An Overview of Compilation

Uday Khedker

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

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

January 2014

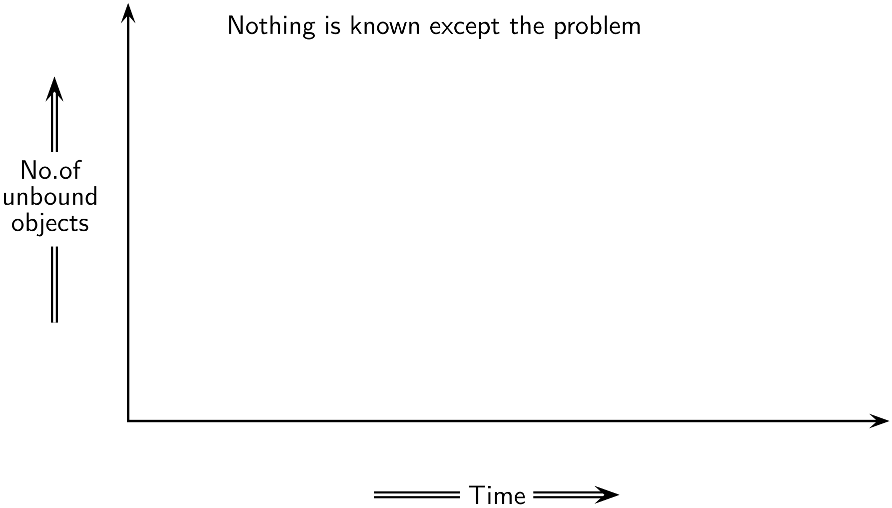
Outline

- Introduction
- compilation sequence
- compilation models

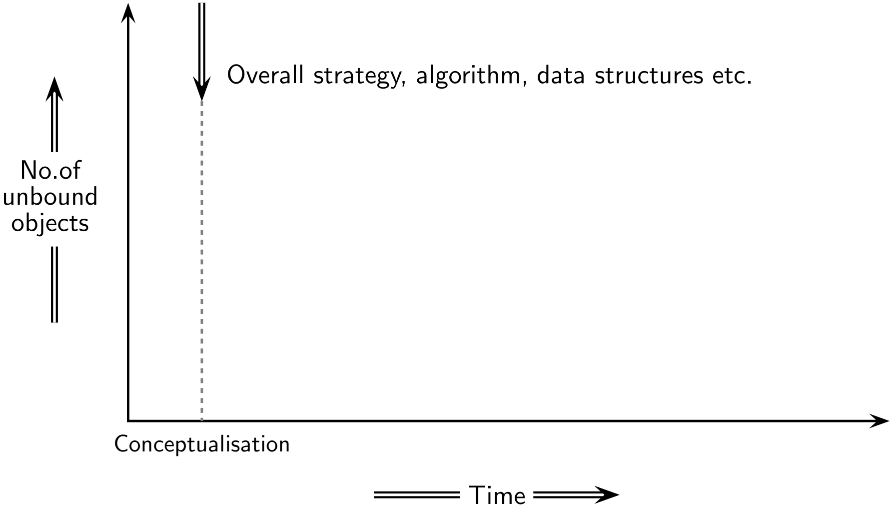
Part 1

Introduction to Compilation

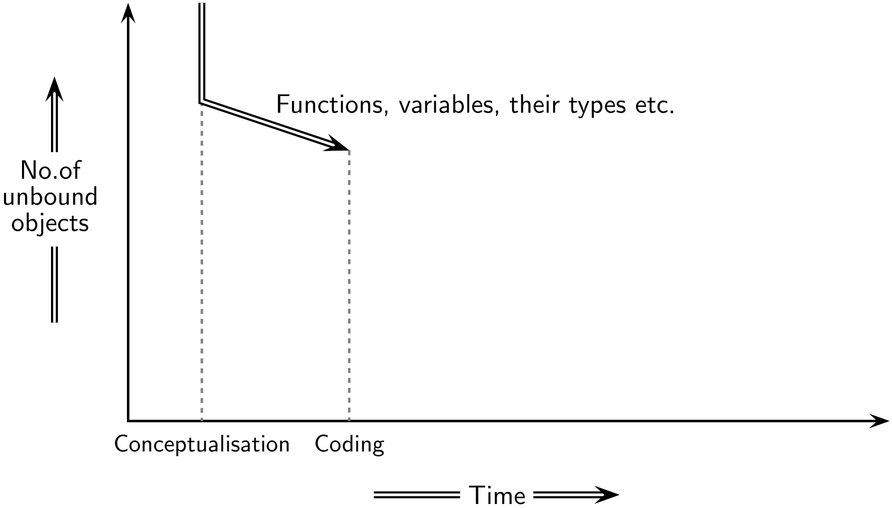
Binding


Nothing is known except the problem
No.of unbound objects
Time

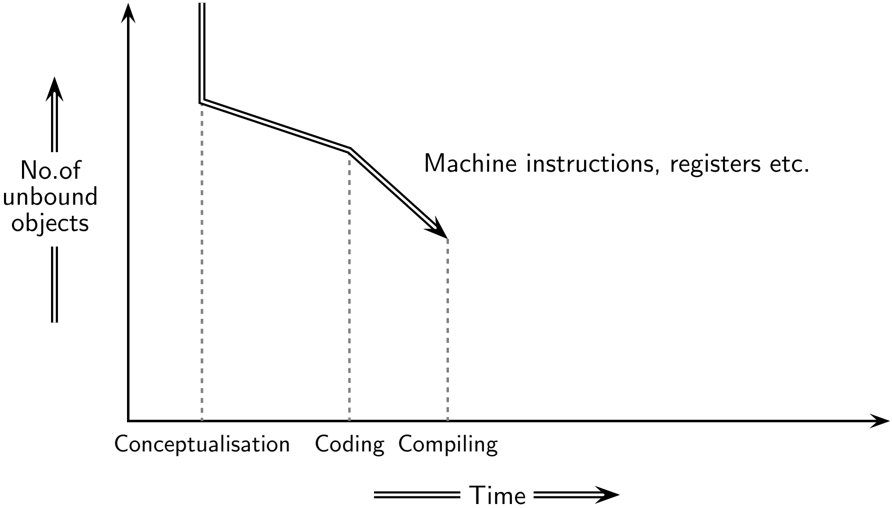
Binding


Overall strategy, algorithm, data structures etc.
No.of unbound objects
Conceptualisation
Time

