Computational Evolution & Digital Organisms

A look at a subset of Artificial Life

Computational Evolution

- Attempts to elucidate principles of evolution
  - Builds models of self-replicating organisms
    - Computational cost limits physical fidelity of the model.
    - Digital or chemical models
  - Mutation creates variation in populations
  - Reproduction can be sexual or asexual
  - Ability to (out) reproduce its genome is the usual fitness measure
    - For some research, other fitness measures are used.

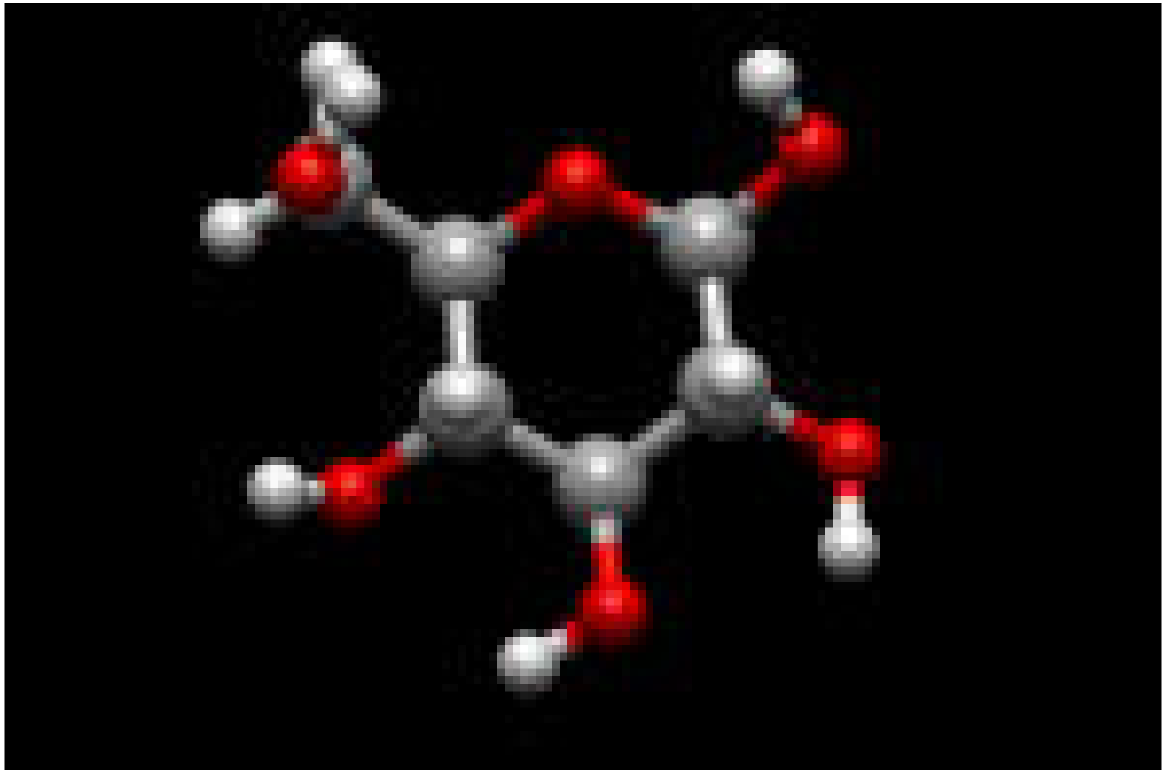
Not to be Confused With Evolutionary Computing

- A Search Technique inspired by biology
  - Points in search space represented as “genomes”
  - Crossover produces new points in search space
  - Mutation ensures variety
    - Ensures more of search space is sampled
  - Fitness function determines which subset of population become progenitors
  - Larger populations increase coverage of space.
  - Search usually walks through “invalid” points

Overview of Talk

- Motivation: The complexity of cellular life
- Tierra and the evolution of digital organisms
- Avida and other Tierra inspired work
- Lessons/Future Research

Complexity of Cellular Life I: Metabolism of Glucose to produce ATP



Molecules of glucose pathway.

- PDB Molecule of the Month The Glycolytic Enzymes

It's all chicken and egg

- Where did glucose come from?
- Where did all those intermediate products come from?
- Where did all those wonderful enzymes come from?
- Take away any of the enzymes, and the system collapses.

