specRTL: A Language for GCC Machine Descriptions

Uday Khedker

GCC Resource Center,
Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

April 2011
Outline

• Introduction and motivation
• Introduction to specRTL
• Conclusions and Future Work
Outline

- Introduction and motivation
- Introduction to specRTL
- Conclusions and Future Work
- Disclaimer: Preliminary work that is still evolving
Part 1

Introduction
Compilation Models

Aho Ullman Model

Davidson Fraser Model
Compilation Models

Aho Ullman Model

Front End

AST

Input Source Program

Davidson Fraser Model
Compilation Models

Aho Ullman Model

Davidson Fraser Model

Input Source Program

Front End

AST

Optimizer

Target Indep. IR
Compilation Models

Aho Ullman Model

Front End → AST → Optimizer → Target Indep. IR → Code Generator → Target Program

Davidson Fraser Model

Input Source Program

Uday Khedker
GRC, IIT Bombay
Compilation Models

Aho Ullman Model

Front End

AST

Optimizer

Target Indep. IR

Code Generator

Target Program

Davidson Fraser Model

Input Source Program

Front End

AST
Compilation Models

**Aho Ullman Model**

- Front End
  - AST
  - Optimizer
  - Target Indep. IR
  - Code Generator
  - Target Program

**Davidson Fraser Model**

- Front End
  - AST
  - Expander
  - Register Transfers
  - Target Program

---

Uday Khedker

GRC, IIT Bombay
Compilation Models

**Aho Ullman Model**

- Front End
- AST
- Optimizer
- Target Indep. IR
- Code Generator
- Target Program

**Davidson Fraser Model**

- Front End
- AST
- Expander
- Register Transfers
- Optimizer
- Register Transfers

Input Source Program
Compilation Models

**Aho Ullman Model**

- Front End
  - AST
  - Optimizer
  - Target Indep. IR
  - Code Generator
  - Target Program

**Davidson Fraser Model**

- Input Source Program
  - Front End
  - AST
  - Expander
  - Register Transfers
  - Optimizer
  - Register Transfers
  - Recognizer
  - Target Program

Uday Khedker
GRC, IIT Bombay
Compilation Models

Aho Ullman Model

Front End → AST → Optimizer → Target Indep. IR → Code Generator → Target Program

Aho Ullman: Instruction selection
- over optimized IR using
- cost based tree pattern matching

Davidson Fraser Model

Front End → AST → Expander → Register Transfers → Optimizer → Register Transfers → Recognizer → Target Program

Davidson Fraser: Instruction selection
- over AST using
- structural tree pattern matching
- naive code which is
  - target dependent, and is
  - optimized subsequently

Uday Khedker
GRC, IIT Bombay
## Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<thead>
<tr>
<th></th>
<th>Aho Ullman Model</th>
<th>Davidson Fraser Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Selection</td>
<td>• Machine independent IR is expressed in the form of trees</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Machine instructions are described in the form of trees</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Trees in the IR are “covered” using the instruction trees</td>
<td></td>
</tr>
<tr>
<td>Optimization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<thead>
<tr>
<th></th>
<th>Aho Ullman Model</th>
<th>Davidson Fraser Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instruction</strong></td>
<td>• Machine independent IR is expressed in the form of trees</td>
<td></td>
</tr>
<tr>
<td><strong>Selection</strong></td>
<td>• Machine instructions are described in the form of trees</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Trees in the IR are “covered” using the instruction trees</td>
<td></td>
</tr>
<tr>
<td><strong>Optimization</strong></td>
<td>Cost based tree pattern matching</td>
<td></td>
</tr>
</tbody>
</table>
# Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<thead>
<tr>
<th></th>
<th>Aho Ullman Model</th>
<th>Davidson Fraser Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instruction</strong></td>
<td>• Machine independent IR is expressed in the form of trees</td>
<td>• Machine instructions are described in the form of trees</td>
</tr>
<tr>
<td>Selection</td>
<td>• Machine instructions are described in the form of trees</td>
<td>• Trees in the IR are “covered” using the instruction trees</td>
</tr>
<tr>
<td></td>
<td>• Trees in the IR are “covered” using the instruction trees</td>
<td></td>
</tr>
<tr>
<td><strong>Optimization</strong></td>
<td>Cost based tree pattern matching</td>
<td>Structural tree pattern matching</td>
</tr>
</tbody>
</table>

