第12A期
2002年12月
电　子　学　报
ACTA ELECTRONICA SINICA
Vol.30　No.12A
Dec.　2002

编译系统中间代码的一种抽象表示

戴桂兰，张素琴，田金兰，蒋维杜
（清华大学计算机科学与技术系，北京 100084）

摘　要： 中间表示是提高编译器的可移植性和代码生成的有效性的关键技术.为提高编译成份的可复用性，简化编译器的构造，本文提出了一种描述程序语言抽象语法及编译器内部数据结构的抽象中间表示 AIR(Abstract Intermediate Representation).AIR 以代数数据类型为主体，并用面向对象特征对其进行扩展，从而使之具有简洁的语法，较强的表达能力、灵活性和可扩展性.AIR 将抽象描述与其具体实现相分离，可方便地用不同的高级程序语言实现，从而有助于提高编译成份间的互操作性.

关键词： 中间表示；代数数据类型；面向对象；编译系统
中图分类号： TP314　**文献标识码：** A　**文章编号：** 0372-2112 (2002) 12A-2134-04

An Abstract Intermediate Representation in Compilation Systems

DAI Gui-lan, ZHANG Su-qing, TIAN Jin-lan, JIANG Wei-du
(*Dept. of Computer Science & Technology, Tsinghua University, Beijing* 100084, *China*)

Abstract: The design of an intermediate representation is critical to compiler portability and code generation efficiency. In order to increase the reusability of compiler components, and to simplify the development process of compilers, the paper presents an abstract intermediate representation (AIR) that provides a concise notation for describing the abstract syntax of programming languages and the inner data structures of compilers. AIR integrates algebraic data types into the object-oriented paradigm and thus makes it have stronger expressive power, flexibility, and extensibility. AIR separates the abstract descriptions from the concrete implementation. This makes it easier to be implemented in different high-level languages and thus improves the interoperatibility of compiler components. We use AIR to simulate the core of the SUIF written in C + +. The result states that AIR has stronger expressive power and more compact syntax than the other languages for describing the abstract syntax of programming languages.

Key words: intermediate representations; algebraic datatypes; object-orientation; compiler systems

1　引言

随着嵌入式应用的迅速发展以及高性能体系结构不断推陈出新，对高质量编译器的快速开发提出了新的挑战和需求[1].面向我国嵌入式系统的应用需求，我们正在研制具有我国自主版权的编译基础设施.其主要设计目标在于使通过仅修改或替换某一成份，组装一个完整的编译系统成为可能，从而支持多源语言、多目标机编译器的快速开发.对编译前端的研究已有比较成熟的理论基础和实用的开发工具，并有一系列的高级语言的前端产品，如：C、C + + 和 Java 前端等.这些编译前端所输出的中间表示是多种多样的，主要包括语法树、三元式、四元式、DAG 等.它们虽为全局流分析、循环优化、数据流优化等编译趟提供了适当的模型，但并不能满足开发重定目标编译器的需要[2].而中间表示是提高编译器的可移植性和代码生成的有效性的关键技术[3].为此，我们提出了一种能够对现有的各种中间表示和编译器内部数据结构提供自然编码的抽象中间表示 AIR，其主要目标在于：一方面，利用对现有中间表示的抽象描述设施，可充分利用现有的编译前端，且方便于编译成份的不同层次的集成；另一方面，将前端所输出的中间代码用这种抽象中间表示进行描述，并以之作为后端的固定输入，从而简化编译后端的构造.

2　代数数据类型的扩展

如所知，函数式语言中的代数数据类型具有易于表达复杂的数据结构，从而使语法简洁等特点，而面向对象范型易于复用.AIR 的主要设计目标之一语言简洁，具有良好的可复用性和可扩展性，且支持模块化特征.因此，在 AIR 的设计过程中，我们充分利用了函数式范型与面向对象范型的互补性特征，沿用了标准的 ML 中的代数类型[4]、利用代数类型定义用户定义数据的结构，且利用面向对象方法对其进行扩展，以提高 AIR 的适用性和灵活性.已有一些学者在利用面向对象方法扩展代数数据类型方面做了一些研究工作[5].由于对这些

收稿日期：2002-06-06；修回日期：2002-09-10
基金项目：国家自然科学基金（No.60083004）

工作的分析比较不是本文的重点,下面对 *AIR* 中的扩展方式及其特性进行讨论.

