



普通高等教育“十一五”国家级规划教材

编译原理

(第 2 版)

陈意云 张 昱



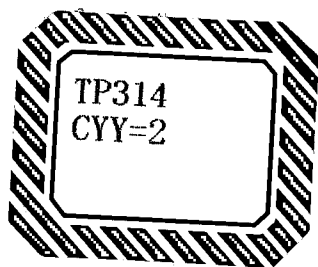
高等教育出版社
Higher Education Press

普通高等教育“十一五”国家级规划教材

编译原理

(第2版)

陈意云 张 昱



高等教育出版社

内容简介

本书介绍编译器构造的一般原理和基本实现方法,其内容包括词法分析、语法分析、语义分析、中间代码生成、目标代码生成、独立于机器的优化和依赖于机器的优化等。除了介绍命令式编程语言的编译技术外,本书还介绍面向对象语言和函数式编程语言的实现技术。本书还强调一些相关的理论知识,如形式语言和自动机理论、语法制导的定义和属性文法、类型论和类型系统等。

本书内容丰富,讲解深入,并注意理论联系实际,可作为高等学校计算机科学及相关专业的教材,也可供计算机软件工程技术人员参考使用。

图书在版编目(CIP)数据

编译原理/陈意云,张昱.—2版.—北京:高等教育出版社,2008.6

ISBN 978-7-04-023963-8

I. 编… II. ①陈…②张… III. 编译程序—程序设计—高等学校—教材 IV. TP314

中国版本图书馆 CIP 数据核字(2008)第 060077 号

策划编辑 刘 艳 责任编辑 萧 潇 封面设计 于文燕 责任绘图 郝 林
版式设计 马敬茹 责任校对 姜国萍 责任印制 韩 刚

出版发行 高等教育出版社
社 址 北京市西城区德外大街 4 号
邮政编码 100120
总 机 010-58581000

经 销 蓝色畅想图书发行有限公司
印 刷 北京外文印刷厂

开 本 787×1092 1/16
印 张 26.75
字 数 600 000

购书热线 010-58581118
免费咨询 800-810-0598
网 址 <http://www.hep.edu.cn>
<http://www.hep.com.cn>
网上订购 <http://www.landaco.com>
<http://www.landaco.com.cn>
畅想教育 <http://www.widedu.com>

版 次 2003 年 8 月第 1 版
2008 年 6 月第 2 版
印 次 2008 年 6 月第 1 次印刷
定 价 33.20 元

本书如有缺页、倒页、脱页等质量问题,请到所购图书销售部门联系调换。

版权所有 侵权必究

物料号 23963-00

前 言

本书改版的动力主要来自下列几个方面:

- (1) 编译器技术的发展,特别是计算机体系结构的发展对编译器技术的推动;
- (2) 第1版教材在使用中得到的积极反馈和建议;
- (3) 作者在教学实践中的心得和体会,在相关科研活动中的进展与收获。

第2版增加了有关新理论和新技术的介绍,重写了因理论和技术发展而需要调整的部分,改写了原先有疏漏而需要改善的地方,并且继承和发扬了前一版的特色。改动主要体现在:

(1) 增加了第10章“依赖于机器的优化”。简要介绍现代计算机体系结构、指令调度、基本块调度、全局调度、软件流水、并行性和数据局部性优化。

(2) 重写了第9章“独立于机器的优化”。突出数据流分析的理论基础,强调在数据流分析的一般框架下解决各个具体数据流问题,使本章立足于更高层次讨论代码优化。

(3) 改写了第5章“类型检查”的前两节。减少一些不必要的概念,把保留的概念区分得更清楚一些。

(4) 删掉一些过时的或较少使用的内容。此外,考虑到Pascal语言目前已较少使用,因此,把第1版中采用的大部分Pascal示例代码改为C语言示例代码。

本次改版主要参考了文献[9],在此向作者表示衷心的感谢。

特别感谢中国科学院计算技术研究所张兆庆研究员和冯晓兵研究员,他们在百忙之中仔细审阅了全稿,并对书稿提出了宝贵的意见。中国科学技术大学为本书的编写提供了充分的资金支持。付雄博士为新增的第10章准备了初稿。在此谨向所有为本次改版提供支持帮助的人致以最诚挚的谢意。

