GDFA: Generic Data Flow Analyser for GCC

Uday Khedker

(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

August 2009

Part 1

About These Slides

Copyright

These slides constitute the lecture notes for CS618 Program Analysis course at IIT Bombay and have been made available as teaching material accompanying the book:

- Uday Khedker, Amitabha Sanyal, and Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. CRC Press (Taylor and Francis Group). 2009.

Outline

- Motivation
- Common abstractions in data flow analysis
- Implementing data flow analysis using *gdfa*
- Design and Implementation of *gdfa*

Motivation behind gdfa

- Specification Vs. implementation
- Orthogonality of specification of data flow analysis and the process of performing data flow analysis
- Practical significance of generalizations
- Ease of extending data flow analysers

Part 2

Common Abstractions in Bit Vector Data Flow Frameworks

Common Form of Data Flow Equations

Data Flow Information

So far we have seen sets (or bit vectors). Could be entities other than sets for non-bit vector frameworks.

Flow Function

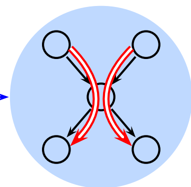
$$X_i = f(Y_i)$$

$$Y_i = \bigsqcap X_j$$

Confluence

So far we have seen $\cup$ and $\cap$. Could be other operations for non-bit vector frameworks.

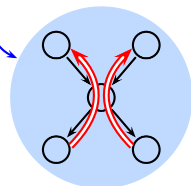
A Taxonomy of Bit Vector Data Flow Frameworks

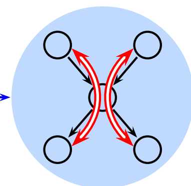


Any Path

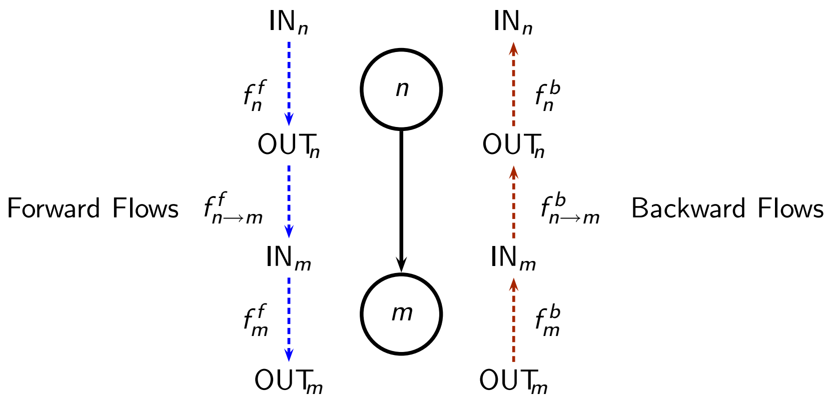
All Paths

	Confluence	
	Union	Intersection
Forward	Reaching Definitions	Available Expressions
Backward	Live Variables	Anticipable Exressions
Bidirectional (limited)		Partial Redundancy Elimination (Original M-R Formulation)



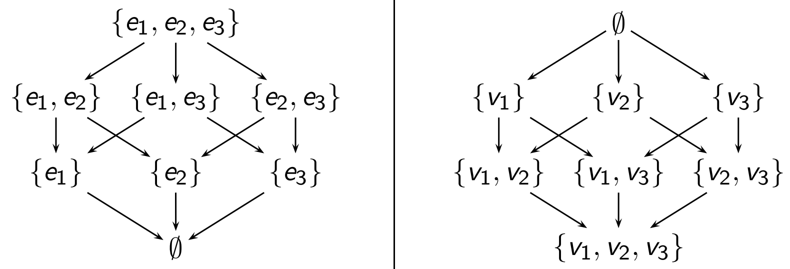


The Abstraction of Flow Functions



The Abstraction of Data Flow Values

Available Expressions Analysis	Live Variables Analysis



Available Expressions Analysis	Live Variables Analysis
$\sqsubseteq$ is $\subseteq$	$\sqsubseteq$ is $\supseteq$
$\sqcap$ is $\cap$	$\sqcap$ is $\cup$

The Abstraction of Data Flow Equations

$$\text{IN}_n = \begin{cases} \mathit{Boundaryinfo} \sqcap f_n^b(\text{OUT}_n) & n = \mathit{Start} \\ \left(\bigsqcap\limits_{m \in pred(n)} f_{m\to n}^f(\text{OUT}_m) \right) \sqcap f_n^b(\text{OUT}_n) & \text{otherwise} \end{cases}$$

$$\text{OUT}_n = \begin{cases} \mathit{BI}_{\mathit{End}} \sqcap f_n^f(\text{IN}_n) & n = \mathit{End} \\ \left(\bigsqcap\limits_{m \in succ(n)} f_{m\to n}^b(\text{IN}_m) \right) \sqcap f_n^f(\text{IN}_n) & \text{otherwise} \end{cases}$$

Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

- *Round Robin.* Repeated traversals over nodes in a fixed order
  Termination : After values stabilise
  - \+ Simplest to understand and implement
  - − May perform unnecessary computations

  Our examples use this method.
- *Work List.* Dynamic list of nodes which need recomputation
  Termination : When the list becomes empty
  - \+ Demand driven. Avoid unnecessary computations.
  - − Overheads of maintaining work list.

