Compilers and the Graph Coloring Problem

Andrew Parmet
September 16, 2016

Overview

- About me
- Brief overview of compilers
- Register allocation
  - Interference graphs
  - NP completeness and Kempe's algorithm
- Demo

About me

- Cornell Physics ‘16
- Avid ultimate player
- Cajun food connoisseur

Brief Overview of Compilers

- A compiler is a tool for abstraction
- We rely on them constantly
- Some compilers:
  - Oracle's javac
  - GNU's gcc
  - Google's GWT
  - Ocsigen's js_of_ocaml

Stages of Compilation

1. Preprocessing
2. Compilation (code generation)
3. Assembling
4. Linking

Stages of Compilation (Code Generation Step)

Lexical Analysis	Sequence of lexical tokens
Parsing	Abstract syntax tree (AST)
Semantic Analysis	Annotated syntax tree (type-checked)
Intermediate Code Generation	Intermediate representation (IR) tree
Optimization of IR	Intermediate representation (IR) tree
Assembly Generation	Assembly program

Assembly Generation

1. Tiling the IR tree (instruction selection)
    - Greedy algorithms vs. dynamic programming
2. Liveness analysis
    - Worklists and interference graph generation
3. Register allocation

Assembly Generation (specifically, register allocation)

- The sixteen x86-64 registers:

  rax, rbx, rcx, rdx, rsi, rdi, rbp, rsp, r8, r9, r10, ..., r15

- The challenge: get all variables into registers when they are needed by assembly instructions without clobbering other needed values

First: Tiling and Pseudo-Assembly

Assembly with temp names preserved

```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (MOVE (TEMP _arg_i25) (CONST 48))
   (EXP (CALL (NAME _I_alloc_i) (TEMP _arg_i25)))
   (MOVE (TEMP _array2) (TEMP _RET0))
   (MOVE (MEM (TEMP _array2)) (CONST 5))
   (MOVE (TEMP _array2) (ADD (CONST 8) (TEMP _array2)))
   (MOVE (MEM (ADD (CONST 0) (TEMP _array2))) (CONST 72))

...
```

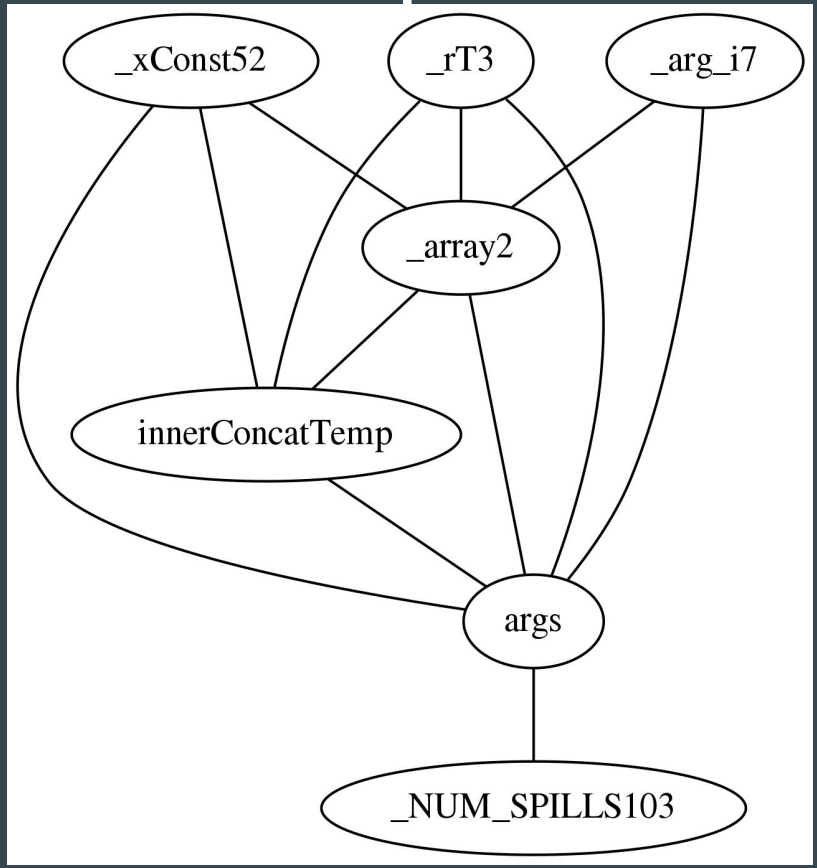
```
    .globl _Imain_paai
_Imain_paai:
    pushq rbp
    pushq rbx
    pushq r12
    pushq r13
    pushq r14
    pushq r15
    movq rsp, rbp
    subq _NUM_SPILLS103, rsp
    movq rdi, args
    movq $48, _xConst52
    movq _xConst52, _arg_i7
    movq _arg_i7, rdi
    call _I_alloc_i
    movq rax, _array2
    movq _array2, _rT3
    movq $5, (_rT3)
    addq $8, _array2
    movq _array2, _rT8
    movq $72, (_rT8)
    movq $101, 8(_rT8)
    movq $108, 16(_rT8)

...
```

