Talking to the Rooster

Communicating with Coq via XML

Tom Hutchinson

Talking to the Rooster

Communicating with Coq via XML

Tom Hutchinson

Note: I get really nervous when giving talks.
Please stop me if I start speaking really fast.

Outline

- Motivation
- XML in ML
- XML in Coq
- Reflections

Motivation

- Most common method of communicating with Coq: copy and paste
- Can write things in ML
  - Largely undocumented
  - Scary for users
  - Many users (even experts) have never seen the source
    - ex: Adam Chlipala
  - Changes to source easily break your code

Existing XML

- Claudio Sacerdoti Coen's extraction
  - mathematical library
  - focus on logical information
  - DTD for terms
  - no Coq specific information
- Hugo Herbelin's external tactic
  - uses Claudio's term DTD
  - call external program inside a proof
  - send and receive terms

Working with XML

- XML supposed to be simple
- Alphabet soup of standards around it
  - DTD, DOM, SAX, XSTL, SOAP, et cetera...
- Which tool(s) to use?
- How does XML fit into function programming?

XML Parsing

- Tree based
  - DOM standard
  - Read whole document first, then use
  - + Validating
  - + Fairly easy to use
  - - Slow
  - - Uses a lot of memory

XML Parsing

- Streaming style
  - SAX standard
  - XML document as a stream of events
  - + Fast
  - + Low memory overhead
  - - Non-validating
  - - Requires writing your own parser

Streaming Parsing

- Push style (SAX)
  - Came before pull style
  - Register callbacks
  - Tedious to use
- Pull style
  - User requests next event
  - Lazy list of XML events

What an XML Parser gives you

- Not much!
- Still need to write code to use it
- More like a lexer
- Handling code generally either:

  efficient or readable

XML Handling

- While parsing document, build a tree
- In imperative world, use references
  - Set to null, write as parse elements
- Function world
  - Grow the stack (easy / inefficient)
  - Manage a data stack of open tags
    - Big pain

XML in ML

- Typically see definitions like

```
type xml =
    | Element of string * (string*string) list * xml list
                (*     tag          attributes          subtrees    *)
    | PCData of string
```

- Imagine if in the kernel was

```
type constr =
    | Term of string * string list
```

!!!

XML in ML

- Most XML parser in OCaml are basically translations of C parsers or OO parsers
- If only we had a good way of representing trees in our language...
- ADTs are a good fit for representing XML
  - ...almost
  - Would like to only be able to represent valid XML
  - XDuce and CDuce adding separate type system of regular expressions

XML in ML

- Can enforce validity with phantom types
  - Used in Ocsigen for HTML
  - Phantom types made up of polymorphic variants according to DTD
  - Must go through constructor functions
    - After fancy tricks, Ocsigen still uses strings :-(

XML in ML

- Still, regular old ADTs are much better than strings
- Having tons of strings floating around everywhere
  - Ugly / unsafe code
  - Hard to change
  - Mutable strings lead to huge duplication

Working with XML

- Treat XML parser as a lexer
- Use a parser generator
- XML parsing only needs a DFA
- LALR bottom up method well suited for building ADTs
- Surprisingly almost no one does this
  - Recently saw a paper about this by Luca Padovani and Stefano Zacchiroli – wish I read this two years ago

XML in Coq

- Using ocamllex and ocamlyacc
- Lex turns XML tags into tokens
- Yacc turns tokens into Coq internal datatypes
- Can easily replace ocamllex with ulex (handles unicode) or a real XML parser
  - Xmlm (one .ml and .mli)
  - PXP
  - Binding to a C library

XML in Coq

- Current capabilities:
  - From Coq, call external program
  - Communicate via pipes
    - Switch to sockets?
  - Tactic to send goal, receive Ltac expression
  - Command to receive Gallina expression

Difficulties

- Coq parsing is centralized
  - Focal points where structured data travels through
- Coq printing is not :-(
  - Spread out through codebase
  - Extensible
  - By the time it gets to a cental point, just a bunch of strings

Design Considerations

- How “smart” should API/interface be?
- PCoq example
  - The more the interface knows about, the more fragile it is
- Terms very stable, commands less so, tactics always changing
- Driving consideration: How to make this maintainable?

Future Directions

- External programs using Coq as a tool
  - Drive Coq remotely
- External programs called from Coq being able to ask questions
  - Start external program, send request, then program can ask Coq for more information before sending an answer
- Documentation and big examples

Demo

Reflections

High Barrier of Entry

- Hard to become a Coq developer !
- Large undocumented codebase
  - Please write more !
    - Look at Agda – worried about what would happen if Ulf leaves
  - Use English
  - Matita has big paper documenting kernel
  - Arnaugh's paper about new new proof engine
    - Brief, accessible
  - Leave more records of what you do / what you are thinking – does not have to be paper quality

Rant

- Coq is like OCaml
- Problems of OCaml are the problems of Coq
  - Users assume monolithic development
    - “I wish INRIA would add XYZ”
  - Learn through apprenticeship
    - Master / pupil
    - Learn by reading code, separately reinventing the wheel
    - Produces great people but too few
    - Wish: Design Patterns for ML / Coq
  - Development has to be tied to “research”

Rant

- Look at Haskell community : GHC
  - Small core development team
  - Large outside group of system hackers
  - Open to outside improvements
- Look at ::gasp:: Java
  - Great tools
  - Great books
  - Real documentation
  - (Though complicated)

    assumption is everyone can do it

Questions