Compiler Construction: Assignment 6

Fabian Krause

February 7, 2022

Assignment 6: Compiler for $\mathcal{L}_{Fun}$

10 passes:

1. Shrink: $\mathcal{L}_{Fun} \rightsquigarrow \mathcal{L}_{Fun}$
2. Reveal Functions: $\mathcal{L}_{Fun} \rightsquigarrow \mathcal{L}_{Fun}^{FunRef}$
3. Limit Functions: $\mathcal{L}_{Fun}^{FunRef} \rightsquigarrow \mathcal{L}_{Fun}^{FunRef}$
4. Expose Allocation: $\mathcal{L}_{Fun}^{FunRef} \rightsquigarrow \mathcal{L}_{Fun}^{FunRef}$
5. Remove Complex Operands: $\mathcal{L}_{Fun}^{FunRef} \rightsquigarrow \mathcal{L}_{Fun}^{mon}$
6. Explicate Control: $\mathcal{L}_{Fun}^{mon} \rightsquigarrow \mathcal{C}_{Fun}$
7. Select Instructions: $\mathcal{C}_{Fun} \rightsquigarrow x86_{callq^*}^{Var}$
8. Register Allocation: $x86_{callq^*}^{Var} \rightsquigarrow x86_{callq^*}$
9. Patch Instructions: $x86_{callq^*} \rightsquigarrow x86_{callq^*}$
10. Prelude and Conclusion: $x86_{callq^*} \rightsquigarrow x86_{callq^*}$

1. Shrink Pass

- Add cases for `Call` and `Return`
- Gobble up top-level statements into a new `main` function

2. Reveal Functions Pass

- Get arity of all functions by iterating over defs, store in dict `funs` : `dict[str, int]`
- Recursively process body, replace each `Name(f)` with `FunRef(f, funs[f])`

3. Limit Functions Pass

- Translate parameters in function definitions:
  $(x_1, T_1), \ldots, (x_5, T_5), (x_6, T_6), \ldots, (x_n, T_n)$ becomes
  $(x_1, T_1), \ldots, (x_5, T_5), (tup, (T_6, \ldots, T_n))$
- Transform function body, replacing occurences of x_i, $i \geq 6$, with tuple accesses $tup[i - 6]$
- Transform arguments in function calls: $[e_1, \ldots, e_5, e_6, \ldots, e_n]$ becomes $[e_1, \ldots, e_5, (e_6, \ldots, e_n)]$

4. Expose Allocation Pass

- Add cases for `Call` and `Return`

5. Remove Complex Operands Pass

- `FunRef` and `Call` are complex
- Arguments to `Call` must be atomic
- Expression in `Return` should not be transformed

6. Explicate Control Pass

- Add case for `Call` in `explicate_pred`
- Implement `explicate_tail`:
  - Replace `Call` with `TailCall`
  - Assign atomic expressions to temporary variable, return that variable

7. Select Instructions Pass

Statements

- Translate assignments $x = \texttt{FunRef}(\ldots)$ to `leaq` instruction
- Translate calls
  - Handle argument passing in calls by using the argument passing registers
  - Take result from `rax`, if necessary
- Translate `return` by `mov` to `rax`
- Translate `TailCall` to `TailJump`

Function definitions

- Transform statements
- Move parameters from argument passing registers into local variables
- Add `conclusion` and `start`

8. Register Allocation Pass

`get_read_write_locations`

- `IndirectCallq`, `TailJump`:
  - Read: Argument passing registers, target
  - Write: Caller-saved registers

`build_interference`

- Spill `TupleType` variables that are live at a function call $\Rightarrow$ add interferences edges to callee-saved registers

`assign_homes`

- Perform register allocation for each function separately
- Add cases for `IndirectCallq`, `IndirectJump` and `TailJump`

9. Patch Instructions Pass

- Destination argument of `leaq` must be register
- Argument of `TailJump` must be `rax`

10. Prelude & Conclusion Pass

- The usual prelude and conclusion, but for each function
- Initialization of garbage collector in `main` prelude
- Translation of `TailJump`

Questions?