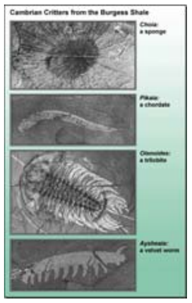
Computational Development



- Life on earth was single-celled for 3 billion years
- Multicellular life appeared at the Cambrian Explosion,
- Large organisms with many body plans appeared
- Exploited new niches

Computational Development (2)

- EC is limited to small problems like acellular life
- A Cambrian Explosion in EC might allow the evolution of solutions to large and complex problems
  - Microprocessor design
  - Passenger jet design
  - Skyscraper design
- The process that turns a single cell into a large organism is *development*

Benefits of Development

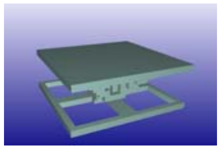
- Primarily *scalability*:
- Evolution creates large, complex phenotypes using development
- It’s the only solution nature found in 4 billion years
- By modelling features of development we might be able to create large complex phenotypes too

Other Benefits

- Multicellular organisms are fault tolerant
  - Redundancy on many levels
- The developmental process itself is robust to faults
  - Splitting an embryo produces twins
  - Starfish, newts regrow missing limbs
  - These phenomena rely on development
- So developmental robustness provides another avenue to fault tolerance
- Biological development is poorly understood
  - Modelling might provide insight into how it really works

Other Benefits (2)

- 2D / 3D structure design
  - nature has evolved a way to map from 1D chromosomes to 3D structures over aeons
  - the basic concepts might allow us to easily map to 2D structures too



Problems that might benefit from scalability

- Problems where we know that EC is outperformed by other methods are currently under exploration:
  - Circuit design (c.f. traditional methodologies)
  - Software design (c.f. traditional methodologies)
  - ANN design (c.f. biological NN design)
- But in reality many problems are eventually limited by the scalability of EAs

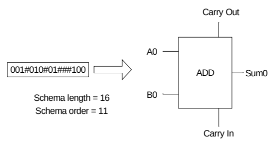
What is the scalability problem?

- EC works well with small chromosomes
  - Small search space, easily sampled
- Large chromosomes don't work well
  - Combinatorial explosion of search space
  - EC can't sample space effectively
  - Converges to suboptimal solution
- Why?

Relating Schema Theorem

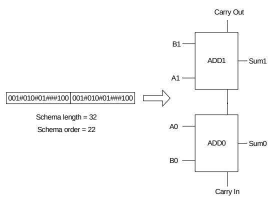
- We can think of it in terms of schema theorem
- Large, high order schemata are likely to be disrupted by evolutionary operators
- Solutions to large real-world problems are likely require large, high order schemata

Toy Example: Adders



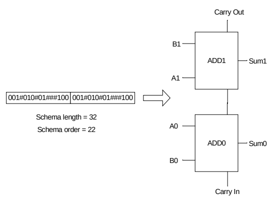
- A 1 bit adder can be evolved easily
- Assume fitness is only rewarded when perfect schema is discovered
- (In reality fitness reward is more gradual)
- Evolution must discover length 16, order 11 schema before fitness reward

Toy Example: Adders (2)



- A 2 bit adder can be made by joining 2x1 bit adder
- Evolution must learn the design for ADD 1 independently of its discovery of ADD0
- A pain, but not a show-stopper

Epistasis



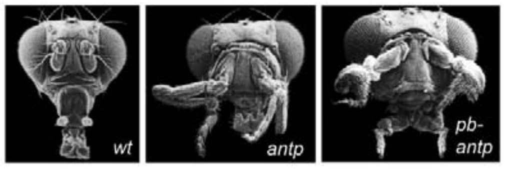
- The problem is worse:
- Output Sum1 *relies on* the carry out from ADD0

∴ Fitness from Sum1 will only be rewarded if ADD0 is intact

- Fitness for ADD1 only rewarded if requires schema twice as long and twice as big as the schema for Sum0
- Linkage between genes that results in nonlinear fitness payoff is called *epistasis*

How does development tackle large problems

- We don't completely know, but:
- A mutation in a single gene can transform one complex feature into another
- e.g. antennae->legs, proboscis->legs



What does this mean?

