Compiler construction 4: Extending compiler

Danny Hendrix
Harm Berntsen

Radboud University Nijmegen

Recap

- Language: Haskell
- Higher order functions
- Simple garbage collection (explained in block 3)
- Compile to LLVM

Overview

Sample.spl

```
a id(a a) { return a; }

[b] map (a->b f, [a] l)
{
    if(isEmpty(l))
                return [];
    return f(l.hd) :
                map(f, l.tl);
}

Void main() {
    [Int] ll = 5 : 6 : 11;
    print(8+id(12));
    print(ll.tl);
}
```

LALR

Parser

Type checker

SSM Code gen.

LLVM Code gen.

Pretty print / error report

Info About the function 'id'
Error When analyzing the statement return b;
Error line 1 position 20 Identifier with id b not found

Info About the function 'main'
Info About variable ll
Error Operator type mismatch, Int : Int not allowed
Error line 10 position 24 Type Int does not match expected type@[10:22]: [Int]

./testPrograms/presentatie.spl

```
1   b id(b a)
2   {
3       return b;
4   }
5   [b] map(a->b f, [a] l)
6   {
7       if(isEmpty(l))
8           return [];
9       return (f(l.hd) : map(f, l.tl));
10  }
11  Void main()
12  {
13      [Int] ll = (5 : (6 : 11));
14      print((12 + id(9)));
15      print(ll.tl);
16  }
```

Type Int does not match expected type@[10:22]: [Int]

```
; ModuleID = 'spl module'

@.printfarg = private unnamed_addr constant [4 x i8] c"%i\0A\00"

declare i32 @printf(i8* nocapture, i32)

declare i8* @malloc(i32)

define i32 @main() {
entry:
  %0 = call i32 @gcd(i32 13, i32 377)
  %1 = inttoptr i32 %0 to i8*
  %2 = ptrtoint i8* %1 to i32
  %3 = call i32 @printf(i8* getelementptr inbounds ([4 x i8]* @.printfarg, i32 0, i32 0), i32 %2)
  %4 = call i32 @gcdr(i32 13, i32 377)
  %5 = inttoptr i32 %4 to i8*
  %6 = ptrtoint i8* %5 to i32
  %7 = call i32 @printf(i8* getelementptr inbounds ([4 x i8]* @.printfarg, i32 0, i32 0), i32 %6)
  ret i32 0
}

define private i32 @gcd(i32 %a, i32 %b) {
entry:
  %0 = alloca i32
  store i32 %a, i32* %0
  %1 = alloca i32
  store i32 %b, i32* %1
  %c = alloca i32
  store i32 0, i32* %c
  br label %while.cond

while.cond:                                       ; preds = %while.loop, %entry
  %2 = load i32* %0
  %3 = icmp ne i32 %2, 0
  br i1 %3, label %while.loop, label %while.exit

while.loop:                                       ; preds = %while.cond
  %4 = load i32* %0
```

```
    LDR HP
    LDC 1
    ADD
    STR HP
    LDR SP
    LDC 2000
    STA 0
    BRA _prog_main
    HALT
_prog_main:
    NOP
    BSR main
    HALT
main:
;Void main()
;{
;   [Int] ll = (5 : (6 : 11));
;   Int b = (True + 5);
;   print((12 + id(9)));
;   print(ll.tl);
;}
    LDC -1
    STS -2
    ANNOTE SP -1 -1 green "-#args-1 main"
    LINK 0
;(5 : (6 : 11))
;(6 : 11)
;11
    LDC 11
    ANNOTE SP 0 0 darkGray "11"
;6
    LDC 6
    ANNOTE SP 0 0 darkGray "6"
;begin halloc memory block
    LDC 3
    ANNOTE SP 0 0 green "halloc total block size"
    BSR halloc
    AJS -1
```

Higher order functions

- Extended grammar

```
Bool foo(Int a, Int b){}        Int->Int->Bool
```

- Example:

```
1   a id(a a)
2   {
3       return a;
4   }
5   [b] mapReverse([b] acc, a->b f, [a] l)
6   {
7       if(isEmpty(l))
8           return acc;
9       acc = (f(l.hd) : acc);
10      return mapReverse(acc, f, l.tl);
11  }
12  Int main()
13  {
14      [Int]->[Int] mapId = mapReverse([], id);
15      [Int] ll = mapId((3 : (5 : [])));
16      [Bool] bl = mapReverse([], id, (True : (False : [])));
17      return 0;
18  }
```

