Unifying Domain-specific Tools with Rosetta

Nicolas Frisby
University of Kansas
15 September 2008
Analysis Tools

- “General-purpose”
  - Theorem provers (logics)
  - Model-checkers (temporal logics and state machines)

- Domain-specific
  - Power estimators (RTL and/or ISA)
  - Security analysis (channel specifications)
  - Netlist generation (HDLs)
Research Objective

- Encapsulate these disparate semantic domains
  - Use Rosetta as the shared semantic domain
  - Use a language processing DSL
The Rosetta Nexus

- HOL
- CoC
- PVS
- Custom
- SAL
- VHDL
- C
Why Rosetta?

- The Rosetta language is designed for heterogeneous, system-level specification
  - The **domain** construct models vocabulary and axioms
  - The **functor** construct models transformations between domains
- **e.g.** the **state_based** domain
- **e.g.** the functor to the **static** domain
Why use a Language Processing DSL?

▶ Numerous syntactic transformations
  ▶ from Rosetta to each tool’s input language
  ▶ and back to Rosetta

▶ Issues
  ▶ similarities between transformations (duplication)
  ▶ changes in tools’ input languages (flux)
  ▶ language subsets (partiality)

▶ Answers
  ▶ reuse avoids duplication
  ▶ modularity and extensibility mitigate flux
  ▶ component-based approach supports partiality
InterpreterLib

- Write language processors with InterpreterLib
- A Haskell library (hence DSL) implementing
  - generic programming for modularity
  - Modular Monadic Semantics for extensibility and components
  - algebra combinators for reuse
Software-defined Radios (SDRs)

- Troop safety $\rightarrow$ mission critical $\rightarrow$ formal methods
- Specifications exhibit heterogeneity
- Specifications exhibit system-level properties
- $\therefore$ a good problem for demonstrating the methodology
SDRs?

Conventional radio

versus

Software-defined radio
Heterogeneity in SDRs

Waveform Architecture

Set-up

Tear-down

TX
- Compression
- Encryption
- Error Cntl
- Spreading
- Modulation
- Transmit

RX
- Decompress
- Decryption
- Error Cntl
- De-Spread
- De-Mod
- Receive

Confidentiality
Integrity
Availability

Authentication
Coding Scheme

Frequency, hop rate, hop set

Send Data

Receive Data

Power
Cost
Form factor
Evaluation and certification
Products

- **Completed**
  - Implemented InterpreterLib
  - Identified SDRs as an appropriate problem space

- **Tasks**
  - Use SAL model-checker for confidentiality analysis
    - domain for SAL
    - functors to SAL and back
  - Use another tool for SDRs (unidentified yet)
  - Demonstrate reuse between the two analyses
  - Demonstrate modularity/extensibility in the staged development of the analyses
Concerns

- Exploratory: large engineering effort for uncertain pay-off
- Rosetta semantics are in the standardization process
- Rosetta expression language
  - Too expressive (e.g. higher-order functions, dependent types)
  - Available regardless of domain
InterpreterLib Basics

data Arith x = Literal Int | Add x x

data Fix f = In (f (Fix f))
type Language = Fix Arith

phi (Literal i) = i
phi (Add x y) = x + y

cata :: (f a -> a) -> Fix f -> a
  cata phi = phi . fmap (cata phi) . out

evaluate :: Language -> Integer
evaluate = cata phi
Extensibility/Partiality

```haskell
data Mult x = Mult x x

type Language2 = Fix (Mult :+: Arith)

phi2 (Mult x y) = x * y

evaluate2 :: Language2 -> Integer
evaluate2 = cata (phi @+++ phi2)
```

Generic Programming

data Lam x = Var String | Lambda String String x

phi = updateAlgebra upd Data.Foldable.fold
upd (Var x) = Set.singleton x
upd (Lambda n body) = Set.delete n body

freeVars :: Fix f -> Set String
freeVars = cata phi
Algebra Combinators

\[
\begin{align*}
\text{phiType} &:: \text{VHDL Type} \rightarrow \text{Type} \\
\text{phiNetlist} &:: \text{VHDL Type} \rightarrow \text{VHDL Netlist} \rightarrow \text{Netlist} \\
\text{synthesize} &\equiv \text{cata (phiType} \ \\
&\quad \text{`sequenceAlgebra` phiNetlist)}
\end{align*}
\]