- One gene can’t describe an entire leg design
- Nature seems to have a simple mechanism to *reuse* leg design
- The generation of the mis-placed legs is almost perfect
- As they develop they are not *interacting* with the surrounding tissues
- Its generation seems to be *independent* of surrounding tissues
- It can be thought of as a developmental *module.*

How can this improve adder evolution?

- Large adders could be built by *re-using* the 1 bit adder design
- A bias towards *modules* that do not interact might minimise the problem of epistasis
- Later it will be shown how development allows re-use and provides modularity
- Later it will be shown how we can model development to gain these features

Fundamental Processes – Regional Specification

- RS is simply pattern formation
- Process where spatial and temporal pattern of cell activities is organised
- Cells acquire different identities
- Identities defined by chemical differences
- Differentiation into functional cell types happens later
- Occurs throughout early stages of development

Fundamental Processes – Cell Differentiation

- Cells become structurally and functionally different from each other
- Cells assume one of a few distinct cell types
  - e.g. skin, liver etc
- One-way process
- Gradual process, occurs throughout development

Fundamental Processes - Morphogenesis

- Movement of cells and tissues that alter the form of the embryo
- Active during early/mid-development
- Many strategies
  - Alteration in cell adhesion
  - Cell division
  - Apoptosis (Programmed cell death)

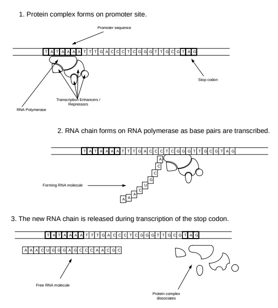
Fundamental Processes - Growth

- Embryos do not increase in size until the basic structure of the embryo has developed
- Most size increase results from growth at the end of development
- Growth is mostly due to cell division
- Some morphogenesis can arise through differences in growth rates of cells

Differential Gene Activation

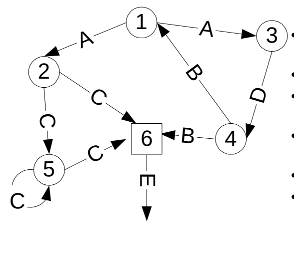
- Development’s engine is gene activation, producing proteins
- Engine is directed by the differential activation of genes
- Activation of genes in different cells produces different chemical environments
- Gives cells different identities, allows differentiation
- The majority of DGA results from DNA *transcription*

DNA Transcription


1. Protein complex forms on promoter site.
Promoter sequence
TATAAAATTTGACCCTCGGGTTGCGTAG
Stop codon
Transcription Enhancers / Repressors
RNA Polymerase
2. RNA chain forms on RNA polymerase as base pairs are transcribed.
Forming RNA molecule
AAACUGCCA
3. The new RNA chain is released during transcription of the stop codon.
AAACUGGGAGCCCAACGC
Free RNA molecule
Protein complex dissociates

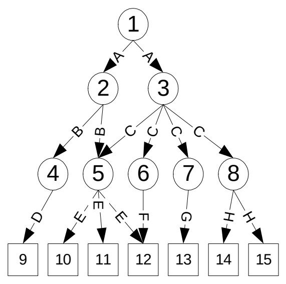
- Protein complex binds to a gene
- Complex travels along gene
- Generates complementary RNA
- RNA translated into protein by cellular machinery
- Complex made of proteins called *transcription factors*
- Complex is gene-specific
- Correct combination of TFs must be present for gene to be trascribed
- ∴ TFs control transcription rate
- Transcription factors are themselves gene products

Gene Regulatory Networks



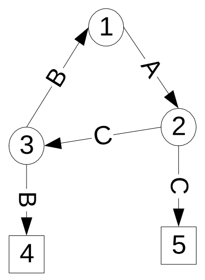
- Gene interaction can be modelled as a network
- GRN is a directed graph with labelled nodes and edges
- Nodes represent genes
- Edges represent gene products (TFs)
- Edges link parent to regulated gene
- Genes generate a single TF
- TFs can control multiple genes

