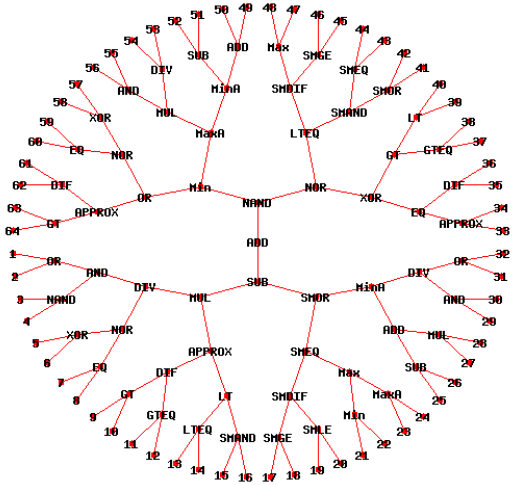
Previous Lecture

- 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

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

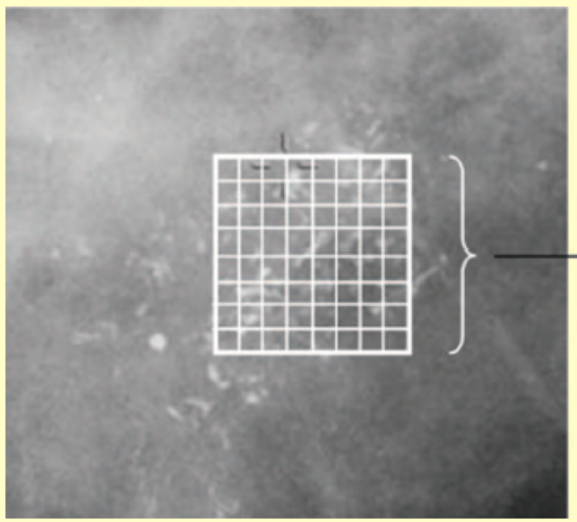
Cartesian Genetic Programming and Its Application to Medical Diagnosis

I. Introducti

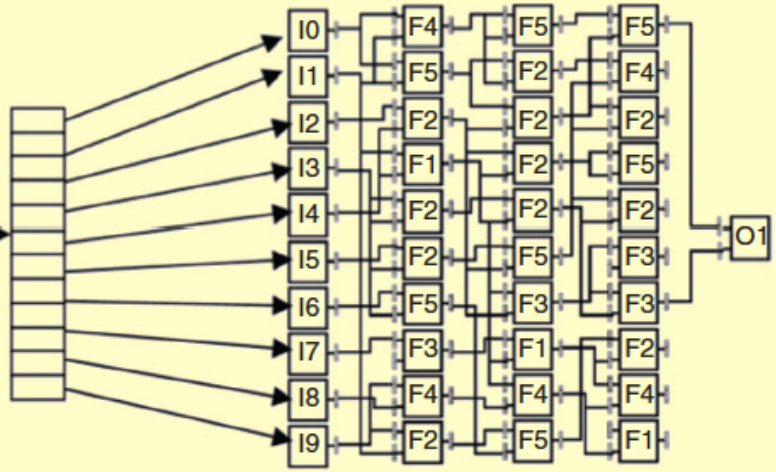
Comp
tions
relev
lific
and biometri

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

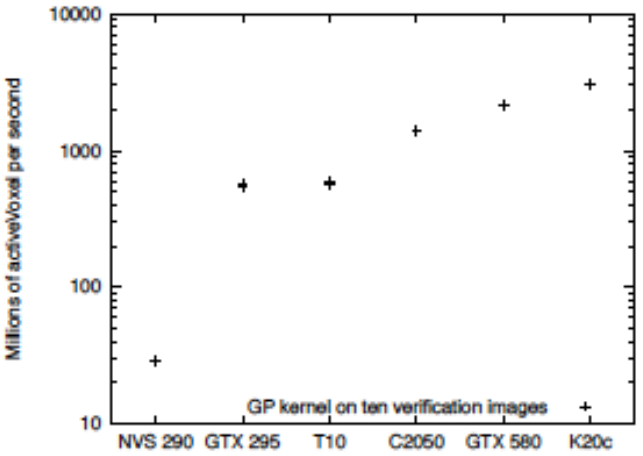


Figure 1: Performance of modified reg_spline_getDeformationField3D CUDA kernel after optimisation

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.

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

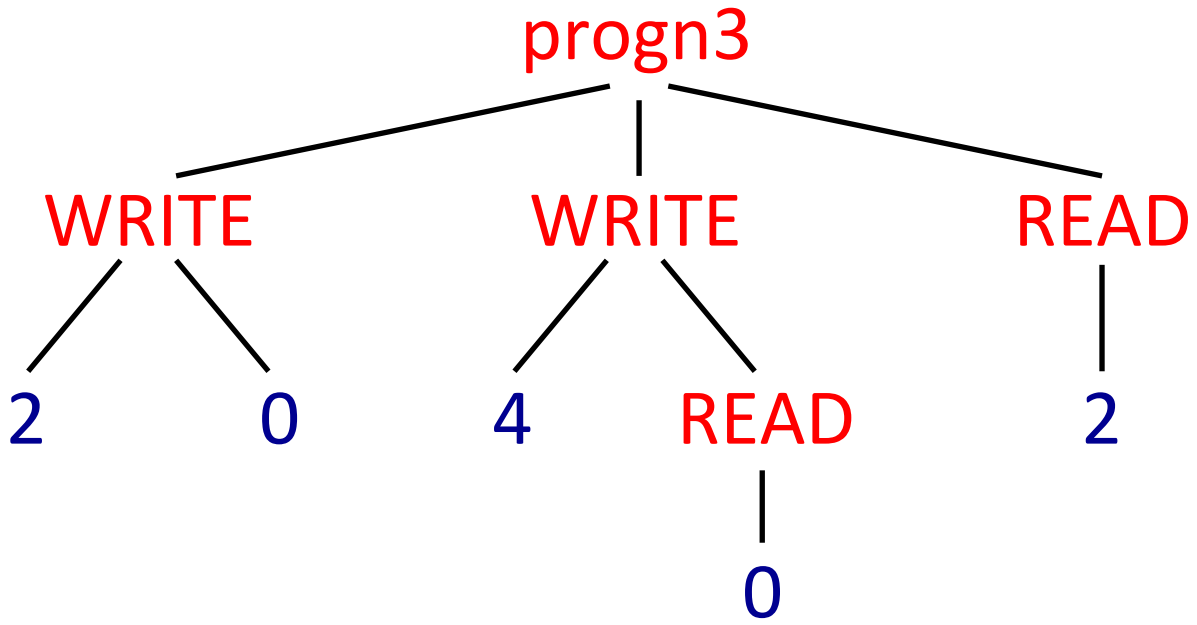
Adding Memory

- 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:

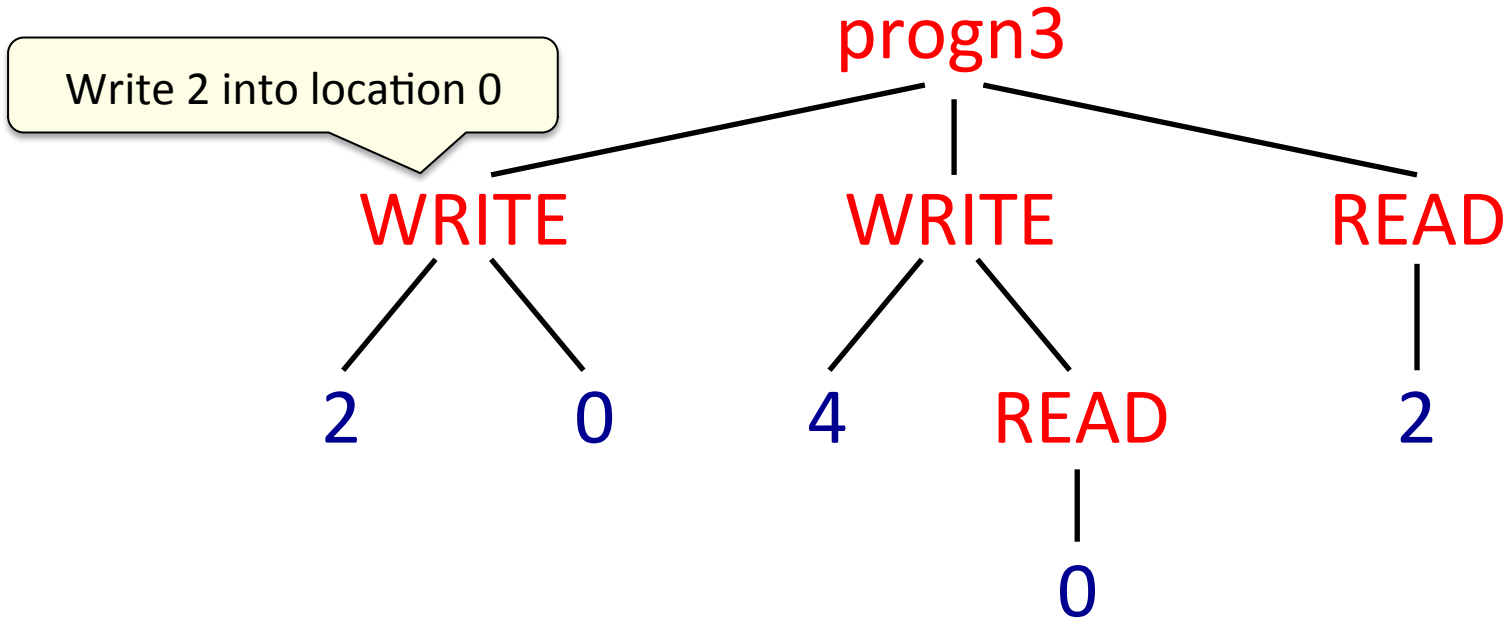
  - And 2 new functions:
    - READ(X) – read value from memory location X
    - WRITE(Y, X) – write value Y to memory location X

