Some ideas for the Future of Cabal

Duncan Coutts

Some ideas for the Future of Cabal

- A language for build systems
- Constraint solving problems in package deployment

Build systems

- Everybody has one
- Few people are satisfied

- Classic make
- GNU make
- SCons
- OMake
- CMake
- Ant
- Vesta
- Maak
- ... 100's more

What make does right

- Tracks dependencies
  - if you write correct rules
  - If you do not use recursive make
- Parallel builds
- Incremental rebuilds

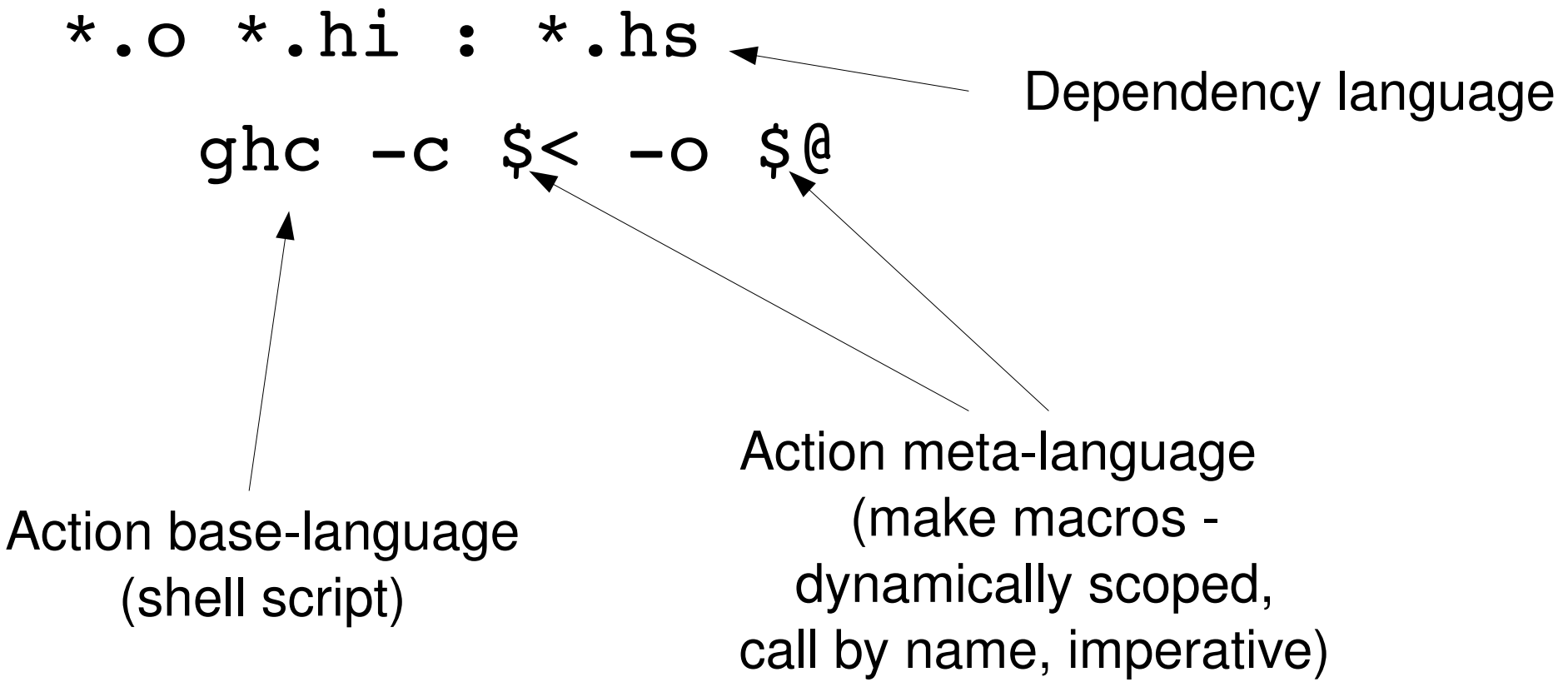
What is wrong with make

- Easy to write incorrect rules:

```
foo : bar
    cat bar baz > foo
```

What is wrong with make

- Horrible hodgepodge of languages



What is wrong with make

- Not expressive enough for dynamic dependencies

```
*.o *.hi : *.hs $(depsOf $<)
    ghc -c $< -o $@

where depsOf hs = ...
```

