Magi Language Compiler

Zhijian LIU

Shanghai Jiao Tong University

May 13, 2016

The Gift of the Magi

The Gift of the Magi is a short story, written by O. Henry, about a young married couple and how they deal with the challenge of buying secret Christmas gifts for each other with very little money. As a sentimental story with a moral lesson about gift-giving, it has been a popular one for adaptation, especially for presentation at Christmas time. The plot and its "twist ending" are well-known, and the ending is generally considered an example of comic irony.

Project Overview

- Time period: from **Mar 22** to **May 13**
- Git commits: **172** commits
- Java code length: **13485** lines

Highlights

- Use a well-organized framework with good extensibility
- Feed back a user-friendly compilation error message
- Support most features of **OOP** including member function, private modifier, class inheritance, member initialization and constructor function
- Achieve an outstanding compilation quality
- Implement the **SSA** form and do some optimizations on it including the useless code elimination and dominator-based value numbering technique
- Do not use any peephole and data-oriented optimizations
- Use the **global** register allocation and improve the algorithm
- Make good use of the version control system

Highlights

- Use a well-organized framework with good extensibility
- Feed back a user-friendly compilation error message
- Support most features of **OOP** including member function, private modifier, class inheritance, member initialization and constructor function
- Achieve an outstanding compilation quality
- Implement the **SSA** form and do some optimizations on it including the useless code elimination and dominator-based value numbering technique
- Do not use any peephole and data-oriented optimizations
- Use the **global** register allocation and improve the algorithm
- Make good use of the version control system

Demonstration: Compilation Error Message

- `1:31: the function "angry" cannot have two parameters named "haha"!`
- `2:4: "haha" is not a symbol name!`
- `6:10: the number of parameters in the function-call expression is wrong!`
- `7:12: two int/string-type expressions are expected in the addition expression!`

Highlights

- Use a well-organized framework with good extensibility
- Feed back a user-friendly compilation error message
- Support most features of **OOP** including member function, private modifier, class inheritance, member initialization and constructor function
- Achieve an outstanding compilation quality
- Implement the **SSA** form and do some optimizations on it including the useless code elimination and dominator-based value numbering technique
- Do not use any peephole and data-oriented optimizations
- Use the **global** register allocation and improve the algorithm
- Make good use of the version control system

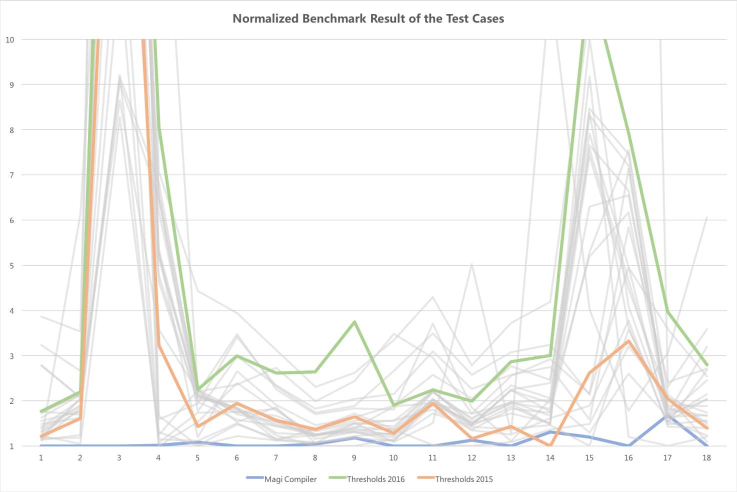
Demonstration: Object Oriented Programming

```
class Pair {
    public int first = 0;
    public int second = 0;
    public bool equal = true;

    public Pair() {
        this.first = this.second = 0;
        this.equal = true;
    }

    public Pair(int first, int second) {
        this.first = first;
        this.second = second;
        this.equal = (first == second);
    }

    public void print() {
        println(toString(this.first) + " " + toString(this.second));
    }
}

class Triple extends Pair {
    public int third = 0;
    public bool equal = true;

    public Triple() {
        this.first = this.second = this.third = 0;
        this.equal = true;
    }

    public Triple(int first, int second, int third) {
        this.first = first;
        this.second = second;
        this.third = third;
        this.equal = (first == second && second == third);
    }

    public void print() {
        println(toString(this.first) + " " + toString(this.second) + " " + toString(this.third));
    }
}

int main() {
    Pair pair = new Pair(1, 2);
    pair.print();
    Triple triple = new Triple(1, 2, 3);
    triple.print();
    return 0;
}
```