Complexity of Regulatory Mechanisms

Nature made this from

- Molecules with differential binding affinities for DNA.
- Overlapping control regions.
- Positive and negative feedback.
- Cooperative binding.
- How did it make the recipe?

Tierra, a Platform for Digital Evolution

- Design Requirements/Inventions:
  - Organisms must be self-reproductive
  - Ability to out-reproduce the competition only fitness criteria
    - Avoids “artificial” fitness functions.
  - Control (jumps/calls) is effected through *templates* and *targets*, which are complementary “bit strings”
    - Jump nop1 nop0 nop1 goes to nop0 nop1 nop0
  - Organisms sense the environment
    - Dynamic “fitness” function

Tierra’s Digital Organisms

- Each organism (cpu) has
  - 4 registers (A, B, C, D)
  - Instruction pointer
  - 10 word stack
- Time slicing “implements” parallel organisms
- When space for new organisms is needed, the oldest organisms are reaped (as a rule).

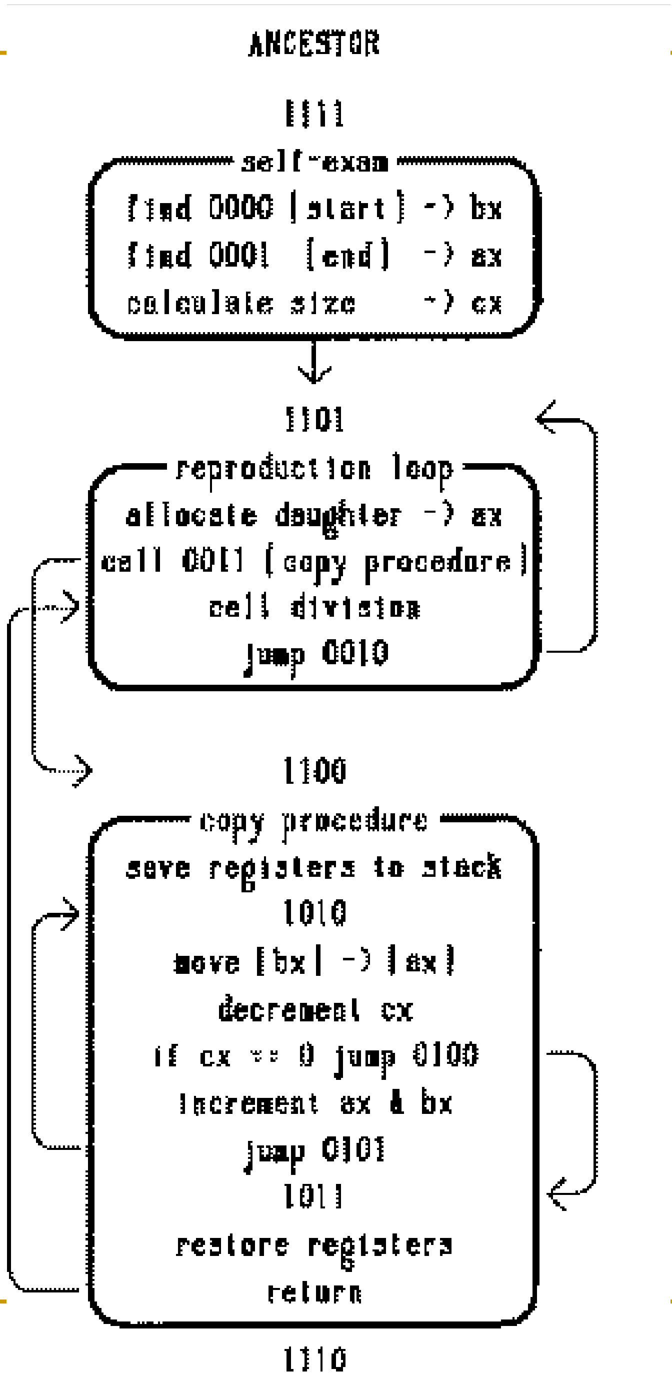
Tierra's Instruction Set

- Data Movement
  - PushA, PopA, PushB, PopB, etc for C and D
  - MOVDC (D <- C), MOVBA, COPY ([A] to [B])
- Control
  - JumpO, JumpB, Call, Ret, IfZ, nop0, nop1
- Calculation
  - subcab, subaac, inca, incb, decc, incd, zero, not, shl
- Biological and Sensing
  - adr, adrb, adrF, mal (allocate memory), divide

