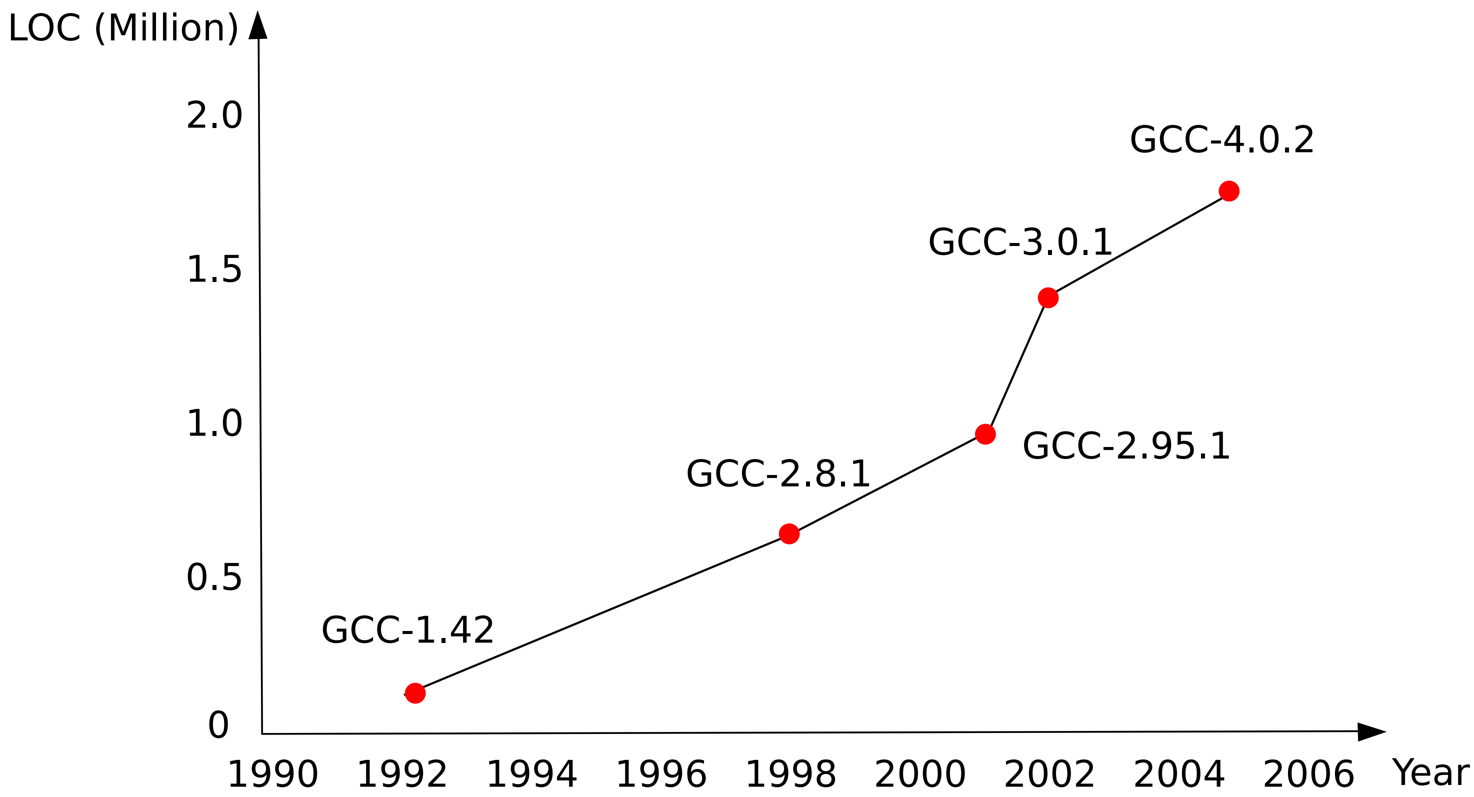
A Domain-Specific Language for Generating Dataflow Analyzers