Dynamic dependencies are everywhere

- Environment
- Configuration
  - Not all dependencies are files

Generating Makefiles does not work

- Many tools generate Makefiles
  - Analyse environment, configuration and generate dependencies
- Preprocessors and code generators break this model
  - requires interleaving dependency discovery and running build actions

Existing research

- Classic “recursive make considered harmful”
  - Lesson: track dependencies precisely
- Vesta (PLDI '00), Maak (SCM '03)
  - Functional build description languages
  - Tracks dependencies precisely

If you have to `make clean`

your build system is broken!

Properties of a better system

- Track all dependencies precisely
  - Preferably correct by construction
- Interleave dependency discovery and reduction
- Correct
  - for some value of correct
- Expressive
  - To describe tricky build systems

Properties of a better system

- Make it impossible to specify untracked dependencies
  - Specify dependencies and action together
  - Small set of correct primitives

    `readFile, writeFile, statFile`
  - Combinators for building bigger actions

Idea for a build DSL

- DSL Embedded in Haskell

```
data Build m a

build :: Build IO a → IO a
```

- Parametrise over underlying monad

```
test :: Build FakeIO a → Trace
```

  - Pure instance can be checked with QuickCheck
  - Correctness properties expressed in terms of traces

Idea for a build DSL

- Applicative functor combinator for “static” composition of dependency graphs

```
(<*>) :: m (a → b) → m a → m b
```

  - Can see the graph structure of both sides statically
  - Opportunity for parallelism and incremental rebuilds

Idea for a build DSL

- Monadic bind combinator for “dynamic” composition of dependency graphs

```
(=<<) :: (a → m b) → m a → m b
```

  - Right hand graph has to be reduced to a value before we can see the shape of the left hand graph
  - Expresses dynamic dependencies
- Should not be over-used
  - Linear chains in the dependency graph prevent incremental and parallel rebuilds

Some ideas for the Future of Cabal

- A language for build systems
- Constraint solving problems in package deployment

Package Deployment

- `apt-get xmonad`
  - Install xmonad and all of its dependencies
- Database of packages
  - Packages identified by name and version

    `xmonad-0.8`
  - Dependencies on other packages by name and version range
    - ==, <, >=, _ && _, _ || _
  - e.g. `xmonad-0.8` depends on `X11 >= 1.4.1`

The package problem

- Given
  - A target (eg `xmonad`)
  - A set of packages (eg debian or hackage repo)
- Find a solution
  - A graph of exact package versions where each package name appears at most once
  - Satisfying all dependency version predicates

The package problem

- Example
  - xmonad-0.4 build-depends: X11 >= 1.2.1
  - xmonad-0.8 build-depends: X11 >= 1.4.1
  - X11-1.4.1
  - X11-1.4.2
- Solution
  - pick xmonad-0.8 and X11-1.4.2

The package problem

- In principle very hard
  - NP-complete (proof from CNF-SAT [pdf])
  - Someone made the point by encoding Sudoku using debian repositories and `apt-get`
- Usually easy
  - Large but easy instances of hard problem
  - Different versions of the same package usually have similar dependencies

The package problem

- Some solutions are better than others
  - Sometimes prefer using already installed packages
  - Sometimes prefer the latest version
  - General soft preferences “parsec < 3”
- Also require good error messages
  - People want to know what to fix

Current Cabal approach

- Constraint solving based approach
  - Written by someone who doesn't know how to write constraint solvers
- No backtracking, incomplete solver
  - But guarantees polynomial time
- Heuristics for the ordering of choices
- Almost good enough in practise

What we want in a solver

- Easily express package constraints
  - We also have conditional dependencies
- Handle soft constraints / preferences
- Find solutions more often
- Produce comprehensible errors
- Fast enough
  - On repositories of a few 1,000 packages

A more principled approach?

- Can we apply standard solver techniques?
  - SAT
  - CP
- You could write a paper about a good solution!
- See also:
  - http://www.edos-project.org/
  - openSUSE 11 uses a SAT solver