《编译方法、技术与实践》第一版
修订日志

第一章

第二章

1. 2.2.8 节标题处
原文：2.2.8 GUN Bison 介绍
修改为：2.2.8 GNU Bison 介绍

2. 2.1.2 表 2.3 有误
原文：

r|(s|t)=(r|s)|r	| 是可结合的
r|(s|t)=rs|rt; (s|t)r=sr|tr	连接对 | 是可分配的

修改为：

r|(s|t)=(r|s)|t	| 是可结合的
r(s|t)=rs|rt; (s|t)r=sr|tr	连接对 | 是可分配的

3. 2.1.3 文字叙述错误
原文：同一个符号可以标记从个同一状态触发到达多个目标状态的多条边
修改为：同一个符号可以标记从同一状态触发到达多个目标状态的多条边

4. 2.1.4 图 2.7 的缩进问题
原文：

```
开始时，ε-closure(s0)是 Dstates 中的唯一状态，且它未加标记;
while (在 Dstates 中有一个未标记状态 T) {
    给 T 加上标记;
    for (每个输入符号 a) {
    U = ε-closure(move(T, a));
    if (U 不在 Dstates 中)
        将 U 加入到 Dstates 中，且不加标记;
        Dtran[T, a] = U;
    }
}
```

图 2.1 子集构造算法

修改为：

```
开始时，ε-closure(s0)是 Dstates 中的唯一状态，且它未加标记;
while (在 Dstates 中有一个未标记状态 T) {
        给 T 加上标记;
    for (每个输入符号 a) {
        U = ε-closure(move(T, a));
        if (U 不在 Dstates 中)
            将 U 加入到 Dstates 中，且不加标记;
        Dtran[T, a] = U;
        }
}
```

图 2.2 子集构造算法

5. 2.1.7 如果一个文法能够推导出具有两棵不同语法树的句子，则该文法是二义性的

原文： 如果一个文法能够推导出具有两棵不同语法树的句子，则该文法是二义性的（Ambiguous）。

修改为：如果一个文法对于同一个句子或句型，能够推导出具有两棵不同语法树，则该文法是二义性的（Ambiguous）。

6. 2.1.1 当词法分析器发现了一个标识符的词素时，它需要将这个词素添加到符号表中。…例如它的词素、类型、第一次出现的位置…都保存在符号表中。

原文： 当词法分析器发现了一个标识符的词素时，它需要将这个词素添加到符号表中。

…例如它的词素、类型、第一次出现的位置…都保存在符号表中。

修改为：词法分析时登记符号表应谨慎。词法分析时只能识别出标识符，不能确定它是函数还是变量，也不能确定哪个标识符属于哪个函数，如果第一次出现登记到符号表，会导致不同函数的同名变量只登记一次；如果每次出现都登记，则

同一变量会因声明和引用登记多次。

7. 2.1.4 识别相同语言的 DFA 的状态数有可能是 NFA 状态数的指数倍

原文：一般情况下，识别相同语言的 DFA 的状态数有可能是 NFA 状态数的指数倍，在这种情况下，我们试图实现 DFA 时会十分困难。
修改为：在一些情况下，识别相同语言的 DFA 的状态数有可能是 NFA 状态数的指数倍，这样一来，我们试图实现 DFA 时会十分困难。

8. 2.2.11 中表述修订

原文：略
修改为：

为什么会出现这种问题呢？主要原因在于，Bison并不会主动替我们维护这些位置信息，我们需要在Flex源代码文件中自行维护。不过只要稍加利用Flex中的某些机制，维护这些信息并不需要编写大量代码。我们可以在Flex源文件的开头部分定义变量yycolumn，并添加宏定义YY_USER_ACTION：

```
%option yylineno
…
%{
  /* 此处省略#include 部分 */
  int yycolumn = 1;
  #define YY_USER_ACTION \
    yylloc.first_line = yylloc.last_line = yylineno; \
    yylloc.first_column = yycolumn; \
    yylloc.last_column = yycolumn + yyleng - 1; \
    yycolumn += yyleng;
%}
```

在上述代码中，yylineno是内置变量，用于记录当前处理的行号；yylloc也是内置变量，表示当前词法单元所对应的位置信息；YY_USER_ACTION是宏，表示在执行每一个动作之前需要先被执行的一段代码，默认为空，而这里我们将其改成了对位置信息的维护代码。除此之外，最后还要在Flex源代码文件中做更改，即发现换行符之后对变量yycolumn进行复位：

```
…
%%
…
\n  { yycolumn = 1; }
```

另外，我们要在Bison源代码文件的第一部分加上%locations：

```
%locations
…
%%
…
```

在上述代码中，%locations指令用于启用Bison的位置跟踪功能，使Bison能够记录每个语法符号在源代码中的位置信息。

第三章

1. 3.1.4 表格（这部分只有一个表）
原文：缺少 caption。另外产生中那个 n，是否应该是个换行符\n?
修改思路：那个地方的 n 是作为结束符表示的，不是\n；应当在书中进行文字说明（TODO）