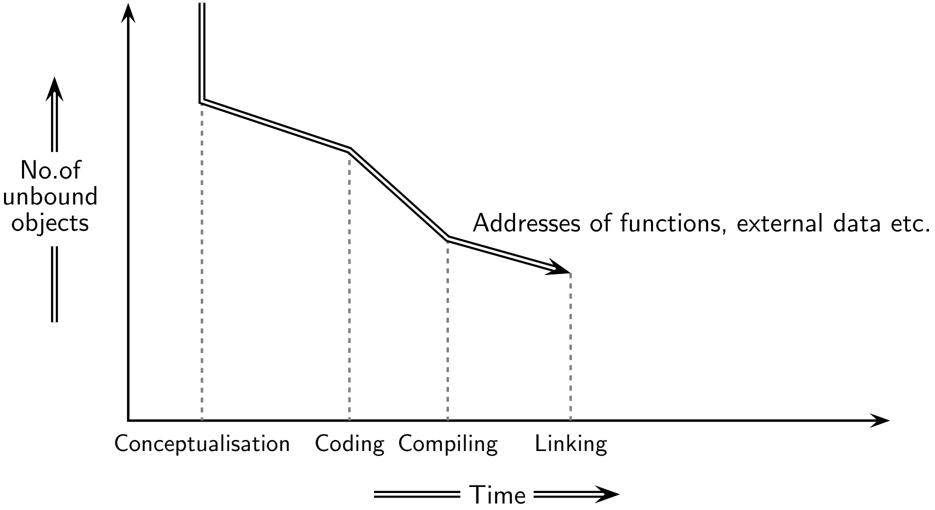
Binding


Functions, variables, their types etc.
No.of unbound objects
Conceptualisation
Coding
Time

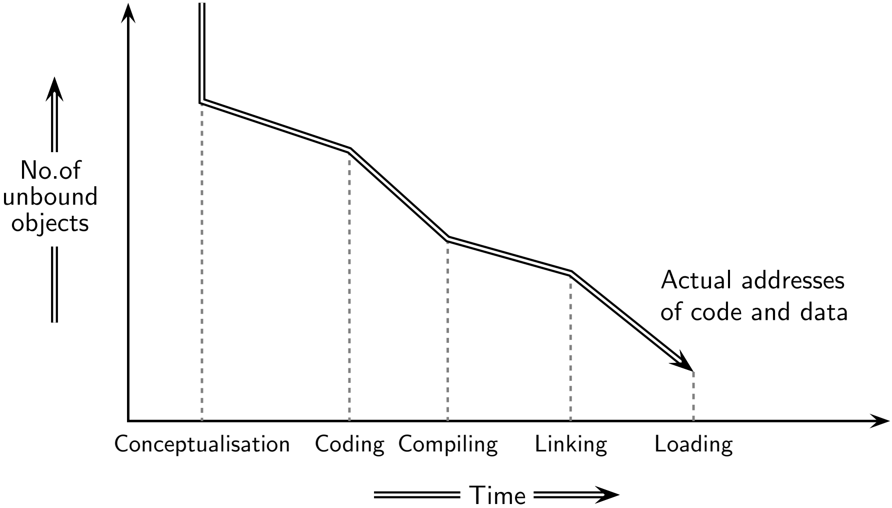
Binding


No.of
unbound
objects
Machine instructions, registers etc.
Conceptualisation
Coding
Compiling
Time

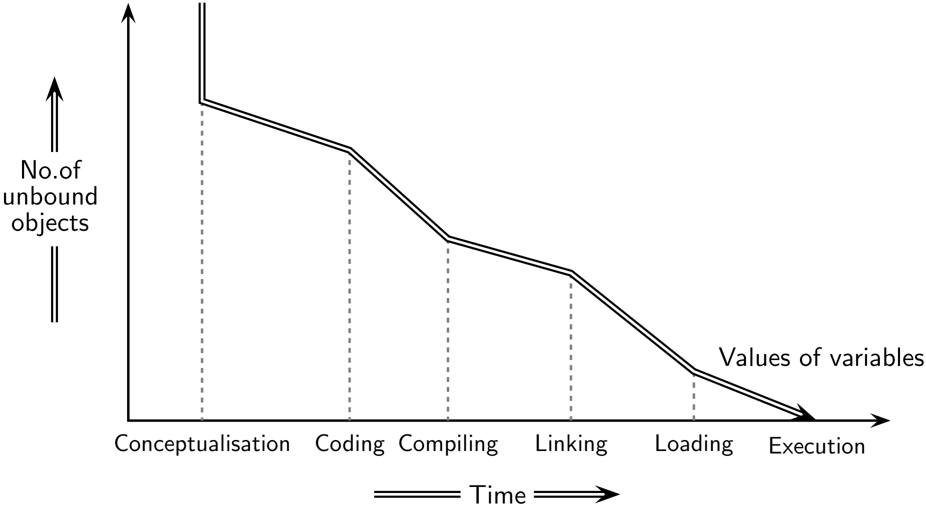
Binding


No.of unbound objects
Addresses of functions, external data etc.
Conceptualisation
Coding
Compiling
Linking
Time

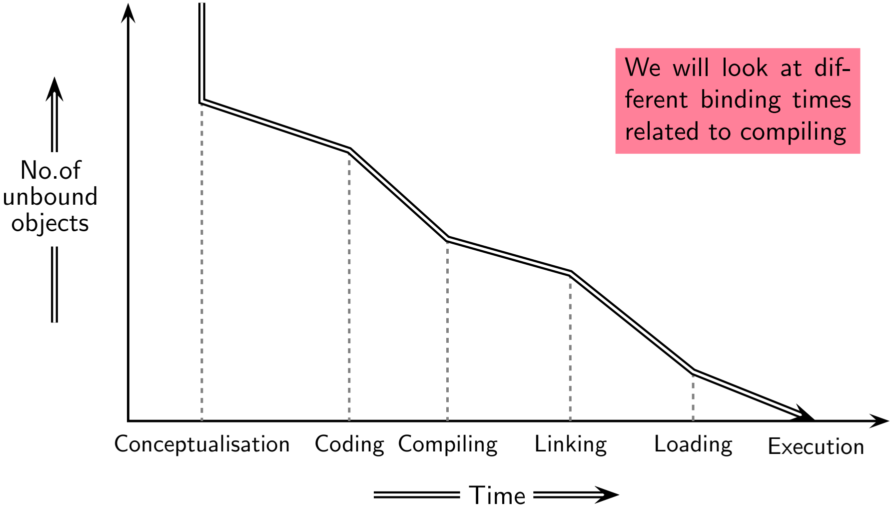
Binding


No.of
unbound
objects
Actual addresses
of code and data
Conceptualisation
Coding
Compiling
Linking
Loading
Time

Binding


No.of unbound objects
Values of variables
Conceptualisation
Coding
Compiling
Linking
Loading
Execution
Time

