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:


*These slides are being made available under GNU FDL v1.2 or later purely for academic or research use.*
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

Data Flow Information

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

\[ X_i = f(Y_i) \]

\[ Y_i = \bigcap X_j \]

Flow Function

Confluence

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

Part 2

Common Abstractions in Bit Vector Data Flow Frameworks

A Taxonomy of Bit Vector Data Flow Frameworks
**The Abstraction of Flow Functions**

Forward Flows $f^f_{n \rightarrow m}$

Backward Flows $f^b_{n \rightarrow m}$

**The Abstraction of Data Flow Values**

**Available Expressions Analysis**

- $\{e_1, e_2, e_3\}$
- $\{e_1, e_2, e_3\}$
- $\{e_1\}$
- $\{e_2\}$
- $\{e_3\}$

**Live Variables Analysis**

- $\emptyset$
- $\{v_1\}$
- $\{v_2\}$
- $\{v_3\}$
- $\{v_1, v_2, v_3\}$

**The Abstraction of Data Flow Equations**

**Iterative Methods of Performing Data Flow Analysis**

- **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
  \[
  \begin{align*}
  \text{Gen}_n(X) &= \text{ConstGen}_n \cup \text{DepGen}_n(X) \\
  \text{Kill}_n(X) &= \text{ConstKill}_n \cup \text{DepKill}_n(X)
  \end{align*}
  \]

- For bit vector frameworks
  \[
  \begin{align*}
  \text{Gen}_n(X) &= \text{ConstGen}_n \\
  \text{Kill}_n(X) &= \text{ConstKill}_n
  \end{align*}
  \]

Defining Flow Functions for Bit Vector Frameworks

- Live variables analysis

<table>
<thead>
<tr>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConstGen</td>
<td>Variable</td>
<td>Use</td>
</tr>
<tr>
<td>ConstKill</td>
<td>Variable</td>
<td>Modification</td>
</tr>
</tbody>
</table>

- Available expressions analysis

<table>
<thead>
<tr>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gen</td>
<td>Expression</td>
<td>Use</td>
</tr>
<tr>
<td>Kill</td>
<td>Expression</td>
<td>Modification</td>
</tr>
</tbody>
</table>

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

```c
struct gimple_pfbv_dfa_spec gdfa_ave =
{
    entity_expr,        /* entity */ ONES,    /* top_value */ ZEROS,   /* entry_info */ ONES,    /* exit_info */ FORWARD, /* traversal_order */
    /* 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

```c
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

```c
struct tree_opt_pass pass_gimple_pfbv_ave_dfa =
{
    "gdfa_ave",        /* name */
    NULL,              /* gate */
    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

```c
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.

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

Specifying Live Variables Analysis

- Entity should be entity_var
- ⊤, BoundaryInfo and BEnd 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

```c
gdf: Design and Implementation

```struct gimple_pfbv_dfa_spec{
  entity entity;
  top_value_spec initial_value;
  entry_info initial_value;
  exit_info initial_value;
  traversal_order traversal_direction;
  confluence meet_operation;
  gen_effect entity_manipulation;
  gen_exposition entity_occurrence;
  kill_effect entity_manipulation;
  kill_exposition entity_occurrence;
  preserved_dfi dfi_to_be_preserved;
  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

<table>
<thead>
<tr>
<th>Enumerated Type</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity_name</td>
<td>entity_expr, entity_var, entity_defn</td>
</tr>
<tr>
<td>initial_value</td>
<td>ONES, ZEROS</td>
</tr>
<tr>
<td>traversal_direction</td>
<td>FORWARD, BACKWARD, BIDIRECTIONAL</td>
</tr>
<tr>
<td>meet_operation</td>
<td>UNION, INTERSECTION</td>
</tr>
<tr>
<td>entity_operation</td>
<td>entity_use, entity_mod</td>
</tr>
<tr>
<td>entity_occurrence</td>
<td>up_exp, down_exp, any_where</td>
</tr>
<tr>
<td>dfi_to_be_preserved</td>
<td>all, global_only, no_value</td>
</tr>
</tbody>
</table>

### Pre-Defined Edge Flow Functions

#### Edge Flow Functions

<table>
<thead>
<tr>
<th>Edge Flow Function</th>
<th>Returned value</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity_forward_edge_flow(src, dest)</td>
<td>CURRENT_OUT(src)</td>
</tr>
<tr>
<td>identity_backward_edge_flow(src, dest)</td>
<td>CURRENT_IN(dest)</td>
</tr>
<tr>
<td>stop_flow_along_edge(src, dest)</td>
<td>top_value</td>
</tr>
</tbody>
</table>

#### Node Flow Functions

<table>
<thead>
<tr>
<th>Node Flow Function</th>
<th>Returned value</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity_forward_node_flow(bb)</td>
<td>CURRENT_IN(bb)</td>
</tr>
<tr>
<td>identity_backward_node_flow(bb)</td>
<td>CURRENT_OUT(bb)</td>
</tr>
<tr>
<td>stop_flow_along_node(bb)</td>
<td>top_value</td>
</tr>
<tr>
<td>forward_gen_kill_node_flow(bb)</td>
<td>CURRENT_GEN(bb) ∪ CURRENT_IN(bb) - CURRENT_KILL(bb)</td>
</tr>
<tr>
<td>backward_gen_kill_node_flow(bb)</td>
<td>CURRENT_GEN(bb) ∪ CURRENT_OUT(bb) - CURRENT_KILL(bb)</td>
</tr>
</tbody>
</table>

### The Generic Driver for Global Data Flow Analysis

```c
pfbv_dfi ** gdfa_driver(struct simple_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

<table>
<thead>
<tr>
<th>Exposition</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>up_exp</td>
<td>backward</td>
</tr>
<tr>
<td>down_exp</td>
<td>forward</td>
</tr>
<tr>
<td>any_where</td>
<td>don't care</td>
</tr>
</tbody>
</table>

- Solve the recurrence

\[
\text{accumulated_entities} = (\text{accumulated_entities} - \text{remove_entities}) \cup \text{add_entities}
\]

Example for Available Expressions Analysis

Entity is \( \text{entity_expr} \).

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

<table>
<thead>
<tr>
<th>Exposition</th>
<th>Manipulation</th>
<th>( a = b \times c )</th>
<th>( b = b \times c )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>add</td>
<td>remove</td>
<td>add</td>
</tr>
<tr>
<td>upwards</td>
<td>use</td>
<td>( b \times c )</td>
<td>( \text{expr}(a) )</td>
</tr>
<tr>
<td>downwards</td>
<td>use</td>
<td>( b \times c )</td>
<td>( \text{expr}(a) )</td>
</tr>
<tr>
<td>upwards</td>
<td>modification</td>
<td>( \text{expr}(a) )</td>
<td>( b \times c )</td>
</tr>
<tr>
<td>downwards</td>
<td>modification</td>
<td>( \text{expr}(a) )</td>
<td>( b \times c )</td>
</tr>
</tbody>
</table>

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

Future Work

Main thrust
- Supporting general data flow frameworks
- Supporting interprocedural analysis