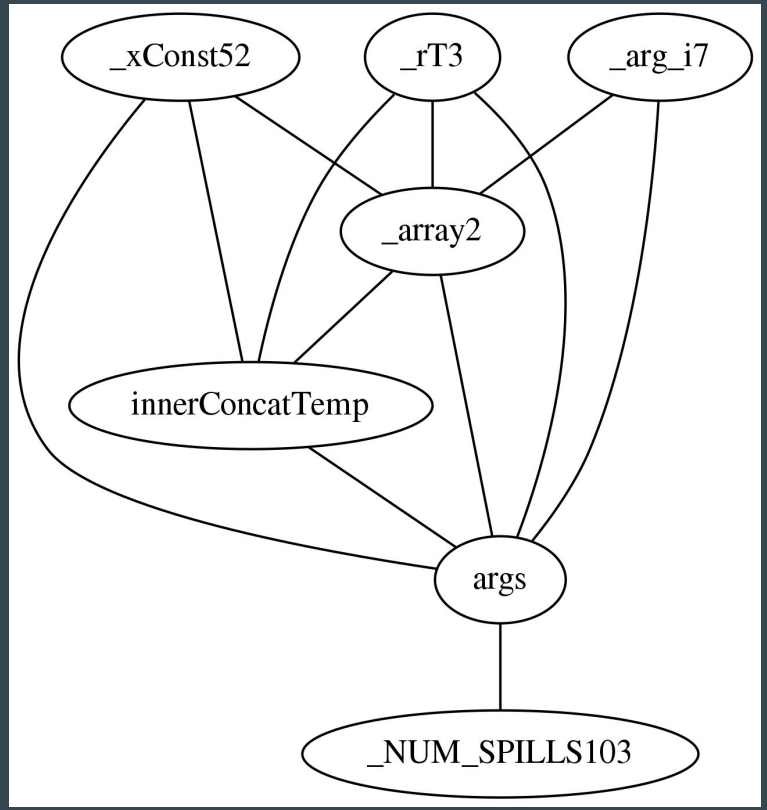
Second: Liveness Ranges and Interference Graph

```
    .globl _Imain_paai
_Imain_paai:
    pushq rbp
    pushq rbx
    pushq r12
    pushq r13
    pushq r14
    pushq r15
    movq rsp, rbp
    subq _NUM_SPILLS103, rsp
    movq rdi, args
    movq $48, _xConst52
    movq _xConst52, _arg_i7
    movq _arg_i7, rdi
    call _I_alloc_i
    movq rax, _array2
    movq _array2, _rT3
    movq $5, (_rT3)
    addq $8, _array2

...
```



Third: Graph Coloring



?

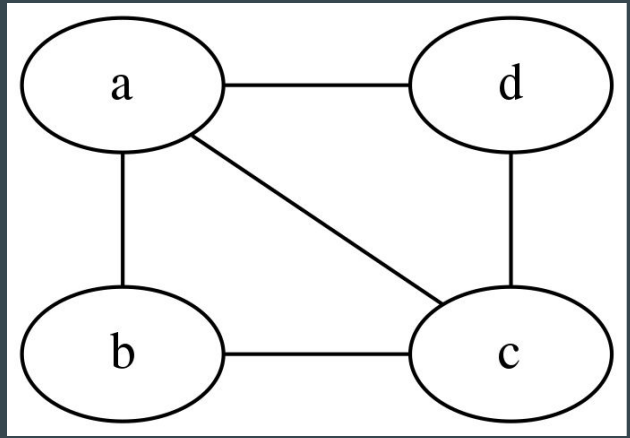
NP-Complete!

