HERIOT WATT UNIVERSITY

Genetic Programming: Memory, Loops and Modules

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

Distinctly Global
www.hw.ac.uk

Previous Lecture

- What GP is and when you should use it
  - Evolutionary algorithms that optimise programs
  - Useful for discovering novel solutions to problems
  - Or solutions to problems you don't know how to solve

- Introduction to tree-based GP:
  - Initialisation, crossover, mutation
  - Symbolic regression
  - Conditional execution
  - Handling types
  - Bloat and bloat avoidance

Homework

- 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
- Be ready to discuss what you've learnt next week…

Discussion

- ECJ
  - What do you think of it?
  - Any problems?
  - Does the class structure make sense?

- Genetic Programming
  - Did you discover anything interesting?
  - Did you find anything unexpected?
  - Did you think of any good applications?

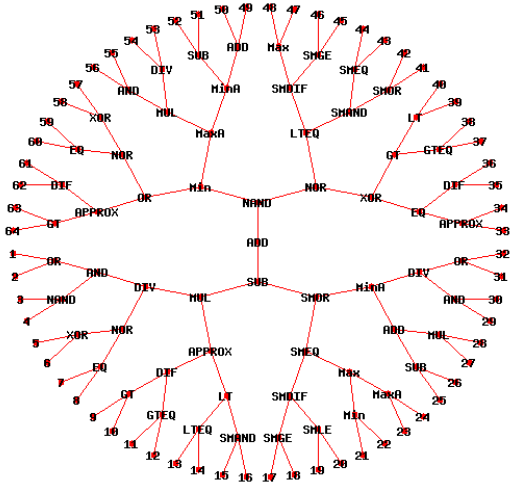
ECJ

HERIOT WATT UNIVERSITY

- A 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**

Questions

- Can GP expressions be “pretty printed”?
  - Yes!
  - For output options (latex etc.) in ECJ, see: http://cs.gmu.edu/~eclab/projects/ecj/docs/tutorials/tutorial4/index.html
  - There’s also a gawk script for circular trees: http://www0.cs.ucl.ac.uk/staff/W.Langdon/gp2lattice/gp2lattice.html



Questions

- Can GP be used for medical image analysis?
  - Yes, there are quite a few examples of this
  - Google “genetic programming medical image”

Medical Example

HERIOT WATT UNIVERSITY

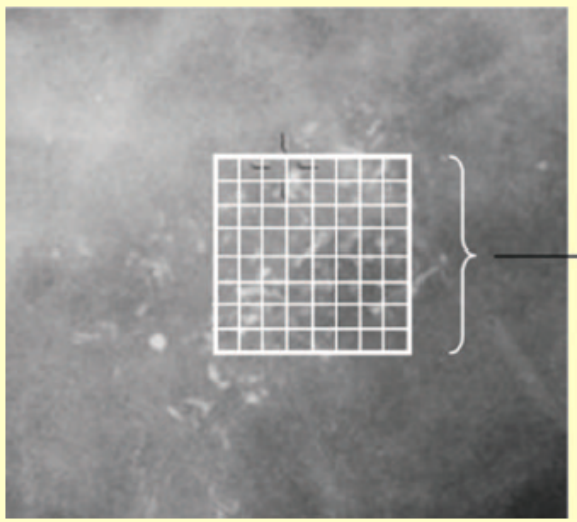
Cartesian Genetic Programming and Its Application to Medical Diagnosis

I. Introducti

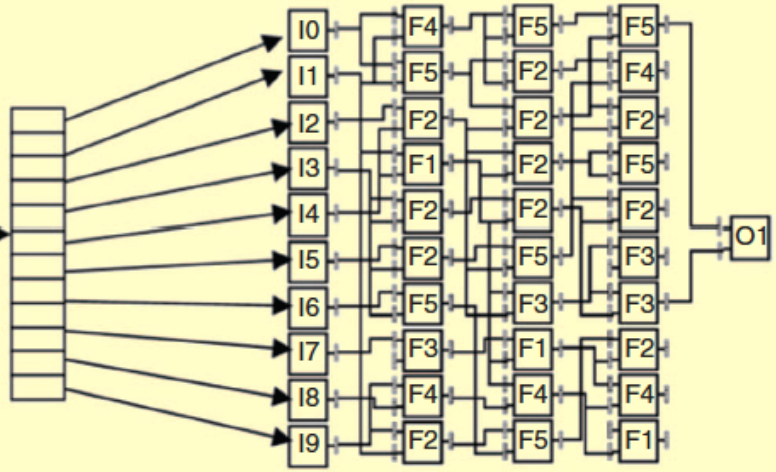
Comp
tions
relev
lific
and biometri

Digital Object Identifi
Date of publication:

56 IEEE COMPUTATIONAL IN



Region of Interest in Mammogram



SGLD Matrix

CGP Network

- **SL Smith,** Cartesian Genetic Programming and Its Application to Medical Diagnosis, *IEEE Computational Intelligence Magazine,* November 2011.

Medical Example

HERIOT WATT UNIVERSITY

- http://dl.acm.org/citation.cfm?id=2598244

Improving 3D Medical Image Registration CUDA Software with Genetic Programming

William B. Langdon, Marc Modat, Justyna Petke, Mark Harman
Dept. of Computer Science, University College London Gower Street, WC1E 6BT, UK
W.Langdon@cs.ucl.ac.uk

ABSTRACT

Genetic Improvement (GI) is shown to optimise, in some cases by more than 35%, a critical component of healthcare industry software across a diverse range of six nVidia graphics processing units (GPUs). GP and other search based software engineering techniques can automatically optimise the current rate limiting CUDA parallel function in the Nifty Reg open source C++ project used to align or register high resolution nuclear magnetic resonance NMRI and other diagnostic NIfTI images. Future Neurosurgery techniques will require hardware acceleration, such as GPGPU, to enable real time comparison of three dimensional in theatre images with earlier patient images and reference data. With millimetre resolution brain scan measurements comprising more than ten million voxels the modified kernel can process in excess of 3 billion active voxels per second.

Categories and Subject Descriptor I.2.8 [search]: heuristic

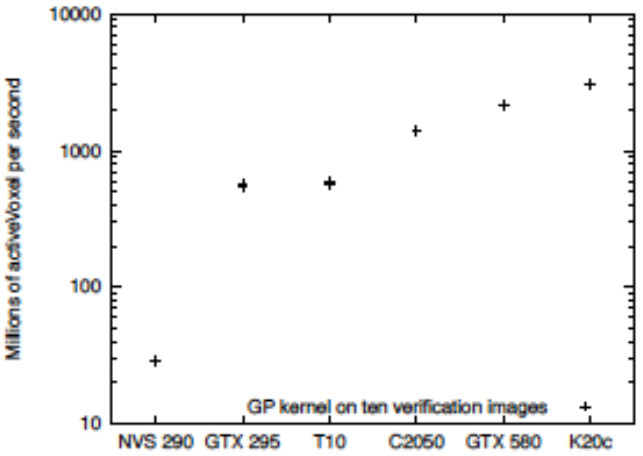


Figure 1: Performance of modified reg_spline_getDeformationField3D CUDA kernel after optimisation