Highlights

- Use a well-organized framework with good extensibility
- Feed back a user-friendly compilation error message
- Support most features of **OOP** including member function, private modifier, class inheritance, member initialization and constructor function
- Achieve an outstanding compilation quality
- Implement the **SSA** form and do some optimizations on it including the useless code elimination and dominator-based value numbering technique
- Do not use any peephole and data-oriented optimizations
- Use the **global** register allocation and improve the algorithm
- Make good use of the version control system

Demonstration: Normalized Benchmark Result



Highlights

- Use a well-organized framework with good extensibility
- Feed back a user-friendly compilation error message
- Support most features of **OOP** including member function, private modifier, class inheritance, member initialization and constructor function
- Achieve an outstanding compilation quality
- Implement the **SSA** form and do some optimizations on it including the useless code elimination and dominator-based value numbering technique
- Do not use any peephole and data-oriented optimizations
- Use the **global** register allocation and improve the algorithm
- Make good use of the version control system

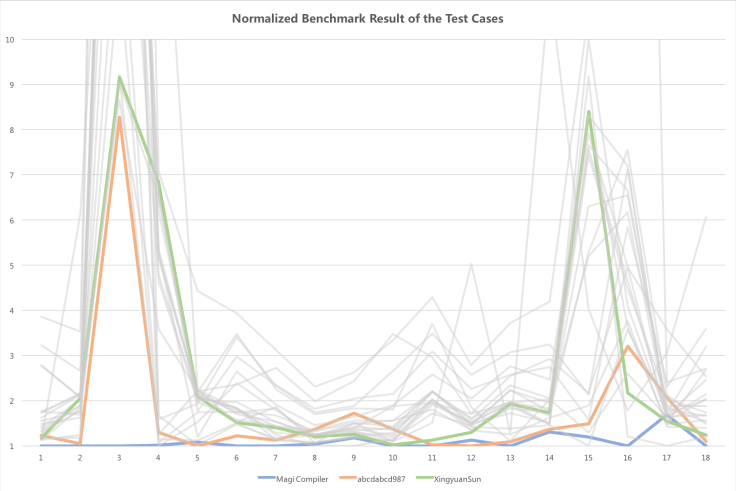
Demonstration: Single Static Assignment

```
func min $p0 $p1 {
    %min.0.enter:
        $t4 = move $p1
        $t5 = move $p0
        jump %min.1.entry

    %min.1.entry:
        $t3 = slt $t5 $t4
        br $t3 %min.2.if_true %min.3.if_false

    %min.2.if_true:
        $t2 = move $t5
        jump %min.4.if_merge

    %min.4.if_merge:
        ret $t2
        jump %min.5.exit

    %min.3.if_false:
        $t2 = move $t4
        jump %min.4.if_merge

    %min.5.exit:
        $p1 = move $t4
        $p0 = move $t5
}
```

Demonstration: Single Static Assignment

```
func min $p0.0 $p1.0 {
    %min.0.enter:
        $t4.0 = move $p1.0
        $t5.0 = move $p0.0
        jump %min.1.entry

    %min.1.entry:
        $t3.0 = slt $t5.0 $t4.0
        br $t3.0 %min.2.if_true %min.3.if_false

    %min.2.if_true:
        $t2.0 = move $t5.0
        jump %min.4.if_merge

    %min.4.if_merge:
        $t2.1 = phi %min.2.if_true $t2.0 %min.3.if_false $t2.2
        ret $t2.1
        jump %min.5.exit

    %min.3.if_false:
        $t2.2 = move $t4.0
        jump %min.4.if_merge

    %min.5.exit:
        $p1.1 = move $t4.0
        $p0.1 = move $t5.0
}
```