Example



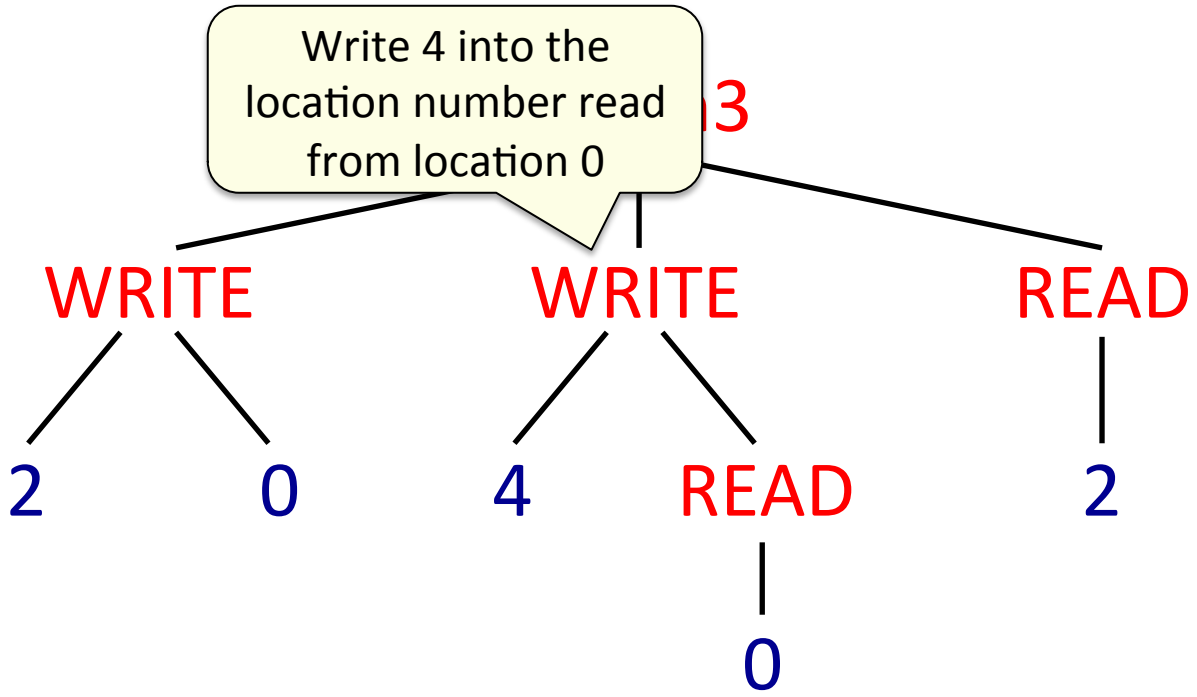
Example


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

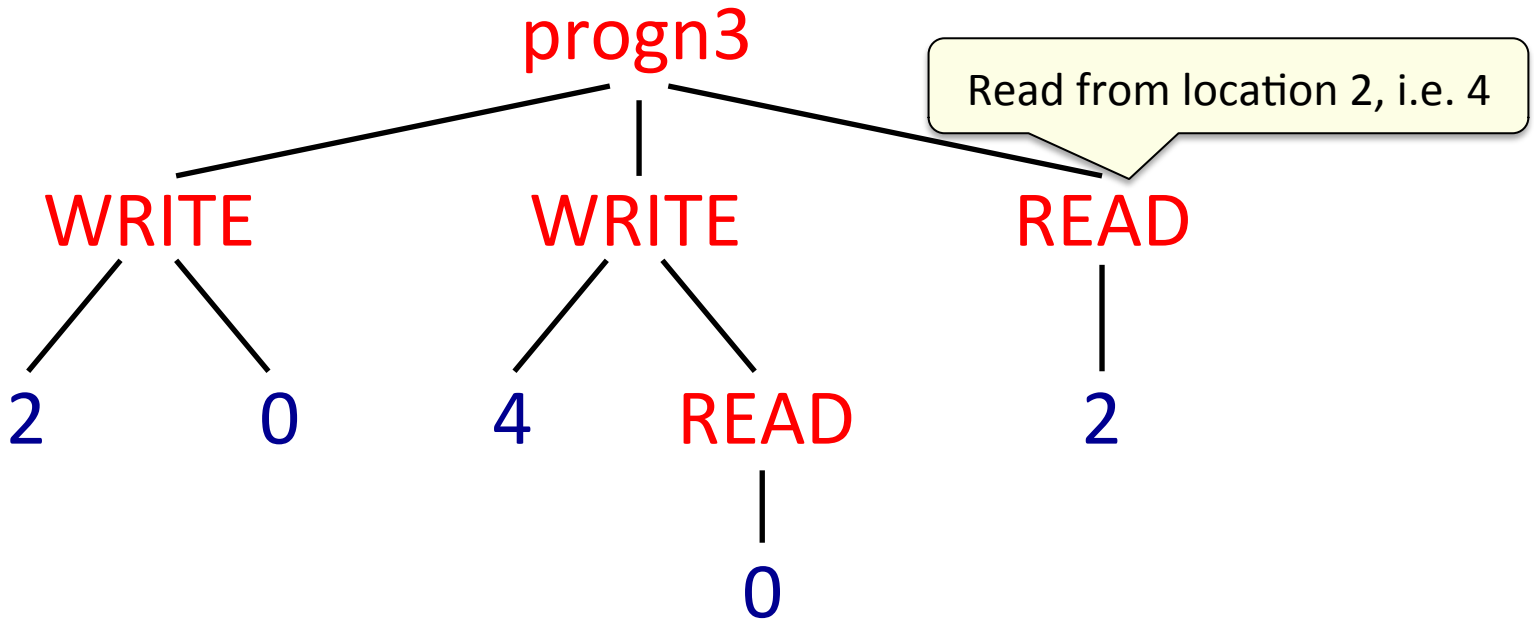
Example


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