未加星号的各章已构成编译原理知识的一个基本架构,本科阶段的教学可在此基础上适当地增删,加星号的各章可供更深入学习时使用。与本书配套的习题解答是由作者编写、高等教育出版社出版的《编译原理习题精选与解析》。由作者编写的课程实践教材《编译原理实验教程》将在2008年底由高等教育出版社出版。作者讲授编译原理课程的教学课件和历年试题可在作者的教学网页(<http://staff.ustc.edu.cn/~yiyun>, <http://staff.ustc.edu.cn/~yuzhang>)上下载。

限于时间和学识水平,书中难免存在不妥和错误之处。如果发现本书有任何错误或有任何建议,欢迎给作者发送电子邮件(yiyun@ustc.edu.cn, yuzhang@ustc.edu.cn)批评指正,作者将及时改正错误。

作 者

于中国科学技术大学

2008年3月

第 1 版前言

本书介绍编译器构造的一般原理、基本设计方法和主要实现技术,可作为高等院校计算机科学及相关专业的教材。

虽然只有少数人从事构造或维护程序设计语言编译器的工作,但是编译原理和技术对高校学生和计算机软件工程技术人员来说仍是重要的基础知识之一。本书能使读者对程序设计语言的设计和实现有深刻的理解,对和程序设计语言有关的理论有所了解,对宏观上把握程序设计语言来说,能起一个奠基的作用。本书的学习还有助于读者快速理解、定位和解决在程序调试与运行中出现的问题。

对软件工程来说,编译器是一个很好的实例(基本设计、模块划分、基于事件驱动的编程等),本书所介绍的概念和技术能应用到一般的软件设计之中。

大多数程序员同时也是语言的设计者,虽然是一些简单语言(如输入输出、脚本语言)的设计者,但学习本书有助于提高他们设计这些语言的水平。

编译技术在软件安全、程序理解和软件逆向工程等方面有着广泛的应用。

作为一本教材,本书有如下一些特点:

(1) 在介绍语言实现技术的同时,强调一些相关的理论知识,如形式语言和自动机理论、语法制导的定义和属性文法、类型论和类型系统等。它们是计算机专业理论知识的一个重要部分,在本书中结合应用来介绍这些知识,有助于学生较快领会和掌握。

(2) 在介绍编译器各逻辑阶段的实现时,强调形式化描述技术,并以语法制导定义作为翻译的主要描述工具。

(3) 强调对编译原理和技术的宏观理解及全局把握,而不把读者的注意力分散到一些枝节的算法上,如计算开始符号集合和后继符号集合的算法、回填技术等。出于同样的目的,本书较详细地介绍了编译系统和运行系统。

(4) 本书还介绍面向对象语言和函数式语言的实现技术,有助于加深读者对语言实现技术的理解。书中带星号的章节,作为教学的可选部分。

(5) 作为原理性教材,本书介绍基本的理论和方法,而不偏向于某种源语言或目标机器。

(6) 我们鼓励读者用所学的知识去分析和解决实际问题,因此本书中有很多习题是从实际碰到的问题中抽象出来的。这些习题也能激发读者学习编译原理和技术的积极性。

(7) 为了便于读者学习,本书配有习题解答(见参考文献 8)。

本书的多数章节是参考了参考文献 1 和参考文献 7 编写的,部分习题取自参考文献 8。在此向有关作者表示感谢。



本书第 1 章到第 6 章以及第 12 章主要由陈意云编写,第 7 章到第 11 章主要由张昱编写。作者的学生陈晖准备了 10.2 节的初稿,李筱青准备了 10.3 节的初稿,吴萍和项森也为第 10 章的编写做了很多技术工作。