2.1 代数数据类型变体的扩展

利用面向对象方法对代数数据类型进行扩展的一种方式是利用继承性以实现对代数类型变体的扩展.代数数据类型可建模为其各个变体的和.每个变体为一个构造子.为便于精确描述代数数据类型及其变体间的关系,我们引入下述定义:

定义 1: 偏序关系"$<$".若代数数据类型 Y 通过加入新的变体以对代数类型 X 进行扩充,则 $Y<X$ 成立.

定义 2: 函数 $allcases(Y)$ 表示代数类型 Y 的所有变体的集合,函数 $owncases(Y)$ 表示类型 Y 显式提供变体的集合,函数 $inherited(Y)$ 表示类型 Y 从其它类型中继承的变体的集合.且它们满足方程:

$$allcases(Y) = inherited(Y)\, owncases(Y)$$

其中,$owncases(Y) = \bigcup_i \{y_i\}$,

$$inherited(Y) = \bigcup_{Y<X,\, Y \neq X} owncases(X)$$

例如:代数类型 A 由两个构造子 C_1 和 C_2 组成,代数类型 B 继承了代数类型 A 的所有变体,并扩充了新变体 C_3.

$$\text{datatype } A = C_1(T_{1,1}X_{1,1}, \cdots, T_{1,r_1}X_{1,r_1}) \mid C_2(T_{2,1}X_{2,1}, \cdots, T_{2,r_2}X_{2,r_2});$$

$$\text{datatype } B{:}\text{public } A = C_3(T_{3,1}X_{3,1}, \cdots T_{3,r_3}X_{3,r_3});$$

即,$allcases(B) = inherited(B) \cup owncases(B) = allcases(A)\,(owncases \cup B) = \{C_1, C_2, C_3\}$.根据标准的类型规则,若代数类型 X 的所有变体也是 Y 的变体,即 $allcases(X) \subseteq allcases(Y)$,则 X 是 Y 的子类型.由此可见,在 *AIR* 中,代数类型变体的扩展为子类型扩展.

2.2 数据与操作的扩展

利用面向对象方法对代数数据类型进行扩展的另一种方式利用子类和重载特征为代数类型或构造子扩充新的操作或数据.例如:代数类型 B 扩充代数类型 A 通过添加新的操作 *print*( )来实现:

```
datatype B :public A
  with print( );
```

代数类型 B 的所有变体必须被子类化以便为新的操作提供实现.因此,为使这种扩展不需要对源代码进行修改,我们允许类型强制.即,当调用新的操作时,我们必须将接收者强制到扩充后的代数类型.例如:若调用代数类型 A 的实例 a 的 print()方法,则需要使用$((B)a)$.print(),否则将不能访问新的操作.数据的扩展也是通过这种方式实现的.

3 AIR 的基本语法结构

AIR 描述主要由代数类型、构造子和产生式三个基本成份组成.其中,代数类型由枚举该类型所有的构造子的产生式来定义.每个构造子有一系列域,用于描述与构造子有关的值的类型.为保持 AIR 命名空间与其它中间表示命名空间之间的简单映射,AIR 的类型名和构造子名为现有中间表示有效标识符的交集.另为便于区分代数类型名和构造子名,代数类型必须以大写字母开头,而构造子以小写字母开头.

3.1 代数类型的声明与实例化

由于多数中间表示采用了类树形的数据结构,为使 AIR 能对现有中间表示的语法和语义提供完整的描述,树形数据结构模板应是 AIR 的主要成份.另外,程序的内部信息需要基于图的数据结构来表示,如:控制流图、数据依赖图和符号表项等[7].为便于这些数据结构的灵活多变实现和管理,且易于在其上做一些优化工作,AIR 也提供了图结构的形式化模板.为提高语言的一致性,树结构和图结构采用了类似的定义方式.