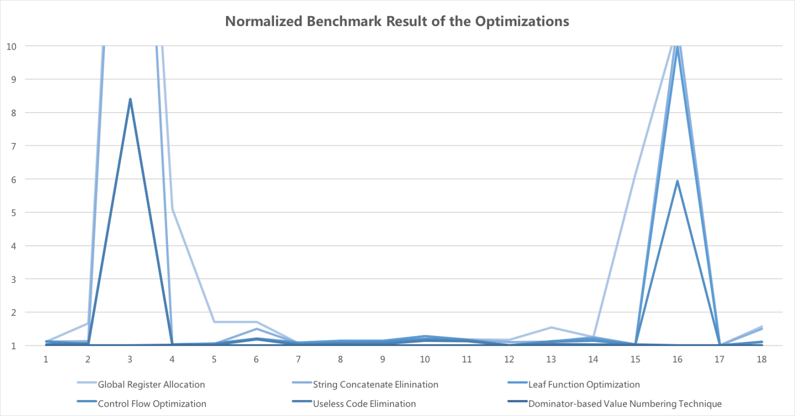
Demonstration: Normalized Benchmark Result



Highlights

- Use a well-organized framework with good extensibility
- Feed back a user-friendly compilation error message
- Support most features of **OOP** including member function, private modifier, class inheritance, member initialization and constructor function
- Achieve an outstanding compilation quality
- Implement the **SSA** form and do some optimizations on it including the useless code elimination and dominator-based value numbering technique
- Do not use any peephole and data-oriented optimizations
- Use the **global** register allocation and improve the algorithm
- Make good use of the version control system

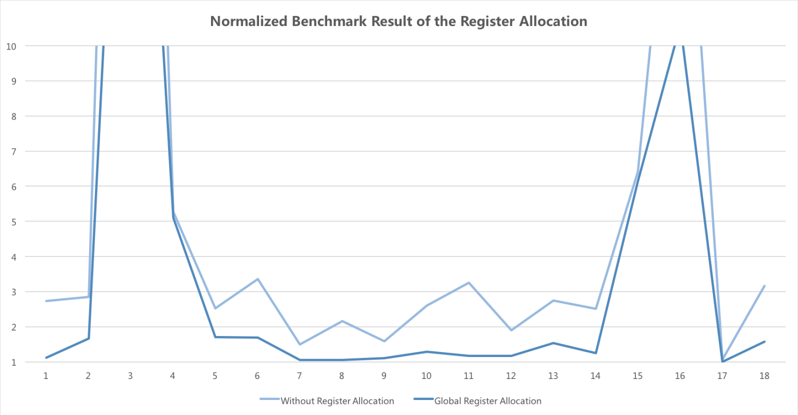
Demonstration: Normalized Benchmark Result



Highlights

- Use a well-organized framework with good extensibility
- Feed back a user-friendly compilation error message
- Support most features of **OOP** including member function, private modifier, class inheritance, member initialization and constructor function
- Achieve an outstanding compilation quality
- Implement the **SSA** form and do some optimizations on it including the useless code elimination and dominator-based value numbering technique
- Do not use any peephole and data-oriented optimizations
- Use the **global** register allocation and improve the algorithm
- Make good use of the version control system

Demonstration: Normalized Benchmark Result



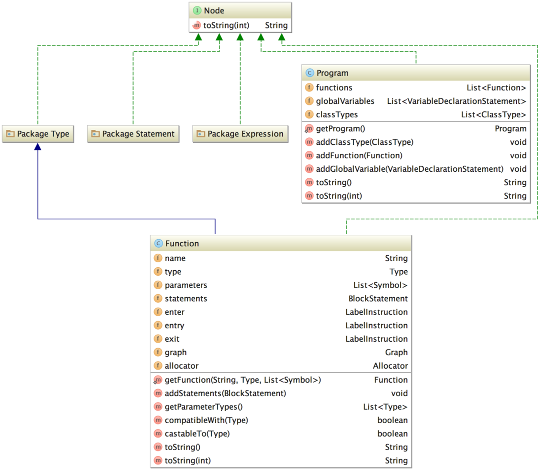
Highlights

- Use a well-organized framework with good extensibility
- Feed back a user-friendly compilation error message
- Support most features of **OOP** including member function, private modifier, class inheritance, member initialization and constructor function
- Achieve an outstanding compilation quality
- Implement the **SSA** form and do some optimizations on it including the useless code elimination and dominator-based value numbering technique
- Do not use any peephole and data-oriented optimizations
- Use the **global** register allocation and improve the algorithm
- Make good use of the version control system