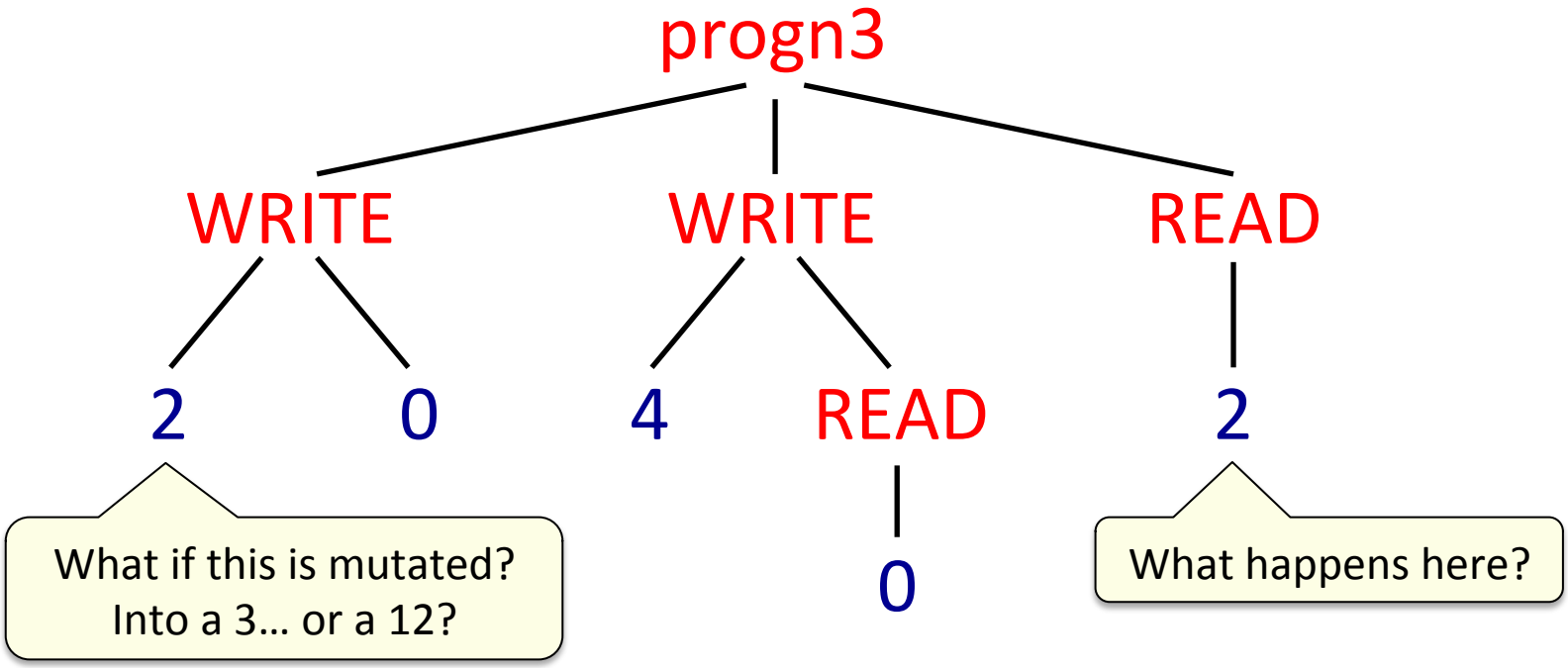
Example


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

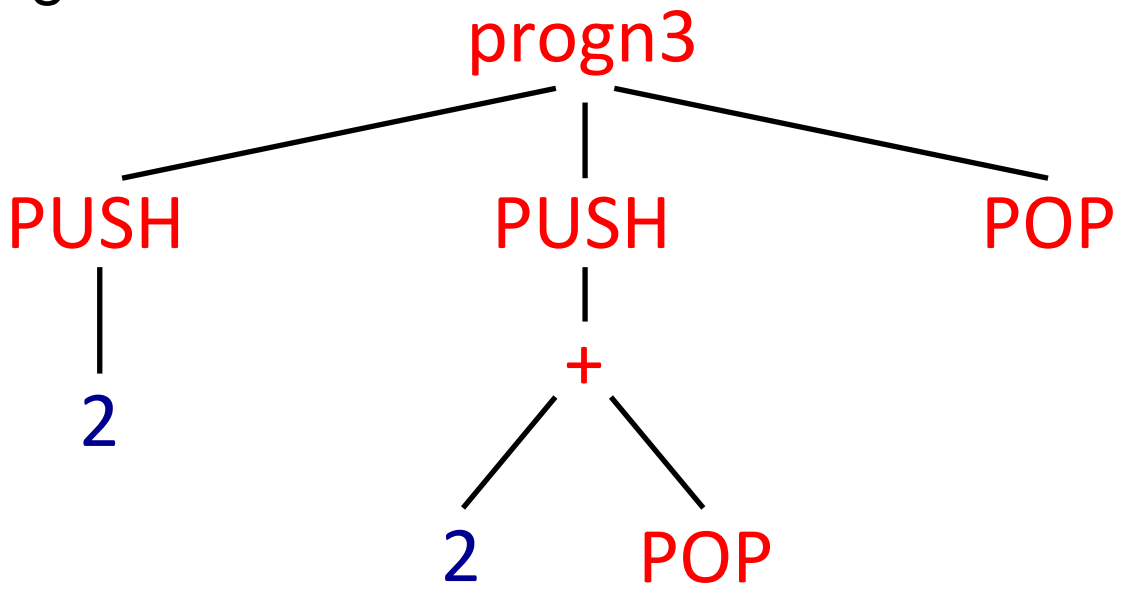
Example


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:

