CAL: Concise Animation Langarange

Jason (Tianliang) Sun, ts2825
Xinan Xu, xx2153
Jingyi Guo, jg3421

December 20, 2013

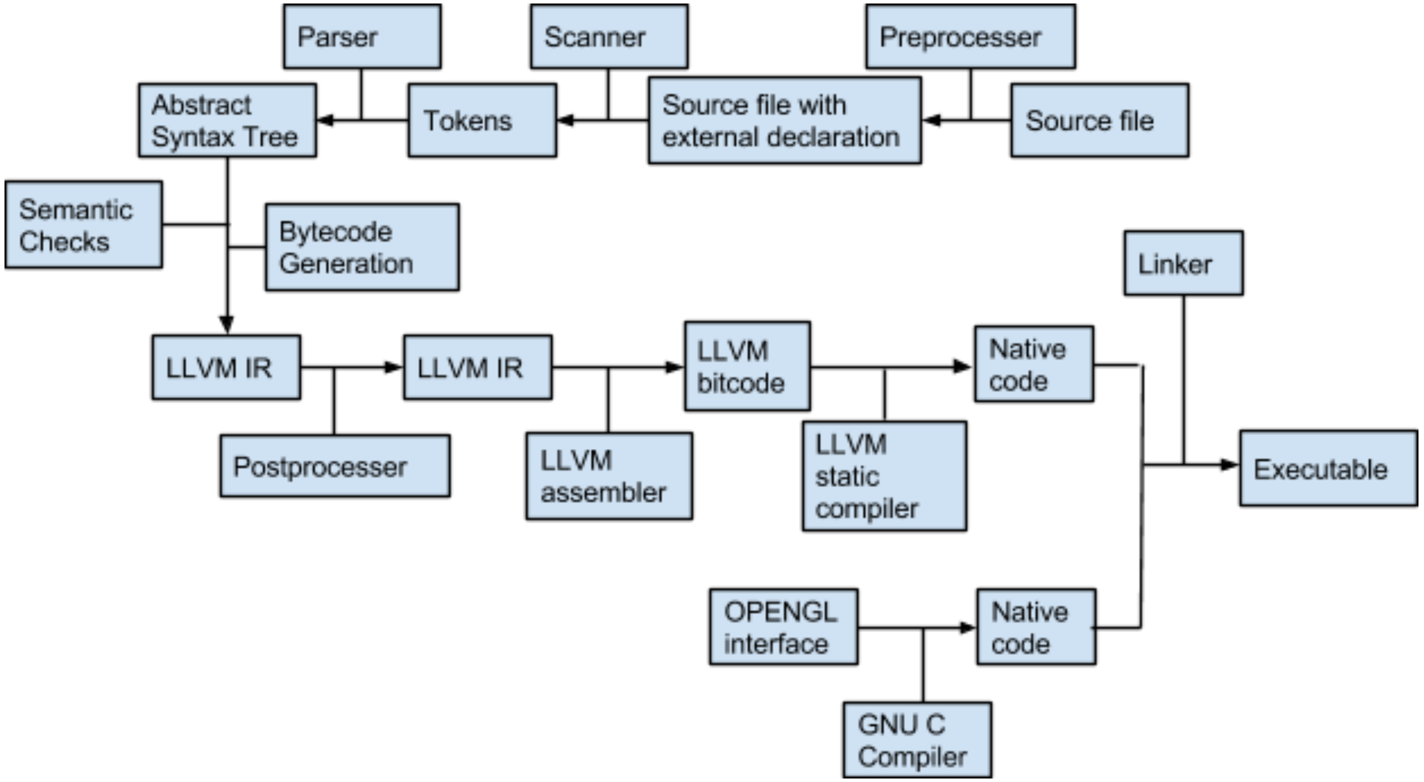
An introduction to CAL

- Overview
- Architectural Design
- Parsing and Abstract Syntax Tree
- Semantics Check
- Intermediate Representation
- OpenGL Interface
- Testing
- Challenges
- Demo

Overview

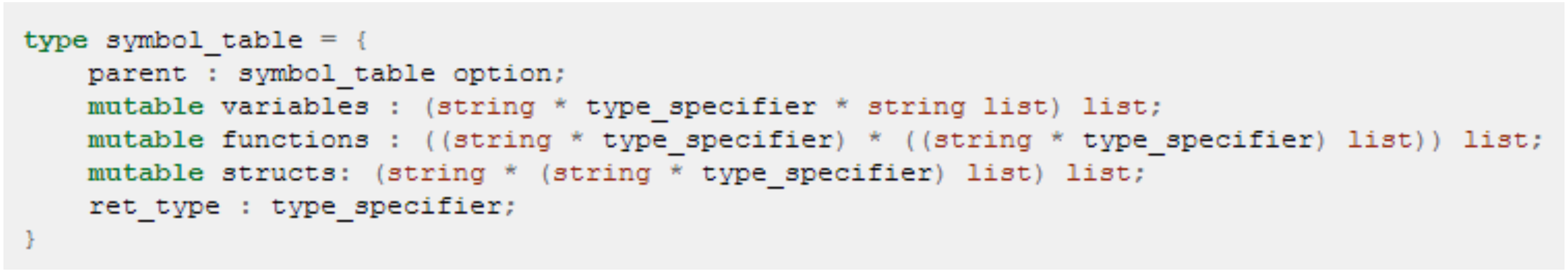
- CAL Compact
- C like syntax
- Scope
  - Global
  - Function
- Data Types
  - int, double, string,struct
  - self-defined: point,list,shape
- Recursion

