Pocaml: Poor Man’s OCaml

Leo Qiao, Peter Choi, Yiming Fang, Yunlan Li
Advised by John Hui
Language Introduction & Demo

- "poor man’s OCaml"

- Has main features of OCaml, such as higher-order functions, partial application, pattern matching, parametric polymorphism, and much of the same syntactic sugar.

- Includes builtins and a standard library for common operations on lists and I/O.
Compiler Pipeline
Lambda Lifting Demo

```
Code/cs@columbia/plt/pocaml_docker git:(main)
→ ./pocaml -l --
let lambda =
  let a = 1 in
  let b = 2 in
  let str = "pocaml" in
  fun lst ->
    list_iter
    ( fun el ->
      match el with
      | 1 -> (fun x -> print_string str) 1
      | _  -> print_int (a + b)
    )
  lst

let _ = lambda [ 6; 1; 9 ]
make: Nothing to be done for `default'.
3pocaml3%
```
- produces the correct output “3pocaml3”
- demonstrates the correctness of lambda lifting in
  - let-in expression
  - applications
  - match arms
  - lambda
Lambda Lifting

- makes lambdas function properly in Pocaml
- happens after the lower_ast compiler pass
- rules: lift into top level functions all lambdas except
  top-level lambdas:
    let a = fun b -> b
- Immediately nested lambdas:
  let add3 = let a = 3 in fun x -> fun y -> x + y + a

- implementation
- example

```ocaml
let increment = let i = 1 in
  let j = 2 in
  fun x -> x + i * j
```

```ocaml
let lambda_1 = fun x -> ( fun i -> ( fun j -> x + i * j ) )
let increment = let i = 1 in
  let j = 2 in
  lambda_1 j i
```

Fig: Reduced Abstract Syntax Tree after lower_ast
Codegen: run-time value representation

- `_pml_val`
  - pointer to `_pml_val_internal`
- `_pml_val_internal`
  - type information
    - added to support operator overloading
  - union of all Pocaml data types
- closure:
  - representation for lambda
  - run-time support for partial application
  - lambda creation and application are done with C run-time library:
    - `_pml_val_make_closure(_pml_func *fp, _pml_int num_args);`
    - `_pml_val_apply_closure(_pml_val closure, _pml_val arg);`
- uniform representation
  - parametric polymorphism
Codegen: program representation

- Pocaml: sequential evaluation of top-level definitions
- LLVM: evaluation of an entry main function
- solution:
  - top-level variable -> global variable
  - value evaluation -> _init_ functions
  - sequential evaluation -> call _init_ functions in main
- example:
  - generated LLVM with parts omitted
  - notice
    - lambda =\= function
    - lambda == closure
    - _init_f() stores the closure in @f

```ocaml
let f x = x + 1
let y = f 2
```
C built-ins

- Built-ins functions exist in the form of closure.

- During codegen, the built-ins initializer, `_init__builtins`, is declared.

- The C code for built-in operators and functions is linked to the rest of the LLVM code so that it can be accessed.

```ocaml
(* Declare the builtin-init function *)
let builtins_init : L.llvalue =
  L.declare_function "_init__builtins" pml_init_t the_module

(* Build call in main for the builtin-init function *)
let _ = L.build_call builtins_init [] "" main_builder

# build builtins C static library
cd builtins
cp ${builtins_ar} ../${build_dir}

cd ..

# link the generated llvm with builtin
cd ${build_dir}
$LLC -relocation-model=pic ${basename}.ll > ${basename}.s
$CC -o ${basename}.exe ${basename}.s ${builtins_ar}
```
C built-ins

- One example of a function called by \_init\_builtins is \_init\_add .

- A closure containing the execution instructions is made public, created in the same as for a lambda expression and used in the same way during codegen.

```c
_pml_val _add;

_pml_val _builtin__add(_pml_val *args)
{
    _pml_val left, right;
    left = (_pml_val)args[0];
    right = (_pml_val)args[1];
    _pml_int res = _pml_get_int(left) + _pml_get_int(right);
    return _make_int(res);
}

void _init__add()
{
    _add = _make_closure(_builtin__add, 2);
}
```
Standard Library

- List
  - length, hd, tl, append
  - iter, filter, map, mem
  - fold_left, fold_right
- I/O
  - print functions for all types
  - print functions for printing lists
  - to_string functions for all types
- example:
  - Implementing graph algorithms with stdlib
  - Demo
Automated Testing

- **Unit Testing**
  - Used during active development
  - Pretty printing for AST and IR
  - Utilized OCaml’s ppx_expect functionality to auto-generate expected value

- **Integration Testing**
  - Automatic shell script
  - MicroC-style
  - Checks the output/error against reference
  - Saves the execution details to log

- **Test suites:**
  - More than 50 test cases for integration test
  - Include both tests that should pass and should fail
Conclusions and Lessons Learned

- "Be the compiler"

- The power of using the team to solve tough problems, rather than fighting alone

- Viewing programming languages from a more critical lens

- Clean code can be easily explained to others