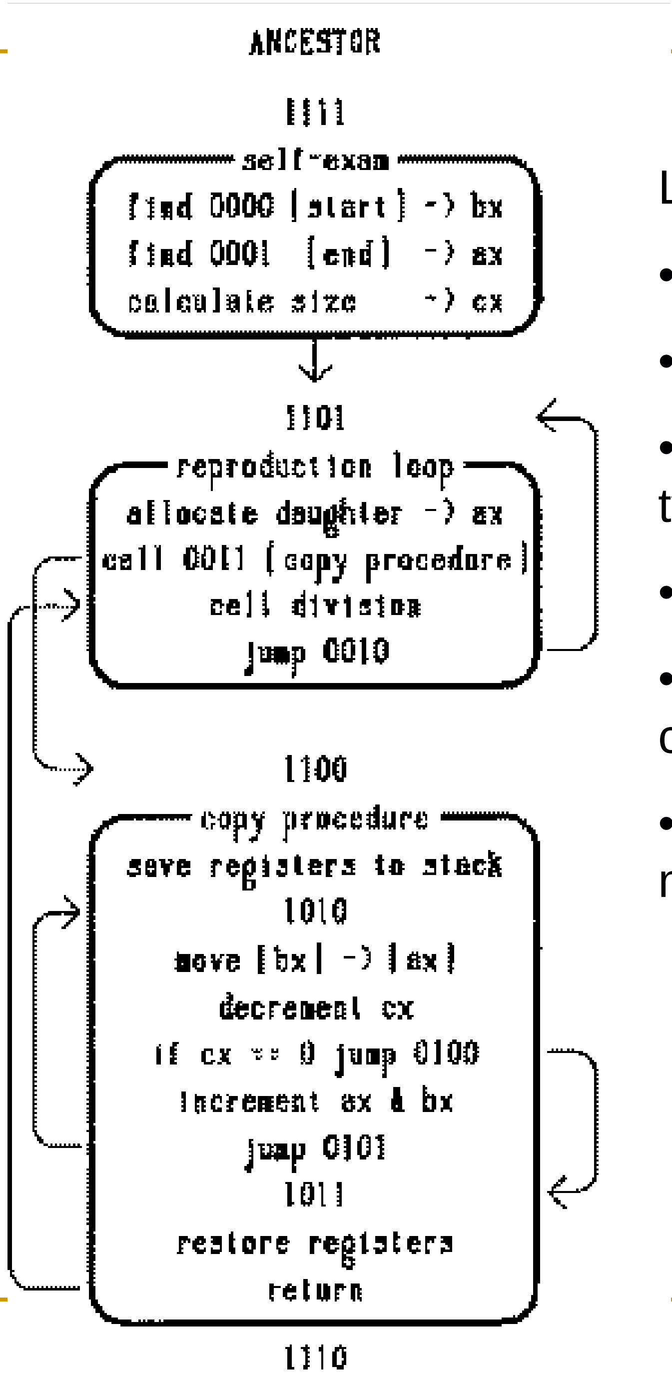
Mutational Sources

- A copy error every X copy instructions
- Cosmic rays
  - A bit in the soup gets flipped every Y instructions
  - Works because no cells are autosomes
  - Biased, not random
- Probabilistic results of instructions
  - Every so often an instruction misfires
  - E.g., incA adds 2
- No Insertion/deletions