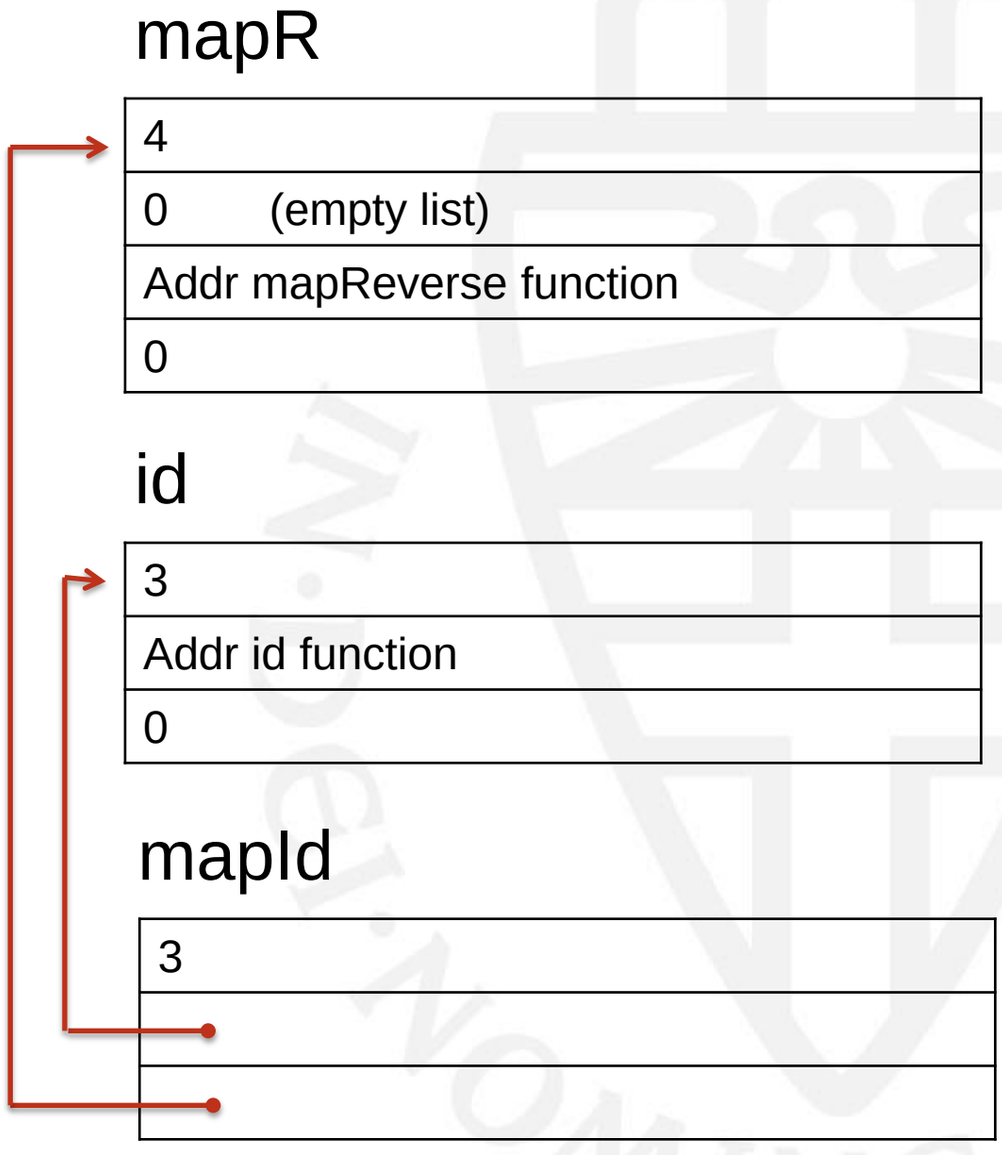
Higher order functions – Heap blocks

Object header (= length of block)
Last additional supplied argument
…
First additional supplied argument
[Function pointer]
Pointer to previous arguments, 0 if none

Higher order functions – Heap blocks

```
(Int->Int)->[Int]->[Int] mapR = mapReverse([]);
[Int]->[Int] mapId = mapR(id);
[Int] ll = mapId((3 : (5 : [])));
```