Kempe's Algorithm for a k-coloring (1879)

Key observation: Given a graph G that contains a node n with degree less than k, the graph is k-colorable if and only if G with n removed is k-colorable.

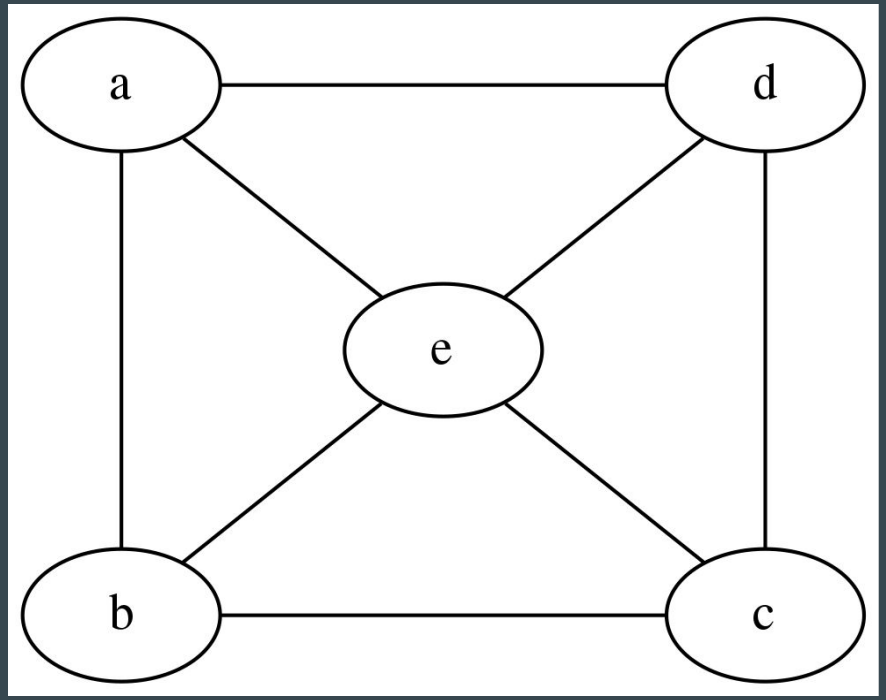
This is inductively true.

Try iteratively removing nodes with degree < k.
If all nodes are removed, then G is definitely k-colorable



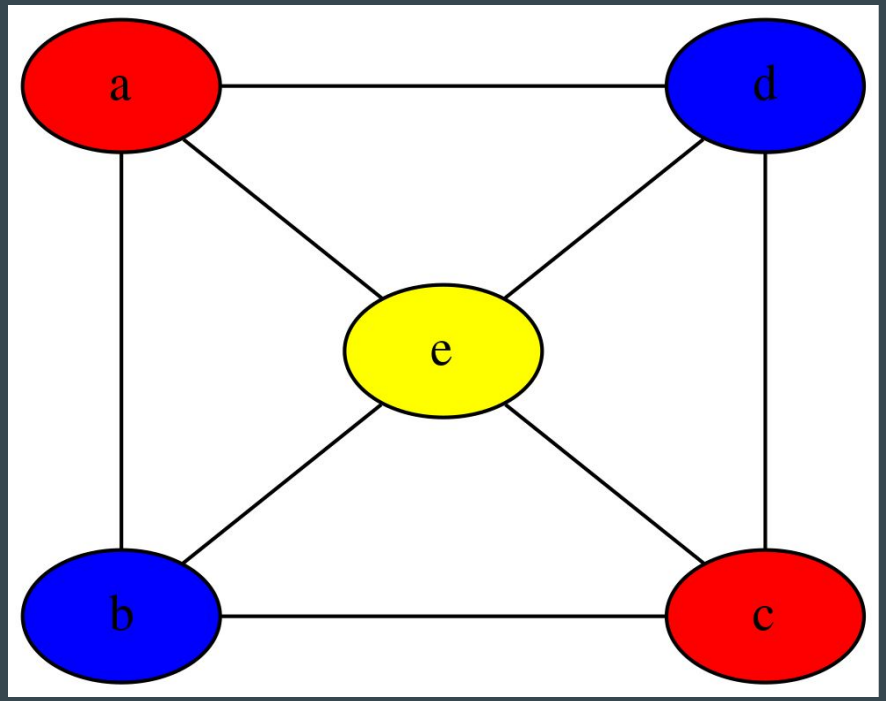
k = 3

Kempe’s Algorithm for a k-coloring (1879)



k = 3?

