Introduction to Machine Descriptions

Uday Khedker

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

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

13 June 2014

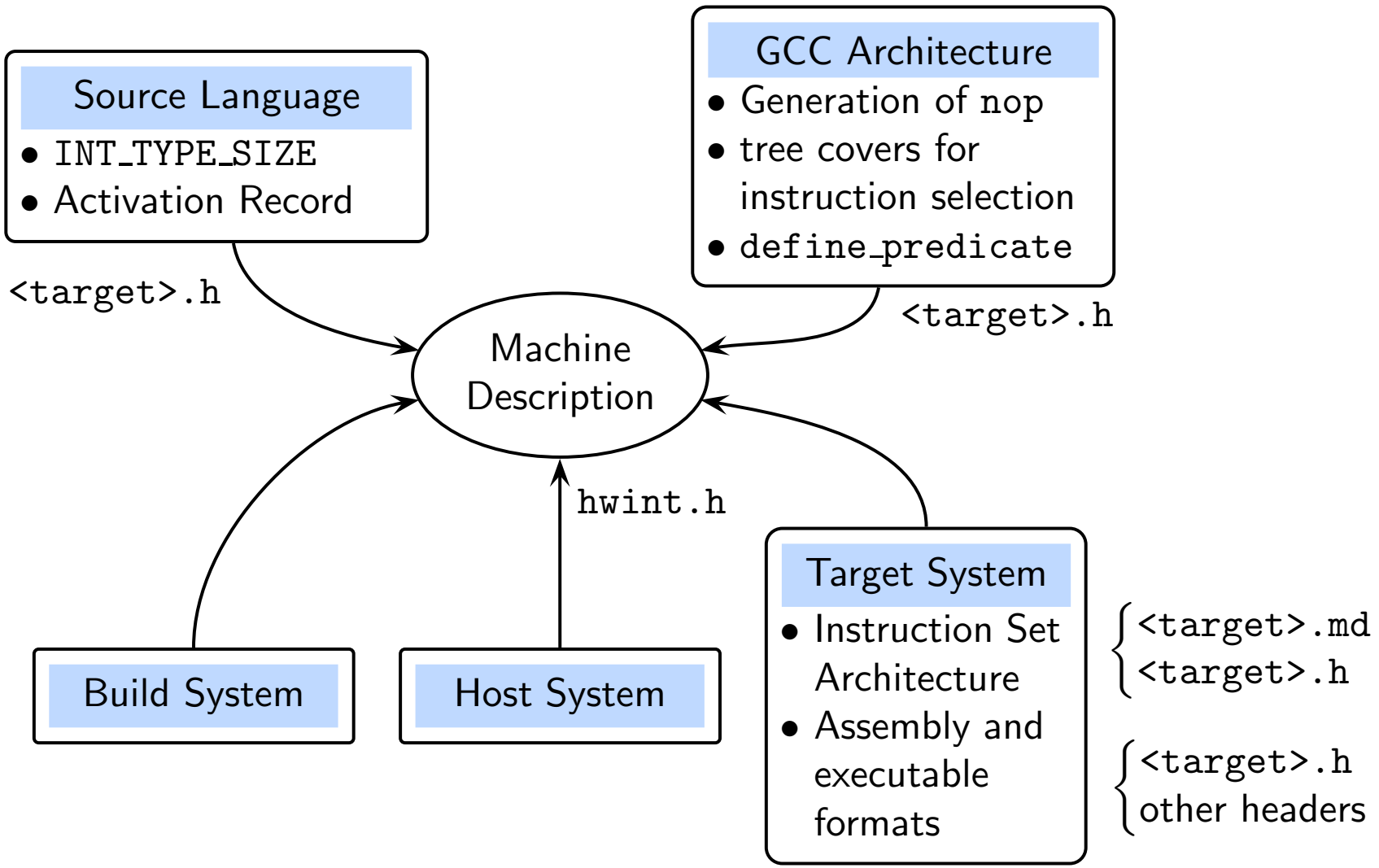
Outline

- Influences on GCC Machine Descriptions
- Organization of GCC Machine Descriptions
- Machine description constructs
- The essence of retargetability in GCC

Part 1

Influences on Machine Descriptions

Examples of Influences on the Machine Descriptions



Part 2

Organization of GCC MD

GCC Machine Descriptions

- Processor instructions useful to GCC
- Processor characteristics useful to GCC
- Target ASM syntax
- Target specific optimizations as IR-RTL $\rightarrow$ IR-RTL transformations

  (GCC code performs the transformation computations,
  MD supplies their *target patterns*)

  - Peephole optimizations
  - Transformations for enabling scheduling

Syntactic Entities in GCC MD

- Necessary Specifications
  - Processor instructions useful to GCC
    - One GIMPLE → One IR-RTL `define_insn`
    - One GIMPLE → More than one IR-RTL `define_expand`
  - Processor characteristics useful to GCC `define_cpu_unit`
  - Target ASM syntax part of `define_insn`
  - IR-RTL → IR-RTL transformations `define_split`
  - Target Specific Optimizations `define_peephole2`
- Programming Conveniences

  (eg. `define_insn_and_split`, `define_constants`, `define_cond_exec`, `define_automaton`)

File Organization of GCC MD

The GCC MD comprises of

- `<target>.h`: A set of C macros that describe
  - HLL properties: e.g. `INT_TYPE_SIZE` to h/w bits
  - Activation record structure
  - Target Register (sub)sets, and characteristics (lists of read-only regs, dedicated regs, etc.)
  - System Software details: formats of assembler, executable etc.
- `<target>.md`: Target instructions described using MD constructs.
- `<target>.c`: Optional, but usually required.
  C functions that implement target specific code
  (e.g. target specific activation layout).

File Organization of GCC MD

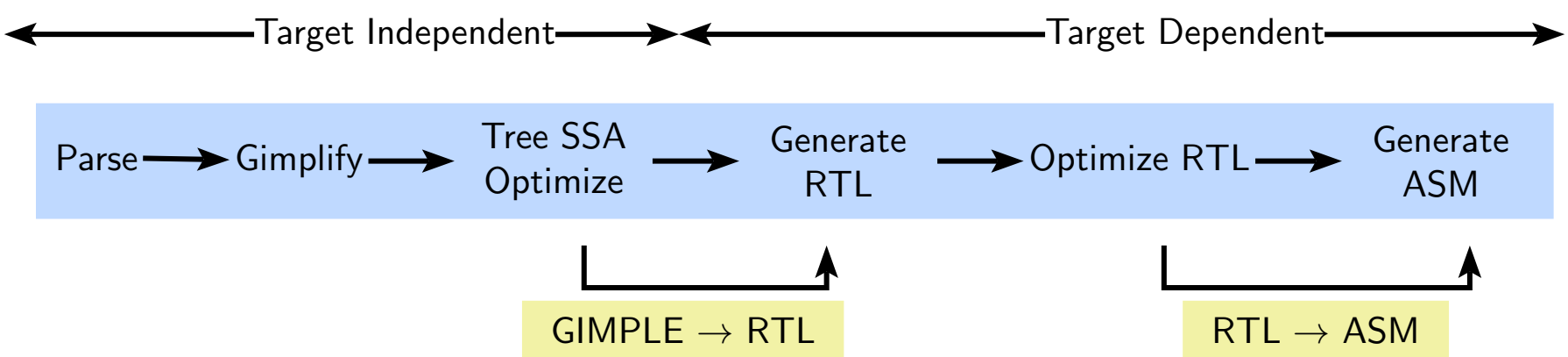
The GCC MD comprises of

- `<target>.h`: A set of C macros that describe
  - HLL properties: e.g. `INT_TYPE_SIZE` to h/w bits
  - Activation record structure
  - Target Register (sub)sets, and characteristics
    (lists of read-only regs, dedicated regs, etc.)
  - System Software details: formats of assembler, executable etc.
- `<target>.md`: Target instructions described using MD constructs.
  (Our main interest!)
- `<target>.c`: Optional, but usually required.
  C functions that implement target specific code
  (e.g. target specific activation layout).

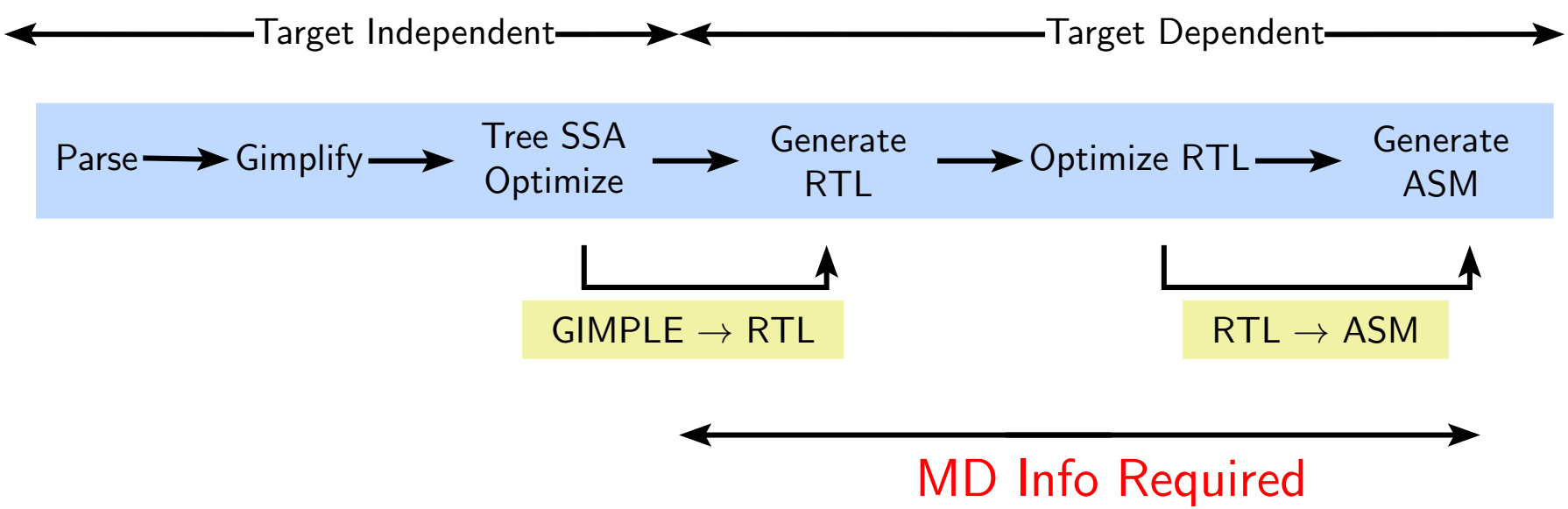
Part 3

Essential Constructs in Machine Descriptions

The GCC Phase Sequence



The GCC Phase Sequence



The GCC Phase Sequence

Observe that

- RTL is a target specific IR
- GIMPLE $\rightarrow$ non strict RTL $\rightarrow$ strict RTL.
- Standard Pattern Name (SPN):

  "Semantic Glue" between GIMPLE and RTL

  - operator match + coarse operand match, and
  - refine the operand match
- Finally: Strict RTL $\Leftrightarrow$ Unique target ASM string

Consider generating RTL expressions of GIMPLE nodes

- Two constructs available: `define_insn` and `define_expand`

Running Example

Consider a *data move* operation

- reads data from source location, and
- writes it to the destination location.
- GIMPLE node: `GIMPLE_ASSIGN`
- SPN: "`movsi`"

Some possible combinations are:

- Reg $\leftarrow$ Reg : Register move
- Reg $\leftarrow$ Mem : Load
- Reg $\leftarrow$ Const : Load immediate
- Mem $\leftarrow$ Reg : Store
- Mem $\leftarrow$ Mem : Illegal instruction
- Mem $\leftarrow$ Const : Illegal instruction

Specifying Target Instruction Semantics

```
(define_insn
   "movsi"
    (set
      (match_operand 0 "register_operand" "=r")
      (match_operand 1 "const_int_operand" "k")
      )
   "" /*  C boolean expression, if required */
   "li %0, %1"
)
```


Specifying Target Instruction Semantics

Introduce instruction pattern

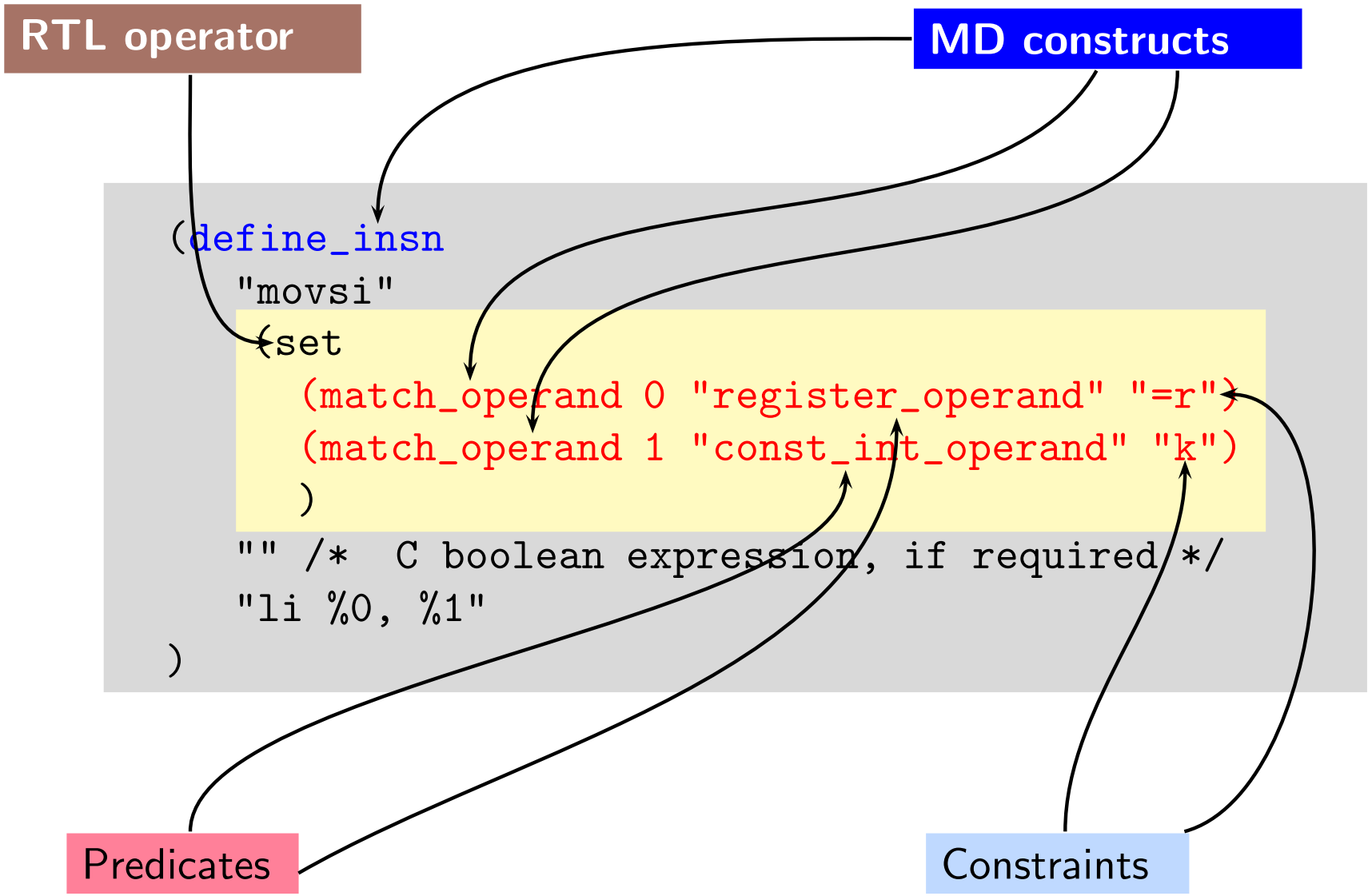
Standard Pattern Name

```
(define_insn
   "movsi"
    (set
      (match_operand 0 "register_operand" "=r")
      (match_operand 1 "const_int_operand" "k")
      )
   "" /*  C boolean expression, if required */
   "li %0, %1"
)
```

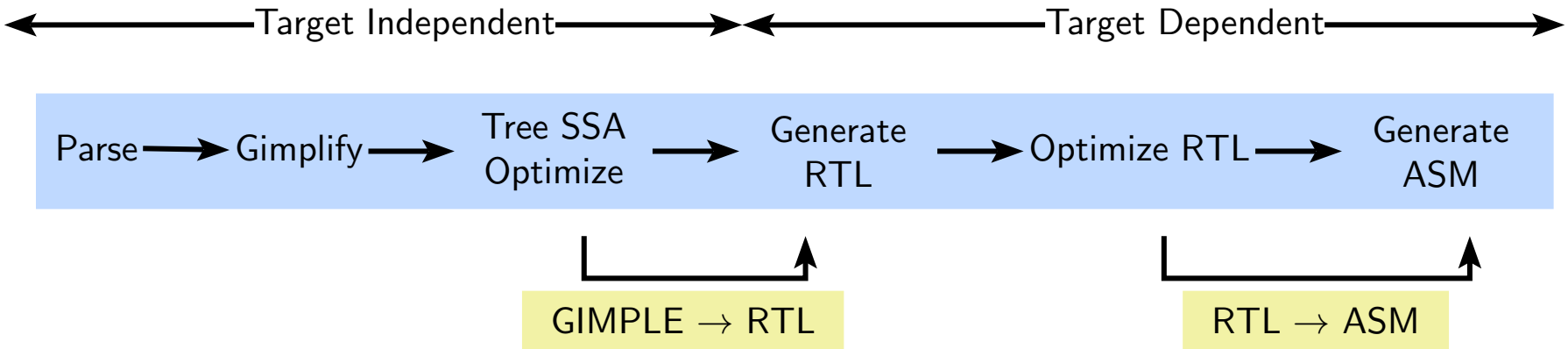
RTL Expression (RTX):
Semantics of target instruction

Target assembly instruction =
Concrete syntax for RTX

Specifying Target Instruction Semantics

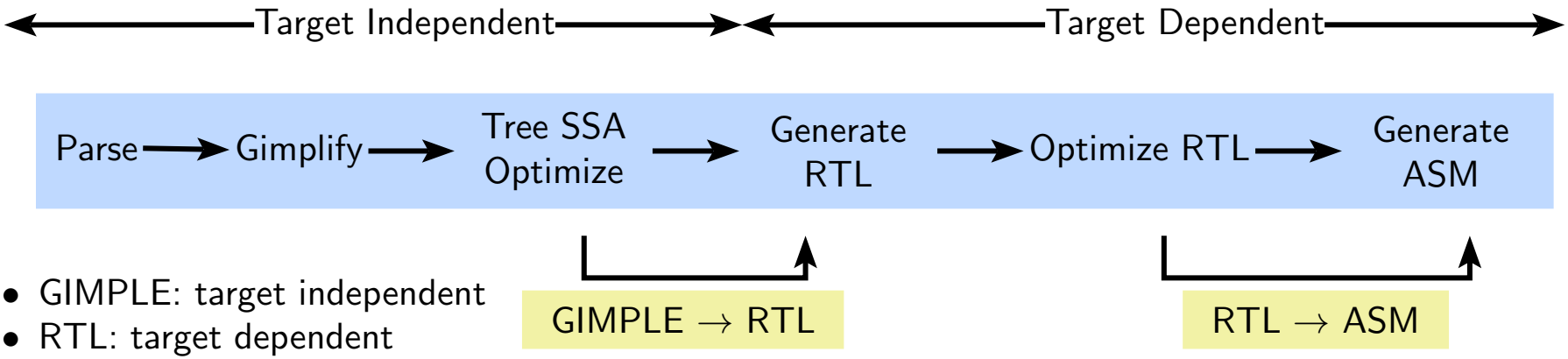


Instruction Specification and Translation




```
(define_insn "movsi"
   (set (match_operand 0 "register_operand" "=r")
        (match_operand 1 "const_int_operand" "k"))
   "" /* C boolean expression, if required */
   "li %0, %1"
)
```


Instruction Specification and Translation

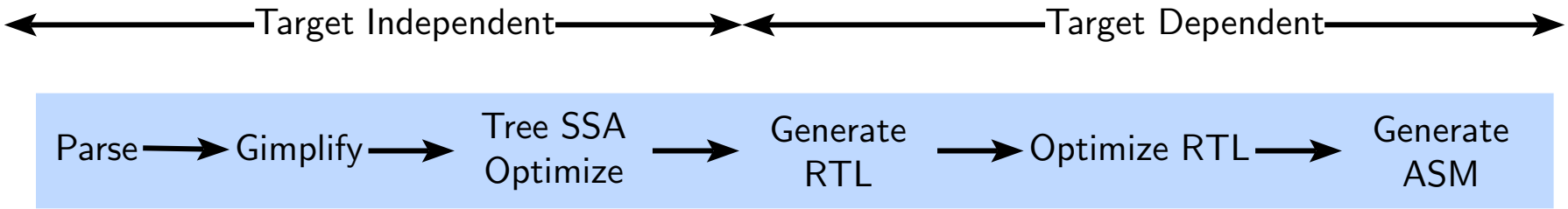


- GIMPLE: target independent
- RTL: target dependent
- Need: associate the *semantics*

⇒GCC Solution: Standard Pattern Names

```
(define_insn "movsi"
   (set (match_operand 0 "register_operand" "=r")
        (match_operand 1 "const_int_operand" "k"))
   "" /* C boolean expression, if required */
   "li %0, %1"
)
```


Instruction Specification and Translation



- GIMPLE: target independent
- RTL: target dependent
- Need: associate the *semantics*

⇒GCC Solution: Standard Pattern Names

GIMPLE → RTL

RTL → ASM

RTL Template

ASM

GIMPLE_ASSIGN

```
(define_insn "movsi"
   (set (match_operand 0 "register_operand" "=r")
        (match_operand 1 "const_int_operand" "k"))
   "" /* C boolean expression, if required */
   "li %0, %1"
)
```


General Move Instruction

```
(define_insn "maybe_spn_like_movsi"
    (set  (match_operand 0 "general_operand" "")
          (match_operand 1 "general_operand" ""))
    ""
    "mov  %0, %1"
)
```

- This `define_insn` can generate data movement patterns of all combinations
- Even Mem $\rightarrow$ Mem is possible.
- We need a mechanism to generate more restricted data movement RTX instances!