中国科学院软件研究所研究员程虎先生审阅了全书,并提出了许多宝贵的意见,在此表示衷心的感谢。

由于作者水平有限,书中难免还存在一些缺点和错误,恳请广大读者批评指正。

作 者

于中国科学技术大学

2003 年 5 月

目 录

第 1 章 引论	1	2.3 有限自动机	23
1.1 编译器概述	1	2.3.1 不确定的有限自动机	23
1.1.1 词法分析	1	2.3.2 确定的有限自动机	24
1.1.2 语法分析	3	2.3.3 NFA 到 DFA 的变换	25
1.1.3 语义分析	3	2.3.4 DFA 的化简	28
1.1.4 中间代码生成	4	2.4 从正规式到有限自动机	30
1.1.5 代码优化	4	2.5 词法分析器的生成器	32
1.1.6 代码生成	5	习题 2	35
1.1.7 符号表管理	5	第 3 章 语法分析	38
1.1.8 阶段的分组	6	3.1 上下文无关文法	38
1.1.9 解释器	7	3.1.1 上下文无关文法的定义	39
1.2 编译器技术的应用	7	3.1.2 推导	40
1.2.1 高级语言的实现	7	3.1.3 分析树	41
1.2.2 针对计算机体系结构的 优化	8	3.1.4 二义性	42
1.2.3 新计算机体系结构的设计	9	3.2 语言和文法	43
1.2.4 程序翻译	10	3.2.1 正规式和上下文无关文法的 比较	43
1.2.5 提高软件开发效率的工具	11	3.2.2 分离词法分析器的理由	44
习题 1	12	3.2.3 验证文法产生的语言	44
第 2 章 词法分析	13	3.2.4 适当的表达式文法	45
2.1 词法记号及属性	13	3.2.5 消除二义性	46
2.1.1 词法记号、模式、词法单元	14	3.2.6 消除左递归	47
2.1.2 词法记号的属性	15	3.2.7 提左因子	48
2.1.3 词法错误	16	*3.2.8 非上下文无关的语言构造	49
2.2 词法记号的描述与识别	16	*3.2.9 形式语言鸟瞰	51
2.2.1 串和语言	16	3.3 自上而下分析	52
2.2.2 正规式	17	3.3.1 自上而下分析的一般方法	52
2.2.3 正规定义	19	3.3.2 LL(1) 文法	53
2.2.4 状态转换图	20	3.3.3 递归下降的预测分析	54

3.3.4	非递归的预测分析	56	计算	111	
3.3.5	构造预测分析表	58	4.2.1	语法树	111
3.3.6	预测分析的错误恢复	59	4.2.2	构造语法树的语法 制导定义	112
3.4	自下而上分析	62	4.2.3	S 属性的自下而上计算	113
3.4.1	归约	62	4.3	L 属性定义的自上而下 计算	115
3.4.2	句柄	63	4.3.1	L 属性定义	116
3.4.3	用栈实现移进-归约分析	64	4.3.2	翻译方案	116
3.4.4	移进-归约分析的冲突	65	4.3.3	预测翻译器的设计	120
3.5	LR 分析器	67	4.3.4	用综合属性代替继承 属性	121
3.5.1	LR 分析算法	67	4.4	L 属性的自下而上计算	122
3.5.2	LR 文法和 LR 分析方法的 特点	71	4.4.1	删除翻译方案中嵌入的 动作	122
3.5.3	构造 SLR 分析表	72	4.4.2	分析栈上的继承属性	123
3.5.4	构造规范的 LR 分析表	79	4.4.3	模拟继承属性的计算	125
3.5.5	构造 LALR 分析表	83	习题 4	127	
3.5.6	非二义且非 LR 的上下文 无关文法	86	第 5 章	类型检查	130
3.6	二义文法的应用	87	5.1	类型在编程语言中的作用	130
3.6.1	使用算符的优先级和结合性 来解决冲突	88	5.1.1	执行错误和安全语言	131
3.6.2	使用其他约定来解决冲突	90	5.1.2	类型化语言和类型系统	131
3.6.3	LR 分析的错误恢复	91	5.1.3	类型化语言的优点	133
3.7	语法分析器的生成器	93	5.2	描述类型系统的语言	134
3.7.1	分析器的生成器 Yacc	93	5.2.1	定型断言	135
3.7.2	用 Yacc 处理二义文法	96	5.2.2	定型规则	136
3.7.3	Yacc 的错误恢复	99	5.2.3	类型检查和类型推断	137
习题 3		100	5.3	一个简单类型检查器的 规范	137
第 4 章	语法制导的翻译	106	5.3.1	一个简单的语言	137
4.1	语法制导的定义	106	5.3.2	类型系统	138
4.1.1	语法制导定义的形式	107	5.3.3	类型检查	140
4.1.2	综合属性	108	5.3.4	类型转换	141
4.1.3	继承属性	108	*5.4	多态函数	142
4.1.4	属性依赖图	109	5.4.1	为什么要使用多态函数	143
4.1.5	属性计算次序	110			
4.2	S 属性定义的自下而上				



