Bootstrapping an OODL

Jonathan Bachrach
MIT AI Lab
So You Want to Write a New OODL

• But, your language G doesn’t exist to write it in?
• Of course it doesn’t, but why would you want to write G in G anyways?
  – Because it’s a superior language
  – To Grove it’s a capable language
  – Need a G runtime anyways
Start with Interpreter in D

• Write an interpreter for G in D

• Advantages
  – D has a complete IDE
  – D is a powerful language

• Disadvantages
  – It will only run on machines that D runs on
  – It will never be faster than an interpreter
How about the runtime and libraries?

- Start by writing most of runtime and support libraries in D
  - Leverage D as much as possible
Stuck in D-land

• Write a cross compiler G2C_D
  – But still have most of G system written in D

• Rewrite G runtime & interpreter in C and G
  – Can’t assume anything cause it’s C
  – Build all runtime objects by hand
  – Objects must be constructed in order
G2C_D

• Still want to write G2C in D because of
  – Speed debuggability and interactivity
• Will have an entire G system in D including
  – Runtime
  – Object System
  – Libraries
  – Interpreter
  – G2C
G2C__G

- Finally port G2C to G
- Ensure that
  - $G2C\_G.G \Rightarrow G2C\_D \Rightarrow G2C\_G$
  - $G2C\_G.G \Rightarrow G2C\_G \Rightarrow G2C\_G'$
  - $G2C\_G.G \Rightarrow G2C\_G' \Rightarrow G2C\_G''$
  - ...
Break Even Point

- G is now
  - Free of D
  - Powerful enough to write a G compiler in it
- You can now write new versions of G in G
- You have reached the break-even point
  - congratulations
Standard MFTL Putdown

• “Has it been used for anything besides its own compiler?”*
  – On the other hand, a language that cannot even be used to write its own compiler is beneath contempt.*

• *From hacker’s dictionary
Bootstrapping

- Boot from Dylan
- Boot steady State
Native Boot Steady State

• Goals
  – Simple
  – Reduce throw away code

• Purely dynamic boot
  – no reliance on compiler -- sequential execution
  – Macros
    • Keep object definitions looking as they do
    • Define mappers that extract needed info
  – Ordering
    • Slowly build up world
    • Finally original code can get pushed through
Boot Order

- Define boot objects in macro
- Build empty prototype objects
- Setup `<list>` basics
- Setup hierarchy
- Define tagged and boxed objects
- Make slots and accessors
- Finalize slots
- Patch instances
- Define repeated objects
- prepare for functions
- Define functions
- Patch early generics
- Define object system