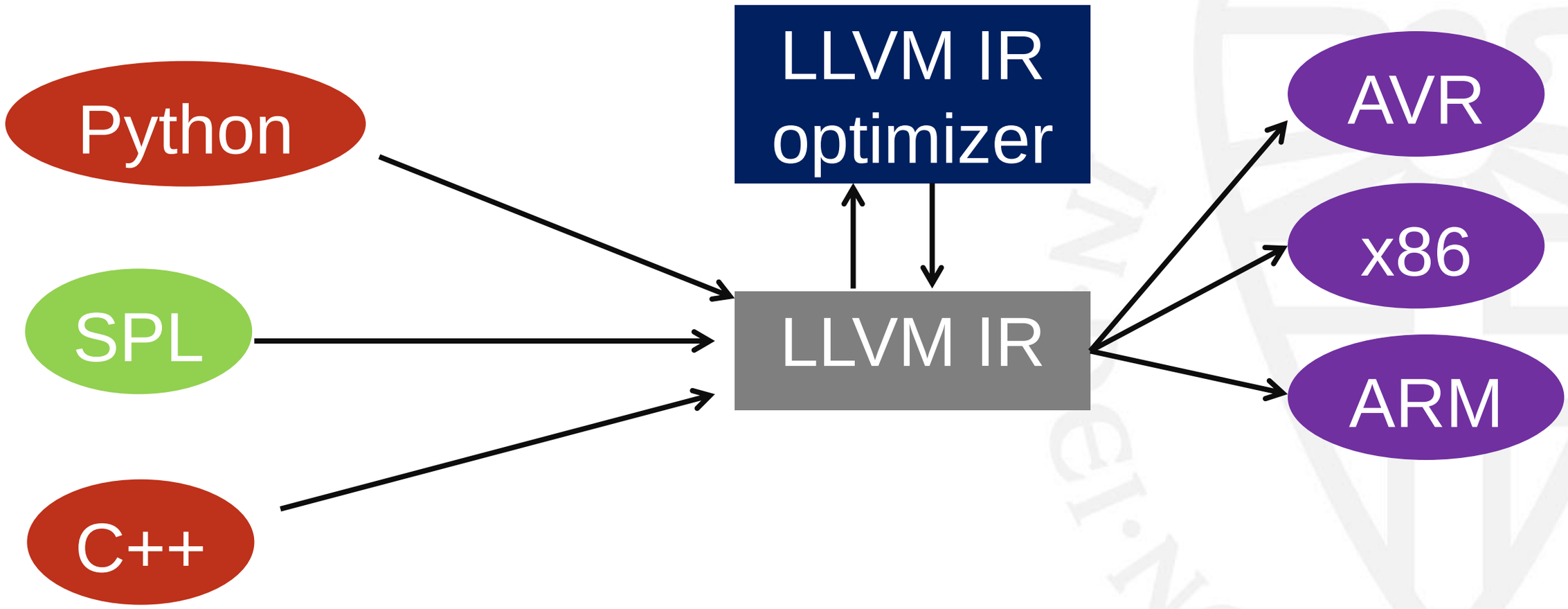
Object header (= length of block)
Last additional supplied argument
…
First additional supplied argument
[Function pointer]
Pointer to previous arguments, 0 if none



Higher order functions – Calling

- Recursively load all arguments from blocks to stack
- Last block contains pointer to function
- Function stores the length of the arguments on the stack
- When function completed, remove the arguments from stack

LLVM



LLVM – Intermediate representation

- Low level assembly language
- Types
  - Integers :: i1, i32, i12, i67 …
  - Array :: [4 x float]
  - …
- Unlimited registers
- Single assignment

LLVM - example

```
Int add(Int a, Int b)
{
    return (a + b);
}
Int main()
{
    Int res = add(1, 2);
    if((res == 3))
    {
        print(res);
    }
    else
    {
        print(-1);
    }
    return 0;
}
```

```
@.printfarg = private unnamed_addr constant [4 x i8] c"%i\0A\00"
declare i32 @printf(i8* nocapture, i32)

define i32 @main() {
entry:
  %res = alloca i32
  %0 = call i32 @add(i32 1, i32 2)
  store i32 %0, i32* %res
  %1 = load i32* %res
  %2 = icmp eq i32 %1, 3
  br i1 %2, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  %3 = load i32* %res
  %4 = inttoptr i32 %3 to i8*
  %5 = ptrtoint i8* %4 to i32
  %6 = call i32 @printf(i8*
        getelementptr inbounds ([4 x i8]* @.printfarg, i32 0, i32 0),
    i32 %5)
  br label %exit

if.else:                                          ; preds = %entry
  %7 = call i32 @printf(i8*
        getelementptr inbounds ([4 x i8]* @.printfarg, i32 0, i32 0),
    i32 -1)
  br label %exit

exit:                                             ; preds = %if.else, %if.then
  ret i32 0
}

define private i32 @add(i32 %a, i32 %b) {
entry:
  %0 = alloca i32
  store i32 %a, i32* %0
  %1 = alloca i32
  store i32 %b, i32* %1
  %2 = load i32* %0
  %3 = load i32* %1
  %4 = add i32 %2, %3
  ret i32 %4
}
```

