

《编译原理》书面作业

截止日期：2025 年 12 月 9 日

若发现问题，或有任何想法（改进题目、调整任务量等等），请及时联系助教

Q1. 参考讲义中进行语句翻译的 L-翻译模式片断及所用到的语义函数。无需考虑 `break` 语句。

1. 若在基础文法中增加产生式 $E \rightarrow E \bar{\wedge} E$ ，其中与非运算符 $\bar{\wedge}$ 定义为 $P \bar{\wedge} Q \equiv \neg(P \wedge Q)$ ，试给出该产生式的语义动作集合。请利用短路运算进行翻译。
2. 若在基础文法中增加产生式 $S \rightarrow \text{repeat } S \text{ until } E$ ，试给出该产生式的语义动作集合。

参考答案：

1.

```
E → { E1.false := E.true; E1.true := newlabel } E1  $\bar{\wedge}$ 
      { E2.false := E.true; E2.true := E.false } E2
      { E.code := E1.code || gen(E1.true ':') || E2.code }
```

2.

```
S → repeat { S1.next := newlabel } S1 until
      { E.true = S.next; E.false = newlabel } E
      { S.code := gen(E.false ':') || S1.code || gen(S1.next ':') || E.code }
```

Q2. 参考讲义中使用拉链与代码回填技术进行语句翻译的 S-翻译模式片断及所用到的语义函数。拉链与代码回填技术中用到的只有空产生式的非终结符用 M 表示。无需考虑 `break` 语句。

1. 若在基础文法中增加产生式 $E \rightarrow E \bar{\wedge} E$ ，其中与非运算符 $\bar{\wedge}$ 定义为 $P \bar{\wedge} Q \equiv \neg(P \wedge Q)$ ，试给出该产生式的语义动作集合。请利用短路运算进行翻译。
2. 若在基础文法中增加产生式 $S \rightarrow \text{repeat } S \text{ until } E$ ，试给出该产生式的语义动作集合。

参考答案：

1.

```
E → E1  $\bar{\wedge}$  ME2 { backpatch(E1.truelist, M.gotostm);
                  E.truelist := merge(E1.falselist, E2.falselist);
                  E.falselist := E2.falselist; }
```

2.

$$S \rightarrow \text{repeat } M_1 S_1 \text{ until } M_2 E$$

```

{ backpatch(S1.nextlist, M2.gotostm);
  backpatch(E.falselist, M1.gotostm);
  S.nextlist := E.truelist; }

```

Q3. 考虑一个简单的栈式虚拟机。该虚拟机维护一个存放整数的栈，并支持如下 3 条指令：

- Push n ：将整数 n 压栈；
- Plus：弹出栈顶元素 n_1 和次栈顶元素 n_2 ，计算 $n_1 + n_2$ 的值，把结果压栈；
- Minus：弹出栈顶元素 n_1 和次栈顶元素 n_2 ，计算 $n_1 - n_2$ 的值，把结果压栈。

一条或多条指令构成一个指令序列。初始状态下，虚拟机的栈为空。给定一个仅包含加法和减法的算术表达式语言：

$$A \rightarrow A + A \mid A - A \mid (A) \mid \underline{int}$$

终结符 $\underline{int}$ 表示一个整数，用 $\underline{int}.val$ 取得语法符号对应的语义值。任何一个算术表达式都可以翻译为一个指令序列，使得该虚拟机执行完此指令序列后，栈中仅含一个元素，且它恰好为表达式的值。简单起见，我们用 “||” 来拼接两个指令序列。例如，算术表达式 $1 + 2 - 3$ 可翻译成指令序列

Push 3|| Push 2|| Push 1|| Plus || Minus

执行完成后，栈顶元素为 0。

1. 上述翻译过程可描述成如下 S-翻译模式，其中综合属性 $A.instr$ 表示 A 对应的指令序列。请补全空缺的部分。

$$\begin{aligned}
 A &\rightarrow A_1 + A_2 \{ A.instr := \dots \} \\
 A &\rightarrow A_1 - A_2 \{ A.instr := \dots \} \\
 A &\rightarrow (A_1) \{ A.instr := A1.instr \} \\
 A &\rightarrow \underline{int} \{ A.instr := \text{Push } \underline{int}.val \}
 \end{aligned}$$