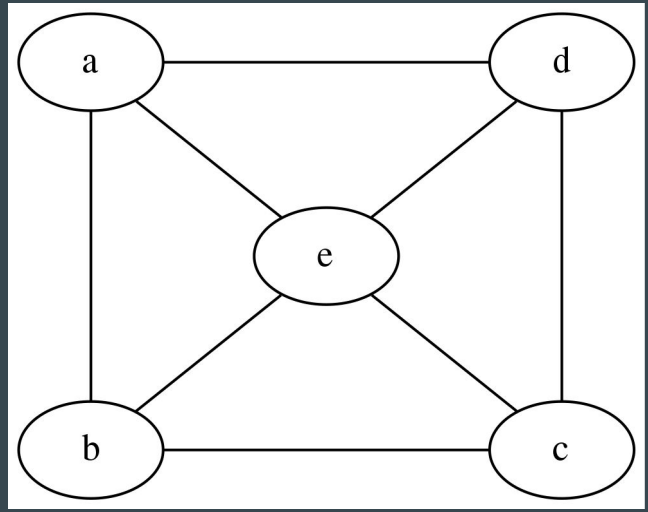
Kempe’s Algorithm for a k-coloring (1879)



k = 3?

Kempe's Algorithm for a k-coloring (1879)

- Remove each node of degree < k, pushing onto a stack.
- (Best case) If all are removed, rebuild the graph, coloring as we go.
- **If we get stuck (i.e., no nodes of degree < k), remove any high-degree node and continue.**



Chaitin's Allocator, 1981

- **Build:** construct interference graph
- **Simplify:** remove nodes (Kempe)
- **Spill:** when necessary, remove high-degree nodes and mark as potential spills
- **Select:** rebuild graph, coloring as we go. If a potential spill can't be colored, mark it as an actual spill and continue
- **Start over:** if there are spills, write spill code and start over

Demo: hello.xi

```
use io

main(args : int[][]) {
    a:int[] = "Hello";
    b:int[] = "world!";
    println(a + ", " + b);
}
```

Conclusion

- Allocating registers is hard - and compilers have to do it quickly

- Luckily Kempe's algorithm performs well in practice

- It's thanks to good allocation that we can use computers to compile and allow us to think using higher-level abstractions

References

Cornell's CS 4120: Introduction to Compilers, taught by Andrew Myers

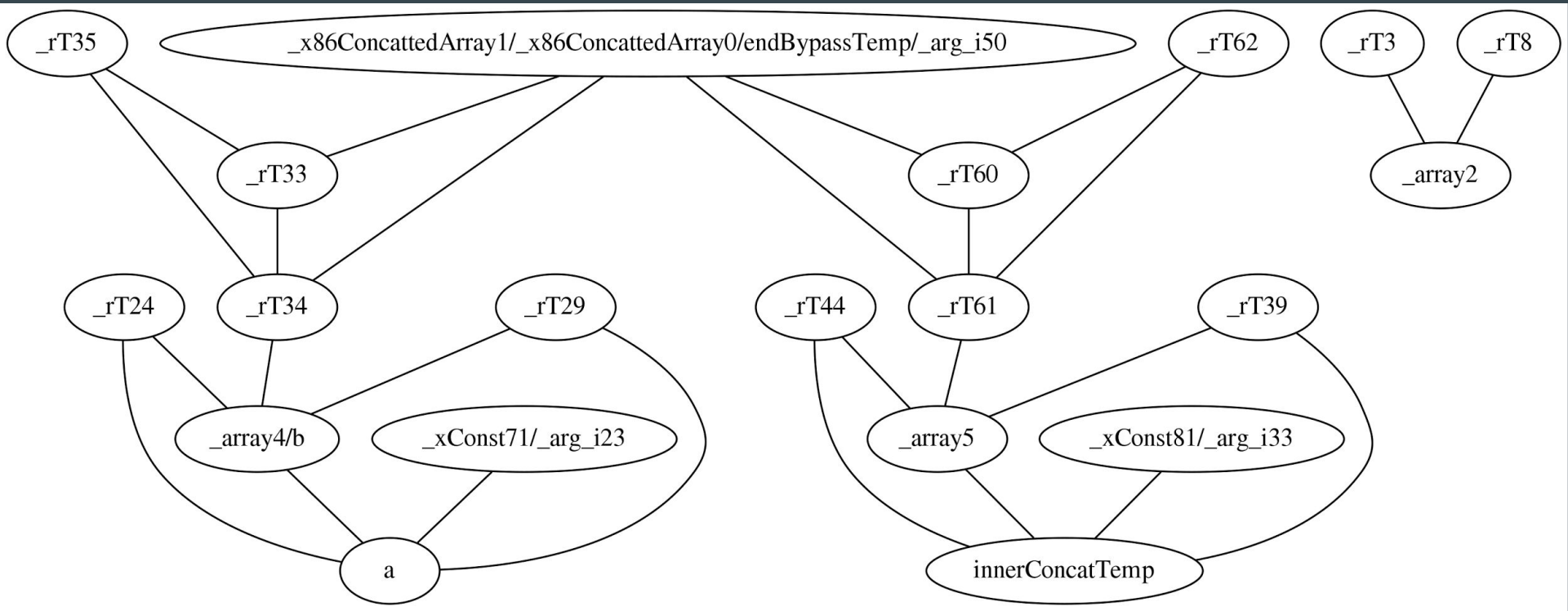
Peter Lee's compilers course at CMU:
http://www.cs.cmu.edu/afs/cs/academic/class/15745-s06/web/handouts/10.pdf