5.4.2 类型变量	144	6.4.1 值调用	180
5.4.3 一个含多态函数的语言	145	6.4.2 引用调用	181
5.4.4 代换、实例和合一	146	6.4.3 换名调用	181
5.4.5 多态函数的类型检查	147	*6.5 堆管理	182
5.5 类型表达式的等价	151	6.5.1 内存管理器	183
5.5.1 类型表达式的结构等价	151	6.5.2 计算机内存分层	184
5.5.2 类型表达式的名字等价	152	6.5.3 程序局部性	185
5.5.3 记录类型	153	6.5.4 手工回收请求	186
5.5.4 类型表示中的环	154	习题6	186
5.6 函数和算符的重载	154	第7章 中间代码生成	199
5.6.1 子表达式的可能类型		7.1 中间语言	200
集合	155	7.1.1 后缀表示	200
5.6.2 缩小可能类型的集合	156	7.1.2 图形表示	200
习题5	157	7.1.3 三地址代码	201
第6章 运行时存储空间的组织和		7.1.4 静态单赋值形式	203
管理	163	7.2 声明语句	204
6.1 局部存储分配	164	7.2.1 过程中的声明	204
6.1.1 过程	164	7.2.2 作用域信息的保存	205
6.1.2 名字的作用域和绑定	165	7.2.3 记录的域名	207
6.1.3 活动记录	165	7.3 赋值语句	207
6.1.4 局部数据的安排	166	7.3.1 符号表中的名字	208
6.1.5 程序块	167	7.3.2 数组元素的地址计算	208
6.2 全局栈式存储分配	168	7.3.3 数组元素地址计算的翻译	
6.2.1 运行时内存的划分	168	方案	209
6.2.2 活动树和运行栈	169	7.3.4 类型转换	212
6.2.3 调用序列	171	7.4 布尔表达式和控制流语句	213
6.2.4 栈上可变长度数据	173	7.4.1 布尔表达式	214
6.2.5 悬空引用	174	7.4.2 控制流语句的翻译	214
6.3 非局部名字的访问	175	7.4.3 布尔表达式的控制流	
6.3.1 无过程嵌套的静态		翻译	216
作用域	175	7.4.4 开关语句的翻译	218
*6.3.2 有过程嵌套的静态		7.4.5 过程调用的翻译	220
作用域	176	习题7	221
*6.3.3 动态作用域	179	第8章 代码生成	227
6.4 参数传递	180	8.1 代码生成器设计中的问题	227