Jia Zeng
Columbia Univ.
jia@cs.columbia.edu

Chuck Mitchell
Microsoft Co.
chuckm@microsoft.com

Stephen A. Edwards
Columbia Univ.
sedwards@cs.columbia.edu

Motivation



GCC Code Sizes

Motivation

- Algorithm: Live variable analysis in Dragon Book

Input: A flow graph with *def* and *use*
Output: $out[B]$

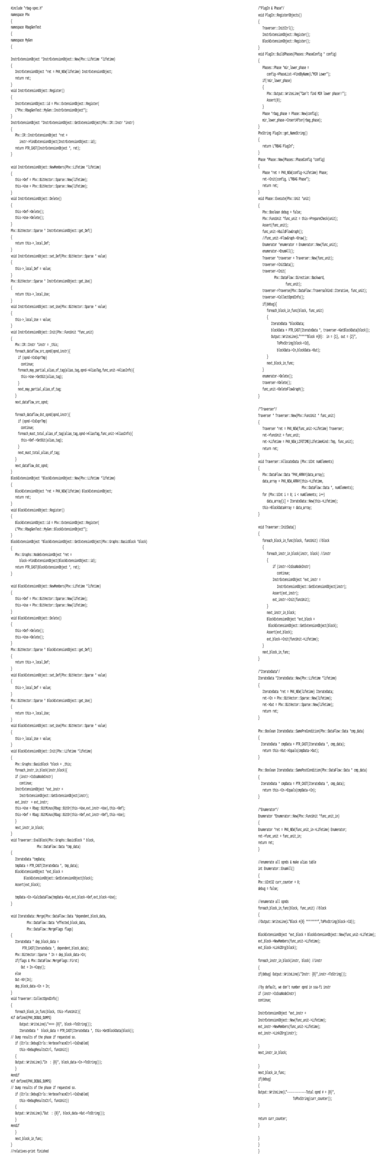
for each block B **do** $in[B] = \emptyset$
while changes to any of the *in*'s occur **do**
  for each block B **do begin**
    $out[B] = \cup_{S \text{ a successor of } B} in[S]$
    $in[B] = use[B] \cup (out[B] - def[B])$
  end

Motivation: AG vs. Phoenix C++

Input: A flow graph with *def* and *use*
Output: *out[B]*

for each block B **do** $in[B] = \emptyset$
while changes to any of the *in*'s occur **do**
  for each block B **do begin**
    $out[B] = \cup_{S \text{ a successor of } B}\, in[S]$
    $in[B] = use[B] \cup (out[B] - def[B])$
  end

```
Phase Liveness(Func func){

   extend class Opnd{
      Set<Alias> Def;
      Set<Alias> Use;

      void Init(){
         Opnd opnd = this;

 if(opnd->IsDef){//defs
   foreach_must_total_alias_of_tag(alias_tag, opnd->AliasTag,AliasInfo)
     opnd->Def += alias_tag;
 }
 else{//uses
   foreach_may_partial_alias_of_tag(alias_tag,opnd->AliasTag,AliasInfo)
     opnd->Use += alias_tag;
         }
      }
   }

   extend class Instr {

     Set<Alias> Def; //set will be translate to sparse bit vector on opnd->ID
     Set<Alias> Use;

     void Init() {

       Instr instr = this;
       foreach(Opnd opnd in instr where (dataflow && src)){
          instr->Use += opnd->Use;
       }
   foreach(Opnd opnd in instr where (dataflow && dst)){
          instr->Def += opnd->Def;
       }
     }
   }

   extend class Block{
      Set<Alias> Def;
      Set<Alias> Use;

      void Init(){
         Block block = this;

         foreach(Instr instr in block) {
   block->Use = (block->Use + instr->Use) - block->Def;
   block->Def = (block->Def + instr->Def) - block->Use;

         }
      }
   }

   Set<Alias> TransFunc(Backward,Iterative,Block){

      Compose(N){
         In = (Out - N->Def) + N->Use;
      }

      Meet(N){
         Out += N->In;
      }

      Closure(N){
      }

      Result(N){
Print(In);
Print(Out);
      }
   }
}
```