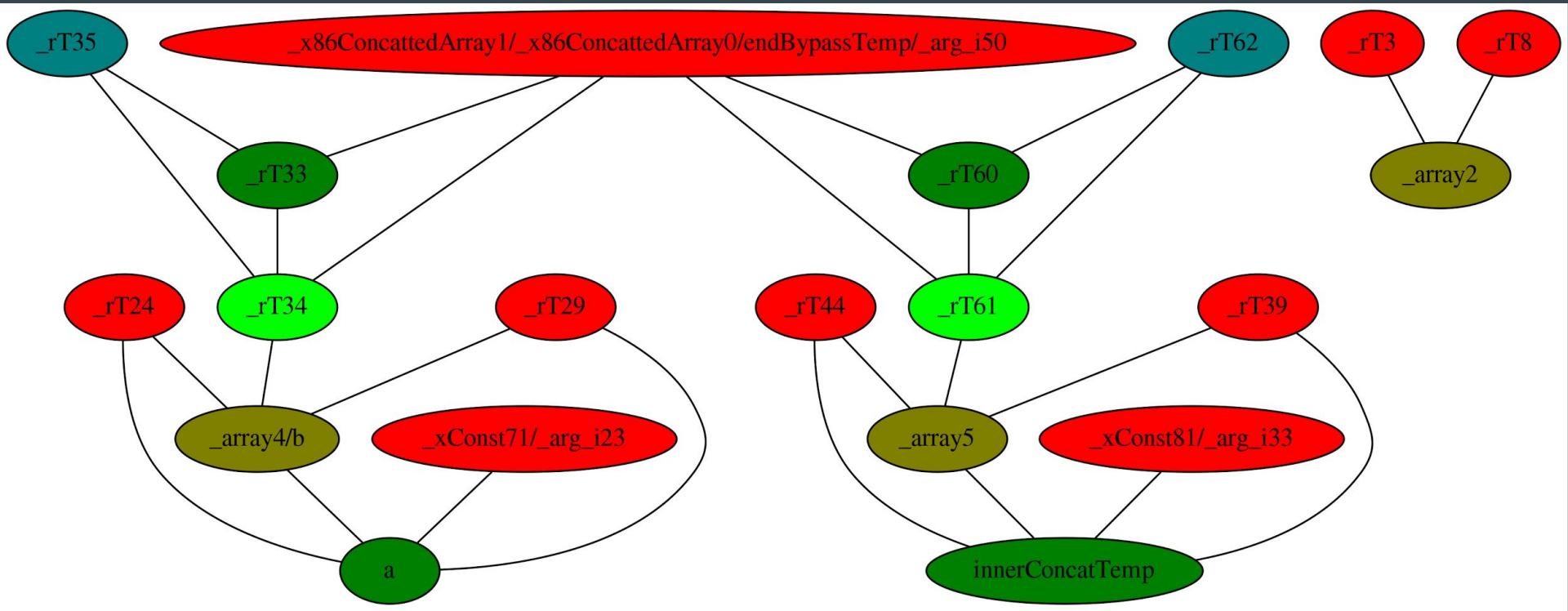
Appel's *Modern Compiler Implementation in Java*

Inria OCaml Compilers page:
http://caml.inria.fr/pub/docs/oreilly-book/html/book-ora065.html

Constructive generation of very hard 3-colorability instances, Mizuno & Nishihara (2006)

Questions?