Binding

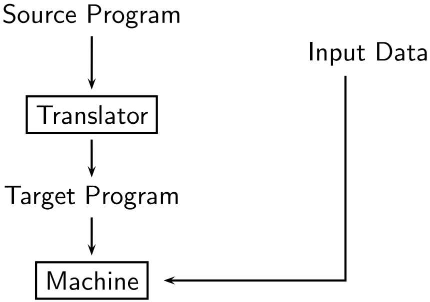


We will look at different binding times related to compiling

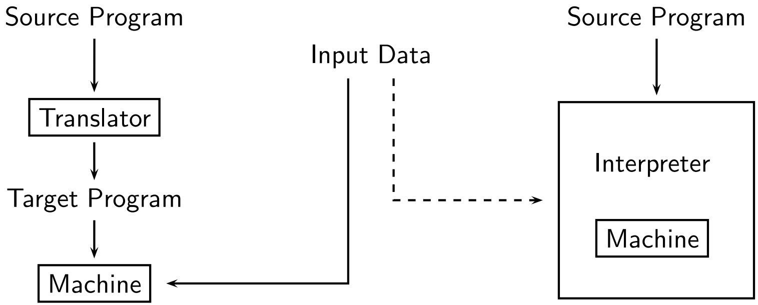
Implementation Mechanisms

$$
\begin{array}{c}
\text{Source Program} \\
\downarrow \\
\boxed{\text{Translator}} \\
\downarrow \\
\text{Target Program} \\
\downarrow \\
\boxed{\text{Machine}}
\end{array}
$$

Implementation Mechanisms

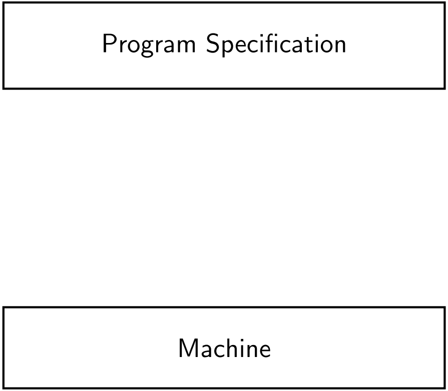


Implementation Mechanisms



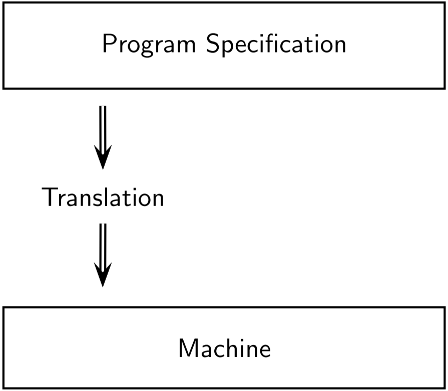
Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution



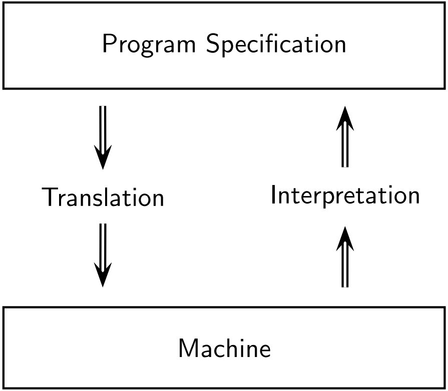
Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution



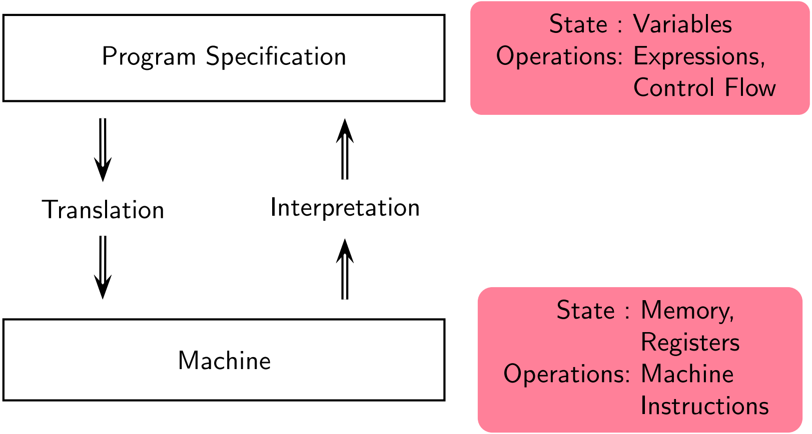
Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution



Implementation Mechanisms as “Bridges”

- “Gap” between the “levels” of program specification and execution



High and Low Level Abstractions

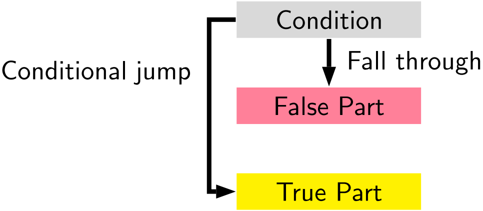
Input C statement

```
a = b<10?b:c;
```

Spim Assembly Equivalent

```
    lw   $t0, 4($fp)   ;     t0 <- b          # Is b smaller
    slti $t0, $t0, 10  ;     t0 <- t0 < 10    # than 10?
    not  $t0, $t0      ;     t0 <- !t0
    bgtz $t0, L0:      ;     if t0>0 goto L0
    lw   $t0, 4($fp)   ;     t0 <- b          # YES
    b    L1:           ;     goto L1
L0: lw   $t0, 8($fp)   ;L0: t0 <- c          # NO
L1: sw   0($fp), $t0   ;L1: a <- t0
```

High and Low Level Abstractions



Input C statement

```
a = b<10?b:c;
```

Spim Assembly Equivalent

```
     lw   $t0, 4($fp)   ;     t0 <- b          # Is b smaller
     slti $t0, $t0, 10  ;     t0 <- t0 < 10    # than 10?
     not  $t0, $t0      ;     t0 <- !t0
     bgtz $t0, L0:      ;     if t0>0 goto L0
     lw   $t0, 4($fp)   ;     t0 <- b          # YES
     b    L1:           ;     goto L1
L0:  lw   $t0, 8($fp)   ;L0:  t0 <- c          # NO
L1:  sw   0($fp), $t0   ;L1:  a <- t0
```

High and Low Level Abstractions

NOT Condition

True Part

False Part

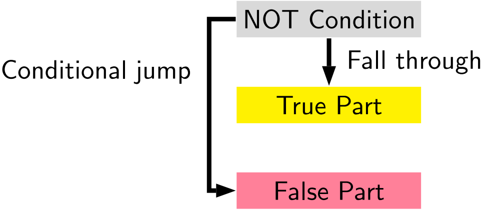
Input C statement

```
a = b<10?b:c;
```

Spim Assembly Equivalent

```
     lw   $t0, 4($fp)   ;     t0 <- b          # Is b smaller
     slti $t0, $t0, 10  ;     t0 <- t0 < 10    # than 10?
     not  $t0, $t0      ;     t0 <- !t0
     bgtz $t0, L0:      ;     if t0>0 goto L0
     lw   $t0, 4($fp)   ;     t0 <- b          # YES
     b    L1:           ;     goto L1
L0:  lw   $t0, 8($fp)   ;L0:  t0 <- c          # NO
L1:  sw   0($fp), $t0   ;L1:  a <- t0
```