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*

```
1: minMax ← minimum allowed maximum depth
2: maxMax ← maximum allowed maximum depth
3: FunctionSet ← function set

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

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.

Expressiveness

This is the capacity for a program representation to express different kinds of behaviours.

For genetic programming, you can't evolve a program if you can't express it.

In practice, there is often a trade-off between expressiveness and evolvability.

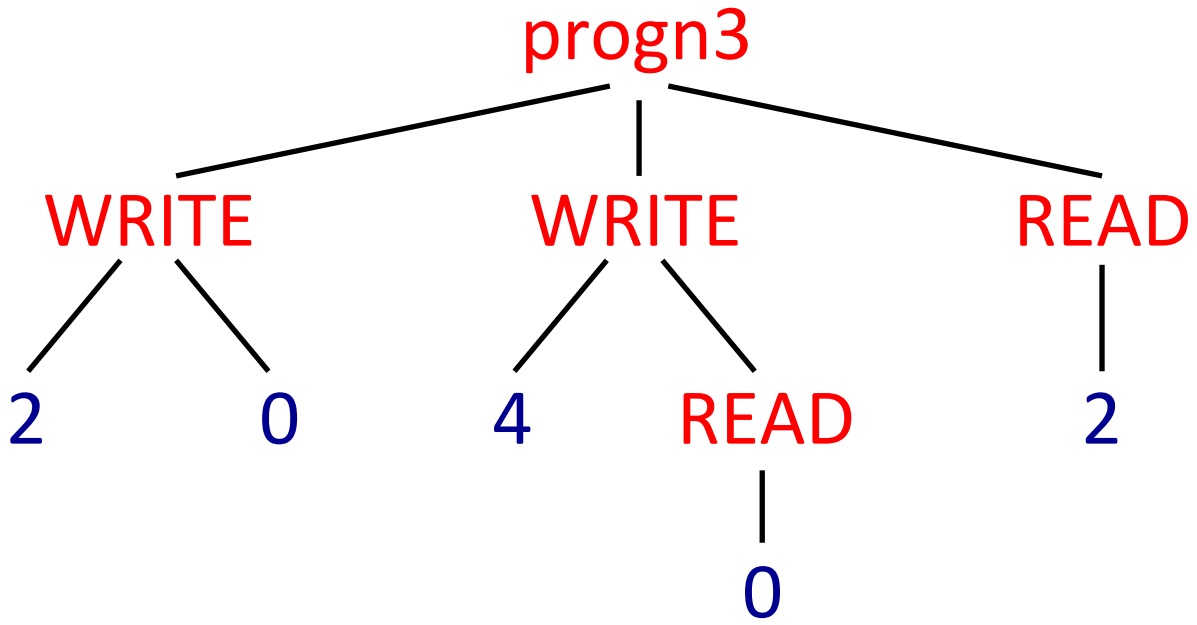
Adding Memory

- There are various ways of adding memory to GP
  - However, in practice these are not widely used

- Consider the approach used by Astro Teller [1994]
  - This introduces a memory, of width M:

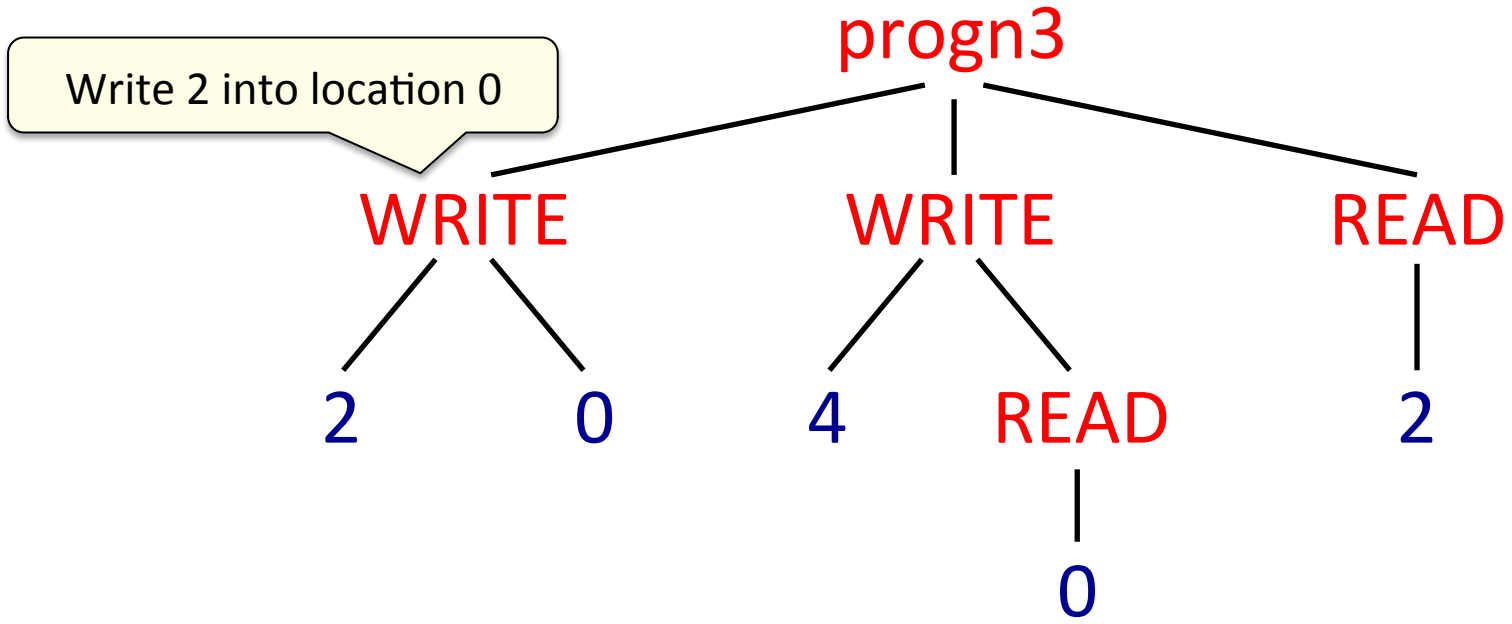
  - And 2 new functions:
    - READ(X) – read value from memory location X
    - WRITE(Y, X) – write value Y to memory location X

Example



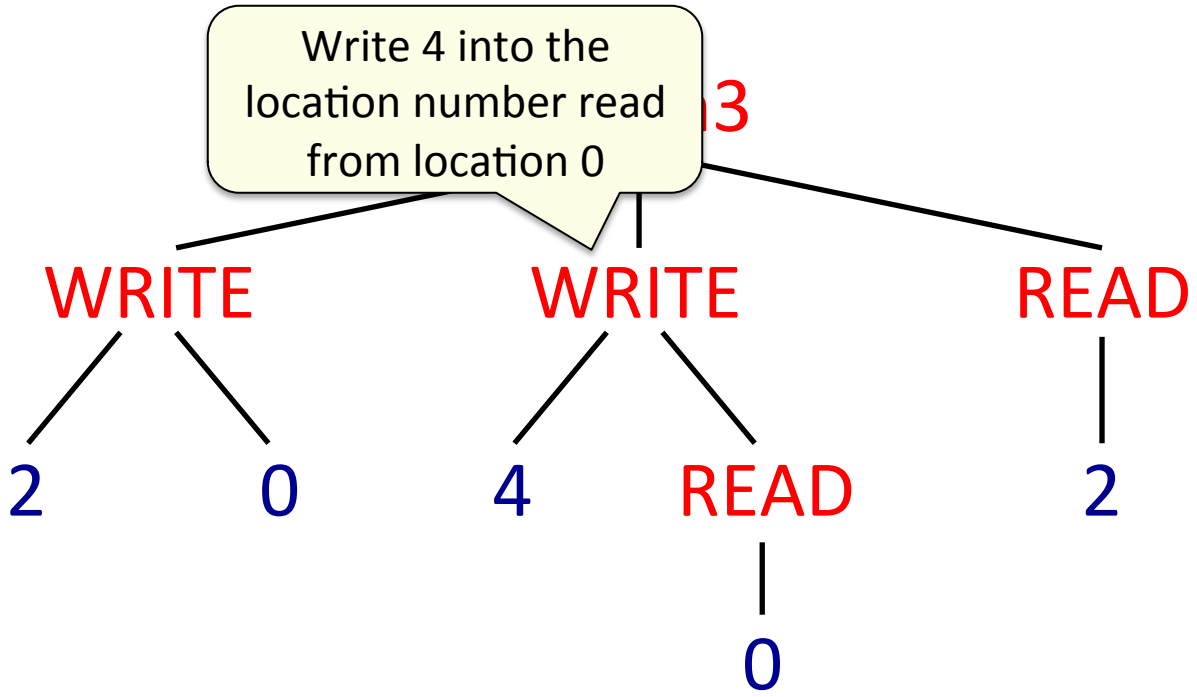
Example


2
0
0
0
0
0
0
0
0
0


Write 2 into location 0
progn3
WRITE
WRITE
READ
2