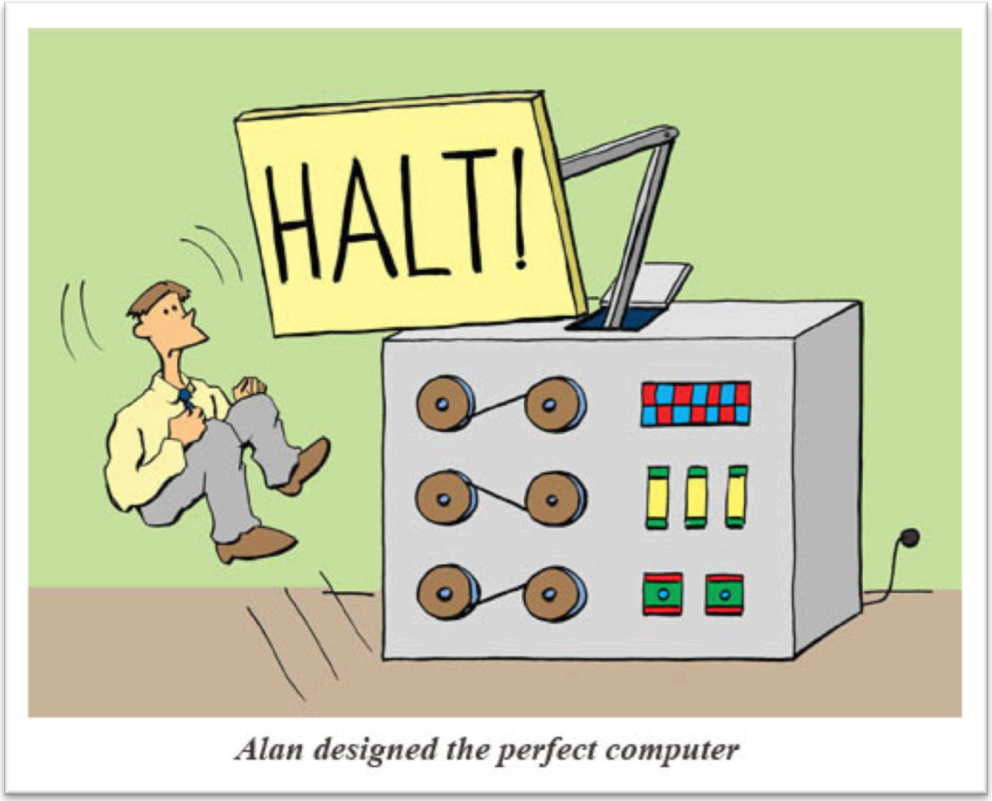


Other Memory Approaches

- "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

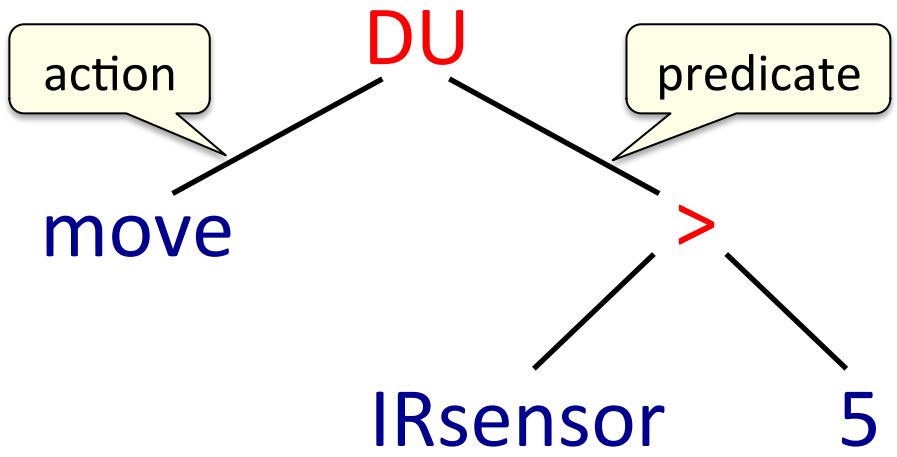
- This is possible, but…
  - Requires caution!
  - Halting Problem: can't predict termination
  - Need to use constraints to prevent infinite loops
  - (e.g. max iterations)
  - Or use a time-out during solution evaluation



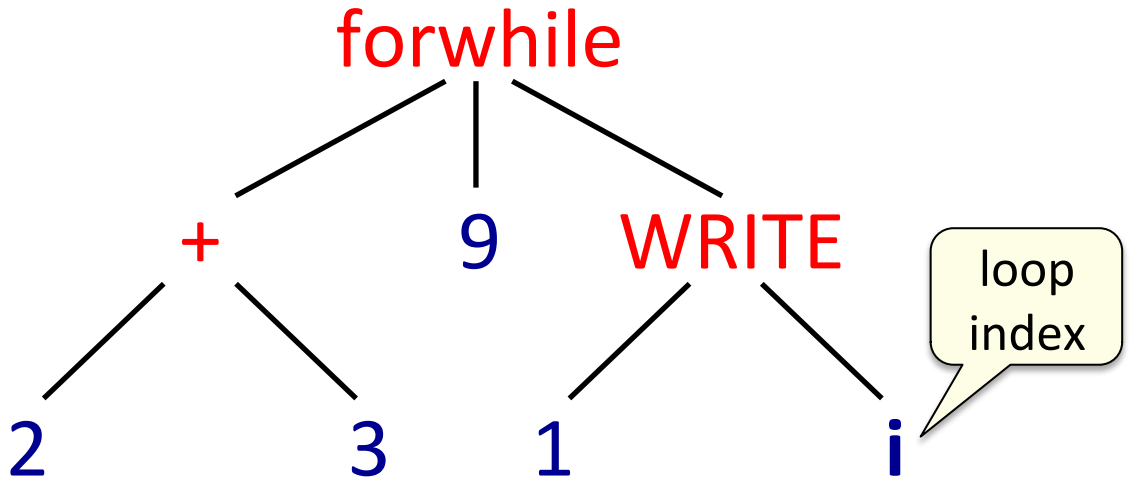
http://www.coopertoons.com

Adding Loops

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



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



Loops for Image Processing

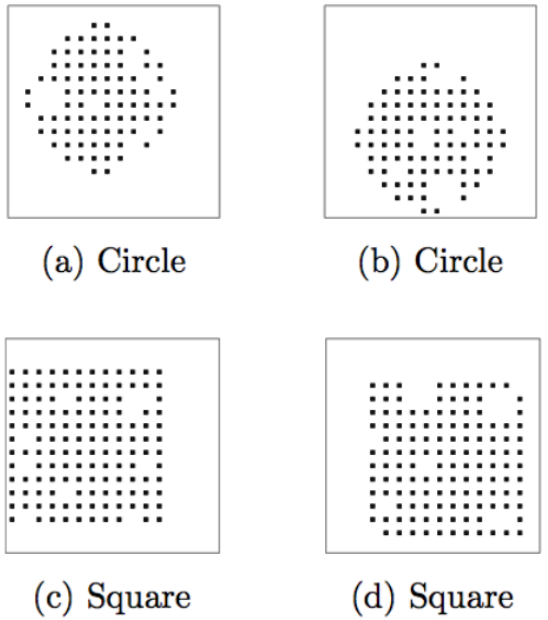
- Iteration has been found useful for image analysis
  - Often processed in segments, e.g. every 10x10 pixels
  - Evolved loops can be more efficient and effective
    - http://goanna.cs.rmit.edu.au/~vc/papers/aust-ai04.pdf

Using Loops in Genetic Programming for A Two Class Binary Image Classification Problem

Xiang Li and Vic Ciesielski

{xiali, vc}@cs.rmit.edu.au
School of Computer Science and Information Technology
RMIT University, GPO Box 2476V, Melbourne, Victoria 3001

Abstract. Loops are rarely used in genetic programming (GP), because they lead to massive computation due to the increase in the size of the search space. We have investigated the use of loops with restricted semantics for a problem in which there are natural repetitive elements, that of distinguishing two classes of images. Using our formulation, programs with loops were successfully evolved and performed much better than programs without loops. Our results suggest that loops can successfully used in genetic programming in situations where domain knowledge is available to provide some restrictions on loop semantics.



