Problem set 4

Arve Nygård

22.02.2014

Problem 1: Symbol Tables

a) What is a symbol table, and why is it needed?

A symbol table is a table containing symbol names, and properties of those symbols names, such as type, address and other semantic info. When we see a declaration, we enter the symbol and the properties into the table. Later, when we see a useage of the symbol, we look it up in the table to find the properties.

We need a symbol table because a CFG cannot keep track of variable names (which would be context - the value of a variable depends on the context we are in.)

b) What kind of information is typically stored in a symbol table?

Type information, memory offsets, number of args

c) List pros and cons of three different data structures

Unordered list (simple array)

- Pros: Simple implementation. Fast O(1) inserts.
- Slow. O(N) lookup (Slowest for symbols belonging to the outermost scope, which are at the end of the list.. Impractical in practice.

Binary Search tree

- Pros: Average insert and lookup: O(log n)
- Cons: Possibly very unbalanced in practice. Worst-case O(n). Can be improved by using balanced BBST

Hash table

- Pros: Insert & Lookup: average O(1), in practice O(1).
- Cons: Can be slightly complicated to implement.

Problem 2: Symbol Tables and blocks

a) Symbol tables as stack of tables

Each table is a stack frame. Stack grows downward.

Position 1

Name	Offset	Type
a	-4	int
b	-8	float

Name	Offset	Type
b	-12	bool

Position 2

Name	Offset	Type
a	-4	int
b	-8	float

Name	Offset	Type
b	-12	int
c	-16	float

Name	Offset	Type
a	-20	bool
c	-24	int

b) Symbol tables as single table

Position 1

Depth	Name	Offset	Type
0	a	-4	int
0	b	-8	float
1	b	-12	bool

Position 2

Depth	Name	Offset	Type
0	a	-4	int
0	b	-8	float
1	b	-12	int
1	c	-16	float
2	a	-20	bool
2	c	-24	int

c) Advatages and disadvantages of each approach

Stack of symbol tables

Advantage: Scoping is handled implicitly by the stack of symbol tables. Easy implementation.

Disadvantage: Slow lookup for symbols in outermost scopes, because we have to scan the intermediate scopes before reaching them. In practice most lookups happen at either the innermost or the outermost scope, giving this disadvantage a relatively big impact.

One big symbol table

Advantage: Fast search for any symbol (binary search).

Disadvantage: We must handle scoping "manually". The table has to cope with multiple entries of the same symbol. Often solved by storing depth as a field in the entry.

Problem 3: Type Checking

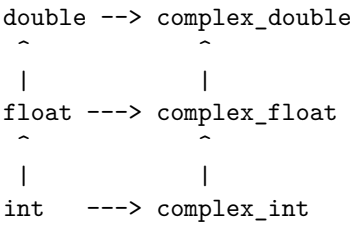
a) What is the difference between type synthesis and type inference?

> Type checking can take on two forms: synthesis and inference. Type synthesis builds up the type of an expression from the types of its subexpressions. (…) Type inference determines the type of a language construct from the way it is used.
> - Dragon book p.410

An example of type synthesis is when declaring variables: `int a = 3`. The symbol a is of type int, by synthesis.

An example of type inference is when a variable is used: `a + 3`. The type of this expression is int, because `a` is an int and `3` is (or can be, depending on language) an int.

b) FORTRAN type hierarchy



Assumptions: The assumptions i have made should be fairly evident from the diagram. I have assumed that any cast following an arrow is allowed.

Problem 4: Types, SSDs

a) Complete the SDD, so that the size attribute of T stores the size of the type.

Production	Semantic rules
T ::= int	T.size = 4
T ::= float	T.size = 8
T ::= bool	T.size = 1
T ::= T_1[num]	T.size = T1.size * num.value
T ::= (L)	T.size = L.size
L ::= L_1,T	T.size = L1.size + T.size
L ::= T	L.size = T.size

b) Annotated parse trees:

Figure 1: Annotated parse tree for ‘int[4][3][5]

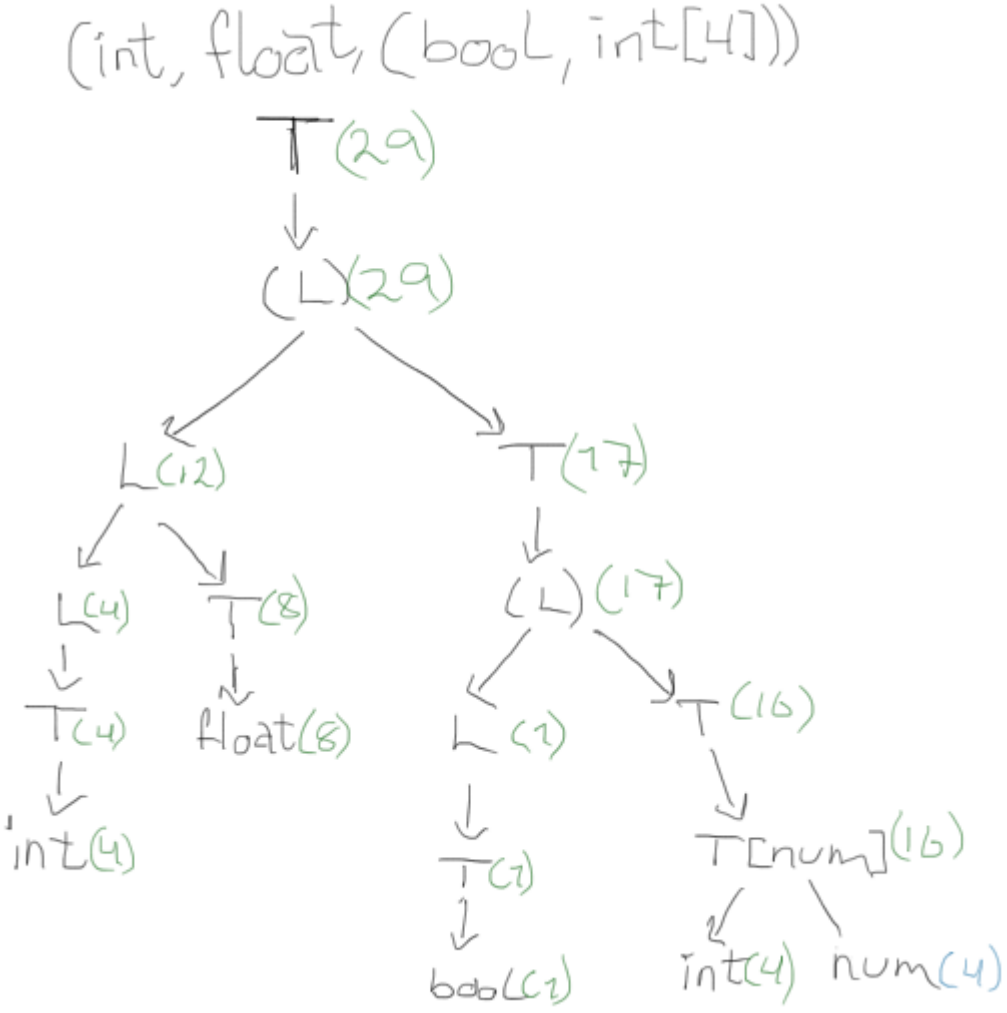


Figure 2: Annotated parse tree for `(int, float, (bool, int[4]))`