Front End

1. Use **AN**other **T**ool for **L**anguage **R**ecognition to generate the lexer and parser
2. Convert the **C**oncrete **S**yntax **T**ree to the **A**bstract **S**yntax **T**ree by the listener mode of **ANTLR**

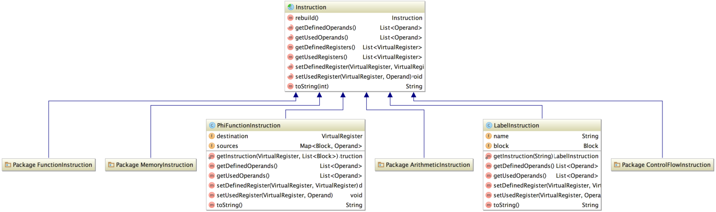
Abstract Syntax Tree Framework



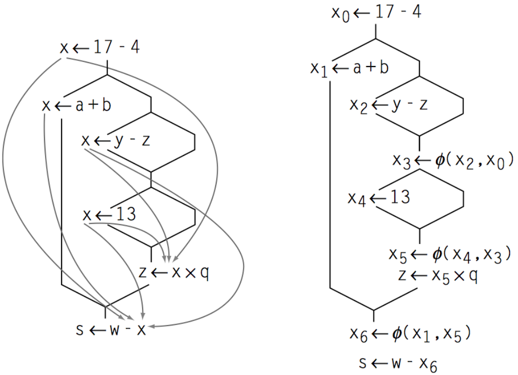
Back End

1. Convert the **A**bstract **S**yntax **T**ree to the linear **I**ntermediate **R**epresentation
2. Build the **C**ontrol **F**low **G**raph by scanning the linear **I**ntermediate **R**epresentation

Intermediate Representation Framework



Single Static Assignment



(a) Original Code Fragment (b) With x in SSA Form

Single Static Assignment

1. Compute the immediate dominator and the dominance frontier
2. Insert ϕ-functions
3. Rename the variables and the temporaries
4. Do some optimizations based on SSA form
5. Translate out of SSA form

Single Static Assignment: Analyze the Dominance

Definition

In a flow graph with entry node s, node u **dominates** node v iff u lies on every path from s to v

Data flow equation

$$\mathrm{dom}_v = \{v\} \cup \left(\bigcap_{p \in pred_v} dom_p \right)$$

Single Static Assignment: Analyze the Dominance

Definition

In a flow graph with entry node s, node u **dominates** node v iff u lies on every path from s to v

Data flow equation

$$\mathrm{dom}_v = \{v\} \cup \left(\bigcap_{p \in pred_v} dom_p \right)$$

Single Static Assignment: Analyze the Dominance

Definitions

- A node u **strictly dominates** a node v if u dominates v and u does not equal v.
- The **immediate dominator** of a node u is the unique node that strictly dominates u but does not strictly dominate any other node that strictly dominates u.
- The **dominance frontier** of a node u is the set of all nodes v such that u dominates an immediate **predecessor** of v, but u does not strictly dominate v. It is the set of nodes where u's dominance stops.