8.1.1	目标程序	227	9.2.5	可用表达式	267
8.1.2	指令选择	228	9.2.6	小结	269
8.1.3	寄存器分配	229	9.3	数据流分析的基础	270
8.1.4	计算次序选择	229	9.3.1	半格	270
8.2	目标语言	230	9.3.2	迁移函数	273
8.2.1	目标机器的指令集	230	9.3.3	一般框架的迭代算法	274
8.2.2	指令的代价	231	9.3.4	数据流解的含义	276
8.3	基本块和流图	233	9.4	常量传播	278
8.3.1	基本块	233	9.4.1	常量传播框架的数据 流值	278
8.3.2	基本块的优化	234	9.4.2	常量传播框架的迁移 函数	279
8.3.3	流图	235	9.4.3	常量传播框架的单调性	280
8.3.4	下次引用信息	236	9.4.4	常量传播框架的非 分配性	280
8.4	一个简单的代码生成器	237	9.4.5	结果的解释	281
8.4.1	寄存器描述和地址描述	238	9.5	部分冗余删除	282
8.4.2	代码生成算法	238	9.5.1	冗余的根源	282
8.4.3	寄存器选择函数	239	9.5.2	能否删除所有的冗余	284
8.4.4	为变址和指针语句 产生代码	241	9.5.3	惰性代码移动问题	285
8.4.5	条件语句	241	9.5.4	预期表达式	286
习题 8		242	9.5.5	惰性代码移动算法	286
*第 9 章	独立于机器的优化	250	9.6	流图中的循环	291
9.1	优化的主要种类	250	9.6.1	支配结点	291
9.1.1	优化的主要源头	250	9.6.2	回边和可归纳性	293
9.1.2	一个实例	251	9.6.3	流图的深度	294
9.1.3	公共子表达式删除	252	9.6.4	自然循环	294
9.1.4	复写传播	255	9.6.5	迭代流图算法的收敛 速度	296
9.1.5	死代码删除	256	习题 9		297
9.1.6	代码外提	256	*第 10 章	依赖于机器的优化	306
9.1.7	强度削弱和归纳变量 删除	257	10.1	处理器体系结构	307
9.2	数据流分析介绍	258	10.1.1	指令流水线 and 分支延迟	307
9.2.1	数据流抽象	259	10.1.2	流水化的执行	308
9.2.2	数据流分析模式	260	10.1.3	多指令发射	308
9.2.3	到达一定值	261			
9.2.4	活跃变量	265			



10.2 代码调度的约束	309	10.6.3 循环级并行	334
10.2.1 数据相关	309	10.6.4 数据局部性	335
10.2.2 发现内存访问中的 相关性	310	10.6.5 矩阵乘法算法	336
10.2.3 寄存器使用和并行执行 之间的折中	311	10.6.6 矩阵乘法算法的优化	338
10.2.4 寄存器分配和代码调度的 次序安排	312	习题 10	340
10.2.5 控制相关	313	第 11 章 编译系统和运行系统	343
10.2.6 投机执行的支持	313	11.1 C 语言的编译系统	343
10.2.7 一个基本的机器模型	314	11.1.1 预处理器	344
10.3 基本块调度	315	11.1.2 汇编器	345
10.3.1 数据依赖图	315	11.1.3 连接器	346
10.3.2 基本块的表调度	316	11.1.4 目标文件的格式	347
10.3.3 区分优先级的拓扑次序	317	11.1.5 符号解析	349
10.4 全局代码调度	318	11.1.6 静态库	350
10.4.1 简单的代码移动	318	11.1.7 可执行目标文件及装入	352
10.4.2 向上的代码移动	319	11.1.8 动态连接	353
10.4.3 向下的代码移动	320	11.1.9 处理目标文件的一些 工具	354
10.4.4 更新数据相关	321	11.2 Java 语言的运行系统	355
10.4.5 全局调度的其他问题	321	11.2.1 Java 虚拟机语言简介	355
10.4.6 静态调度器和动态调度器 的交互	322	11.2.2 Java 虚拟机	356
10.5 软件流水	323	11.2.3 即时编译器	357
10.5.1 引言	323	* 11.3 无用单元收集	359
10.5.2 循环的软件流水	324	11.3.1 标记和清扫	359
10.5.3 寄存器分配和代码生成	326	11.3.2 引用计数	360
10.5.4 do-across 循环	327	11.3.3 拷贝收集	361
10.5.5 软件流水的目标和约束	328	11.3.4 分代收集	363
10.5.6 软件流水算法	329	11.3.5 渐增式收集	364
10.5.7 无环数据依赖图的调度	330	11.3.6 编译器与收集器之间的 相互影响	364
10.6 并行性和数据局部性优化 概述	331	习题 11	367
10.6.1 多处理器	331	* 第 12 章 面向对象语言的编译	371
10.6.2 应用中的并行性	333	12.1 面向对象语言的概念	371
		12.1.1 对象和对象类	371
		12.1.2 继承	372
		12.1.3 信息封装	374



