Assignment 8

Contents

1. Background

In this assignment we complete our "universal intermediate language" (UIL), an intermediate language that is independent of the source language, and thereby a suitable target for multiple source languages.

To complete UIL, we need to add primitives for allocating storage, storing data in allocated storage, and retrieving data from allocated storage.

2. UIL

A grammar for UIL is below. To our previous intermediate language, it adds `alloc`, `mref`, and `mset!` primitives.

```
Program  ⟶  (letrec ([label (lambda (uvar*) Body)]*) Body)
Body     ⟶  (locals (uvar*) Tail)
Tail     ⟶  Triv
          |  (alloc Value)
          |  (mref Value Value)
          |  (binop Value Value)
          |  (Value Value*)
          |  (if Pred Tail Tail)
          |  (begin Effect* Tail)
Pred     ⟶  (true)
          |  (false)
          |  (relop Value Value)
          |  (if Pred Pred Pred)
          |  (begin Effect* Pred)
Effect   ⟶  (nop)
          |  (set! uvar Value)
          |  (mset! Value Value Value)
          |  (Value Value*)
          |  (if Pred Effect Effect)
          |  (begin Effect* Effect)
Value    ⟶  Triv
          |  (alloc Value)
          |  (mref Value Value)
          |  (binop Value Value)
          |  (Value Value*)
          |  (if Pred Value Value)
          |  (begin Effect* Value)
Triv     ⟶  uvar | int | label
```

Unique variables (*uvar*), labels (*label*), integers (*int*), binary operators (*binop*), and relational operators (*relop*) are unchanged from the intermediate language described in the preceding assignment. The machine constraints on integer values also remain from the preceding subset.

3. Semantics

`(alloc` *expr*`)` evaluates *expr* to produce a value *n*, reserves *n* bytes of storage by incrementing the allocation-pointer register, and returns the address of the base of that storage. The value *n* should be a multiple of the word size for the host machine, The allocation-pointer register should be determined via the new helpers.ss variable `allocation-pointer-register`.

`(mset!` *base-expr offset-expr expr*`)` evaluates *base-expr*, *offset-expr*, and *expr* to produce the values *base*, *offset*, and *val*. It stores *val* at address *base* + *offset*.

`(mref` *base-expr offset-expr*`)` evaluates *base-expr* and *offset-expr* to produce the values *base* and *offset*. It returns the value stored at address *base* + *offset*.

4. Things to do

A new pass, `verify-uil`, must be written and will remain a permanent part of our compiler, even after we begin creating the language-dependent part of the compiler, as a check to make sure that the language-dependent portion creates well-formed UIL code. This pass is a straightforward modification of last week's verifier.

A major challenge of this assignment is to decide for yourself the changes needed to support the three new primitives, beyond adding `verify-uil`. We will discuss possibilities for the overall strategy in lecture. You may use any other resource you can find, including current and former students of this class. As always, you must acknowledge any collaboration, and the changes to your code must be primarily your own work. The instructors will help you with problems that arise as you try to carry out your plan, but they will try to avoid giving you any strategic guidance outside of moderating discussions in lecture.

Documentation is particularly important this week and should include a detailed description of your strategy. We encourage you to draft this description early and keep it up-to-date as you work on your compiler. This will help you refine your thinking.

5. Boilerplate and Run-time Code

The run-time code does not change, but the boilerplate code must initialize the allocation-pointer register to the base of the heap. The helpers.ss `emit-program` now generates boilerplate code that does this.

6. Testing

A small set of invalid and valid tests for this assignment will be posted later in the week in tests8.ss and may include tests submitted by students in the class. You should make sure that your compiler passes work at least on this set of tests.