0
4
READ
2
0

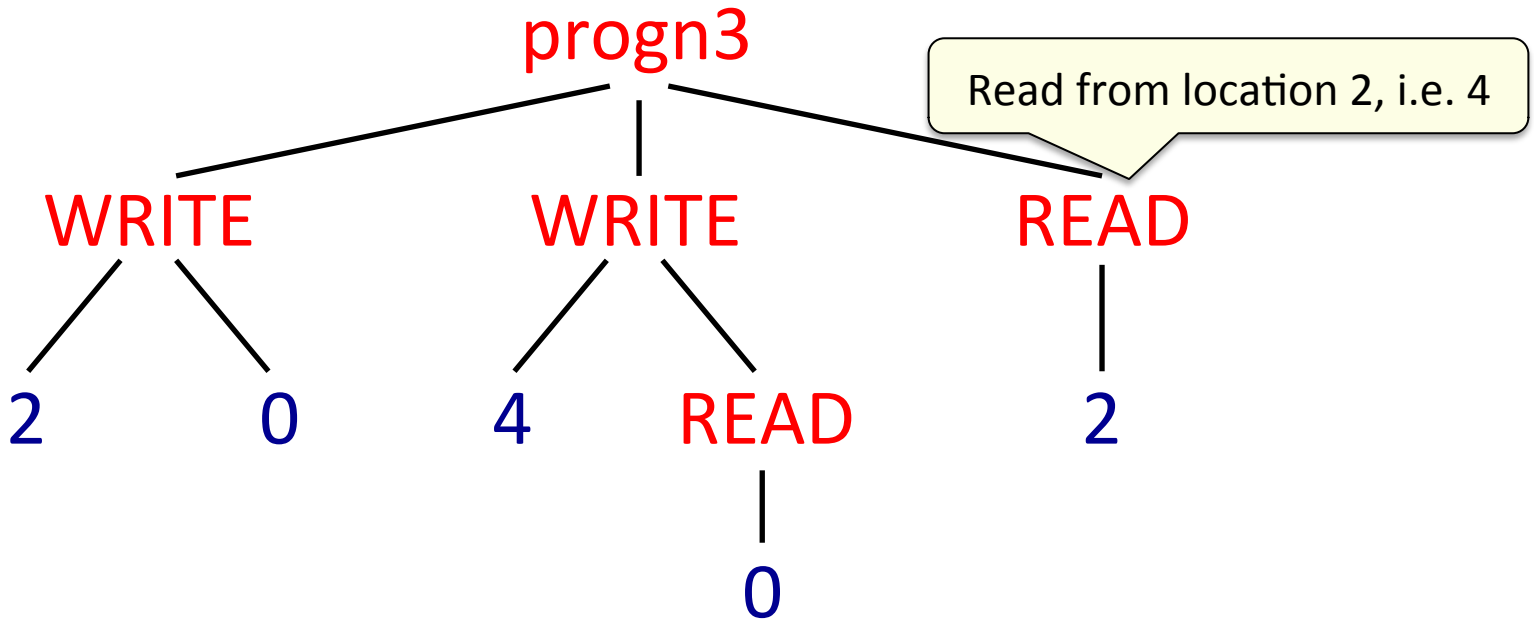
Example


2 0 4 0 0 0 0 0 0 0


Write 4 into the location number read from location 0
WRITE
WRITE
READ
2
0
4
READ
2
0

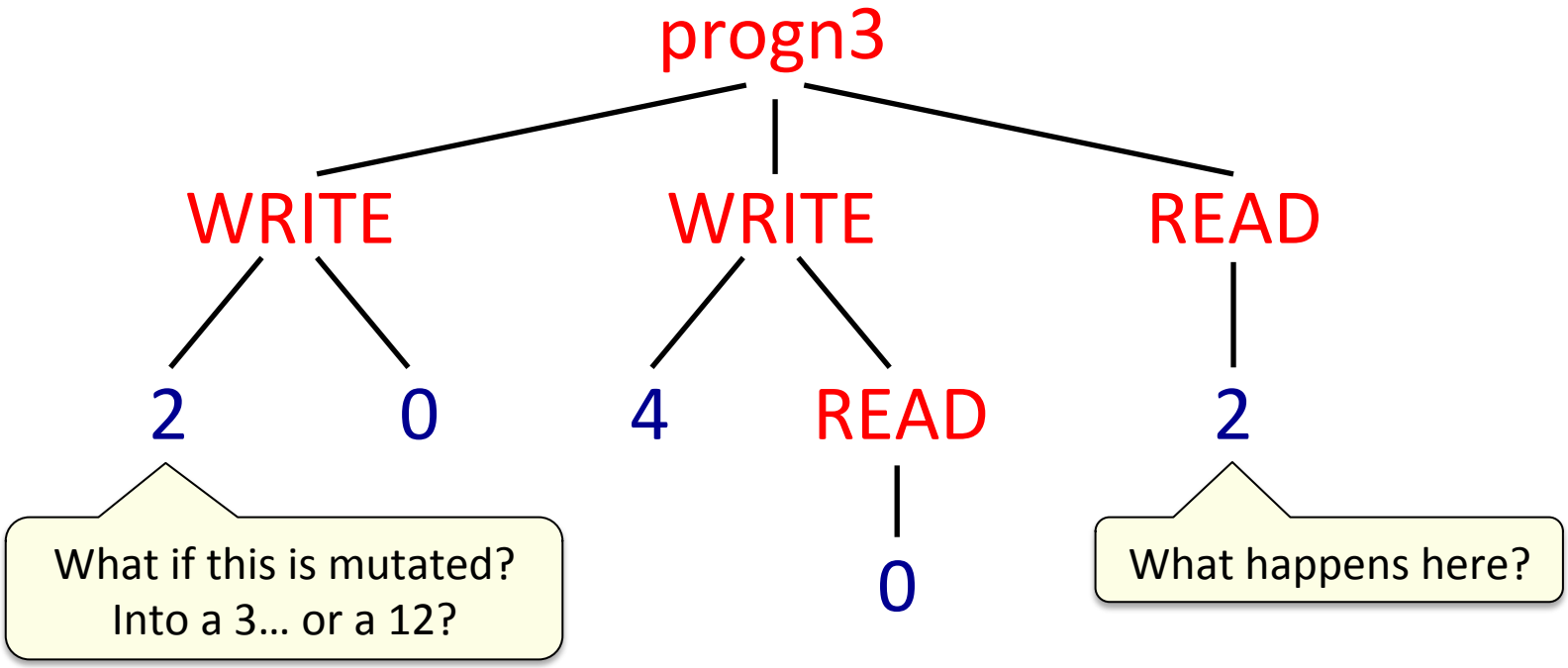
Example


2 0 4 0 0 0 0 0 0 0


progn3
Read from location 2, i.e. 4
WRITE
WRITE
READ
2
0
4
READ
2
0

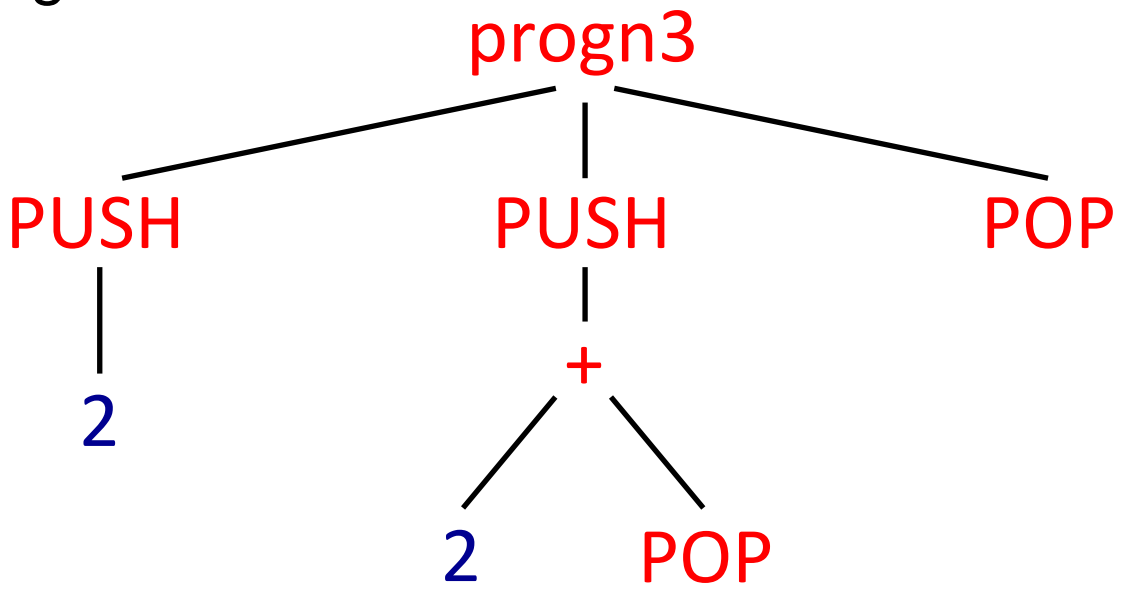
Example


2 0 4 0 0 0 0 0 0 0


progn3
WRITE
WRITE
READ
2
0
4
READ
2
0
What if this is mutated?
Into a 3… or a 12?
What happens here?

Data Structures

- Using addressable memory is problematic
- An alternative is to use data structures
  - These are less expressive than ‘full memory’
  - But less susceptible to bad mutations
  - e.g. using a stack:

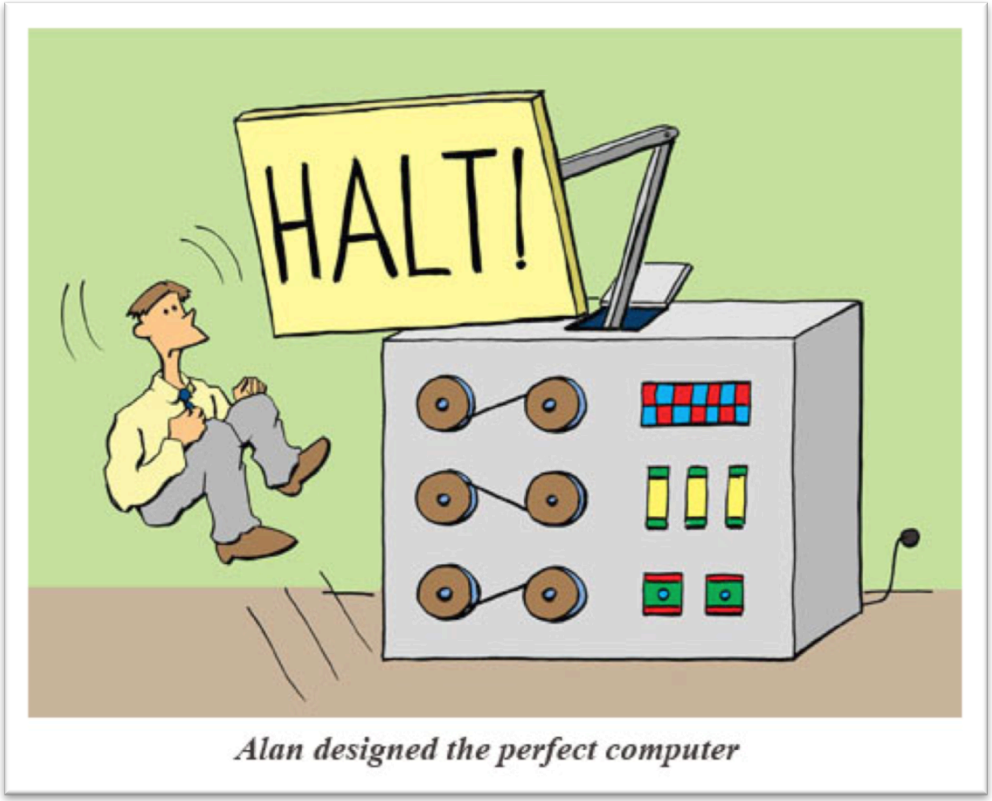


Other Memory Approaches

- "Soft memory" [Poli et al 2009]
  - Blends new assignments with old memories
  - Intended to be more evolvable
  - http://cswww.essex.ac.uk/staff/poli/papers/PoliMcPheeCitiCraneJAEA2009.pdf
- Cultural memories [Spector & Luke 1996]
  - Memory is shared amongst the population
  - Allows programs to communicate
