

Last Time

- ◆ **Entertaining C behaviors**
 - **Undefined**
 - **Implementation defined**
 - **Unspecified**
- ◆ **Operational semantics**
- ◆ **Norrish's C semantics**

Today

- ◆ **More C semantics**
- ◆ **New homework on the web – due next Tues**

Operational Semantics vs. Dataflow Analysis

- ◆ **What are the key differences?**

Order of Evaluation

- ◆ What can C print while evaluating this expression?
 - `printf("a")+printf("b")+printf("c")`
- ◆ All six orders are permitted
- ◆ To model this in the semantics:

$$\frac{e_1 \rightarrow e'_1}{e_1 + e_2 \rightarrow e'_1 + e_2}$$

$$\frac{e_2 \rightarrow e'_2}{e_1 + e_2 \rightarrow e_1 + e'_2}$$

$$\frac{}{v_1 + v_2 \rightarrow v_1 + v_2}$$

- ◆ **A C program changes memory through side effects**
- ◆ **Side effects are...**
 - **Assignment through =, +=, -=, etc.**
 - **++ and --**
 - **Accessing a volatile object**
 - **Calling a function that interacts with the external world**
 - **Usually through a system call**
 - **Calling a function that has a side effect**
- ◆ **Side effects do not occur immediately, but may be kept pending**
 - **Why would this seem like a good idea?**

- ◆ **Strategy for dealing with side effects in the C semantics:**
 - **Keep a multi-set of pending side effects**
 - **Empty the multi-set in non-deterministic order**
- ◆ **C defines certain sequence points**
 - **All side effects from code before the SP must fire before it**
 - **No side effects from code after the SP can fire before it**
- ◆ **Can think of sequence points as a language-level equivalent of bi-directional memory-system barriers**

Where are Sequence Points?

- ◆ Point of calling a function, after all arguments are evaluated
- ◆ End of evaluating the first operand to && or ||
- ◆ End of evaluating the first operand to ? :
- ◆ End of each operand to the comma operator
- ◆ Completing the evaluation of a full expression, defined as:
 - Evaluating an initializer
 - Expression in a regular statement terminated by a ;
 - Controlling expressions in do, while, switch, for
 - The other two expressions in a for
 - Expression in a return

Dealing with side effects...

- ◆ **Semantics keeps track of**
 - **Pending side effects**
 - **Parts of memory that have been updated**
 - **Parts of memory that have been referred to**
 - **All three of these are multi-sets**
- ◆ **How do these components get updated?**
 - **When a non-array lvalue becomes a value, a reference to the lvalue is added to update map**
 - **When a side effect is applied, it is removed from the set of pending side effects and added to the update map**
 - **When an assignment completes, a side effect is added to update the assigned location, and a reference to the assigned location is removed**

◆ C standard tells us that

- Any location may be updated only once between any pair of sequence points
- Cannot both refer to and update a location in between sequence points
- Violating these results in undefined behavior

◆ Which are defined?

- `v + v++`
- `*p + (i=1)`
- `(x=0) + (x=0)`
- `i = i + 1`
- `i = (i = i + 1)`

- ◆ **What do C's rules mean for you, the programmer?**
- ◆ **Easy approach: Make sure there's a sequence point between any two side effects**

Factorial

```
unsigned long fact (unsigned long n)
{
    unsigned long result = 1;
    while (n > 1) {
        result *= n;
        n--;
    }
    return result;
}
```

Factorial Specification

- ◆ $n! \leq \text{ULONG_MAX}$
- ◆ Stack space can be allocated for two long integers
- ◆ There exists a final program state where the return value of fact is $n!$

- ◆ **An improved version of Norrish's semantics is available**
- ◆ **Writing a translator from C AST to HOL would be a great project**
 - **CIL provides a C AST**

Summary