国 际 电 信 联 盟

CCITT

国 际 电 报 电 话 咨 询 委 员 会

黄 皮 书

卷 Ⅵ.8

CCITT高级语言(CHILL)

建议 Z.200

第 七 次 全 体 会 议

1980年11月10－21日 日内瓦

1984年 北京

国 际 电 信 联 盟

CCITT

国 际 电 报 电 话 咨 询 委 员 会

黄 皮 书

卷 Ⅵ.8

CCITT高级语言(CHILL)

建议 Z.200

第 七 次 全 体 会 议

1980年11月10－21日 日内瓦

1984年 北京

CCITT图 书 目 录

适用于第七次全体会议（1980年）以后

黄 皮 书

第 I 卷 全会的记录和报告

意见和决议

建议：CCITT的组织机构和工作程序（A系列）；措词的含义（B系列）；综合电信统计（C系列）。

研究组的名单和要研究的课题

第 II 卷

II·1分册 一般收费原则——国际电信业务的收费和计算，D系列建议（第III研究组）

II·2分册 国际电话业务——操作，建议E.100-E.232（第II研究组）

II·3分册 国际电话业务——网路管理——话务工程建议E.401-E543（第II研究组）

II·4分册 电报和信息通信业务操作，F系列建议（第I研究组）

第 III 卷

III·1分册 国际电话接续和电路的一般特性，建议G.101-G.171（第XV、XVI研究组，CMBD）

III·2分册 国际模拟载波系统，传输媒介——特性，建议G.211-G.651（第XV研究组，CMBD）

III·3分册 数字网路——传输系统和复接设备，建议G.701-G.941（第XVIII研究组）

III·4分册 非电话信号线路传输，声音节目和信号传输，H和J系列建议（第XV研究组）

第 IV 卷

IV·1分册 维护：一般原则、国际载波系统、国际电话电路，建议M.10-M.761（第IV研究组）

IV·2分册 维护：国际话频电报和传真、国际出租电路，建议M.800-M.1235（第IV研究组）

IV·3分册 维护：国际声音节目和电视传输电路，N系列建议（第IV研究组）

IV·4分册 测量设备技术规程，O系列建议（第IV研究组）

第 V 卷 电话传输质量，P系列建议（第XII研究组）

第 VI 卷

VI·1分册 电话交换和信号的一般建议，海上业务的接口，建议Q.1-Q.118 bis（第XI研究组）

VI·2分册 四号和五号信号系列技术规程，建议Q.120-Q.180（第XI研究组）

VI·3分册 六号信号系统技术规程，建议Q.251-Q.300（第XI研究组）

VI·4分册 R1和R2信号系统技术规程，建议Q.310-Q.490（第XI研究组）

VI·5分册 国内国际应用的数字转接局，信号系统的交互工作，建议Q.501-Q.685（第XI研究组）

VI·6分册 七号信号系统技术规程，建议Q.701-Q.741（第XI研究组）

VI·7分册 功能规格和描述语言（SDL），人机语言（MML），建议Z.101-Z.104和Z.311-Z.341（第XI研究组）

VI·8分册 CCITT高级语言（CHILL），建议Z.200（第XI研究组）

第 VII 卷

VII·1分册 电报传输和交换，R和U系列建议（第IX研究组）

VII·2分册 电报和信息通信业务终端设备，S和T系列建议（第VIII研究组）

第 VIII 卷

VIII·1分册 电话网上的数据通信，V系列建议（第XVII研究组）

VIII·2分册 数据通信网：服务和设施、终端设备和接口，建议X.1-X.29（第VII研究组）

VIII·3分册 数据通信网：传输、信号和交换；网路问题；维护；管理部门的安排，建议X.40-X.180（第VII研究组）

第 IX 卷 干扰的防护，K系列建议（第V研究组）；电缆护套和杆路的防护，L建议（第VI研究组）

第 X 卷

X·1分册 术语和定义

X·2分册 黄皮书索引

建议Z.200

CCITT高级语言（CHILL）
（日内瓦，1980）

黄皮书 卷Ⅵ·8 目 录

1.0 导言 …… 1
1.1 概述 …… 1
1.2 语言简况 …… 1
1.3 模式和类 …… 2
1.4 地点及其访问 …… 2
1.5 值及其操作 …… 3
1.6 动作 …… 3
1.7 程序结构 …… 3
1.8 并发执行 …… 4
1.9 一般语义性质 …… 4
1.10 异常处理 …… 4
1.11 实现任选 …… 4

2.0 预备知识 …… 6
2.1 元语言 …… 6
2.1.1 上下文无关语法的描述 …… 6
2.1.2 语义描述 …… 6
2.1.3 例子 …… 7
2.1.4 元语言的约束规则 …… 7
2.2 词汇表 …… 7
2.3 空格的使用 …… 7
2.4 注释 …… 7
2.5 格式作用符 …… 8
2.6 编译命令 …… 8

3.0 模式和类 …… 9
3.1 概述 …… 9
3.1.1 模式 …… 9
3.1.2 类 …… 9
3.1.3 模式和类的性质以及它们之间的关系 …… 9
3.2 模式定义 …… 10
3.2.1 概述 …… 10
3.2.2 异名模式定义 …… 11
3.2.3 新模式定义 …… 11
3.3 模式分类 …… 11
3.4 离散模式 …… 12
3.4.1 概述 …… 12
3.4.2 整数模式 …… 12
3.4.3 布尔模式 …… 12
3.4.4 字符模式 …… 13
3.4.5 集合模式 …… 13

3.4.6 区段模式 ····· 14
3.5 幂集模式 ····· 15
3.6 关联模式 ····· 15
3.6.1 概述 ····· 15
3.6.2 约束关联模式 ····· 15
3.6.3 自由关联模式 ····· 16
3.6.4 行模式 ····· 16
3.7 过程模式 ····· 16
3.8 样品模式 ····· 17
3.9 同步模式 ····· 17
3.9.1 概述 ····· 17
3.9.2 事件模式 ····· 18
3.9.3 缓冲区模式 ····· 18
3.10 组合模式 ····· 18
3.10.1 概述 ····· 18
3.10.2 串模式 ····· 19
3.10.3 数组模式 ····· 19
3.10.4 结构模式 ····· 20
3.10.5 层次结构表示法 ····· 23
3.10.6 数组模式和结构模式的布局描述 ····· 25
3.11 动态模式 ····· 28
3.11.1 概述 ····· 28
3.11.2 动态串模式 ····· 28
3.11.3 动态数组模式 ····· 28
3.11.4 动态参数化结构模式 ····· 28

4.0 地点及其访问 ····· 29
4.1 说明 ····· 29
4.1.1 概述 ····· 29
4.1.2 地点说明 ····· 29
4.1.3 地点等同说明 ····· 30
4.1.4 有基说明 ····· 30
4.2 地点 ····· 31
4.2.1 概述 ····· 31
4.2.2 访问名字 ····· 31
4.2.3 非关联化约束关联 ····· 32
4.2.4 非关联化自由关联 ····· 32
4.2.5 串元素 ····· 33
4.2.6 子串 ····· 33
4.2.7 数组元素 ····· 33
4.2.8 子数组 ····· 34
4.2.9 结构场 ····· 35
4.2.10 地点过程调用 ····· 35
4.2.11 地点内部子程序调用 ····· 35
4.2.12 地点转换 ····· 35
4.2.13 串切片 ····· 36
4.2.14 数组切片 ····· 36
4.2.15 非关联化行 ····· 37

5.0 值及其操作 …… 38
5.1 异名定义 …… 38
5.2 原值 …… 38
5.2.1 概述 …… 38
5.2.2 地点内容 …… 39
5.2.3 值名字 …… 39
5.2.4 直接量 …… 40
5.2.4.1 概述 …… 40
5.2.4.2 整数直接量 …… 40
5.2.4.3 布尔直接量 …… 41
5.2.4.4 集合直接量 …… 41
5.2.4.5 空直接量 …… 41
5.2.4.6 过程直接量 …… 41
5.2.4.7 字符串直接量 …… 41
5.2.4.8 字位串直接量 …… 42
5.2.5 多元组 …… 43
5.2.6 值串元素 …… 45
5.2.7 值子串 …… 45
5.2.8 值串切片 …… 46
5.2.9 值数组元素 …… 46
5.2.10 值子数组 …… 47
5.2.11 值数组切片 …… 47
5.2.12 值结构场 …… 48
5.2.13 被关联地点 …… 48
5.2.14 表达式转换 …… 48
5.2.15 值过程调用 …… 49
5.2.16 值内部子程序调用 …… 49
5.2.17 开动表达式 …… 51
5.2.18 接收表达式 …… 51
5.2.19 零目运算符 …… 52
5.3 值和表达式 …… 52
5.3.1 概述 …… 52
5.3.2 表达式 …… 52
5.3.3 运算数 1 …… 53
5.3.4 运算数 2 …… 53
5.3.5 运算数 3 …… 54
5.3.6 运算数 4 …… 55
5.3.7 运算数 5 …… 56
5.3.8 运算数 6 …… 57

6.0 动作 …… 58
6.1 概述 …… 58
6.2 赋值动作 …… 58
6.3 条件动作 …… 59
6.4 情况动作 …… 60
6.5 循环动作 …… 61
6.5.1 概述 …… 61
6.5.2 步长型控制 …… 62

6.5.3 当型控制 …… 64
6.5.4 结构型部分 …… 64
6.6 出口动作 …… 65
6.7 调用动作 …… 65
6.8 结果和返回动作 …… 66
6.9 转向动作 …… 67
6.10 断言动作 …… 67
6.11 空动作 …… 67
6.12 引发动作 …… 67
6.13 开动动作 …… 68
6.14 停止动作 …… 68
6.15 继续动作 …… 68
6.16 延迟动作 …… 68
6.17 延迟情况动作 …… 69
6.18 发送动作 …… 69
6.18.1 概述 …… 69
6.18.2 发送信号动作 …… 69
6.18.3 发送缓冲区动作 …… 70
6.19 接收情况动作 …… 70
6.19.1 概述 …… 70
6.19.2 接收信号情况动作 …… 71
6.19.3 接收缓冲区情况动作 …… 72

7.0 程序结构 …… 73
7.1 概述 …… 73
7.2 范围和嵌套 …… 73
7.3 分程序 …… 75
7.4 过程定义 …… 75
7.5 进程定义 …… 77
7.6 模块 …… 78
7.7 区域 …… 78
7.8 程序 …… 78
7.9 存储分配和生存期 …… 79

8.0 并发执行 …… 80
8.1 进程和它们的定义 …… 80
8.2 互斥和区域 …… 80
8.2.1 概述 …… 80
8.2.2 区域性 …… 81
8.3 进程的延迟 …… 82
8.4 进程的重新活化 …… 82
8.5 信号定义语句 …… 83

9.0 一般语义性质 …… 84
9.1 模式检验 …… 84
9.1.1 模式和类的性质 …… 84
9.1.1.1 新鲜性 …… 84
9.1.1.2 只读模式 …… 84

9.1.1.3 只读性质 …… 84
9.1.1.4 关联性质 …… 85
9.1.1.5 带标签的参数化性质 …… 85
9.1.1.6 同步性质 …… 85
9.1.1.7 根模式 …… 85
9.1.1.8 结果类 …… 86
9.1.2 模式和类的关系 …… 86
9.1.2.1 “被定义”关系 …… 86
9.1.2.2 模式的等价关系 …… 86
9.1.2.3 “读相容”关系 …… 89
9.1.2.4 “可限制为”关系 …… 89
9.1.2.5 模式和类的相容性 …… 89
9.1.2.6 类的相容性 …… 90
9.1.3 情况选择 …… 90
9.1.4 语义范畴的定义和概况 …… 91
9.1.4.1 名字 …… 92
9.1.4.2 地点 …… 93
9.1.4.3 表达式 …… 93
9.1.4.4 其它语义范畴 …… 93
9.2 可见性和名字的约束 …… 93
9.2.1 概述 …… 93
9.2.2 可见性和名字的建立 …… 94
9.2.3 隐含名字 …… 95
9.2.4 范围内的可见性 …… 95
9.2.5 可见性和程序块 …… 96
9.2.6 可见性和模片 …… 96
9.2.6.1 概述 …… 96
9.2.6.2 开放语句 …… 96
9.2.6.3 移入语句 …… 97
9.2.7 场名字的可见性 …… 97
9.2.8 名字的约束 …… 98

10.0 异常处理 …… 99
10.1 概述 …… 99
10.2 处理程序 …… 99
10.3 处理程序的识别 …… 99

11.0 实现任选 …… 101
11.1 实现定义的内部子程序 …… 101
11.2 实现定义的整数模式 …… 101
11.3 实现定义的寄存器名字 …… 101
11.4 实现定义的进程名字和异常名字 …… 101
11.5 实现定义的处理程序 …… 101
11.6 语法任选 …… 102

附录A：CHILL 程序的字符集
A.1 CCITT 第5号字母表国际参考版 …… 103
A.2 表示CHILL程序的最小字符集 …… 104

附录B：专用符号 …… 105

附录C：CHILL专用名字 …… 106
C.1 保留名字 …… 106
C.2 预定义名字 …… 107
C.3 CIHLL异常名字 …… 107
C.4 CHILL命令 …… 107

附录D：程序例子 …… 108

附录E：语法图 …… 131

附录F：产生式规则索引 …… 141

附录G：索引 …… 148

1.0 导言

本建议书定义了CCITT高级程序设计语言CHILL。CHILL是CCITT High Level Language 的缩写。

CHILL的另一种以严格数学形式表示的定义，发表在CCITT手册中。还有一本《CHILL概述》是CHILL语言的一个引言。

1.1 概述

CHILL主要是为存储程序控制电话交换机的编程而设计的。但它具有极大的通用性，可以应用于其它方面（如报文交换、分组交换、模拟等）。

在设计CHILL时考虑了下列要求（参见1977～1980研究阶段文件：课题8/XI）：

- 能进行大量的编译时检验以提高可靠性；
- 能生成高效率的目标代码；
- 使用灵活、功能强，能覆盖所要求的应用范围，并能发挥各种硬件的作用；
- 利于模块化和结构化的程序开发；
- 容易学习和使用。

CHILL隐含有一个程序开发环境。这个环境除其它功能项外，还包括：分块编译、输入/输出和调试工具。这些功能项在本建议书中没有给出定义。

对于SPC电话交换中已经使用或建议使用的机器类型，CHILL程序能够以独立于机器的形式书写。

CHILL并不打算为上面提到的每个应用领域提供特殊的语言结构，但是有通用的基础结构，为具体应用提供一些合于使用的可能手段。

作为一种语言，CHILL是独立于机器的。但是在具体实现时，可以包含一些实现定义的语言成分。包含这类成分的程序一般是不可移植的。

设计CHILL的前提是要能把它的源码编译成目标代码。设计时并没有考虑到一次扫描编译方法的特殊需要，也没有考虑到使编译程序的体积达到最小。

为了确保安全而又不致于使效率的损失大到不可接受的程度，许多检验可以静态地进行，只有少数语言规则要在运行时检验。违反这些规则将引起运行时的异常。但是，除非程序员自己规定了异常处理程序，这些异常时的运行检验是任选的。

1.2 语言简况

CHILL程序基本上由三部分组成：

- 数据对象的描述；
- 作用于数据对象的动作描述；
- 程序结构的描述。

数据对象由数据语句（说明语句和定义语句）描述，动作由动作语句描述，程序结构由程序结构语句确定。

CHILL可以处理的数据对象是值和存放值的地点。动作定义作用于数据对象的操作，以及把值存入地点或从地点中检索值的次序。程序结构确定数据对象的生存期和可见性。

对于在给定上下文中数据对象的使用，CHILL提供了充分的静态检验。

以下各节概述了CHILL中出现的各种概念。每一节介绍与该节同名的一章的内容，概念的详细描述在有关章中说明。

1.3 模式和类

CHILL可处理的数据对象是**值**和可以存放值的**地点**。

每个地点有一个**模式**。地点的模式定义了可以存放于该地点的值的集合，也定义了地点具有的其它性质以及可能存放的值的性质（注意，单靠地点的模式不能确定该地点的所有性质）。地点的性质是：大小、内部结构、只读性、可关联性等。值的性质是：内部表示、次序、可应用的操作等。

每个值有一个**类**。值的类确定可以存放值的地点的各模式。

CHILL提供下列诸模式范畴：

离散模式 整数、字符、布尔、集合（符号）模式以及这些模式的区段；

幂集模式 某些离散模式的元素组成的集合；

关联模式 约束关联、自由关联以及用作对地点的关联的行；

组合模式 字符串、数组和结构模式；

过程模式 过程看作可处理的数据对象；

样品模式 对进程的标识；

同步模式 用于进程同步和通信的事件模式和缓冲区模式。

CHILL提供一组标准模式的标志。用**模式定义**可以引进由程序定义的模式。这类语言结构有一个所谓的**动态模式**。动态模式是这样一种模式，它的某些性质只能动态地确定。动态模式都是带有运行时参数的参数化模式。非动态模式称为**静态模式**。在CHILL程序中显式标志的模式总是静态的。

在CHILL中，动态模式和类都没有标志。它们只出现在元语言中，用来描述静态和动态的上下文条件。

1.4 地点及其访问

地点是（抽象的）位置，可以把值存进去，也可以从其中取出值来。为了存放或得到一个值，就必须访问一个地点。

说明语句定义用来访问一个地点的名字。它们是：

1. 地点说明；
2. 地点等同说明；
3. 有基说明。

第一种说明开辟地点并建立访问新开辟地点的名字。后两种说明对已开辟过的地点建立访问它们的新名字。

新地点除了通过地点说明建立外，还可以通过内部子程序**GETSTACK**建立，该子程序将为新开辟的地点提供一个关联值（见下面）。

一个地点可以是**可关联的**。这意味着存在一个该地点的**关联值**。对可关联的地点施行**关联操作**，即可得到关联值。对关联值施行**非关联化**，即可得到所关联的地点。CHILL要求某些地点总是可关联的，但其它地点是否可关联要由实现来确定。可关联性应该是地点的一个可静态确定的性质。

地点可以是**只读**的，这意味着只能通过访问它得到一个值，而不能存放一个新的值进去（除了赋初值）。

地点可以是**组合的**，这意味着拥有可以分别访问的子地点。子地点不一定是可关联的。如果一个地点至少含有一个**只读**的子地点，则称该地点具有**只读性**。提供子地点（或子值）的访问方法分别为：对于字符串和数组是**求子串**、**求元素**和**求片**，对于结构是**选择**。

每个地点有一个模式。若模式是动态的，该地点称为动态模式地点。（注意，动态一词只是对模式而言。尽管运行时地点发生变化，并不意味着它是动态的。地点是动态的，只是说它的性质不能完全静态地确定。）

地点的下列性质虽然可以静态地确定，但不是模式的一部分：

可关联性：对某个地点是否存在一个关联值；

存储类：地点是否被静态地分配；

区域性：该地点是否在某个区域内被说明。

1.5 值及其操作

值是在其上面定义了特定操作的基本对象。一个值是一个(CHILL)**有定义的值**,或一个(在CHILL意义上)**未定义的值**。在指明的上下文中使用未定义的值,将导致出现(在CHILL意义上)无定义情况,此时程序被认为是错误的。