  - http://www.cs.gmu.edu/~sean/papers/culture-gp96.pdf

Adding Loops

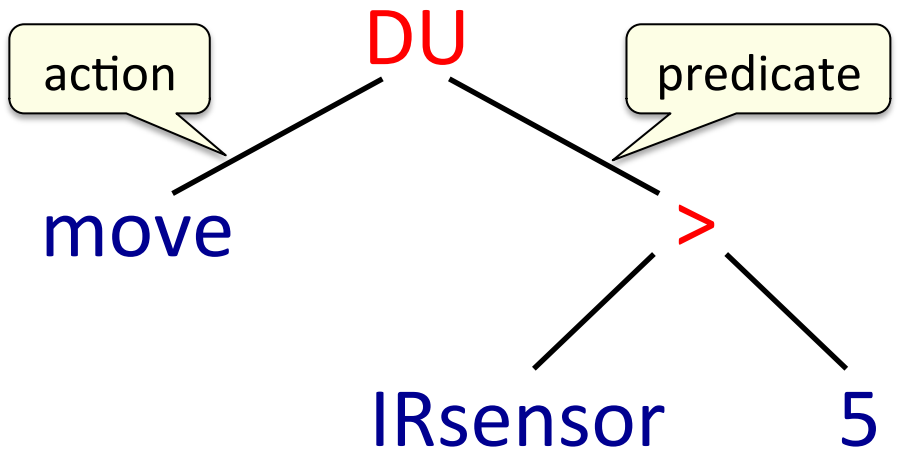
- This is possible, but…
  - Requires caution!
  - Halting Problem: can't predict termination
  - Need to use constraints to prevent infinite loops
  - (e.g. max iterations)
  - Or use a time-out during solution evaluation



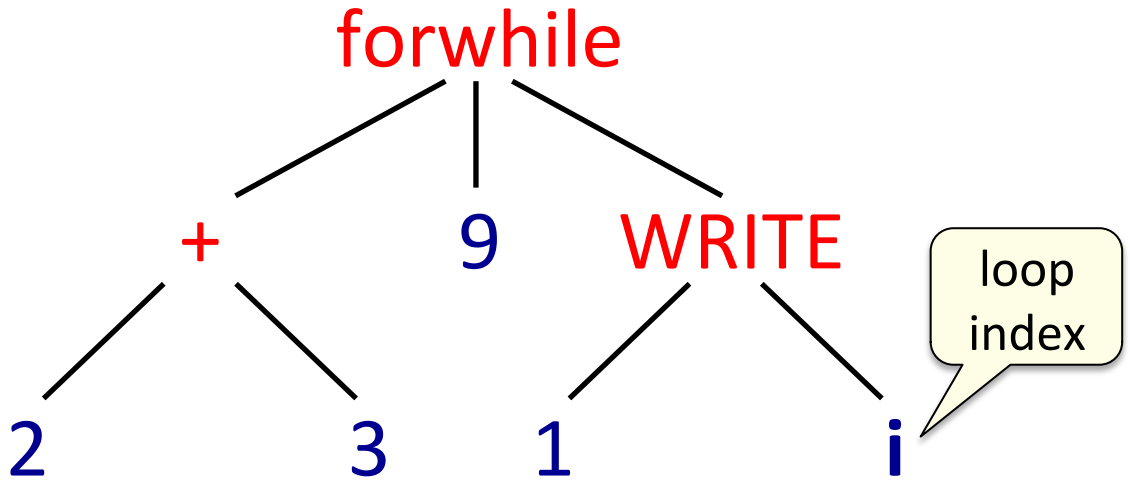
http://www.coopertoons.com

Adding Loops

- Koza-style:
  - do {
    move();
    until(IRsensor>5)



- Langdon [1996]:
  - for(i=2+3; i<9)
    {
    write(1,i);
    }

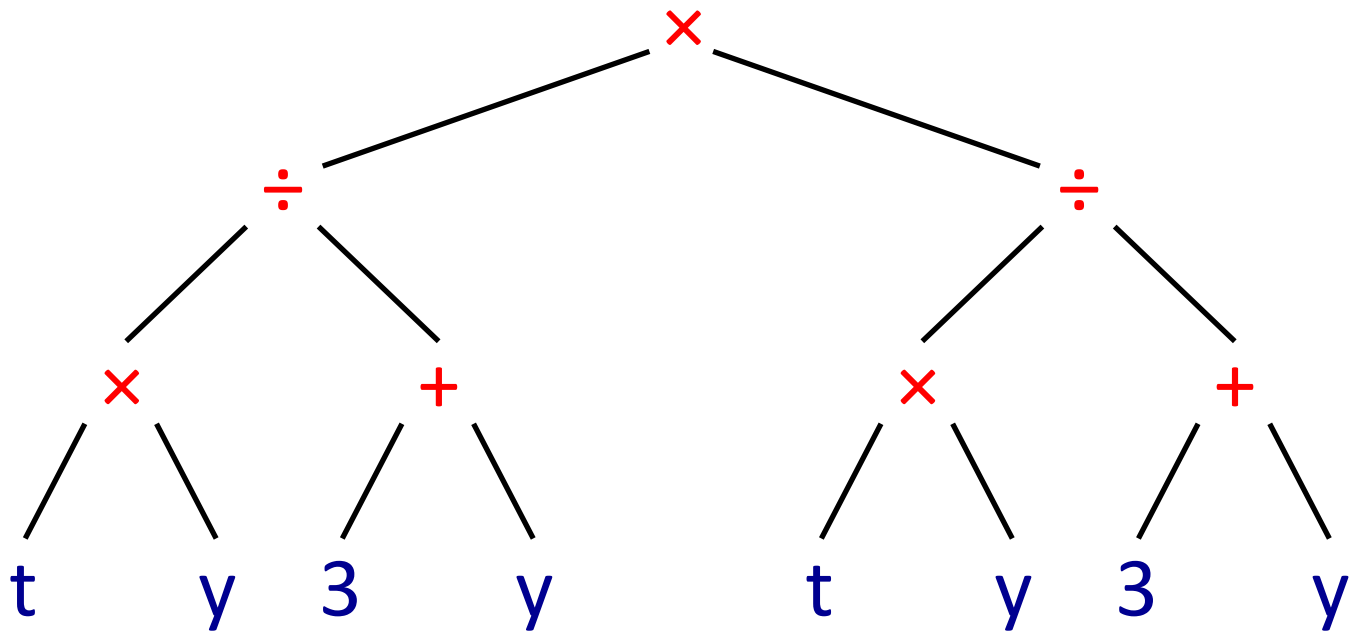


Recap

- It is possible to add syntactic features to GP:
  - Types ✔
  - Memory ✔
  - Loops ✔
- But this has consequences:
  - More complex initialisation and variation operators
  - More constraints during evolution
  - Possible biases within the search landscape
  - **So, only use them when necessary**

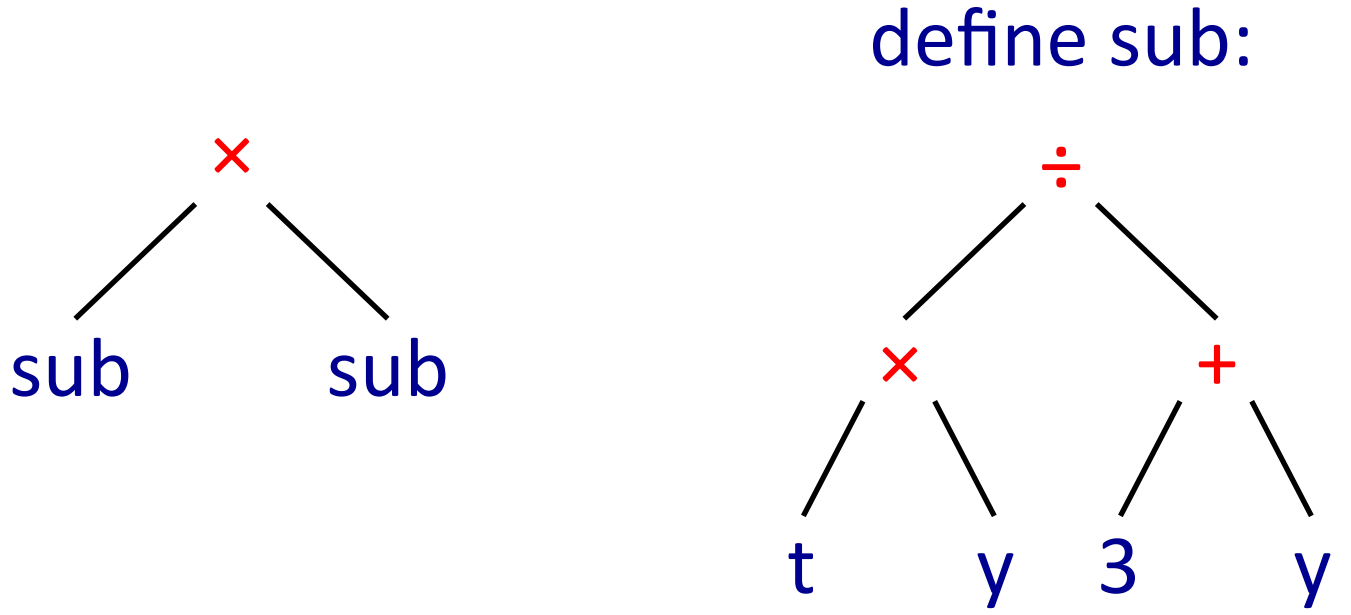
Modularity

- Sub-expressions frequently re-occur in a program
  - Ideally, we only want to evolve each one once
  - So, not this:



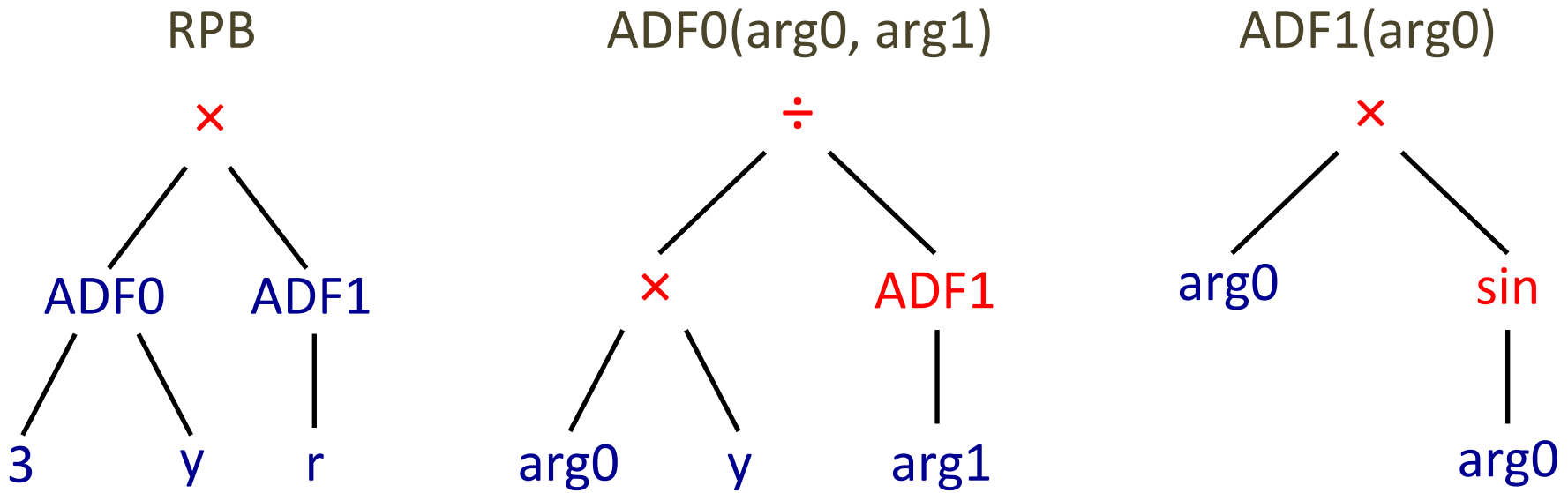
Modularity