LLVM - optimizer

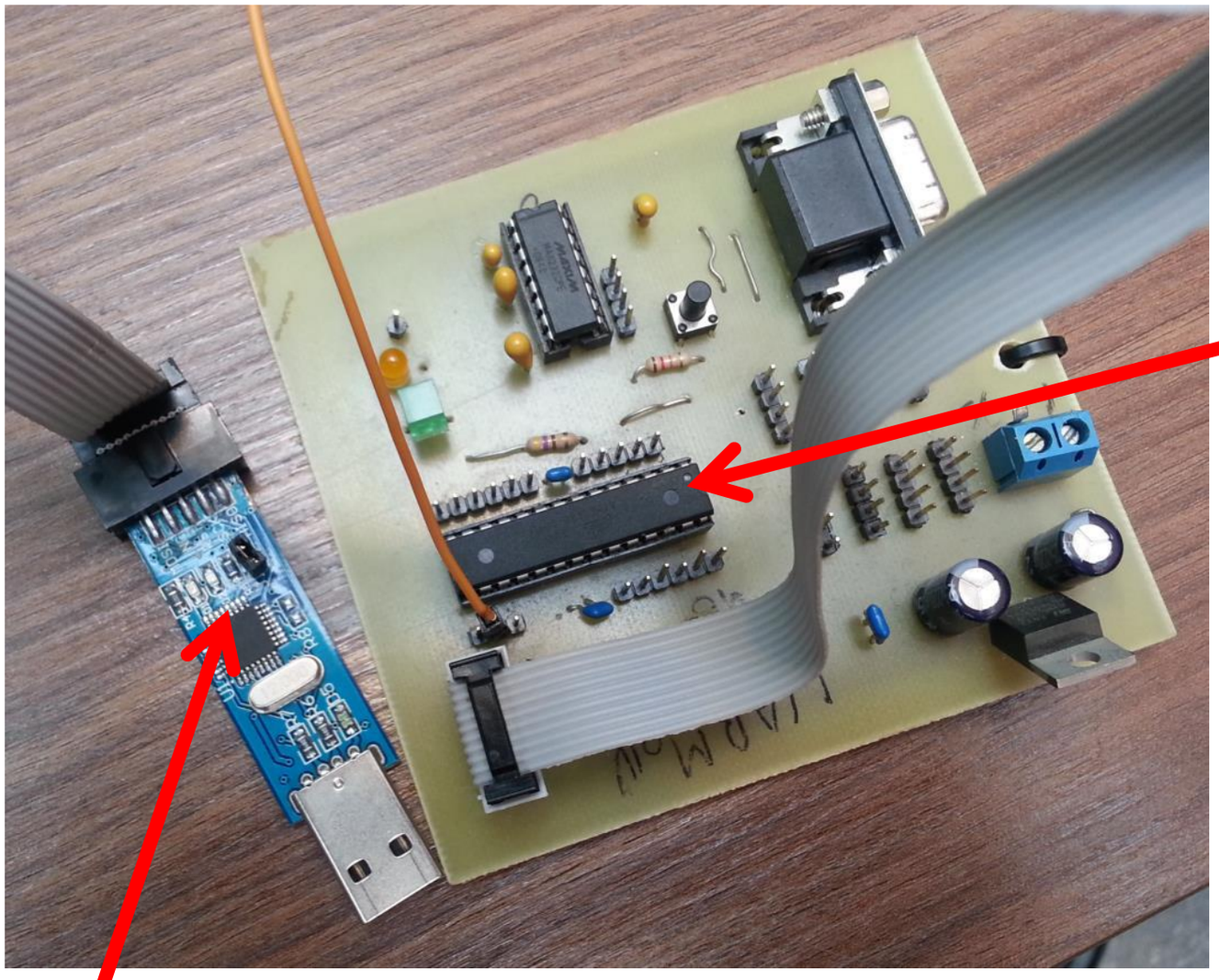
```
Int gcd(Int a, Int b)
{
    Int c = 0;
    while((a != 0))
    {
        c = a;
        a = (b % a);
        b = c;
    }
    return b;
}
Int gcdr(Int a, Int b)
{
    if((a == 0))
        return b;
    return gcdr((b % a), a);
}
Int main()
{
    print(gcd(13, 377));
    print(gcdr(13, 377));
    return 0;
}
```

```
@.printfarg = private unnamed_addr constant [4 x i8] c"%i\0A\00"

declare i32 @printf(i8* nocapture, i32)

define i32 @main() {
gcdr.exit:
  %0 = tail call i32 @printf(i8*
    getelementptr inbounds ([4 x i8]* @.printfarg, i32 0, i32 0),
    i32 13)
  %1 = tail call i32 @printf(i8*
    getelementptr inbounds ([4 x i8]* @.printfarg, i32 0, i32 0),
    i32 13)
  ret i32 0
}
```

LLVM – in practice



ATMEL ATMEGA 88

Programmer

Problems

- Type checking of higher order polymorphic functions
- LLVM types
- AVR not an official target

Questions