The Tierran Ancestor



The Tierran Ancestor

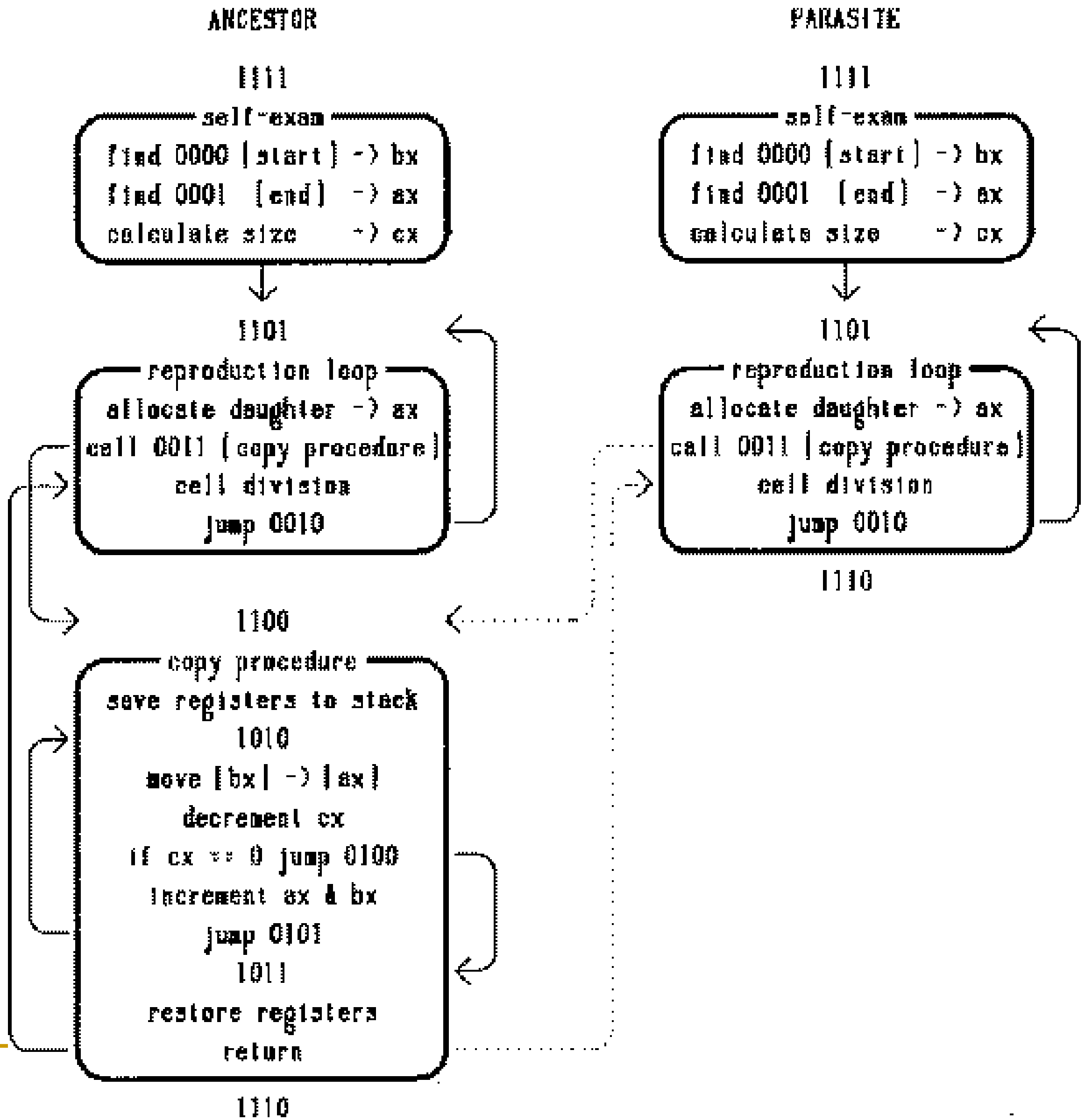


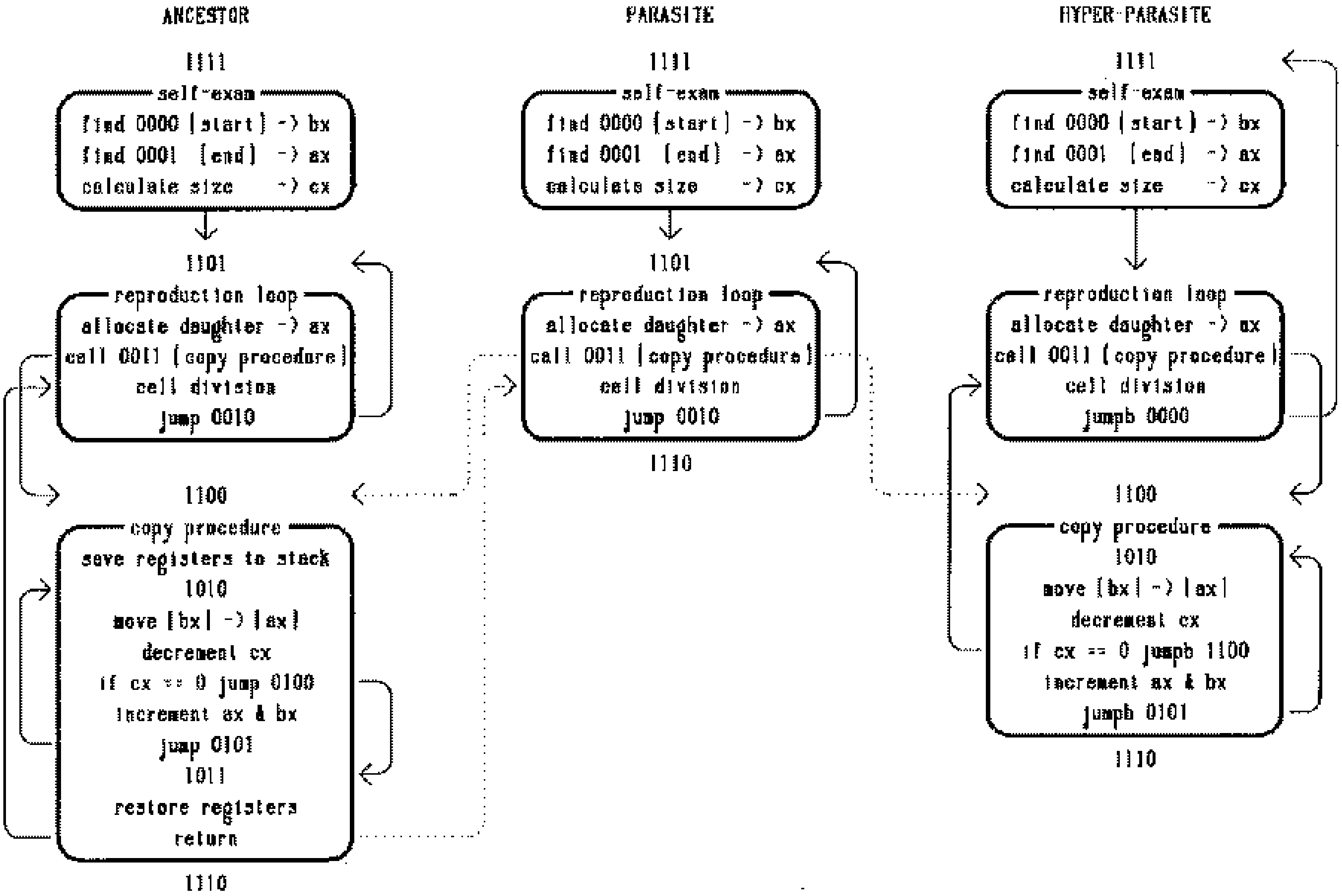
Lots of redundancy

- Labels can be shortened
- Different control constructs
- Cells only replicate once or twice
- Templates can be labels
- Various return addresses can be used
- Control can use any matching code

Ancestor Code

Evolution in Action: Parasites!!





ANCESTOR
1111
self-exam
find 0000 [start] -> bx
find 0001 [end] -> ax
calculate size -> cx
1101
reproduction loop
allocate daughter -> ax
call 0011 [copy procedure]
cell division
jump 0010
1100
copy procedure
save registers to stack
1010
move [bx] -> [ax]
decrement cx
if cx == 0 jump 0100
increment ax & bx
jump 0101
1011
restore registers
return
1110
PARASITE
1111
self-exam
find 0000 [start] -> bx
find 0001 [end] -> ax
calculate size -> cx
1101
reproduction loop
allocate daughter -> ax
call 0011 [copy procedure]
cell division
jump 0010
1110
HYPER-PARASITE
1111
self-exam
find 0000 [start] -> bx
find 0001 [end] -> ax
calculate size -> cx
reproduction loop
allocate daughter -> ax
call 0011 [copy procedure]
cell division
jumpb 0000
1100
copy procedure
1010
move [bx] -> [ax]
decrement cx
if cx == 0 jumpb 1100
increment ax & bx
jumpb 0101
1110