(a) Circle (b) Circle

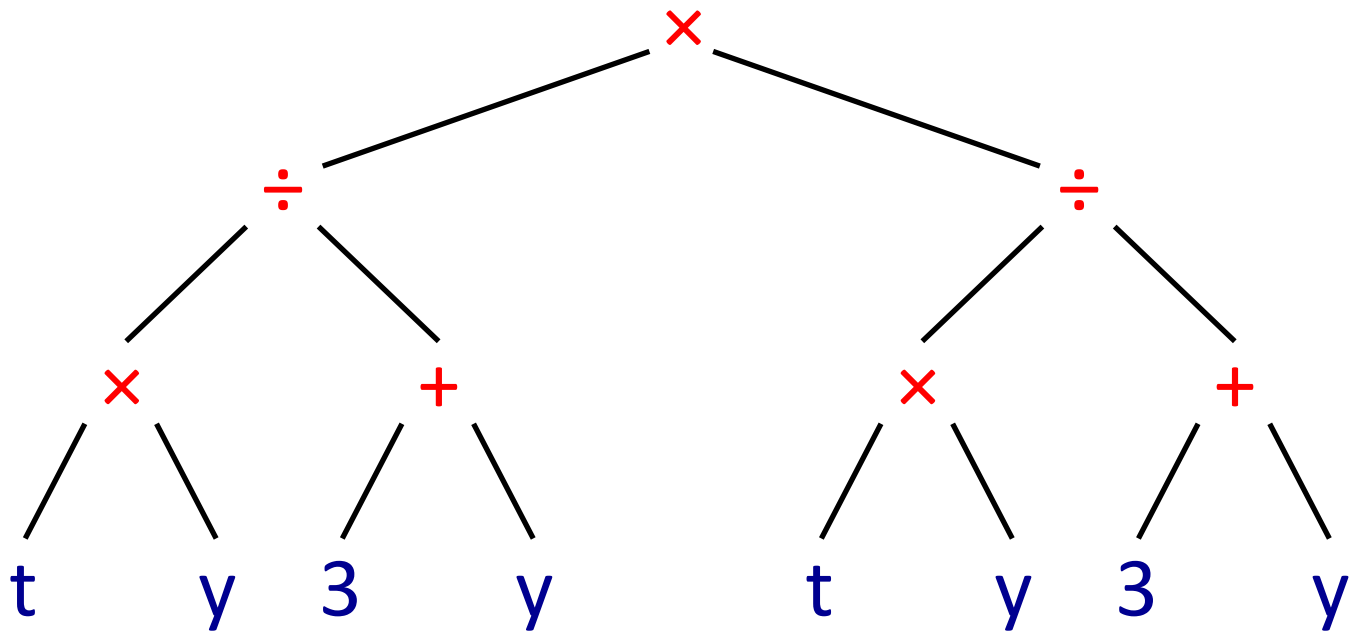
(c) Square (d) Square

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**

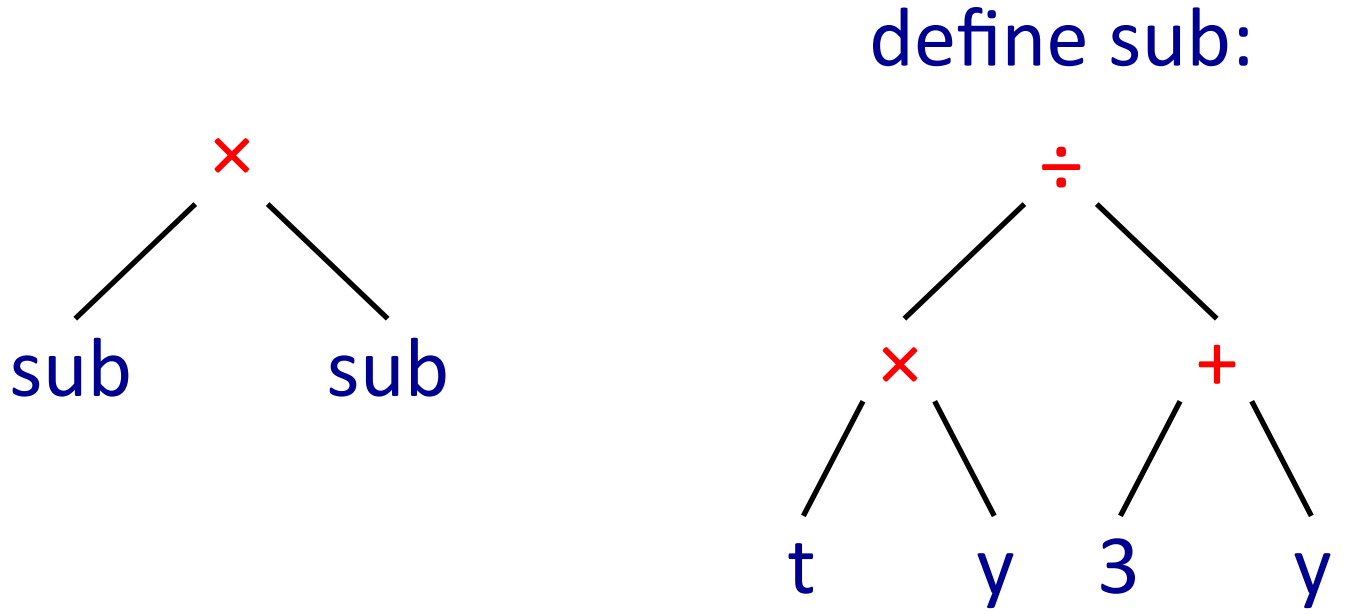
Modularity

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



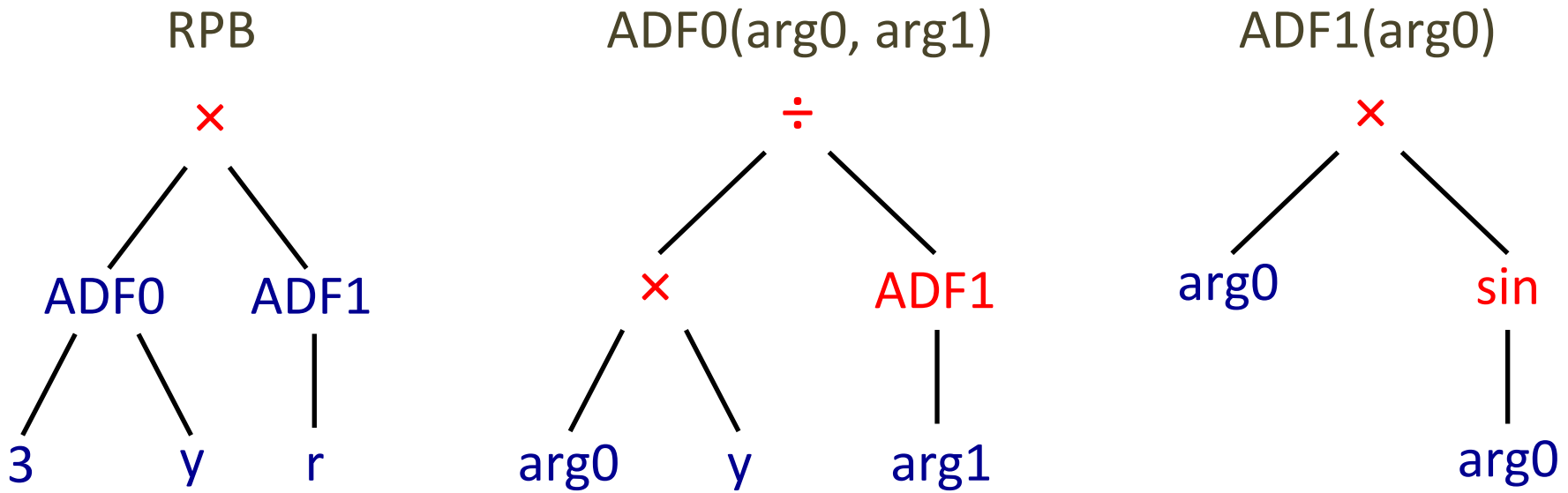
Modularity

- 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

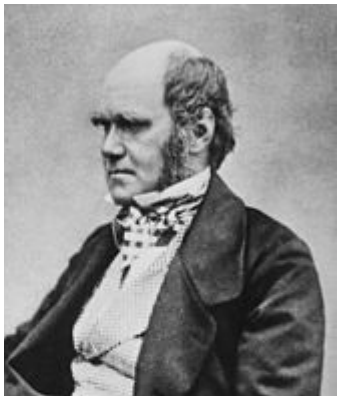
- ADFs must be defined before a GP run
  - Both the number of ADFs, and the number of arguments for each ADF must be specified in advance
  - This is a prominent limitation of ADFs

- 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

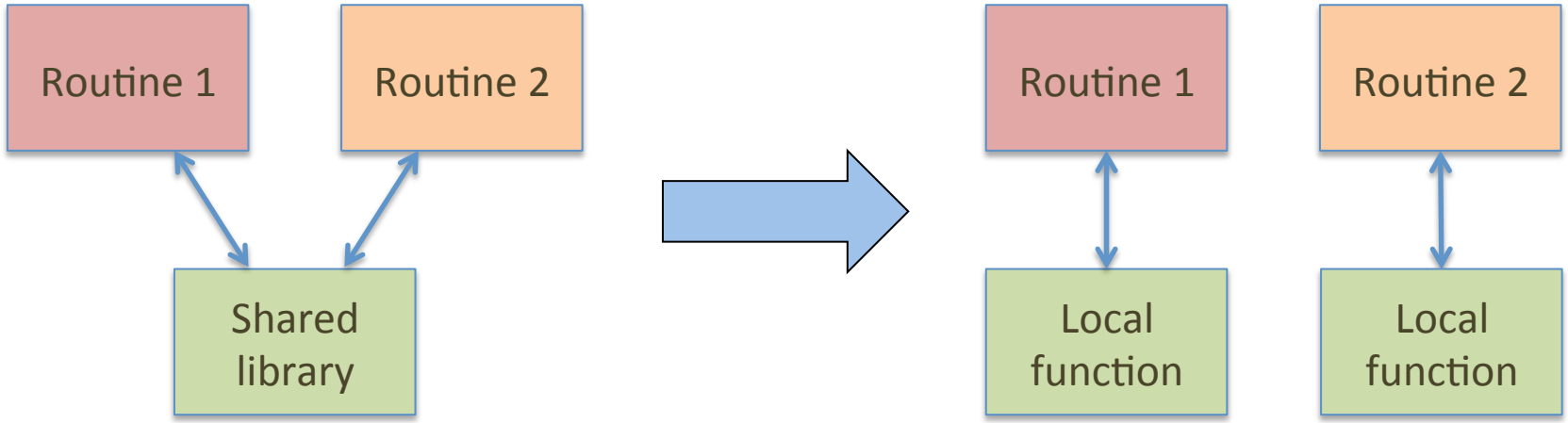
- Good things about ADFs ☺
  - They can reduce overall program size
  - They make it easy to solve modular problems
  - They have been used to solve hard problems
  - See Koza's books!

