

GDFA: Generic Data Flow Analyser for GCC

GCC Resource Center
(www.cse.iitb.ac.in/grc)

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



July 2010

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



Outline

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



Motivation behind gdfa

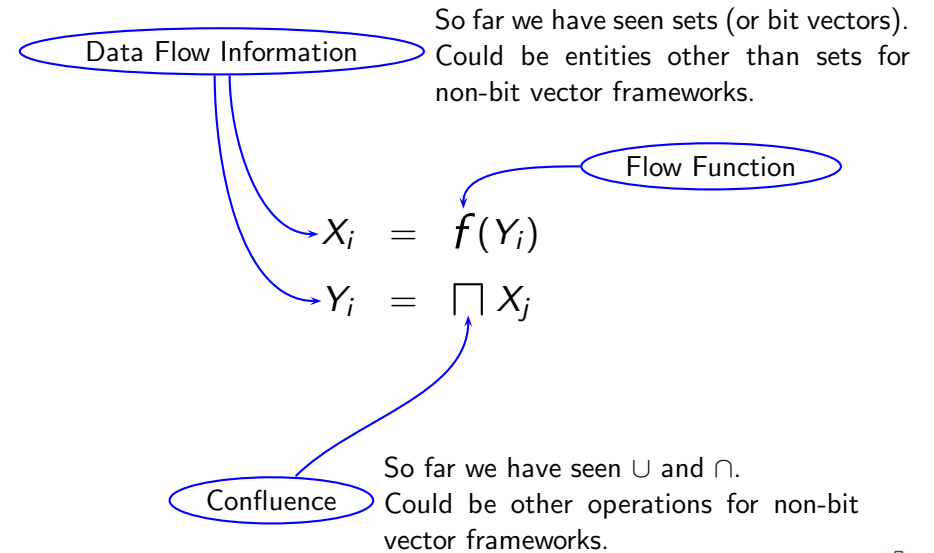
Notes



Part 2

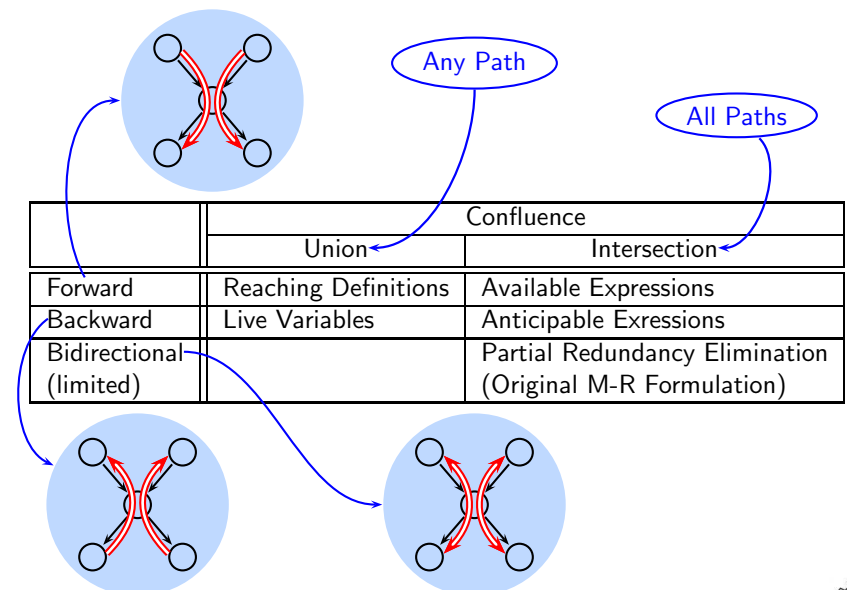
Common Abstractions in Data Flow Analysis

Common Form of Data Flow Equations



Common Form of Data Flow Equations

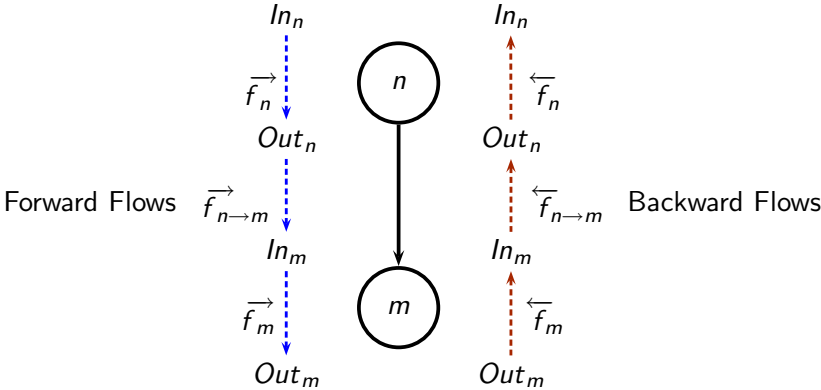
A Taxonomy of Bit Vector Data Flow Frameworks