GRNs Capture Modularity



- GRNs capture one possible mechanism for modularity
- Activation of a single master control gene causes a cascade of gene activity
- This can generate a complex feature in the phenotype
- The genes involved are a developmental *module*

GRNs Capture Re-use



- GRNs capture one possible mechanism for re-use
- A feedback loop in a GRN can cause a gene to be activated multiple times
- If it is a master control gene it may allow re-use of a complex feature

Intercellular Communication

- DGA explains how cells can differentiate
- DGA explains how a gene can be repeatedly activated
- Doesn’t explain how development forms iterative structures over space
  - segments
- This requires information to be transferred between cells
- 2 mechanisms
  - Cell division
  - Cell induction

Communication – Cell Division

- Cell division occurs twice
  - embryo cleavage
  - growth
- *Cytoplasmic determinant*: substance that guarantees that a cell assumes a particular state
- Inhomogeneities occur in CD concentrations within cells
- Cells divide
- Daughter cells contain different concentrations
- Results in cells with different states

Communication - Induction

- The main form of intercellular communication
- Transmission of chemical signals b/w cells
- Proteins (gene products) too big to pass through cell membranes
- Nature must use more complex processes

Methods of Induction

1. Intercellular proteins bind to receptors in cell membrane
   - activates a TF in the cell
2. Protein catalyses production of small molecule
   - passes through both cell membranes
3. TFs interact directly before cleavage
   - no cell membranes present
4. Pass signal through gap junction
   - cells must be touching

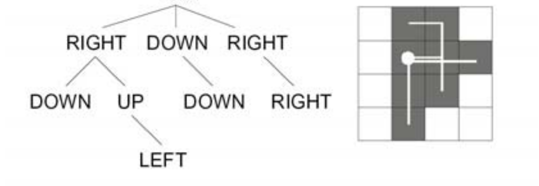
Models of Development used in EC

- Anything that the genotype is *executed* as a program to generate the phenotype
  - i.e. the phenotype 'grows'
- Models of development are surprisingly common
  - e.g. a tree can describe growth rules
  - it could be evolved using GP
- This is a very broad description – most models have more in common with biological development
- Developmental models can be broken into
  - Explicit
  - Implicit
    - L-Systems
    - Cellular

Explicit Development

- Developmental program is applied to a fixed 'embryonic' phenotype
- Explicitly specifies each developmental step like a computer program
- GP often used to represent the developmental program
- Nodes contain growth & modification instructions
  - Split component, change component,

Explicit Development: Toy Example



Where's the benefit?

- No implicit modularity
- No implicit re-use

BUT

- Additional control structures can provide this:
  - Modularity (e.g. ADFs)
  - Iteration (e.g. ADIs, ADLs)
  - Recursion (e.g. ADRs)

Implicit Development

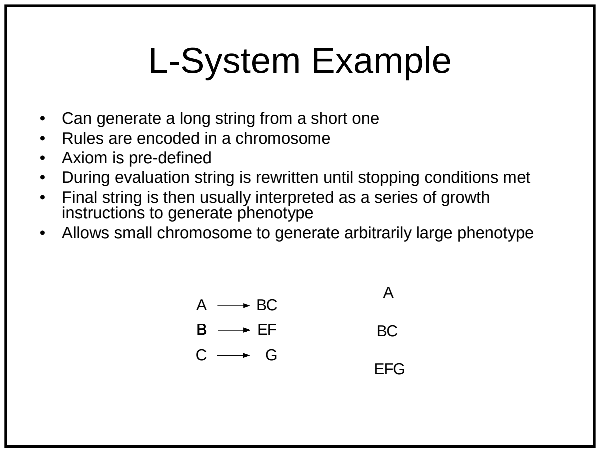
- Similar to the GRN model used in biology
- Consists of a set of rules
- Rule set implicitly defines a program through their interactions
- Rule's postcondition can be thought of as modelling a gene product
- Rule precondition can be thought of as modelling transcription factors required for rule activation
- Current work generally decomposes into 2 approaches:
  - L-Systems
  - Cellular