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



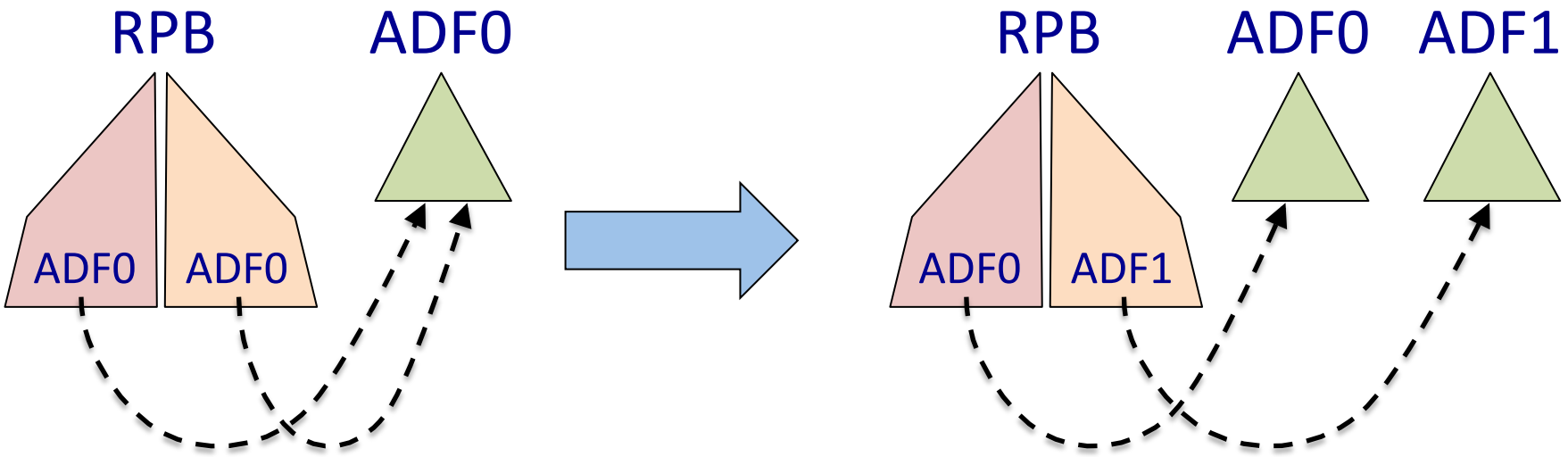
Modular Dependencies

- 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

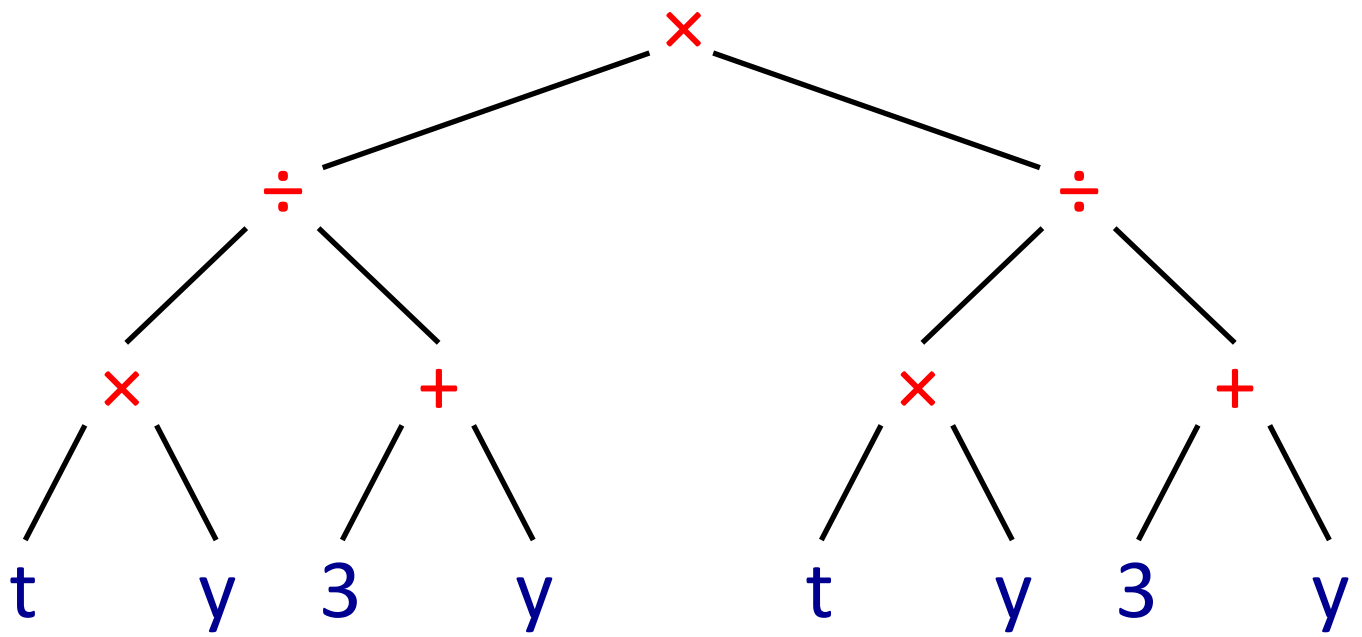
Recap

- 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**

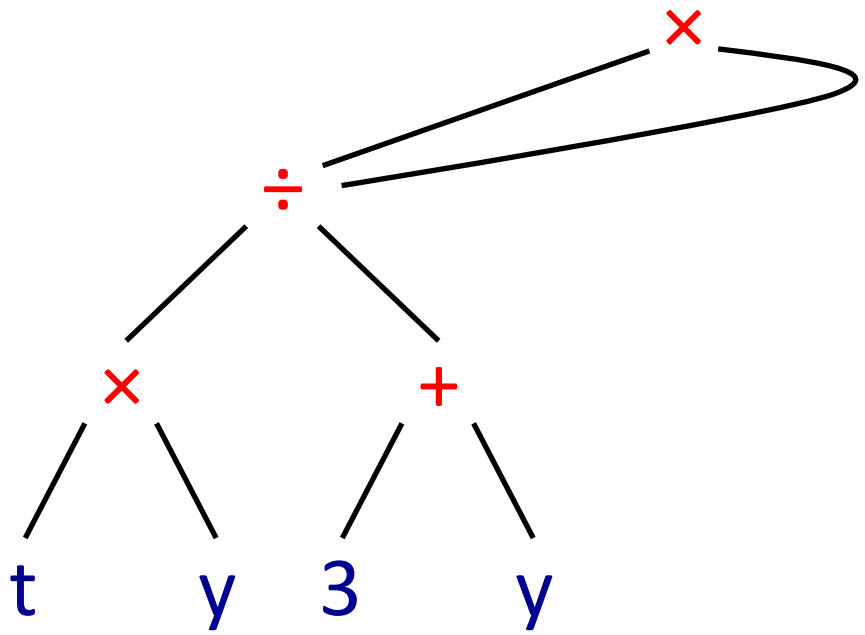
Going Graphical

- Some limitations of GP are due to using trees:



Going Graphical

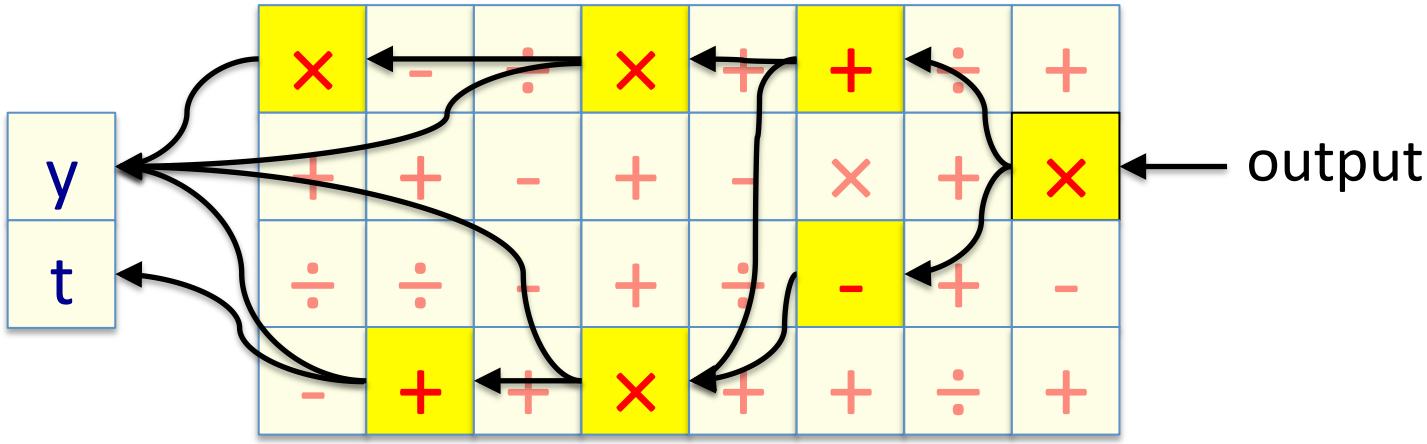
- There are advantages to using graphs instead
  - Instant reuse!



Going Graphical

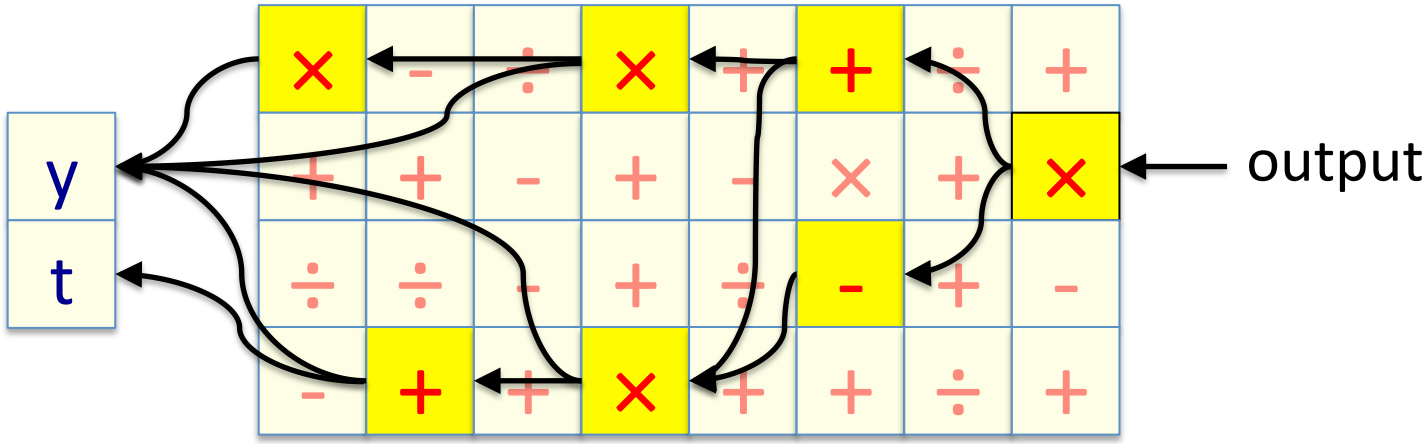
- There are a number of graph-based GPs
  - PADO, PDGP, GNP, CGP, …