Uday Khedker
GRC, IIT Bombay
## Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<thead>
<tr>
<th>Instruction Selection</th>
<th>Aho Ullman Model</th>
<th>Davidson Fraser Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>● Machine independent IR is expressed in the form of trees</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Machine instructions are described in the form of trees</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Trees in the IR are “covered” using the instruction trees</td>
<td></td>
</tr>
<tr>
<td>Cost based tree pattern matching</td>
<td></td>
<td>Structural tree pattern matching</td>
</tr>
<tr>
<td>Optimization</td>
<td>Machine independent</td>
<td></td>
</tr>
</tbody>
</table>
## Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<thead>
<tr>
<th></th>
<th>Aho Ullman Model</th>
<th>Davidson Fraser Model</th>
</tr>
</thead>
</table>
| **Instruction Selection** | • Machine independent IR is expressed in the form of trees  
• Machine instructions are described in the form of trees  
• Trees in the IR are “covered” using the instruction trees |  
| **Optimization** | Machine independent                                                             | Machine dependent                      |
## Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<thead>
<tr>
<th></th>
<th>Aho Ullman Model</th>
<th>Davidson Fraser Model</th>
</tr>
</thead>
</table>
| **Instruction Selection** | • Machine independent IR is expressed in the form of trees  
• Machine instructions are described in the form of trees  
• Trees in the IR are “covered” using the instruction trees | Cost based tree pattern matching  
Structural tree pattern matching |
| **Optimization**      | Machine independent                                                               | Machine dependent                   |
|                      |                                                                                 | Key Insight: Register transfers are target specific but their form is target independent |
GCC’s Adaptation of Davidson Fraser Model

GROW 2011, Chamonix
specRTL: Introduction

Uday Khedker
GRC, IIT Bombay
## Comparing Code Generators in Davidson Fraser Model

<table>
<thead>
<tr>
<th>Expander</th>
<th>GCC</th>
<th>Zephyr/VPO</th>
<th>Quick C--</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformation Trees ($TT$)</td>
<td>RTL templates</td>
<td>RTL templates</td>
<td>Expansion tiles</td>
</tr>
<tr>
<td>Nature of $TT$</td>
<td>Target dependent</td>
<td>Target dependent</td>
<td>Target independent</td>
</tr>
<tr>
<td>Fixing shapes of $TT$</td>
<td>MD writing</td>
<td>MD writing</td>
<td>Framework design</td>
</tr>
<tr>
<td>$TT \rightarrow Inst$ method</td>
<td>Pattern matching using finite automaton</td>
<td>LR parsing (Yacc based)</td>
<td>Pattern matching</td>
</tr>
<tr>
<td>$TT \rightarrow Inst$ mapping</td>
<td>Fixed manually</td>
<td>Discovered automatically</td>
<td>Discovered automatically</td>
</tr>
<tr>
<td>Time of devising $TT \rightarrow Inst$ mapping</td>
<td>MD writing</td>
<td>Compilation</td>
<td>Compiler construction</td>
</tr>
</tbody>
</table>
Part 2

Motivation
The Need for Improving Machine Descriptions

The Problems:

- The specification mechanism for Machine descriptions is quite adhoc

- Adhoc design decisions
The Need for Improving Machine Descriptions

The Problems:

- The specification mechanism for Machine descriptions is quite adhoc
- Adhoc design decisions
The Need for Improving Machine Descriptions

The Problems:

- The specification mechanism for Machine descriptions is quite adhoc
  - Only syntax borrowed from LISP, neither semantics nor spirit!
  - Non-composable rules
  - Mode and code iterator mechanisms are insufficient

- Adhoc design decisions
The Need for Improving Machine Descriptions

The Problems:

• The specification mechanism for Machine descriptions is quite adhoc
  ▶ Only syntax borrowed from LISP, neither semantics not spirit!
  ▶ Non-composable rules
  ▶ Mode and code iterator mechanisms are insufficient

• Adhoc design decisions
  ▶ Honouring operand constraints delayed to global register allocation
    During GIMPLE to RTL translation, a lot of C code is required
  ▶ Choice of insertion of NOPs
Symptom of Poor Specification Mechanism

- Machine descriptions are large, verbose, repetitive, and contain large chunks of C code
  Size in terms of line counts for gcc-4.5.0

<table>
<thead>
<tr>
<th>Target</th>
<th>*.md</th>
<th>*.c</th>
<th>*.h</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>i386</td>
<td>35766</td>
<td>28643</td>
<td>15694</td>
<td>80103</td>
</tr>
<tr>
<td>mips</td>
<td>12930</td>
<td>12572</td>
<td>5105</td>
<td>30607</td>
</tr>
</tbody>
</table>
Symptom of Poor Specification Mechanism

- Machine descriptions are large, verbose, repetitive, and contain large chunks of C code
  Size in terms of line counts for gcc-4.5.0

<table>
<thead>
<tr>
<th>Target</th>
<th>*.md</th>
<th>*.c</th>
<th>*.h</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>i386</td>
<td>35766</td>
<td>28643</td>
<td>15694</td>
<td>80103</td>
</tr>
<tr>
<td>mips</td>
<td>12930</td>
<td>12572</td>
<td>5105</td>
<td>30607</td>
</tr>
</tbody>
</table>

- Machine descriptions are difficult to construct, understand, debug, and enhance
Typical Instruction Specification in GCC

```
(define_insn
  "movsi"
  [(set
    (match_operand:SI 0 "register_operand" "r")
    (match_operand:SI 1 "const_int_operand" "k"))]

  "" /* C boolean expression, if required */
  "li %0, %1"
)
```
Typical Instruction Specification in GCC

**Define instruction pattern**

```c
(define_insn
  "movsi"
  [(set
     (match_operand:SI 0 "register_operand" "r")
     (match_operand:SI 1 "const_int_operand" "k"))]

  "li %0, %1"

  /* C boolean expression, if required */

  "li %0, %1"
)
```

**Standard Pattern Name**

**RTL Expression (RTX):**
Semantics of target instruction

**target asm inst. =**
Concrete syntax for RTX
Typical Instruction Specification in GCC

(define_insn "movsi"
  [(set
     (match_operand:SI 0 "register_operand" "r")
     (match_operand:SI 1 "const_int_operand" "k"))]
  "li %0, %1"
  /* C boolean expression, if required */
  "li %0, %1"
Design Flaws in Machine Descriptions

Multiple patterns with same structure

- Repetition of almost similar RTL expressions across multiple define_insn and define_expand patterns
  - Some Modes, Predicates, Constraints, Boolean Condition, or RTL Expression may differ everything else may be identical
  - One RTL expression may appears as a sub-expression of some other RTL expression