一个代数类型能够继承一个或多个代数数据类型的特性,其效果为该代数类型所有的变体都将继承这些代数数据类型的特性.代数类型的声明用保留字 datatype 引导.每个 datatype 可以引导一个代数类型或代数类型缩写集合.每个代数类型应在相应的实现文件中实例化,并根据代数类型量词采用不同的实现方法.例如:若代数类型量词为 inline,则使用内联方法实现.若代数类型量词为 extern,则使用非内联方法实现.其中 inline 为缺省情况.其语法用扩展的 BNF 范式描述如下:

〈代数类型声明〉::= datatype
　[〈代数类型规格说明〉{and〈代数类型规格说明〉}];
　　[where type〈类型规格说明〉
　　{and〈类型规格说明〉}]
〈代数类型规格说明〉::=
　　〈标识符〉
　　[:〈继承列表〉]
　　[::〈代数类型量词〉]
　　[=〈构造子列表〉]
　　[node:〈结点定义〉{,〈结点定义〉}]
　　[edge:〈边定义〉{,〈边定义〉}]
〈代数类型实例化〉::= instantiate [extern] datatype
　　〈代数类型名〉[〈类型表达式〉{,〈类型表达式〉}]
　{,〈代数类型名〉[〈类型表达式〉{,〈类型表达式〉}]

其中,保留字 node 和 edge 用于引导图结构的结点和边的定义.AIR 提供了四种形式的代数类型构造子:(1)空构造子,这种构造子没有参数,不消耗堆空间;(2)元组参数构造子,这种构造子具有一个或多个无命名参数;(3)记录参数构造子,这种构造子具有一个或多个有命名参数.(4)空参数构造子,这种构造子有一个空参数.

3.2 代数类型的细化设施

代数类型的扩展可以通过代数类型变体的扩展与数据和操作的扩展两种方式实现.另外,为便于与软件工程逐步细化的原则保持一致,AIR 提供了的代数类型细化设施,即将代数类型定义与其操作的分离机制.利用这种机制可以提高 AIR 描述的可读性.代数类型的细化定义用保留字 refine 引导.利用保留字 with,成员函数和额外的属性可附加到该类型的各种变体中.

〈代数类型细化声明〉::= refine
〈代数类型细化规格说明〉{and〈代数类型细化规格说明〉};
〈代数类型细化规格说明〉::=

〈代数类型名或构造子名〉{,〈代数类型名或构造子名〉}
[:〈继承列表〉]
[::〈代数类型量词〉]
{〈代数类型体〉}
[node:〈结点定义〉{,〈结点定义〉}]
[edge:〈边定义〉{,〈边定义〉}]

其中,代数类型体的定义方式与 C++中类的属性或方法的定义方式相类似.

4 分析与评价

如前所述,具有简洁的语法,能对现有的中间表示提供自然的编码是 AIR 的主要设计目标.如所知,SUIF 是用 C++实现的一种典型的高级中间表示,它提供了表达语言程序的共同格式,其核心部分采用了基于类库的面向对象层次结构[6].下面以 AIR 对 SUIF 的中间格式的核心部分的描述为例,从表达能力、代码尺寸等方面对 AIR 进行定量和定性的分析评价.