High and Low Level Abstractions



Input C statement

```
a = b<10?b:c;
```

Spim Assembly Equivalent

```
     lw   $t0, 4($fp)   ;     t0 <- b          # Is b smaller
     slti $t0, $t0, 10  ;     t0 <- t0 < 10    # than 10?
     not  $t0, $t0      ;     t0 <- !t0
     bgtz $t0, L0:      ;     if t0>0 goto L0
     lw   $t0, 4($fp)   ;     t0 <- b          # YES
     b    L1:           ;     goto L1
L0:  lw   $t0, 8($fp)   ;L0:  t0 <- c          # NO
L1:  sw   0($fp), $t0   ;L1:  a <- t0
```

Implementation Mechanisms

- Translation $=$ Analysis $+$ Synthesis

  Interpretation $=$ Analysis $+$ Execution

Implementation Mechanisms

- $$\begin{aligned}\text{Translation} &= \text{Analysis} + \text{Synthesis}\\ \text{Interpretation} &= \text{Analysis} + \text{Execution}\end{aligned}$$

- Translation Instructions $\Longrightarrow$ Equivalent Instructions

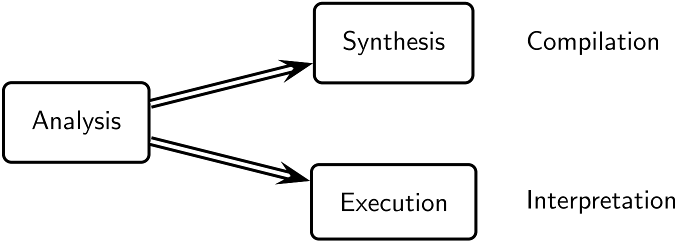
Implementation Mechanisms

- Translation $=$ Analysis $+$ Synthesis
  Interpretation $=$ Analysis $+$ Execution

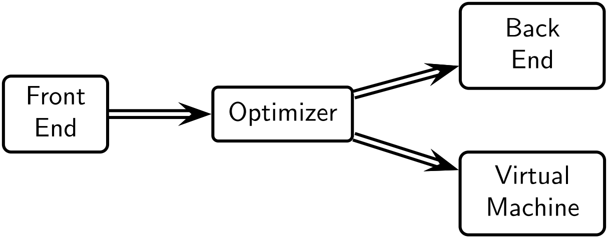
- Translation Instructions $\Longrightarrow$ Equivalent Instructions

  Interpretation Instructions $\Longrightarrow$ Actions Implied by Instructions

Language Implementation Models



Language Processor Models



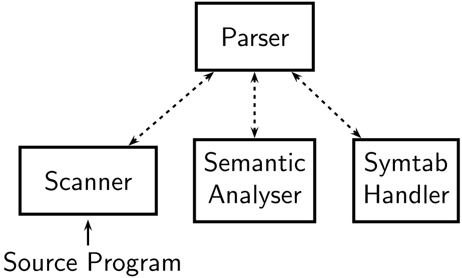
C,C++

Java, C#

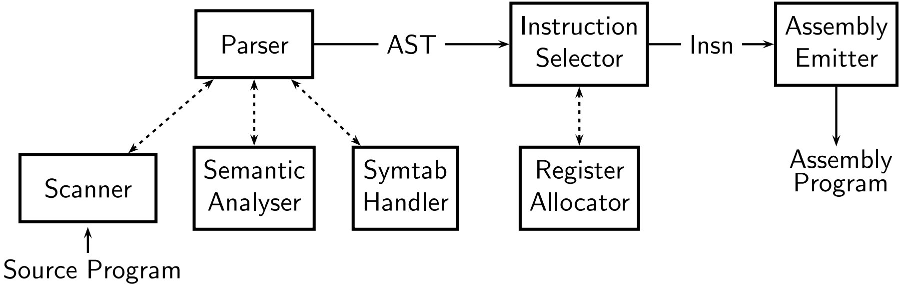
Part 2

An Overview of Compilation Phases

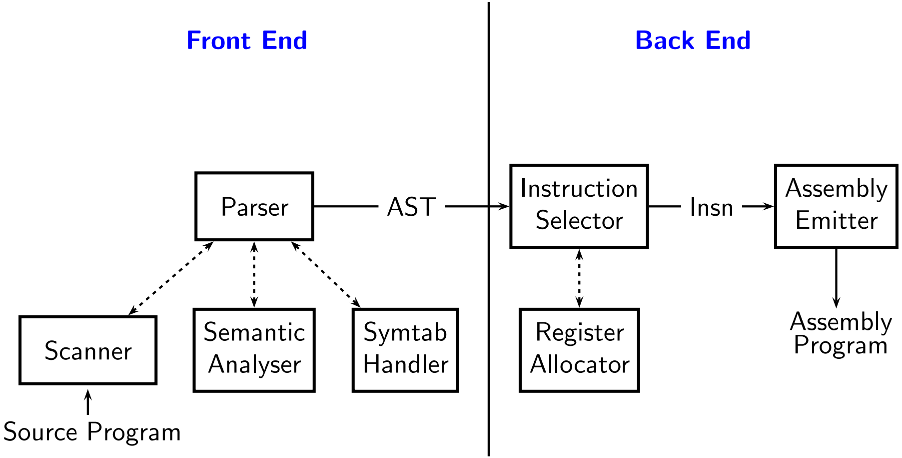
The Structure of a Simple Compiler



The Structure of a Simple Compiler



The Structure of a Simple Compiler

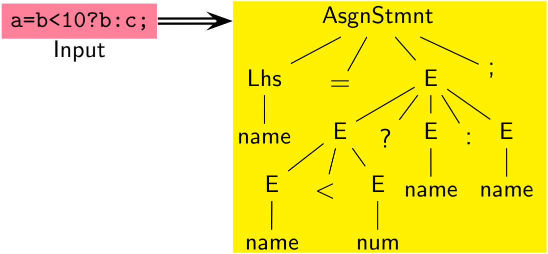


Translation Sequence in Our Compiler: Parsing

`a=b<10?b:c;`

Input

Translation Sequence in Our Compiler: Parsing

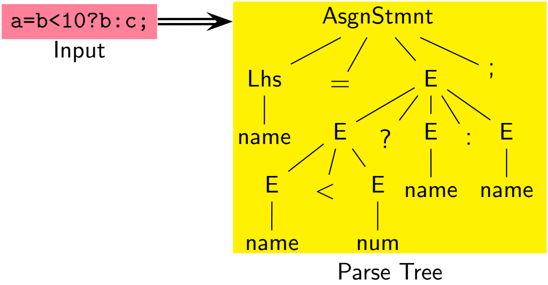


Parse Tree

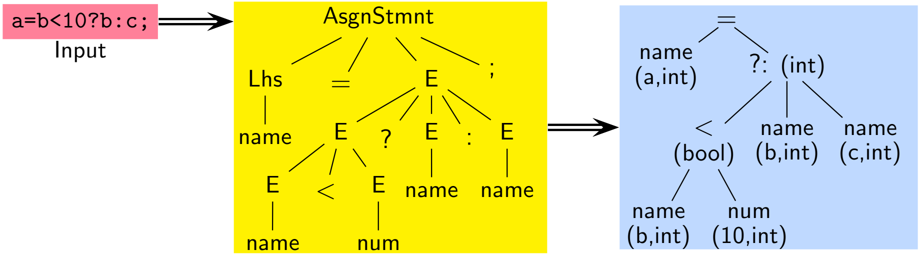
Issues:

- Grammar rules, terminals, non-terminals
- Order of application of grammar rules
  eg. is it `(a = b<10?)` followed by `(b:c)`?
- Values of terminal symbols
  eg. string "10" vs. integer number 10.