- Repetition of C code along with RTL expressions in these patterns.
Redundancy in MIPS Machine Descriptions: Example 1

\[
\begin{align*}
&\text{(set (match_operand:} m 0 "register_operand" "c0")} \\
&(\text{plus:} m \text{ (match_operand:} m 1 "register_operand" "c1")} \\
&(\text{match_operand:} m 2 "p" "c2"))))
\end{align*}
\]
Redundancy in MIPS Machine Descriptions: Example 1

\[
[(\text{set} \ (\text{match}\_\text{operand}:m\ 0 \ "\text{register}\_\text{operand}" \ "c0")
\text{ (plus}:m\ (\text{match}\_\text{operand}:m\ 1 \ "\text{register}\_\text{operand}" \ "c1")
\text{ (match}\_\text{operand}:m\ 2 \ "p" \ "c2")))]
\]
Redundancy in MIPS Machine Descriptions: Example 1

\[
\text{[(set (match Operand: } m \text{ 0 "register_operand" "c0")}
\]
\[
\text{(plus: } m \text{ (match Operand: } m \text{ 1 "register_operand" "c1")}
\]
\[
\text{(match Operand: } m \text{ 2 "p" "c2")})]
\]
Redundancy in MIPS Machine Descriptions: Example 2

\[
\begin{aligned}
&\text{(set (match_operand: } m \text{ 0 "register_operand" "c0")}
\\&
&\text{(mult: } m \text{ (match_operand: } m \text{ 1 "register_operand" "c1")}
\\&
&\text{(match_operand: } m \text{ 2 "register_operand" "c2")))}
\end{aligned}
\]
Redundancy in MIPS Machine Descriptions: Example 2

\[
\begin{align*}
&\text{[(set (match_operand:}_m\ 0 \ "register_operand" \ "c0")}
&\quad (\text{mult:}_m (\text{match_operand:}_m\ 1 \ "register_operand" \ "c1")}
&\quad (\text{match_operand:}_m 2 \ "register_operand" \ "c2")))]
\end{align*}
\]
Redundancy in MIPS Machine Descriptions: Example 2

\[
\begin{align*}
\text{(set (match_operand: } m \ 0 \ "\text{register_operand}" \ "c0")} \\
\text{(mult: } m \ (\text{match_operand: } m \ 1 \ "\text{register_operand}" \ "c1")} \\
\text{(match_operand: } m \ 2 \ "\text{register_operand}" \ "c2"))\end{align*}
\]

**RTL Template**

- Structure
- Details

<table>
<thead>
<tr>
<th>Pattern name</th>
<th>m</th>
<th>c0</th>
<th>c1</th>
<th>c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>define_insn *mul&lt;mode&gt;3</td>
<td>SCALARF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>define_insn *mul&lt;mode&gt;3_r4300</td>
<td>SCALARF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>define_insn mulv2sf3</td>
<td>V2SF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>define_expand_mul&lt;mode&gt;3</td>
<td>GPR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>define_insn mul&lt;mode&gt;3_mul3_loongson</td>
<td>GPR</td>
<td>=d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>define_insn mul&lt;mode&gt;3_mul3</td>
<td>GPR</td>
<td>d,1</td>
<td>d,d</td>
<td>d,d</td>
</tr>
</tbody>
</table>
Redundancy in MIPS Machine Descriptions: Example 3

\[
\text{(set } (\text{match$_{\text{operand}}$:}_m 0 \text{ "register$_{\text{operand}}$" } \text{"}_c0\text{"}) \text{ (plus:}_m \\
(\text{mult:}_m (\text{match$_{\text{operand}}$:}_m 1 \text{ "register$_{\text{operand}}$" } \text{"}_c1\text{"}) \\
(\text{match$_{\text{operand}}$:}_m 2 \text{ "register$_{\text{operand}}$" } \text{"}_c2\text{"})))) \\
(\text{match$_{\text{operand}}$:}_m 3 \text{ "register$_{\text{operand}}$" } \text{"}_c3\text{"}))))
\]