Single Static Assignment: Immediate Dominator

```
for all nodes, b    // initialize the dominators array
    IDoms[b] ← Undefined
IDoms[b0] ← b0
Changed ← true
while (Changed)
   Changed ← false
   for all nodes, b, in reverse postorder (except root)
       NewIDom ← first (processed) predecessor of b   // pick one
       for all other predecessors, p, of b
           if IDoms[p] ≠ Undefined   // i.e., Doms[p] already calculated
               then NewIdom ← Intersect(p, NewIdom)
       if IDoms[b] ≠ NewIdom then
           IDoms[b] ← NewIdom
           Changed ← true

Intersect(i, j)
    finger1 ← i
    finger2 ← j
    while (finger1 ≠ finger2)
        while (RPO(finger1) > RPO(finger2))
            finger1 = IDoms[finger1]
        while (RPO(finger2) > RPO(finger1))
            finger2 = IDoms[finger2]
    return finger1
```

Single Static Assignment: Insert ϕ-functions

```
Globals ← ∅
Initialize all the Blocks sets to ∅
for each block b
    VarKill ← ∅
    for each operation i in b, in order
        assume that op_i is "x ← y op z"
        if y ∉ VarKill then
            Globals ← Globals ∪ {y}
        if z ∉ VarKill then
            Globals ← Globals ∪ {z}
        VarKill ← VarKill ∪ {x}
        Blocks(x) ← Blocks(x) ∪ {b}
```

(a) Finding Global Names

```
for each name x ∈ Globals
    WorkList ← Blocks(x)
    for each block b ∈ WorkList
        for each block d in DF(b)
            if d has no φ-function for x then
                insert a φ-function for x in d
                WorkList ← WorkList ∪ {d}
```

(b) Rewriting the Code

Single Static Assignment: Rename the Variables

```
for each global name i
  counter[i] ← 0
  stack[i] ← Ø
Rename(n0)
```

```
NewName(n)
 i ← counter[n]
 counter[n] ← counter[n] + 1
 push i onto stack[n]
 return "n_i"
```

```
Rename(b)
  for each φ-function in b, "x ← φ(···)"
      rewrite x as NewName(x)

  for each operation "x ← y op z" in b
      rewrite y with subscript top(stack[y])
      rewrite z with subscript top(stack[z])
      rewrite x as NewName(x)

  for each successor of b in the CFG
      fill in φ-function parameters

  for each successor s of b in the dominator tree
      Rename(s)

  for each operation "x ← y op z" in b
      and each φ-function "x ← φ(···)"
      pop(stack[x])
```

Optimizations

Regular optimizations

1. Leaf function optimization
2. Control flow optimization

Optimizations based on SSA form

1. Useless code elimination
2. Dominator-based value numbering technique

Optimization: Control Flow Optimization

```
Clean( )
  while the CFG keeps changing
    compute postorder
    OnePass( )

OnePass( )
  for each block i, in postorder
    if i ends in a conditional branch then
        if both targets are identical then
           replace the branch with a jump
    if i ends in a jump to j then
        if i is empty then
           replace transfers to i with transfers to j
        if j has only one predecessor then
           combine i and j
        if j is empty and ends in a conditional branch then
           overwrite i's jump with a copy of j's branch
```

Optimization: Useless Code Elimination

```
Mark( )
 WorkList ← ∅

 for each operation i
     clear i's mark
     if i is critical then
        mark i
        WorkList ← WorkList ∪ {i}

 while (WorkList ≠ ∅)
     remove i from WorkList
        (assume i is x ← y op z)

     if def(y) is not marked then
        mark def(y)
        WorkList ← WorkList ∪ {def(y)}
     if def(z) is not marked then
        mark def(z)
        WorkList ← WorkList ∪ {def(z)}

     for each block b ∈ RDF(block(i))
        let j be the branch that ends b
        if j is unmarked then
           mark j
           WorkList ← WorkList ∪ {j}
```

(a) The *Mark* Routine

```
Sweep( )
 for each operation i
     if i is unmarked then
        if i is a branch then
           rewrite i with a jump
             to i's nearest marked
             postdominator

        if i is not a jump then
           delete i
```