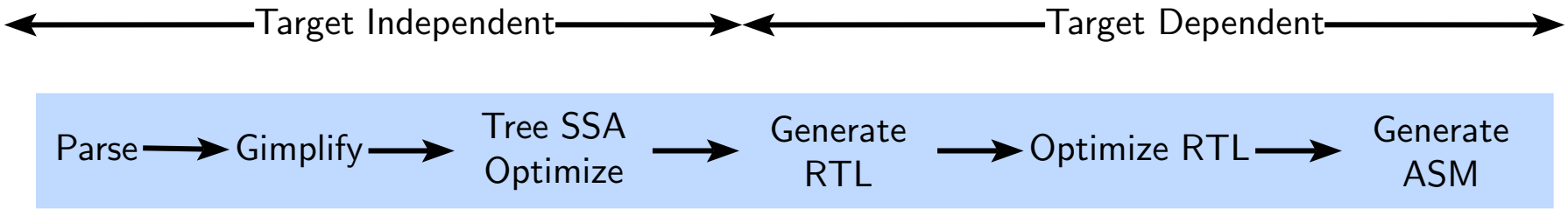
The define_expand Construct

```
(define_expand "movsi"
  [(set (match_operand:SI 0 "nonimmediate_operand" "")
        (match_operand:SI 1 "general_operand" "")
    )]
  ""
  {
    if (GET_CODE (operands[0]) == MEM &&
        GET_CODE (operands[1]) != REG)
      if (can_create_pseudo_p())
        operands[1] = force_reg (SImode, operands[1]);
  }
)
```


Part 4

The Essence of Retargetability

Instruction Specification and Translation: A Recap



- GIMPLE: target independent
- RTL: target dependent
- Need: associate the *semantics*

⇒GCC Solution: Standard Pattern Names

GIMPLE → RTL

RTL → ASM

RTL Template

ASM

GIMPLE_ASSIGN

```
(define_insn "movsi"
   (set (match_operand 0 "register_operand" "=r")
        (match_operand 1 "const_int_operand" "k"))
   "" /* C boolean expression, if required */
   "li %0, %1"
)
```

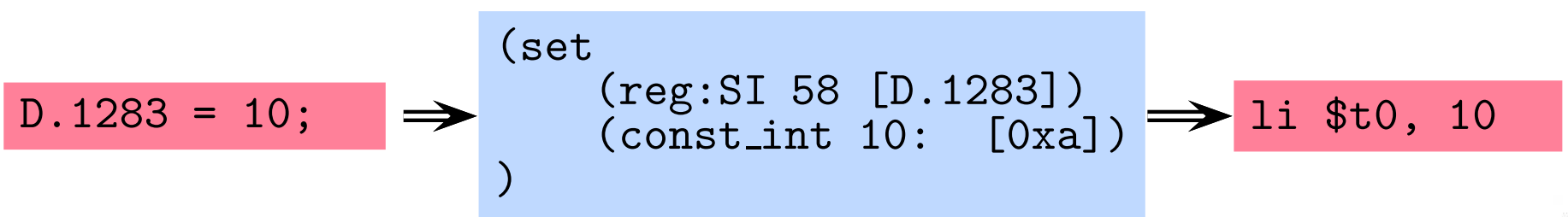

Translation Sequence in GCC

```
(define_insn
   "movsi"
    (set
      (match_operand 0 "register_operand" "=r")
      (match_operand 1 "const_int_operand" "k")
      )
   "" /*  C boolean expression, if required */
   "li %0, %1"
)
```


Translation Sequence in GCC

```
(define_insn
   "movsi"
    (set
      (match_operand 0 "register_operand" "=r")
      (match_operand 1 "const_int_operand" "k")
      )
   "" /*  C boolean expression, if required */
   "li %0, %1"
)
```

Development



Use

The Essence of Retargetability

When are the machine descriptions read?

The Essence of Retargetability

When are the machine descriptions read?

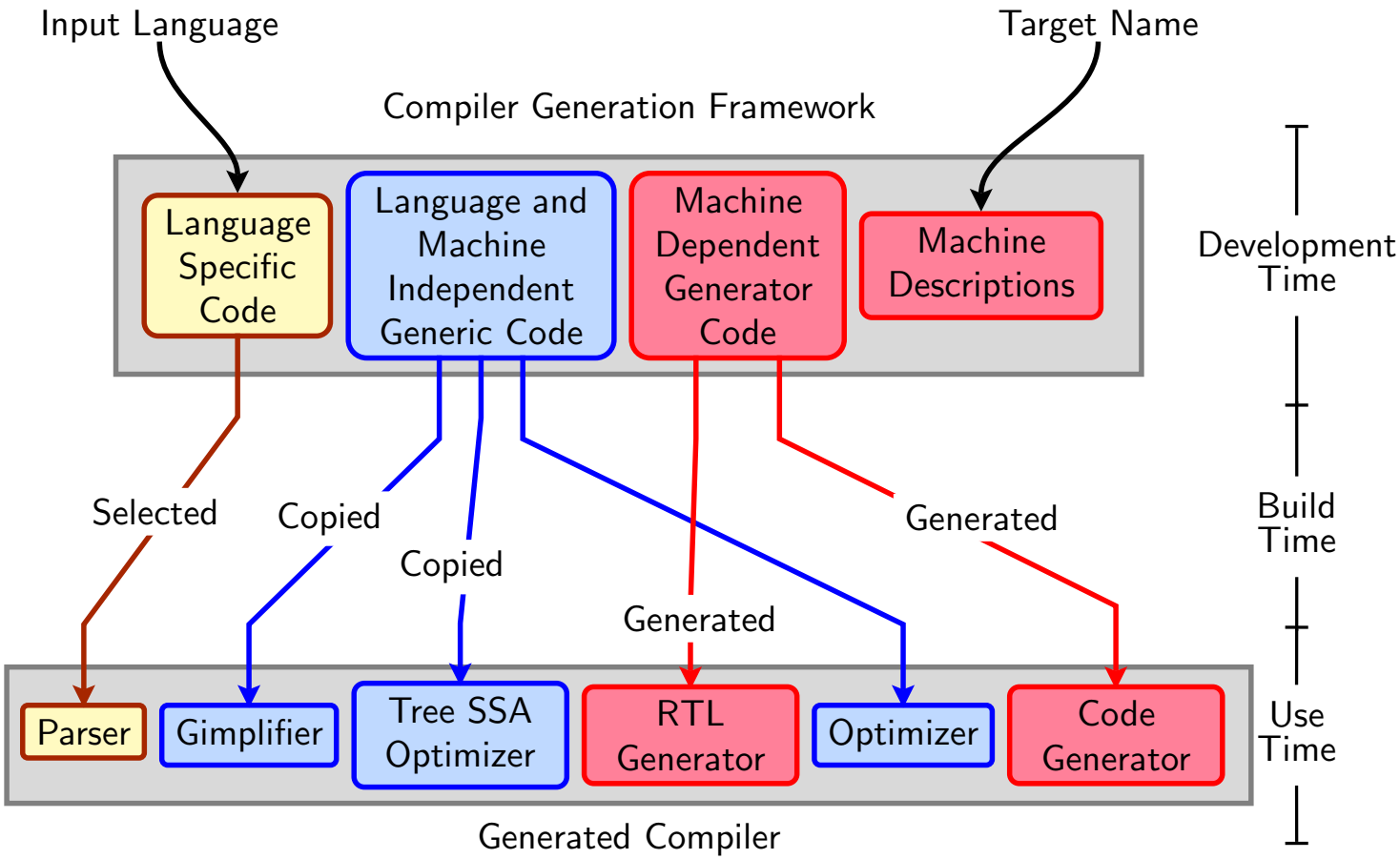
- During the build process

The Essence of Retargetability

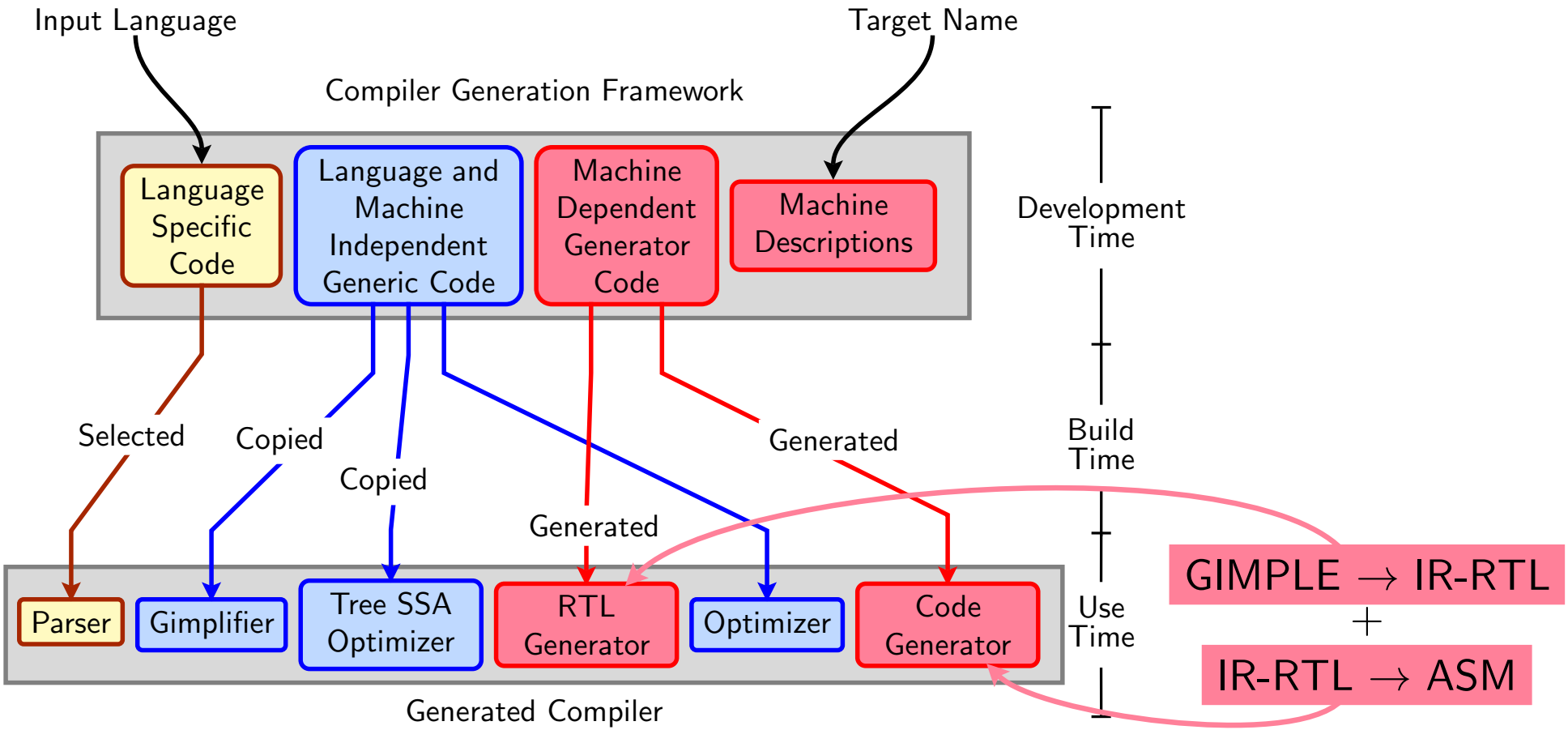
When are the machine descriptions read?