*RTL Template*
Redundancy in MIPS Machine Descriptions: Example 3

\[
\text{RTL Template} = \left( \begin{array}{c}
\text{Structure} \\
\end{array} \right)
\]

\[
[(\text{set (match_operand: } m \ 0 \ "\text{register_operand}" \ "c0") \ (\text{plus: } m \\
(\text{mult: } m \ (\text{match_operand: } m \ 1 \ "\text{register_operand}" \ "c1") \\
(\text{match_operand: } m \ 2 \ "\text{register_operand}" \ "c2")))]) \\
(\text{match_operand: } m \ 3 \ "\text{register_operand}" \ "c3"))]
\]
Redundancy in MIPS Machine Descriptions: Example 3

\[
[(\text{set}(\text{match\_operand}: m 0 "register\_operand" "c0") \ (\text{plus}: m \\
(\text{mult}: m (\text{match\_operand}: m 1 "register\_operand" "c1") \\
(\text{match\_operand}: m 2 "register\_operand" "c2")))) \\
(\text{match\_operand}: m 3 "register\_operand" "c3")))]
\]

**RTL Template**

**Structure**

**Details**

<table>
<thead>
<tr>
<th>Pattern name</th>
<th>(m)</th>
<th>(c0)</th>
<th>(c1)</th>
<th>(c2)</th>
<th>(c3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>*mul_acc_si</td>
<td>SI</td>
<td>=l*???,d?</td>
<td>d,d</td>
<td>d,d</td>
<td>0,d</td>
</tr>
<tr>
<td>*mul_acc_si_r3900</td>
<td>SI</td>
<td>=l*???,d??,d?</td>
<td>d,d,d</td>
<td>d,d,d</td>
<td>0,1,d</td>
</tr>
<tr>
<td>*macc</td>
<td>SI</td>
<td>=l,d</td>
<td>d,d</td>
<td>d,d</td>
<td>0,1</td>
</tr>
<tr>
<td>*madd4&lt;mode&gt;</td>
<td>ANYF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>*madd3&lt;mode&gt;</td>
<td>ANYF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
<td>0</td>
</tr>
</tbody>
</table>
Insufficient Iterator Mechanism

- Iterators cannot be used across define_insn, define_expand, define_peephole2 and other patterns
- Defining iterator attribute for each varying parameter becomes tedious
- For same set of modes and rtx codes, change in other fields of pattern makes use of iterators impossible
- Mode and code attributes cannot be defined for operator or operand number, name of the pattern etc.
- Patterns with different RTL template share attribute value vector for which iterators cannot be used
Many Similar Patterns Cannot be Combined

(define_expand "iordi3"
  [(set (match_operand:DI 0 "nonimmediate_operand" "")
        (ior:DI (match_operand:DI 1 "nonimmediate_operand" "")
                (match_operand:DI 2 "x86_64_general_operand" "")))]
  (clobber (reg:CC FLAGS_REG)))
"TARGET_64BIT"
"ix86_expand_binary_operator (IOR, DImode, operands); DONE;"

(define_insn "*iordi_1_rex64"
  [(set (match_operand:DI 0 "nonimmediate_operand" "=rm,r")
        (ior:DI (match_operand:DI 1 "nonimmediate_operand" "%0,0")
                (match_operand:DI 2 "x86_64_general_operand" "re,rme")))]
  (clobber (reg:CC FLAGS_REG)))
"TARGET_64BIT"
&& ix86_binary_operator_ok (IOR, DImode, operands)"
"or{q}\t{%2, %0|%0, %2}"
[(set_attr "type" "alu")
 (set_attr "mode" "DI")]}
# Measuring Redundancy in RTL Templates

<table>
<thead>
<tr>
<th>MD File</th>
<th>Total number of patterns</th>
<th>Number of primitive trees</th>
<th>Number of times primitive trees are used to create composite trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>i386.md</td>
<td>1303</td>
<td>349</td>
<td>4308</td>
</tr>
<tr>
<td>arm.md</td>
<td>534</td>
<td>232</td>
<td>1369</td>
</tr>
<tr>
<td>mips.md</td>
<td>337</td>
<td>147</td>
<td>921</td>
</tr>
</tbody>
</table>
Part 3