L-Systems

- An L-System is a set of rules
- Applied to a string called an *axiom*
- If symbol in rule's precondition is found in string it is replaced with symbols in rule's postcondition
- Successive applications repeatedly rewrite the string

- Rules are applied in parallel

- In most L-Systems:
- Rule precondition is always only one symbol long
- Rule postcondition is one or more symbols

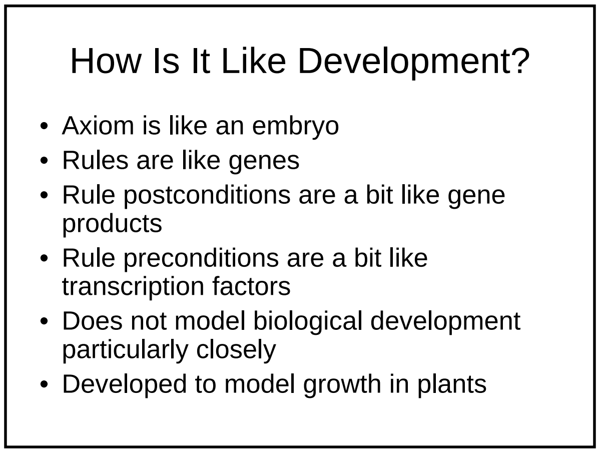


L-System Example

- Can generate a long string from a short one
- Rules are encoded in a chromosome
- Axiom is pre-defined
- During evaluation string is rewritten until stopping conditions met
- Final string is then usually interpreted as a series of growth instructions to generate phenotype
- Allows small chromosome to generate arbitrarily large phenotype

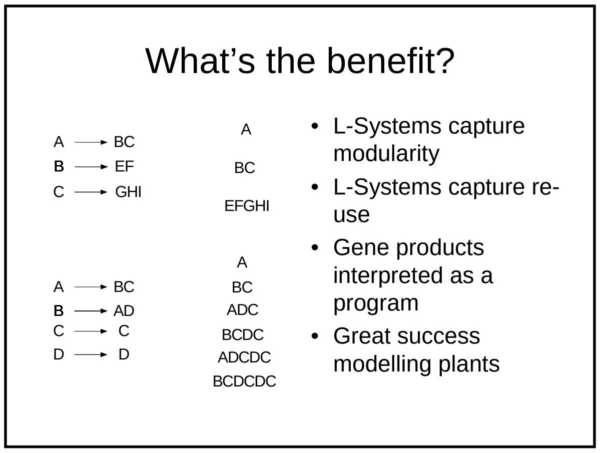
$A \longrightarrow BC$
$B \longrightarrow EF$
$C \longrightarrow G$

A
BC
EFG



How Is It Like Development?

- Axiom is like an embryo
- Rules are like genes
- Rule postconditions are a bit like gene products
- Rule preconditions are a bit like transcription factors
- Does not model biological development particularly closely
- Developed to model growth in plants



What's the benefit?

$A \longrightarrow BC$
$B \longrightarrow EF$
$C \longrightarrow GHI$

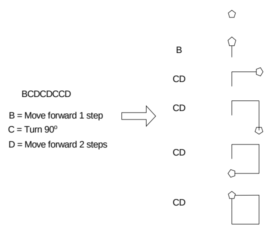
A
BC
EFGHI

$A \longrightarrow BC$
$B \longrightarrow AD$
$C \longrightarrow C$
$D \longrightarrow D$

A
BC
ADC
BCDC
ADCDC
BCDCDC

- L-Systems capture modularity
- L-Systems capture re-use
- Gene products interpreted as a program
- Great success modelling plants

Turtle Graphics



- Turtle graphics can generate structures from L-system strings
- Symbols are interpreted as instructions to move an object
- Object leaves trail behind it
- Plants modelled this way