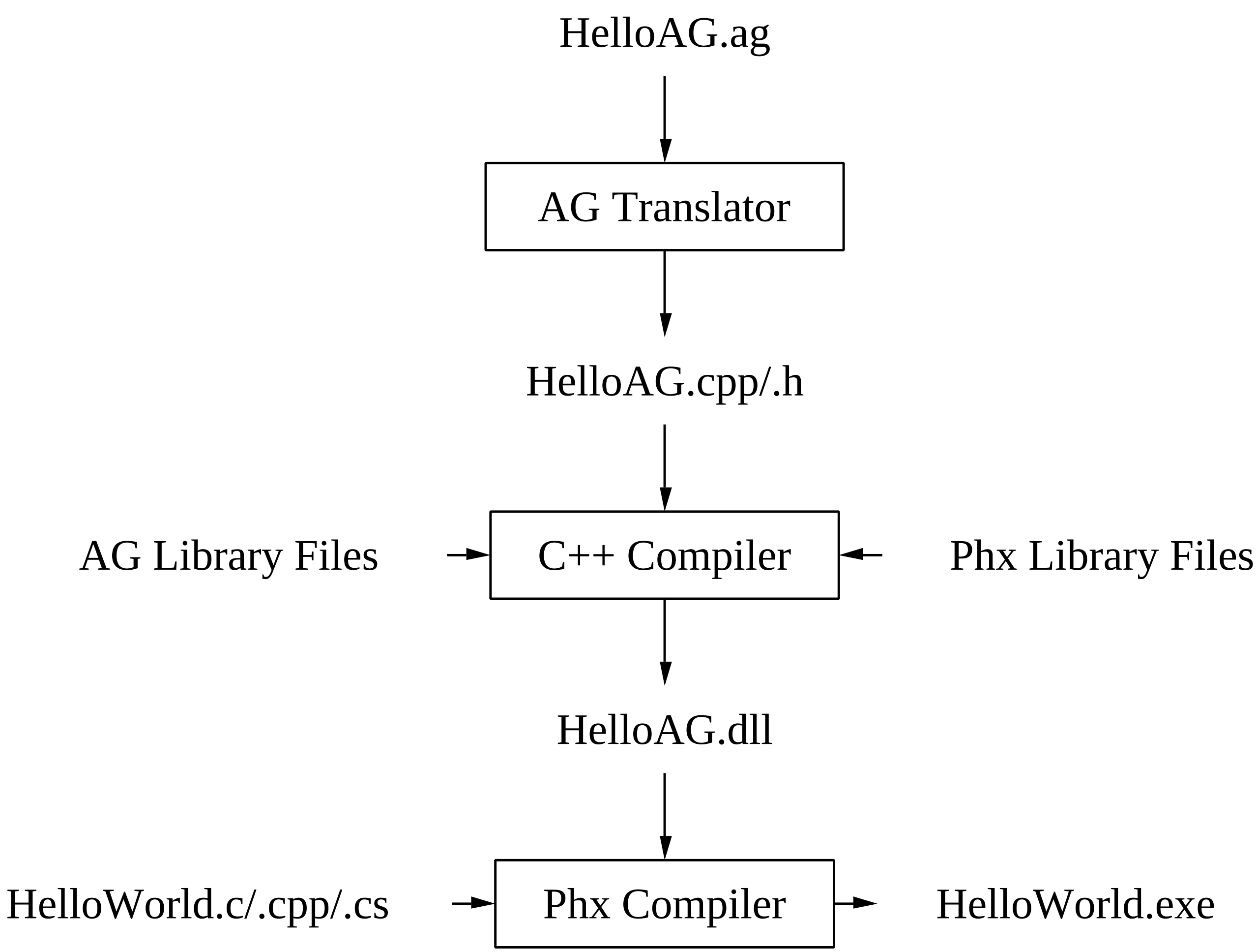


8 : 55 : 519

Previous Work

- Theory of dataflow analysis
  - Gary A. Kildall [1973]: unified lattice-based framework
  - Kam and Ullman [1976]: iterative approach
- Applicable tools
  - Tjiang's Sharlit [1992]: efficiency > simplicity
  - Alt and Martin's PAG [1995]: lattice specification
  - Yi & Harrison's work [1993]: interprocedural analysis

Overview of AG Tool



An AG Program

```
Phase name {
   extend class name {
      //field declarations...
      //method declarations...
      void Init() { ... }
   }
   ...
   type TransFunc(direction) {
      Meet(P) { ... }
      Compose(N) { ... }
      Result(N) { ... }
   }
}
```

Example: Reaching Definitions

A *definition* of a variable is the operand in an instruction that may assign a value to the variable.