_rT35
_x86ConcattedArray1/_x86ConcattedArray0/endBypassTemp/_arg_i50
_rT62
_rT3
_rT8
_rT33
_rT60
_array2
_rT24
_rT34
_rT29
_rT44
_rT61
_rT39
_array4/b
_xConst71/_arg_i23
_array5
_xConst81/_arg_i33
a
innerConcatTemp


_rT35
_x86ConcattedArray1/_x86ConcattedArray0/endBypassTemp/_arg_i50
_rT62
_rT3
_rT8
_rT33
_rT60
_array2
_rT24
_rT34
_rT29
_rT44
_rT61
_rT39
_array4/b
_xConst71/_arg_i23
_array5
_xConst81/_arg_i33
a
innerConcatTemp

A Baby Xi Program: hello.xi

```
use io

main(args : int[][]) {
    a:int[] = "Hello";
    b:int[] = "world!";
    print(a + ", " + b);
}
```

Lexical Analysis

```
use io

main(args : int[][]) {
   a:int[] = "Hello";
   b:int[] = "world!";
   print(a + ", " + b);
}
```

```
1:1 use
1:5 id io
3:1 id main
3:5 (
3:6 id args
3:11 :
3:13 int
3:16 [
3:17 ]
3:18 [
3:19 ]
3:20 )
3:22 {
4:4 id a

...
```

Parsing

```
1:1 use
1:5 id io
3:1 id main
3:5 (
3:6 id args
3:11 :
3:13 int
3:16 [
3:17 ]
3:18 [

...
```

```
use io

main(args : int[][]) {
   a:int[] = "Hello";
   b:int[] = "world!";
   print(a + ", " + b);
}
```

```
(((use io))
 ((main
   ((args ([] ([] int))))
   ()
   ((= (a ([] int)) "Hello")
    (= (b ([] int)) "world!")
    (print (+ (+ a ", ") b))))))
```

Semantic Analysis
(Type checking)

```
use io

main(args : int[][]) {
   a:int[] = "Hello";
   b:int[] = "world!";
   print(a + ", " + b);
}
```

```
(((use io))
 ((main
   ((args ([] ([] int))))
   ()
   ((= (a ([] int)) "Hello")
    (= (b ([] int)) "world!")
    (print (+ (+ a ", ") b))))))
```

Valid Xi Program

IR Generation

(Intermediate Representation)

```
(((use io))
 ((main
   ((args ([] ([] int))))
   ()
   ((= (a ([] int)) "Hello")
    (= (b ([] int)) "world!")
    (print (+ (+ a ", ") b))))))
```

```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (SEQ
    (MOVE
     (ESEQ (SEQ) (TEMP a))
     (ESEQ
      (SEQ
       (MOVE (TEMP _array149) (CALL (NAME _I_alloc_i) (CONST 48)))
       (MOVE (MEM (TEMP _array149)) (CONST 5))
       (MOVE (TEMP _array149) (ADD (CONST 8) (TEMP _array149)))
       (MOVE (MEM (ADD (CONST 0) (TEMP _array149))) (CONST 72))
       (MOVE (MEM (ADD (CONST 8) (TEMP _array149))) (CONST 101))
       (MOVE (MEM (ADD (CONST 16) (TEMP _array149))) (CONST 108))
       (MOVE (MEM (ADD (CONST 24) (TEMP _array149))) (CONST 108))
       (MOVE (MEM (ADD (CONST 32) (TEMP _array149))) (CONST 111)))
      (TEMP _array149)))
...
```

