Maintaining Verified Software

Joe Leslie-Hurd

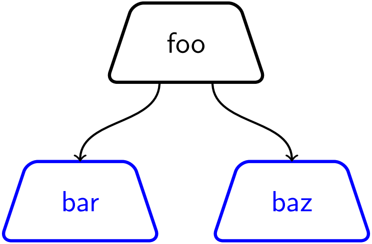
Intel Corp.
joe.leslie-hurd@intel.com

Haskell Symposium
23 September 2013

Talk Plan

Code Reuse Using Haskell Packages

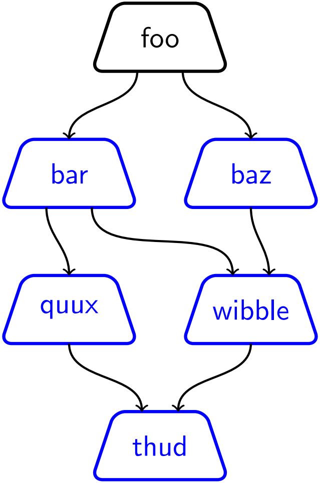
- The Haskell language and platform conspire to make it easy for developers to build on the work of others.
- **Example:** Consider a Haskell package `foo` that pulls in useful functionality from packages `bar` and `baz`:



- **Warning!** The behaviour (and thus correctness) of `foo` depends on the behaviour of `bar` and `baz`.

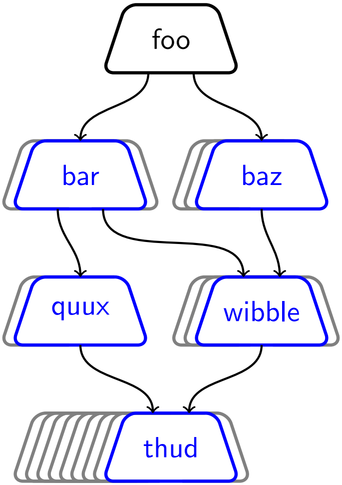
Package Dependency Graphs

- Packages `bar` and `baz` may also depend on other packages.
- Recursively expand these dependencies to construct the package dependency graph.
- Correctness of `foo` depends on the behaviour of every reachable package.



Evolving Package Dependencies

- New versions of packages are constantly being released.
- Package `foo` has no direct control over which version of `quux` it is built upon.
- Correctness of `foo` may depend on the behaviour of future versions of reachable packages.
- You are now in Haskell dependency hell.



OpenTheory Project

- The OpenTheory project aims to bring the benefits of software engineering to theorem proving.[1]
- OpenTheory logical theory packages offer an alternative to Haskell dependency hell:
  1. Formally verified Haskell packages can be automatically synthesized from OpenTheory logical theory packages.
  2. Haskell package dependencies can be automatically synthesized by reasoning on logical theory packages.
- **This Talk:** We will present this technique in two parts:
  1. Checking dependencies between logical theories.
  2. Instantiating to formally verified Haskell packages.

[1] *Theory engineering*, or *"proving in the large."*

Talk Plan

1. Haskell Package Dependencies
2. Logical Theory Dependencies
3. Verified Haskell Packages
4. Summary

Logical Theory Packages

- A theory $\Gamma \triangleright \Delta$ of higher order logic consists of:
  1. A set Γ of assumptions.
  2. A set Δ of theorems.
  3. A formal proof that the theorems in Δ logically derive from the assumptions in Γ.
- The OpenTheory standard package format for higher order logic theories allows us to:
  - Liberate theories from the theorem proving system in which they were created.
  - Compose theories from different origins.
  - Process theories with a diverse array of tools.

Proof Articles

```
# TINY EXAMPLE ARTICLE
#
# Construct the hypothesis list
nil
# Construct the conclusion term
"T"
const
"bool"
typeOp
nil
opType
constTerm
1
def
# Import an assumption: ⊢ T
axiom
# Export a theorem: ⊢ T
nil
1
remove
thm
```

- Higher order logic proofs are encoded as standard article files.
- Articles are executed by a stack-based virtual machine.
- Articles can import assumptions Γ and export theorems Δ.
- The result is a theory $\Gamma \rhd \Delta$.