CHILL允许在需要值的上下文处使用地点。在这种情况下,对地点作访问就可取得其中包含的值。

每个值有一个**类**。除了类以外还有模式的值称为**强值**。在这种情况下,这个值总是该模式所定义的值之一。类用于相容性检验,而模式用于描述值的性质。某些上下文要求这些性质是已知的,这时就要求强值。

一个值可以是**直接量**,在此情况下它标志一个与实现无关的,在编译时已知的离散值。值可以是**常量**,在此情况下它永远提供同一个值,即它只要计算一次。**直接量**和**常量**值都假定是在运行之前就计算好了的,它们不可能产生运行时异常。值可以是**区域性的**,在此情况下它能以某种方式与区域地点相关联。值可以是**组合的**,即包含子值。

异名定义语句建立新名字以标志常量值。

1.6 动作

动作构成CHILL程序的算法部分。

赋值动作把(计算好的)值存入一个或多个地点中。**过程调用**调用一个过程,**内部子程序调用**调用一个内部子程序(内部子程序是这样的一个过程,其定义不用CHILL书写,它有较为通用的参数和结果返回机制)。为了从过程调用返回及/或建立过程调用的结果,要使用**结果**和**返回动作**。

为了控制顺序动作流,CHILL提供了下列控制动作流:

条件动作 用于两叉分支;

情况动作 用于多叉分支。分支的选择可以多个值为基础,类似于判定表;

循环动作 用于叠代或加括号;

出口动作 用于以结构化方式中离开加括号内的动作;

引发动作 用于引发一个特定的异常;

转向动作 用于无条件地转到加了标号的程序点。

动作和数据语句可以组合起来以构成一个模块或分程序,后者构成一个(复合)动作。

为了控制并发动作流,CHILL提供了**开动**、**停止**、**延迟**、**继续**、**发送**、**延迟情况**和**接收情况**等动作,或对**接收表达式**进行计值。

1.7 程序结构

程序结构语句有**分程序**、**模块**、**过程**、**进程**和**区域**。程序结构语句提供控制地点的**生存期**和名字的**可见性**的手段。

地点的生存期是指地点在程序内存在的时间。地点可以(在地点说明中)**显式地说明**,或(通过调用内部子程序**GETSTACK**)**生成**,也可以作为某些语言成分的使用结果而**隐式地说明**或**生成**。

如果一个名字可在程序的某一点上使用,则称此名字在该点是**可见的**。名字的**作用域**包含所有使它可见的点,即该名字在那里可以识别它所标志的对象。

分程序既确定名字的可见性,又确定地点的生存期。

模块用于限制名字的可见性,以防止它们被非法使用。利用**可见性语句**可以控制名字在程序的不同部分的可见性。

过程是一个(可能是参数化的)子程序,它可在程序的不同位置被召用(调用),可以送返一个值(**值过程**)或一个地点(**地点过程**),或不提供结果。在后一种情况下,该过程只能在过程调用动作中被调用。

进程和**区域**提供用于实现并发执行结构的手段。

完整的CHILL**程序**是一系列模块或区域,并认为由一个(虚拟的)进程定义所包含。这个最外层的进程

被系统开动，程序就在系统的控制下执行。

1.8 并发执行

CHILL允许各程序单位**并发执行**。**进程**是并发执行的单位。**开动动作**创建它所指定的**进程定义**的一个新**进程**。此进程即被认为是与开动它的进程并发执行。CHILL允许具有相同或不同定义的一个或多个进程在同一时刻活动。执行**停止动作**导致一个进程终止执行。

每个进程总是处于**活动**或**延迟**这两种**状态**之一。从活动状态转为延迟状态称为**进程延迟**，从延迟状态转为活动状态称为进程**重新活化**。对事件执行延迟动作，对缓冲区或信号执行接收动作，或对缓冲区执行发送动作，都能使执行这些动作的进程被延迟。对事件执行继续动作，对缓冲区或信号执行发送动作，或对缓冲区执行接收动作，都能使被延迟的进程重新活动起来。

缓冲区和**事件**都是地点，它们的使用是有限制的。**发送、接收和接收情况**等操作是定义在缓冲区上的，**延迟、延迟情况和继续**操作等是定义在事件上的。缓冲区是进程之间同步和传递信息的手段。事件只用于同步。**信号**定义在**信号定义语句**中。它们标志进程之间传递的组合和非组合的值系列、**发送动作**和**接收情况动作**用于值系列的通信和同步。

区域是一种特殊的模块，它用于实现对多个进程共享的数据结构的互斥访问。

1.9 一般语义性质

CHILL的（非上下文无关的）语义条件是模式和类的相容性条件（**模式检验**）及可见性条件（**作用域检验**）。模式检验规则确定名字的使用方式，而作用域检验规则确定名字可以在何处使用。

模式检验规则用模式之间、类之间以及模式和类之间的相容性要求来描述。模式和类之间以及类和类之间的相容性要求用模式之间的等价关系来定义。如果涉及动态模式，则模式检验部分是动态的。

作用域规则通过程序结构和显式的可见性语句确定名字的可见性。显式的可见性语句确定所提到的名字的作用域，也确定这些名字可能**隐含的名字**的作用域。

在程序中引进的名字有一个被定义或说明的地方。这个地方称为该名字的**定义性出现**。而名字被使用的地方称为该名字的**应用性出现**。**名字约束**规则把名字的唯一的定义性出现和它的每一个应用性出现结合起来。

1.10 异常处理

CHILL的动态语义条件是那些（非上下文无关的）一般不能静态地确定的条件。(在运行时是否产生代码以检验动态条件，要留待实现来决定。)违反动态语义条件将引起运行时**异常**。

异常也可以由执行**引发动作**而被引发，还可以由于执行**断言动作**而有条件地被引发。如果该异常是可以指明处理程序的(即它有一个名字)，并且指明了处理程序则当该异常在给定的程序点上出现时，控制就转向相应的异常处理程序。至于是否已经在某一点上为某一异常指明了异常处理程序，是可以静态地确定的。如果没有显式说明的异常处理程序，控制可以转向一个由实现定义的异常处理程序。

大多数异常都有名字。这个名字是一个CHILL定义的异常名字，即实现定义的异常名字，或程序定义的异常名字。注意，当为CHILL定义的异常名字指明一个处理程序时，应该检验有关的动态条件。

1.11 实现任选

CHILL允许有**实现定义的整数模式、实现定义的内部子程序、实现定义的进程定义和实现定义的异常处理程序**。

实现定义的整数模式必须用实现定义的模式名字来标志。这个名字被认为是没有在CHILL语句中说明过的新模式定义语句所定义的。在CHILL语法和语义规则中，允许把现有的CHILL定义的算术操作推广到实现定义的整数模式中去。实现定义的整数模式的例子是**长整数**和**短整数**。

内部子程序是一个未用CHILL语句定义过的过程，它比CHILL过程有更一般的参数传递和结果返回规则。

内部进程名字是一个不用CHILL语句定义的进程名字。CHILL进程可以和实现定义的进程协同运行或开动这些进程。

由实现定义的异常处理程序是附属于虚拟的最外层的进程定义的处理程序。如果在出现异常后转到该处理程序处，则此时应采取什么动作将由实现来决定。

2.0 预备知识

2.1 元语言

CHILL的描述由两部分组成：

- 上下文无关语法的描述；
- 语义条件的描述。

2.1.1 上下文无关语法的描述

上下文无关语法用扩充的巴科斯——瑙尔范式描述。语法范畴是用一个或多个异体汉字词组(或英文词组)组成并用尖括号将其括起来（〈和〉），称之为非终结符号。对于每个非终结符号，都有相应的语法部分给出其产生规则。非终结符号的产生规则是这样组成的：在符号::=的左边是这个非终结符号，在符号::=的右边是由非终结产生式和/或终结产生式组成的一个或多个结构。这些结构之间用竖线(|)分开，它们代表这个非终结符号的可供选用的产生式。

有时，非终结符号包括一个下划线的部分。这个下划线的部分并不构成上下文无关描述的一部分，而是定义了一个语义子范畴（参见2.1.2节）。

语法元素可以用花括号（{和}）括在一起。花括号组的重复用星号(*)或加号（+）表示。星号表示此组合是任选的，也可以重复任意多次；加号表示此组合必须出现，并可重复任意多次。例如,[A]*表示由任意多个A，包括零个A，组成的序列，而{A}$^{+}$表示由至少一个A组成的任意序列。如果语法元素用方括号([和])括起来，则此组合是任选的。

严格语法和**导出语法**是有区别的。严格语法的语义条件是直接给出的。导出语法可以看作是严格语法的扩充。导出语法的语义用相应的严格语法间接解释。

注意，在本文件中，上下文无关语法的描述方式是根据语义描述的需要选定的，并不是为了适应任何特定的语法分解算法的需要（例如，为了表达清楚，引进了一些上下文无关的二义性）。

2.1.2 语义描述

每个语法范畴（非终结符号）的语义描述在**语义**、**静态性质**、**动态性质**、**静态条件**和**动态条件**等部分给出。

语义部分描述由语法范畴表示的概念（即它们的意义和行为）。

静态性质部分定义该语法范畴的可静态地确定的语义性质。在该语法范畴在相应的章节使用时，这些性质可用来描述其静态的和/或动态的条件。

动态性质部分视需要而设立，定义此语法范畴的只能是动态确定的性质。

静态条件部分描述上下文有关的并可静态地检验的条件，当用到该语法范畴时，这些条件必须被满足。在语法中，某些静态条件是通过非终结符号中的下划线部分来表达的(参见2.1.1节)。这种使用方式要求该非终结符是一个特定的语义子范畴。例如，〈布尔表达式〉在上下文无关的意义下恒等于〈表达式〉，但是在语义上要求该表达式属于布尔类。下划线部分有时在文本中用作形容词以限定非终结符。例如,“此表达式是**常量**”这句话相当于“此表达式是常量表达式”。

动态条件部分描述在程序执行时应该满足的上下文有关条件。在某些情况下，当且仅当不涉及动态模式时，条件是静态的。在这种情况下，该条件在**静态条件**部分被提到，而在**动态条件**部分被引用。

在语义描述中，非终结符号用不带尖括号的异体写出，以此来标示语法对象。

2.1.3 例子

对大多数语法章节来说，都有一个**例子**部分，其中给出所定义的语法范畴的一个或几个例子。这些例子是从附录D中作为例题的一组程序中抽出来的。索引指出每个例子是通过哪一个语法规则产生的，以及它是从哪一个例题中抽取出来的。

例如，6.20 (d+ 5)/5 (1.2) 表示例子是一个终结字符串 (d+5)/5，它由相应语法章节的规则 (1.2) 产生，取自程序例题 6 的第20行。

2.1.4 元语言的约束规则

有时语义描述提到CHILL**专用的**名字(见附录C)。这些专用名字在使用中总是带有它的CHILL意义，因此不受实际CHILL程序约束规则的影响。

2.2 词汇表

程序用CCITT第 5 号字母表，建议书V .3（见附录A 1）表示。但每个CHILL程序都可以用一个最小字符集表示，它是CCITT第 5 号字母表基本码的一个子集（见附录A 2）。

CHILL的词法元素是：

- 特殊符号。
- 名字。
- 直接量。

特殊符号在附录B中给出。

名字根据下列语法构成：

语法：

〈名字〉::= (1)

〈字母〉{〈字母〉|〈数字〉|_}* (1.1)

下划线符号{_}是名字的一部分。即，名字LIFE-TIME不同于名字LIFETIME。在有小写字母可供使用的情况下，这些字母也可用于名字中。小写字母和大写字母是不同的，例如Status和status是两个不同的名字。

本语言有一些具有预先确定意义的**专用**名字，见附录C。它们中有些是**保留**的，即，除非用释放命令显式地释放，它们是不能派作其它用途的。

在大、小写字母都可以使用的情况下，专用名字可以或者全用大写字母表示，或者全用小写字母表示。保留名字只在选定的表示形式中保留（例如，若选定小写形式，则row是保留的，而ROW则不保留）。

2.3 空格的使用

空格可用来作为程序中词法元素的分隔。词法元素在第一个不能成为词法元素一部分的字符处结束。例如，IFBTHEN将被认为是一个名字而不是一个动作IF B THEN的开始。//*将被看作是连接符{//}后面跟一个星号{*}，而不是一个除号{/}后面跟一个注释起始括号{/*}。连续的空格和单个空格具有相同的分隔效果。

2.4 注释

语法：

〈注释〉::= (1)

/* 〈字符串〉*/ (1.1)

〈字符串〉::= (2)

{〈字符〉}* (2.1)

语义 注释向阅读者传递有关程序的信息。它对程序的语义没有影响。

静态性质 只要是允许使用空格作分隔符的地方，都可以插入注释。

静态条件 字符串不得包含特殊序列{*/}。

例子：

4.1 /*from collected algorithm from CACM nr. 93 */ (1.1)

2.5 格式作用符

在CHILL上下文无关语法描述中没有提到CCITT第5号字母表（从位置FE。到FE_5）的格式作用符BS（退一格），CR（回车），FF（换页），HT（水平制表），LF（换行）和VT（垂直制表）。但是，在实现时可以在CHILL程序中使用这些格式控制符。当使用时，它们有和空格一样的分隔作用，但不能在词法元素内部使用。

2.6 编译命令

语法：

〈命令子句〉::= (1)

〈〉〈命令〉{,〈命令〉}*〔〈〉〕 (1.1)

〈命令〉::= (2)

〈CHILL命令〉 (2.1)

|〈实现命令〉 (2.2)

〈CHILL命令〉::= (3)

〈释放命令〉 (3.1)

〈释放命令〉::= (4)

FREE(〈保留名字表〉) (4.1)

〈名字表〉::= (5)

〈名字〉{,〈名字〉}* (5.1)

语义： 命令子句向编译程序提供信息。除释放命令外，该信息由实现定义的格式来说明。

实现命令不得影响程序的语义。即，当且仅当不带实现命令的程序在CHILL意义上是正确时，加上实现命令的同一程序也是正确的。

释放命令用于编译单元，它将释放在保留名字表中指明的保留名字，使它们可以在该编译单元中被重新定义。

静态性质 命令子句可以插入所有允许空格出现的地方。它与空格有同样的分隔作用。在命令子句中使用的名字服从实现定义的名字约束规则，该规则不影响CHILL的名字约束规则（参见9.2.8节）。

静态条件 任选的命令结束符号{〈〉}只当它正好位于一个分号之前时方可被省去（即命令子句以第一个〈〉或分号结束。但分号不属于命令子句。因此，命令既不能包含〈〉符号，也不能包含分号，除非它位于圆括号中间，参见下面）。若圆括号在实现命令中出现，它们必须正确地配对，并且，如果一个分号或命令结束符号出现于圆括号之内，则它们并不终止该命令。

例子：

15.1 〈〉FREE(STEP) (1.1)

15.1 FREE(STEP) (4.1)

3.0 模式和类

3.1 概述

每个地点有一个模式，每个值有一个类。地点的模式定义该地点可以存放的值的集合、该地点的访问方式，以及允许在值上进行的操作。值的类是用以确定可以存放该值的地点的模式的一种手段。某些值是强的。每个强值有一个类和一个模式。这个模式总是与值的类是相容的，而该值是该模式定义的值中的一个。如果某个值所在的上下文需要模式信息，则该值应是强值。

3.1.1 模式

CHILL有**静态模式**(即模式的所有性质皆可静态地确定）和**动态模式**（即模式的某些性质只能在运行时确定）。动态模式都是具有运行时参数的参数化模式。

在程序中，静态模式均以语法范畴模式的终结产生式来表示。

动态模式在CHILL中没有标志。但是，为了便于描述，本文件中引进了**虚拟标志**，用以标志动态模式。这些虚拟标志的前面有一个&记号。例如，&VM(i)标志具有运行时参数i的参数化动态模式。

此外，某些地方也为一些静态模式引进了虚拟标志。这些静态模式未在程序中显式地标志，或不能在程序中显式地标志，它们是被某些语言成分虚拟地引入的。这些模式也由前面加了&记号的虚拟标志来标志。

3.1.2 类

类在CHILL中没有标志。

类有下列几种，CHILL程序中的每个值都属于这几种类之一：

- 对任何模式M，存在有**M值类**。属于这样一个类的所有值，并且只有这些值是强值。值的模式是M。
- 对任何一个新鲜性为*nil*的模式M（见9.1.1.1节）都有一个**M导出类**。
- 对任何模式M，有一个**M关联类**。
- **空类**。
- **完全类**。

最后两类是常类，即它们不依赖于模式M。当且仅当一个类是M值类或M导出类，并且M是一个动态模式时，这个类称为是**动态的**。

3.1.3 模式和类的性质以及它们之间的关系

模式和类的所有性质和它们之间的基本关系在第9章中定义。下面是这些性质和关系的概述：

1. 每个模式M有**新鲜性**。
2. 模式M可以是**只读的**。
3. 模式M可以有**只读性**。
4. 模式M可以有**关联性**。
5. 模式M可以有**带标签的参数化性质**。
6. 模式M可以有**同步性**。
7. 模式M可以**被模式N所定义**。
8. 模式M可以和一个模式N**读相容**（非对称的）。
9. 模式M可以和类C**相容**（在此情况下C称为与M相容）。
10. 类C可以有**根模式**。
11. 类C可以与类D**相容**（对称）。

12. 给定一组相容的类，存在一个结果类。

每个模式的特殊性质在适当的段中给出。如果当一个性质对某个特定模式成立时，它对由该模式定义的所有模式名字都成立，则此性质称为继承的。因此，不为模式名字显式地定义它们的继承性质。任何性质如果对模式成立，则当该模式前加有关键字READ时它亦成立（在涉及到只读性质的某些情况下例外，这些例外情况将特别指出）。因此，前面加了READ的模式的性质将不显式地定义。

3.2 模式定义

3.2.1 概述

语法：

〈模式定义〉::= (1)

〈名字表〉=〈定义模式〉 (1.1)

〈定义模式〉::= (2)

〈模式〉 (2.1)

导出语法： 如果名字表中有多于一个名字，则此模式定义是由多个模式定义导出的，其中每个相应于一个名字，它们由逗号隔开，并具有相同的定义模式。如

NEWMODE DOLLAR,POUND=INT；是由NEWMODE DOLLAR=INT,POUND=INT；导出的。

语义： 模式定义把一个或多个名字定义为模式名字，即标志模式的名字。模式定义在新模式定义语句和异名模式定义语句中出现。新模式和异名模式的区别在于模式等价算法的处理不同(见9.1节)。根据定义，定义模式的所有继承性质传给了被定义的模式名字。模式定义可以是（相互）递归的。

静态性质： 每个模式名字或者是语言定义的模式名字INT,BOOL,CHAR,PTR,INSTANCE,EVENT中的一个，或者是在模式定义中定义的名字。

每个不是由语言定义的模式名字都有一个唯一的定义模式，它就是在定义该模式名字的模式定义中出现的定义模式。

递归定义集合是模式定义组或异名定义组（见5.1节）。其中，每个模式定义中的定义模式，或每个异名定义中的常量值或模式，都是，或者直接包含，模式名字、或异名名字，或该集合中某定义所定义的集合元素名字。