IR Generation

(Intermediate Representation)

```
(((use io))
 ((main
   ((args ([] ([] int))))
   ()
   ((= (a ([] int)) "Hello")
    (= (b ([] int)) "world!")
    (print (+ (+ a ", ") b))))))
```

```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (SEQ
    (MOVE
     (ESEQ (SEQ) (TEMP a))
     (ESEQ
      (SEQ
       (MOVE (TEMP _array149) (CALL (NAME _I_alloc_i) (CONST 48)))
       (MOVE (MEM (TEMP _array149)) (CONST 5))
       (MOVE (TEMP _array149) (ADD (CONST 8) (TEMP _array149)))
       (MOVE (MEM (ADD (CONST 0) (TEMP _array149))) (CONST 72))
       (MOVE (MEM (ADD (CONST 8) (TEMP _array149))) (CONST 101))
       (MOVE (MEM (ADD (CONST 16) (TEMP _array149))) (CONST 108))
       (MOVE (MEM (ADD (CONST 24) (TEMP _array149))) (CONST 108))
       (MOVE (MEM (ADD (CONST 32) (TEMP _array149))) (CONST 111)))
      (TEMP _array149)))
...
```

IR Generation
(Intermediate Representation)

```
(((use io))
 ((main
   ((args ([] ([] int))))
   ()
   ((= (a ([] int)) "Hello")
    (= (b ([] int)) "world!")
    (print (+ (+ a ", ") b))))))
```

```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (SEQ
    (MOVE
     (ESEQ (SEQ) (TEMP a))
     (ESEQ
      (SEQ
       (MOVE (TEMP _array149) (CALL (NAME _I_alloc_i) (CONST 48)))
       (MOVE (MEM (TEMP _array149)) (CONST 5))
       (MOVE (TEMP _array149) (ADD (CONST 8) (TEMP _array149)))
       (MOVE (MEM (ADD (CONST 0) (TEMP _array149))) (CONST 72))
       (MOVE (MEM (ADD (CONST 8) (TEMP _array149))) (CONST 101))
       (MOVE (MEM (ADD (CONST 16) (TEMP _array149))) (CONST 108))
       (MOVE (MEM (ADD (CONST 24) (TEMP _array149))) (CONST 108))
       (MOVE (MEM (ADD (CONST 32) (TEMP _array149))) (CONST 111)))
      (TEMP _array149)))
...
```

IR Lowering

```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (SEQ
    (MOVE
     (ESEQ (SEQ) (TEMP a))
     (ESEQ
      (SEQ
       (MOVE (TEMP _array149)
...
```

```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (MOVE (TEMP _arg_i25) (CONST 48))
   (EXP (CALL (NAME _I_alloc_i) (TEMP _arg_i25)))
   (MOVE (TEMP _array2) (TEMP _RET0))
   (MOVE (MEM (TEMP _array2)) (CONST 5))
   (MOVE (TEMP _array2) (ADD (CONST 8) (TEMP _array2)))
   (MOVE (MEM (ADD (CONST 0) (TEMP _array2))) (CONST 72))
...
```

IR Lowering

```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (SEQ
    (MOVE
     (ESEQ (SEQ) (TEMP a))
     (ESEQ
      (SEQ
       (MOVE (TEMP _array149)
...
```

```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (MOVE (TEMP _arg_i25) (CONST 48))
   (EXP (CALL (NAME _I_alloc_i) (TEMP _arg_i25)))
   (MOVE (TEMP _array2) (TEMP _RET0))
   (MOVE (MEM (TEMP _array2)) (CONST 5))
   (MOVE (TEMP _array2) (ADD (CONST 8) (TEMP _array2)))
   (MOVE (MEM (ADD (CONST 0) (TEMP _array2))) (CONST 72))
...
```

IR Lowering

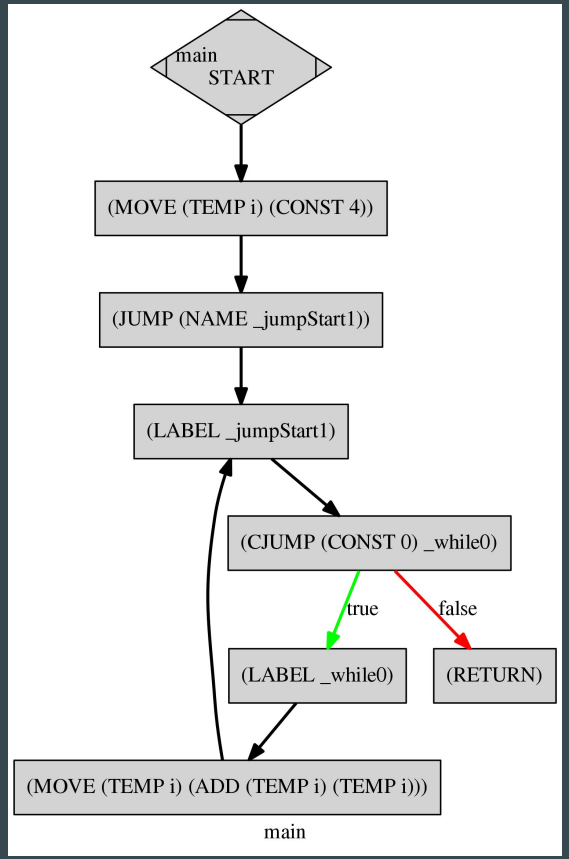
```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (SEQ
    (MOVE
     (ESEQ (SEQ) (TEMP a))
     (ESEQ
      (SEQ
       (MOVE (TEMP _array149)
...
```