Theory (Tiny example result)

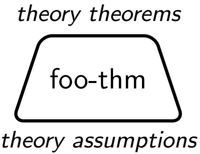
```
1 external type operator: bool
1 external constant: T
1 assumption:
  ⊢ T
1 theorem:
  ⊢ T
```

Basic Theory Packages

- A basic theory package wraps a proof article with some meta-data.
- We depict theory packages $\Gamma \triangleright \Delta$ as named proof boxes that build up from an assumption set Γ to a theorem set Δ.

Theory (Basic theory package)

```
name: foo-thm
version: 1.0
author: Joe Leslie-Hurd <joe@gilith.com>

main {
  article: "foo-thm.art"
}
```

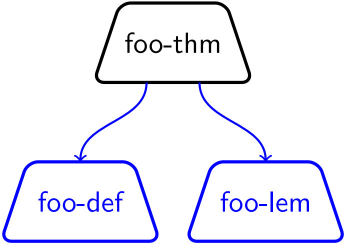


Required Theory Packages

- Theorems of required theories listed in a package must collectively satisfy all theory assumptions.
- In this way we can specify and check logical dependencies between a collection of theory packages.

Theory (Required theories)

```
name: foo-thm
version: 1.0
author: JLH <joe@gilith.com>
requires: foo-def
requires: foo-lem

main {
  article: "foo-thm.art"
}
```

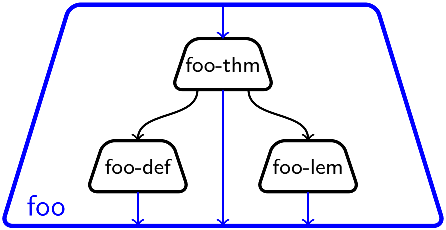


Nested Theory Packages

Theory (Nested theories)

```
name: foo
version: 1.0
author: JLH <joe@gilith.com>

def {
  package: foo-def-1.0
}

lem {
  package: foo-lem-1.0
}

thm {
  import: def
  import: lem
  package: foo-thm-1.0
}

main {
  import: thm
}
```

- Theory packages can contain nested theories.
- Proofs of nested theories are replayed (with optional renaming of symbols).



Building Logical Theories

- Importing a theory $\Gamma \rhd \Delta$ into a theorem set Θ means:
  1. Grounding all external symbols with defined symbols in Θ
     $\leadsto$ results in a substitution σ
  2. Satisfying all assumptions $\Gamma[\sigma]$ with theorems in Θ
     $\leadsto$ results in a theorem set $\Delta[\sigma]$
- Building a theory package $\Gamma \rhd \Delta$ means proving all of its theorems *'from scratch'*:
  1. Recursively build every required theory package $\Gamma_i \rhd \Delta_i$
     $\leadsto$ results in a theorem set Θ_i
  2. Import the theory $\Gamma \rhd \Delta$ into $\bigcup_i \Theta_i$
     $\leadsto$ results in a theorem set $\Delta[\sigma]$

What Can Go Wrong?

- **Circular Reasoning:** Theory package dependency graphs must not contain any loops!
  - Theory packages are representations of proofs, which are directed acyclic graphs.
- **Inconsistent Definitions:** The same symbol name must not be defined in multiple required theory packages.
  - **Example:** The two theories

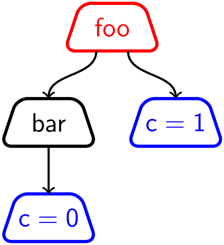
    $$\emptyset \rhd \{\vdash \mathsf{c} = 0\} \qquad \text{and} \qquad \emptyset \rhd \{\vdash \mathsf{c} = 1\}$$

    are individually fine, but must never be required by the same theory package.

Theory Dependency Checking

A theory dependency graph is up-to-date if the following pass:

- **Global Checks of the Theory Graph**
  1. No cycles.
  2. Definitional consistency.
- **Local Checks of Required Theories**
  1. No unsatisfied assumptions.
  2. No ungrounded external symbols.