- Sub-expressions frequently re-occur in a program
  - Ideally, we only want to evolve each one once
  - This:



Automatically-Defined Functions

- A program is defined as a forest of trees
  - One “result producing branch” (RPB), i.e. main()
  - One of more ADFs, each with one or more arguments



Automatically-Defined Functions

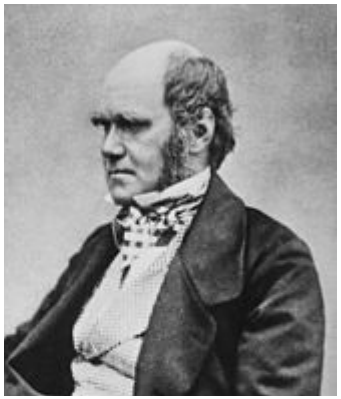
- ADFs must be defined before a GP run
  - Both the number of ADFs, and the number of arguments for each ADF must be specified in advance
  - This is a prominent limitation of ADFs

- Some heuristics for choosing these values:
  - *A priori* knowledge of problem decomposition
  - Over-specification, i.e. more than will ever be needed
  - Affordable capacity, since ADFs tend to increase the complexity and hence the execution time of programs

Automatically-Defined Functions

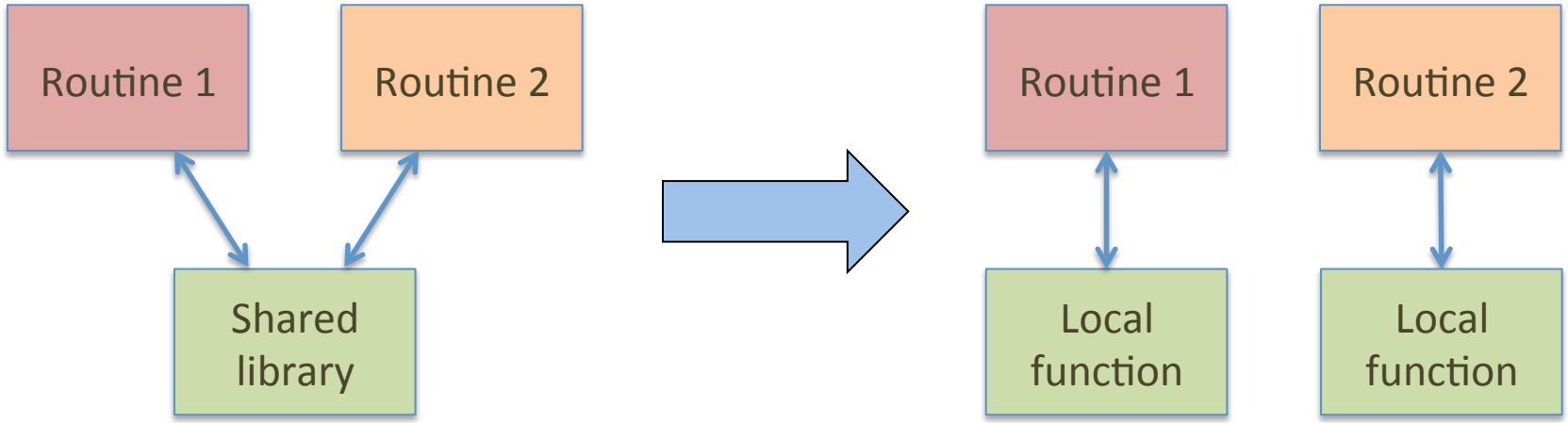
- Good things about ADFs ☺
  - They can reduce overall program size
  - They make it easy to solve modular problems
  - They have been used to solve hard problems
  - See Koza's books!

- Bad things about ADFs ☹