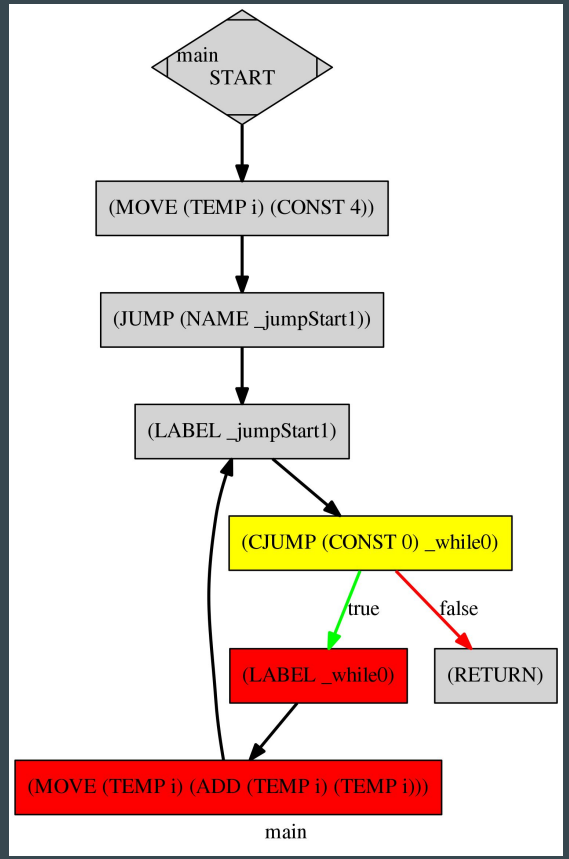
```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (MOVE (TEMP _arg_i25) (CONST 48))
   (EXP (CALL (NAME _I_alloc_i) (TEMP _arg_i25)))
   (MOVE (TEMP _array2) (TEMP _RET0))
   (MOVE (MEM (TEMP _array2)) (CONST 5))
   (MOVE (TEMP _array2) (ADD (CONST 8) (TEMP _array2)))
   (MOVE (MEM (ADD (CONST 0) (TEMP _array2))) (CONST 72))
...
```

IR Optimization

Example: Unreachable code elimination

```
main() {
  i:int = 4;
  while (i == 1) {
    i = i + i;
  }
}
```





Assembly Generation

```
(COMPUNIT
 hello
 (FUNC
  _Imain_paai
  (SEQ
   (MOVE (TEMP args) (TEMP _ARG0))
   (MOVE (TEMP _arg_i25) (CONST 48))
   (EXP (CALL (NAME _I_alloc_i) (TEMP _arg_i25)))
   (MOVE (TEMP _array2) (TEMP _RET0))
   (MOVE (MEM (TEMP _array2)) (CONST 5))
   (MOVE (TEMP _array2) (ADD (CONST 8) (TEMP _array2)))
   (MOVE (MEM (ADD (CONST 0) (TEMP _array2))) (CONST 72))

...
```

```
  .globl _Imain_paai
_Imain_paai:
    pushq %rbp
    pushq %rbx
    pushq %r13
    pushq %r14
    pushq %r15
    movq %rsp, %rbp
    movq %rdi, %r10
    movq $48, %r10
    movq %r10, %rdi
    call _I_alloc_i
    movq %rax, %r11
    movq %r11, %r10
    movq $5, (%r10)
    addq $8, %r11
    movq %r11, %r10
    movq $72, (%r10)
    movq $101, 8(%r10)
    movq $108, 16(%r10)

...
```