Translation Sequence in Our Compiler: Semantic Analysis



Translation Sequence in Our Compiler: Semantic Analysis



Parse Tree

Abstract Syntax Tree
(with attributes)

Issues:

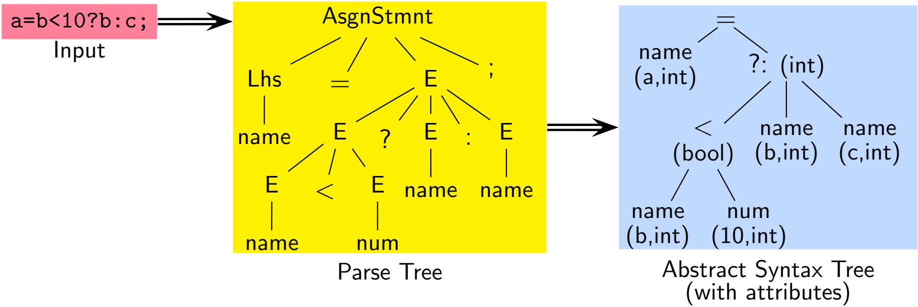
- Symbol tables

  Have variables been declared? What are their types? What is their scope?

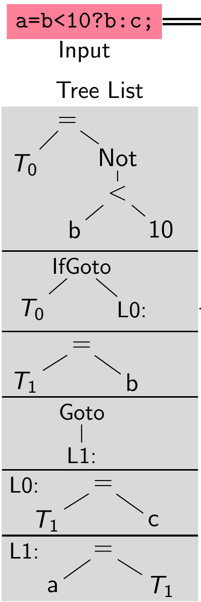
- Type consistency of operators and operands

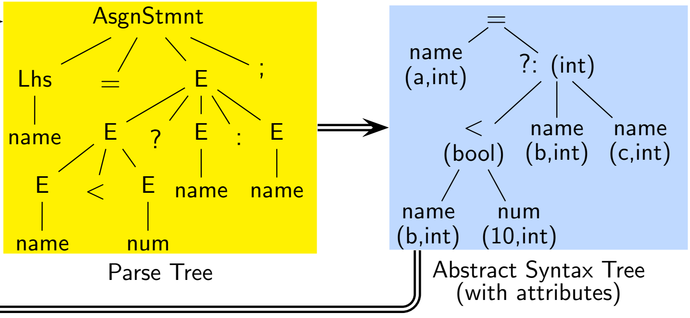
  The result of computing `b<10?` is bool and not int

Translation Sequence in Our Compiler: IR Generation



Translation Sequence in Our Compiler: IR Generation





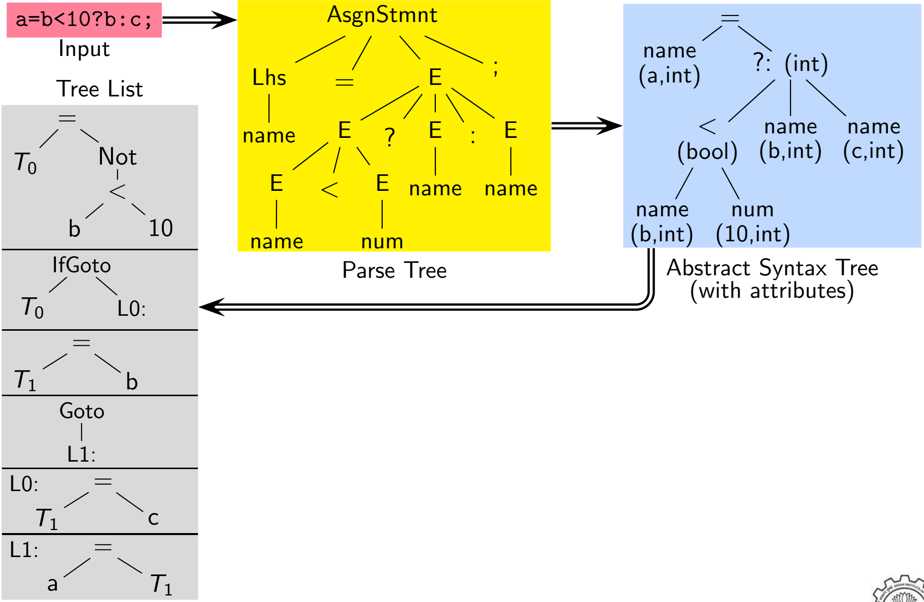
Issues:

- Convert to maximal trees which can be implemented without altering control flow

  Simplifies instruction selection and scheduling, register allocation etc.

- Linearise control flow by flattening nested control constructs

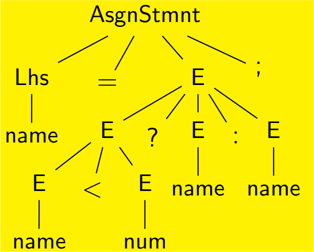
Translation Sequence in Our Compiler: Instruction Selection



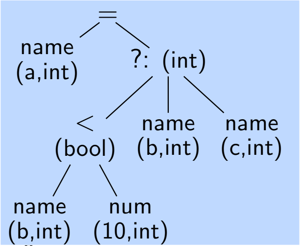
Translation Sequence in Our Compiler: Instruction Selection

`a=b<10?b:c;`

Input

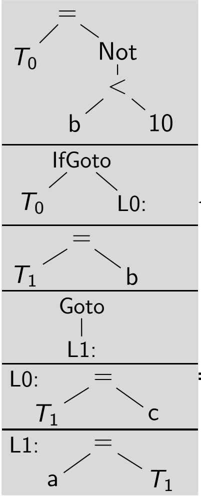


Parse Tree



Abstract Syntax Tree
(with attributes)

Tree List



Instruction List

$T_0 \leftarrow$ b
$T_0 \leftarrow T_0 < 10$
$T_0 \leftarrow\, !\, T_0$
if $T_0 > 0$ goto L0:
$T_1 \leftarrow$ b
goto L1:
L0: $T_1 \leftarrow$ c
L1: a $\leftarrow T_1$

Issues:

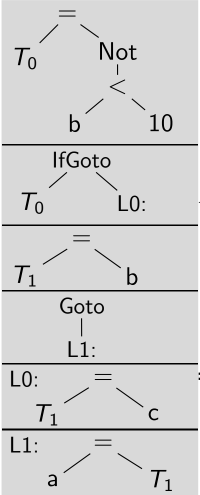
- Cover trees with as few machine instructions as possible
- Use temporaries and local registers

