A Brief Introduction to Using LLVM

Nick Sumner
Spring 2013

What is LLVM?

- A compiler?

What is LLVM?

- A compiler?
- A set of formats, libraries and tools.

What is LLVM?

- A compiler?

- A set of formats, libraries and tools.
  - A simple, typed IR (*bitcode*)
  - Program analysis / optimization libraries
  - Machine code generation libraries
  - Tools that compose the libraries to perform task

What is LLVM Bitcode?

- A (Relatively) Simple IR

```
#include<stdio.h>

void
foo(unsigned e) {
  for (unsigned i = 0; i < e; ++i) {
    printf("Hello\n");
  }
}

int
main(int argc, char **argv) {
  foo(argc);
  return 0;
}
```

Code

clang -c -emit-llvm
(and llvm-dis)

```
@str = private constant [6 x i8] c"Hello\00"

define void @foo(i32 %e) {
  %1 = icmp eq i32 %e, 0
  br i1 %1, label %._crit_edge, label %.lr.ph

.lr.ph:                                ; preds = %.lr.ph, %0
  %i = phi i32 [ %2, %.lr.ph ], [ 0, %0 ]
  %str1 = getelementptr
                    [6 x i8]* @str, i64 0, i64 0
  %puts = tail call i32 @puts(i8* %str1)
  %2 = add i32 %i, 1
  %cond = icmp eq i32 %2, %e
  br i1 %cond, label %.exit, label %.lr.ph

.exit:                                 ; preds = %.lr.ph, %0
  ret void
}

define i32 @main(i32 %argc, i8** %argv) {
  tail call void @foo(i32 %argc)
  ret i32 0
}
```

IR

What is LLVM Bitcode?

- A (Relatively) Simple IR

```c
#include<stdio.h>

void
foo(unsigned e) {
  for (unsigned i = 0; i < e; ++i) {
    printf("Hello\n");
  }
}

int
main(int argc, char **argv) {
  foo(argc);
  return 0;
}
```

clang -c -emit-llvm
(and llvm-dis)

```
@str = private constant [6 x i8] c"Hello\00"

define void @foo(i32 %e) {
  %1 = icmp eq i32 %e, 0
  br i1 %1, label %._crit_edge, label %.lr.ph

.lr.ph:                                ; preds = %.lr.ph, %0
  %i = phi i32 [ %2, %.lr.ph ], [ 0, %0 ]
  %str1 = getelementptr
                    [6 x i8]* @str, i64 0, i64 0
  %puts = tail call i32 @puts(i8* %str1)
  %2 = add i32 %i, 1
  %cond = icmp eq i32 %2, %e
  br i1 %cond, label %.exit, label %.lr.ph

.exit:                                 ; preds = %.lr.ph, %0
  ret void
}

define i32 @main(i32 %argc, i8** %argv) {
  tail call void @foo(i32 %argc)
  ret i32 0
}
```

What is LLVM Bitcode?

- A (Relatively) Simple IR

```
#include<stdio.h>

void
foo(unsigned e) {
  for (unsigned i = 0; i < e; ++i) {
    printf("Hello\n");
  }
}

int
main(int argc, char **argv) {
  foo(argc);
  return 0;
}
```

Functions

```
@str = private constant [6 x i8] c"Hello\00"

define void @foo(i32 %e) {
  %1 = icmp eq i32 %e, 0
  br i1 %1, label %._crit_edge, label %.lr.ph

.lr.ph:                                ; preds = %.lr.ph, %0
  %i = phi i32 [ %2, %.lr.ph ], [ 0, %0 ]
  %str1 = getelementptr
                    [6 x i8]* @str, i64 0, i64 0
  %puts = tail call i32 @puts(i8* %str1)
  %2 = add i32 %i, 1
  %cond = icmp eq i32 %2, %e
  br i1 %cond, label %.exit, label %.lr.ph

.exit:                                 ; preds = %.lr.ph, %0
  ret void
}

define i32 @main(i32 %argc, i8** %argv) {
  tail call void @foo(i32 %argc)
  ret i32 0
}
```

What is LLVM Bitcode?

- A (Relatively) Simple IR

```
#include<stdio.h>

void
foo(unsigned e) {
  for (unsigned i = 0; i < e; ++i) {
    printf("Hello\n");
  }
}

int
main(int argc, char **argv) {
  foo(argc);
  return 0;
}
```

Basic Blocks

```
@str = private constant [6 x i8] c"Hello\00"

define void @foo(i32 %e) {
  %1 = icmp eq i32 %e, 0
  br i1 %1, label %._crit_edge, label %.lr.ph

.lr.ph:                                ; preds = %.lr.ph, %0
  %i = phi i32 [ %2, %.lr.ph ], [ 0, %0 ]
  %str1 = getelementptr
                    [6 x i8]* @str, i64 0, i64 0
  %puts = tail call i32 @puts(i8* %str1)
  %2 = add i32 %i, 1
  %cond = icmp eq i32 %2, %e
  br i1 %cond, label %.exit, label %.lr.ph

.exit:                                 ; preds = %.lr.ph, %0
  ret void
}

define i32 @main(i32 %argc, i8** %argv) {
  tail call void @foo(i32 %argc)
  ret i32 0
}
```

What is LLVM Bitcode?