  - They can increase program complexity
  - Incorrect parameter settings hinder evolution
  - Modular dependencies may hinder evolution…



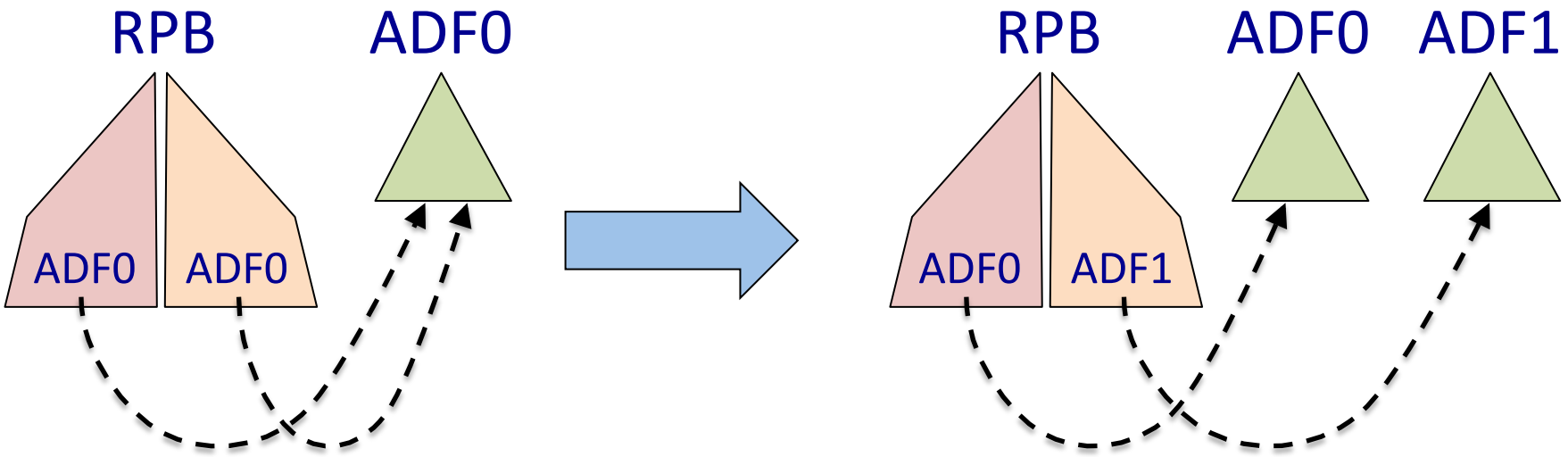
Modular Dependencies

- This is also seen in software engineering
  - Two routines make use of shared code
  - The routines' requirements diverge
  - The shared code must be copied and modified
  - i.e. it is no longer shared code!



Removing Dependencies

- Koza introduced a similar mechanism to GP
  - Called ‘case splitting’
  - Part of a group of ‘architecture altering operators’
  - Basically mutation operators that refactor code



Analytical Modularity

- Another group of methods for achieving modularity
  - These attempt to identify useful code blocks
  - And then turn them into modules
  - For potential use elsewhere in evolving programs

- Adaptive Representation through Learning (ARL)
  - A successful form of analytical modularity
  - First, it identifies programs with improved fitness
  - Then extracts the most-executed code blocks
  - Placing these in a library shared between programs
  - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.3333

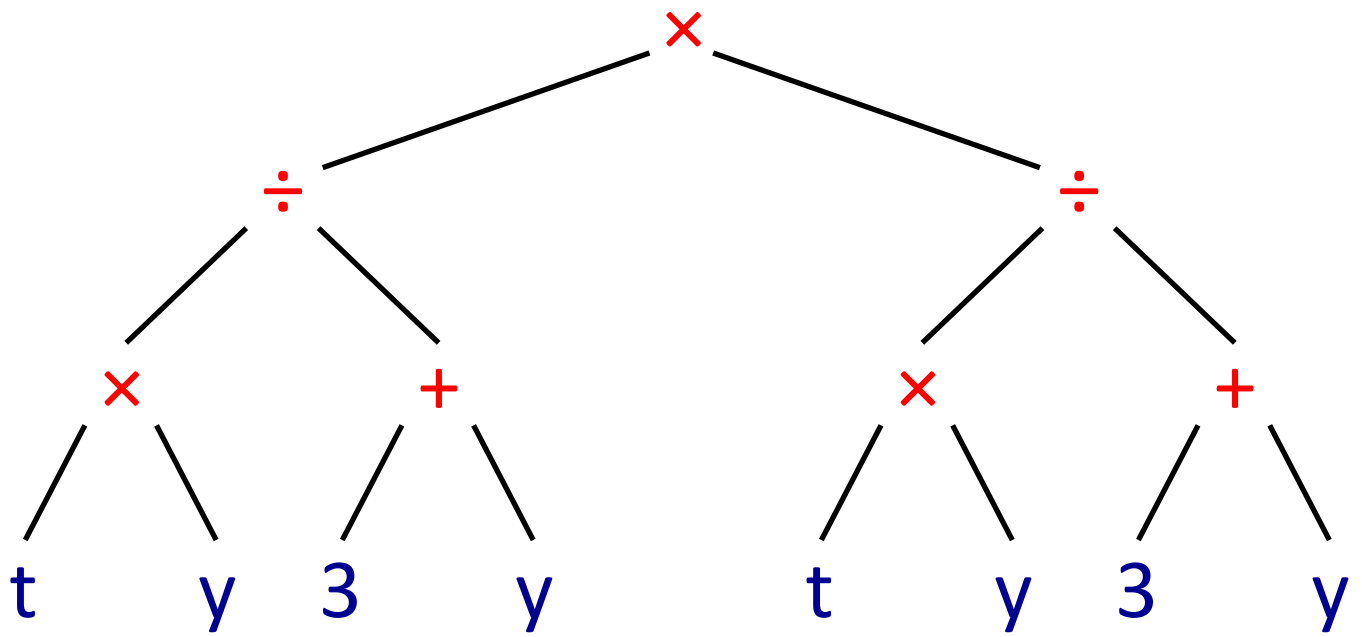
Recap

- It is possible to add syntactic features to GP:
  - Types ✔
  - Memory ✔
  - Loops ✔
  - Modularity ✔

- But this has consequences:
  - More complex initialisation and variation operators
  - More constraints during evolution
  - Possible biases within the search landscape
  - **So, only use them when necessary**

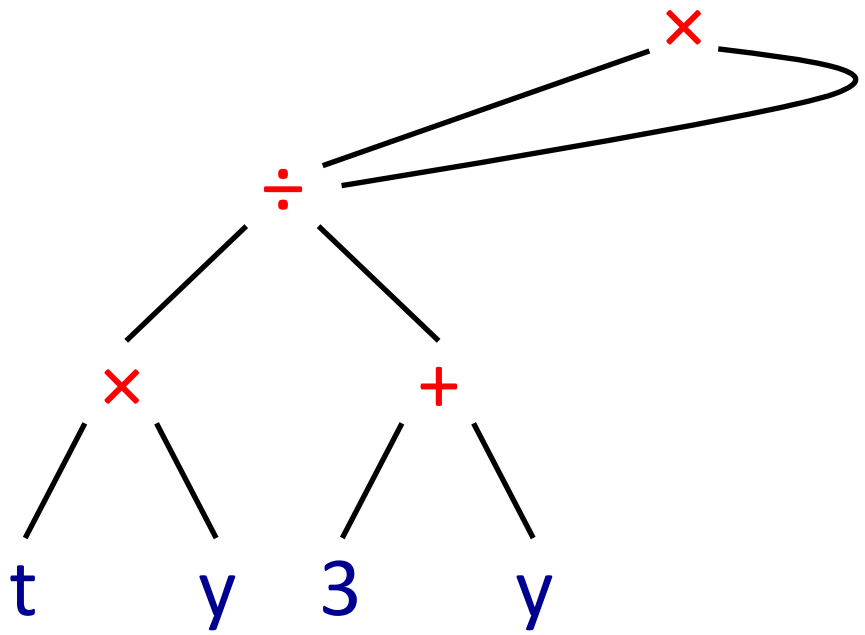
Going Graphical

- Some limitations of GP are due to using trees:



Going Graphical

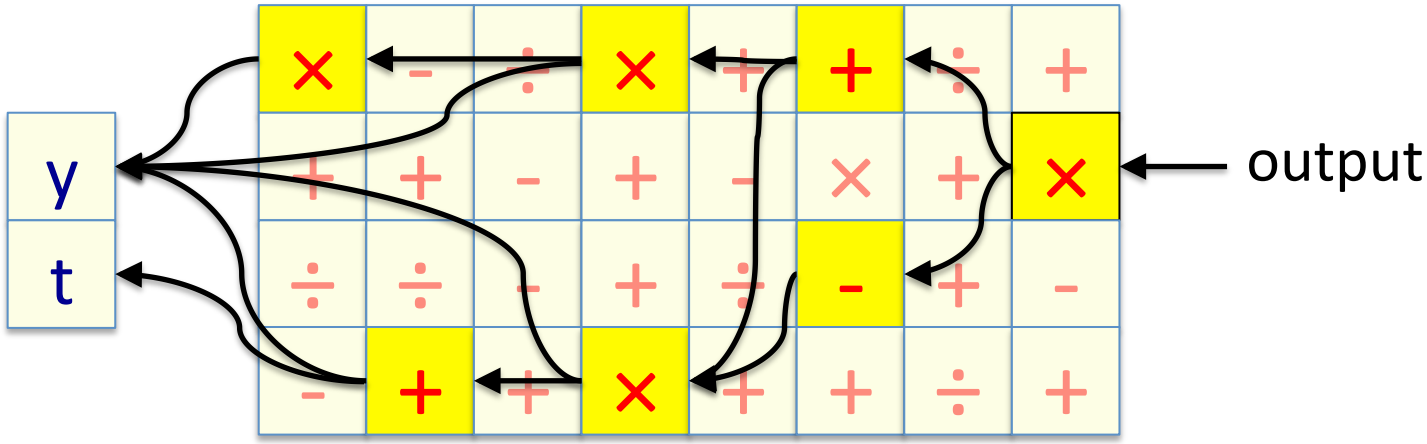
- There are advantages to using graphs instead
  - Instant reuse!



Going Graphical

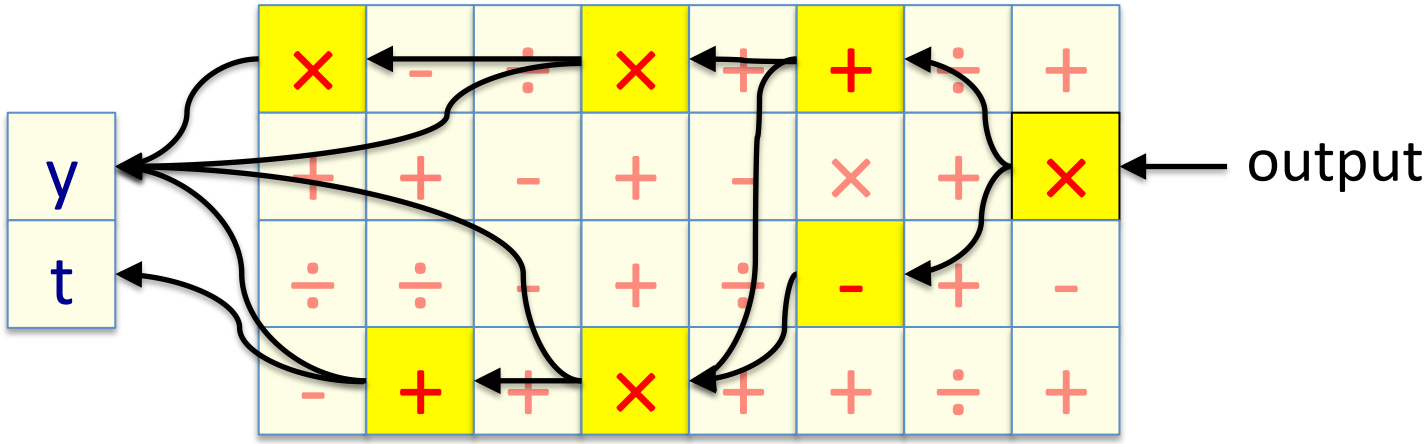
- There are a number of graph-based GPs
  - PADO, PDGP, GNP, CGP, …

- Cartesian GP (CGP) is the best known [Miller]
  - Functions are arranged on a Cartesian grid



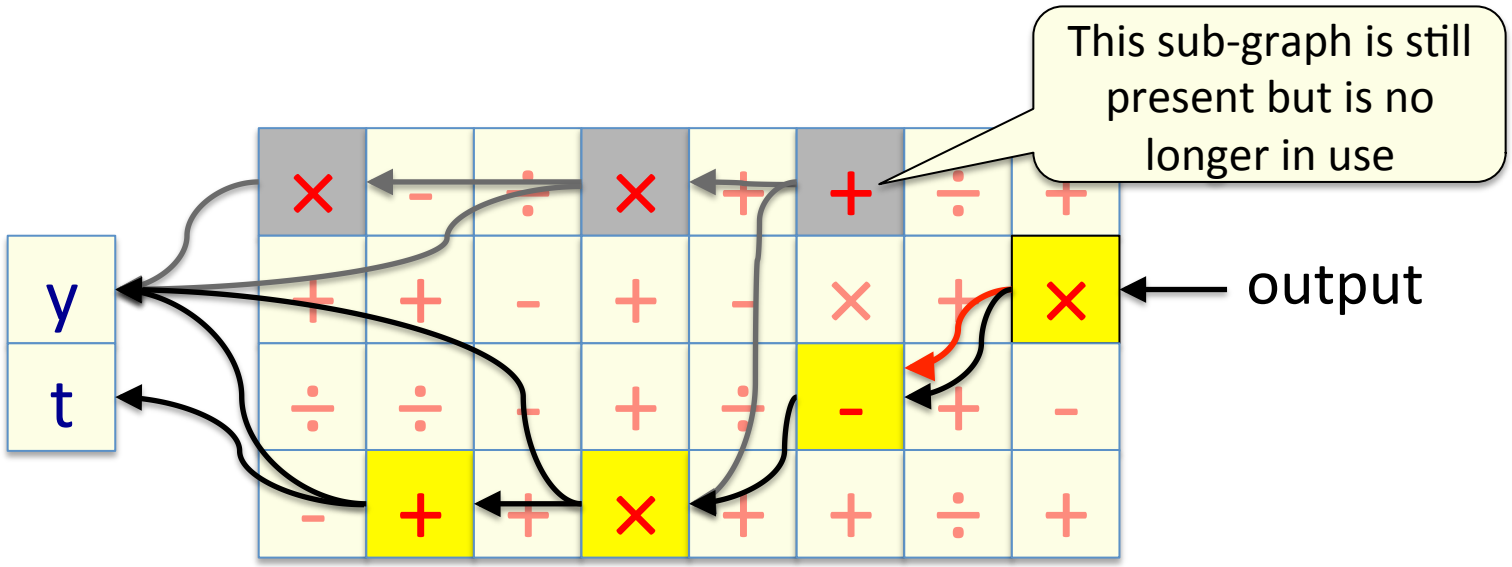
Cartesian GP

- Other notable properties of CGP
  - Constrained grid limits program size (no bloat!)
  - Mutation can connect/disconnect nodes
  - Disconnected nodes are a form of redundancy