(b) The *Sweep* Routine

Optimization: Dominator-based Value Numbering

```
procedure DVNT(B)
   allocate a new scope for B
   for each φ-function of the form ‘‘n ← φ(...)’’ in B
      if p is meaningless or redundant then
         VN[n] ← the value number for p
         remove p
      else
         VN[n] ← n
         Add p to the hash table

   for each assignment a of the form ‘‘x ← y op z’’ in B
      overwrite y with VN[y]
      overwrite z with VN[z]

      let expr ← ‘‘y op z’’
      if expr can be simplified to expr′ then
         replace a with ‘‘x ← expr′
         expr ← expr′

      if expr has a value number v in the hash table then
         VN[x] ← v
         remove statement a

      else
         VN[x] ← x
         add expr to the hash table with value number x

   for each successor s of B
      adjust the φ-function inputs in s

   for each child c of B in the dominator tree
      DVNT(c)

   deallocate the scope for B
```

Global Register Allocation

1. Analyze the liveliness
2. Build the interference graph
3. Use the bottom-up coloring

Global Register Allocation: Liveliness Analysis

Definitions

- v **live** at e iff there is a path p from e to i such that
  - instruction i uses v
  - no instructions defining v on the path p
- v **live-in** at i iff v **live** at e, where $e \in \textit{in-edge}_i$
- v **live-out** at i iff v **live** at e, where $e \in \textit{out-edge}_i$

Data flow equation

$$\textit{live-in}_i = \textit{use}_i \cup (\textit{live-out}_i \setminus \textit{def}_i)$$

$$\textit{live-out}_i = \bigcup_{s \in \textit{succ}_i} \textit{live-in}_s$$

Global Register Allocation: Liveliness Analysis

Definitions

- v **live** at e iff there is a path p from e to i such that
  - instruction i uses v
  - no instructions defining v on the path p
- v **live-in** at i iff v **live** at e, where $e \in \textit{in-edge}_i$
- v **live-out** at i iff v **live** at e, where $e \in \textit{out-edge}_i$

Data flow equation

$$\textit{live-in}_i = \textit{use}_i \cup (\textit{live-out}_i \setminus \textit{def}_i)$$

$$\textit{live-out}_i = \bigcup_{s \in \textit{succ}_i} \textit{live-in}_s$$

Global Register Allocation: Liveliness Analysis

1. Calculate the *def* and *use* for each basic block
2. Do the liveliness analysis on each basic block by using **the fix-point algorithm**
3. Calculate the liveliness information for each instruction in each basic block by **the one-pass backward calculation**

Global Register Allocation: Interference Graph

- For a **move** instruction i,
  add **forbidden** edges between def_i and live-out$_i \setminus use_i$
  add **recommend** edges between def_i and use_i
- For a **non-move** instruction i,
  add **forbidden** edges between def_i and live-out$_i$

Global Register Allocation: Bottom-up Coloring

Algorithm 1 Computing the coloring order for $G = (V, E)$

initialize *stack* to empty
while V is not empty **do**
  if $\exists v \in V$ with $deg_v < k$ **then**
    candidate $\leftarrow v$
  else
    candidate $\leftarrow v$ **picked** from V
  end if
  remove *candidate* and its edges in the graph G
  push *candidate* onto *stack*
end while

Global Register Allocation: Bottom-up Coloring

Algorithm 2 Coloring bottom-up for $G = (V, E)$

1: **while** *stack* is not empty **do**
2: $\quad v \leftarrow pop(stack)$
3: $\quad$ insert v and its edges into the graph G
4: $\quad$ **color** v
5: **end while**

Other work

- Give some of my classmates advice on the framework and optimization
- Provide some IR test data for my classmates
- Provide a unit test file for my classmates
- Help some of my classmates with debugging

Other work

- Give some of my classmates advice on the framework and optimization
- Provide some **IR** test data for my classmates
- Provide a unit test file for my classmates
- Help some of my classmates with debugging

Other work

- Give some of my classmates advice on the framework and optimization
- Provide some **IR** test data for my classmates
- Provide a unit test file for my classmates
- Help some of my classmates with debugging

Other work

- Give some of my classmates advice on the framework and optimization
- Provide some **IR** test data for my classmates
- Provide a unit test file for my classmates
- Help some of my classmates with debugging

Summary

- I am ashamed to do only a little bit of the work
- Thank all of you, my **TA**s and my classmates