CSC 347 - Concepts of Programming Languages

Undefined Behavior

Instructor: James Riely

Learning Objectives

- What should happen if the language does not specify the meaning of some construct?
- What should happen when `x` is holding 8 bits, `x=7` and then `x = x+1`?

  - Identify undefined behavior in C

Datatype Bounds

- Natural numbers are only partly useful as integer arithmetic semantics

$$\frac{}{\langle \mathbf{n}, \xi \rangle \Downarrow \langle n, \xi \rangle}\ (\text{Num})$$

- Add integer bounds

$$\frac{-2^{63} \leq n \leq 2^{63} - 1}{\langle \mathbf{n}, \xi \rangle \Downarrow \langle n, \xi \rangle}\ (\text{Num})$$

- What to do with overflow?
  - Error: $\dfrac{\langle \mathbf{e}_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle \mathbf{e}_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle \quad \neg(-2^{63} \leq v_1 + v_2 \leq 2^{63} - 1)}{\langle \mathbf{e}_1 \texttt{+} \mathbf{e}_2, \xi_0 \rangle \Downarrow \langle \texttt{error}, \xi_2 \rangle}\ (\text{Add}), \ldots$

Under and Overflow

```c
#include <stdio.h>

int isMinValue (int x) {
  return (x-1) > x;
}
int main () {
  int i = -2000000000;
  while (!isMinValue(i))
    i--;
  printf ("Min value is %d\n", i);
}
```

```
$ gcc -O1 undefined.c && ./a.out
Min value is -2147483648
```

```
$ gcc -O2 undefined.c && ./a.out
^C #infinite loop

```

Order of Operations

- Recall rule for sequential composition
- Rule define an order of evaluating subexpressions

$$\frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{e}_1\,\mathtt{;}\,\mathtt{e}_2,\xi_0\rangle \Downarrow \langle v_2,\xi_2\rangle}\,(;) \qquad \frac{\langle \mathtt{e}_1,\xi_0\rangle \Downarrow \langle v_1,\xi_1\rangle \quad \langle \mathtt{e}_2,\xi_1\rangle \Downarrow \langle v_2,\xi_2\rangle}{\langle \mathtt{e}_1\mathtt{+}\mathtt{e}_2,\xi_0\rangle \Downarrow \langle v_1+v_2,\xi_2\rangle}\,(\text{Add})$$

  - by chaining the stores
  - ξ_0 is used in evaluating $\mathtt{e}_1$ and produces ξ_1
  - ξ_1 is used in evaluating $\mathtt{e}_2$ and produces ξ_2

Undefined Order of Operations

```c
#include <stdio.h>
int count = 0;

int f() { return ++count; }

int g() { return ++count >= 2 ? 5 : 3; }

int h(int a, int b) { return a+b; }

int main() {
  int x = h(f(), g());
  printf ("%d\n", x);

  int y = 3;
  y = (y += 1) + (y = y*y);
  printf ("%d\n", y);
}
```

```
$ clang -Wall undefined.c
undefined.c:13:21: warning:
unsequenced modification and
access to 'y' [-Wunsequenced]
  y = (y += 1) + (y = y*y);
          ~~          ^
1 warning generated.
$ ./a.out
6
20
```

- `a=f(), b=g(), a+b`
- `y=y+1; y=y+y*y`

Undefined Order of Operations

```c
#include <stdio.h>
int count = 0;
int f() { return ++count; }

int g() { return ++count >= 2 ? 5 : 3; }

int h(int a, int b) { return a+b; }

int main() {
  int x = h(f(), g());
  printf ("%d\n", x);

  int y = 3;
  y = (y += 1) + (y = y*y);
  printf ("%d\n", y);
}
```

```
$ gcc -Wall -O3 undefined.c
undefined.c:13:21: warning:
unsequenced modification and
access to 'y' [-Wunsequenced]
   y = (y += 1) + (y = y*y);
     ^
$ ./a.out
5
32
```

- `b=g(), a=f(), a+b`
- `y=y+1; y=y*y; y=y+y;`

Compiler Optimizations

- For undefined executions, the compiler can do what it likes
- This can lead to some surprising compiler optimizations
- C null pointer optimization 1

Summary

- Undefined behavior resolved in the compiler (compiler decides what is the meaning of undefined programs)
- Undefined behavior often presents security risks
- More examples of undefined behavior
- More undefined behavior
- Formal semantics can be analyzed for contradictions and completeness