- During the build process
- When a program is compiled by gcc the information gleaned from machine descriptions is consulted

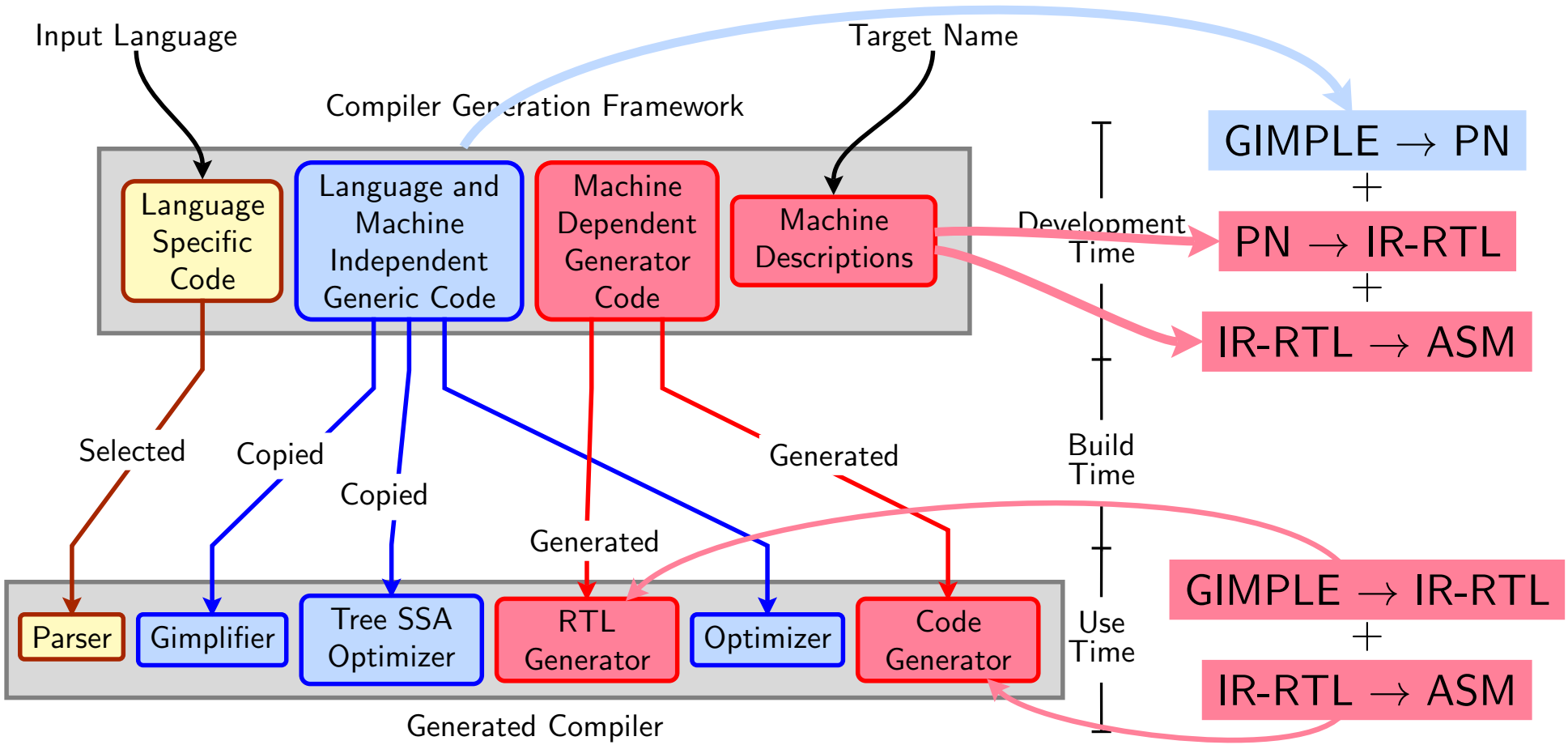
Retargetability Mechanism of GCC



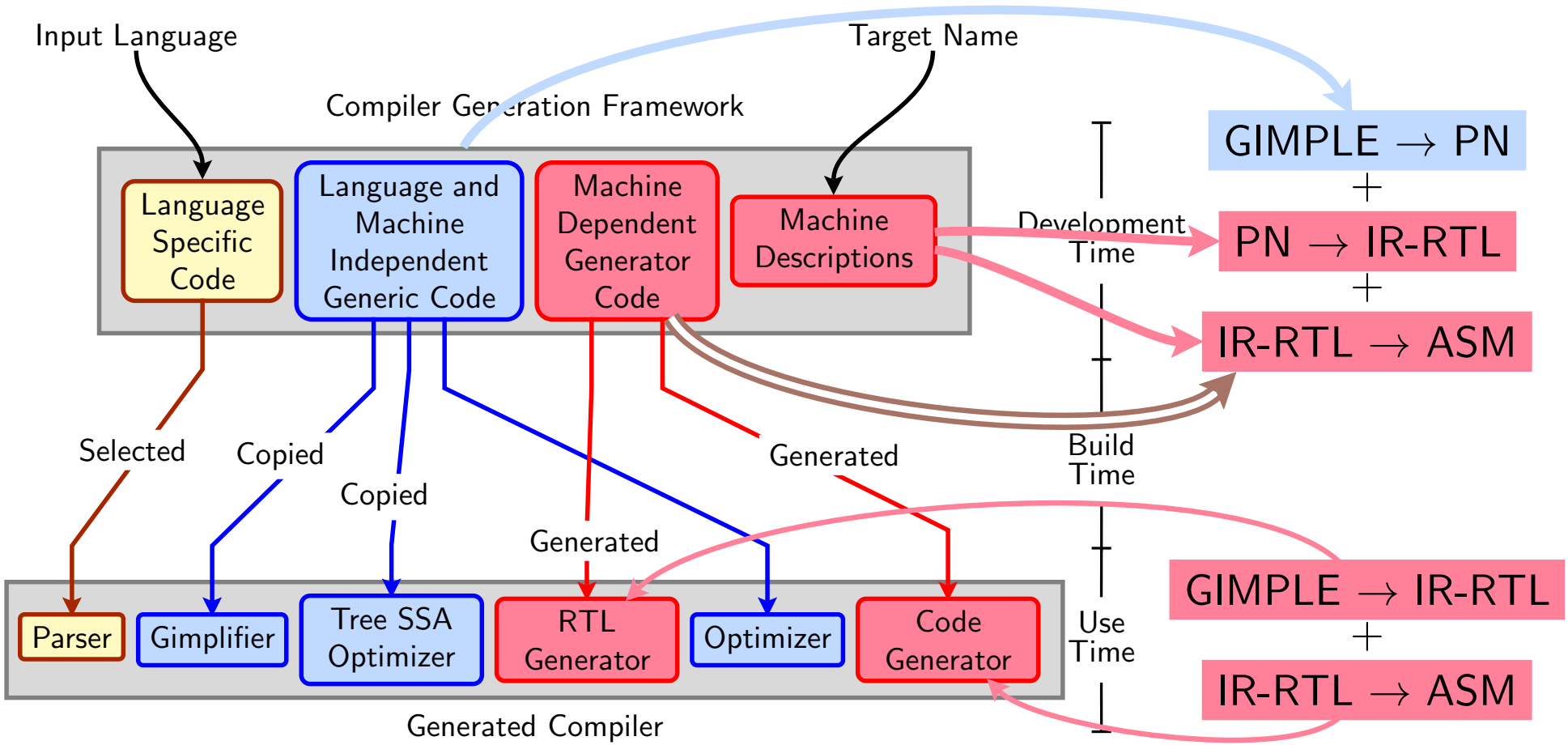
Retargetability Mechanism of GCC



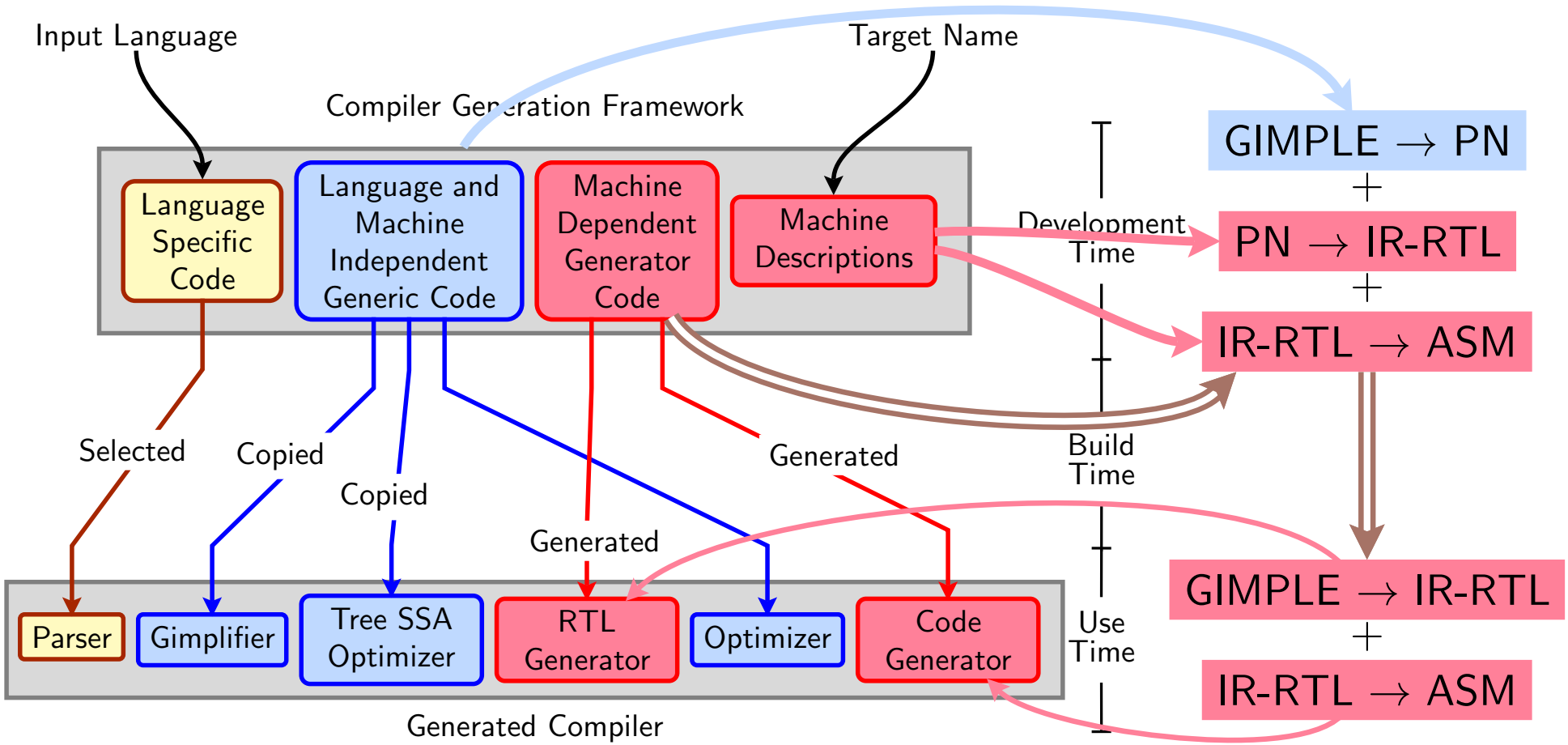
Retargetability Mechanism of GCC



Retargetability Mechanism of GCC



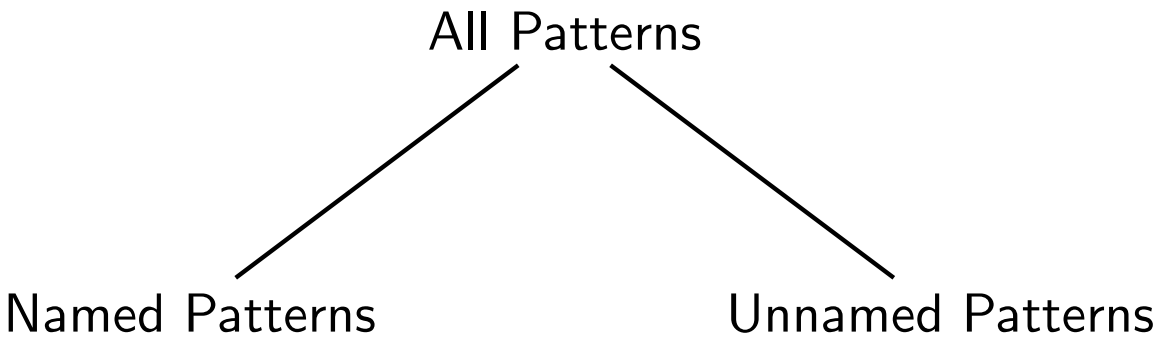
Retargetability Mechanism of GCC



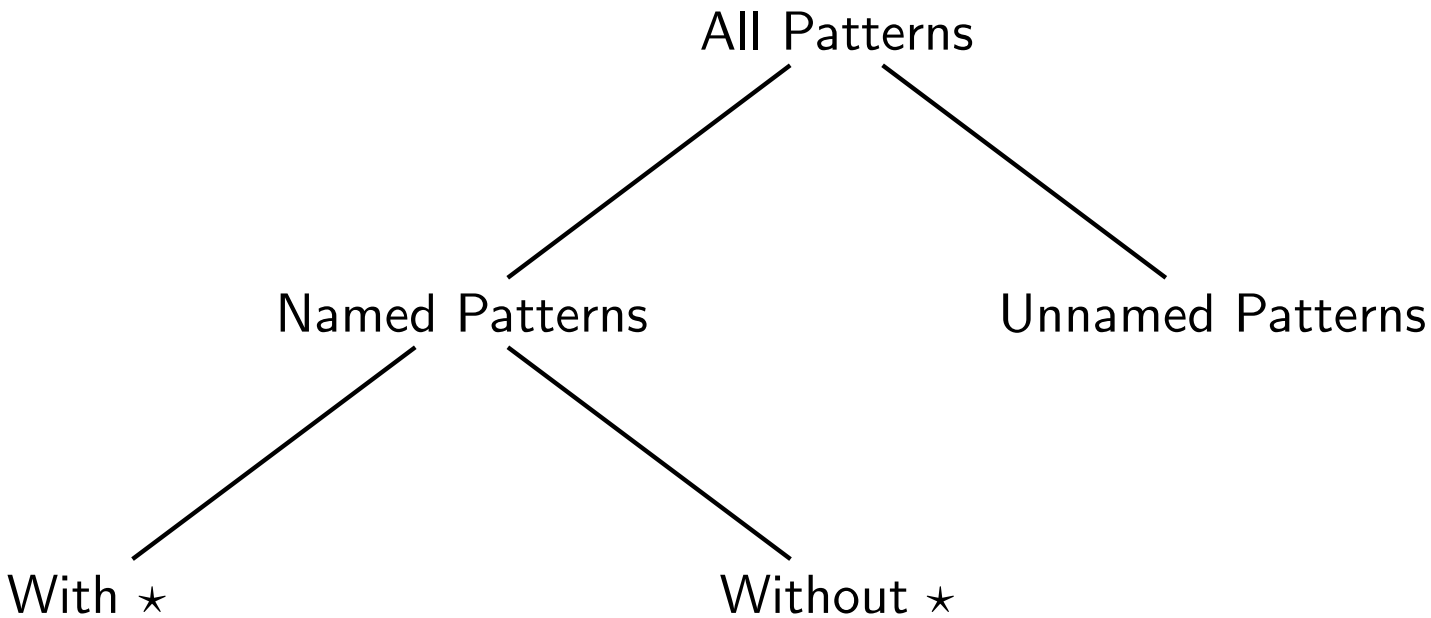
Part 5

More Features of MD

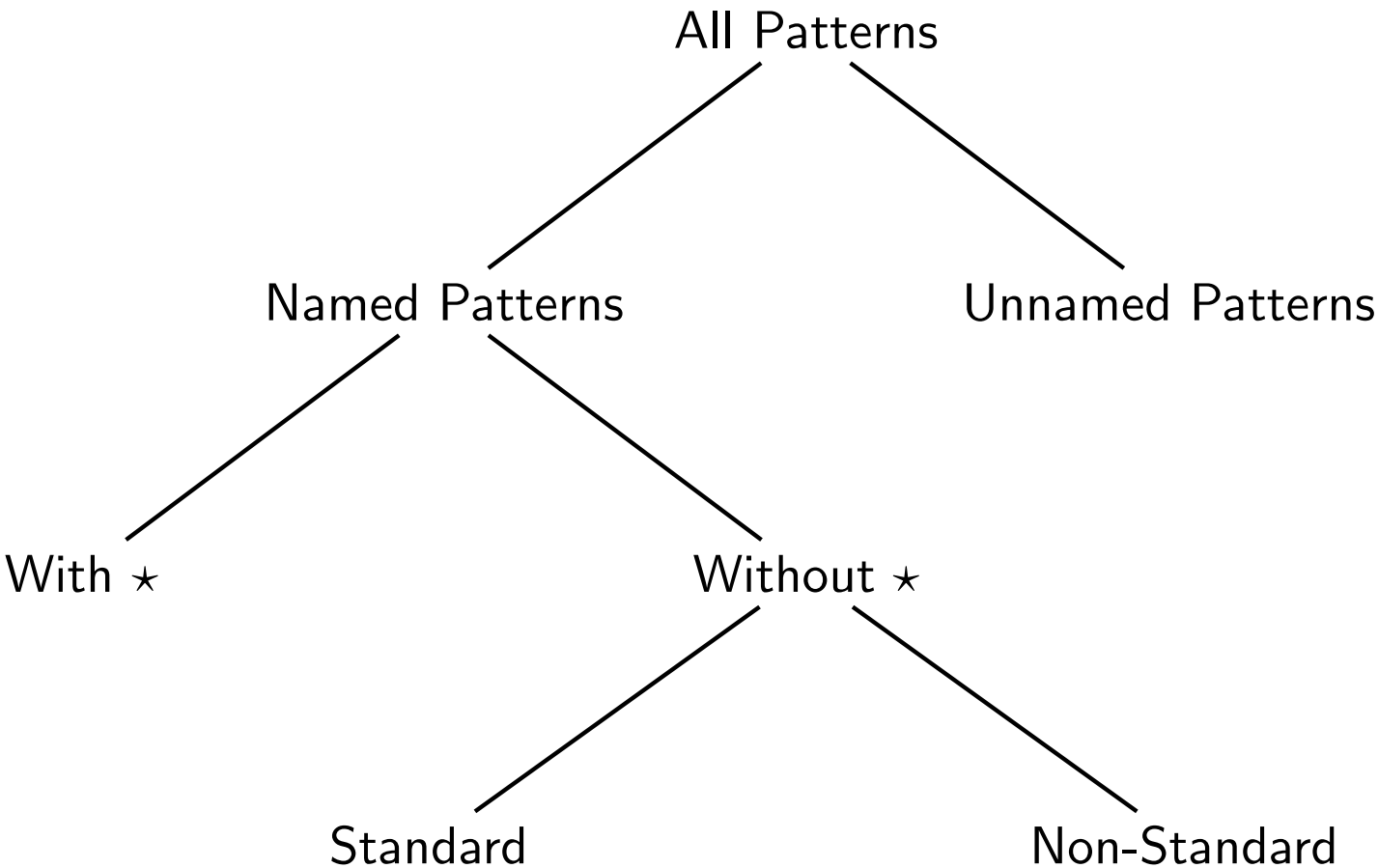
Pattern Names in `.md` File