$$
\begin{array}{l}
y = 0; \\
\textbf{if } (x > 0)\ \{ \\
\quad y = x * 2; \\
\quad x - -; \\
\} \\
\textbf{else} \\
\quad x + +; \\
z = y * x;
\end{array}
$$

Example: Reaching Definitions

A *definition* of a variable is the operand in an instruction that may assign a value to the variable.

```
y = 0;
if (x > 0) {
    y = x * 2;
    x--;
}
else
    x++;
z = y * x;
```

Example: Reaching Definitions

for each block B **do begin**

    Compute $gen[B]$ and $kill[B]$;

end

for each block B **do** $in[B] = \emptyset$;

$change$ = **true**;

while $change$ **do begin**

    $change$ = **false**;

    for each block B **do begin**

        $oldout = out[B]$;

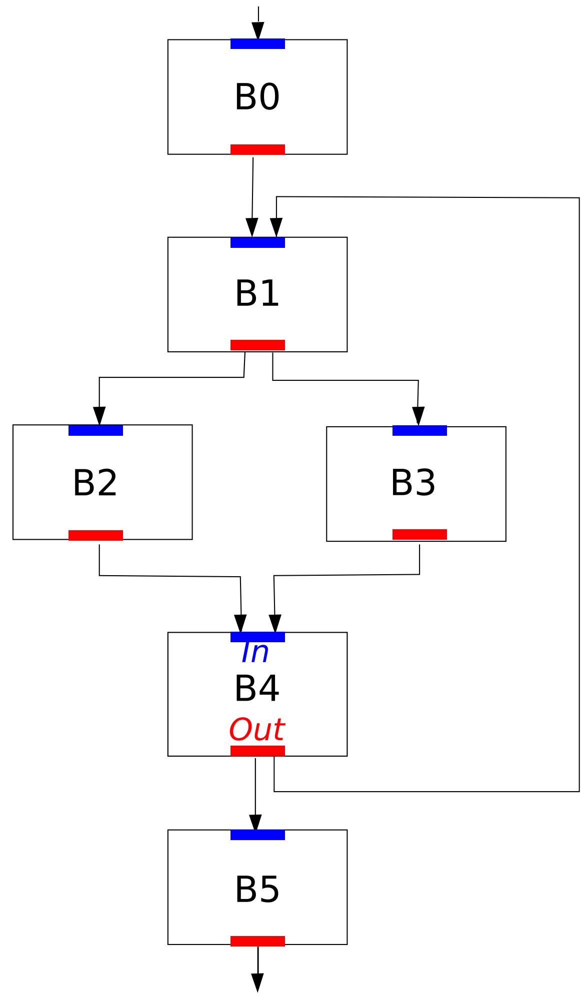
        $in[B] = \cup_{P \text{ a predecessor of } B} out[P]$;

        $out[B] = gen[B] \cup (in[B]\text{-}kill[B])$;

        if $out[B] \neq oldout$ **then** $change$ = **true**;

    end

end



Example: Reaching Definitions in AG

for each block B **do begin**
  Compute $gen[B]$ and $kill[B]$;
end
for each block B **do** $in[B] = \emptyset$;
$change$ = **true**;
while $change$ **do begin**
  $change$ = **false**;
  for each block B **do begin**
    $oldout = out[B]$;
    $in[B] = \cup_{P \text{ a predecessor of } B}\, out[P]$;
    $out[B] = gen[B] \cup (in[B]\text{-}kill[B])$;
    if $out[B] \neq oldout$ **then** $change$ = **true**;
  end
end

```
Phase name {
  extend class name {
    //field declarations...
    //method declarations...
    void Init() { ... }
  }
  ...
  type TransFunc(direction)
    Meet(P) { ... }
    Compose(N) { ... }
    Result(N) { ... }
  }
}
```

Example: Reaching Definitions in AG

```
Phase ReachingDefs {
   extend class Opnd {
      ...
   }
   extend class Instr {
      ...
   }
   extend class Block {
      ...
   }
   Set<Opnd> TransFunc(Forward) {
      ...
   }
}
```

Open Classes

```
class Instr {
  int id;
  list<Opnd> srcOpnds;
  list<Opnd> dstOpnds;
}

class MyInstr : Instr {
  list<Opnd> gen;
  list<Opnd> kill;
  void Init() {...}
}
```