第四章

1. 4.2.7 第一个公式，最后一项
原文： array[i][j][
修改为：array[i][j][t]

2. 4.1.6、4.2.5 生成用于存放标号的新的临时变量，理论部分用的 NewLabel，实践部分用的 new_label
原文： NewLabel()
修改为：Label()
备注：代码中是写的 Label()，但是文本表述写的 NewLabel()；以代码中为准；

第五章

1. 图 5.1，有两个基本块 B6
原文： 略
修改为：需要更新图片（**TODO**）

2. 5.1.3 循环与分支处理中，“删除条件不成立时需要执行的无条件转移指令 5 和 11”
原文： 略
修改为： 思路如下：
似乎不需要修改，因为我看了一下，大概是正确的，这个地方，是想说我们可以把基本块重新排列，去掉一些不必要的 goto 语句。这个地方不是说 5.1 中的 V1<=1 我们就不跳转了，而是不先跳到图 5.1 中的 B3，再跳到 5.1 中的 B7。而是说可以因为 B3 没有其他内容的语句，只有一条 GOTO 语句，我们可以直接跳转到 B7，从而省去一个 GOTO。

另，我们那个地方示例是用的 149 页的代码片段，上面有代码序号；

3. 5.1.6 死代码消除部分的例题
原文： 略
修改为： 思路是“优化后的第 3 行代码应为 j label1，这是优化前的第 3 行代码得到的，而优化前的第 4 行代码 j label2 已经被删除。”（**TODO**）

3. 5.1.6 控制流优化部分的例题

原文： 略
修改为：思路是“优化前目标代码第 5-6 行，应当确保没有其他语句跳转到 label1 后才能删除。”（**TODO**）

第六章

1. 6.1.1 缩进的小更新

原文： 前面章节中已经介绍过，程序执行过程中，
修改为：前面章节中已经介绍过，程序执行过程中，
备注：缩进不大对，进行了调整。

2. 6.1.3 图 6.24 没有引用问题

原文：IN[B3]表示在基本块 B3 入口处的程序状态。
修改为：图 6.24 展示了状态合并的过程，IN[B3]表示在基本块 B3 入口处的程序状态。

3. 6.3.7 样例 （选做要求），测试用例符号错误

原文：

样例 4（循环不变代码外提）:
输入:

```
1   FUNCTION main :
2   READ t1
3   v1 := t1
4   READ t2
5   v2 := t2
6   t3 := v1 + #1
7   v3 := t3
8   t4 := v2 * #2
9   v4 := t5
10  t5 := v3 * v4
11  v5 := t5
12  t6 := v5 * v4
13  t7 := v1 + t6
14  v6 := t7
15  t8 := v3 - v1
16  v7 := t8
17  LABEL label1 :
18  v1 := v7
19  t9 := v7 * v4
20  v2 := t9
21  IF v3 > v4 GOTO label2
22  GOTO label3
23  LABEL label2 :
24  v3 := #4
25  t10 := v1 * #5
26  v4 := t10
27  GOTO label4
28  LABEL label3 :
29  t11 := v1 - #3
30  v8 := t11
31  LABEL label4 :
32  IF v1 > v2 GOTO label1
33  GOTO label5
```

```
34  LABEL label5 :
35  t12 := #2 * v4
36  v9 := t12
37  RETURN #0
```

修改为：

输入:

```
1   FUNCTION main :
2   READ t1
3   v1 := t1
4   READ t2
5   v2 := t2
6   t3 := v1 + #1
7   v3 := t3
8   t4 := v2 * #2
9   v4 := t4
10  t5 := v3 * v4
11  v5 := t5
12  t6 := v5 * v4
13  t7 := v1 + t6
14  v6 := t7
15  t8 := v3 - v1
16  v7 := t8
17  LABEL label1 :
18  v1 := v7
19  t9 := v7 * v4
20  v2 := t9
21  IF v3 > v4 GOTO label2
22  GOTO label3
23  LABEL label2 :
24  v3 := #4
25  t10 := v1 * #5
26  v4 := t10
27  GOTO label4
28  LABEL label3 :
29  t11 := v1 - #3
30  v8 := t11
31  LABEL label4 :
32  IF v1 > v2 GOTO label1
33  GOTO label5
34  LABEL label5 :
35  t12 := #2 * v4
36  v9 := t12
37  RETURN #0
```

4. 6.2.2 图叙述问题

原文：6.2.2 约束函数与传递函数的一个不同即在于，UNDEF∪c=c，其原因如下图

修改为：6.2.2 约束函数与传递函数的一个不同即在于，UNDEF∪c=c，其原因如下图 6.55 所示

5. 6.2.2 图 6.57 的引用问题

原文：在简单常量传播的基础上，我们介绍另外三种常量传播框架。

修改为：在简单常量传播的基础上，我们介绍另外三种常量传播框架。三种常量传播框架如**错误!未找到引用源。**所示。

备注：图 6.57 的题目改为“三种常量传播框架”