Checking Type Safety of Foreign Function Calls

Mike Furr

FFIs

- High level languages have become very popular
  - Most include a foreign function interface (FFI)
- Still rely on C for certain operations:
  - OS system calls / low level libraries usually impossible to call directly
  - Large legacy libraries may be infeasible to reprogram in a new language
  - Performance critical code may require C

Glue Code

- To use an FFI, programmer must write "glue code"
  - Convert data structures and semantics between host and foreign languages
  - Typically only written in one of the languages
- One approach: interface generators (e.g., SWIG)
  - Not as flexible as hand written code
- Hand written glue code is low level and it is easy to make mistakes
  - Little or no static checking is provided

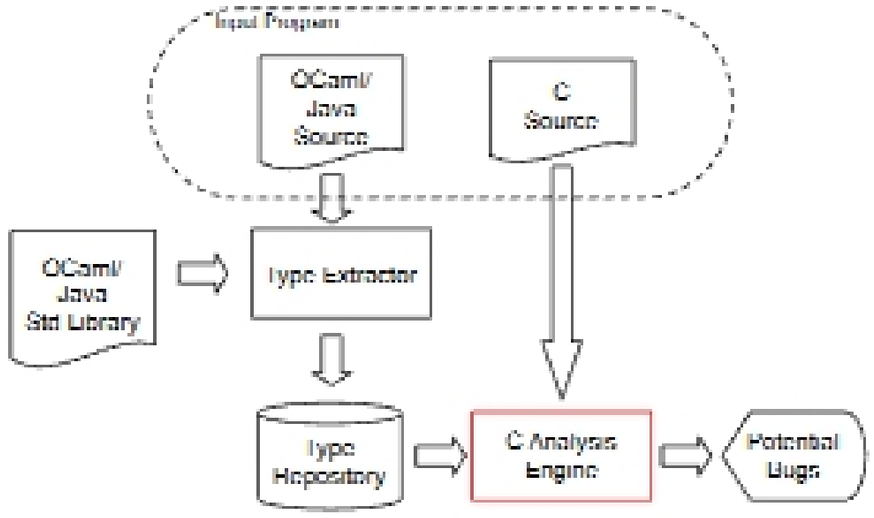
Saffire

- Static Analysis of Foreign Function InteRfacEs
  - Static type inference for OCaml and Java FFIs
- Most programmer work is done on the C side
  - Concentrate our analysis there
  - Ensure that C code uses high level types correctly
- General C code is rather difficult to analyze
  - However, "glue code" tends to use C in simple ways

Implementation

- OCaml and Java are very different languages
  - Contrast pattern matching vs method dispatch
  - Combined, provide a cross section of FFI designs
- High level approach to checking FFIs quite similar
  - Implementations shared a lot of infrastructure
    - Both built with CIL
  - Type systems specialized to the high level language

Architecture



OCaml FFI

OCaml:

```
external ml_fun: int → int list → unit = "c_fun"
```

C:

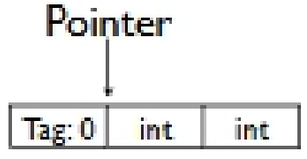
```
value c_fun(value int_arg, value int_list_arg)...
```

- `value` can be either a primitive (`int`) or a pointer into the heap (`int list`)
- No static checking that `value` is used correctly

OCaml Types in C

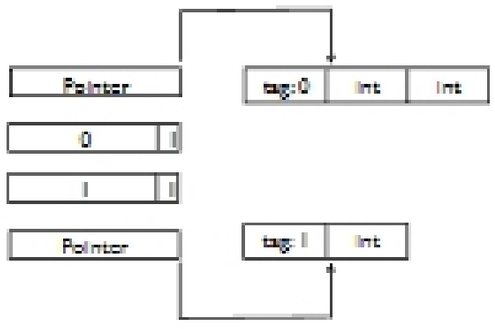
Integers are stored unboxed with lowest bit set to 1

Complex types like `(int×int)` are represented in boxed form:



Sum Types

type jargon =
  Foo of int×int
| Bar
| Baz
| Qux of int



Accessing Values

- Int_val() and Val_int() are used to shift in/out the lowest bit for integers
  - Easy to confuse
  - Can be applied to pointers without warning
- Tag_val() and Field(v,i) extract tag and data from complex types
  - Can be applied to integers without warning
- Is_long() used to distinguish pointers and integers

Pattern Matching

```
value speak(value j) {
  if(Is_long(j)) {
    if(Int_val(j) == 0)
      /* Bar */
    if(Int_val(j) == 1)
      /* Baz */
  } else {
    if(Tag_val(j) == 0)
      /* Foo */
    if(Tag_val(j) == 1)
      /* Qux */
}
```

type jargon =
  Foo of int×int
| Bar
| Baz
| Qux of int

Need to track the results of conditionals

Representation Overlap

type jargon =
  Foo of int×int
| Bar
| Baz
| Qux of int