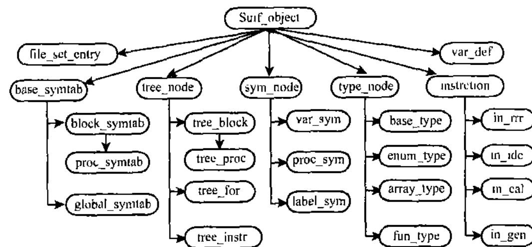

图 1

如图 1 所示[7],类 *tree_node*、*base_symtab* 和 *instruction* 等分别为类 *suif_object* 的子类,类 *block_symtab* 和 *global_symtab* 是 *base_symtab* 的子类,而类 *proc_symtab* 又是 *block_symtab* 的子类.这些类之间构成了三层继承关系.虽然利用代数类型的继承性可直接模拟这三层继承关系,但在这里,若将 *tree_node* 等类模拟为代数类型显然不太合适(不够简洁).因此,我们将类 *suif_object* 和类 *base_symtab* 模拟为代数类型,而将类 *block_symtab* 和 *global_symtab* 作为代数类型 *base_symtab* 的构造子.由于构造子本身不是一个代数类型,其特性不能被继承.为模拟类 *proc_symtab* 和 *block_symtab* 间的继承关系,必须引入额外的中间数据类型 *Block_symtab_Aux*,它由 *proc_symtab* 和 *block_symtab* 两个构造子组成.其中,构造子 *block_symtab* 具有类 *block_symtab* 的所有特征,而构造子 *proc_symtab* 不仅具有类 *proc_symtab* 的特性,而且还继承了类 *block_symtab* 所有特性,从而模拟了类 *proc_symtab* 和 *block_symtab* 间的继承关系,具体描述如图 2 所示.

```
datatype Suif_object
and Base_symtab:public Suif_object
    = block_symtab_aux(Block_symtab_aux)
    | global_symtab()
and Block_symtab_aux
    = block_symtab(ty1 f1,…,tyn fn)
    | proc_symtab(ty1 f1,…,tyn fn,ty'1 f'1,…,ty'n f'n);
```

图 2

通过分析 SUIF 核心部分的 C++源文件,我们给出了相应的 AIR 描述.AIR 描述与 C++源文件使用了同样的标识符集合.虽然 AIR 描述并不能实现对 C++实现的逐字转换,但它能以比较自然的方式捕捉 C++中的多数特征,且 AIR 编码比 C++实现具有较少的限制.因此,在将 SUIF 的 C++实现变换为等价的 AIR 描述的过程中没有失去任何信息.利用相应的工具对有关文件的行数、字数及字符数的统计,以比较 SUIF 的 AIR 描述与 ASDL 描述及其 C++实现代码的尺寸.从表 1 中我们可以看出,AIR 描述比 ASDL 描述略简洁一些,比 C++实现要简洁得多.

另外,在 C++中,指向树结点或外部数据结构(如:符号表)的指针形成了任意的图结构,由于 AIR 提供了图结构的模板类型,对这些问题的处理比较容易,且使程序较短,比较直观,从而方便了程序的正确性证明.

表 1

	文件数	行数	字数	字符数
C++核	10	2316	6 533	60 984
ASDL 描述[9]	1	204	562	6 921
AIR 描述	1	195	544	6 223

5 相关工作

声明性语言,如正则表达式和上下文无关文法,借助 LEX 和 YACC 等工具,有助于程序语言的具体语法的形式化表示的推广,而描述性语言,将为形式化地描述程序语言的抽象语法及编译器内部数据结构提供有效的手段.近年来,人们在抽象语法表示方面做了一些卓有成效的工作,研制了一些在理论和实践中有重要价值的抽象语法描述语言.但其中一些语言中包含的某些特征是编译器中间表示所不需要的.例如:ASN.1 的 13 种不同的字符串原子类型[8];SGML 的"标记最小化"(tag minimization)特征,使之定义的格式易于人写,但不便于进行分析[9].尽管 SGML 和 ASN.1 确实解决了成份间的互操作问题,但它们的语法结构比较复杂,且具有一些冗余信息,从而使之难以使用.Java 的超集 Pizza 是基于代数类型的抽象语法描述语言,它可以精确地描述 Java 类树形的数据结构,实现了代数类型的自动序列化.但 Pizza 不支持基于图的数据结构,且其产生的序列化不是与语言无关的[10].Zephyr 的 ASDL 与 AIR 具有基本相同的设计目标,但 ASDL 利用属性设施,只能处理两层继承,且对图数据结构的处理等方面也存在着不足[7].