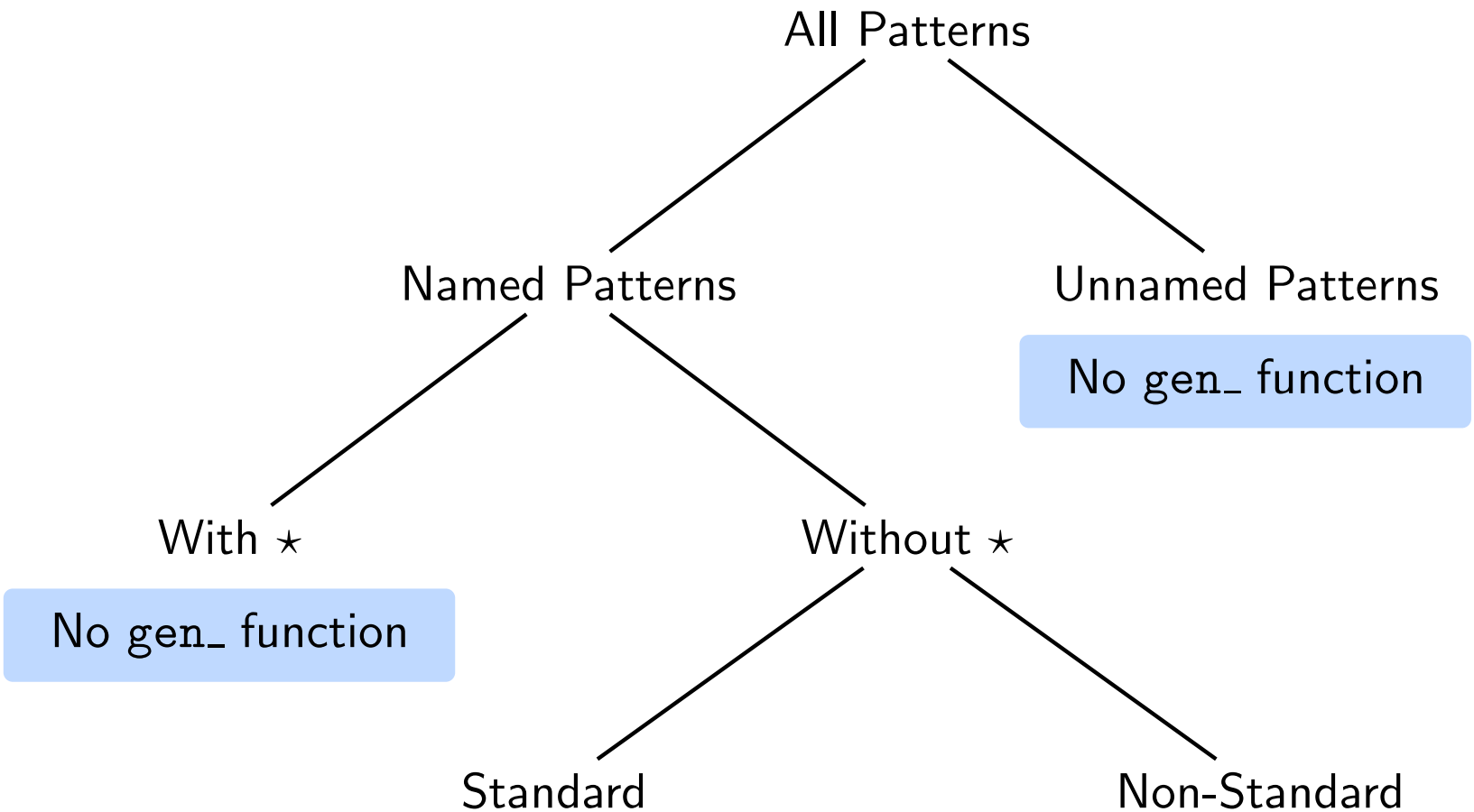
Pattern Names in `.md` File



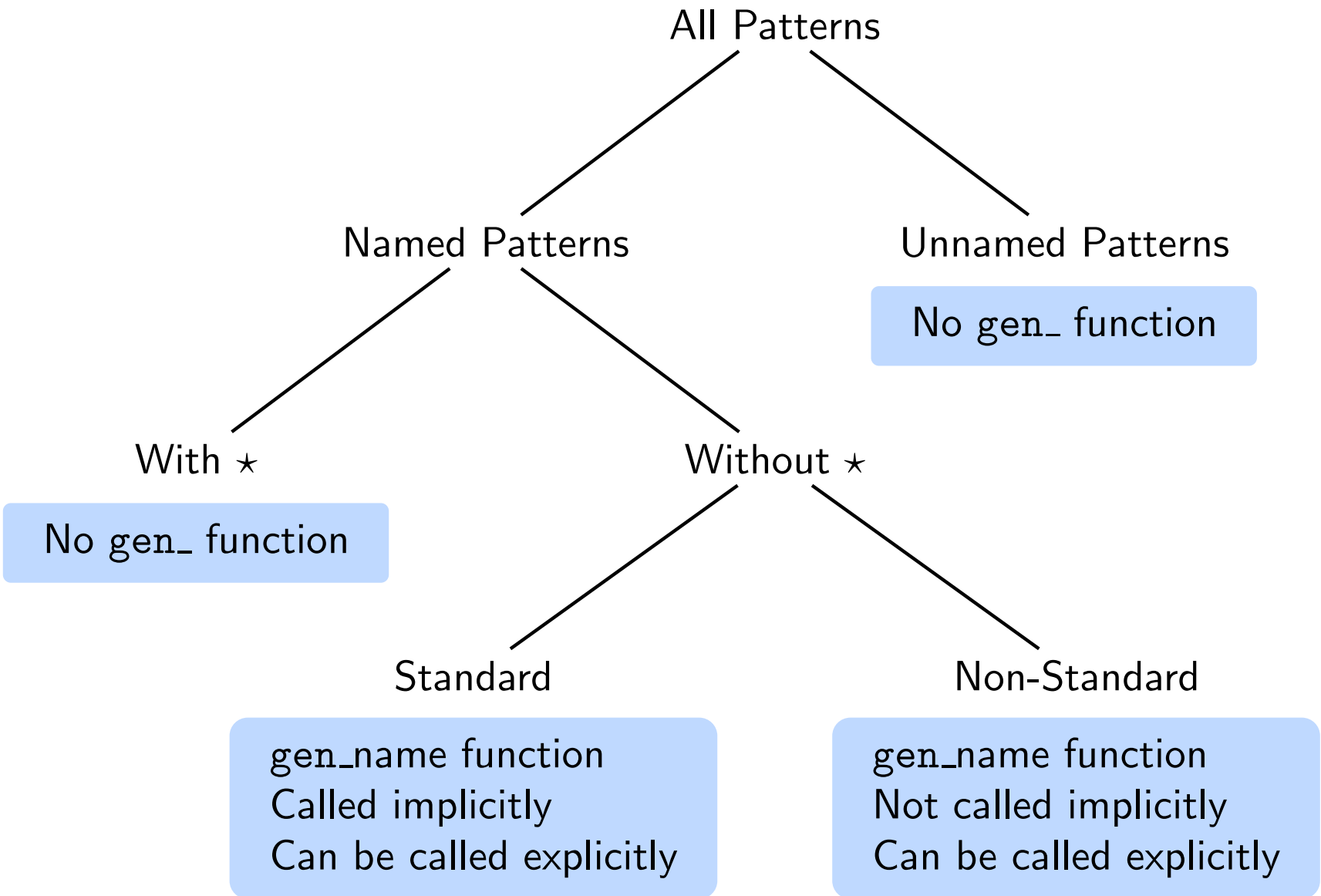
Pattern Names in `.md` File



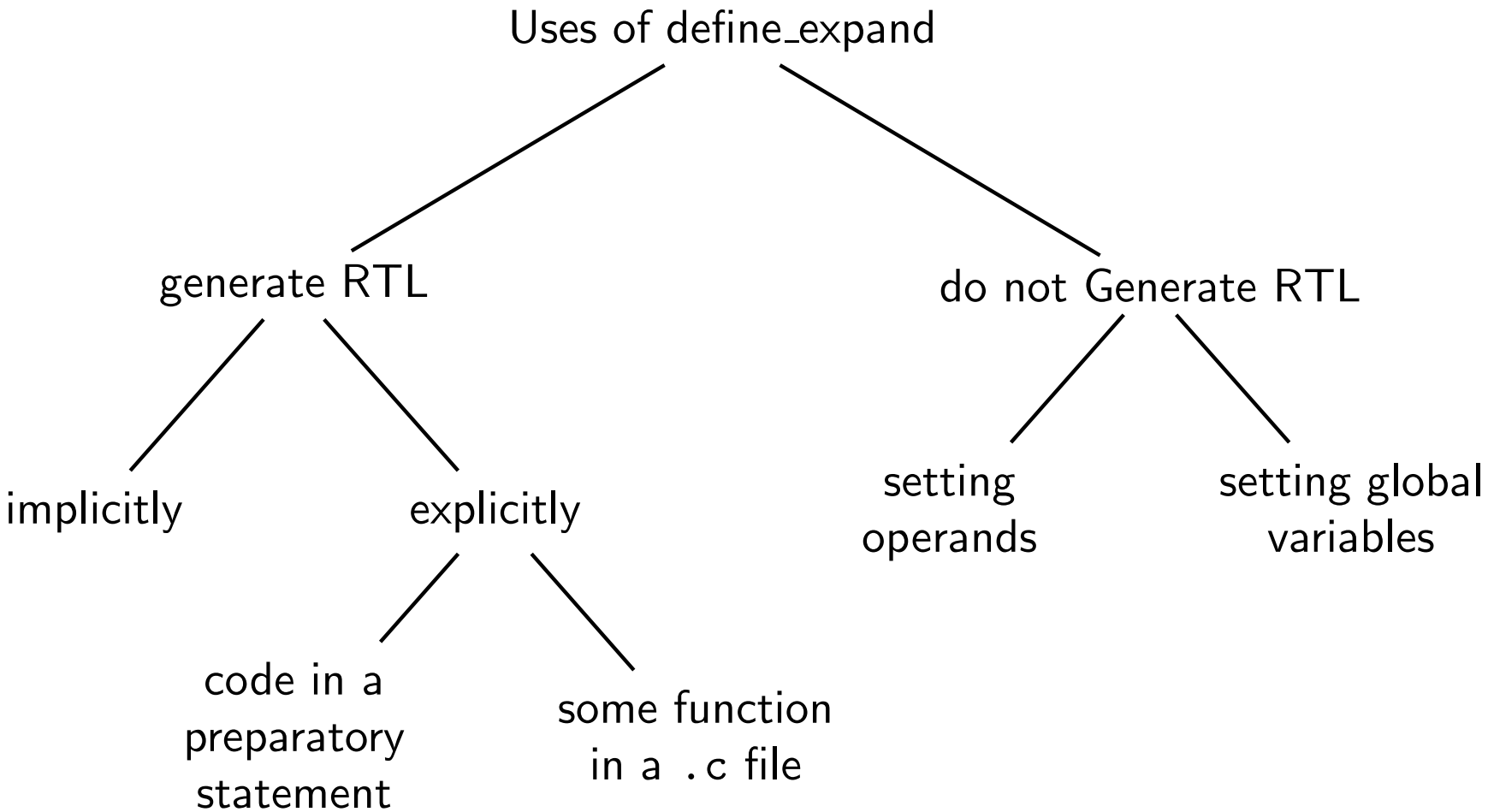
Pattern Names in `.md` File



Pattern Names in `.md` File



Role of `define_expand`



Use of Predicates

```
(define_insn "<name>"
  [(set (match_operand:SI 0 "general_operand" "=r")
        (plus:SI (match_dup 0)
            (match_operand:SI 1 "general_operand" "r")))]
  ""
  "...")
```

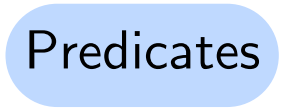
Predicates are using for matching operands

- For constructing an insn during expansion