A Taxonomy of Bit Vector Data Flow Frameworks

The Abstraction of Flow Functions

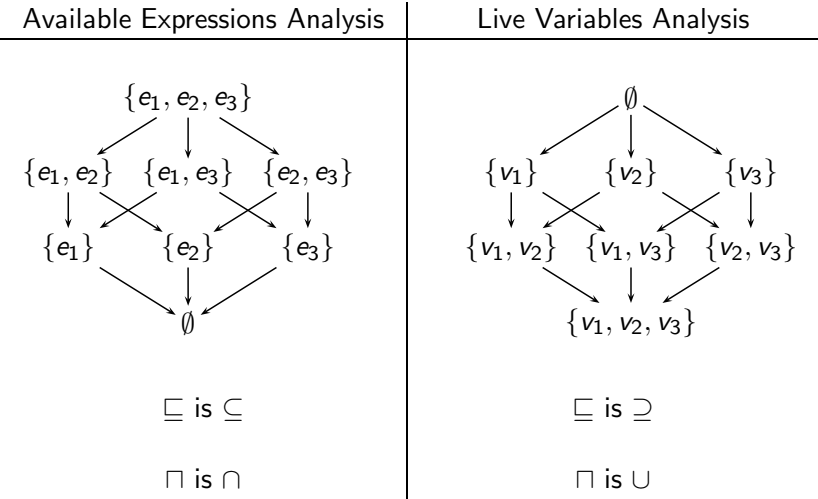
Notes



The Abstraction of Flow Functions

The Abstraction of Data Flow Values

Notes



The Abstraction of Data Flow Values

Notes



The Abstraction of Data Flow Equations

Notes



The Abstraction of Data Flow Equations

$$\begin{aligned}
 In_n &= \begin{cases} BStart \sqcap \overleftarrow{f}_n(Out_n) & n = Start \\ \left(\bigcap_{m \in pred(n)} \overrightarrow{f}_{m \rightarrow n}(Out_m) \right) \sqcap \overleftarrow{f}_n(Out_n) & \text{otherwise} \end{cases} \\
 Out_n &= \begin{cases} BEnd \sqcap \overrightarrow{f}_n(In_n) & n = End \\ \left(\bigcap_{m \in succ(n)} \overleftarrow{f}_{m \rightarrow n}(In_m) \right) \sqcap \overrightarrow{f}_n(In_n) & \text{otherwise} \end{cases}
 \end{aligned}$$



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.



Iterative Methods of Performing Data Flow Analysis

Notes



Common Form of Flow Functions

Notes



Common Form of Flow Functions

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

- For General Data Flow Frameworks

$$\text{Gen}_n(X) = \text{ConstGen}_n \cup \text{DepGen}_n(X)$$

$$\text{Kill}_n(X) = \text{ConstKill}_n \cup \text{DepKill}_n(X)$$

- For bit vector frameworks

$$\text{Gen}_n(X) = \text{ConstGen}_n$$

$$\text{Kill}_n(X) = \text{ConstKill}_n$$



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



Defining Flow Functions for Bit Vector Frameworks

Notes



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



Part 3

Implementing Data Flow Analysis using gdfa

Implementing Available Expressions Analysis

Notes



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 1: Specifying Available Expressions Analysis

Notes



Step 2: Implementing Available Expressions Analysis Pass

Notes



Step 3.1: Declaring the Available Expressions Analysis Pass

```
struct gimple_opt_pass pass_gimple_pfbv_ave_dfa =
{
  { GIMPLE_PASS,
    "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 gimple_opt_pass pass_gimple_pfbv_ave_dfa;
```



Step 3.1: Declaring the Available Expressions Analysis Pass

Notes



Step 3.2: Registering the Available Expressions Analysis Pass

Notes



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);
```



Enabling gdfa and Examining Data Flow Values

- Enabling gdfa pass
`gcc -S -fdump-tree-all -fgdfa`
- Enabling gdfa dump
`gcc -S -fdump-tree-all -fgdfa-details`
- Dump file name extension for available expressions analysis
`.gdfa_ave`



Step 3.3: Positioning the Pass

Notes



Enabling gdfa and Examining Data Flow Values

Notes



Specifying Live Variables Analysis

- Entity should be `entity_var`
- T , BI_{Start} and BI_{End} 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`



Part 4

gdfa: Design and Implementation

Specifying Live Variables Analysis

Notes



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 Data Structure

Notes



Specification Primitives

Notes



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)	$CURRENT_GEN(bb) \cup (CURRENT_IN(bb) - CURRENT_KILL(bb))$
backward_gen_kill_node_flow(bb)	$CURRENT_GEN(bb) \cup (CURRENT_OUT(bb) - CURRENT_KILL(bb))$



Pre-Defined Edge Flow Functions

Notes



The Generic Driver for Global Data Flow Analysis

Notes



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 (\equiv 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

Notes



The Generic Driver for Local Data Flow Analysis

Notes



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

```
accumulated_entities = (accumulated_entities
                        - remove_entities)
                        ∪ add_entities
```



Example for Available Expressions Analysis

Entity is entity_expr.

Let $\text{expr}(x)$ denote the set of all expressions of x

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

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



Example for Available Expressions Analysis

Notes



Future Work

Notes



Future Work

Main thrust

- Supporting general data flow frameworks
- Supporting interprocedural analysis