12.2 方法的编译	374	13.3 抽象机的体系结构	396
12.3 继承的编译方案	377	.. 13.3.1 抽象机的栈	396
12.3.1 单一继承的编译方案	378	13.3.2 抽象机的堆	397
12.3.2 重复继承的编译方案	380	13.3.3 名字的寻址	398
习题 12	384	13.3.4 约束的建立	399
* 第 13 章 函数式语言的编译	387	13.4 指令集和编译	400
13.1 函数式编程语言简介	387	13.4.1 表达式	400
13.1.1 语言构造	387	13.4.2 变量的引用性出现	401
13.1.2 参数传递机制	389	13.4.3 函数定义	402
13.1.3 变量的自由出现和约束 出现	390	13.4.4 函数应用	404
13.2 函数式语言的编译简介	392	13.4.5 构造和计算闭包	407
13.2.1 几个受启发的例子	392	13.4.6 letrec 表达式和局部 变量	409
13.2.2 编译函数	394	习题 13	410
13.2.3 环境与约束	394	参考文献	412

第 1 章

引 论

从理论上说,构造专用计算机来直接执行某种高级语言写的程序是可能的。但是,实际上目前的计算机能执行的都是非常低级的机器语言。那么,一个基本问题是:高级语言的程序是怎样变成能在计算机上执行的机器语言程序的?

能够完成从一种语言到另一种语言的变换的软件称为翻译器,这两种语言分别叫做该翻译器的源语言和目标语言。编译器是一种翻译器,它进行语言变换的特点是目标语言比源语言低级。

本章通过简要描述编译器的各个组成部分以及编译器技术的各种应用来介绍编译这个课题。该课题涉及编程语言、计算机体系结构、形式语言理论、类型论、算法和软件工程等方面的知识。

1.1 编译器概述

编译器的的工作可以分成若干阶段,每个阶段的工作都是把源程序从一种表示变换成另一种表示。编译过程的一种典型分解见图 1.1,图中的每个方框表示它的一个阶段。

本节以赋值语句 $\text{position} = \text{initial} + \text{rate} * 60$ 的翻译(假定变量都是实型)为例,简要介绍编译的各个阶段。

1.1.1 词法分析

词法分析阅读构成源程序的字符流,按编程语言的词法规则把它们组成词法记号(token)流。对于一个词法单元,词法分析产生的记号是

(记号名,属性值)

二元组。记号名是同类词法单元共用的名称,而属性值是一个词法单元有别于同类中其他词法单元的特征值。赋值语句(1.1)的字符流在词法分析时被依次组成下面这些记号。

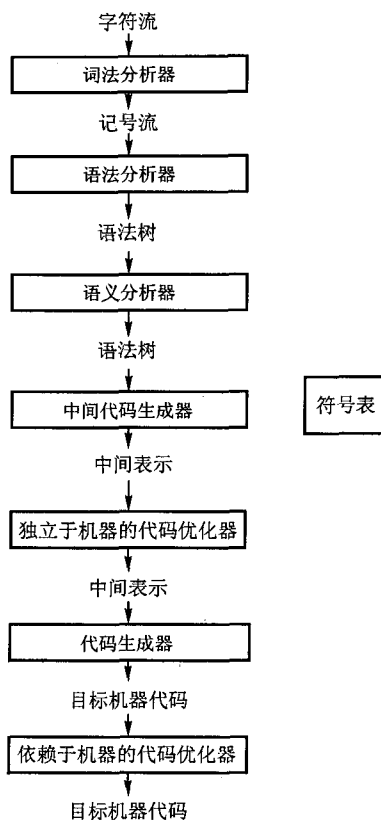


图 1.1 编译的各个阶段

(1) 标识符 position 形成的记号是 $\langle \text{id}, 1 \rangle$, 其中 **id** 是标识符的总称, 1 代表 position 在符号表中的条目, 符号表的条目用来存放标识符的各种属性, 如它的名字和类型。

(2) 赋值号 = 形成的记号是 $\langle \text{assign} \rangle$, 因为该记号只有一个实例, 因此不需要属性值来区分实例。为了直观起见, 下面直接用赋值号作为记号名, 写成 $\langle = \rangle$ 。