Common Form of Flow Functions

$$f_n(X) = (X - \text{Kill}_n(X)) \cup \text{Gen}_n(X)$$

- For General Data Flow Frameworks

$$\begin{aligned}\text{Gen}_n(X) &= ConstGen_n \cup DepGen_n(X)\\ \text{Kill}_n(X) &= ConstKill_n \cup DepKill_n(X)\end{aligned}$$

- For bit vector frameworks

$$\begin{aligned}\text{Gen}_n(X) &= ConstGen_n\\ \text{Kill}_n(X) &= ConstKill_n\end{aligned}$$

Defining Flow Functions for Bit Vector Frameworks

- Live variables analysis

	Entity	Manipulation	Exposition
$ConstGen_n$	Variable	Use	Upwards
$ConstKill_n$	Variable	Modification	Anywhere

- Available expressions analysis

	Entity	Manipulation	Exposition
Gen_n	Expression	Use	Downwards
Kill_n	Expression	Modification	Anywhere

Part 3

Implementing Data Flow Analysis using gdfa

Implementing Available Expressions Analysis

1. Specifying available expressions analysis
2. Implementing the entry function of available expressions analysis pass
3. Registering the available expressions analysis pass
   3.1 Declaring the pass
   3.2 Registering the pass
   3.3 Positioning the pass

Step 1: Specifying Available Expressions Analysis

```
struct gimple_pfbv_dfa_spec gdfa_ave =
{
      entity_expr,                /* entity            */
      ONES,                       /* top_value         */
      ZEROS,                      /* entry_info        */
      ONES,                       /* exit_info         */
      FORWARD,                    /* traversal_order   */
      INTERSECTION,               /* confluence        */
      entity_use,                 /* gen_effect        */
      down_exp,                   /* gen_exposition    */
      entity_mod,                 /* kill_effect       */
      any_where,                  /* kill_exposition   */
      global_only,                /* preserved_dfi     */
      identity_forward_edge_flow, /* forward_edge_flow */
      stop_flow_along_edge,       /* backward_edge_flow */
      forward_gen_kill_node_flow, /* forward_node_flow */
      stop_flow_along_node        /* backward_node_flow */
};
```


Step 2: Implementing Available Expressions Analysis Pass

```
pfbv_dfi ** AV_pfbv_dfi = NULL;

static unsigned int
gimple_pfbv_ave_dfa(void)
{

        AV_pfbv_dfi = gdfa_driver(gdfa_ave);

        return 0;
}
```


Step 3.1: Declaring the Available Expressions Analysis Pass

```
struct tree_opt_pass pass_gimple_pfbv_ave_dfa =
{
  "gdfa_ave",             /* name */
  NULL,                   /* gate */
  gimple_pfbv_ave_dfa,    /* execute */
  NULL,                   /* sub */
  NULL,                   /* next */
  0,                      /* static_pass_number */
  0,                      /* tv_id */
  0,                      /* properties_required */
  0,                      /* properties_provided */
  0,                      /* properties_destroyed */
  0,                      /* todo_flags_start */
  0,                      /* todo_flags_finish */
  0                       /* letter */
};
```


Step 3.2: Registering the Available Expressions Analysis Pass

In file file `tree-pass.h`

```
extern struct tree_opt_pass pass_gimple_pfbv_ave_dfa;
```

Step 3.3: Positioning the Pass

In function init_optimization_passes in file passes.c.

```
  NEXT_PASS (pass_build_cfg);
/* Intraprocedural dfa passes begin */
  NEXT_PASS (pass_init_gimple_pfbvdfa);
  NEXT_PASS (pass_gimple_pfbv_ave_dfa);
```


Part 4

gdfa: Design and Implementation

Specifying Live Variables Analysis

- Entity should be `entity_var`
- ⊤, *Boundaryinfo* and *BIEnd* should be `ZEROS`
- Direction should be `BACKWARD`
- Confluence should be `UNION`
- Exposition should be `up_exp`
- Forward edge flow should be `stop_flow_along_edge`
- Forward node flow should be `stop_flow_along_node`
- Backward edge flow should be `identity_backward_edge_flow`
- Backward node flow should be `backward_gen_kill_node_flow`