递归模式定义的集合是仅有模式定义的一个递归定义的集合。（任何递归定义的集合必须是递归模式定义的集合；见5.1节）。

若某模式是一个模式名字，或包含一个模式名字，而此模式名字又是在一个递归模式定义的集合中定义的，则称此模式标志了一个递归模式。递归模式定义的集合中的一条路径是一系列的模式名字，其中每个名字都带一个下标作为标记，并且：

- 路径中的所有名字都有不同的定义；
- 每个名字的后继是该名字的定义模式，或者直接出现在该名字的定义模式中（最后一个名字的后继是第一个名字）；
- 一个名字的标记唯一地指出了该名字在它的先驱的定义模式中的位置（第一个名字的先驱是最后一个名字）。

(例子：*NEWMODE M=STRUCT (i M,n REF M)*；包含两条路径：{*Mi*}和{*Mn*})

当且仅当在一条路径中至少有一个名字在所标记的位置上包含在关联模式中，或行模式中，或过程模式中时，这条路径是安全的。这个模式应该就是该模式名字的先驱的定义模式，或包含于此定义模式中。

静态条件： 任何递归模式定义集合的所有路径都应该是安全的。(上例中的第一条路径不是安全的。)

例子：

1.12 *operand−mode=INT* (1.1)

3.3 *complex=STRUCT(re,im INT)* (1.1)

3.2.2 异名模式定义

语法：

〈异名模式定义语句〉::= (1)

SYNMODE 〈模式定义〉{,〈模式定义〉}*; (1.1)

语义： 异名模式定义语句定义一些名字，这些名字标志的模式和它们的定义模式是同义的。在异名模式定义中定义的名字的确切处理方式在9.1节中解释。

静态性质： 当且仅当一个名字在一个异名模式定义语句的模式定义中被定义时，该名字称为**异名模式**名字。当且仅当下列条件成立时，一个**异名模式**名字称为和给定的模式同义（反过来说，给定模式也称为与**异名模式**名字同义）：

- 或者给定模式是**异名模式**名字的定义模式；
- 或者**异名模式**名字的定义模式本身就是一个与给定模式同义的**异名模式**名字。

例子：

6.3 *SYNMODE month=SET (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec);* (1.1)

3.2.3 新模式定义

语法：

〈新模式定义语句〉::= (1)

NEWMODE 〈模式定义〉{,〈模式定义〉}*; (1.1)

语义： 新模式定义语句定义一些名字，这些名字用来标志与定义模式非同义的模式。新模式定义的值就是定义模式定义的值。在新模式定义语句中定义的名字的确切处理方式在9.1节中解释。

静态性质： 当且仅当一个名字是在一个新模式定义语句中的一个模式定义中被定义时，此名字称为一个**新模式**名字。

若定义模式是一个区段模式，则随着被定义的**新模式**名字，引进一个新的虚拟名字，它以&名字-父本标志，标志着**新模式**名字的**父本**模式。由这个虚拟父本模式定义的值就是该区段模式（作为定义模式）的父本模式的值。虚拟父本模式的上界和下界就是区段模式（作为定义模式）的父本模式的上界和下界。

若定义模式是一个字符串模式，则对于大于该**新模式**名字的字符串长度的任何一个i，引进新的虚拟模式&名字(i)，其中&名字表示被引进的新模式。i代表虚拟模式的串长。该**新模式**名字的继承性质**字位**串或**字符**串被传递给诸虚拟模式。

例子：

11.4 *NEWMODE Line= INT (1: 8);* (1.1)

11.10 *NEWMODE board=ARRAY(line) ARRAY(column)square;* (1.1)

3.3 模式分类

语法：

〈模式〉::= (1)

〈非组合模式〉 (1.1)

| 〈组合模式〉 (1.2)

〈非组合模式〉::= (2)

〈离散模式〉 (2.1)

| 〈幂集模式〉 (2.2)

| 〈关联模式〉 (2.3)

| 〈过程模式〉 (2.4)

| 〈样品模式〉 (2.5)
| 〈同步模式〉 (2.6)

语义： 在CHILL程序中，模式用语法范畴模式的终结产生式来标志。本章的其余部分，将定义不同模式的特殊性质。相等关系(=)和不等关系(/=)被定义在任一给定模式的值的集合中(见5.3节)。

静态性质： 模式有**大小**，它是$SIZE(M)$提供的值，其中M是虚拟的**异名模式**名字，与模式同义。

3.4 离散模式

3.4.1 概述

语法：

〈离散模式〉::= (1)
〈整数模式〉 (1.1)
| 〈布尔模式〉 (1.2)
| 〈字符模式〉 (1.3)
| 〈集合模式〉 (1.4)
| 〈区段模式〉 (1.5)

语义： 离散模式定义良序值的集和子集。所有不是区段模式的离散模式都可以是区段模式的父本模式(见3.4.6节)。所有离散模式都定义了**上界**和**下界**，它们分别标志最大和最小值。

3.4.2 整数模式

语法：

〈整数模式〉::= (1)
〔READ〕INT (1.1)
| 〔READ〕BIN (1.2)
| 〔READ〕〈整数模式名字〉 (1.3)

导出语法： BIN是INT的导出语法。

语义： 整数模式定义一组带正负号的整数值，这些值位于实现定义的界限之间，在其上定义了通常的排序运算和算术运算(见5.3.2节)。实现可以定义带不同界限的其它整数模式（例如$LONG_INT, SHORT_INT$, …），这些整数模式也可用作区段的父本模式（见11.2节）。

静态性质： 整数模式有下列继承性质：

- 整数模式的**上界**和**下界**都是直接量，分别表示由该整数模式定义的最大值和最小值。
- 整数模式的**值的个数**由实现定义。

例子：

1.4 *INT* (1.1)

3.4.3 布尔模式

语法：

〈布尔模式〉::= (1)
〔*READ*〕*BOOL* (1.1)
| 〔*READ*〕〈布尔模式名字〉 (1.2)

语义： 布尔模式定义逻辑真假值（TRUE和EALSE），以及通常的布尔运算（见5.3.2节）。TRUE大于FALSE。

静态性质： 布尔模式有下列继承性质：

- 布尔模式的**上界**是TRUE，**下界**是FALSE。

- 布尔模式定义的值的个数是 2。

例子：

5.4 *BOOL* (1.1)

3.4.4 字符模式

语法：

〈字符模式〉 ::= (1)
〔*READ*〕 *CHAR* (1.1)
| 〔*READ*〕〈字符模式名字〉 (1.2)

语义： 字符模式定义了字符值，该字符值由国际参考版本CCITT第五号字母表所描述（建议书V 3，见附录A 1）。这个字母表也规定了字符的次序。

静态性质： 字符模式有下列继承性质：

- 字符模式的上界和下界是长度为1的字符串直接量，分别标志由CHAR定义的最大值和最小值。
- 字符模式定义的值的个数是128。

例子：

8.4 *CHAR* (1.1)

3.4.5 集合模式

语法：

〈集合模式〉 ::= (1)
〔*READ*〕 *SET* (〈集合表〉) (1.1)
| 〔*READ*〕〈集合模式名字〉 (1.2)
〈集合表〉 ::= (2)
〈编号集合表〉 (2.1)
| 〈未编号集合表〉 (2.2)
〈编号集合表〉 ::= (3)
〈编号集合元素〉{，〈编号集合元素〉}* (3.1)
〈编号集合元素〉 ::= (4)
〈名字〉 = 〈整数直接量表达式〉 (4.1)
〈未编号集合元素表〉 ::= (5)
〈集合元素〉{，〈集合元素〉}* (5.1)
〈集合元素〉 ::= (6)
〈名字〉 (6.1)
| 〈未命名值〉 (6.2)
〈未命名值〉 ::= (7)
* (7.1)

语义： 集合模式定义了命名值或未命名值的集合。命名值由集合表中的名字表示，未命名值是其它的值。命名值的内部表示是和命名值相配的整数值（见下面）。这个表示也定义了值的次序。

静态性质： 集合模式有下列继承性质：

- 集合模式有一个由**集合元素**名字构成的集合，这就是它的集合表中的元素名字的集合。
- 集合模式的每个**集合元素**名字都有一个整数（表示）值。在编号集合表的情况下，它就是该**集合元素**名字在其中出现的编号集合元素中的整数直接量表达式提供的值，否则，就是0、1、2、……等值之一，根据它在未编号集合表中所占的位置而定。例如，SET(*,A,*,B,*)，A的表示值是1，而B的表示值是3。
- 集合模式都有**上界和下界**，它们分别是该集合中标志最大和最小命名值的**集合元素**名字。
- 集合模式的**值的个数**是这样确定的：在编号集合表的情况下，它是诸**集合元素**名字具有的值中最

大的那个值再加一，否则，它是未编号集合表中集合元素出现的个数。

- 当且仅当集合表中名字出现的个数少于集合模式的值的个数时，该集合模式称为**有孔的集合模式**。

静态条件： 集合表中每个整数直接量表达式必须提供如下意义的不同的非负整数值，即对任意两个表达式e_1和e_2，NUM(e_1)和NUM(e_2)提供不同的结果。

一个集合模式至少定义一个命名的值。

例子：

11.5 $SET(occupied, free)$ (1.1)

6.3 $month$ (1.2)

3.4.6 区段模式

语法：

〈区段模式〉 ::= (1)

〔*READ*〕〈离散模式名字〉（〈直接量区段〉） (1.1)

| 〔*READ*〕*RANGE*（〈直接量区段〉） (1.2)

| 〔*READ*〕*BIN*（〈整数直接量表达式〉） (1.3)

| 〔*READ*〕〈区段模式名字〉 (1.4)

〈直接量区段〉 ::= (2)

〈下界〉：〈上界〉 (2.1)

〈下界〉 ::= (3)

〈离散直接量表达式〉 (3.1)

〈上界〉 ::= (4)

〈离散直接量表达式〉 (4.1)

导出语法： 标志*BIN*(n)是由$INT(0:2^n-1)$导出的。例如，$BIN(2+1)$代表$INT(0:7)$。

语义： 区段模式定义位于直接量区段指明的边界之间的值的集合(边界包括在内)。区段取自特定的父本模式，该父本模式确定了区段中值的操作和次序。

静态性质： 区段模式有如下的(非继承)性质，它有一个唯一的父本模式，确定的方法是：

- 若区段模式的形式为：

  〈离散模式名字〉（〈直接量区段〉）

  则，如果该离散模式名字不是区段模式，则父本模式就是该离散模式名字，否则，它是该离散模式名字的父本模式。
- 若区段模式的形式为：

  RANGE（〈直接量区段〉）

  则父本模式是直接量区段中上界和下界的类的结果类的根模式。
- 若区段模式为异名模式名字，则它的父本模式是该异名模式名字的定义模式的父本模式。
- 若区段模式是一个新模式名字，则它的父本模式是虚拟引进的父本模式（见3.2.3节）。

区段模式有如下的继承性质：

- 区段模式有一个**下界**和一个**上界**，它们都是直接量，分别标志直接量区段中的下界和上界提供的值。
- 区段模式中**值的个数**是由$NUM(U)-NUM(L)+1$提供的值，其中U和L分别标志区段模式的**上界和下界**。
- 当且仅当一个区段模式的父本模式是一个**有孔的集合模式**，并且有一个未命名值位于由此区段模式指明的区段中时，该区段模式称为**有孔的区段模式**。

静态条件： 上界的类和下界的类应相容，并且这两个类都必须和离散模式名字相容，如果指明了这样的名字的话。

下界提供的值必须小于或等于上界提供的值。并且，如果指明了离散模式名字的话，这两个值必须位于该模式名字所定义的值的区段之内。

例子：

9.4 *INT*(2:*max*) (1.1)

11.11 *line* (1.4)

9.4 2:*max* (2.1)

3.5 幂集模式

语法：

〈幂集模式〉::= (1)

〔*READ*〕 *POWERSET* 〈成员模式〉 (1.1)

| 〔*READ*〕 〈幂集模式名字〉 (1.2)

〈成员模式〉::= (2)

〈离散模式〉 (2.1)

语义： 幂集模式定义的值是它的成员模式的值组成的集合。幂集值的范围遍及成员模式的所有子集。通常的集合论运算定义在幂集值上（参见5.3节）。

静态性质： 幂集模式有如下的继承性质：

- 它有一个唯一的成员模式，即由成员模式标志的模式。

例子：

8.4 *POWERSET CHAR* (1.1)

9.4 *POWERSET INT*(2:*max*) (1.1)

9.4 *number_list* (1.2)

3.6 关联模式

3.6.1 概述

语法：

〈关联模式〉::= (1)

〈约束关联模式〉 (1.1)

| 〈自由关联模式〉 (1.2)

| 〈行模式〉 (1.3)

语义： 关联模式定义对可关联地点的关联(地址或描述符)。按照定义，约束关联关联于给定静态模式的地点；自由关联可以关联具有任意静态模式的地点，行关联于具有动态模式的地点。

在关联值上定义非关联化运算（见4.2.3,4.2.4和4.2.15节),运算的结果提供被关联的地点。

当且仅当两个关联值关联于同一地点,或两个关联值都不关联于任何地点(即它们的值是NULL)时，这两个关联值相等。

3.6.2 约束关联模式

语法：

〈约束关联模式〉::= (1)

〔*READ*〕 *REF* 〈被关联模式〉 (1.1)

| 〔*READ*〕 〈约束关联模式名字〉 (1.2)

〈被关联模式〉::= (2)

〈模式〉 (2.1)

语义： 约束关联对所指明的被关联模式的地点定义关联值。

静态性质： 约束关联模式具有下列继承性质：

· 它有唯一的被关联模式，即被关联模式标志的模式。

例子：

10.38 *REF cell* (1.1)

3.6.3 自由关联模式

语法：

〈自由关联模式〉::= (1)

〔*READ*〕*PTR* (1.1)

| 〔*READ*〕〈自由关联模式名字〉 (1.2)

语义： 自由关联模式对任一静态模式的地点定义关联值。

例子：

19.5 *PTR* (1.1)

3.6.4 行模式

语法：

〈行模式〉::= (1)

〔*READ*〕*ROW*〈串模式〉 (1.1)

| 〔*READ*〕*ROW*〈数组模式〉 (1.2)

| 〔*READ*〕*ROW*〈变体结构模式名字〉 (1.3)

| 〔*READ*〕〈行模式名字〉 (1.4)

语义： 行模式对具有动态模式的地点定义关联值(这是一些带有不可静态地确定的参数的参数化模式的地点)。

行值可以关联于：

其长度不可静态确定的串地点，

其上界不可静态确定的数组地点，

其参数不可静态确定的参数化结构地点。

静态性质： 行模式有下列继承性质：

· 它有被关联的原始模式，这个被关联的原始模式分别是串模式、数组模式或变体结构模式名字。

例子：

8.6 *ROW CHAR(max)* (1.1)

3.7 过程模式

语法：

〈过程模式〉::= (1)

〔*READ*〕*PROC* ([〈参数表〉])〔〈结果说明〉〕

〔*EXCEPTIONS* (〈异常表〉)〕〔*RECURSIVE*〕 (1.1)

| 〔*READ*〕〈过程模式名字〉 (1.2)

〈参数表〉::= (2)

〈参数说明〉{,〈参数说明〉}* (2.1)

〈参数说明〉::= (3)

〈模式〉〔〈参数属性〉〕〔〈寄存器名字〉〕 (3.1)

〈参数属性〉::= (4)

IN|*OUT*|*INOUT*|*LOC* (4.1)

〈结果说明〉::= (5)

〔*RETURNS*〕(〈模式〉〔*LOC*〕〔〈寄存器名字〉〕) (5.1)

〈异常表〉::= (6)

〈异常名字〉[,〈异常名字〉]* (6.1)

〈异常名字〉::= (7)

〈名字〉 (7.1)

导出语法： 不带任选关键词*RETURNS*的结果说明是带*RETURNS*的结果说明的导出语法。

语义： 过程模式定义(通用)过程值，即由**通用过程**名字标志的对象，这些名字是在过程定义语句或入口定义语句中定义的。过程值在动态上下文中指示成片的代码。过程模式使我们能动态地处理过程，如把它作为参数传递给另一些过程，作为报文值发送给一个缓冲区，或把它存在一个地点中，等等。

过程值可以被调用（见6.7节）。

当且仅当两个过程值在同一个动态上下文中标志同一个过程，或都不标志任何过程（即，它们的值都为*NULL*）时，这两个过程值是相等的。

静态性质： 过程模式有下列继承性质：

- 它有一张**参数说明表**，每个**参数说明**由模式以及可能还有参数属性和/或寄存器名字组成。**参数说明**由参数表定义。
- 它有任选的**结果说明**，该**结果说明**由模式及任选的*LOC*属性和/或**寄存器**名字组成。**结果说明**由结果说明定义。
- 它有一个由**异常**名字组成的可能为空的集，这些名字就是异常表中提到的那些。
- 它有递归性，当指明*RECURSIVE*时，它是递归的，否则，由实现来缺省地指明它是**递归的**或**非递归的**。

静态条件： 在异常表中提到的所有名字都必须是不同的。

只有在参数说明或结果说明中指明*LOC*，其中的模式才可能有同步性质。

3.8 样品模式

语法：

〈样品模式〉::= (1)

[*READ*] *INSTANCE* (1.1)

| [*READ*] 〈样品模式名字〉 (1.2)

语义： 样品模式定义的值唯一地识别进程。新进程的建立（见5.2.17和8.1节）提供一个唯一的样品值，供识别这个被建立的进程用。

当且仅当两个样品值识别同一个进程时，或当它们都不识别任何进程时(即它们的值为*NULL*)，它们是相等的。

例子：

15.29 *INSTANCE* (1.1)

3.9 同步模式

3.9.1 概述

语法：

〈同步模式〉::= (1)

〈事件模式〉 (1.1)

| 〈缓冲区模式〉 (1.2)

语义： 同步模式的地点提供进程之间(见第8章)的同步和通信的手段。在CHILL中不存在标志一个由同步模式定义的值的表达式。因此，并未定义在这些值上的操作。

3.9.2 事件模式

语法：

〈事件模式〉 ::= (1)

〔*READ*〕 *EVENT* 〔(〈事件长度〉)〕 (1.1)

| 〔*READ*〕 〈事件模式名字〉 (1.2)

〈事件长度〉 ::= (2)

〈整数直接量表达式〉 (2.1)

语义： 事件模式地点提供进程之间的同步手段。在事件模式地点上定义的操作是继续动作、延迟动作和延迟情况动作，这些动作分别在6.15、6.16和6.17节中描述。

静态性质： 事件模式有下列继承性质：

- 它可能附有**事件长度**，这是由NUM（事件长度）提供的值。

静态条件： 事件长度的值必须是正数。

例子：

14.10 *EVENT* (1.1)

3.9.3 缓冲区模式

语法：

〈缓冲区模式〉 ::= (1)

〔*READ*〕 *BUFFER* 〔(〈缓冲区长度〉)〕

〈缓冲区元素模式〉 (1.1)

| 〔*READ*〕 〈缓冲区模式名字〉 (1.2)

〈缓冲区长度〉 ::= (2)

〈整数直接量表达式〉 (2.1)

〈缓冲区元素模式〉 ::= (3)

〈模式〉 (3.1)