Spoiler Alert! Cabal package selection will take care of everything except no unsatisfied assumptions.

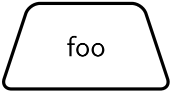
Incremental Theory Dependency Checking

- There is an efficient incremental algorithm for local dependency checking of a theory package:
  1. Initialize by carrying out local dependency checking with the latest versions of the required theory packages.
  2. Suppose for each required theory package we have found a version range such that every version selection is guaranteed to pass local dependency checking.
  3. Efficiently test whether adding an earlier version of a required theory package will preserve local dependency checking.
- In this way we can automatically compute maximal version ranges of required theory packages.

Talk Plan

The Logical Theory of a Haskell Package

defined symbols
satisfying properties



external symbols
satisfying assumptions

- **Explicit:** Symbols and their types.
  - Build tools can use them to automatically match dependencies.
  - Explains propensity of Haskellers to encode all properties in types.
- **Implicit:** All other properties.
  - Invisible to build tools.
  - Some properties can be encoded as tests (assertions/QuickCheck).
  - Package assumptions must be encoded as version ranges.

Idea: Automatically match dependencies between formally verified Haskell packages where all properties are explicit.

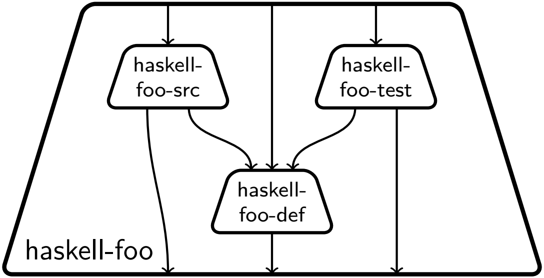
Formal Verification of Haskell Packages

- There is a well-known correspondence between higher order logic functions and a pure subset of the Haskell language.[2]
- **Developing Formally Verified Haskell:**
  1. Manually define type operators and constants in higher order logic, and prove properties of them.
  2. Automatically synthesize Haskell from these properties using a shallow embedding.
- The synthesis tool operates at the package level:
  - [OpenTheory package] $\mapsto$ [Haskell package]
- **Important:** The theory dependencies of the OpenTheory package must faithfully model the Haskell package.

[2] Haftmann, *From Higher-Order Logic to Haskell*, PEPM 2010.

Inside the OpenTheory Package

Consider the OpenTheory package `haskell-foo`:



1. `def`: Defining 'Haskell' symbols in terms of higher order logic.
2. `src`: Deriving computational forms for the Haskell symbols.
3. `test`: Deriving executable properties of the Haskell symbols.

Example OpenTheory Package: `haskell-prime`

1. `haskell-prime-def`: Define the Haskell symbols:

   $$\vdash \text{H.Prime.all} = \text{Prime.all}$$

   Defining new symbols ensures theory dependencies will be traced back to this package.

2. `haskell-prime-src`: Derive computational forms:

   $$\vdash \text{H.Prime.all} = \text{H.unfold H.Prime.next H.Prime.initial}$$

   These proofs depend on theories of all Haskell symbols that appear in the computational forms.

3. `haskell-prime-test`: Derive executable properties:

   $$\vdash \text{H.nth H.Prime.all } 0 \neq 0$$

Automatically Synthesizing a Haskell Package

1. The Haskell source code is generated by pretty-printing the computational forms in the `src` nested package:

```
all :: [OpenTheory.Natural]
all = OpenTheory.unfold next initial
```

2. A QuickCheck test suite is generated from the executable properties in the `test` nested package:

```
assertion0 :: Bool
assertion0 = not (OpenTheory.nth all 0 == 0)
```

3. *Most* of the Haskell package meta-data is derived from the OpenTheory package meta-data:

```
name: opentheory-prime
version: 1.25
```