  <name> must be a standard pattern name

- For recognizing an instruction (in subsequent RTL passes including pattern matching)

Use of Predicates




```
(define_insn "<name>"
  [(set (match_operand:SI 0 "general_operand" "=r")
        (plus:SI (match_dup 0)
            (match_operand:SI 1 "general_operand" "r")))]
  ""
  "...")
```

Predicates are using for matching operands

- For constructing an insn during expansion

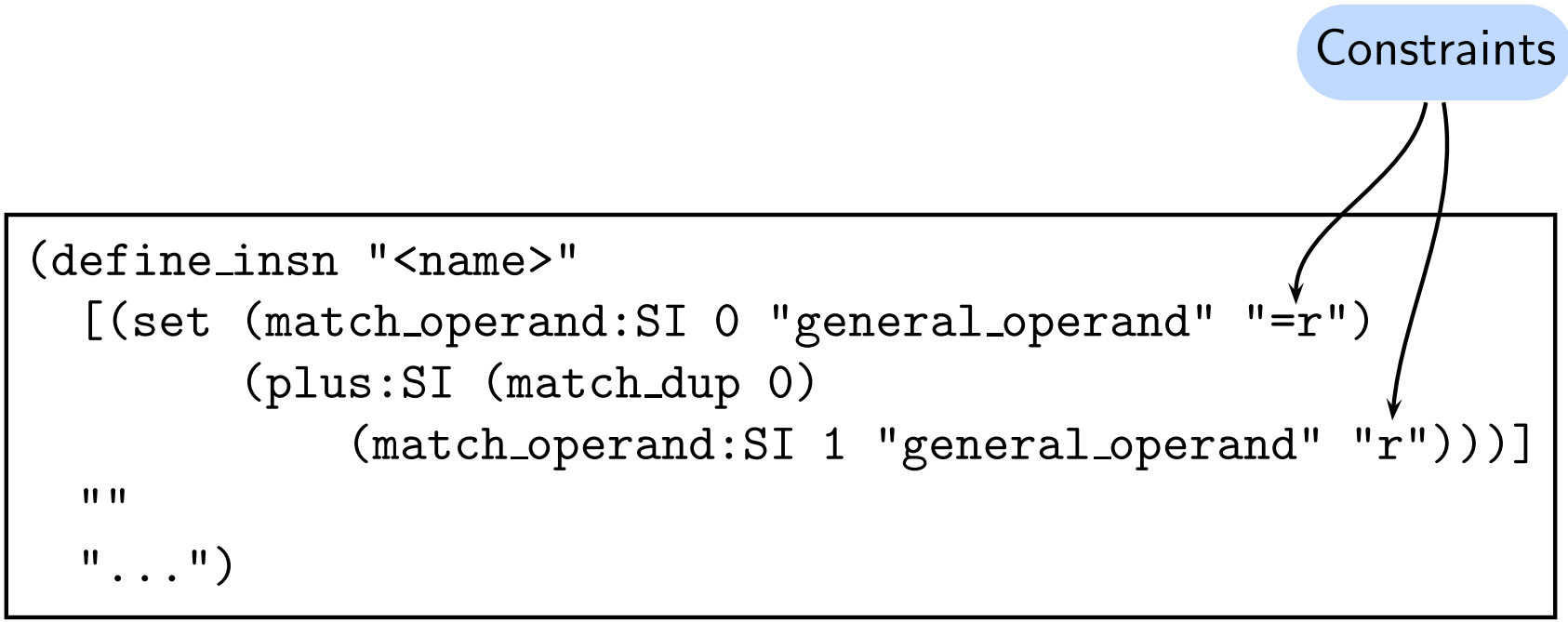
  <name> must be a standard pattern name

- For recognizing an instruction (in subsequent RTL passes including pattern matching)