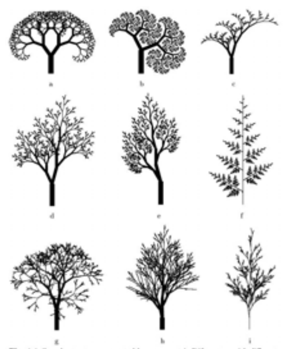


Fig. 4.4. Sample structures generated by a parametric D0L-system with different values of constants

Disadvantages

- Doesn't inherently map to sensible 2D/3D operations
  - Can be interpreted as *instructions:* growth, turtle etc.
  - If you want 2D/3D structure then other approaches might be more suitable
- The bigger problem is that they are *context-free*

Context-Free vs. Context Sensitive

- Context-free rules have only one symbol in their precondition
- All instances of the symbol found in the string are re-written
- Context-sensitive rules have more than one symbol in their precondition
- Only substrings that match the rule are rewritten
- This means that a symbol’s neighbours affect what happens to it
- i.e. the *context* that the symbols are found in alters the developmental process

Benefits of Context

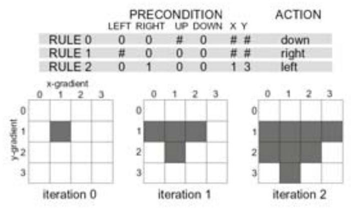
- How can context be useful?
  - Context allows more precise control over how development proceeds
  - Might be useful to use environmental cues as context
    - plants grow towards light
    - neurons are guided by chemical gradients

Cellular Models

- Similar to production rules
- Usually designed to model biology more closely
  - Use terms like genes, proteins etc. to describe rules
- Context sensitive rules
- Product of rule interaction is phenotype
  - not instructions to build it, c.f. L-Systems

- Effectively 2D or 3D context sensitive production rules

Cellular: Toy Example



	PRECONDITION						ACTION
	LEFT	RIGHT	UP	DOWN	X	Y	
RULE 0	0	0	#	0	#	#	down
RULE 1	#	0	0	0	#	#	right
RULE 2	0	1	0	0	1	3	left

- Rules applied to every filled cell
- Cells are sensitive to left, right, above, below neighbours
- In precondition 0: Absent, 1: Present, # Don't care
- Cells are sensitive to concentration of 2 chemicals, X and Y
- Rule only fires if 4 terms in precondition are true
- Rules applied in parallel

Cellular Features

- Cellular systems model Biology a lot more closely
- Rules interact and can be modelled with GRNs
- Context-sensitivity is communication between cells
- Development works by producing increasingly large and complex patterns of proteins from simple starting conditions
- Communication between cells can allow re-use over space: see following e.g.

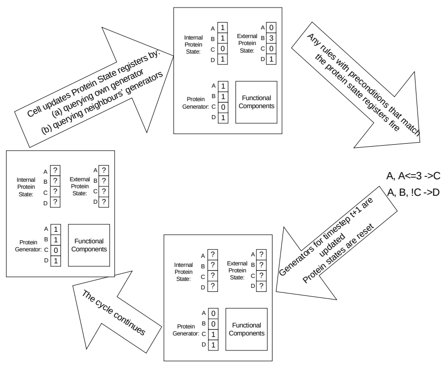
Cellular – Adder Example

- 2 D array of cells
- 2 layers to each cell: Protein layer & Architecture layer
- Development carried out for 30 timesteps
- 2 Types of rules
  - – Regulatory: Affect how proteins interact in the protein layer
  - – Architecture: Affect how the circuit develops over time

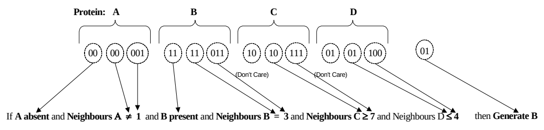
Protein Layer

- Only concerned with how proteins interact
- Each cell communicates with 4 neighbours
- 4 proteins (A,B,C,D) interact through a set of rules
- Cells generate proteins in unit concentrations
- Rule precondition specifies what proteins must be present or absent for rule activation
- Rule postcondition defines what protein is generated
- Must set simple starting conditions somewhere in the array to begin protein generation