(3) 标识符 initial 形成的记号是 $\langle \text{id}, 2 \rangle$ 。

(4) 加号 + 形成的记号是 $\langle + \rangle$ 。

(5) 标识符 rate 形成的记号是 $\langle \text{id}, 3 \rangle$ 。

(6) 乘号 * 形成的记号是 $\langle * \rangle$ 。

(7) 数 60 形成的记号是 $\langle 60 \rangle$ 。从技术上讲, 程序中出现的常数也要放到符号表或单独的常数表中, 形成记号 $\langle \text{number}, 60 \text{ 在表中的条目} \rangle$ 。这个问题的讨论留到词法分析中具体介绍。为直观起见, 这里直接使用 60 在源程序中的字符序列作为记号名。

分隔记号的空格通常在词法分析时被删去。

于是, 赋值语句 (1.1) 在词法分析后形成的记号流是

$$\langle \text{id}, 1 \rangle \langle = \rangle \langle \text{id}, 2 \rangle \langle + \rangle \langle \text{id}, 3 \rangle \langle * \rangle \langle 60 \rangle \quad (1.2)$$

第2章讨论词法分析,词法分析也叫做线性分析或扫描。

1.1.2 语法分析

语法分析(syntax analysis)简称为分析(parsing),它按编程语言的语法规则检查词法分析输出的记号流是否符合这些规则,并依据这些规则所体现出的该语言的各种语言构造(construct,如函数、语句、表达式等)的层次性,用各记号的第一元建成一种树形的中间表示,这个中间表示用抽象语法的方式描绘了该记号流的语法情况。一种典型的中间表示是语法树。其中,内部结点表示运算,子结点代表该运算的运算对象。为记号流(1.2)建立的语法树见图1.2(a)。

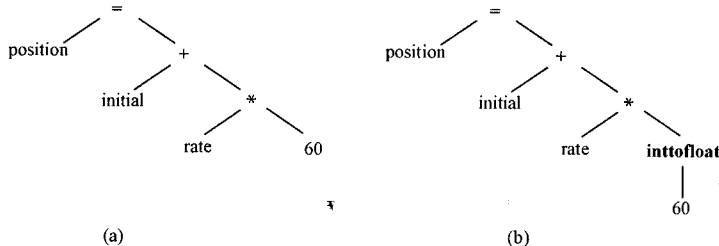


图 1.2 语义分析插入了类型转换

编译器后面各个阶段使用这种语法树进一步分析源程序和生成目标代码。第3章专门讨论语法分析算法,图1.2的语法树在4.2节讨论。在第4章的语法制导翻译中,还将详细讨论编译器如何利用输入所含的层次结构来产生语法树。

1.1.3 语义分析

语义分析阶段使用语法树和符号表中的信息,依据语言定义来检查源程序的语义一致性,以保证程序各部分能有意义地结合在一起。它还收集类型信息,把它们保存在符号表或语法树中。

语义分析的一个重要部分是类型检查,编译器检查每个算符的运算对象,看它们的类型是否适当。例如,当实数作为数组的下标时,许多语言的定义都要求编译器报告错误。语言定义也可能允许运算对象的类型隐式转换,例如当二元算术算符作用于一个整数和一个实数时,编译器会把其中的整数转换为实数。

例 1.1 在机器内部,整数的二进制表示和实数的二进制表示是有区别的,不论它们是否有相同的值。在图1.2中,所有的变量都是实型。另外,由60本身可知它是整数。对图1.2(a)进行类型检查会发现*作用于实型变量rate和整数60,可以建立一个额外的算符结点inttfloat

[见图 1.2(b)], 它显式地把整数转变为实数。 □

类型检查和语义分析在第 5 章讨论。

1.1.4 中间代码生成

经过语法分析和语义分析后,许多编译器为源程序产生更低级的显式中间表示,可以认为这种中间表示是一种抽象机的程序。中间表示必须具有两个性质:易于产生并且易于翻译成目标程序。