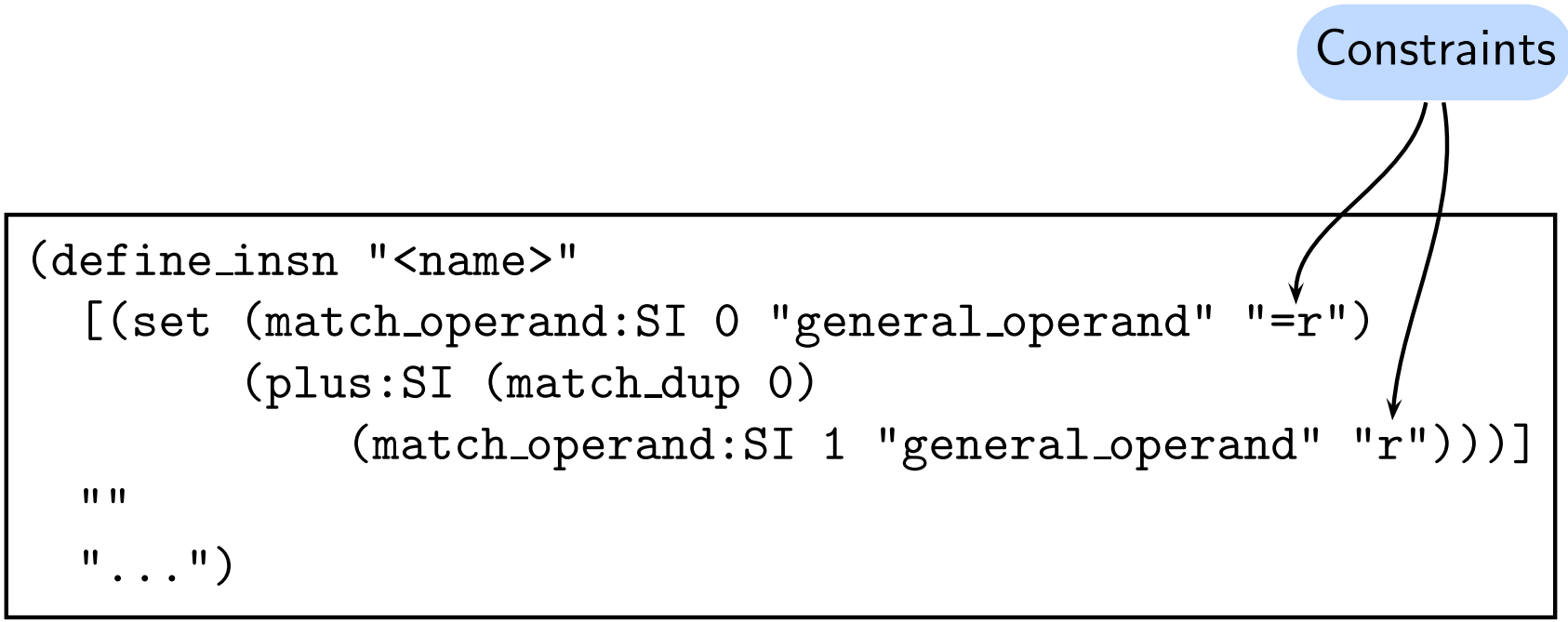
Understanding Constraints

```
(define_insn "<name>"
  [(set (match_operand:SI 0 "general_operand" "=r")
        (plus:SI (match_dup 0)
            (match_operand:SI 1 "general_operand" "r")))]
  ""
  "...")
```


Understanding Constraints

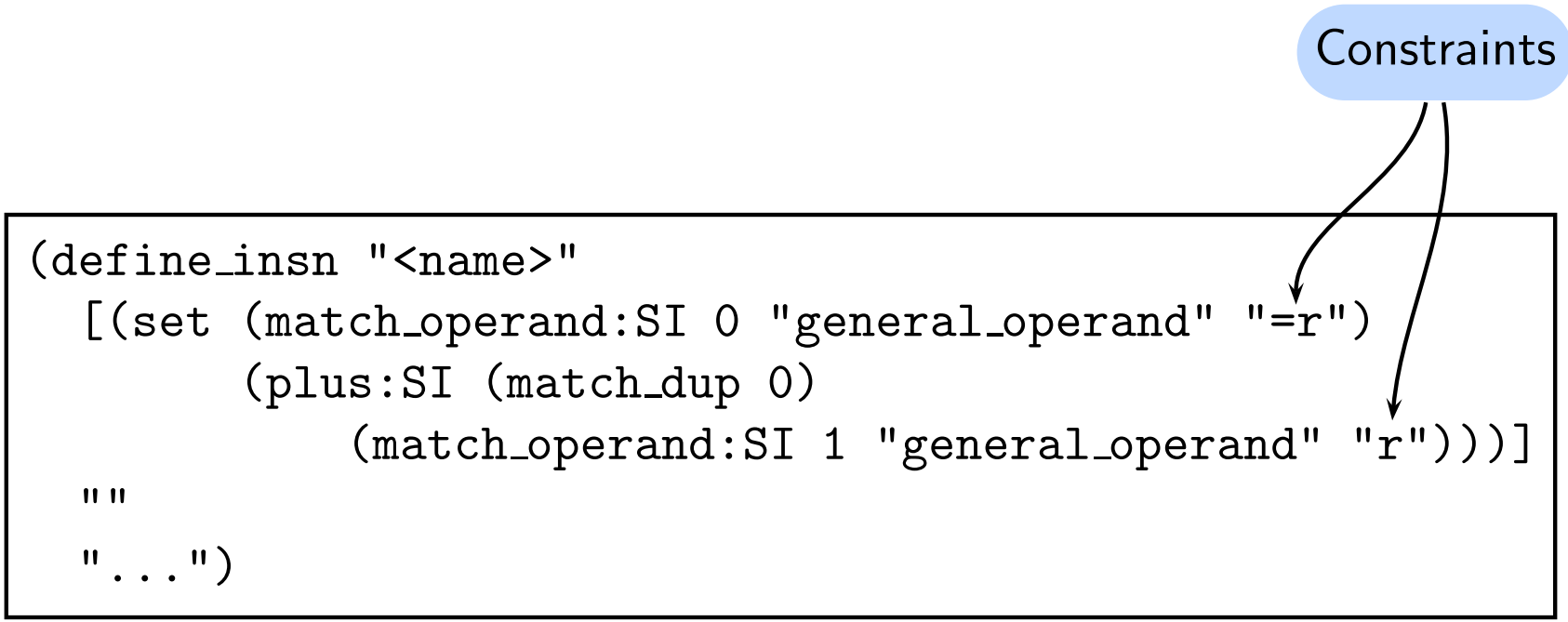


Understanding Constraints



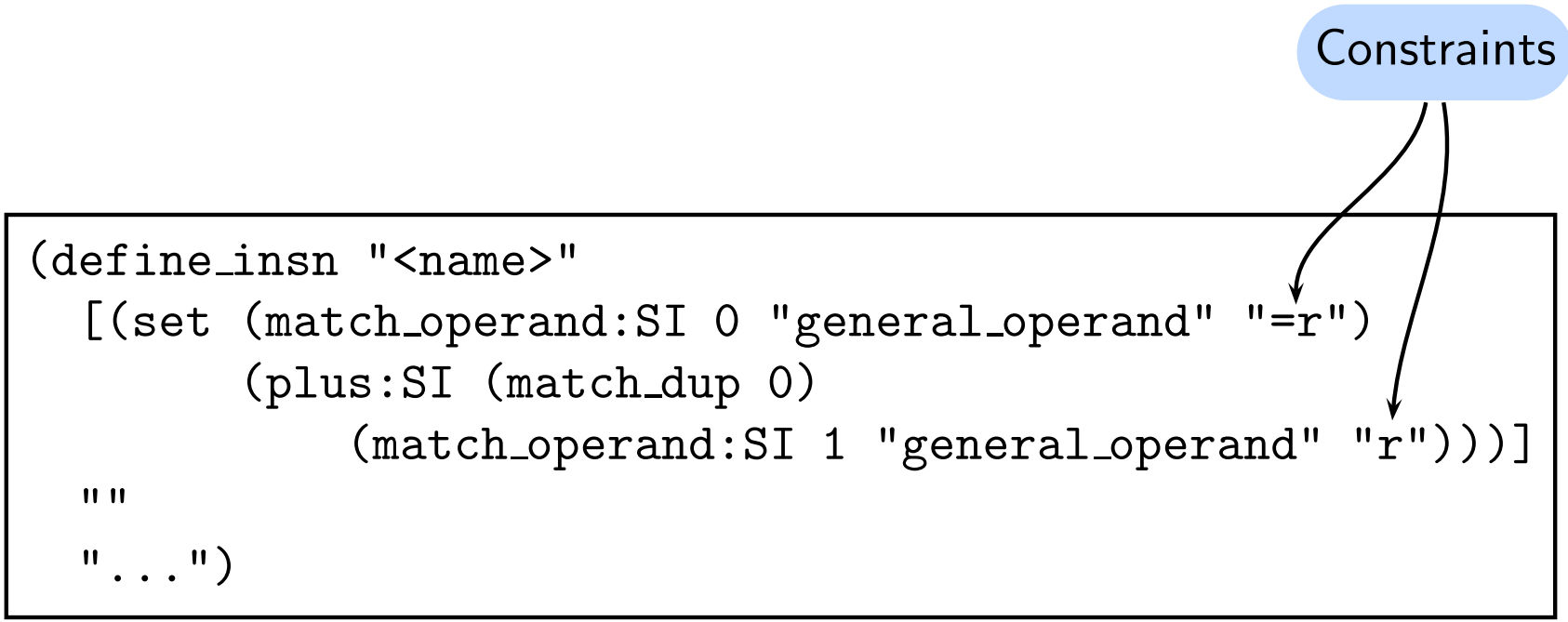
- Reloading operands in the most suitable register class

Understanding Constraints



- Reloading operands in the most suitable register class
- Fine tuning within the set of operands allowed by the predicate

Understanding Constraints



- Reloading operands in the most suitable register class
- Fine tuning within the set of operands allowed by the predicate
- If omitted, operands will depend only on the predicates

Part 6

Machine Descriptions in specRTL

The Need for Improving Machine Descriptions

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

Design Flaws in Machine Descriptions

Multiple patterns with same structure

- Repetition of almost similar RTL expressions across multiple `define_insn` an `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.