Cell Cycle

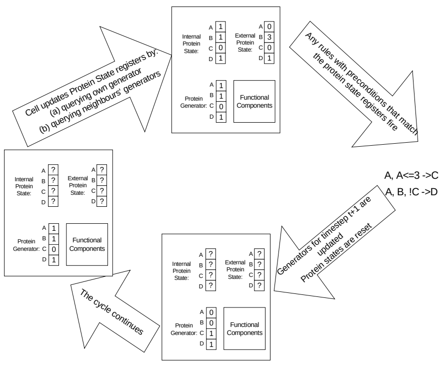

Internal Protein State:
External Protein State:
Protein Generator:
Functional Components
A
B
C
D
1
1
0
1
0
3
0
1
Any rules with preconditions that match the protein state registers fire
A, A<=3 ->C
A, B, !C ->D
Generators for timestep t+1 are updated
Protein states are reset
?
0
0
1
1
The cycle continues
1
1
0
1
Cell updates Protein State registers by:
(a) querying own generator
(b) querying neighbours' generators

Rule Structure

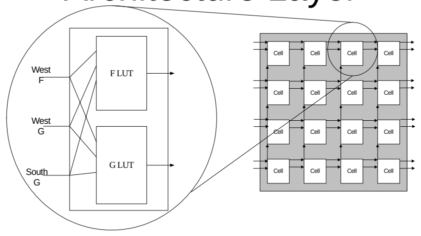

Protein: A
B
C
D
00
00
001
11
11
011
10
10
111
01
01
100
01
(Don't Care)
(Don't Care)
If **A absent** and **Neighbours A ≠ 1** and **B present** and **Neighbours B = 3** and **Neighbours C ≥ 7** and Neighbours D ≤ 4 then **Generate B**

- For each protein: 2 terms in precondition
- 1st term, tests cell's internal protein conc.: T,F,D/C
- 2nd term tests external protein conc.
- 1st 2 bits: precedence or inequality operator: !=, <=, >=, ==
- Final 3 bit: Concentration value: Value b/w 0 & 7
- Concentration values >4 can be used as don't cares, don't fires
- If all 8 terms are true, rule fires and protein is generated
- Postcondition defines which protein is generated

Cell Cycle


Internal Protein State:
External Protein State:
Protein Generator:
Functional Components
A B C D
Cell updates Protein State registers by:
(a) querying own generator
(b) querying neighbours' generators
Any rules with preconditions that match the protein state registers fire
A, A<=3 ->C
A, B, !C ->D
Generators for timestep t+1 are updated
Protein states are reset
The cycle continues

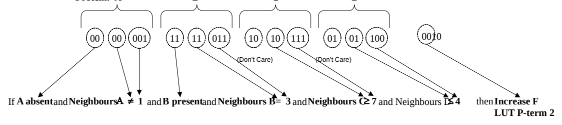
Architecture Layer



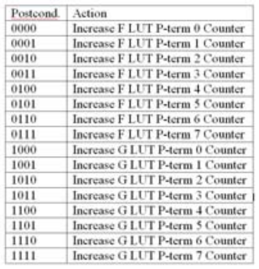
- Each cell contains 2 x 3 Input LUTs
- Fixed routing: each cell receives signals from West F, West G, South G
- LUTs have identical inputs
- Development alters only the LUTs

Architecture Rules

- Identical to protein rules except the postcondition
- Postconditions makes a change to a LUT


Protein: A
B
C
D
00 00 001
11 11 011
10 10 111
01 01 100
0010
(Don't Care)
(Don't Care)
If A absent and Neighbours A ≠ 1 and B present and Neighbours B= 3 and Neighbours C≥ 7 and Neighbours D≤ 4 then Increase F LUT P-term 2