An interesting chicken-and-egg mutation

- <C = size, B=@self>
- nop1 nop1 nop0 nop1
- mal
- call nop0 nop0 nop1 nop1
- divide
- jump nopo nop0 nop1 nop0
- ifz
- nop1 nop1 nop0 nop0
- <copy loop>

An interesting chicken-and-egg mutation

- <C = size, B=@self>
- nop1 nop1 nop0 nop1
- mal
- call nop0 nop0 nop1 nop1
- divide
- pushb (was jump) nopo nop0 nop1 nop0
- ifz
- ret (was nop1) nop1 nop0 nop0
- <copy loop>

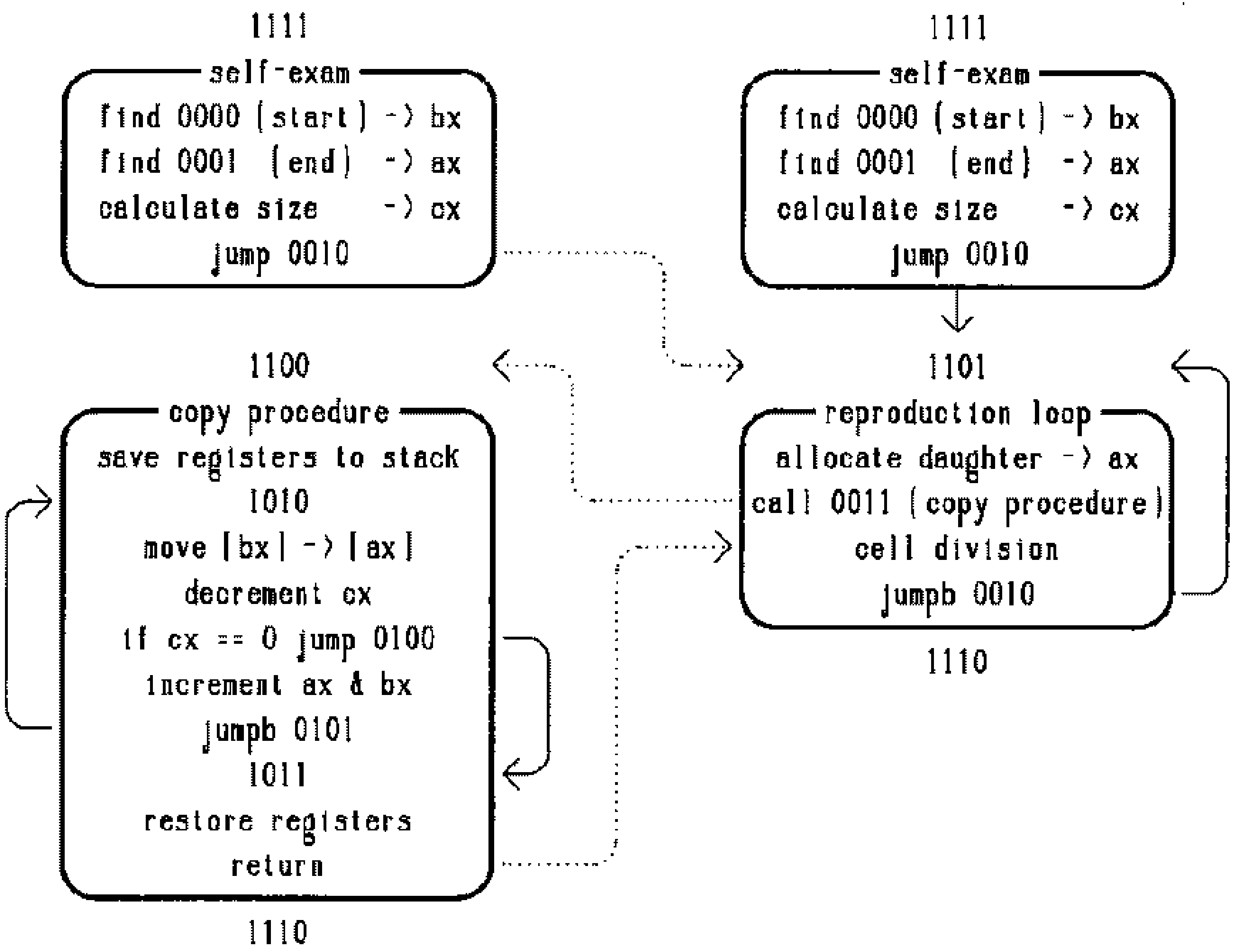
A Copy-Once Parasite

- Stays just ahead of the reaper
  - nop1 nop1 zero not0 shl shl movdc
  - adrb nop0 nop0 pushc nop0
  - subaac
  - movba pushd nop0
  - adr nop0 nop1
  - inca
  - subcab pusha nop1 pushd nop1
  - mal
  - call nop0 nop0 nop1 nop0
  - divide