Architectural Design



Parsing and Abstract Syntax Tree

- Similar to C’s syntax, but simpler.
  - No pointers/references, storage classes, bitwise operators, etc.
  - Support user defined structures, and language defined structures such as point and shape (image was intended)
- Program Structure
- Translation Environment



```
type symbol_table = {
    parent : symbol_table option;
    mutable variables : (string * type_specifier * string list) list;
    mutable functions : ((string * type_specifier) * ((string * type_specifier) list)) list;
    mutable structs: (string * (string * type_specifier) list) list;
    ret_type : type_specifier;
}
```

Semantics Check

- Types
  - arithmetic operators
  - assignment
  - function argument
- Declarations
  - variables, functions, structs
- References

Intermediate Representation

1. External function declaration
2. Global structure definition; global variable definition
3. Function definition
4. Function args allocation
5. Numeric Operators

```
%struct.foo = type { double, int, [100 x i8], [100 x
[100 x i32]] }
@size = global i32 10
```

```
define i32 @add_point_or_shape(i32 %x_val, i32
%y_val, %struct.point_or_shape* byval %pos)
nounwind
```

```
%x = alloca i32
%retval = alloca i32
```

```
%reg1 = load i32* %x, align 1
%reg2 = load i32* %y, align 1
%reg3 = icmp eq i32 %reg1, %reg2
```

6. Control Flow:

```
    %reg14 = icmp eq i32 %reg13, 1
    br i1 %reg14, label %v6bb2, label %v6bb1
v6bb1:
    store i32 0, i32* %reg5, align 1
    br label %v6bb3
v6bb2:
    store i32 1, i32* %reg5, align 1
    br label %v6bb3
v6bb3:
```

7. String "=" and "+="

```
%reg2 = bitcast [100 x i8]* %s0 to i8*
%reg3 = call i8* @strcat(i8* noalias %reg2, i8* noalias
getelementptr inbounds ([100 x i8]* @.str.main.1, i64 0,
i64 0)) nounwind
%reg4 = bitcast [100 x i8]* %s0 to i8*
%reg5 = call i8* @strcpy(i8* noalias %reg4, i8* noalias
getelementptr inbounds ([100 x i8]* @.str.main.3, i64 0,
i64 0)) nounwind
```

Intermediate Representation

8. Array/Struct dereference

```
%reg17 = getelementptr inbounds %struct.point_or_shape* %pos, i32 0, i32 1
```

9. Function call

```
%reg24 = call i32 @add_point(%struct.point* byval %reg23) nounwind
```

10. Implicit type conversion

```
%reg19 = load i32* @i, align 1
%reg20 = sitofp i32 %reg19 to double
%reg21 = fmul double %reg18, %reg20
```

```
let (s2,i2,t2) = llvm_expr (str,index,typ
expr env in
```

11. Struct assignment

```
%reg90 = bitcast %struct.point* %reg89 to i8*
call  void @llvm.memcpy.i64(i8* %reg90, i8* %reg88, i64 56, i32 1)
```

12. Truncation, zero extension
    (i32 <--> i1)

```
%reg16 = trunc i32 %reg15 to i1
br i1 %reg16, label %v16bb0, label %v16bb1
```

13. Function with struct return

```
define void @func(%struct.foo* noalias sret %agg.result, %struct.foo* byval %x) nounwind {
entry:
   %agg.result1 = bitcast %struct.foo* %agg.result to i8*
   %reg1 = bitcast %struct.foo* %x to i8*
   call  void @llvm.memcpy.i64(i8* %agg.result1, i8* %reg1, i64 116, i32 8)
   br label %return
return:
   ret void
}
```

OpenGL Interface

$$\begin{tikzcd}
\text{*-cal.ll} \arrow[d, "\text{*-cal.s}"] & \text{glsupport.h} \arrow[l, "\text{inclusion}"'] \arrow[r, "\text{inclusion}"] & \text{glsupport.c:} \arrow[dll, "\text{gl.o}"] \\
\text{Executable} & &
\end{tikzcd}$$

***-cal.ll**

```
declare void @llvm.memcpy.i64(i8* nocapture, i8*
nocapture, i64, i32) nounwind
declare i8* @strcpy(i8* noalias, i8* noalias)
nounwind
declare i8* @strcat(i8* noalias, i8* noalias)
nounwind
declare i32 @add_shape(%struct.shape* byval)
declare i32 @add_point(%struct.point* byval)
declare i32 @setup()
declare i32 @run()
declare i32 @pop_point()
declare i32 @pop_shape()
declare i32 @wait(double)
declare i32 @byebye()
```

glsupport.h

```
struct point {
  double x;
  double y;
  double r;
  double g;
  double b;
  double vx;
  double vy;
};
struct shape {
  double size;
  double x;
  double y;
  double r;
  double g;
  double b;
  double vx;
  double vy;
  double theta;
  double omega;
};
```

glsupport.c:

```
struct point pt_arr[MAX];
struct shape shape_arr[MAX];
int pop_point(){};
int pop_shape(){};
int add_point(struct point pt){};
int add_shape(struct shape shp){};
void* run(void*) {
  glutDisplayFunc(display);
}
int setup() {
  pthread_create(&th,NULL,run,NULL);
};
int wait(double seconds){};
int byebye() {};
```

Testing

- Unit testing
    - whitebox
    - automated
- Integration testing
    - blackbox

Challenges

- Time
    - We didn’t quite follow the standard SE approach…
- Putting things together
    - Defined interfaces

Demo Time!