注意，上述语法在与数组模式的语法连用时是有二义性的。此时适用如下的解释：若关键词*BUFFER*之后紧跟着一个开圆括号，则紧跟在开圆括号后的那段程序应看作是任选的缓冲区长度指示的开始，而不属于该缓冲区元素模式。

语义： 缓冲区模式地点提供进程之间同步和通信的手段。定义在缓冲区地点上的操作是发送动作、接收情况动作和接收表达式，这些操作分别在6.18、6.19和5.2.18节中描述。

静态性质： 缓冲区模式有下列继承性质：

- 它有任选的**缓冲区长度**，即由*NUM*（缓冲区长度）提供的值。
- 它有**缓冲区元素**模式，即由缓冲区元素模式标志的模式。

静态条件： 缓冲区长度必须提供非负的值。缓冲区元素模式不能有同步性质。

例子：

16.28 *BUFFER*(1)*USER_MESSAGES* (1.1)

16.32 *USER_BUFFERS* (1.2)

3.10 组合模式

3.10.1 概述

语法：

〈组合模式〉 ::= (1)

〈串模式〉 (1.1)
| 〈数组模式〉 (1.2)
| 〈结构模式〉 (1.3)

语义： 组合地点和组合值分别拥有可被访问或获取的子地点和子值(参见4.2.5-9,4.2.13-14和5.2.6-12节)。

3.10.2 串模式

〈串模式〉::= (1)
〔READ〕〈串类型〉(〈串长度〉) (1.1)
| 〈参数化串模式〉 (1.2)
| 〔READ〕〈串模式名字〉 (1.3)
〈参数化串模式〉::= (2)
〔READ〕〈原始串模式名字〉(〈串长度〉) (2.1)
| 〔*READ*〕〈参数化串模式名字〉 (2.2)
〈原始串模式名字〉::= (3)
〈串模式名字〉 (3.1)
〈串类型〉::= (4)
CHAR (4.1)
| *BIT* (4.2)
〈串长度〉::= (5)
〈整数直接量表达式〉 (5.1)

语义： 串模式定义位串值或字符串值，值的长度由串模式指出或隐含地指出。

一个给定的串模式的串值是良序的。对字符串值来说,它的次序就是CCITT第五号字母表定义的词典次序。位串值的排序方法是：等于1的字位比等于0的字位大。

在串值上定义有连接运算符，在位串值上定义有通常的逻辑运算符（见5.3节）。

静态性质： 串模式有下列继承性质：

- 它可以是**字位**串模式或**字符**串模式，取决于串类型指明的是*BIT*还是*CHAR*,也取决于原始串模式名字是**字位**串模式还是**字符**串模式。
- 它具有**串长度**，即由*NUM*（串长度）提供的值。

静态条件： 串长度必须提供非负值。

直接包含在参数化串模式中的串长度提供的值必须小于或等于原始串模式名字的串长度。

例子：

7.4.5 *CHAR* (20) (1.1)

3.10.3 数组模式

语法：

〈数组模式〉::= (1)
〔*READ*〕〔*ARRAY*〕(〈下标模式〉{,〈下标模式〉}*)〈元素模式〉{〈元素布局〉}* (1.1)
| 〈参数化数组模式〉 (1.2)
| 〔*READ*〕〈数组模式名字〉 (1.3)
〈参数化数组模式〉::= (2)
〔READ〕〈原始数组模式名字〉(〈上界标〉) (2.1)
| 〔*READ*〕〈参数化数组模式名字〉 (2.2)
〈原始数组模式名字〉::= (3)
〈数组模式名字〉 (3.1)
〈下标模式〉::= (4)

〈离散模式〉 (4.1)
|〈直接量区段〉 (4.2)
〈上界标〉::= (5)
〈直接量表达式〉 (5.1)
〈元素模式〉::= (6)
〈模式〉 (6.1)

导出语法： 关键词*ARRAY*是任选的。不带关键词*ARRAY*的(既非数组模式名字又非参数化数组模式)数组模式是由带关键词*ARRAY*的数组模式导出的。

下标模式记号〈直接量区段〉是由离散模式*RANGE*(〈直接量区段〉)导出的。若数组模式的下标模式多于一个(标志一个'多维的'数组)，则此数组模式是从一个其元素模式为数组模式的数组模式中导出的。

例如 *ARRAY*(1:20, 1:10)*INT*

是从 *ARRAY*(*RANGE*(1:20))*ARRAY*(*RANGE*(1:10))*INT* 中导出的。

只有使用这种导出语法时，才允许有多于一个的元素布局出现。元素布局的出现个数必须小于或等于下标模式的出现个数。在这种情况下，最左边的元素布局对应于最内层的元素模式，如此等等。

语义： 数组模式定义组合值。它们是一些由其元素模式定义的值的表。一个数组地点或数组值的物理布局可由元素布局说明所控制(见3.10.6节)。当且仅当两个数组值的相应元素的值都相等时它们自己也相等。

静态性质： 数组模式有下列继承性质：

- 它有一个**下标模式**。如果此数组模式不是一个参数化数组模式，则**下标模式**是由下标模式标志的离散模式，否则就是按如下方法构造的区段模式：

  &名字(下界：上界)

  其中&名字是一个**虚拟的异名模式名字**，它与原始数组模式名字的**下标模式**同义，下界是原始数组模式名字的**下标模式**的下界，上界是上界标。
- 它有**上界和下界**，它们分别是其**下标模式**的上界和下界。
- 它具有**元素模式**，或为M，或为*READ M*，其中M分别是元素模式，或原始数组模式名字的**元素模式**。当且仅当M不是只读模式且数组模式是只读模式时**元素模式**是*READ* M。
- 它具有**元素布局**。当它是参数化数组模式时，此布局是它的原始数组模式名字的**元素布局**，否则，它是所指明的元素布局；或者要由实现决定。在后一种情况下，它是*PACK*或*NOPACK*。
- 当且仅当指明了元素布局，并指明为步时，它是**被映射**的模式。
- 它有**元素个数**，此个数为：

  NUM(上界) − NUM(下界)+ 1 提供的值。

静态条件： 上界标的类必须与原始数组模式名字的**下标模式**相容，它的值必须在该**下标模式**定义的区段中。下标模式不能是有孔的集合模式，也不能是有孔的区段模式。

例子：

5.30 *ARRAY*(1:16) *STRUCT*(*C*4, *C*2, *C*1 *BOOL*) (1.1)
11.10 *ARRAY*(*line*) *ARRAY*(*column*)*square* (1.1)
11.15 *board* (1.3)

3.10.4 结构模式

语法：

〈结构模式〉::= (1)
〈嵌套结构模式〉 (1.1)
|〈层次结构模式〉 (1.2)
|〈参数化结构模式〉 (1.3)
|[*READ*]〈结构模式名字〉 (1.4)
〈嵌套结构模式〉::= (2)
[*READ*] *STRUCT*(〈场〉{,〈场〉}*) (2.1)

〈场〉::= (3)

〈固定场〉 (3.1)

|〈选用场〉 (3.2)

〈固定场〉::= (4)

〈名字表〉〈模式〉[〈场布局〉] (4.1)

〈选用场〉::= (5)

CASE[〈标签〉]*OF*

〈变体选择对象〉{,〈变体选择对象〉}*

[*ELSE*[〈变体场〉{,〈变体场〉}*]]*ESAC* (5.1)

〈变体选择对象〉::= (6)

[〈情况标号说明〉]

:[〈变体场〉{,〈变体场〉}*] (6.1)

〈标签〉::= (7)

〈标签场名字〉{,〈标签场名字〉}* (7.1)

〈变体场〉::= (8)

〈名字表〉〈模式〉[〈场布局〉] (8.1)

〈参数化结构模式〉::= (9)

[*READ*]〈原始变体结构模式名字〉

(〈直接量表达式表〉) (9.1)

|[*READ*]〈参数化结构模式名字〉 (9.2)

〈原始变体结构模式名字〉::= (10)

〈变体结构模式名字〉 (10.1)

〈直接量表达式表〉::= (11)

〈直接量表达式〉{,〈直接量表达式〉}* (11.1)

导出语法： 层次结构模式是嵌套结构模式的导出语法，见3.10.5节中的解释。

如果名字表由不止一个名字组成，则固定场出现和变体场出现分别是只有一个名字的多个固定场出现或变体场出现的导出语法，其中每个出现都有被指明的模式和一个任选的场布局。在有场布局的情况下，此场布局不能是地位。例如：

STRUCT(*I*, *J BOOL PACK*) 是从

STRUCT(*I BOOL PACK*, *J BOOL PACK*) 导出的。

语义： 结构模式定义的组合值由一组值构成。这些值可通过分量名字来选择。每个值都由分量名字的模式定义。结构值可存放于（组合的）结构地点中，其中分量名字作访问子地点用。结构值或结构地点的分量称为**场**，它们的名字称为**场名字**。

有固定结构、变体结构和参数化结构。

固定结构仅由固定场组成，这些场总是存在的，对它们访问时无需任何动态检验。

变体结构有变体场，这些场不总是存在。对于带标签的变体结构来说，这些场的存在与否，只能在运行时，根据某些称为标签场的固定场的值来判断。无标签的变体结构没有标签场。由于变体结构的组合可以在运行时改变，变体结构地点的体积应根据变体选择对象的最大可能(最坏可能)来确定。

用直接量表达式在变体结构模式中静态地指明如何选用变体，即得到参数化结构。这种组合自建立参数化结构之始就确定了下来，并且在运行时不再改变。如果有标签场存在，则这些标签场是只读的并自动取指明的值为初值。对于参数化结构地点来说，在它说明或生成之时即可精确地分配存储。注意，(虚拟的)动态参数化结构模式也是存在的，它们的语义在3.11.4节中定义。

结构地点或结构值的布局可以用一个场布局说明来控制(见3.10.6节)。

当且仅当两个结构值的相应分量值都相等时，这两个结构值也相等。但如果两个结构值中至少有一个是无标签变体结构值，则比较的结果由实现来定义。

静态性质：

一般性质：

结构模式有下列的继承性质：

- 当且仅当结构模式由嵌套（或层次）结构模式所标志，且后者不直接包含选用场出现时，前者是一个固定的结构模式。
- 当且仅当结构模式由嵌套（或层次）结构模式标志并至少包含一个选用场出现时，此结构模式是变体结构模式。
- 当且仅当结构模式由参数化结构模式标志时，此结构模式是参数化结构模式。
- 结构模式有一组场名字，它们分别按不同情况确定如下：当且仅当一个名字在一个结构模式的固定场或变体场的名字表中被定义时，这个名字称为场名字。一个给定的结构模式的每个场名字有唯一的场模式，它或者是M，或者是$READ\ M$，其中M是紧跟在场名字之后的模式。如果场名字之后的模式不是只读模式，并且它是参数化结构模式的标签场名字(见下面)，或者结构模式是只读的模式，则场模式是$READ\ M$。

  一个给定的结构模式的场名字有唯一的场布局。如果场名字后面有场布局，它就是这个场布局，否则就是缺省的场布局，即$PACK$或$NO\ PACK$。当且仅当场名字的场布局是$NO\ PACK$时，该场名字是（语言）可关联的。
- 当且仅当结构模式的场名字有一个场布局，并且就是地位时，该结构模式标志一个被映射的模式。

固定结构：

固定结构模式有下列继承性质：

- 它有一组场名字，即由固定场中任何一个名字表所定义的那组名字。这组场名字是固定场名字。

变体结构：

变体结构模式有下列继承性质：

- 它有一组场名字，这组名字是固定场中任一名字表定义的名字的集合和选用场中任一名字表定义的名字的集合之并集。固定场中名字表定义的场名字是变体结构模式的固定场名字，它的其余场名字是变体场名字。

  当且仅当变体结构模式的场名字在选用场的任一标签中出现时，该场名字称为标签场名字。未指明标签的选用场称为无标签选用场。在无标签选用场中，由变体场的任一名字表定义的变体场名字是无标签变体场名字。其它变体场名字是带标签变体场名字。
- 当且仅当一个变体结构模式的所有选用场出现都是无标签时，该模式也是无标签变体结构模式。否则就是带标签变体结构模式。
- 当且仅当变体结构模式是一个带标签变体结构模式，或是一个无标签变体结构模式，并且在后一种情况下，对选用场的每一个出现中的变体选择对象的全体出现都给出一个情况标号说明时，该变体结构模式是可参数化的变体结构模式。
- 每个可参数化的变体结构模式都有一个类表，按如下方式确定：
  - 当它是一个带标签变体结构模式时，这个类表由一些Mi值类组成，其中Mi是各标签场名字的模式，其排列次序和它们在固定场中的定义次序相同；
  - 当它是一个无标签变体结构模式时，这个类表可按如下方式得到：取各选用场的结果类表，把这些类表按选用场出现的次序连接在一起，即得到可参数化变体结构模式的类表。而每个选用场出现的结果类表又正是在此选用场出现中的情况标号说明出现表的结果类表(见9.1.3节)。

参数化结构：

参数化结构有下列继承性质：

- 它有一个原始变体结构模式，即由原始变体结构模式名字标志的模式。
- 当且仅当它的原始变体结构模式是带标签变体结构模式时，它就是带标签参数化结构模式，否则，此参数化结构模式就是无标签的。
- 它有一组场名字，这些名字是两个集合的并集。其中一个是它的原始变体结构模式的固定场名字的集合，另一个是它的原始变体结构模式的那些在变体选择对象出现中定义的变体场名字的集合，而这些变体选择对象出现是由直接量表达式表定义的值表所选出的。

  参数化结构模式的标签场名字的集合就是它的原始变体结构模式的标签场名字的集合。
- 每个参数化结构模式有一个由直接量表达式表定义的值表。

静态条件：

一般条件：

结构模式的所有场名字都应是不同的。

如果有任何一个场以地位为其场布局，则所有的场均应有场布局，且都必须是地位。

变体结构：

每个标签场名字必须是一个固定场名字，且必须在行文上定义于所有在其标签中提到的选用场之前。(因此，任一标签场应位于所有依赖于它的变体场之前。)标签场名字的模式必须是离散模式。

在变体结构模式中，所有的选用场出现要末全部是带标签的，要末全部是无标签的。对于无标签的选用场来说，要末在所有的变体选择对象出现中都省略情况标号说明，要末在所有的变体选择对象出现中都配上情况标号说明。

如果无标签变体结构模式的任何一个选用场有情况标号说明，则所有它的选用场都应有情况标号说明。

对于选用场来说，情况选择条件是必须满足的（见9.1.3节),并且，凡是情况动作所要求的完备性、一致性和相容性条件，在这里也必须同样成立（见6.4节)。如果有标签的话，标签中的每一个标签场名字都是M值类的一个情况选择符，其中M是标签场名字的模式。在无标签选用场的情况下，不对情况选择符进行检验。

可参数化变体结构模式的类表中任意一个类都不能是完全类。(带标签变体结构模式自动满足这个条件。)

参数化结构：

原始变体结构模式名字必须是可参数化的。

直接量表达式表中的直接量表达式的个数必须和原始变体结构模式名字的类表中的类的个数一样多。每个直接量表达式的类必须和类表中（按位置）对应的类相容。如果后者的类是一个M值类，则直接量表达式提供的值应该是一个由M定义的值。

例子：

3.3 *STRUCT (re, im INT)* (2.1)

11.5 *STRUCT (status SET (occupied, free), CASE status OF (occupied): p piece, (free): ESAC)* (2.1)

2.5 *fraction* (1.4)

11.5 *status SET (occupied, free)* (4.1)

11.6 *status* (7.1)

11.7 *p piece* (8.1)

3.10.5 层次结构表示法

导出语法：

〈层次结构模式〉 ::= (1)

1[〈数组说明〉]

[*READ*] {, 〈(2)层次场〉}$^+$ (1.1)

〈(n)层次场〉 ::= (2)

〈(n)层次固定场〉 (2.1)

| 〈(n)层次选用场〉 (2.2)

〈(n)层次固定场〉 ::= (3)

n〈名字表〉〈模式〉[〈场布局〉] (3.1)

| *n* 〈名字表〉[〈数组说明〉]

[*READ*] [〈场布局〉] {, 〈(*n*+1)层次场〉}$^+$ (3.2)

〈(*n*)层次选用场〉 ::= (4)

CASE[〈标签〉]*OF*

〈(*n*)层次选择对象〉{, 〈(*n*)层次选择对象〉}*

[*ELSE*[〈(*n*)层次变体场〉

{, 〈(*n*)层次变体场〉}*]]

ESAC (4.1)

〈(*n*)层次选择对象〉::= (5)

[〈情况标号说明〉

{,〈情况标号说明〉}*]

:[〈(*n*)层次变体场〉

{,〈(*n*)层次变体场〉}*] (5.1)

〈(*n*)层次变体场〉::= (6)

n〈名字表〉〈模式〉[〈场布局〉] (6.1)

| *n*〈名字表〉[〈数组说明〉]

[*READ*][〈场布局〉]{,〈(*n*+1)层次场〉}+ (6.2)

〈数组说明〉::= (7)

[*READ*][*ARRAY*](〈下标模式〉{,〈下标模式〉}*)

{〈元素布局〉}* (7.1)

注意，上面描述的结构的层次号表示法扩充了第二章中解释的语法描述方法：在这里，语法是利用结构层次号(*n*)作为参数而递归地定义的。

语义： 层次结构模式是对于唯一的嵌套结构模式的导出语法。

嵌套表示法被看作是严格的语法，所有的语义、性质和条件都用它来解释(见3.10.4节)。

如果一个结构中的某些场本身又是结构或结构的数组，则形成了结构的层次关系，每个场都可授予一个层次号。

例子：

```
SYNMODE m=STRUCT(b BOOL,S ARRAY(1:10)STRUCT(T INT,U BOOL));
```

整个结构的层次号是1，B和S的层次号是2，T和U的层次号是3。在层次结构模式中，也可以不写嵌套结构模式，而把层次号写在名字的前边。

例子：

```
SYNMODE M= 1, 2 B   BOOL,
              2 S   ARRAY(1:10),
                3 T   INT,
                3 U   BOOL;
```

在模式定义中和带有模式的异名定义中第一层是没有名字的。只有在说明中和在形式参数指明部分中第一层才有名字，这个名字就放在层次1位置的后面。

例子：

```
DCL  1  A,
        2  B  BOOL
        2  S  ARRAY(1:10),
           3  T  INT,
           3  U  BOOL;
```

如果在说明中或参数说明和结果说明中有属性和初始值，则应在层次1的位置后边加说明。

例子：

```
P:   PROC(1  X  INOUT,
          2  B  BOOL,
          2  C  INT);
```

如果在层次结构模式中指明了一个结构数组，则该数组说明应在层次指示符后面给出。

静态条件： 嵌套和层次表示法不得混合使用。

例子：

```
19.9  DCL  1  BASED(P),
           2  I  INFO  POS(0, 8:31),
           2  PREV  PTR  POS(1, 0:15),
           2  NEXT  PTR  POS(1, 16:31)                    (1.1)
```

3.10.6 数组模式和结构模式的布局描述

语法：

〈元素布局〉::= (1)

PACK | *NO PACK* | 〈步〉 (1.1)

〈场布局〉::= (2)

PACK | *NO PACK* | 〈地位〉 (2.1)

〈步〉::= (3)

STEP(〈地位〉[,〈步长〉[,〈图长〉]]) (3.1)