Open Classes

```
class Instr {
  int id;
  list<Opnd> srcOpnds;
  list<Opnd> dstOpnds;
}

class MyInstr : Instr {
  list<Opnd> gen;
  list<Opnd> kill;
  void Init() {...}
}
```

```
class Instr {
  int id;
  list<Opnd> srcOpnds;
  list<Opnd> dstOpnds;

  //added fields
  list<Opnd> gen;
  list<Opnd> kill;

  //added methods
  void Init() {...}
}
```

Previous Work

	AspectJ	**MultiJava**	**AG**
Usage	General Purpose	General Purpose	Dataflow Analysis
Methods	$+$	$+$	$+$
Fields	$+$	$-$	$+$
Recompiling	Yes	No	No
Dynamic	$-$	$+$	$+$

Example AG: Extend Class Block

```
extend class Block {
   Set<Opnd> Gen;
   Set<Opnd> Kill;

   void Init() {
      Block block = this;

      foreach (Instr instr in block) {
         block->Gen = instr->Gen +
               (block->Gen - instr->Kill);
         block->Kill = block->Kill + instr->Kill
               - instr->Gen;
      }
   }
}
```

Example AG: Extend Class Instr

```
extend class Instr {
   Set<Opnd> Gen;
   Set<Opnd> Kill;

   void Init() {
      Instr instr = this;
      foreach (Opnd dstOpnd in instr
            where (dataflow && dst)) {
         instr->Gen += dstOpnd->Gen;
         instr->Kill += dstOpnd->Kill;
      }
   }
}
```

Example AG: Extend Class Opnd

```
extend class Opnd {
   Set<Opnd> Gen;
   Set<Opnd> Kill;

   void Init() {
      Opnd opnd = this;
      if (opnd->IsDef) {
         opnd->Gen += opnd;
         foreach_must_total_alias_of_tag(alias_tag,
               opnd->AliasTag, AliasInfo) {
            opnd->Kill += DstAliasTable(alias_tag);
         }
         opnd->Kill -= opnd;
   }
}
```

Example AG: C++ Code Generated

```
void OpndExtensionObject::Init( Phx::FuncUnit *func_unit,
           Phx::BitVector::Sparse *PHX_ARRAY(dst_alias_table))
{
  Phx::IR::Opnd *opnd = _this;
  if(opnd->IsDef) {
    this->Gen->SetBit(this->uid);
    foreach_must_total_alias_of_tag(alias_tag,
         opnd->AliasTag, func_unit->AliasInfo) {
      this->Kill->Or(dst_alias_table[alias_tag]);
    }
    next_must_total_alias_of_tag;
    this->Kill->ClearBit(this->uid);
  }
}
```

Example AG: Transfer Functions

```
Set<Opnd> TransFunc(Forward) {
   Meet(P) {
      In += P->Out;
   }

   Compose(N) {
      Out = In - N->Kill + N->Gen;
   }
}
```

AG Syntax - Highlights

data types	Set Map int bool void
special variables	In Out
operators	$+ - * = += -= *=$ && || ! !=
built-in classes	Opnd Instr Block Alias Expr Func Region
built-in constants	Forward Backward
statements	foreach (*type var* in *range* where *cond*. *direction*) { ... }
	phoenix-iterator (...) { ... }
	lvalue = *expression*;
	if (*expression*) { ... } else { ... }
	/% arbitrary C++ code %/
built-in functions	DstAliasTable SrcAliasTable Print

Experimental Results

	Reaching Definitions	**Live Variables**	**Uninitialized Variables**
C++ LOC (manual)	791	303	108
C++ Style	dynamic	global var	SSA
AG LOC (manual)	64	55	94
C++ LOC (generated)	626	519	682
C++ runtime	7.3s	0.8s	
AG runtime	7.4s	3.1s	13.6s

- Benchmark:

  Phoenix Microsoft Intermediate Language reader

  ($>$ 500,000 lines)

Conclusions

- Strengths
  - Very compact code: $< 1/10$ of manually-written C++ program
  - Similar performance: speed penalty < 4 times
  - Mechanism for extending existing IR classes
- Weaknesses
  - Fixed framework: iterative, MIR-based
  - Debug environment

Acknowledgments

- ☺ Thanks to Al Aho!
- ☺ Thanks to the Microsoft Phoenix group!