  - Redundancy has evolutionary advantages



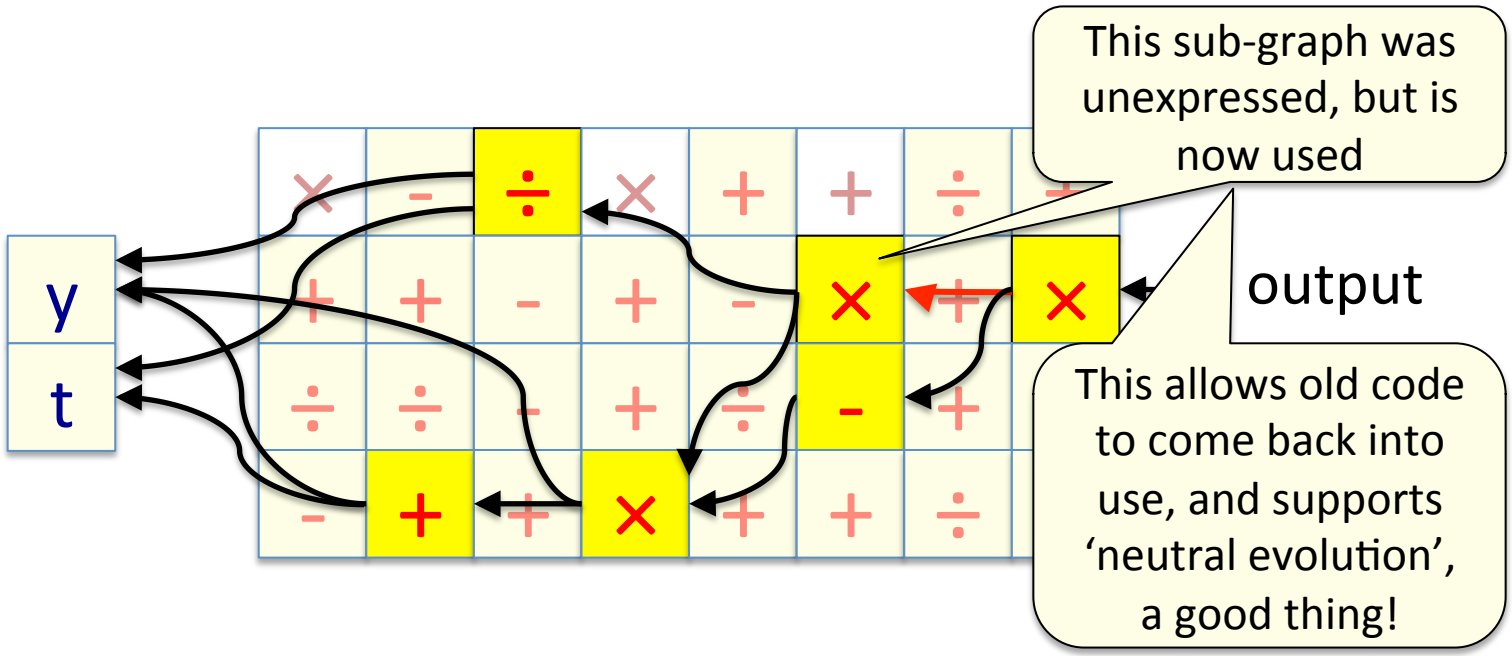
Cartesian GP

- Other notable properties of CGP
  - Constrained grid limits program size (no bloat!)
  - Mutation can connect/disconnect nodes
  - Disconnected nodes are a form of redundancy
  - Redundancy has evolutionary advantages



Cartesian GP

- Other notable properties of CGP
  - Constrained grid limits program size (no bloat!)
  - Mutation can connect/disconnect nodes
  - Disconnected nodes are a form of redundancy
  - Redundancy has evolutionary advantages



Summary

- This week
  - Some ways of making tree GP more expressive
  - Some thoughts on how this effects evolvability
  - The potential for using different representations

- Next week
  - Other ways of representing programs
  - Turing complete genetic programming
  - What we can learn from biology

Homework

- ADFs
  - ECJ has support for these
  - Try them out on a few problems
  - Try using different numbers of ADFs

- CGP
  - Add-on available for ECJ
  - http://oranchak.com/cgp/contrib-cgp-18.zip
  - http://oranchak.com/cgp/doc/
  - Have a look, try it out