Olmar: Manipulating C and C++ abstract syntax trees in Ocaml

Hendrik Tews
SoS group, Radboud University Nijmegen
http://www.cs.ru.nl/~tews

Announce Olmar and recruit new Olmar users

Olmar makes it possible to use Ocaml/pattern matching to manipulate/process abstract syntax trees of C/C++ programs.

C++ Parsers

- there only few freely available ones
- those few are written in C or C++

Formal Methods support

- formal methods for C++ require automated C++ Ast manipulations (semantics compiler)
- in my opinion C++ provides no means to process abstract data types (trees with variant node types)
- especially the visitor pattern is absolutely inadequate

Dilemma

- recursive functions defined by pattern matching are much more adequate
- however, there is no C++ parser in a functional language and writing one is not an option

Solution

- mount an Ocaml backend to an existing C++ parser
- the Ocaml backend converts the internal abstract syntax tree into a pure Ocaml variant type to be further processed by pure Ocaml programs

Elsa: The chosen one

- BSD licensed C++ parser from UC Berkeley
- contains disambiguation, a type checker and template instantiation
- not toy-like: parses Mozilla, Linux, ...
- huge number of regression tests

- Elsas Ast splits into syntax nodes and type nodes
- the syntax nodes form a regular tree
- type nodes come from the type checker, they introduce cycles in the Ast
- for the syntax nodes: C++ types and lots of code (e.g., visitors) are generated by astgen from an ML-style variant type definition (in C++ syntax and braces)
- very easy to add new features to the Ast: patch astgen to generate the required things

Discarded: XML

- Elsa can output its internal Ast in XML
- Gerd Stolpmanns excellent XML library is only missing one feature: parsing XML into a variant type according to a DTD

Current: astgen meta programming in C++

- small enhancements to the astgen input language (e.g., to distinguish fields that can contain a null pointer)
- astgen generates now additionally:
  - Ocaml variant type definition for the Ast
  - C++ code to create the Ocaml version of the Ast
  - some Ocaml glue code
- all this and the Ocaml runtime is linked into the elsa executable

Future: astgen meta programming in Ocaml

- the astgen-internal syntax tree to represent astgen files is itself generated by astgen
- add Ocaml reflection capabilities to astgen to export astgen-Ast's to Ocaml
- generate all the needed code from Ocaml programs that process those astgen-Ast's

Enumerations

- 16 enumerations with 219 constants
- not (yet) processed by astgen
- used emacs functions to generate the necessary code

Type Nodes

- are not (yet) astgen generated
- required a lot of boring code to be hand-written

Null pointers

- translated into an 'a option in Ocaml
- used the regression tests to identify fields that can contain Null

Sharing

- cache Ocaml objects
- gives a performance penalty because roots in C++ for the Ocaml garbage collector are scanned at every minor collection

Cycles

- cycles pose a problem because I prefer to allocate the Ocaml objects inside an Ocaml function
- hand select *cycle closing edges (CCEs)*
- CCEs get a option ref type in ocaml
- CCEs are not recursed into during the generation of the Ocaml Ast
- instead they are queued in a list of broken cycles, to be processed and updated later

Correctness of the Ocaml Ast

- an error in the Ocaml Ast generation might crash pure Ocaml programs that process the broken Ast
- Memcheck library for dynamic type checks in Ocaml [1]
  - uses Camlp4 to compile Ocaml type definitions into a data type
  - provides a function that performs runtime type checking (with limitations)
  - does not support polymorphic shared objects
  - outperforms the alternative from the type-safe unmarshalling branch (50s for 7MB instead of 5 hours)

References

[1] H. Tews. Memcheck: runtime typechecking for Ocaml. Web site http://www.cs.ru.nl/~tews/memcheck/, December 2006.

- preprocess (no preprocessor in Elsa yet)
- parse C++ sources and save Ast to disk
- [optional] check consistency with Memcheck
- run Ocaml application on saved Ast
  (could also be linked directly into the parser)

Olmar applications

- count_ast
- check_oast
- ast_graph

Achieved

- Ocaml Ast reflection pass mounted to Elsa
- manipulate C++ Asts from Ocaml
- download at http://www.cs.ru.nl/∼tews/olmar/

Future Work

- new less invasive structure
- integration in the Oink repositories
- merge type nodes and enumerations into astgen

The Java community profited a lot from

- the tight standard of the Java language
- a cross-research-group specification language — JML

What can we do to form a prospering C/C++ formal methods community?