2. 给上述虚拟机新增一个变量表，支持读取和写入变量对应的整数值。新增如下指令：

- Load x ：从表中读取变量 x 的值，将其压栈；
- Store x ：将栈顶元素弹出，作为变量 x 的值写入表中；
- Cmp：若栈顶元素大于或等于 0，则修改栈顶元素为 1；否则，修改栈顶元素为 0；
- Cond：若栈顶元素非 0，则弹出栈顶元素；否则，弹出栈顶元素和次栈顶元素后，压入整数 0。

考虑一个仅支持赋值语句的简单语言 L:

$$\begin{aligned}
 S &\rightarrow \underline{id} := E \mid S; S \\
 E &\rightarrow A \mid E \text{ if } B \\
 A &\rightarrow A + A \mid A - A \mid (A) \mid \underline{int} \mid \underline{id} \\
 B &\rightarrow A > A \mid B \wedge B \mid \neg B \mid \underline{true} \mid \underline{false}
 \end{aligned}$$

终结符 $\underline{id}$ 表示一个变量, 用 $\underline{id.val}$ 取得语法符号对应的语义值。算术表达式新增 $\underline{id}$, 用来读取变量 $\underline{id}$ 的值。赋值语句 $\underline{id} := E$ 表示将表达式 E 的值写入变量 $\underline{id}$ 。条件表达式 $E \text{ if } B$ 的语义为: 若布尔/关系表达式 B 求值为真, 则该表达式的值为 E 的值, 否则为 0。布尔/关系表达式中, $>$ 为大于, $\wedge$ 为逻辑与, $\neg$ 为逻辑非, $\underline{true}$ 为真, $\underline{false}$ 为假。设 P 为 L 语言的一个程序, 若 P 中所有被读取的变量, 在读取之前都已经被赋过值, 那么称 P 为合法程序。任何一个 L 语言的合法程序都可以翻译为一个指令序列, 使得该虚拟机执行完此指令序列后, 对任意程序中出现的变量, 表中所存储的值等于程序执行后的实际值。

上述翻译过程可描述成如下 S-翻译模式 (与 (1) 中相同的部分已省略)。请补全空缺的部分。

$$\begin{aligned}
 S &\rightarrow \underline{id} := E \{ S.instr := E.instr \mid \mid \text{Store } \underline{id} \} \\
 S &\rightarrow S_1; S_2 \{ S.instr := S1.instr \mid \mid S2.instr \} \\
 E &\rightarrow E_1 \text{ if } B \{ E.instr := \dots \} \\
 A &\rightarrow \underline{id} \{ A.instr := \text{Load } \underline{id.val} \} \\
 B &\rightarrow A_1 > A_2 \{ B.instr := \dots \} \\
 B &\rightarrow B_1 \wedge B_2 \{ B.instr := \dots \} \\
 B &\rightarrow \neg B_1 \{ B.instr := \dots \} \\
 B &\rightarrow \underline{true} \{ B.instr := \text{Push } 1 \} \\
 B &\rightarrow \underline{false} \{ B.instr := \text{Push } 0 \}
 \end{aligned}$$

参考答案:

1.

$$\begin{aligned}
 A &\rightarrow A_1 + A_2 \{ A.instr := A2.instr \mid \mid A1.instr \mid \mid \text{Plus} \} \\
 A &\rightarrow A_1 - A_2 \{ A.instr := A2.instr \mid \mid A1.instr \mid \mid \text{Minus} \}
 \end{aligned}$$