〈地位〉::= (4)

POS(〈字〉,〈起始位〉,〈长度〉) (4.1)

| *POS* (〈字〉[,〈起始位〉[:〈结束位〉]]) (4.2)

〈图长〉::= (5)

〈整数直接量表达式〉 (5.1)

〈字〉::= (6)

〈整数直接量表达式〉 (6.1)

〈步长〉::= (7)

〈整数直接量表达式〉 (7.1)

〈起始位〉::= (8)

〈整数直接量表达式〉 (8.1)

〈结束位〉::= (9)

〈整数直接量表达式〉 (9.1)

〈长度〉::= (10)

〈整数直接量表达式〉 (10.1)

语义： 数组或结构的布局是可以控制的，为此只需在它的模式中给出压缩或映射的信息。压缩信息是*PACK*或*NO PACK*。对于数组模式来说，映射信息是步。对于结构模式来说，映射信息是地位。如果在数组或结构模式中不出现*元素布局*或*场布局*，则这种情况总是解释为压缩信息，即*PACK*或*NO PACK*。

如果对一个数组的元素或一个结构的场指明了*PACK*，这意味着数组元素或结构场的存储空间的使用是优化的，而*NO PACK*则意味着数组元素或结构场的访问时间是优化的。*NO PACK*也隐含了可关联性。

*PACK*和*NO PACK*信息只用于一层。即它只用于数组的元素或结构的场，而不能用于数组元素的或结构场的可能的分量。布局信息总是附属于它能应用的最近的模式，且此模式应是尚未有布局信息的。例如，如果缺省的压缩信息是*NO PACK*，则：

STRUCT(*F ARRAY*(0:1)*M PACK*) 等价于

STRUCT(*F ARRAY*(0:1)*M PACK NO PACK*)

也可对一个组合对象的精确布局进行控制，方法是为模式中的各分量指明定位信息。定位信息按下列方式给出：

- 对于数组模式，定位信息对所有的元素要一起给出，其形式为数组模式后面的步。
- 对于结构模式，定位信息对每个场要分别给出，其形式是场模式后面的地位。

以*C*表示对象的一个分量，即元素或场，则*C*的精确定位由下列Wc、Bc和Lc三个常数给出，其中

Wc是被*C*（可能部分地）占用的第一个字相对于该对象（可能部分地）占用的第一个字之间按字数计算的距离，*C*即该对象的一个分量。

Bc是被*C*占用的第一个字位相对于被*C*（可能部分地）占用的第一个字的最左边的字位之间按字位个数计算的距离。

Lc是*C*占用的字位个数。

如果对象是完整的（即它不是另一个对象的分量），则为此对象诸分量给出的定位信息将精确地确定它们的位置。但如果对象不是完整的，则分量的精确定位与该对象本身的精确定位有关。

为一个数组的元素指明的步是一种用以显式地逐个表示每个元素地位的简写方法。非形式地说，地位和步长指明了一批元素的“定位图象”，在假定数组为完整的情况下，这批元素能完全装进个数等于图长的头几个字中。第一个元素的定位由地位确定；能完全装进个数等于图长的头几个字中的下余元素的定位是这样确定的；这些元素所占用的各个第一个字位之间的距离，按字位个数计算，正好等于步长。以这种方式指明的定位图象需要重复多少次就重复多少次，每重复一次就表明一组字，每组字的个数等于图长。

地位

给定一个具有被映射模式的对象O，其中为该对象的一个分量C指明了形式为：

POS（〈字号〉，〈起始位〉，〈长度〉）的一个地位，则此分量C的精确定位按下列方式确定：

- 若对象O（C是它的一个分量）是完整的，则

  Wc是NUM（字号），

  Bc 是NUM（起始位），

  Lc 是NUM（长度）

- 若对象O（C是它的一个分量）不是完整的，则

  Wc是NUM（字号）+（NUM（起始位）+ Bo）/$WIDTH$。

  Bc 由（NUM(起始位) + Bo）$MOD\ WIDTH$所标志。

  Lc 由NUM(长度)所标志。

  其中$WIDTH$是一个字中字位的个数。

步

令元素 i 为：

- 具有最小下标的元素，若 $i = 0$ 的话。
- 否则，以元素 n 为下标的后继元素为下标的元素，其中 $n = i - 1$。

令i_0为元素 0 的定位图象中位于元素 0 之前的元素个数。给定形式为：

$STEP$（〈地位〉，〈步长〉，〈图长〉），的一个步属性，则一个元素的地位（相对于元素 0 的定位图象的开始）按如下方式确定：

元素 i 的地位是：

POS（NUM(图长)*(($i + i_0$)/$DENS$) + $RBPOS_i$/$WIDTH$，$RBPOS_i\ MOD\ WIDTH$，长度）。

其中 $1 \leqslant i \leqslant$ ‘元素个数’，

而$RBPOS_i$是：

NUM(字号)*$WIDTH$ + NUM(起始位) + NUM(步长)*(($i + i_0$)$MOD\ DENS$)，

且$DENS$是：

(NUM(图长)*$WIDTH$)/NUM(步长)，

且$WIDTH$是一个字中字位的个数。

缺省规则

式子：

POS（〈字号〉，〈起始位〉：〈结束位〉）

在语义上等价于：

POS（〈字号〉，〈起始位〉，

NUM(结束位) − NUM(起始位) + 1）。

式子：

POS（〈字号〉，〈起始位〉）

在语义上等价于：

POS(〈字号〉,〈起始位〉, $BSIZE$)。

其中$BSIZE$是指明地位的那个分量所需占用的最少字位的个数。

式子:

POS(〈字号〉)

在语义上等价于:

POS(〈字号, 0, $WSIZE$*$WIDTH$)

其中$WSIZE$是指明地位的那个分量的模式的体积。

式子:

$STEP$(〈地位〉,〈步长〉)

在语义上等价于:

$STEP$(〈地位〉,〈步长〉, $PSIZE$)

其中$PSIZE$是使得:

$PSIZE$*$WIDTH \geqslant NUM$(步长)

成立的最小整数。

式子:

$STEP$(〈地位〉)

在语义上等价于:

$STEP$(〈地位〉, $SSIZE$)

其中$SSIZE$是在地位中指明的〈长度〉,或根据上述规则可从地位中导出的〈长度〉。

静态性质: 对于被映射的数组模式的每一个地点,该模式的元素布局确定了它的子地点(包括子数组、子切片)的(语言)可关联性,方式如下:

- 或者所有的子地点都是(语言)可关联的,或者全都不是;
- 若元素布局为$NOPACK$,则所有子地点都是(语言)可关联的。

对于被映射的结构模式的每一个地点,场名字选出的结构场的可关联性要由该场名子的场布局确定,方式如下:

- 若场布局是$NOPACK$,则场名字是(语言)可关联的。

静态条件: 若给定的数组模式的元素模式或给定的结构模式的场名字的场模式本身是数组模式或结构模式,那么,在给定的数组或结构模式是被映射模式时,则它也必须是被映射模式,否则就不是。

每一个整数直接量表达式的出现必须提供一个非负的值。此外,长度,步长和图长必须提供一个非零值,起始位和结束位提供的值必须小于$WIDTH$。其中$WIDTH$是一个字中字位的个数(由实现定义)。并且,起始位提供的值不得大于结束位提供的值。

对被映射结构模式的每个场名字来说,它的场布局中的长度不得小于该场所需占用的最少字位个数。

对每个被映射数组模式来说,其元素布局中地位的长度不得小于这些元素所需占用的最少字位个数。此外,对每个元素布局来说,下列诸条件必须成立:

- NUM(步长)$\geqslant NUM$(长度)
- (NUM(图长)*$WIDTH$) MOD NUM(步长)
  $\geqslant NUM$(字号)*$WIDTH$+ NUM(起始位)

一致性和可行性

一致性:

在指明一个数组或结构对象的分量时,不得使该分量占用同一对象其它分量所占用的任何字位,除非在同一个选用场出现中定义了两个变体场名字;但在后一种情况下,两个变体场名字不得定义于同一个变体选择对象,也不得都跟在$ELSE$后边。

可行性:

不存在由语言定义的可行性要求,唯一的例外是由下列规则推出的要求,即任何一个(可关联或不可关联的)地点的子地点的可关联性仅由(元素或场)布局决定,这是地点模式的一个性质。这一点对自身具有可关联分量的分量的映射附加了某些限制。

例子：

17.5 $PACK$ (1.1)

19.11 $POS(1, 0:15)$ (4.2)

3.11 动态模式

3.11.1 概述

动态模式是这样的一种模式，它的某些性质只能在运行时知道。动态模式总是带有一个或多个运行时参数的参数化模式。在$CHILL$中，动态模式没有标志。但是，为描述方便起见，在文本中引进了虚拟标志。这些虚拟标志前面都加上了&记号，以便把它们与$CHILL$程序中可能出现的实际标志区分开来。

3.11.2 动态串模式

虚拟标志： &〈原始串模式名字〉(〈整数表达式〉)

语义： 动态串模式是带有不能静态确定长度的参数化串模式。动态串长度值由整数表达式提供。

静态性质：

- 当且仅当原始串模式名字是一个**字位（字符）**串模式时，该动态串模式是一个**字位（字符）**串模式。

动态性质：

- 动态串模式有一个**动态长度**，这是由NUM(整数表达式)提供的值。

3.11.3 动态数组模式

虚拟标志： &〈原始数组模式名字〉(〈离散表达式〉)

语义： 动态数组模式是具有不能静态确定上界的参数化数组模式。下界、下标模式和元素模式是可以静态确定的，动态上界值由离散表达式提供。

静态性质：

- 动态数组模式附有**下标模式、元素模式、元素布局和下界**，它们分别是原始数组模式名字的**下标模式、元素模式、元素布局和下界**。

动态性质：

- 动态数组模式有一个**动态上界**，这是由离散表达式提供的值，还有一个动态的**元素个数**，这是由NUM(上界)$-NUM$(下界)$+1$提供的值。

3.11.4 动态参数化结构模式

虚拟标志： &〈原始变体结构模式名字〉
(〈表达式表〉)

语义： 动态参数化结构模式是一个具有不能静态地确定参数的参数化结构模式。结构模式的组成只能从表达式表提供的值表中动态地确定。

静态性质：

- 动态参数化结构模式有一个唯一的**原始变体结构模式**，即由原始变体结构模式名字标志的模式。
- 当且仅当动态参数化结构模式的**原始变体结构模式**是**带标签变体结构模式**时，它本身也是**带标签**的，否则就是**无标签**的。
- 动态参数化结构模式的**场名字（固定场名字，标签场名字，变体场名字）**之集合正是它的**原始变体结构模式的场名字（固定场名字标签场名字、变体场名字）**之集合。

动态性质：

- 动态参数化结构模式附有值表，它就是表达式表中各表达式提供的值表。

4.0 地点及其访问

4.1 说明

4.1.1 概述

语法：

〈说明语句〉::= (1)

DCL 〈说明〉{, 〈说明〉}*; (1.1)

〈说明〉::= (2)

〈地点说明〉 (2.1)

|〈地点等同说明〉 (2.2)

|〈有基说明〉 (2.3)

语义： 说明语句说明了一个或多个名字作为对一个地点的访问。

例子：

6.9 *DCL j INT*:=*julian-day-number*, *d*, *m*, *y INT*; (1.1)

6.10 *d*, *m*, *y INT* (2.1)

11.34 *starting-square LOC*:=*b*(*m*.*lin*−1) (*m*.*col*−1) (2.2)

4.1.2 地点说明

语法：

〈地点说明〉::= (1)

〈名字表〉〈模式〉[*STATIC*][〈初始化〉] (1.1)

〈初始化〉::= (2)

〈范围初始化〉 (2.1)

|〈生存期初始化〉 (2.2)

〈范围初始化〉::= (3)

〈赋值符号〉〈值〉[〈处理程序〉] (3.1)

〈生存期初始化〉::= (4)

INIT 〈赋值符号〉〈常量值〉 (4.1)

语义： 由一个地点说明可以建立一批地点，其个数等于名字表中表明的名字的个数。

在范围初始化中，每当进入该说明所在的范围时（见7.2节），其中的值就被计算一次，所得的结果被赋给那个(些)地点。在值计算之前，地点中包含有未定义的值。(除非说明具有带标签参数化性质或同步性质的模式；见下面)。

在生存期初始化中，由常量值提供的值仅在这（些）地点的生存期开始时被赋给它（们）一次(见7.2和7.9节)。

不指明初始化在语义上与指明生存期初始化为未定义值等价（见5.3节)。但是，如果模式有带标签的参数化性质，该地点的标签场子地点即以与它相结合的参数化结构模式的相应值为初始化值。

把未定义值指明为一个同步地点的初始化值的意义是：所建立的(诸)事件和/或缓冲区(子)地点被自动置初值“空”,即该事件或缓冲区不附有任何被延迟的进程，缓冲区中也没有任何信息。

*STATIC*和处理程序的语义可分别在7.9节和第10章中找到。

静态性质： 在地点说明中说明的名字就是地点名字。地点名字的模式就是在地点说明中指明的模式。地点名字是（语言）可关联的。

静态条件：值或常量值的类必须和该模式相容。

若该模式有只读性质，则必须指明初始化。若该模式有同步性质，则必须指明范围初始化。

动态条件：在范围初始化的情况下，值相对于模式的赋值条件必须得到遵守(见6.2节)。

例子：

```
5.8  K2, X, W, t, s, r   BOOL                                    (1.1)
6.9  :=julian-day-number                                         (3.1)
8.4  INIT:=['A':'Z']                                             (4.1)
```

4.1.3 地点等同说明

语法：

```
<地点等同说明>::=                                                  (1)
<名字表><模式>LOC<赋值符号>
<静态模式地点>[<处理程序>]                                         (1.1)
```

语义：地点等同说明建立一批对被指明的静态模式地点的访问，其个数等于名字表中指明的名字个数。

如果该静态模式地点是动态地计算的，则每次进入该地点等同说明所在的范围时，它都要被计算一次。在这种情况下，在被说明的名字所标志的访问的生存期中第一次计算该地点之前，被说明的名字标志一个**未定义的地点**（见7.2和7.9节）。

静态性质：在地点等同说明中说明的名字是**地点等同**名字。**地点等同**名字的模式就是地点等同说明中指明的模式。当且仅当被指明的静态模式地点是（语言）可关联时，**地点等同**名字也是（语言）可关联的。

静态条件：被指明的模式必须和静态模式地点的模式读相容。

例子：

```
11.34  starting square LOC:=
  b(m · lin-1) (m · col-1)                                       (1.1)
```

4.1.4 有基说明

语法：

```
〈有基说明〉::=                                                    (1)
〈名字表〉〈模式〉BASED
[(〈约束或自由关联地点名字〉)]                                     (1.1)
```

导出语法：没有约束或自由关联地点名字的有基说明是异名模式定义语句的导出语法。如

DCL I INT BASED; 是从SYNMODE I=INT; 导出的。

被说明的名字是**异名模式**名字，与被指明的模式同义。

语义：带有约束或自由关联地点名字的有基说明指明一批访问，其个数与名字表中的名字一样多。在有基说明中说明的名字是对地点进行访问的又一种方法，它通过对关联值进行非关联化而访问该地点。这个关联值包含在约束或自由关联地点名字指明的地点中。当且仅当通过被说明的**有基**名字实行访问时，执行**非关联化操作**。

静态性质：在有基说明中说明的名字是**有基**名字。**有基**名字的模式是在有基说明中指明的模式。**有基**名字是（语言）可关联的。

静态条件：若约束或自由关联地点名字的模式是一个约束关联模式，则被指明的模式必须与该约束或自由关联地点名字的模式的**被关联**模式读相容。

例子：

```
19.9  1 X  BASED(P),
        2 I  INFO  POS(0, 8:31),
        2 PREV  PTR  POS(1, 0:15),
        2 NEXT  PTR  POS(1, 16:31)                               (1.1)
```

4.2 地点

4.2.1. 概述

语法：

〈地点〉::= (1)
〈静态模式地点〉 (1.1)
| 〈动态模式地点〉 (1.2)
〈静态模式地点〉::= (2)
〈访问名字〉 (2.1)
| 〈非关联化约束关联〉 (2.2)
| 〈非关联化自由关联〉 (2.3)
| 〈串元素〉 (2.4)
| 〈子串〉 (2.5)
| 〈数组元素〉 (2.6)
| 〈子数组〉 (2.7)
| 〈结构场〉 (2.8)
| 〈地点过程调用〉 (2.9)
| 〈地点内部子程序调用〉 (2.10)
| 〈地点转换〉 (2.11)
〈动态模式地点〉::= (3)
〈串切片〉 (3.1)
| 〈数组切片〉 (3.2)
| 〈非关联化行〉 (3.3)

语义： 地点是可以包含值的对象。地点可以被静态模式地点所标志，即它的模式是可静态确定的。也可以被动态模式地点所标志，即该模式的部分信息只能动态地获得。访问地点是为了存储或取值。

静态性质： 静态模式地点有一个静态模式。仅仅为了便于描述，给每个动态模式地点配上一个虚拟动态模式(见3.1节)。动态模式地点所要求的相容性检验只有在运行时才能完全执行。动态部分检验失败将导致*RANGEFAIL*或*TAGFAIL*异常。

4.2.2 访问名字

语法：

〈访问名字〉::= (1)
〈地点名字〉 (1.1)
| 〈地点等同名字〉 (1.2)
| 〈有基名字〉 (1.3)
| 〈地点枚举名字〉 (1.4)
| 〈地点直接场名字〉 (1.5)

语义： 访问名字是对地点的一个访问。

访问名字有下列几种：

- **地点名字**，即在地点说明中显式地说明的名字或在没有LOC属性的形式参数中隐含地说明的名字；
- **地点等同名字**，即在地点等同说明中显式地说明的名字，或在有LOC属性的形式参数中隐式地说明的名字；
- **有基名字**，即在有基说明中说明的名字；

- **地点枚举名字**，即在地点枚举中的循环计数器；
- **地点直接场名字**，即带结构型部分的循环动作中用作直接访问的场名字。

如果由地点直接场名字标志的地点是无标签变体结构地点的变体场，则其语义由实现来定义。

静态性质： 访问名字的模式分别是地点名字、地点等同名字、有基名字、地点枚举名字或地点直接场名字的模式。

当且仅当访问名字是地点名字、可关联地点等同名字，有基名字，可关联地点枚举名字，或可关联地点直接场名字时，它才是（语言）可关联的。

动态条件： 地点等同名字不得标志未定义的地点。

当通过有基名字进行访问时，应该遵守的动态条件与相应的有基说明中对约束或自由关联地点名字进行非关联化时所应遵守的动态条件相同（见4.2.3和4.2.4节）。

通过地点直接场名字进行访问时，如果被标志的地点满足下列两个条件之一，将引发 *TAGFAIL* 异常：

- 它是带标签变体结构模式地点的变体场，并且相应(诸)标签场的值表明该场并不存在；
- 它是动态参数化结构模式地点的变体场，并且相应的值表表明该场并不存在。

例子：

4.11 a (1.1)

11.38 *starting* (1.2)

19.14 *X* (1.3)

15.25 *EACH* (1.4)

5.11 *C 1* (1.5)