Specification Data Structure

```
struct gimple_pfbv_dfa_spec
{
   entity_name               entity;
   initial_value             top_value_spec;
   initial_value             entry_info;
   initial_value             exit_info;
   traversal_direction       traversal_order;
   meet_operation            confluence;
   entity_manipulation       gen_effect;
   entity_occurrence         gen_exposition;
   entity_manipulation       kill_effect;
   entity_occurrence         kill_exposition;
   dfi_to_be_preserved       preserved_dfi;
   dfvalue (*forward_edge_flow)(basic_block src, basic_block dest);
   dfvalue (*backward_edge_flow)(basic_block src, basic_block dest);
   dfvalue (*forward_node_flow)(basic_block bb);
   dfvalue (*backward_node_flow)(basic_block bb);
};
```

Specification Primitives

Enumerated Type	Possible Values
entity_name	entity_expr, entity_var, entity_defn
initial_value	ONES, ZEROS
traversal_direction	FORWARD, BACKWARD, BIDIRECTIONAL
meet_operation	UNION, INTERSECTION
entity_manipulation	entity_use, entity_mod
entity_occurrence	up_exp, down_exp, any_where
dfi_to_be_preserved	all, global_only, no_value

Pre-Defined Edge Flow Functions

- Edge Flow Functions

Edge Flow Function	Returned value
identity_forward_edge_flow(src, dest)	CURRENT_OUT(src)
identity_backward_edge_flow(src, dest)	CURRENT_IN(dest)
stop_flow_along_edge(src, dest)	top_value

- Node Flow Functions

Node Flow Function	Returned value
identity_forward_node_flow(bb)	CURRENT_IN(bb)
identity_backward_node_flow(bb)	CURRENT_OUT(bb)
stop_flow_along_node(bb)	top_value
forward_gen_kill_node_flow(bb)	CURRENT_GEN(bb) ∪ (CURRENT_IN(bb) - CURRENT_KILL(bb))
backward_gen_kill_node_flow(bb)	CURRENT_GEN(bb) ∪ (CURRENT_OUT(bb) - CURRENT_KILL(bb))

The Generic Driver for Global Data Flow Analysis

```
pfbv_dfi ** gdfa_driver(struct gimple_pfbv_dfa_spec dfa_spec)
{   if (find_entity_size(dfa_spec) == 0) return NULL;
    initialize_special_values(dfa_spec);
    create_dfi_space();
    traversal_order = dfa_spec.traversal_order;
    confluence = dfa_spec.confluence;

    local_dfa(dfa_spec);

    forward_edge_flow = dfa_spec.forward_edge_flow;
    backward_edge_flow = dfa_spec.backward_edge_flow;
    forward_node_flow = dfa_spec.forward_node_flow;
    backward_node_flow = dfa_spec.backward_node_flow;
    perform_pfbvdfa();

    preserve_dfi(dfa_spec.preserved_dfi);
    return current_pfbv_dfi;
}
```


The Generic Driver for Local Data Flow Analysis

- The Main Difficulty: Interface with the intermediate representation details
- State of Art: The user is expected to supply the flow function implementation
- Our Key Ideas:
  - Local data flow analysis is a special case of global data flow analysis Other than the start and end blocks (≡ statements), every block has just one predecessor and one successor
  - $ConstGen_n$ and $ConstKill_n$ are just different names given to particular sets of entities accumulated by traversing these basic blocks

The Generic Driver for Local Data Flow Analysis

- Traverse statements in a basic block in appropriate order

Exposition	Direction
up_exp	backward
down_exp	forward
any_where	don't care

- Solve the recurrence

$$\texttt{accumulated\_entities} = (\texttt{accumulated\_entities} - \texttt{remove\_entities}) \cup \texttt{add\_entities}$$

Example for Available Expressions Analysis

Entity is `entity_expr`.

Let expr(x) denote the set of all expressions of x

Exposition	Manipulation	$a = b * c$		$b = b * c$	
		add	remove	add	remove
upwards	use	$b * c$	expr(a)	$b * c$	expr(b)
downwards	use	$b * c$	expr(a)	$\emptyset$	expr(b)
upwards	modification	expr(a)	$b * c$	expr(b) - $\{b * c\}$	$b * c$
downwards	modification	expr(a)	$b * c$	expr(b)	$\emptyset$

Note: In the case of modifications, if we first add then remove the entities modication, the set difference is not required

Future Work

Main thrust

- Supporting general data flow frameworks
- Supporting interprocedural analysis