Two chances to find a copy loop

- <C = size, B = @self>
- mal pusha call movii pusha
- call nop0 nop0 nop1 nop1
- divide movii
- pusha
- mal
- call nop0 nop0 nop1 nop1
- divide mal subaac nop1
- ret zero nop1 zero (jumps to start of daughter)
- nop1 nop1 nop1 nop0

SYMBIONTS



1111
self-exam
find 0000 (start) -> bx
find 0001 (end) -> ax
calculate size -> cx
jump 0010
1111
self-exam
find 0000 (start) -> bx
find 0001 (end) -> ax
calculate size -> cx
jump 0010
1100
copy procedure
save registers to stack
1010
move [bx] -> [ax]
decrement cx
if cx == 0 jump 0100
increment ax & bx
jumpb 0101
1011
restore registers
return
1110
1101
reproduction loop
allocate daughter -> ax
call 0011 (copy procedure)
cell division
jumpb 0010
1110

Feature or Bug?
CPU is indepdent of genome

- A very small self-replicating parasite (15 long)
  - Nop1
  - Adrb nop0
  - MovBA
  - Adrf nop0 nop0
  - subAAC
  - Jump nop0 nop0 nop1 nop0
  - Nop1 nop1
- Even smaller viable program:
  - Nop1

Feature or Bug? Non-local effects

- A template can match *any* nearby target
- A request for memory can kill *any* organism, even one "fitter"
- A daughter cell can be placed *anywhere*
- Allocating a large amount of memory for a daughter can kill tens of organisms, creating a dieoff

Feature or Bug?
Spaghetti Code is a Frequent Occurrence

- Symbionts arise quite frequently
- When a target is mutated, the target in another cell is used.

Bug or Feature?
Parasites require necrophilia

- Instructions are left in memory when an organism is reaped.
- “Parasites” keep using these instructions.

Bug or Feature? Sloppy replicators instead of Indels

- Teirra lacks insertion/deletion mutations
  - Biology uses indels
  - Harder to remove instructions without deletions
  - Harder to make room for new instructions
- Tierra makes up for it with sloppy replicators that move instructions around willy nilly
  - Buy maybe this is needed anyway?

Is Sloppiness needed to Bootstrap Complexity?

- Sloppiness (ad-hoc) mixing gave us
  - Mitochondia (ingestion without digestion)
  - Cloroplasts in bacteria (same story)
  - Gene mixing (via viruses)
  - Diploidy from Haploidy

Avida

- Inspired by Tierra, but
  - Controlled instruction pointers (less slopiness)
  - Insertion/Deletion mutations
  - 2 dimensional grid of organisms, not instructions
  - Only local next-neighbor effects
  - Fitness functions to augment reproduction
- Experiments to test biological theories
  - Evolution of Complexity
  - Evolution of Complex Functions
  - Relationship among evolution rate and landscape

Digital Biosphere

- Inspired by Tierra/Avida but
  - Focus is on evolutionary trajectories.
    - Are there principles regarding these trajectories?
  - Will exploit the constraints of physics
    - Conservation Laws!
    - Energy requirements and metabolism
  - Will eventually move to chemical modeling to get closer to biology.

Lessons

- Evolution finds corners of the search space
  - If you build it, they will exploit it
  - Complexity comes from exploiting environment
- Co-evolution makes the problem interesting and different
  - Changing fitness functions
- Designing a system for open-ended evolution is still very much an open-ended problem.

What's it all mean?

We have a source of new insights

- Watching evolving dynamical systems give insight and ideas.
- Biologists aren't trained to do this.
- Many insights will be gained that will eventually transfer over to biological thinking

Last Thought

- Is the complexity of the phage lambda lyse/lytic growth mechansim any more or less complex than the programs that Tierra was evolving?