6 结束语

本文提出了一个中间表示和编译器内部数据结构的抽象描述语言 AIR.AIR 用代数数据类型描述用户定义的数据结构,并利用面向对象方法对代数类型进行扩展,使之不仅具有简洁的语法,而且具有较强的表达能力、可复用性和可扩展性,从而使之能模拟现有的中间表示,又能适应中间表示将来的扩展.同时,AIR 提供了代数类型的细化设施,提高了 AIR 描述的可读性,且与软件工程的主要原则保持一致.另外,AIR 还提供了图结构的模板类型机制,这不仅简化了编译器内部

数据结构的描述,而且有助于对 AIR 描述进行优化.再者,AIR 将抽象描述与具体实现相分离,易于用不同的高级语言实现,这一方面提高了编译成份的互操作性(在另文中讨论),另一方面方便了复杂数据结构的实现.

不同编译前端所输出的各种中间表示能够统一地用 AIR 进行描述.这样,一方面可使编译后端接收固定形式的输入,从而简化编译后端的构造;另一方面可在统一的中间表示上做各种优化工作,从而大大减少开发优化编译器的工作量.

利用 AIR 模拟 SUIF 中间格式的实验表明,AIR 能以比较自然的方式捕捉 C++的多数特征,AIR 编码比 C++实现具有较少的限制且要简洁得多.与 ASDL 相类似,由于构造子不是代数类型,对于三层以上继承关系的模拟问题,是否引入额外的代数类型需要做一些折中考虑.另外,我们目前仅提供了 AIR 描述到 C++实现的转换工具以及 AIR 描述与标准二进制表示间的相互转换工具.它与 Java 等语言的转换工具将是我们下一步的工作.

参考文献:

[1] Nikil D, et al. New directions in compiler technology for embedded systems[A]. Proceedings of the Conference on Asia South Pacific Design Automation Conference[C]. Yokohama Japan: ASP-DAC, 2001.

[2] Ganapathi M. Affix grammar driven code generation[J]. ACM Transactions of Programming Languages and Systems, 1985.560 - 599.

[3] Fraser C W, et al. Engineering a simple, efficient code-generator generator[J]. ACM Letters on Programming Languages and Systems, 1992, 1 (3): 213 - 226.

[4] Harper R. Introduction to standard ML[R]. Technical Report ECS LFCS-86-14, School of Computer Science, Carnegie Mellon University, http://www.disi.unige.it USA, 1997.

[5] Zenger M, et al. Extensible algebraic datatypes with defaults[A]. In Proc. International Conference on Functional Programming (ICFP 2001)[C]. Italy: Firenze, 2001.

[6] Wilson R P, et al. SUIF: An infrastructure for research on parallelizing and optimizing compilers[J]. ACM SIGPLAN Notices, 1994, 29(10): 31 - 37.

[7] Wang D C, et al. The zephyr abstract syntax description[A]. USENIX Conference on Domain-Specific Languages[C]. USA: USENIX, 1997.

[8] ITU-T X.683. ISO and IEC, Information Technology-Abstract Syntax Notation One(ASN.1): Specification of Basic Notation[S]. Telecommunication Standardization Sector of ITU, 2000.7.

[9] Oxford: Clarendon Press, 1990. Goldfarb C F, et al. The SGML Handbook[S].

[10] Odersky M, et al. Pizza into java: translating theory into practice[A]. In Proceeding POPL[C]. Paris, 1997.15 - 17.

作者简介:

戴桂兰 女,1972 年生于河南新县,2000 年毕业于东南大学计算机科学与技术系,获工学博士学位,现为清华大学计算机系软件所博士后,主要研究方向为编译技术,面向对象技术.

张素琴 女,1945 年生于河北石家庄,责任教授,清华大学软件所所长,主要研究领域为编译技术,嵌入式软件开发环境.