Architecture Postconditions



Postcond.	Action
0000	Increase F LUT P-term 0 Counter
0001	Increase F LUT P-term 1 Counter
0010	Increase F LUT P-term 2 Counter
0011	Increase F LUT P-term 3 Counter
0100	Increase F LUT P-term 4 Counter
0101	Increase F LUT P-term 5 Counter
0110	Increase F LUT P-term 6 Counter
0111	Increase F LUT P-term 7 Counter
1000	Increase G LUT P-term 0 Counter
1001	Increase G LUT P-term 1 Counter
1010	Increase G LUT P-term 2 Counter
1011	Increase G LUT P-term 3 Counter
1100	Increase G LUT P-term 4 Counter
1101	Increase G LUT P-term 5 Counter
1110	Increase G LUT P-term 6 Counter
1111	Increase G LUT P-term 7 Counter

- Each cell also contains a set of *counters*
- There is 1 counter for each LUT Entry
- When an architecture rule fires it increments a counter

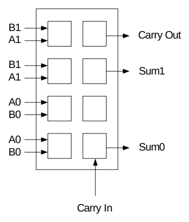
Mapping to a Circuit

- At the end of development each of the cell counters is queried
- If the counter value >= a pre-defined threshold the LUT entry is set TRUE
- Otherwise the LUT entry is set FALSE
- Architecture rules fire at different rates in different cells
- Models gradual differentiation found in biological development
- $\therefore$ LUT entries set/not set in different cells
- Evolution used to find rule set that set LUTs to form adder

Evolving Rules

- Chromosome: 20 protein rules + 14 arch rules
  $\therefore$ Length = 1048 bits
- Population 100
- Random Initialisation
- Tournament Selection 80%
- Uniform Crossover 75%
- Point Mutation: 5 Muts. per chrom.
- Generations: 2500 or optimal solution

Example: 2 Bit Adder With Carry



- Task is to evolve a 2 Bit adder *with carry*
- 5 inputs, 3 outputs
- 2x4 array of FPGA cells developed
- Set starting conditions to Cell 0: A=1, Cell 4: B=1
- Inputs and output points as shown
- Pass all possible input combinations one at a time
- Measure total number of output bits correct for each input combination
- Fitness = sum(correct output bits) MAX 96

Evolved Rules

Protein Rules:

```
1 A==7,B,B==2,C,C!=1,!D,D==0,->D
2 A>=7,B!=0,C==6,!D,D>=4,->A
3 !A,A<=4,!B,B>=4,!C,C<=4,D,D<=4,->D
4 !A,A<=2,B<=0,C,C!=6,D!=5,->B
5 A,A>=5,B,B>=3,C!=6,D==0,->C
6 !A,A<=4,!B,B!=4,C>=1,D<=0,->D
7 A>=2,B,B!=7,!C,C>=4,D==3,->A
8 A<=1,B,B>=1,C,C==7,!D,D<=4,->C
9 A==4,B,B!=0,C!=5,D>=0,->C
10 !A,A!=3,B!=1,!C,C<=7,D!=0,->C
11 A==6,B==0,C>=1,!D,D!=1,->D
12 A>=3,B==4,C==4,D==2,->A
13 A==0,B,B!=7,C!=1,D!=1,->C
14 A,A<=2,!B,B>=3,C!=5,D,D!=0,->C
15 A,A>=7,B,B<=6,C,C<=4,D>=2,->D
16 !A,A<=4,B>=3,C==5,D>=6,->A
17 A!=0,B,B>=6,C,C<=3,D<=4,->A
18 A,A<=2,B<=4,C,C>=2,D!=4,->B
19 A>=7,!B,B==1,!C,C<=2,!D,D>=1,->D
20 A!=6,!B,B>=5,C>=0,!D,D!=7,->A
```

