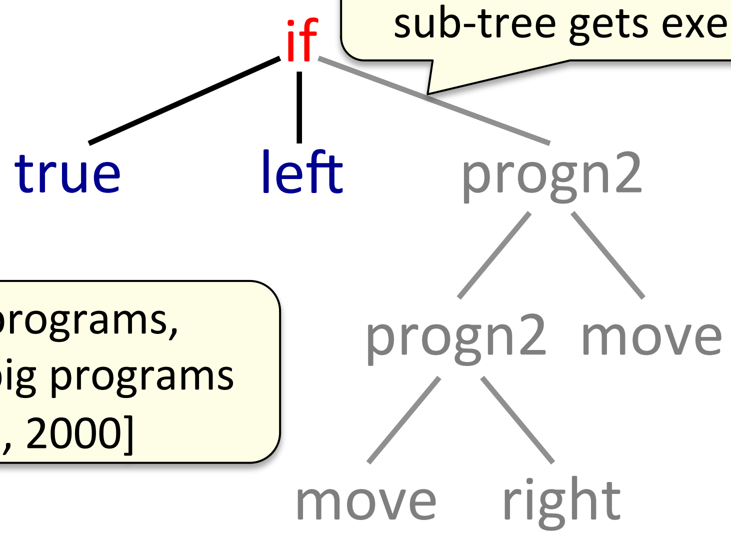
Bloat

- Many theories for why bloat occurs:
  - There are more big programs than small programs
  - GP operators tend to explore larger trees (operator bias)
  - Programs protect themselves with non-functional code
  - [See §11.3 of "Field Guide"]



Few small programs, uncountable big programs [Langdon, 2000]

Nothing in this 'intron' sub-tree gets executed



Bloat

- There are various ways to control bloat
  - Easiest way is to apply **depth constraints**
    e.g. only pick crossover points below depth *N*
  - **Parsimony pressure** involves penalising large programs
    e.g. subtract a term from their fitness in proportion to size
  - **Code editing** involves removing parts of large programs
    e.g. remove the bits that don't do anything
  - An **extra objective** can be added to a multiobjective EA
    e.g. second objective of minimising number of nodes
- For more info, read [Luke, 2006]
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.159.1580&rep=rep1&type=pdf

Things you should know

- What GP is and when you should use it
- Basics of tree-based GP:
  - Sub-tree crossover and mutation operators
  - Closure, why types can be a problem
  - Bloat: why it is a problem, methods for avoiding it
- I don't expect you to know:
  - Details of initialisation methods
  - About the causes of bloat
  - Methods for handling types

Questions

- Where can I see some code?
  - "Essentials of Metaheuristics" sec. 3.3.3 & 4.3 (or ECJ)

Algorithm 55 *The Ramped Half-and-Half Algorithm*

```
minMax ← minimum allowed maximum depth
maxMax ← maximum allowed maximum depth
FunctionSet ← function set

d ← random integer chosen uniformly from minMax to maxMax inclusive
if 0.5 < a random real value chosen uniformly from 0.0 to 1.0 then
    return DoGrow(1, d, FunctionSet)
else
    return DoFull(1, d, FunctionSet)
```

Coursework 3

- Available now!
  - Download the zip file containing ECJ and CW3 files
- It's about getting to know genetic programming
  - Trying out GP on some benchmark problems
  - Understanding how parameters affect GP's behaviour
  - Getting some experience using a well known evolutionary computing framework
- It's not about your Unix skills
  - So let me know if you're struggling with this aspect

Other things to do

- Get to know ECJ:
  - Install it: http://cs.gmu.edu/~eclab/projects/ecj/
  - Read the tutorials, browse the documentation
  - Play around with it
- Get to know GP:
  - Check out the GP facilities in ECJ
  - Have a look at the example problems
  - Play around with parameter files

Bibliography

HERIOT WATT UNIVERSITY

- S. Luke and L. Panait, A Comparison of Bloat Control Methods for Genetic Programming, *Evolutionary Computation* 14(3):309-344, 2006 http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.159.1580
- D. Montana, Strongly Typed Genetic Programming, *Evolutionary Computation* 3(2):199-230, 1995. http://www.cs.bham.ac.uk/~wbl/biblio/cache/http___vishnu.bbn.com_papers_stgp.pdf