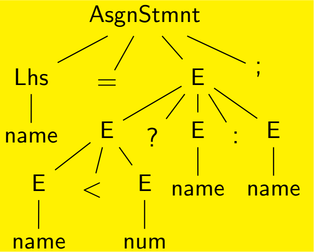
Translation Sequence in Our Compiler: Instruction Selection

`a=b<10?b:c;`

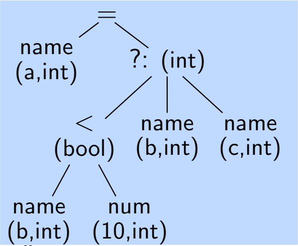
Input

Tree List





Parse Tree



Abstract Syntax Tree
(with attributes)

Instruction List

$T_0 \leftarrow$ b
$T_0 \leftarrow T_0 < 10$
$T_0 \leftarrow ! \ T_0$
if $T_0 > 0$ goto L0:
$T_1 \leftarrow$ b
goto L1:
L0: $T_1 \leftarrow$ c
L1: a $\leftarrow T_1$

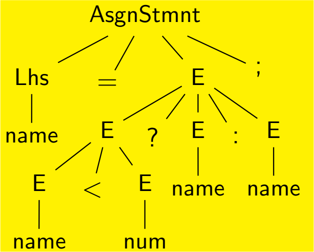
Issues:

- Cover trees with as few machine instructions as possible
- Use temporaries and local registers

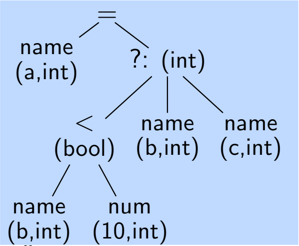
Translation Sequence in Our Compiler: Emitting Instructions

`a=b<10?b:c;`

Input

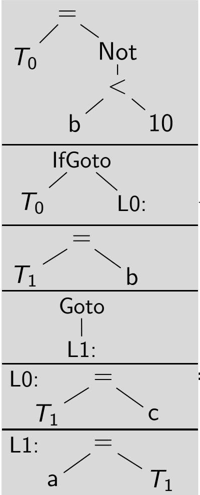


Parse Tree



Abstract Syntax Tree
(with attributes)

Tree List



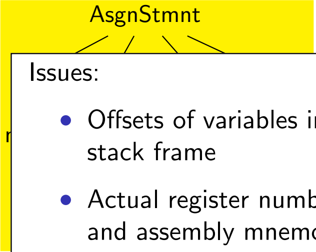
Instruction List

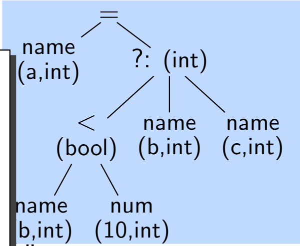
$T_0 \leftarrow$ b
$T_0 \leftarrow T_0 < 10$
$T_0 \leftarrow\, ! \, T_0$
if $T_0 > 0$ goto L0:
$T_1 \leftarrow$ b
goto L1:
L0: $T_1 \leftarrow$ c
L1: a $\leftarrow T_1$

Translation Sequence in Our Compiler: Emitting Instructions

`a=b<10?b:c;`

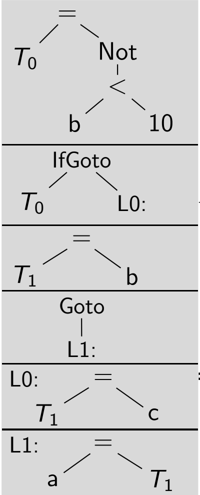
Input





Abstract Syntax Tree (with attributes)

Tree List



Issues:

- Offsets of variables in the stack frame
- Actual register numbers and assembly mnemonics
- Code to construct and discard activation records

Instruction List

	$T_0 \leftarrow$ b
	$T_0 \leftarrow T_0 < 10$
	$T_0 \leftarrow !\ T_0$
	if $T_0 > 0$ goto L0:
	$T_1 \leftarrow$ b
	goto L1:
L0:	$T_1 \leftarrow$ c
L1:	a $\leftarrow T_1$

Assembly Code

```
     lw   $t0, 4($fp)
     slti $t0, $t0, 10
     not  $t0, $t0
     bgtz $t0, L0:
     lw   $t0, 4($fp)
     b    L1:
L0:  lw   $t0, 8($fp)
L1:  sw   0($fp), $t0
```