Architecture rules:

```
1 A<=2,B,B==1,C>=5,!D,D!=0,G: 5
2 A>=3,!B,B>=5,C<=2,D!=5,F: 0
3 A!=1,B<=5,C<=3,!D,D>=0,F: 7
4 !A,A<=6,B!=4,C!=1,D<=5,G: 1
5 A!=3,B>=4,!C,C>=1,D,D!=3,G: 4
6 A==0,B!=6,!C,C!=7,D<=7,F: 0
7 A,A!=6,!B,B<=5,C,C<=7,D<=4,G: 0
8 !A,A<=4,!B,B!=3,C!=4,!D,D<=5,G: 7
9 A==0,B<=3,C,C<=1,D==0,F: 3
10 !A,A!=3,!B,B>=6,!C,C==6,!D,D!=5,F: 5
11 A,A!=6,!B,B>=6,C==2,D==0,G: 3
12 !A,A!=6,B<=0,C,C<=6,D!=2,F: 5
13 !A,A<=7,B==6,C==0,D==4,F: 2
14 A<=4,!B,B!=4,C!=7,D>=0,G: 6
```

Activated Rules

Protein Rules:

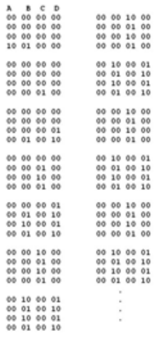
```
4 !A,A<=2,B<=0,C,C!=6,D!=5,->B
6 !A,A<=4,!B,B!=4,C>=1,D<=0,->D
10 !A,A!=3,B!=1,!C,C<=7,D!=0,->C
13 A==0,B,B!=7,C!=1,D!=1,->C
```

Architecture Rules:

```
3 A!=1,B<=5,C<=3,!D,D>=0,F: 7
4 !A,A<=6,B!=4,C!=1,D<=5,G: 1
6 A==0,B!=6,!C,C!=7,D<=7,F: 0
8 !A,A<=4,!B,B!=3,C!=4,!D,D<=5,G: 7
9 A==0,B<=3,C,C<=1,D==0,F: 3
12 !A,A!=6,B<=0,C,C<=6,D!=2,F: 5
14 A<=4,!B,B!=4,C!=7,D>=0,G: 6
```

- 4 protein rules have formed a GRN
- Activate different architecture rules in different cells

Protein Development

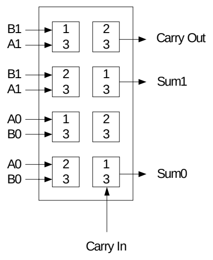


```
A  B  C  D
00 00 00 00     00 00 10 00
00 00 00 00     00 00 01 00
00 00 00 00     00 00 10 00
10 01 00 00     00 00 01 00

00 00 00 00     00 10 00 01
00 00 00 00     00 01 00 10
00 00 00 00     00 10 00 01
00 00 01 00     00 01 00 10

00 00 00 00     00 00 10 00
00 00 00 00     00 00 01 00
00 00 00 01     00 00 10 00
00 01 00 10     00 00 01 00

00 00 00 00     00 10 00 01
00 00 01 00     00 01 00 10
00 00 10 00     00 10 00 01
00 00 01 00     00 01 00 10

00 00 00 01     00 00 10 00
00 01 00 10     00 00 01 00
00 10 00 01     00 00 10 00
00 01 00 10     00 00 01 00

00 00 10 00     00 10 00 01
00 00 01 00     00 01 00 10
00 00 10 00     00 10 00 01
00 00 01 00     00 01 00 10
                      .
00 10 00 01           .
00 01 00 10           .
00 10 00 01           .
00 01 00 10
```

- Initial conditions:
  - Cell 0, A=1
  - Cell 4, B=1
- After initial growth, C and D alternate between 0 and chequed patterns
- Caused by:
  6. C>=1->D, 10. D!=0->C
- B has same pattern as C one step later
- Caused by:
  4. C->B

LUT Development

- Chequed patterns of proteins used by architecture rules
- Chequed pattern of F-LUTs generated
- Uniform Pattern of G-LUTs generated



Where:

WF	WG	SG	1=	2=	3=
0	0	0	1	1	0
0	0	1	0	0	1
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	1	1	1

Re-use and Large Phenotypes

- Development provides a mechanism for design re-use
- Allows large phenotypes to be generated
- This seems to help with scalability
  - Large (in evolutionary terms) adders have been evolved this way: 7 bit + Carry, on 15x2 array, routing architecture developed also