4.2.3 非关联化约束关联

语法：

〈非关联化约束关联〉::= (1)

〈约束关联表达式〉→[〈模式名字〉] (1.1)

语义：

对一个约束关联值实行非关联化后得到的地点就是该约束关联值关联的地点。

静态性质： 非关联化约束关联具有的模式就是模式名字，如果已指明了模式的名字。否则，它是约束关联表达式的模式所关联的模式。非关联化约束关联是（语言）可关联的。

静态条件： 约束关联表达式必须是强的。如果指明了任选的模式名字，则应该和约束关联表达式的模式的被关联模式读相容。

动态条件： 被关联的地点的生存期不能是已经结束的。

当约束关联表达式的值为NULL时，将出现EMPTY异常。

例子：

10.49 P—> (1.1)

4.2.4 非关联化自由关联

语法：

〈非关联化自由关联〉::= (1)

〈自由关联表达式〉→〈模式名字〉 (1.1)

语义： 对自由关联值实行非关联化后得到的地点就是该自由关联值关联的地点。

静态性质： 非关联化自由关联的模式是模式名字。非关联化自由关联是（语言）可关联的。

静态条件： 自由关联表达式必须是强的。

动态条件： 自由关联表达式不得提供一个通过对不可关联的地点实行关联所得到的值（见5.2.13节）。被关联的地点的生存期不得是已结束的。

若自由关联表达式提供的值为NULL，则将出现EMPTY异常。

若模式名字不与被关联地点的模式读相容，则出现MODEFAIL异常。

4.2.5 串元素

语法：

〈串元素〉::= (1)

〈串地点〉(〈定位〉) (1.1)

导出语法： 串元素是长度为1的子串的导出语法（见4.2.6节），例如：

〈串地点〉(〈定位〉)是从〈串地点〉(〈定位〉UP1）中导出的。

例子：

18.16 string→ (i) (1.1)

4.2.6 子串

语法：

〈子串〉::= (1)

〈串地点〉(〈左元素〉:〈右元素〉) (1.1)

| 〈串地点〉(〈定位〉UP〈串长度〉) (1.2)

〈左元素〉::= (2)

〈整数直接量表达式〉 (2.1)

〈右元素〉::= (3)

〈整数直接量表达式〉 (3.1)

〈定位〉::= (4)

〈整数表达式〉 (4.1)

语义： 子串提供了串地点，它是被指明的串地点的子串。

静态性质： 子串的模式是参数化的串模式，构造方法如下：

&名字（子串长度）

其中&名字是虚拟异名模式名字，与该串地点的（可能是动态的）模式同义，其中子串长度是串长度或是

NUM（右元素）—NUM（左元素）+1

静态条件： 左元素，右元素和长度必须提供非负整数值，并有下列关系成立：

1. NUM（左元素）$\leqslant NUM$（右元素）
2. NUM（右元素）$\leqslant L-1$
3. $1 \leqslant NUM$（串长度）$\leqslant L$

其中L是串地点的串长度。（若串地点是一个动态模式地点，则上述关系2和3只能在运行时检验；见下面。）

动态条件： 如果在动态模式串地点的情况下，上述关系2或3中有任何一个不成立；或者如果下列条件之一得到满足，则将出现$RANGEFAIL$异常：

1. NUM（定位）<0
2. NUM（定位）$+NUM$（串长度）$>L$

其中L是该串地点的模式的串长度。

例子：

18.23 string→ (scanstart UP 10) (1.2)

4.2.7 数组元素

语法：

〈数组元素〉::= (1)

〈数组地点〉(〈表达式表〉) (1.1)

〈表达式表〉::= (2)

〈表达式〉{,〈表达式〉}* (2.1)

导出语法： 记号：(〈表达式表〉) 是

(〈表达式〉){(〈表达式〉)}* 的导出语法，其中带括号表达式的个数与表达式表中表达式的个数一样多。因此，严格语法中每个数组元素只有一个（下标）表达式。

语义： 一个数组元素提供一个（子）地点，它是被指明的数组地点的一个元素。

静态性质： 数组元素的模式是数组地点模式的元素模式。

如果数组地点模式的元素布局是 $NOPACK$，则它的数组元素是（语言）可关联的。

静态条件： 表达式的类必须和数组地点模式的下标模式相容。

动态条件： 如果下面的关系中有任何一项成立，则将出现 $RANGEFAIL$ 异常。

1. 表达式＜L
2. 表达式＞U

其中L和U分别是数组地点的模式的下界和（可能是动态的）上界。

例子：

11.34 b(m·lin－1)(m·col－2) (1.1)

4.2.8 子数组

语法：

〈子数组〉::=

〈数组地点〉(〈下元素〉:〈上元素〉)

| 〈数组地点〉

(〈整数表达式〉UP〈数组长度〉)

〈下元素〉::=

〈直接量表达式〉

〈上元素〉::=

〈直接量表达式〉

〈数组长度〉::=

〈整数直接量表达式〉

语义： 子数组提供一个(子)数组地点，它是被指明的数组地点的一部分，后者由下元素和上元素或整数表达式和数组长度标定。子数组的下界等于被指明的数组的下界；上界根据被指明的表达式确定。

静态性质： 子数组的模式是参数化数组模式，定义如下：

&名字（上界标）。

其中&名字是一个虚拟的异名模式名字，与数组地点的（可能是动态的）模式同义。上界标可以是L＋数组长度－1，其中L是数组地点的数组模式的下界，也可以是lit，其中lit是一个直接量，它的类与下元素和上元素的类相容，并且

NUM（lit）＝NUM（L）＋NUM（上元素）－NUM（下元素）。

如果数组地点模式的元素布局是 $NOPACK$，则子数组是（语言）可关联的。

静态条件： 下元素和上元素的类，或整数表达式和数组长度的类应与数组地点的模式的下标模式相容。

下元素、上元素和数组长度提供的值必须满足下列关系：

1. L≤下元素≤上元素
2. 1≤数组长度
3. 上元素≤U
4. 数组长度≤U－L＋1

其中L和U分别是数组地点模式的下界和上界。(如果数组地点是一个动态模式地点，上述关系3和4只能在运行时检验；见下面。)

动态条件： 如果在动态模式数组地点的情况下，上述关系3和4中有任何一项不成立，或者如果下列关系中

有任何一项成立，则将出现RANGEFAIL异常：

1. L＞整数表达式
2. 整数表达式＋数组长度－1＞U

其中L和U分别是数组地点的数组模式的下界和上界。

4.2.9 结构场

语法：

〈结构场〉::= (1)

〈结构地点〉.〈场名字〉 (1.1)

语义： 结构场提供一个（子）地点，它是被指明结构地点的一个场。

若该结构地点有一个无标签变体结构模式，并且场名字是一个变体场名字，则语义由实现来定义。

静态性质： 结构场的模式是场名字的模式。若场名字是(语言)可关联的，则结构场也是(语言)可关联的。

静态条件： 场名字必须是结构地点的模式场名字集中的一个名字。

动态条件： 下列情况将引起TAGFAIL异常：

- 结构地点标志一个带标签变体结构模式地点，且相应的(诸)标签场值指出此场不存在；
- 结构地点标志一个动态的参数化结构模式地点，且相应的值表指出此场不存在。

例子：

10.52 last→info (1.1)

4.2.10 地点过程调用

语法：

〈地点过程调用〉::= (1)

〈地点过程调用〉 (1.1)

语义： 地点过程调用的结果是提供一个地点。

静态性质： 地点过程调用的模式是地点过程调用的结果说明的模式。

动态条件： 地点过程调用不得提供未定义的地点，并且被提供的地点的生存期不得是已结束的。

4.2.11 地点内部子程序调用

语法：

〈地点内部子程序调用〉::= 〈实现地点内部子程序调用〉

语义： 实现地点内部子程序调用的结果是提供一个地点。

静态性质： 地点内部子程序调用的模式是实现地点内部子程序调用的结果模式。

动态条件： 实现地点内部子程序调用不得提供未定义的地点，并且被提供的地点的生存周期不得是已经结束的。

4.2.12 地点转换

语法：

〈地点转换〉::= (1)

〈模式名字〉(〈静态模式地点〉) (1.1)

语义： 地点转换不受CHILL的模式检验和相容性规则的约束。它显式地把一个模式附加到所说明的静态模式地点上。

地点转换的精确动态语义由实现来定义。

静态性质： 地点转换的模式是模式名字。

静态条件： 静态模式地点必须是可关联的。

下列关系必须成立：

SIZE(模式名字)=SIZE（静态模式地点）

4.2.13 串切片

语法：

〈串切片〉::= (1)

〈串地点〉(〈起始〉:〈结束〉) (1.1)

〈起始〉::= (2)

〈整数表达式〉 (2.1)

〈结束〉::= (3)

〈整数表达式〉 (3.1)

注意，若起始和结束都是整数直接量表达式，则语法结构是二义的。此时它被解释为一个子串。

语义： 串切片提供动态模式串地点，即其长度不能静态确定的串。

静态性质： 串切片的动态模式是动态参数化串模式，其构造方法与子串的构造方法相同（见4.2.6节），只是带有一个动态串长度参数，形式为：

NUM（结束）－NUM（起始）＋1

动态条件： 若下列关系中的任何一项成立，则将出现RANGEFAIL异常：

1. NUM（起始）＞NUM（结束）
2. NUM（起始）＜0
3. NUM（结束）≥L

其中L是串地点串模式的（可能是动态的）长度。

例子：

18.26 blanks (count:9) (1.1)

4.2.14 数组切片

语法：

〈数组切片〉::= (1)

〈数组地点〉(〈首〉:〈尾〉) (1.1)

〈首〉::= (2)

〈表达式〉 (2.1)

〈尾〉::= (3)

〈表达式〉 (3.1)

注意，若首和尾都是直接量表达式，则语法结构是二义的，此时它被解释为子数组。

语义： 数组切片提供动态模式数组地点，即其上界不能静态确定的数组。

静态性质： 数组切片的动态模式是动态参数化数组模式，其构造方法与子数组的构造方法相同(见4.2.8节)，只是带有由exp 构成的动态上界标参数，其中exp是表达式,此表达式的类与首和尾的类相容,并且有：

NUM (exp) ＝NUM(L)＋NUM(尾)－NUM(首)

其中L是数组地点模式的下界。

如果数组地点模式的元素布局是NOPACK，则它的数组切片是（语言）可关联的。

静态条件： 首和尾的类必须和数组地点模式的下标模式相容。

动态条件： 如果下列关系中有任何一项成立，则将出现RANGEFAIL异常：

1. 首＞尾
2. 首＜L
3. 尾＞U

其中L和U分别标志数组地点模式的下界和（可能是动态的）上界。

例子：

17.27 res (0 : count－1) (1.1)

4.2.15 非关联化行

语法：

〈非关联化行〉::= (1)

〈行表达式〉→ (1.1)

语义： 非关联化行提供被该行值关联的动态模式地点。

静态性质： 非关联化行的动态模式按下列方式构造：

&原始模式名字(〈参数〉{,〈参数〉}*)

其中原始模式名字是虚拟的异名模式名字，与该(强)行表达式的模式的被关联原始模式同义，其中的参数与被关联原始模式有关：

- 若它是串模式，则参数是动态长度；
- 若它是数组模式，则参数是动态上界；
- 若它是变体结构模式，则参数是与参数化结构地点模式相结合的值表。

非关联化行是（语言）可关联的。

静态条件： 行表达式必须是强的。

动态条件： 被关联地点的生存期不得是已经结束的。

若行表达式提供NULL，则出现EMPTY异常。

例子：

8.10 input→ (1.1)

5.0 值及其操作

5.1 异名定义

语法：

〈异名定义语句〉::= (1)

SYN 〈异名定义〉{,〈异名定义〉}*; (1.1)

〈异名定义〉::= (2)

〈名字表〉[〈模式〉]=〈常量值〉 (2.1)

导出语法： 如果异名定义的名字表由多于一个名字组成，则它是从几个异名定义出现导出的，每个名字对应一个异名定义出现，并且其中带有与异名定义出现相同的常量值和模式，如果原来有这样的常量值和模式的话。例如，SYN I，J＝3 是从SYN I＝3，J＝3；中导出的。

语义： 异名定义定义一个名字，以标志被指明的常量值。

静态性质： 在异名定义中定义的名字是一个**异名**名字。

如果指明了一个模式，则**异名**名字的类是M值类，其中M是该模式，否则就是常量值的类。

当且仅当常量值是未定义值时，**异名**名字也是**未定义**的（见5.3.1节）。

当且仅当常量值是直接量表达式时，**异名**名字也是**直接量**。

静态条件： 如果指明了一个模式，它必须和常量值的类相容，并且由常量值提供的值必须是该模式定义的一个值。

异名定义不得是递归的，也不得与其它异名定义或模式定义构成相互递归。即任何一组递归定义都不得包含异名定义（见3.2.1节）。

例子：

```
1.14  SYN  neutral-for-add=0,
           neutral-for-mult=1;                    (1.1)
2.17       neutral-for-add fraction=[0,1]         (2.1)
```

5.2 原值

5.2.1 概述

语法：

〈原值〉::= (1)

〈地点内容〉 (1.1)

| 〈值名字〉 (1.2)

| 〈直接量〉 (1.3)

| 〈多元组〉 (1.4)

| 〈值串元素〉 (1.5)

| 〈值子串〉 (1.6)

| 〈值串切片〉 (1.7)

| 〈值数组元素〉 (1.8)

| 〈值子数组〉 (1.9)

| 〈值数组切片〉 (1.10)

| 〈值结构场〉 (1.11)

| 〈被关联地点〉 (1.12)

| 〈表达式转换〉 (1.13)
| 〈值过程调用〉 (1.14)
| 〈值内部子程序调用〉 (1.15)
| 〈开动表达式〉 (1.16)
| 〈接收表达式〉 (1.17)
| 〈零目运算符〉 (1.18)

语义： 原值是表达式的基本组成部分。某些原值（动态模式地点的地点内容、某些多元组、值数组切片、值串切片）有动态的类，即以动态模式为基础的类。对于这些原值来说，相容性检验只能在运行时完成。查出错误时，将导致TAGFAIL或RANGEFAIL异常。

静态性质： 原值的类分别是地点内容、值名字等的类。

当且仅当原值是常量值名字、直接量、常量多元组、常量被关联地点、常量表达式转换或常量值内部子程序调用时，原值是常量原值。

当且仅当原值是直接量值名字、离散直接量或直接量值内部子程序调用时，原值是直接量原值。

5.2.2 地点内容

语法：

〈地点内容〉::= (1)
〈地点〉 (1.1)

语义： 地点内容提供了包含在指明的地点中的值。访问该地点是为了取得存储在其中的值。

静态性质： 地点内容的类是M值类，其中M是该地点的（可能是动态的）模式。

静态条件： 地点的模式不得有同步性质。

动态条件： 被提供的值不得是未定义的（见5.3.1节）。

例子：

3.6 c2.im (1.1)

5.2.3 值名字

语法：

〈值名字〉::= (1)
〈异名名字〉 (1.1)
| 〈值枚举名字〉 (1.2)
| 〈值直接场名字〉 (1.3)
| 〈值接收名字〉 (1.4)

语义： 值名字提供一个值。

值名字分下列几种：

- **异名名字**，即在一个异名定义语句中定义的名字；
- 值枚举中的循环计数器；
- **值直接场名字**，即在带结构型部分的循环动作中作为值名字引进的场名字；
- **值接收名字**，即在接收情况动作中引进的名字。

静态性质： 值名字的类分别是异名名字、值枚举名字、值直接场名字和值接收名字的类。

当且仅当值名字是异名名字（直接量异名名字）时，它是常量（直接量）。

静态条件： 异名名字不得是未定义的。

动态条件： 在加工值直接场名字时，如果发生下列情况之一，将引发TAGFAIL异常：

- 被标志的值是带标签变体结构模式值的变体场，并且相应的（诸）标签场指出被标志的场不存在；
- 被标志的值是动态参数化结构模式值的变体场，并且相应的值表指出被标志的场不存在。

例子：

10.9 max (1.1)
8.8 i (1.2)
15.45 THIS_COUNTER (1.4)

5.2.4 直接量

5.2.4.1 概 述

语法：

〈直接量〉::= (1)
〈整数直接量〉 (1.1)
|〈布尔直接量〉 (1.2)
|〈集合直接量〉 (1.3)
|〈空直接量〉 (1.4)
|〈过程直接量〉 (1.5)
|〈字符串直接量〉 (1.6)
|〈字位串直接量〉 (1.7)

语义： 直接量提供在编译时已知的常量值。

静态性质： 直接量的类分别是整数直接量、布尔直接量等的类。整数直接量、布尔直接量、集合直接量、长度为1的字符串直接量，或长度为1的字位串直接量等都叫离散的直接量。

5.2.4.2 整数直接量

语法：

〈整数直接量〉::= (1)
〈十进整数直接量〉 (1.1)
|〈二进整数直接量〉 (1.2)
|〈八进整数直接量〉 (1.3)
|〈十六进整数直接量〉 (1.4)
〈十进整数直接量〉::= (2)
[D′]{〈数字〉|_}$^{+}$ (2.1)
〈二进整数直接量〉::= (3)
B′{0|1|_}$^{+}$ (3.1)
〈八进整数直接量〉::= (4)
O′{0|1|2|3|4|5|6|7|_}$^{+}$ (4.1)
〈十六进整数直接量〉::= (5)
H′{〈十六进数字〉|_}$^{+}$ (5.1)
〈数字〉::= (6)
0|1|2|3|4|5|6|7|8|9 (6.1)
〈十六进数字〉::= (7)
〈数字〉|A|B|C|D|E|F (7.1)

语义： 整数直接量提供整数值，既有通常的十进制表示法，也有二进制、八进制、十六进制和显式十进制表示法。下划线（_）并无意义，它只用于改进可读性，不影响所标志的值。

静态性质： 整数直接量的类是INT导出类。

静态条件： 在撇号（′）后面的串和整个整数直接量不能只由下划线符号组成。

例子：

6.11 1_721_119 (1.1)
--- D′1_721_119 (1.1)

--- B′101011__110100 (1.2)

--- 0′53__64 (1.3)

--- H′AF 4 (1.4)

5.2.4.3 布尔直接量

语法：

〈布尔直接量〉::= (1)

FALSE|TRUE (1.1)

语义： 布尔直接量提供布尔值。

静态性质： 布尔直接量的类是BOOL导出类。

例子：

5.46 FALSE (1.1)

5.2.4.4 集合直接量

语法：

〈集合直接量〉::= (1)

〈集合元素名字〉 (1.1)

语义： 集合直接量提供集合值。集合直接量是在一个集合模式中定义的名字。

静态性质： 集合直接量的类是M导出类，其中M是（在给定上下文中的）一个集合模式，它以被指明的集合元素名字为其集合元素的一个名字。

例子：

6.51 dec (1.1)

11.89 king (1.1)

5.2.4.5 空直接量

语法：

〈空直接量〉::= (1)

NULL (1.1)

语义： 空直接量提供空关联值，即不关联于任何地点的值，或提供空过程值，即不表明任何过程的值，或提供空样品值，即不识别任何进程的值。

静态性质： 空直接量的类是空类。

例子：

10.40 NULL (1.1)

5.2.4.6 过程直接量

语法：

〈过程直接量〉::= (1)

