Project 2: translation from parallel assignments to sequence of assignments

October 12, 2017

1 Project description

The goal of this project is to write a program that will translate a parallel assignment into a sequence of assignments. Here is a very simple example of a parallel assignment (swap):

```
x, y := y, x
```

Here `x` is assigned `y` and vice-versa. Now, if you wanted to do this in C for example, you would need a temporary variable:

```
tmp = x;
x = y;
y = tmp;
```

You can find more examples in [1]. A parallel assignment is written $(x_1, x_1, \dots x_n) := (e_1, e_2, \dots, e_n)$ and describes a computation, where each x_i is assigned e_i at the same time. This can be a very useful construct, and you can implement it in many languages using tuples or structs. However, it doesn't always translate easily to a sequence of assignments, which is a more common programming construct.

Your goal is to translate any parallel assignment, as described by the syntax described in Section 3, into a C code containing a series of assignments. Additionally, you should eliminate common subexpressions in the e_i of the parallel assignment and compute them only once.
You have noticed that for this very simple example we had to create a temporary variable, and you will also have to add temporaries for the common subexpression problem. When doing so, your program should detect automatically the type of the temporaries to create and clearly give a variable declaration in the ouput. You should also be careful not to create too many temporaries that are not used for common subexpression elimination. An easy solution would then be to create as many temporaries as there are variables!
Your last task will be to output a proof of equivalence of the two programs in Dafny.

Notation The project is divided in four objectives. For a program that outputs a correct sequence of assignments and the declaration of the variables to be declared you will get 40 points. Then 20 points if it performs the common subexpression elimination optimization, 20 points if it creates an optimal number of temporary registers and finally 20 points if it outputs a proof of equivalence in Dafny.

2 Language and provided code

You can choose the language you want, and therefore is no code provided. You should be able to parse simple S-expressions as specified by the input syntax in Section 3, but there are no points awarded for this task, so it shouldn't take you too much time.

3 Input syntax

3.1 Basic types

We only use three basic types in the input programs: `int`, `float` and `bool`.

⟨*type*⟩ ::= 'int' | 'float' | 'bool'

3.2 Expressions

⟨*expression*⟩ ::= ⟨*identifier*⟩
| ⟨*constant*⟩
| '(' ⟨*binary-operator*⟩ ⟨*expression*⟩ ⟨*expression*⟩ ')'
| '(' ⟨*unary-operator*⟩ ⟨*expression*⟩ ')'
| '(' 'vector-ref' ⟨*expression*⟩ ⟨*expression*⟩ ')' // Array access
| '(' 'ite' ⟨*expression*⟩ ⟨*expression*⟩ ⟨*expression*⟩ ')' // Conditional expression

⟨*binary-operator*⟩ ::= '+' | '-' | 'min' | 'max' // Arithmetic
| '&&' | '||' // Boolean
| '==' '>' | '<' | '>=' | '<=' // Comparisons

⟨*unary-operator*⟩ ::= '-' // Arithmetic
| '!' // Boolean not

3.3 Input program

The input program is a set of parallel assignments:

⟨*input-program*⟩ ::= ⟨*List ident*⟩ ':=' ⟨*List expression*⟩

where *List X* is a comma-separated list of syntax elements *X*.

References

[1] P. H. Welch. Parallel assignment revisited. *Software: Practice and Experience*, 13(12):1175–1180, 1983.