- A (Relatively) Simple IR

```
#include<stdio.h>

void
foo(unsigned e) {
  for (unsigned i = 0; i < e; ++i) {
    printf("Hello\n");
  }
}

int
main(int argc, char **argv) {
  foo(argc);
  return 0;
}
```

```
@str = private constant [6 x i8] c"Hello\00"

define void @foo(i32 %e) {
  %1 = icmp eq i32 %e, 0
  br i1 %1, label %._crit_edge, label %.lr.ph

.lr.ph:                                ; preds = %.lr.ph, %0
  %i = phi i32 [ %2, %.lr.ph ], [ 0, %0 ]
  %str1 = getelementptr
                    [6 x i8]* @str, i64 0, i64 0
  call i32 @puts(i8* %str1)
  %2 = add i32 %i, 1
  %cond = icmp eq i32 %2, %e
  br i1 %cond, label %.exit, label %.lr.ph

.exit:                                 ; preds = %.lr.ph, %0
  ret void
}

define i32 @main(i32 %argc, i8** %argv) {
  tail call void @foo(i32 %argc)
  ret i32 0
}
```

labels & predecessors

Basic Blocks

What is LLVM Bitcode?

- A (Relatively) Simple IR

```
#include<stdio.h>

void
foo(unsigned e) {
  for (unsigned i = 0; i < e; ++i) {
    printf("Hello\n");
  }
}

int
main(int argc, char **argv) {
  foo(argc);
  return 0;
}
```

```
@str = private constant [6 x i8] c"Hello\00"

define void @foo(i32 %e) {
  %1 = icmp eq i32 %e, 0
  br i1 %1, label %._crit_edge, label %.lr.ph

.lr.ph:                                ; preds = %.lr.ph, %0
  %i = phi i32 [ %2, %.lr.ph ], [ 0, %0 ]
  %str1 = getelementptr
                          [6 x i8]* @str, i64 0, i64 0
          call i32 @puts(i8* %str1)
  %2 = add i32 %i, 1
  %cond = icmp eq i32 %2, %e
  br i1 %cond, label %.exit, label %.lr.ph

.exit:                                 ; preds = %.lr.ph, %0
  ret void
}

define i32 @main(i32 %argc, i8** %argv) {
  tail call void @foo(i32 %argc)
  ret i32 0
}
```

branches & successors

Basic Blocks

What is LLVM Bitcode?

- A (Relatively) Simple IR

```
#include<stdio.h>

void
foo(unsigned e) {
  for (unsigned i = 0; i < e; ++i) {
    printf("Hello\n");
  }
}

int
main(int argc, char **argv) {
  foo(argc);
  return 0;
}
```

Instructions

```
@str = private constant [6 x i8] c"Hello\00"

define void @foo(i32 %e) {
  %1 = icmp eq i32 %e, 0
  br i1 %1, label %._crit_edge, label %.lr.ph

.lr.ph:                                ; preds = %.lr.ph, %0
  %i = phi i32 [ %2, %.lr.ph ], [ 0, %0 ]
  %str1 = getelementptr
                    [6 x i8]* @str, i64 0, i64 0
  %puts = tail call i32 @puts(i8* %str1)
  %2 = add i32 %i, 1
  %cond = icmp eq i32 %2, %e
  br i1 %cond, label %.exit, label %.lr.ph

.exit:                                 ; preds = %.lr.ph, %0
  ret void
}

define i32 @main(i32 %argc, i8** %argv) {
  tail call void @foo(i32 %argc)
  ret i32 0
}
```

Dealing with SSA

- You may ask where certain values came from
  - Useful for tracking dependencies
  - “Where was this variable defined?”

Dealing with SSA

- You may ask where certain values came from
- LLVM IR is in SSA form
  - How many acronyms can I fit into one line?
  - What does this mean?
  - Why does it matter?

Dealing with SSA

- You may ask where certain values came from
- LLVM IR is in SSA form
  - How many acronyms can I fit into one line?
  - What does this mean?
  - Why does it matter?

```
void foo()
  unsigned i = 0;
  while (i < 10) {
    i = i + 1;
  }
}
```

Dealing with SSA

- You may ask where certain values came from
- LLVM IR is in SSA form
  - How many acronyms can I fit into one line?
  - What does this mean?
  - Why does it matter?

```
void foo()
  unsigned i = 0;
  while (i < 10) {
    i = i + 1;
  }
}
```

What is the single definition of `i` at this point?

Dealing with SSA

- Thus the phi instruction
  - It selects which of the definitions to use
  - Always at the start of a basic block

Dealing with SSA

- Thus the phi instruction
  - It selects which of the definitions to use
  - Always at the start of a basic block

```
void foo()
  unsigned i = 0;
  while (i < 10) {
    i = i + 1;
  }
}
```

```
define void @foo() {
  br label %1

; <label>:1
  %i.phi = phi i32 [ 0, %0 ], [ %2, %1 ]
  %2 = add i32 %i.phi, 1
  %exitcond = icmp eq i32 %2, 10
  br i1 %exitcond, label %3, label %1

; <label>:3
  ret void
}
```

Dealing with SSA

- Thus the phi instruction
  - It selects which of the definitions to use
  - Always at the start of a basic block

```
void foo()
  unsigned i = 0;
  while (i < 10) {
    i = i + 1;
  }
}
```

```
define void @foo() {
  br label %1

; <label>:1
  %i.phi = phi i32 [ 0, %0 ], [ %2, %1 ]
  %2 = add i32 %i.phi, 1
  %exitcond = icmp eq i32 %2, 10
  br i1 %exitcond, label %3, label %1

; <label>:3
  ret void
}
```