2. 答案如下所示, 其中 $A_1 > A_2$ 等价于 $(A_1 - A_2 \geq 0) \text{ if } (A_1 - A_2 \neq 0)$; $B_1 \wedge B_2$ 等价于 $B_2 \text{ if } B_1$; $\neg B_1$ 等价于 $1 \text{ if } (B_1 - 1 \neq 0)$ 。

$$\begin{aligned}
 E &\rightarrow E_1 \text{ if } B \{ E.instr := E1.instr \mid \mid B.instr \mid \mid \text{Cond} \} \\
 B &\rightarrow A_1 > A_2 \{ B.instr := A2.instr \mid \mid A1.instr \mid \mid \text{Minus} \mid \mid \text{Cmp} \\
 &\quad \mid \mid A2.instr \mid \mid A1.instr \mid \mid \text{Minus} \mid \mid \text{Cond} \} \\
 B &\rightarrow B_1 \wedge B_2 \{ B.instr := B2.instr \mid \mid B1.instr \mid \mid \text{Cond} \} \\
 B &\rightarrow \neg B_1 \{ B.instr := \text{Push } 1 \mid \mid \text{Push } 1 \mid \mid B1.instr \mid \mid \text{Minus} \mid \mid \text{Cond} \}
 \end{aligned}$$

答案不唯一, 其他正确答案亦可。

Principles and Practice of Compiler Construction HW

Due: 12/9/2025

If you have any suggestions, feel free to contact the TAs.

Q1. Consider the L-translation scheme used for statement translation in the slides, and the semantic functions included.

*No need to consider **break** statement.*

1. If we add production $E \rightarrow E \bar{\wedge} E$ to the basic grammar, where nand operator $\bar{\wedge}$ is defined as $P \bar{\wedge} Q \equiv \neg(P \wedge Q)$. Please give the semantic activity set of the production. Use short-circuit evaluation.
2. If we add production $S \rightarrow \text{repeat } S \text{ until } E$ to the basic grammar, please give the semantic activity set.

Solution:

1.

```
E → { E1.false := E.true; E1.true := newlabel } E1  $\bar{\wedge}$ 
      { E2.false := E.true; E2.true := E.false } E2
      { E.code := E1.code || gen(E1.true ':') || E2.code }
```

2.

```
S → repeat { S1.next := newlabel } S1 until
      { E.true = S.next; E.false = newlabel } E
      { S.code := gen(E.false ':') || S1.code || gen(S1.next ':') || E.code }
```

Q2. Consider the S-translation scheme used for statement translation in the slides when using backpatching technique, and the semantic functions included.

*The nonterminal with only empty production in backpatching is shown as M . No need to consider **break** statement.*

1. If we add production $E \rightarrow E \bar{\wedge} E$ to the basic grammar, where nand operator $\bar{\wedge}$ is defined as $P \bar{\wedge} Q \equiv \neg(P \wedge Q)$. Please give the semantic activity set of the production. Use short-circuit evaluation.
2. If we add production $S \rightarrow \text{repeat } S \text{ until } E$ to the basic grammar, please give the se-

mantic activity set.

Solution:

1.

```
 $E \rightarrow E_1 \bar{\wedge} ME_2$  { backpatch(E1.truelist, M.gotostm);  
                        E.truelist := merge(E1.falselist, E2.falselist);  
                        E.falselist := E2.falselist; }
```

2.

```
 $S \rightarrow$  repeat  $M_1S_1$  until  $M_2E$   
        { backpatch(S1.nextlist, M2.gotostm);  
          backpatch(E.falselist, M1.gotostm);  
          S.nextlist := E.truelist; }
```

Q3. Consider a simple stack-based virtual machine. The virtual machine maintains a stack containing integers, and supports the following 3 instructions:

- Push n : push integer n into the stack;
- Plus: pop the stack top element n_1 and second-top element n_2 , push $n_1 + n_2$ into the stack;
- Minus: pop the stack top element n_1 and second-top element n_2 , push $n_1 - n_2$ into the stack.

One or more instructions make a instruction sequence. In the initial state, the stack is empty. Given a arithmetic expression language containing only addition and subtraction:

$$A \rightarrow A + A \mid A - A \mid (A) \mid \underline{int}$$

Terminal int stands for an integer, we can use int.val to get the semantic value of the grammatical symbol. Any expression can be translated into an instruction sequence, such that, when the virtual machine finishes this instruction sequence, there is only one element in the stack, equaling the value of the expression.