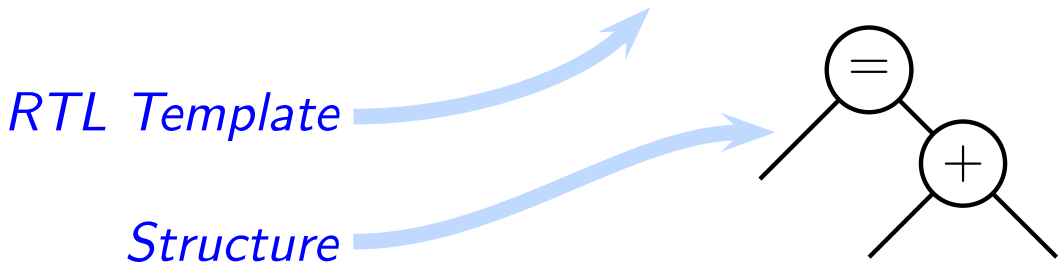
Redundancy in MIPS Machine Descriptions: Example 1

```
[(set (match_operand:m 0 "register_operand" "c0")
      (plus:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "p" "c2")))]
```

RTL Template

Redundancy in MIPS Machine Descriptions: Example 1

```
[(set (match_operand:m 0 "register_operand" "c0")
      (plus:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "p" "c2")))]
```

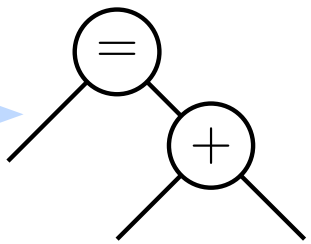


Redundancy in MIPS Machine Descriptions: Example 1

```
[(set (match_operand:m 0 "register_operand" "c0")
      (plus:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "p" "c2")))]
```

RTL Template

Structure



Details

Pattern name	*m*	*p*	*c0*	*c1*	*c2*
define_insn add<mode>3	ANYF	register_operand	=f	f	f
define_expand add<mode>3	GPR	arith_operand			
define_insn *add<mode>3	GPR	arith_operand	=d,d	d,d	d,Q

Redundancy in MIPS Machine Descriptions: Example 2

```
[(set (match_operand:m 0 "register_operand" "c0")
      (mult:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "register_operand" "c2")))]
```

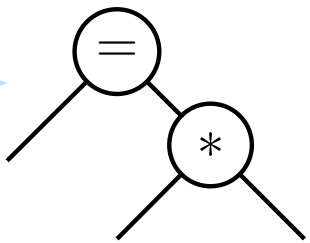
RTL Template

Redundancy in MIPS Machine Descriptions: Example 2

```
[(set (match_operand:m 0 "register_operand" "c0")
      (mult:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "register_operand" "c2")))]
```

RTL Template

Structure

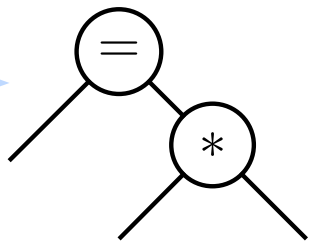


Redundancy in MIPS Machine Descriptions: Example 2

```
[(set (match_operand:m 0 "register_operand" "c0")
      (mult:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "register_operand" "c2")))]
```

RTL Template

Structure



Details

Pattern name	*m*	*c0*	*c1*	*c2*
define_insn *mul<mode>3	SCALARF	=f	f	f
define_insn *mul<mode>3_r4300	SCALARF	=f	f	f
define_insn mulv2sf3	V2SF	=f	f	f
define_expand mul<mode>3	GPR			
define_insn mul<mode>3_mul3_loongson	GPR	=d	d	d
define_insn mul<mode>3_mul3	GPR	d,1	d,d	d,d

Redundancy in MIPS Machine Descriptions: Example 3

```
[(set (match_operand:m 0 "register_operand" "c0") (plus:m
     (mult:m (match_operand:m 1 "register_operand" "c1")
             (match_operand:m 2 "register_operand" "c2")))]
     (match_operand:m 3 "register_operand" "c3")))]
```

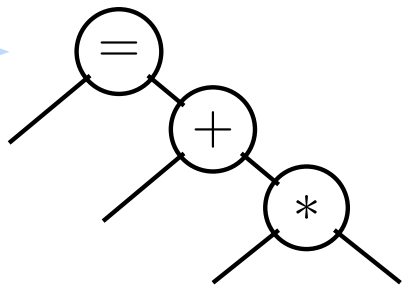
RTL Template

Redundancy in MIPS Machine Descriptions: Example 3

```
[(set (match_operand:m 0 "register_operand" "c0") (plus:m
     (mult:m (match_operand:m 1 "register_operand" "c1")
             (match_operand:m 2 "register_operand" "c2")))]
     (match_operand:m 3 "register_operand" "c3")))]
```

RTL Template

Structure

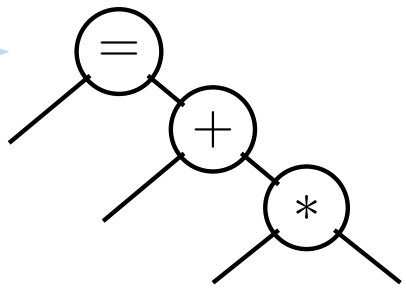


Redundancy in MIPS Machine Descriptions: Example 3

```
[(set (match_operand:m 0 "register_operand" "c0") (plus:m
     (mult:m (match_operand:m 1 "register_operand" "c1")
             (match_operand:m 2 "register_operand" "c2")))]
     (match_operand:m 3 "register_operand" "c3")))]
```

RTL Template

Structure



Details

Pattern name	*m*	*c0*	*c1*	*c2*	*c3*
*mul_acc_si	SI	=l*?*?,d?	d,d	d,d	0,d
*mul_acc_si_r3900	SI	=l*?*?,d*?,d?	d,d,d	d,d,d	0,1,d
*macc	SI	=l,d	d,d	d,d	0,1
*madd4<mode>	ANYF	=f	f	f	f
*madd3<mode>	ANYF	=f	f	f	0

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 can not be used

specRTL: Key Observations

- 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
- Associtiating 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 ≡ tree structure of RTL templates
  - Details ≡ attributes of tree nodes
    (eg. modes, predicates, constraints etc.)

Meeting the Specification Goals: Key Idea

- Separation of shapes from target specific details:
  - Shape ≡ tree structure of RTL templates
  - Details ≡ 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 ≡ tree structure of RTL templates
  - Details ≡ 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: `extends`

Meeting the Specification Goals: Operations

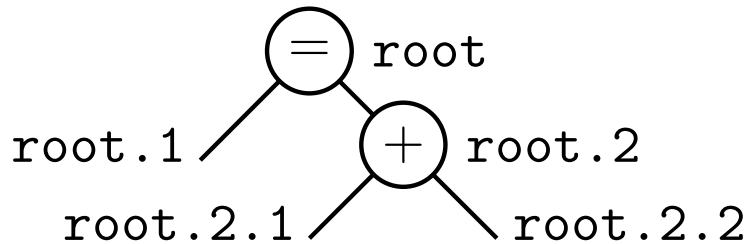
- Creating new shapes by composing shapes: `extends`
- Associtiating concrete details with shapes: `instantiates`

Meeting the Specification Goals: Operations

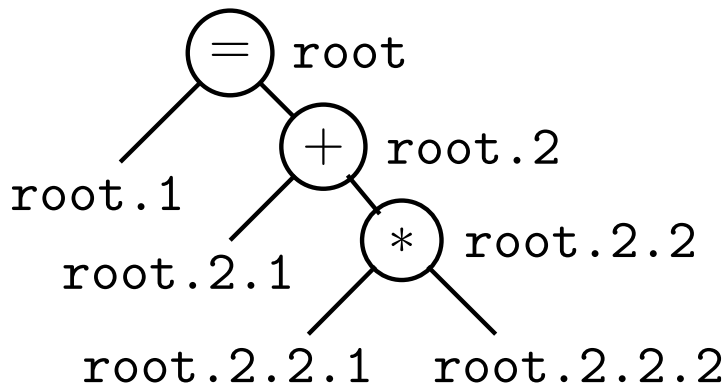
- Creating new shapes by composing shapes: `extends`
- Associtiating concrete details with shapes: `instantiates`
- Overriding concrete details: `overrides`

Creating Abstract Patterns

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

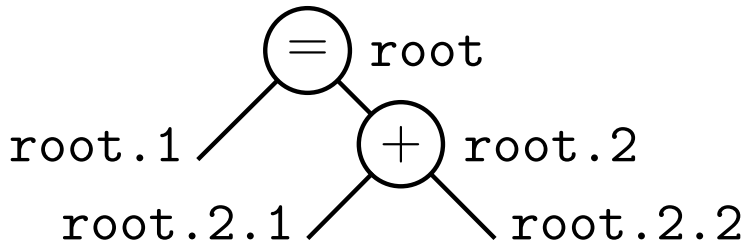



```
abstract set_macc extends
   set_plus
{
  root.2.2 = mult;
}
```



Creating Concrete Patterns

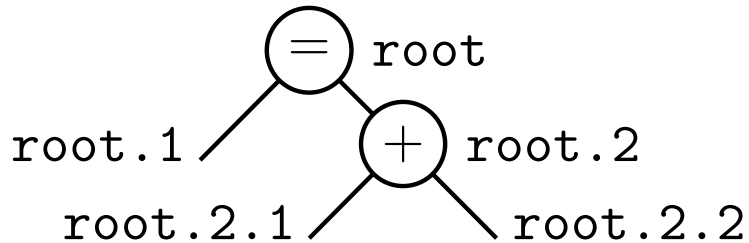
```
abstract set_plus extends set
{
  root.2 = plus;
}
```




```
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;
}
concrete 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

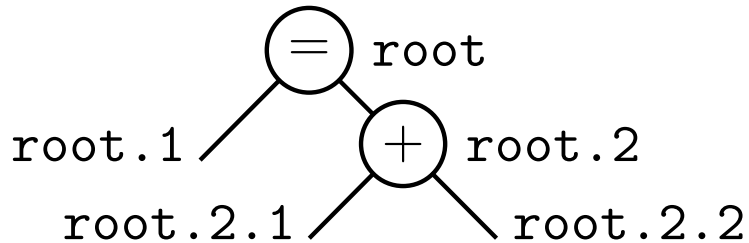
```
abstract set_plus extends set
{
  root.2 = plus;
}
```




```
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;
}
{: /* Conventional Machine Description Fragments */ :}
```


Generating Conventional Machine Descriptions

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



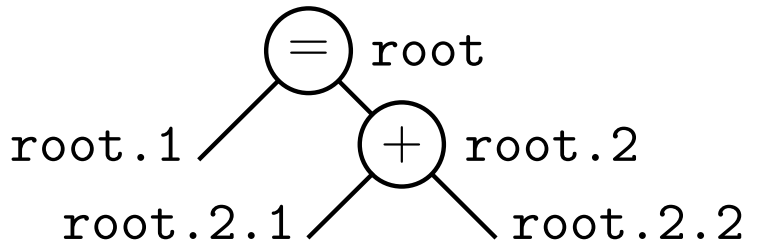

```
concrete add<mode>3.insn instantiates set_plus
{ set_plus(register_operand:ANYF:"=f", register_operand:ANYF:"f",
           register_operand:ANYF:"f");
  root.2.mode = ANYF;
}
{: /* Conventional Machine Description Fragments */ :}
```

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 */
)
```


Overriding Details

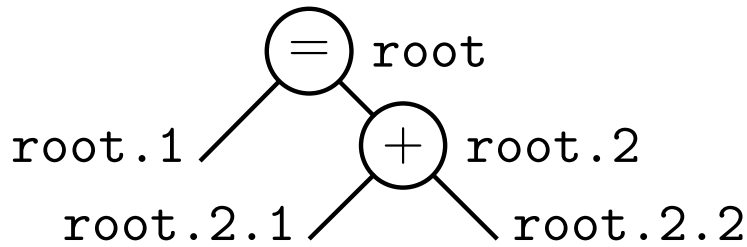
```
abstract set_plus extends set
{
  root.2 = plus;
}
```




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


Overriding Details

```
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"); }
```


Part 7

Conclusions

GCC MD Summary

- GCC achieves retargetability by reading the machine descriptions and generating a back end customised to the machine descriptions
- Machine descriptions are influenced by:
  The HLLs, GCC architecture, and properties of target, host and build systems
- Writing machine descriptions requires:
  specifying the C macros, target instructions and any required support functions
- `define_insn` and `define_expand` are used to convert a GIMPLE representation to RTL

specRTL: Current Status and Plans for Future Work

- specRTL compiler version 2 is ready
- i386 and MIPS machine descriptions have been re-written

specRTL: 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