Part 3

Compilation Models

Compilation Models

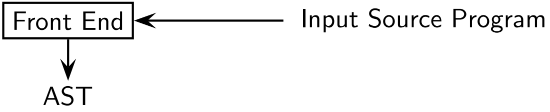
Aho Ullman Model

Davidson Fraser Model

Compilation Models

Aho Ullman Model

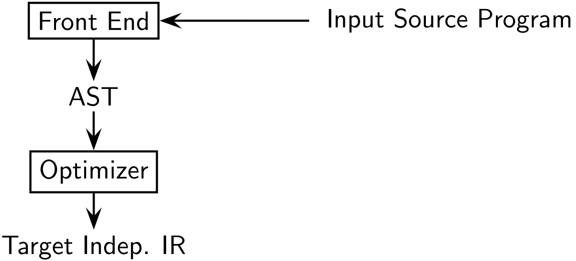
Davidson Fraser Model



Compilation Models

Aho Ullman Model

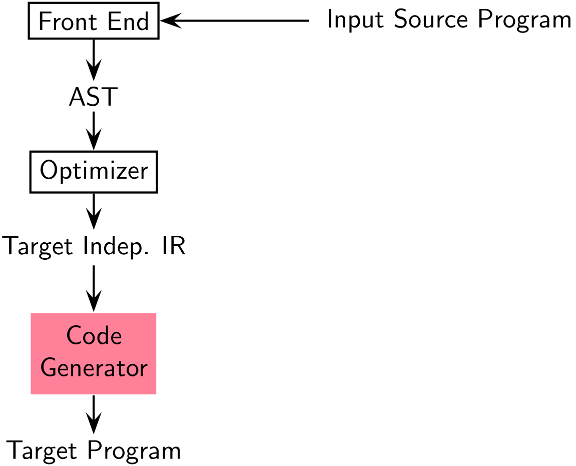
Davidson Fraser Model



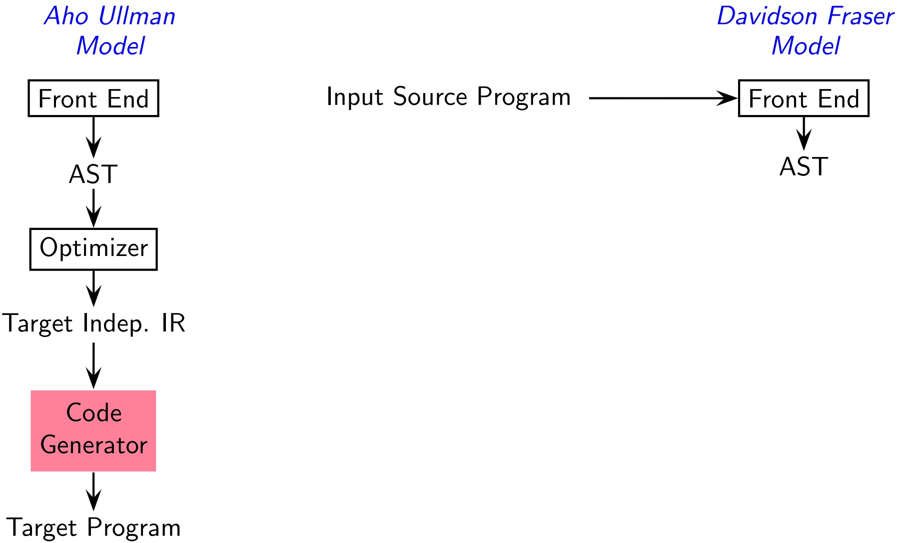
Compilation Models

Aho Ullman Model

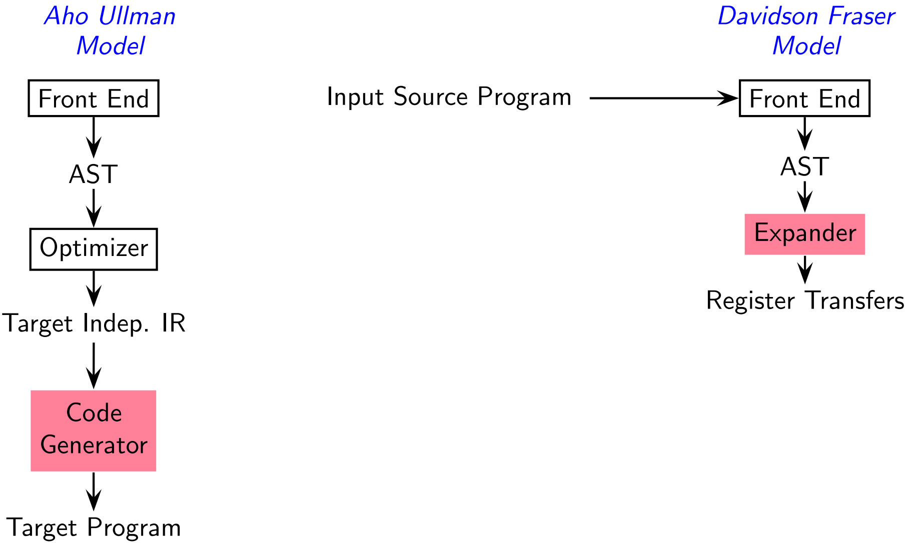
Davidson Fraser Model



Compilation Models

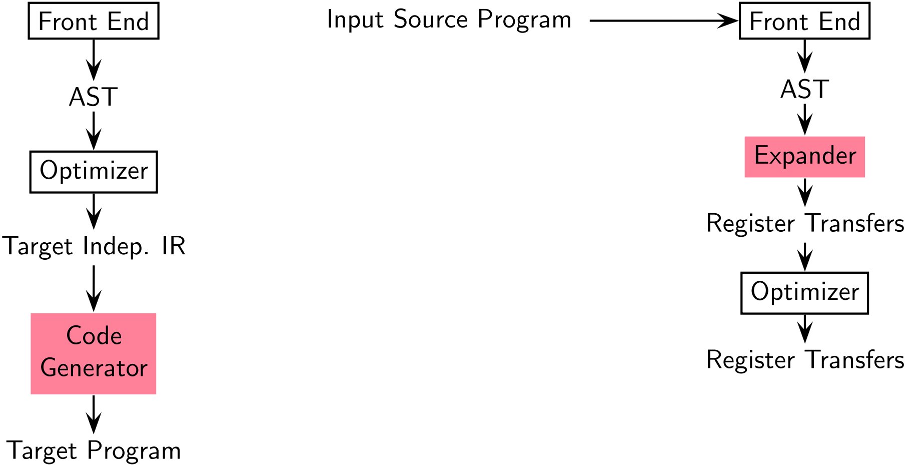


Compilation Models



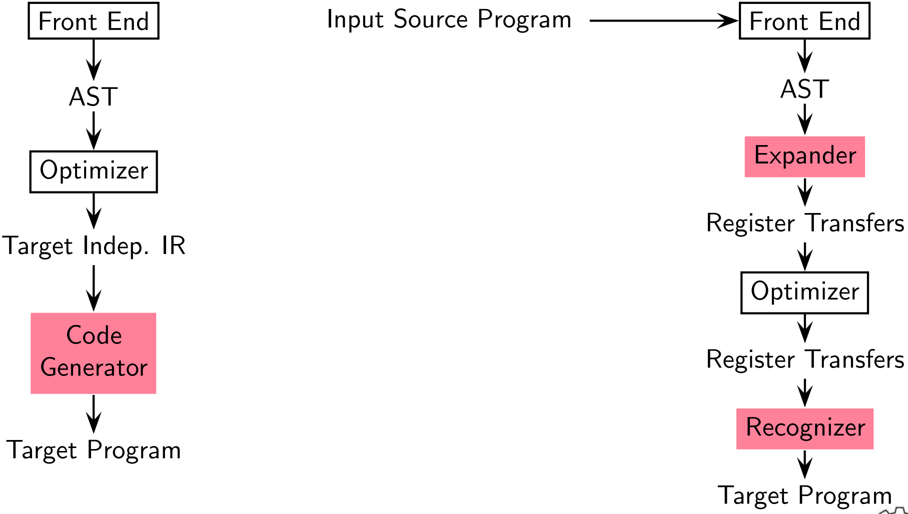
Compilation Models

Aho Ullman Model *Davidson Fraser Model*



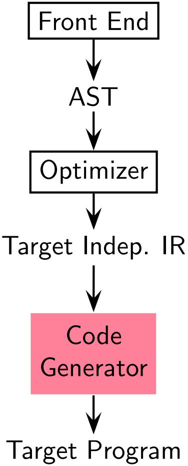
Compilation Models

Aho Ullman Model *Davidson Fraser Model*



Compilation Models

Aho Ullman Model



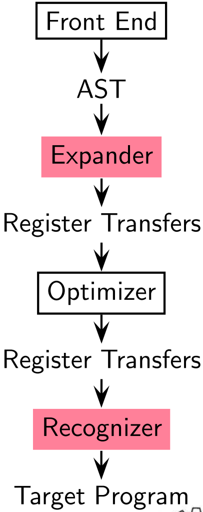
Aho Ullman: Instruction selection

- over optimized IR using
- cost based tree tiling matching

Davidson Fraser: Instruction selection

- over AST using
- simple full tree matching based algorithms that generate
- naive code which is
  - target dependent, and is
  - optimized subsequently