〈通用过程名字〉 (1.1)

语义： 过程直接量提供通用过程值。过程直接量是在过程定义或入口定义中定义的名字（见7.4节）。

静态性质： 过程直接量的类是M导出类，其中M是该通用过程名字的模式。

5.2.4.7 字符串直接量

语法：

〈字符串直接量〉::= (1)

′{〈非撇号字符〉|〈撇号〉}*′ (1.1)

|C′{〈十六进数字〉〈十六进数字〉}*′ (1.2)

〈字符〉::= (2)

〈字母〉 (2.1)

|〈数字〉 (2.2)
|〈符号〉 (2.3)
|〈空格〉 (2.4)
〈字母〉::= (3)
A|B|C|D|E|F|G|H|I|J|K|L|M (3.1)
|N|O|P|Q|R|S|T|U|V|W|X|Y|Z (3.2)
〈符号〉::= (4)
_|'|(|)|*|+|,|-|·|/|:|;|〈
|=|>|? (4.1)
〈空格〉::= (5)
SP (5.1)
〈撇号〉::= (6)
'' (6.1)

注：SP标志"空格"字符；见附录A1。

语义： 字符串直接量提供字符串值，它的长度可以为0。长度为1的字符串直接量可作字符值使用。为了在字符串直接量内部书写撇号('')，必须要书写两次才行('')。上述字符组成了必须具备的最小可印刷字符集。一个实现可以允许CCITT第五号字母表中的任何可印刷字符在字符串直接量中出现(见附录A1)。除了可印刷的表示外，十六进表示也可以使用。每一对十六进数字标志一个字符值，该字符值的表示相应于给定的十六进数（见附录A1)。

静态性质： 字符串直接量的长度是非撇号字符和撇号出现的个数，或者是十六进数字数对的个数。

字符串直接量的类是CHAR(n)导出类，其中n是字符串直接量的长度。

例子：

8.18 'A_B<ZAA9K''' (1.1)
8.18 '' (6.1)

5.2.4.8 字位串直接量

语法：

〈字位串直接量〉::= (1)
〈二进位串直接量〉 (1.1)
〈八进位串直接量〉 (1.2)
|〈十六进位串直接量〉 (1.3)
〈二进位串直接量〉::=
B'{0|1|_}*'
〈八进位串直接量〉::=
O'{0|1|2|3|4|5|6|7|_}*'
〈十六进位串直接量〉::=
H'{〈十六进数字〉|_}*'

语义： 字位串直接量提供字位串值，其长度可以为0。可以使用二进、八进或十六进表示法。下划线符号(_)是无意义的，即只用来改进可读性，不影响所指出的值。

静态性质： 字位串直接量的长度是B'后0和1出现的个数，或者是O'后0,1,2,3,4,5,6和7出现的个数的三倍，或者是H'后十六进数字出现的个数的四倍。

字位串直接量的类是B1T(n)导出类，其中n是字位串直接量的长度。

例子：

--- B'101011_110100' (1.1)
--- 0'53_64' (1.2)
--- H'AF4' (1.3)

5.2.5 多元组

语法：

〈多元组〉::= (1)
[〈模式名字〉](:
{〈幂集多元组〉|〈数组多元组〉
|〈结构多元组〉}:) (1.1)
〈幂集多元组〉::= (2)
[{〈表达式〉|〈区段〉}
{,{〈表达式〉|〈区段〉}}*] (2.1)
〈区段〉::= (3)
〈表达式〉:〈表达式〉 (3.1)
〈数组多元组〉::= (4)
〈无标号数组多元组〉 (4.1)
|〈有标号数组多元组〉 (4.2)
〈无标号数组多元组〉::= (5)
〈值〉{,〈值〉}* (5.1)
〈有标号数组多元组〉::= (6)
〈情况标号表〉:〈值〉
{,〈情况标号表〉:〈值〉}* (6.1)
〈结构多元组〉::= (7)
〈无标号结构多元组〉 (7.1)
|〈有标号结构多元组〉 (7.2)
〈无标号结构多元组〉::= (8)
〈值〉{,〈值〉}* (8.1)
〈有标号结构多元组〉::= (9)
〈场名表〉:〈值〉
{,〈场名表〉:〈值〉}* (9.1)
〈场名表〉::= (10)
.〈场名字〉{,.〈场名字〉}* (10.1)

导出语法： 多元组的开括号和闭括号[和]分别是（:和:）的导出语法。这在语法中并未注明，为的是避免与用作元符号的方括号发生混淆。

语义： 多元组提供幂集值、数组值或结构值。

若它是一个幂集值，就由表达式和/或区段的表组成，该表标志幂集值中的成员值。区段标志位于该区段的两个表达式提供的值之间的那些值，或这两个表达式的值本身。如果第二个表达式提供的值小于第一个表达式提供的值，则区段为空，即不标志任何值。幂集多元组可以标志空幂集值。

若它是一个数组值，则是该数组元素值的（可能有标号的）表。在无标号数组多元组中，值是按照元素下标的递增次序给出的。在有标号的数组多元组中，值是针对在（标定此数组值的）情况标号表中指明其下标的那些元素给出的。它可以用作有许多相同值的大型数组多元组的缩写形式。标号ELSE标志所有未被显式提到的下标值
标号* 标志所有下标值（关于进一步的细节见9.1.4节）。

若它是一个结构值，则是该结构场的（可能有标号的）值集。在无标号结构多元组中，值是按照各场在相应的结构模式中被指明的次序给出的。在有标号结构多元组中，值是针对如下一些场给出的，这些场的名字在该结构值的场名表中说明。

一个多元组中表达式和值的加工次序是无定义的，它们可以看作是按混合次序进行加工的。

静态性质： 多元组的类是M值类，其中M是模式名字，如果指明了这样的名字的话。否则，M根据多元组出现处的上下文，并按照下列规则确定：

· 若多元组是在地点说明的初始化中的值或常量值，则M是地点说明中的模式；

- 若多元组是单个赋值动作中右侧的值，则M是左侧地点的（可能为动态的）模式；
- 若多元组是被说明模式的异名定义中的常量值，则M即该模式；
- 若多元组是过程调用中的实在参数，则M是相应参数指明的模式；
- 若多元组是返回动作或结果动作中的值，则M是结果动作或返回动作的过程名字的结果模式（见6.8节）；
- 若多元组是发送动作中的值，则M是信号名字的信号定义中指明的相应模式或缓冲区地点模式的缓冲区元素模式；
- 若多元组是数组多元组中的表达式，则M是该数组多元组的模式的元素模式；
- 若多元组是无标号结构多元组或有标号结构多元组中的表达式，其中相应的场名表仅由一个场名字组成，则此多元组就是为这个结构多元组中的场而指明的，M就是这个场的模式。

当且仅当多元组中出现的每个值或表达式都是常量时，这个多元组也是常量。

静态条件： 只有在上面指明的上下文中才可省去任选的模式名字。根据被指明的是幂集多元组、数组多元组还是结构多元组，必须分别满足下面的相容性条件：

a. 幂集多元组

1. 多元组的模式必须是幂集模式。
2. 每个表达式的类必须和多元组模式的成员模式相容。
3. 每个表达式提供的值必须是由该成员模式定义的一个值。

b. 数组多元组

1. 多元组的模式必须是数组模式。
2. 每个值的类必须和多元组模式的元素模式相容。

在无标号数组多元组的情况下：

3. 值出现的个数必须与多元组的数组模式的元素个数一样多。

在有标号数组多元组的情况下：

4. 各情况选择条件应对各情况标号表出现组成的表成立(见9.1.3节)。结果类必须与多元组的模式的下标模式相容。

5. 每个情况标号表中的每个直接量表达式提供的值，以及每个情况标号表中的每个模式名字定义的那些值都必须位于多元组的模式的下界和上界之间（包括下界和上界本身在内）。

6. 在无标号数组多元组中，至少应有一个值出现是表达式。在有标号数组多元组中，至少有一个非（ELSE）的情况标号表后面的一个值出现是表达式（见5.3.1节）。

7. 在常量（数组）多元组的情况下，如果多元组模式的元素模式是离散模式，则每个被指明的值，只要不是未定义值，均应提供位于元素模式的两个界之间（界本身在内）的值。

c. 结构多元组：

1. 多元组的模式必须是结构模式。
2. 此模式不得为具有不可见的场名字的结构模式（见9.2.7节）。

在无标号结构多元组的情况下：

- 若多元组的模式既非变体结构模式，又非参数化结构模式，则：

3. 值出现的个数应该与多元组的模式中的场名表中的场名字的个数一样多。
4. 每个值的类应与多元组模式中（按位置）对应的场名字的模式相容。

- 若多元组的模式是带标签变体结构模式或带标签参数化结构模式，则：

5. 每个为标签场指明的值必须是一个直接量表达式。
6. 值出现的个数应与实际存在的场名字的个数一样多，这些场名字的存在由标签场指明的直接量表达式出现提供的值所指出。
7. 每个值的类必须与相应的场名字的模式相容。

- 若多元组的模式是无标签变体结构模式或无标签参数化结构模式，则：

8. 不允许有无标号结构多元组。

在有标号结构多元组的情况下：

- 若多元组的模式既非变体结构模式，又非参数化结构模式，则：

9. 多元组的模式的场名表中每个场名字必须在一个场名表中提到一次，且仅提到一次，并且其

次序要与多元组模式中的次序相同。

10．每个值的类必须与标定该值的场名表中指明的场名字的模式相容。

- 若多元组的模式是带标签变体结构模式或带标签参数化结构模式，则

11．为任一标签场指明的值必须是直接量表达式。

12．只有与实际存在的场相对应的场名字才能被指明，这些场的存在由标签场指明的直接量表达式出现的值所指出。场名字被指明的次序必须是它们在多元组的模式中的次序。

13．每个值的类必须与标定该值的场名表中指明的场名字的模式相容。

- 若多元组的模式是无标签变体结构模式或无标签参数化结构模式，则：

14．在场名表提到的场名字中，凡是在相同的选用场中定义的，均必须在相同的变体选择对象中定义，或在ELSE后定义。一个被选中的变体选择对象的所有场名字，或在ELSE后定义的场名字，均必须按它们在多元组的模式中的同一次序被提到一次，且仅提到一次。

15．每个值的类必须与位于该值之前的场名表中指明的任一场名字的模式相容。

16．若多元组的模式是带标签的参数化结构模式，则为各标签场指明的直接量表达式出现提供的值表必须是该多元组的模式的值表。

17．对于常量（结构）多元组，为具有离散模式的场指明的每个值，除了未定义值之外，都必须位于该场的模式的界（界本身在内）之内。

18．至少必须有一个值出现是表达式。

动态条件： 在幂集多元组、数组多元组或结构多元组的情况下，原来关于其成员模式、元素模式或相应的场模式的值的那些赋值条件都仍适用（见6.2.3节，又见a2 a3， b2， c4， c7， c10， c13和c15等条件）。

若多元组具有动态数组模式，则当条件b3或b5不成立时，出现RANGEFAIL异常。

若多元组具有动态参数化结构模式，则当检验C16失败时，出现TAGFAIL异常。

例子：

9.5 *number-list*[ ] (1.1)

9.6 [2:*max*] (2.1)

8.24 [('A'):3,('B','K','Z'):1,(*ELSE*):0] (6.1)

17.5 [(*):''] (6.1)

12.28 (:*NULL*, *NULL*, 536:) (7.1)

11.16 [.*status*:*occupied*, .*p*:[*white*, *rook*]] (9.1)

5.2.6 值串元素

语法：

〈值串元素〉::= (1)

〈串表达式〉(〈定位〉) (1.1)

导出语法： 值串元素是长度为1的值子串的导出语法（见5.2.7节），即

〈串表达式〉(〈定位〉)

是从〈串表达式〉(〈定位〉UP 1）导出的。

5.2.7 值子串

语法：

〈值子串〉::= (1)

〈串表达式〉(〈左元素〉:〈右元素〉) (1.1)

|〈串表达式〉(〈定位〉UP〈串长度〉) (1.2)

注意：若串表达式是一个串地点，则语法结构是二义的，并将被解释为一个子串（见4.2.6节）。

语义： 值子串提供串值，它是被指明串值的一个子值。

静态性质： 如果值子串的串表达式不是强的，则此值子串的类是CHAR（n）导出类或BIT(n) 导出类

(视串表达式为**字位**串表达式或**字符串**表达式而定)，否则是&名字(n)值类，其中n是〈串长度〉,或者是

NUM（右元素）－NUM（左元素）＋1，且&名字是**虚拟的异名模式名字**，它与该串表达式的模式同义。

静态条件： 左元素、右元素和串长度必须提供非负整数值，使下列关系成立：

1. NUM（左元素）≤NUM（右元素）
2. NUM（右元素）≤L-1
3. 1≤NUM(串长度)≤L

其中L是串表达式的类的根模式的**串长度**。(如果串表达式属动态类，则上述检验项2和3只能在运行时进行；见下面。)

动态条件： 值子串提供的值不得是**未定义**的。下列两种情况下将出现RANGEFAIL异常。即如果串表达式属于动态类而上面提到的检验项2和3中有任何一个不成立，或下面两个条件中有任何一个成立：

1. NUM(定位)＜0
2. NUM(定位)+NUM(串长)＞L

其中L是串表达式的类的根模式的串长度。

5.2.8 值串切片

语法：

〈值串切片〉::= (1)

〈串表达式〉(〈起始〉:〈结束〉) (1.1)

注意： 若串表达式是串地点，则语法结构是二义的，并将被解释为串切片（见4.2.13节)。若起始和结束都是整数直接量表达式，则语法结构是二义的，并将被解释为值子串（见5.2.7节)。

语义： 值串切片提供动态串值，它是被指明的串值的一个子值。

静态性质： 值串切片的类的定义方式与值子串的类的定义方式相似（见5.2.7节),但带有动态参数n，其形式是：

NUM(起始)－NUM(结束)＋1

动态条件： 值串切片提供的值不得是**未定义**的。若下列三种关系中有任何一种成立，则出现RANGEFAIL异常：

1. NUM(起始)＞NUM(结束)
2. NUM(起始)＜0
3. NUM(结束)＞L

其中L是串表达式的类的根模式的（可能是动态的）长度。

5.2.9 值数组元素

语法：

〈值数组元素〉::= (1)

〈数组表达式〉(〈表达式表〉) (1.1)

注意：若数组表达式是数组地点，则语法结构是二义的，并将被解释为数组元素(见4.2.7节)。

导出语法： 见4.2.7节。

语义： 值数组元素提供一个值，它是被指明的数组值的一个元素。

静态性质： 值数组元素的类是M值类，其中M是数组表达式的模式的**元素**模式。

静态条件： 数组表达式必须是强的。表达式的类应与数组表达式的模式的**下标**模式相容。

动态条件： 值数组元素提供的值不得是**未定义**的。

若下列两项关系中有任何一项成立，则出现RANGEFAIL异常。

1. 表达式＜L
2. 表达式＞U

其中L和U分别是数组表达式的模式的下界和（可能是动态的）上界。

5.2.10 值子数组

语法：

〈值子数组〉::= (1)

〈数组表达式〉

(〈下元素〉:〈上元素〉) (1.1)

|〈数组表达式〉

(〈整数表达式〉UP〈数组长度〉) (1.2)

注意：若数组表达式是数组地点，则语法结构是二义的，并将被解释为一个子数组（见4.2.8节）。

语义： 值子数组提供一个（子）数组值，它是被指明的数组值的一部分。值子数组的下界等于被指明的数组值的下界；上界根据被指明的（下标）表达式来确定。

静态性质： 值子数组的类是M值类，其中M是参数化数组模式，确定如下：

&名字（上界标）

其中&名字是虚拟的**异名模式**名字，与数组表达式的（可能是动态的）模式同义，上界标是L＋数组长－1（其中L是数组表达式的模式的下界），或是lit，其中lit是直接量，它的类与下元素和上元素的类相容，并且有：

NUM(lit)＝NUM(L)＋NUM(上元素)－NUM(下元素)

静态条件： 数组表达式必须是强的。下元素和上元素的类，或者整数表达式和数组长度的类必须和数组表达式的模式的下标模式相容。下元素、上元素和数组长度提供的值必须使下列关系成立：

1. L≤下元素≤上元素
2. 1≤数组长度
3. 上元素≤U
4. 数组长度≤U－L＋1

其中L和U分别是数组表达式的数组模式的下界和上界。（如果数组表达式有一个动态类，则上述关系3和4只能在运行时检验；见下面。）

动态条件： 由值子数组提供的值不得是**未定义**的。

如果在动态类的情况下，上述关系3或4中有任何一项不成立，或如果下列关系中有任何一项成立，则出现RANGEFAIL异常。

1. L＞整数表达式
2. 整数表达式＋数组长－1＞U

其中L和U分别是数组表达式的模式的下界和上界。

5.2.11 值数组切片

语法：

〈值数组切片〉::= (1)

〈数组表达式〉(〈首〉:〈尾〉) (1.1)

注意：若数组表达式是一个数组地点，则语法结构是二义的，并将被解释为数组切片（见4.2.14节）。如果首和尾都是直接量表达式，则语法结构是二义的，并将被解释为值子数组（见5.2.10节）。

语义： 值数组切片提供一个动态数组值，它是被指明的数组值的子值。

静态性质： 值数组切片的类是M值类，其中M是动态参数化数组模式，其定义方式与值子数组一样（见5.2.10节），但有动态的上界标参数，形式为e×p，其中e×p是表达式，它的类与首和尾的类相容。且

NUM(exp)＝NUM(L)＋NUM(尾)－NUM(首)

静态条件： 数组表达式必须是强的。首和尾的类应该和数组表达式的模式的下标模式相容。

动态条件： 值数组切片提供的值不得是**未定义**的。

若下列关系中有任何一项成立，则将出现RANGEFAIL异常。

1. 首>尾
2. 首<L
3. 尾>U

其中L和U分别标志数组表达式的模式的下界和（可能是动态的）上界。

5.2.12 值结构场

语法：

〈值结构场〉::= (1)

〈结构表达式〉.〈场名字〉 (1.1)

注意：若结构表达式是结构地点，则语法结构是二义的，并将被解释为结构场（见4.2.9节）。

语义： 值结构场提供一个值，它是被指明的结构值的场。若结构表达式具有无标签变体结构模式，且场名字是变体场名字，则语义随实现而定。

静态性质： 值结构场的类是M值类，其中M是场名字的模式。

静态条件： 结构表达式必须是强的。场名字必须属于结构表达式的模式的场名字集合。

动态条件： 值结构场提供的值不得是未定义的。

若有下列情况之一出现，则出现TAGFAIL异常：

- 结构表达式具有带标签变体结构模式，且相应的（诸）标签场的值指出被标志的场并不存在；
- 结构表达式具有动态参数化结构模式，且相应的值表指出该场并不存在。

例子：

16.49 (RECEIVE USER_BUFFER).ALLOCATOR (1.1)

5.2.13 被关联地点

语法：

〈被关联地点〉::= (1)

—>〈地点〉 (1.1)