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



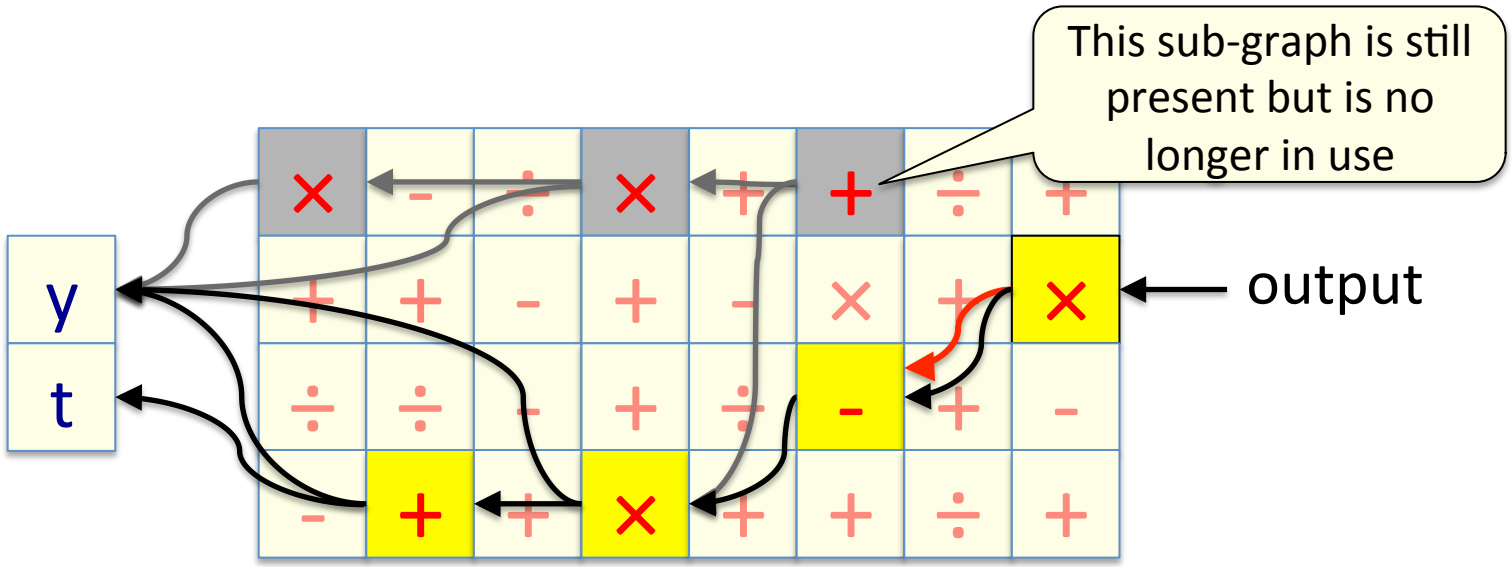
Cartesian GP

- 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



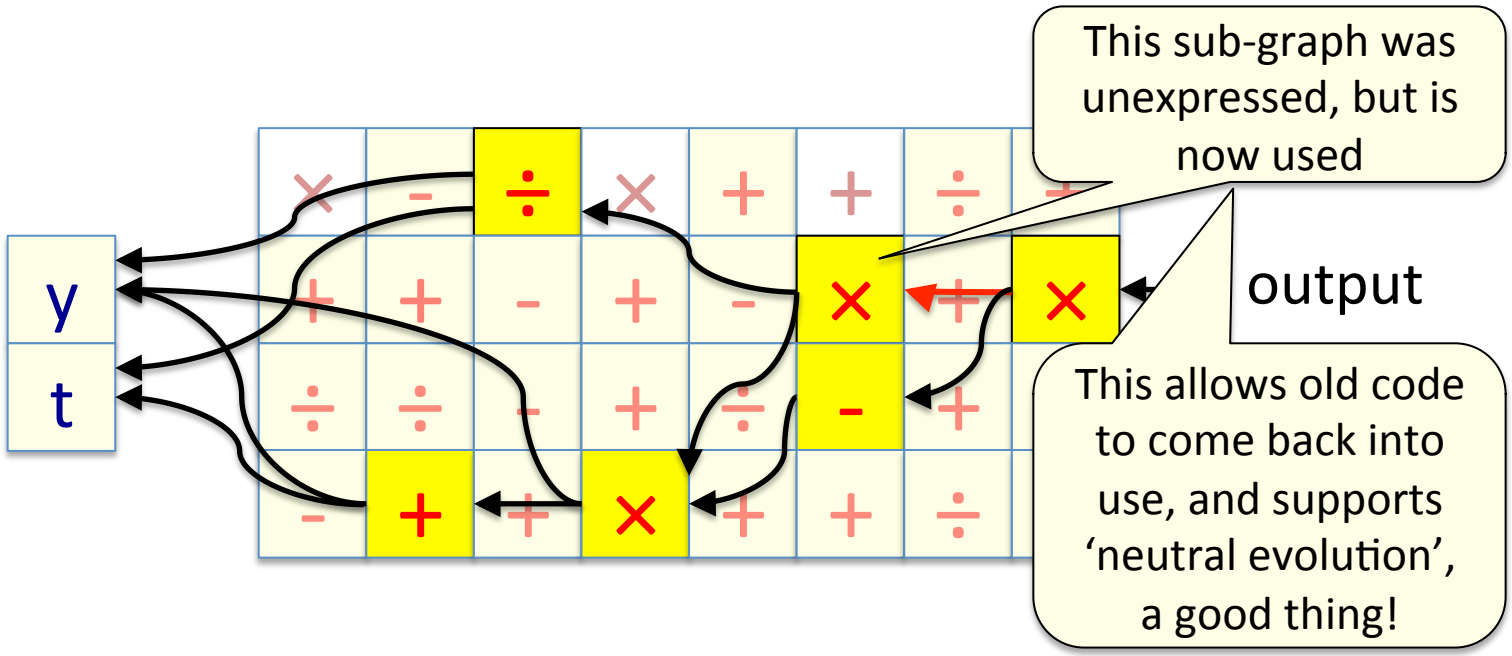
Cartesian GP

- 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



Things you should know

- The meaning of evolvability and expressiveness
  - And that there is often a trade-of between them
  - And be able to give examples of this trade-off
- ADFs: what are they, why use them, rules of thumb
- Cartesian GP: what is it, why use it
- I don’t expect you to know:
  - Details of GP memory and loop implementations
  - About modular dependencies and analytical modularity

Summary

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

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

Things to try out

HERIOT WATT UNIVERSITY

- 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

Bibliography

- W. Langdon, Data Structures and Genetic Programming, PhD thesis, 1996http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/langdon_thesis.html
- J. Miller and P. Thomson, Cartesian genetic programming, EuroGP 2000 http://www.cartesiangp.co.uk/papers/eurogp2000-miller.pdf
- A. Teller, Turing Completeness in the Language of Genetic Programming with Indexed Memory, WCCI 1994, http://astroteller.net/pdf/Turing.pdf