第 7 章主要采用叫做三地址代码的中间表示,它们像一种抽象机器的汇编语言,这种机器的每个存储单元的作用类似于寄存器。三地址代码由三地址指令序列组成,每条三地址指令最多有三个操作数,语句(1.1)的三地址代码如下:

```
t1 = inttfloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

(1.3)

这种中间形式有它的特点。首先,除了赋值算符外,每条指令至多还有一个算符,因此,在生成这些指令时,编译器必须决定运算完成的次序,语句(1.1)的乘优先于加。其次,编译器必须产生临时变量名,用以保留每条指令的计算结果。最后,某些三地址指令的运算对象不足三个,例如指令序列(1.3)的第一条和最后一条指令。

本书在第 7 章叙述编译器用的主要是中间表示。通常,除了计算表达式外,这些中间表示还要做其他事情,例如有条件控制转移、无条件控制转移和过程调用。第 4 章和第 7 章提供为编程语言典型构造产生中间代码的一些算法。

1.1.5 代码优化

独立于机器的代码优化阶段试图改进中间代码,以便产生较好的目标代码。通常,较好是指执行较快,但也可能是其他目标,如目标代码较短或目标代码执行时能耗较低。

如果中间代码生成算法比较简单,那么它将给代码优化留下很多机会。例如,一个比较自然的中间代码生成算法为语法树上的每个算符产生一条指令,因而得到(1.3)这样的中间代码。用这样的中间代码生成算法再跟上一个代码优化阶段是一种可行的办法,因为产生较优代码问题可以在代码优化阶段得以解决。例如,代码优化器会推断出,把 60 从整数转变为浮点数可以在编译时完成,从而用 60.0 代替 60 就可以把 `inttfloat` 运算删去。还有, `t3` 只被引用一次,就是取它的值传给 `id1`,因此用 `id1` 代替 `t3`,把(1.3)的最后一条指令删除也是可以的。这样,优化器可以得到如下结果:

```
t1 = id3 * 60.0
```


$$id1 = id2 + t1 \quad (1.4)$$

不同的编译器完成不同程度的优化,能完成大部分优化的编译器叫做“优化编译器”,但这时编译时间中相当可观的一部分都消耗在这种优化上。简单的优化也可以使目标程序的运行时间大大缩短,而编译速度并没有降低太多。第9章和第10章分别讨论独立于机器和依赖于机器的优化,第8章的代码生成也会涉及代码优化。

1.1.6 代码生成

代码生成取源程序的一种中间表示作为输入并把它映射到一种目标语言。如果目标语言是机器代码,则需要为源程序所用的变量选择寄存器或内存单元,然后把中间指令序列翻译为完成同样任务的机器指令序列。此阶段的一个关键问题是寄存器分配。

例如,使用寄存器 R1 和 R2,(1.4)的中间代码可以翻译成:

```
MOVF id3,R2
MULF #60.0,R2
MOVF id2,R1
ADDF R2,R1
MOVF R1,id1
```

(1.5)

每条指令的第一个和第二个操作数分别代表源和目的操作数,每条指令的 F 告知指令处理浮点数。(1.5)的代码把地址 id3 的内容取入寄存器 R2(暂且认为指令中 id3 代表对象 id3 的地址。变量的存储分配在第6章讨论),然后把它乘上实数 60.0,#号代表 60.0 作为立即数处理。第三条指令把 id2 取入寄存器 R1,第四条指令再把寄存器 R2 的值加上去,最后一条指令把寄存器 R1 的值存入地址 id1。这样,该段代码实现了语句(1.1)的赋值。代码生成在第8章讨论。

1.1.7 符号表管理

编译器的一项重要工作是记录源程序中使用的变量名字,并收集每个名字的各种属性。这些属性提供该名字有关存储分配、类型和作用域等信息。如果是过程名字,还有参数的个数、每个参数的类型、参数传递方式和返回值类型等。

符号表是为每个变量名字保存一个记录的数据结构,记录的域是该名字的属性。该数据结构应该设计成允许编译器迅速地找到一个名字的记录,并在此记录中迅速地存储和读取数据。符号表将在第7章讨论。

语句(1.1)的编译过程总结见图 1.3。