To simplify, we use "||" to join two sequences. For example, expression $1 + 2 - 3$ can be translated into sequence

Push 3|| Push 2|| Push 1|| Plus || Minus

After running, the stack top element is 0.

1. The translation process above can be described as the following S-translation scheme, where synthesized attribute **A.instr** stands for the instruction sequence of A . Please fill

the blanks.

$$\begin{aligned}
 A &\rightarrow A_1 + A_2 \{ \text{A.instr} := \dots \} \\
 A &\rightarrow A_1 - A_2 \{ \text{A.instr} := \dots \} \\
 A &\rightarrow (A_1) \{ \text{A.instr} := A_1.\text{instr} \} \\
 A &\rightarrow \underline{int} \{ \text{A.instr} := \text{Push } \underline{int}.\text{val} \}
 \end{aligned}$$

2. Now give the virtual machine a variable table, enable it to load and store integer values of variables. Following new instructions are included:

- Load x : load the value of x from the table and push it to the stack;
- Store x : pop the stack top value and write into table as x ;
- Cmp: if the stack top element is greater or equal to 0, modify it to 1; otherwise, modify it to 0.
- Cond: if the stack top element is non-zero, pop it; otherwise, pop it and the second-top element, and then push a 0.

Consider a simple language L, which only consists of assignment statements:

$$\begin{aligned}
 S &\rightarrow \underline{id} := E \mid S; S \\
 E &\rightarrow A \mid E \underline{if} B \\
 A &\rightarrow A + A \mid A - A \mid (A) \mid \underline{int} \mid \underline{id} \\
 B &\rightarrow A > A \mid B \wedge B \mid \neg B \mid \underline{true} \mid \underline{false}
 \end{aligned}$$

Terminal $\underline{id}$ stands for a variable, use $\underline{id}.\text{val}$ to get its semantic value.

Add identifier expression $\underline{id}$, to read the value of variable $\underline{id}$.

Assignment statement $\underline{id} := E$ means write the value of E into variable $\underline{id}$. Condition expression $E \underline{if} B$ means: if boolean expression B has true value, the expression has the value of E ; otherwise, the value is 0.

In boolean / relational expressions, $>$ means "greater than", $\wedge$ means "logical and", $\neg$ means "logical not", $\underline{true}$ means "true", $\underline{false}$ means "false".

Let P be a program of language L. If all the loaded variables in P have been assigned before the load, then we call P a legal program.

Any legal program of language L can be translated into an instruction sequence, such that, after running the sequence, any variable in the program has the right value.

The translation process can be described as the following S-translation scheme (parts

included in (1) are ignored). Please fill the blanks.

```

S → id := E { S.instr := E.instr || Store id }
S → S1; S2 { S.instr := S1.instr || S2.instr }
E → E1 if B { E.instr := ... }
A → id { A.instr := Load id.val }
B → A1 > A2 { B.instr := ... }
B → B1 ∧ B2 { B.instr := ... }
B → ¬B1 { B.instr := ... }
B → true { B.instr := Push 1 }
B → false { B.instr := Push 0 }

```

Solution:

1.

```

A → A1 + A2 { A.instr := A2.instr || A1.instr || Plus }
A → A1 - A2 { A.instr := A2.instr || A1.instr || Minus }

```

2. $A_1 > A_2$ is the same as $(A_1 - A_2 \geq 0)$ if $(A_1 - A_2 \neq 0)$; $B_1 \wedge B_2$ is the same as B_2 if B_1 ; $\neg B_1$ is the same as 1 if $(B_1 - 1 \neq 0)$.

```

E → E1 if B { E.instr := E1.instr || B.instr || Cond }
B → A1 > A2 { B.instr := A2.instr || A1.instr || Minus || Cmp
               || A2.instr || A1.instr || Minus || Cond }
B → B1 ∧ B2 { B.instr := B2.instr || B1.instr || Cond }
B → ¬B1 { B.instr := Push 1 || Push 1 || B1.instr || Minus || Cond }

```

The solution is not unique.