Introduction to specRTL
Key Observation Behind specRTL

- Davidson Fraser insight

  Register transfers are target specific but their form is target independent

- GCC’s approach
  - Use Target independent RTL for machine specification
  - Generate expander and recognizer by reading machine descriptions

Main problems with GCC’s Approach

  Although the shapes of RTL statements are target independent, they have to be provided in RTL templates

- Our key idea:

  Separate shapes of RTL statements from the target specific details
Specification Goals of specRTL

Support all of the following

- Separation of shapes from target specific details
- Creation of new shapes by composing shapes
- Associating concrete details with shapes
- Overriding concrete details
Software Engineering Goals of specRTL

- Allow non-disruptive migration for existing machine descriptions
  - Incremental changes
  - No need to change GCC source until we are sure of the new specification

GCC must remain usable after each small change made in the machine descriptions
Meeting the Specification Goals: Key Idea

- Separation of shapes from target specific details:
  - Shape $\equiv$ tree structure of RTL templates
  - Details $\equiv$ attributes of tree nodes
    (eg. modes, predicates, constraints etc.)
Meeting the Specification Goals: Key Idea

- Separation of shapes from target specific details:
  - Shape $\equiv$ tree structure of RTL templates
  - Details $\equiv$ attributes of tree nodes
    (eg. modes, predicates, constraints etc.)

- Abstract patterns and Concrete patterns
  - Abstract patterns are shapes with “holes” in them that represent missing information
  - Concrete patterns are shapes in which all holes are plugged in using target specific information
Meeting the Specification Goals: Key Idea

- Separation of shapes from target specific details:
  - Shape $\equiv$ tree structure of RTL templates
  - Details $\equiv$ attributes of tree nodes
    (eg. modes, predicates, constraints etc.)

- *Abstract patterns* and *Concrete patterns*
  - Abstract patterns are shapes with “holes” in them that represent missing information
  - Concrete patterns are shapes in which all holes are plugged in using target specific information

- Abstract patterns capture *shapes* which can be concretized by providing details
Meeting the Specification Goals: Operations

- Creating new shapes by composing shapes: \texttt{extends}
Meeting the Specification Goals: Operations

- Creating new shapes by composing shapes: \textit{extends}

- Associating concrete details with shapes: \textit{instantiates}
Meeting the Specification Goals: Operations

- Creating new shapes by composing shapes: `extends`

- Associating concrete details with shapes: `instantiates`

- Overriding concrete details: `overrides`
## Properties of Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Base pattern</th>
<th>Derived pattern</th>
<th>Nodes influenced</th>
<th>Can change</th>
</tr>
</thead>
<tbody>
<tr>
<td>extends</td>
<td>Abstract</td>
<td>Abstract</td>
<td>Leaf nodes</td>
<td>Structure</td>
</tr>
<tr>
<td>instantiates</td>
<td>Abstract</td>
<td>Concrete</td>
<td>All nodes</td>
<td>Attributes</td>
</tr>
<tr>
<td>overrides</td>
<td>Abstract</td>
<td>Abstract</td>
<td>Internal nodes</td>
<td>Attributes</td>
</tr>
<tr>
<td></td>
<td>Concrete</td>
<td>Concrete</td>
<td>All nodes</td>
<td>Attributes</td>
</tr>
</tbody>
</table>
abstract set_plus extends set
{
  root.2 = plus;
}

abstract set_macc extends set_plus
{
  root.2.2 = mult;
}
Creating Concrete Patterns

abstract set_plus extends set
{
    root.2 = plus;
}

cancrete add<mode>3 insn instantiates set_plus
{
    set_plus(register_operand:ANYF:"=f",
             register_operand:ANYF:"f",
             register_operand:ANYF:"f");
    root.2.mode = ANYF;
}