Verified Haskell Package Dependencies

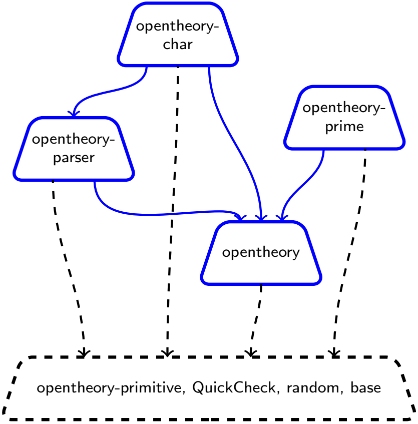
- **Problem:** Even a verified Haskell package will not work correctly in a bad environment.
- **Key Idea:** Check verified software dependencies by formal reasoning on logical theories.
- Cabal package selection already takes care of the necessary global dependency checks.
- Use logical theories to generate version ranges of required packages that satisfy local dependency checks.
- **Solution:** Call the incremental algorithm for theory dependency checking to automatically synthesize the Haskell package `build-depends` meta-data.

Verified Haskell Package Examples

- The synthesis scheme was tested on some example packages.
- They are all available on Hackage.

Code (opentheory-prime)

```
build-depends:
  base >= 4.0 && < 5.0,
  random >= 1.0.1.1 && < 2.0,
  QuickCheck
    >= 2.4.0.1 && < 3.0,
  opentheory-primitive
    >= 1.0 && < 2.0,
  opentheory >= 1.73 && <= 1.74
```



Packaging Verified Software

Using theory packages for verified software addresses many of the logistical needs:

- **Distribution:** Download software from repos, check the proofs, and install on your local machine.
- **Versioning:** Developers can release new versions of software, obsolete packages can be marked.
- **Upgrade:** Can statically guarantee that an upgrade will be safe, so long as the required properties still hold of the new version.

Theory Repository

Gilith OpenTheory Repo ×

opentheory.gilith.com

Gilith OpenTheory Repo

packages • recent • upload

Log in

Welcome to the Gilith OpenTheory repo, which is currently storing 35 theory packages. Each theory package contains a collection of theorems together with their proofs. The proofs have been broken down into the primitive inferences of higher order logic, allowing them to be checked by computer.

This web interface is provided to help browse through the available packages, but the recommended way of downloading and processing theory packages is to use the opentheory package management tool. For more information on OpenTheory please refer to the project homepage.

Recently Uploaded Packages [more]

haskell-prime-1.25 — Prime numbers
Uploaded 7 weeks ago by Joe Leslie-Hurd

haskell-char-1.43 — Unicode characters
Uploaded 7 weeks ago by Joe Leslie-Hurd

haskell-parser-1.119 — Stream parsers
Uploaded 7 weeks ago by Joe Leslie-Hurd

OpenTheory twitter feed:

- haskell-prime-1.25 uploaded by Joe Leslie-Hurd *http://t.co/y6QZ0MOJ* *50 days ago*
- haskell-char-1.43 uploaded by Joe Leslie-Hurd *http://t.co/ohnibphF* *50 days ago*
- haskell-parser-1.119 uploaded by Joe Leslie-Hurd *http://t.co/il0glGWP* *50 days ago*

Gilith OpenTheory Repo, maintained by Joe Leslie-Hurd. ✓

Talk Plan

1. Haskell Package Dependencies
2. Logical Theory Dependencies
3. Verified Haskell Packages
4. Summary

Summary

- This talk demonstrated how to perform verified software dependency checking by formal reasoning on logical theories.[3]
- The Haskell instantiation of this technique was greatly simplified by the language and platform.
- One obstacle was the absence of a built-in `Natural` type of infinite precision non-negative integers—could this be added?

[3] *"Bringing the benefits of logical theories back to software engineering!"*

Any Questions?

joe.leslie-hurd@intel.com   @gilith

gilith.com/research/opentheory

Ungrounded External Symbols

- Consider the theory package `divides-def`:

$$\Gamma \rhd \{ \vdash \forall\, m, n.\ \mathrm{divides}\ m\ n \iff \exists\, k.\ k * m = n \}$$

- The external constant $*$ appears in the theorem but not in the assumptions Γ.
- There's no logical problem because no properties of $*$ are assumed in this theory.
- But during theory import all external symbols must be grounded to defined ones.
- To prevent $*$ from being an ungrounded symbol, it must appear in the theorems of at least one required theory.