Embedding Cryptol in Higher Order Logic

Joe Hurd

Computing Laboratory
University of Oxford

Cambridge University
Tuesday 13 March 2007
1. Introduction
2. Existing Embeddings
3. A Natural Embedding
4. Summary
Cryptol is a domain specific language for cryptographic applications.

- Developed by Galois Connections, Inc. since 2002.
- Programs can be executed by the Cryptol (symbolic) interpreter.
- Or compiled to low-level software or hardware.
What is the meaning of a Cryptol program?

To use Cryptol as a stepping stone in Evaluation Assurance Level 7 (EAL7) of the Common Criteria, must model Cryptol programs in a formal logic.

The Cryptol program can then be formally proved equivalent to a specification or low-level implementation modelled in the same logic.
Higher order logic is a natural choice for modelling Cryptol programs.

The type system is a close match with Cryptol’s.

It is a ‘wide-spectrum’ logic, thus also able to model the specification and/or low-level implementation.

- No need to defend linking two logics in the evaluation case.

Example: verifying a Cryptol implementation of elliptic curve cryptography.
Cryptol has nested mutually recursive sequences.
These are potentially infinite data structures.
  - **OK:** HOL4 already has a theory of lazy lists.
They can contain complex dependencies—evaluating an element might result in program divergence.
  - **Warning:** Higher order logic functions are total.
**Warning:** Cryptol’s type system is more fine-grained than higher order logic.
Verification of Embedded Programs

- Assume we have a semantically faithful embedding of Cryptol into higher order logic.
- How easy is it to prove properties of the embedded programs?
- Rule of thumb: the more ‘natural’ the embedded programs, the easier to verify.
  - ‘Natural’ example 1: only terminating Cryptol programs.
  - ‘Natural’ example 2: encoding sequence length information in higher order logic types.
- **Tradeoff:** More natural embedding = fewer embeddable Cryptol programs.
  - SHA1 : \([N] \rightarrow [160]\)
Deep and Shallow Embeddings

- Cryptol
- Syntax
- HOL
- Syntax
- Functions

deep

shallow

meaning
Li & Slind (2005)

- Shallow Embedding of Cryptol into HOL4.
- Cryptol sequences are embedded as HOL4 lazy lists.
- Cryptol sequence operations (split, join, etc.) can be uniformly defined in higher order logic.
- Arithmetic operations convert finite subsequences of boolean lazy lists to HOL4 words.
- Syntactic sugar for finite and infinite ranges.
- A definition principle for a particular form of terminating mutually recursive sequences.
Matthews (2005)

- Deep Embedding of fCryptol into Isabelle/HOL.
- fCryptol is a subset of $\mu$Cryptol.
- Finite and infinite sequences of signed bitvectors have different types.
- Defines an abstract syntax of fCryptol, including mutually recursive sequence definitions.
- The denotational semantics assigns nonterminating sequences a default value.
Matthews (mid-2006)

- Shallow Embedding of $\mu$Cryptol into Isabelle/HOLCF.
  - HOLCF is an extension of HOL with first-class support for partial functions.
- Mutually recursive sequences are embedded as partial functions.
  - Proving interesting properties require additional proof obligations that expressions terminate.
- Also contains a shallow embedding of the ACL2 logic.
  - Can be used to verify the initial phases of the verifying $\mu$Cryptol compiler mcc.
What is a Natural Embedding?

- What is the most natural embedding of Cryptol into higher order logic?
- **Correlated question:** How can Cryptol programs be embedded to simplify reasoning about the resulting programs?
- Note that this might involve a severe restriction on embeddable Cryptol programs.
- First step: restrict to terminating Cryptol programs, and embed as native higher order logic functions.
- Second step: how much information can be encoded in higher order logic types?
Embed infinite $\alpha$-sequences as

$$\alpha \text{ inf } \equiv \mathbb{N} \rightarrow \alpha .$$

Embed finite $\alpha$-sequences of length $n$ as

$$\alpha \text{ vector } \equiv \tau_n \rightarrow \alpha$$

where $\tau_n$ is a specially constructed type having $n$ elements.

Every sequence in an embedded Cryptol program carries around its length as part of its type.

- No need for side-conditions about infinite or finite sequence length in theorems: good for verification!
Sequence Operations

- Using Harrison’s finite Cartesian products it’s possible to define sequence operations that are polymorphic over $\tau_n$.
- Need finite and infinite versions of the standard sequence operations:

  $$\text{seq\_map\_finite} : (\alpha \to \beta) \to [n]\alpha \to [n]\beta$$

  $$\text{seq\_map\_infinite} : (\alpha \to \beta) \to [\inf]\alpha \to [\inf]\beta$$

- In practice map to standard Cryptol syntax: the HOL4 parser disambiguates by input argument type.
Consider a Cryptol implementation of the Fibonacci sequence:
\[ \text{fib} = [0 \ 1] \# [\mid x + y \mid \mid x \leftarrow \text{drop}(1,\text{fib}) \mid \mid y \leftarrow \text{fib} \mid] \]

The sequence comprehension can be embedded into higher order logic as

\[ \text{map} (\lambda(x, y). \ x + y) (\text{zip} (\text{drop} 1 \ \text{fib}) \ \text{fib}) \, . \]

Print zip using the \(\mu\)-Cryptol symbol \(|\), and introduce a new binder syntax for map:

\[ (\text{seq} (x, y). \ x + y) (\text{drop} 1 \ \text{fib} \mid \text{fib}) \]
Mutually Recursive Sequences

- Two step procedure:
  1. Define the sequences as functions $\mathbb{N} \rightarrow \alpha$.
  2. Prove them equivalent to the syntax supplied by the user.

- Just an extension of Slind’s recursive function definition package TFL.

- Fibonacci example:
  1. $\text{fib } i \equiv \text{if } i < 2 \text{ then } V[0w; 1w] \text{ else } \text{fib } (i - 1) + \text{fib } (i - 2)$.
  2. $\vdash \text{fib } = V[0w; 1w] \# (\text{seq } (x, y). \ x + y) \ (\text{drop} \ 1 \ \text{fib} \mid \text{fib})$.

- Compare with the Cryptol implementation:
  $\text{fib } = [0 \ 1] \# [\mid x + y \mid | x \leftarrow \text{drop} \ (1,\text{fib}) | | y \leftarrow \text{fib} |]$
Motivated and surveyed existing approaches to embedding Cryptol in higher order logic.

Presented a new approach aimed at simplifying verification of embedded programs.
  - So far only know that it can scale to naturally embed TEA.

The ‘right embedding’ will surely depend on the particular reasoning task to be performed, and will borrow ideas from all approaches.