cancrete add<mode>3 expand instantiates set_plus
{
    set_plus(register_operand:GPR:"",
             register_operand:GPR:"",
             arith_operand:GPR:"");
    root.2.mode = GPR;
}
Generating Conventional Machine Descriptions

```
abstract set_plus extends set
{
  root.2 = plus;
}
```

```
concrete add<mode>3.insn instantiates set_plus
{
  set_plus(register_operand:ANYF:"=f", register_operand:ANYF:"f",
           register_operand:ANYF:"f");
  root.2.mode = ANYF;
}
```

```
(define_insn "add<mode>3"
[(set (match_operand:ANYF 0 "register_operand" "=f")
      (plus:ANYF (match_operand:ANYF 1 "register_operand" "f")
                 (match_operand:ANYF 2 "register_operand" "f")))]
  ""
```
Generating Conventional Machine Descriptions

abstract set_plus extends set
{
    root.2 = plus;
}

congrete add<mode>3.insn instantiates set_plus
{
    set_plus(register_operand:ANYF:"=f", register_operand:ANYF:"f",
             register_operand:ANYF:"f");
    root.2.mode = ANYF;
}

/* Conventional Machine Description Fragments */
Generating Conventional Machine Descriptions

abstract set_plus extends set
{
    root.2 = plus;
}

concrete add<mode>3.insn instantiates set_plus
{
    set_plus(register_operand:ANYF:"=f", register_operand:ANYF:"f",
             register_operand:ANYF:"f");
    root.2.mode = ANYF;
}

/* Conventional Machine Description Fragments */

Resulting MD Specification

(define_insn "add<mode>3"
[(set (match_operand:ANYF 0 "register_operand" ":=f")
   (plus:ANYF (match_operand:ANYF 1 "register_operand" "f")
             (match_operand:ANYF 2 "register_operand" "f")))
/* Conventional Machine Description Fragments */
)
abstract set_plus extends set
{
    root.2 = plus;
}

concrete add<mode>3.expand instantiates set_plus
{
    set_plus(register_operand:GPR:"",
             register_operand:GPR:"",
             arith_operand:GPR:""),
    root.2.mode = GPR;
}
abstract set_plus extends set
{
  root.2 = plus;
}

concrete add<mode>3.expand instantiates set_plus
{
  set_plus(register_operand:GPR:"",
            register_operand:GPR:"",
            arith_operand:GPR:""); 
  root.2.mode = GPR;
}

concrete *add<mode>3.insn overrides add<mode>3.expand
{
  allconstraints = ("=d,d", "d,d", "d,Q");
}
Omitting conventional MD fragments

concrete *mul<mode>3 insn instantiates set_mult
{ set_mult(register_operand:SCALARF:"=f",
    register_operand:SCALARF:"f",
    register_operand:SCALARF:"f");
    root.2.mode = SCALARF;
}

concrete *mul<mode>3_r4300 insn overrides *mul<mode>3 insn
{ }
Some More Examples

Omitting conventional MD fragments

concrete *mul<mode>3.insn instantiates set_mult
{ set_mult(register_operand:SCALARF:"=f",
    register_operand:SCALARF:"f",
    register_operand:SCALARF:"f");
    root.2.mode = SCALARF;
}

concrete *mul<mode>3_r4300.insn overrides *mul<mode>3.insn
{}
concrete mulv2sf3 overrides *mul<mode>3.insn
{ SCALARF -> V2SF; }

Uday Khedker  
GRC, IIT Bombay
Part 4

Conclusions
Current Status and Plans for Future Work

- specRTL parser has been augmented with semantic checks
  Emitting conventional machine descriptions is pending
- i386 move instructions and spim add instructions have been rewritten
  Other instructions are being rewritten
- Suggestions have been received to improve the syntax
Conclusions

- Separating shapes from concrete details is very helpful
- It may be possible to identify a large number of common shapes
- Machine descriptions may become much smaller
  Only the concrete details need to be specified
- Non-disruptive and incremental migration to new machine descriptions
- GCC source need not change until these machine descriptions have been found useful
Last but not the least . . .

Thank You!