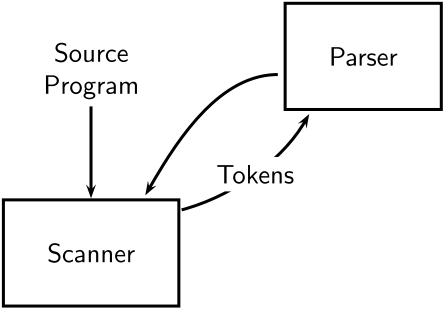
Davidson Fraser Model



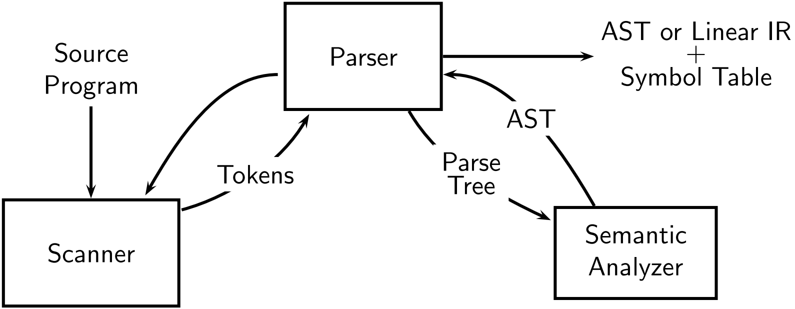
Typical Front Ends

Parser

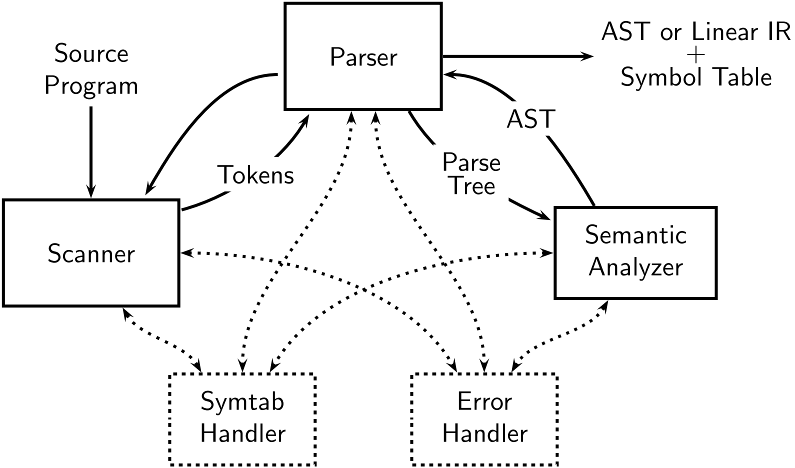
Typical Front Ends



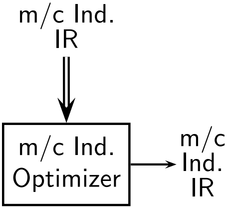
Typical Front Ends



Typical Front Ends

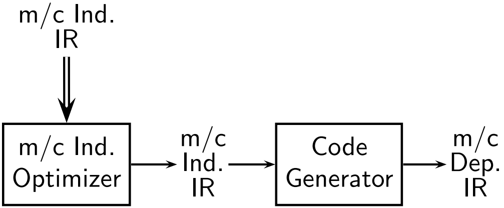


Typical Back Ends in Aho Ullman Model



- Compile time evaluations
- Eliminating redundant computations

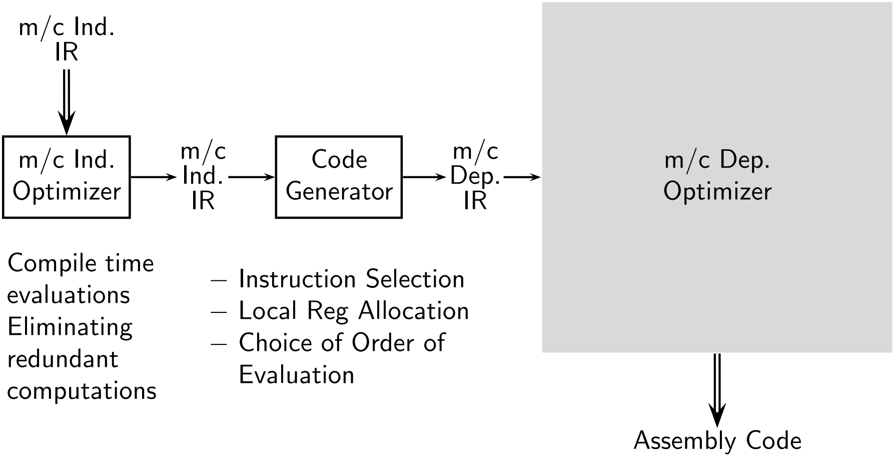
Typical Back Ends in Aho Ullman Model



– Compile time evaluations
– Eliminating redundant computations

– Instruction Selection
– Local Reg Allocation
– Choice of Order of Evaluation

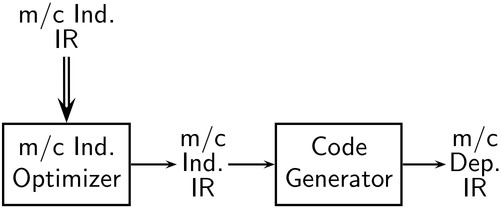
Typical Back Ends in Aho Ullman Model



- Compile time evaluations
- Eliminating redundant computations

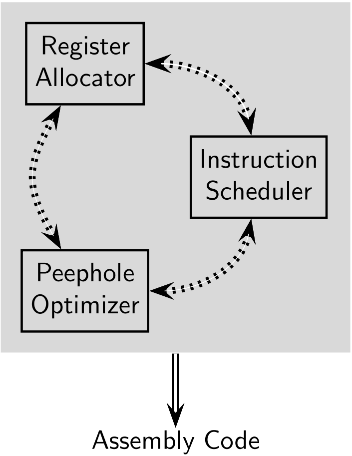
- Instruction Selection
- Local Reg Allocation
- Choice of Order of Evaluation

Typical Back Ends in Aho Ullman Model



- Compile time evaluations
- Eliminating redundant computations

- Instruction Selection
- Local Reg Allocation
- Choice of Order of Evaluation



Retargetability in Aho Ullman and Davidson Fraser Models

	Aho Ullman Model	Davidson Fraser Model
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		

Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davidson Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">• 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</td></tr>
<tr><td>Cost based tree pattern matching</td><td></td></tr>
<tr><td>Optimization</td><td></td><td></td></tr>
</table>

Retargetability in Aho Ullman and Davidson Fraser Models

	Aho Ullman Model	Davidson Fraser Model
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		

Retargetability in Aho Ullman and Davidson Fraser Models

	Aho Ullman Model	Davidson Fraser Model
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	

Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davidson Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">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</td></tr>
<tr><td>Cost based tree pattern matching</td><td>Structural tree pattern matching</td></tr>
<tr><td rowspan="2">Optimization</td><td rowspan="2">Machine independent</td><td>Machine dependent</td></tr>
<tr><td></td></tr>
</table>

Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davidson Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">• 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</td></tr>
<tr><td>Cost based tree pattern matching</td><td>Structural tree pattern matching</td></tr>
<tr><td rowspan="2">Optimization</td><td rowspan="2">Machine independent</td><td>Machine dependent</td></tr>
<tr><td>Key Insight: Register transfers are target specific but their form is target independent</td></tr>
</table>