语义： 若被指明的地点是可关联的，则被关联地点提供对该地点的一个关联。否则，它提供一个不可非关联化的关联值（见4.2.4节），它可以关联于一个由实现定义的地点。

静态性质： 若地点是可关联的，被关联地点的类是一个M关联类，其中M是该地点的模式。否则，被关联地点的类是PTR导出类。当且仅当地点是静态时，被关联地点是常量。

例子：

8.23 →C (1.1)

5.2.14 表达式转换

语法：

〈表达式转换〉::= (1)

〈模式名字〉(〈表达式〉) (1.1)

语义： 表达式转换可使CHILL的模式检验和相容性规则失效。它显式地为表达式指定一个模式。表达式转换的精确动态语义由实现定义，并依赖于值的内部表示。

静态性质： 表达式转换的类是M值类，其中M是模式名字。当且仅当表达式是常量时，表达式转换也是常量。

静态条件： 表达式不得具有动态类。表达式的类必须至少与一个其大小和模式名字的大小相同的模式相容。模式名字不得有同步性质。

5.2.15 值过程调用

语法：

〈值过程调用〉::= (1)
〈值过程调用〉 (1.1)

语义： 值过程调用提供由过程返回的值。

静态性质： 值过程调用的类是M值类，其中M是值过程调用的结果说明的模式。

动态条件： 值过程调用不得提供未定义的值（见5.3.1和6.3节）。

例子：

6.51 julian__day__number([10, dec,1979]) (1.1)
11.68 OK__bishop(b, m) (1.1)

5.2.16 值内部子程序调用

语法：

〈值内部子程序调用〉::= (1)
〈实现值内部子程序调用〉 (1.1)
|〈CHILL值内部子程序调用〉 (1.2)
〈CHILL值内部子程序调用〉::= (2)
NUM(〈离散表达式〉) (2.1)
|PRED(〈离散表达式〉) (2.2)
|PRED(〈约束关联表达式〉) (2.3)
|SUCC(〈离散表达式〉) (2.4)
|SUCC(〈约束关联表达式〉) (2.5)
|ABS (〈整数表达式〉) (2.6)
|ADDR(〈地点〉) (2.7)
|CARD(〈幂集表达式〉) (2.8)
|MAX (〈幂集表达式〉) (2.9)
|MIN (〈幂集表达式〉) (2.10)
|SIZE({〈模式名字〉|〈静态模式地点〉}) (2.11)
|UPPER({〈数组表达式〉|〈串表达式〉}) (2.12)
|GETSTACK(〈取栈顶变元〉) (2.13)
〈取栈顶变元〉::= (3)
〈模式名字〉 (3.1)
|〈数组模式名字〉(〈表达式〉) (3.2)
|〈串模式名字〉(〈整数表达式〉) (3.3)
|〈变体结构模式名字〉(〈表达式表〉) (3.4)

导出语法： ADDR(〈地点〉)是→〈地点〉的导出语法。

语义： 位内部子程序调用可以是由实现定义的内部子程序调用，也可以是由CHILL定义的内部子程序调用。它提供一个值。CHILL值内部子程序调用也就是对CHILL定义的内部子程序的引用，它提供一个值。

NUM提供一个整数值，它与离散变元提供的值有相同的内部表示。对于集合值，NUM按照集合模式的规定提供整数值。对于字符值，NUM按照CCITT第5号字母表（见附录A1）的规定提供整数值。NUM（TRUE）的值为1，NUM（FALSE）的值为0。对于整数值，NUM提供的值就是该整数值。

对于离散值，PRED和SUCC分别提供下一个和上一个离散值，如果它们是存在的。否则将发生异常。若该离散值是带孔集合模式中的集合值，则应跳过这些孔（即在3.4.5节的静态性质的例

子中，SUCC(A)的值为B，而PRED(B)的值为A)。

对于约束关联值，PRED和SUCC仅对关联于数组元素的关联值有定义。它们分别提供关联于下一个和上一个数组元素的关联值，条件是这些元素应该存在。

ABS在整数值上有定义，并提供该整数值的绝对值。

ADDR是关联地点的另一种表示法。

CARD，MAX和MIN在幂集值上有定义。

CARD提供该幂集值中元素值的个数。MAX和MIN分别提供该幂集值中最大和最小的元素值。

SIZE在可关联的静态模式地点和可关联的静态模式上有定义。在第一种情况下，它提供被此地点占用的可访问存储单元的个数。在第二种情况下，它提供具有该模式的可关联地点占用可访问存储单元的个数。在第一种情况下，静态模式地点不在运行时加工。

UPPER在（可能是动态的）数组值和串值上有定义，它提供该数组值的上界下标，或该串值的最高串下标（即串的长度减1）。

GETSTACK在栈上建立一个具有指定模式的地点（见7.9节），并对被建立的地点提供关联值。如果指明模式名字，则建立的是具有该模式的静态模式地点，并提供一个约束关联值。否则，建立的是动态模式地点，它的模式是具有运行时参数的参数化模式，这些参数在GETSTACK变元中指明，并提供关联于该地点的一个行值。

静态性质 NUM内部子程序调用的类是INT导出类。当且仅当其变元是**常量**（**直接量**）时，该内部子程序调用也是**常量**（**直接量**）。

PRED或SUCC内部子程序调用的类是其变元的类。当且仅当其变元是**常量**（**直接量**）时，该内部子程序调用也是**常量**（**直接量**）。

ABS内部子程序调用的类就是变元的类。当且仅当其变元是**常量**（**直接量**）时，该内部子程序调用也是**常量**（**直接量**）。

CARD内部子程序调用的类是INT导出类。当且仅当其变元是**常量**时，该内部子程序调用也是**常量**。

MAX或MIN内部子程序调用的类是一个M值类，其中M是幂集表达式模式的成员模式。当且仅当其变元是**常量**时，该内部子程序调用也是**常量**。

SIZE内部子程序调用的类是INT导出类。该内部子程序调用是一个**常量**。

若UPPER内部子程序调用的变元是数组表达式，则UPPER内部子程序调用的类是M值类，其中M是该（强）数组表达式的数组模式的**下标模式**。若UPPER内部子程序调用的变元是串表达式，则该内部子程序调用的类是INT导出类。当且仅当数组表达式或串表达式的类是静态类时，UPPER内部子程序调用是**常量**和**直接量**。

GETSTACK内部子程序调用的类是M关联类，其中M是模式名字或动态参数化模式，形如：

&＜数组模式名字＞（〈表达式〉），或

&〈串模式名字〉(〈整数表达式〉)，或

&〈变体结构模式名字〉（〈表达式表〉），

依取栈顶变元而定。

静态条件 若PRED或SUCC内部子程序调用的变元是**常量**，则不得提供由变元的类的根模式所定义的最小或最大离散值。

若MAX或MIN内部子程序调用的变元是**常量**，则不得提供空的幂集值。

作为CARD、MAX或MIN内部子程序调用的变元时，幂集表达式必须是强的。

作为PRED或SUCC内部子程序调用的变元时，约束关联表达式必须是强的。

作为UPPER内部子程序调用的变元时，数组表达式必须是强的。

若取栈顶变元不只是一个模式名字，则下列相容性要求必须满足：

- 表达式的类必须与数组模式名字的下标模式相容。
- 表达式表中的表达式个数必须与变体结构模式名字的类表中类的个数相等，并且每个表达式的类必须与变体结构模式名字的类表中相应的类相容。

动态条件： 若把PRED和SUCC应用于由变元的类的根模式所定义的最小或最大离散值，则将引起OVERFLOW异常。若把PRED和SUCC应用于一个关联值，而该关联值又是关联于具有最小或最大下标的数组元素，则将引起RANGEFAIL异常。若约束关联表达式的值为NULL，则PRED和SUCC将引起EMPTY异常。

若把MAX和MIN应用于空的幂集值（即不包含元素组），则将引起EMPTY异常。

若存储要求不能得到满足，则GETSTACK将引起SPACEFAIL异常。

若在取栈顶变元中出现下列情况，则GETSTACK引起RANGEFAIL异常：

- 表达式提供的值不属于由数组模式名字的下标模式定义的值的集合；
- 整数表达式提供一个负值，或其值大于串模式名字的长度；
- 表达式表中任一表达式，它在变体结构模式名字的类表中相应的类是M值类（即强的），此表达式提供的值不属于由M定义的值的集合。

若ABS的结果值超出由变元的类的根模式定义的界之外，则引起OVERFLOW异常。

例子：

9.11 MIN (sieve) (2.10)

11.91 PRED (col-1) (2.2)

11.91 SUCC(col-1) (2.4)

5.2.17 开动表达式

语法：

〈开动表达式〉::= (1)

START〈进程名字〉〔〔〈实在参数表〉〕〕 (1.1)

语义：

加工一个开动表达式将建立并活化一个新进程，它的定义由进程名字所指出（见第8章）。参数传递与过程参数传递相似；但可给出额外的实在参数，其意义由实现定义。开动表达式提供一个唯一的样品值，此值确认被建立的进程。

静态性质： 开动表达式的类是INSTANCE导出类。

静态条件： 实在参数表中实在参数出现的个数不得少于进程名字的进程定义的形式参数表中形式参数出现的个数。若实在参数有m个，形式参数有n个(m≥n)，则对头n个实在参数的相容性要求与对过程参数传递的要求相同（见6.7节）。

动态条件： 开动表达式可以引发任何由实现定义的异常，但异常的名字应附属于进程名字之上(见7.5节)。

对于参数传递，在任何实在值及其相应的形式参数的模式之间适用赋值条件（见6.7节）。

若存储要求不能满足，则开动表达式引起SPACEFAIL异常。

例子：

15.25 START COUNTER ( ) (1.1)

5.2.18 接收表达式

语法：

〈接收表达式〉::= (1)

RECEIVE〈缓冲区地点〉 (1.1)

语义： 接收表达式提供的值来自被指明的缓冲区或任何被延迟的发送进程。若执行一个接收表达式时缓冲区中没有值或不存在被延迟的发送进程，则正在执行的进程将被延迟，直到有一个值被送到缓冲区为止(有关细节请参见第8章)。

静态性质： 接收表达式的类是M值类，其中M是缓冲区地点的模式的缓冲区元素模式。

动态条件： 当正在执行的进程在缓冲区地点上被延迟时，该缓冲区地点的生存期不得中止。

例子：

16.49 RECEIVE USER-BUFFER (1.1)

5.2.19 零目运算符

语法：

〈零目运算符〉::= (1)

THIS (1.1)

语义： 零目运算符提供唯一的样品值，此值确认正在执行该运算符的进程。

静态性质： 零目运算符的类是INSTANCE——导出类。

5.3 值和表达式

5.3.1 概述

语法：

〈值〉::= (1)

〈表达式〉 (1.1)

| 〈未定义值〉 (1.2)

〈未定义值〉::= (2)

* (2.1)

| 〈未定义异名名字〉 (2.2)

语义： 一个值或是**未定义值**，或者是作为表达式的加工结果的（CHILL定义的）值。

静态性质： 值的类就是表达式或未定义值的类。

若未定义值是*，则它的类是**完全**类，否则就是未定义异名名字的类。

当且仅当一个值是未定义值，或等于**常量**表达式时，该值也是**常量**。

动态性质： 若一个值被未定义值所标志，或在本文本中被显式地指出是**未定义的**，则该值是**未定义的**。当且仅当组合值的所有子分量（即子串值、元素值、场值）都是**未定义的**时，该值本身也是**未定义的**。

（注意：只在下列情况下一个值才可以标志一个**未定义的值**：

- 它自己是一个未定义值；
- 它是一个地点内容，包含一个**未定义的值**；
- 它是一个值过程调用，提供一个**未定义的值**；
- 它是一个值子串、一个值串切片、一个值数组元素、一个值子数组、一个值数组切片或一个值结构场，并提供一个**未定义的值**。）

例子：

```
6.40  (146_097*C)/4+(1_461*y)/4
  +(153+m+C)/5+day+1_721_119                    (1.1)
```

5.3.2 表达式

语法：

〈表达式〉::= (1)

〈运算数1〉 (1.1)

| 〈子表达式〉{OR | XOR}〈运算数1〉 (1.2)

〈子表达式〉::= (2)

〈表达式〉 (2.1)

语义： 表达式的各组成部分以及它们的子组成部分等的计算次序是未定义的，并且可以认为是以混合的次序计算的。对计算次序的唯一要求是所提供的值应是唯一确定的。如果表达式是**常量**或**直接量**，则在计算时绝不会引发异常。

如果指明了OR或XOR，则子表达式和运算数1提供的值可以是：

- 布尔值，此时OR和XOR表示通常的逻辑运算符且提供一个布尔值；
- 字位串值，此时OR和XOR表示通常的作用于字位串的逻辑运算且提供一个字位串值；
- 幂集值，此时OR表示两个幂集值的联合，而XOR表示这样的一个幂集值，它的成员正好是被指明的两个幂集值中那些不同时属于两个幂集值的成员(例如，A XOR B = A-B OR B-A)。

静态性质： 若表达式是运算数1，则该表达式的类就是运算数1的类。若指明了OR或XOR，则表达式的类就是子表达式和运算数1的类的结果类。

当且仅当表达式是**常量（直接量）**运算数1，或由双双都是**常量（直接量）**的表达式和运算数1构成时，此表达式也是**常量（直接量）**。

静态条件： 如果指明了OR或XOR，子表达式的类必须与运算数1的类相容。这两个类都应具有布尔、幂集或位串根模式。

动态条件： 在指明OR或XOR的情况下，如果其中的一个或两个运算数具有动态类，且上面提到的相容性检验的动态部分失败，则将出现RANGEFAIL异常。

例子：

```
10.27  i<min                                    (1.1)
10.27  i<min OR i>max                           (1.2)
```

5.3.3 运算数1

语法：

```
〈运算数1〉::=                                   (1)
    〈运算数2〉                                  (1.1)
  | 〈子运算数1〉AND〈运算数2〉                   (1.2)
〈子运算数1〉::=                                 (2)
    〈运算数1〉                                  (2.1)
```

语义： 如果指明了AND，则子运算数1和运算数2提供的值可以是：

- 布尔值，此时AND表示通常的逻辑“与”运算且提供一个布尔值；
- 字位串值，此时AND表示通常的作用于字位串的“与”运算且提供一个字位串值；
- 幂集值，此时AND表示幂集值的交运算且提供一个幂集值作为结果。

静态性质： 若运算数1同时又是运算数2，则运算数1的类就是运算数2的类。

如果指明了AND，运算数1的类是运算数2和子运算数1的类的结果类。

当且仅当运算数1是**常量（直接量）**运算数2，或由双双都是**常量（直接量）**的运算数1和运算数2构成时，它自己也是**常量（直接量）**。

静态条件： 如果指明了AND，则子运算数1的类必须和运算数2的类相容。这两个类必须都具有布尔、幂集或**字位**串根模式。

动态条件： 在指明了AND的情况下，如果一个或两个运算数具有动态类，且上面提到的相容性检验失败，则将出现RANGEFAIL异常。

例子：

```
5.11  (a1 OR b1)                                (1.1)
5.11  NOT k2 AND (a1 OR b1)                     (1.2)
```

5.3.4 运算数2

语法：

```
〈运算数2〉::=                                   (1)
    〈运算数3〉                                  (1.1)
  | 〈子运算数2〉〈运算符3〉〈运算数3〉           (1.2)
〈子运算数2〉::=                                 (2)
```

〈运算数2〉 (2.1)

〈运算符3〉::= (3)

〈关系运算符〉 (3.1)

| 〈成员运算符〉 (3.2)

| 幂集蕴含运算符〉 (3.3)

〈关系运算符〉::= (4)

=|/=|>|>=|<|<= (4.1)

〈成员运算符〉::= (5)

IN (5.1)

〈幂集蕴含运算符〉::= (6)

〈=|>=|<|〉 (6.1)

语义： 相等（=）和不等（/=）运算符在同一模式的所有值之间皆有定义。其它的关系运算符(小于：<，小于等于：<=，大于：>，大于等于：>=）在同一离散模式或串模式的值之间有定义。所有的关系运算符都提供一个布尔值作为结果。

成员运算符在成员值和幂集值之间有定义。若成员值属于所指明的幂集值，则该运算将提供TRUE，否则提供FALSE。

幂集蕴含运算符在幂集值之间有定义，它检查集合值是否包含在：<=，真包含在：<，包含：>=，或真包含：>另一个集合值。幂集蕴含运算符提供布尔值作为结果。

静态性质： 若运算数2同时又是运算数3，则运算数2的类就是运算数3的类。如果指明了运算数3，则运算数3的类是BOOL——导出类。

当且仅当运算数2是**常量**（**直接量**）运算数3，或由双双都是**常量**（**直接量**）的运算数2和运算数3构成时，它自己也是**常量**（**直接量**）。

静态条件： 如果指明了运算符3，则在子运算数2的类和运算数3的类之间的下列相容性要求必须满足：

- 若运算符3是=或/=，两个类应相容；
- 若运算符3是关系运算符，但不是=或/=，两个类必须相容，并必须具有一个离散或串根模式；
- 若运算符3是成员运算符，则运算数3的类必须具有幂集根模式，且子运算数2的类必须与该根模式的成员模式相容；
- 若运算符3是幂集蕴含运算符，则两个类必须相容，并必须具有幂集根模式。

动态条件： 在关系运算符的情况下，如果一个或两个运算符具有动态类，且上面提到的相容性检验的动态部分失败，则将出现RANGE—FAIL或TAGFAIL异常。当且仅当一个动态类是基于一个动态参数化结构模式之上时，出现的是TAGFAIL异常。

例子：

10.46 NULL (1.1)

10.46 last=NULL (1.2)

5.3.5 运算数3

语法：

〈运算数3〉::= (1)

〈运算数4〉 (1.1)

| 〈子运算数3〉〈运算符4〉〈运算数4〉 (1.2)

〈子运算数3〉::= (2)

〈运算数3〉 (2.1)

〈运算符4〉::= (3)

〈算术加减运算符〉 (3.1)

| 〈串连接运算符〉 (3.2)

| 〈幂集求差运算符〉 (3.3)

〈算术加减运算符〉::= (4)

+ | − (4.1)

〈串连接运算符〉::= (5)

// (5.1)

〈幂集求差运算符〉::= (6)

— (6.1)

语义： 如果运算符4是一个算术加减运算符，则两个运算数都提供整数值，且结果整数值是该两个值的和（+）或差（−）。

如果运算符4是一个串连接运算符，则两个运算数提供的是字位串值或字符串值；结果值由这两个值连接而成。

如果运算符4是幂集求差运算符，则两个运算数提供的都是幂集值，结果值也是一个幂集值，其成员值就是所有包含在由子运算数3提供的值中，但是不包含在由运算数4提供的值中的那些成员值。

静态性质： 若运算数3同时又是运算数4，则运算数3的类就是运算数4的类。如果指明运算符4，则运算数3的类根据运算符4按下述方式确定：

- 若运算符4是串连接运算符，则根据运算数4和子运算数3的类，可以确定运算数3的类如下：

— 若它们全不是强的，则该类是BIT (n) 导出类或CHAR(n) 导出类，按两个运算数是**字位**串或**字符**串而定，其中n是两个类的根模式的长度的和。

— 否则，该类是&名字(n)值类，其中&名字是一个虚拟**异名模式**名字，它与强运算数之一的模式同义，n表示两个类的根模式的长度的和。

（若一个或两个运算数具有动态类，则此类也是动态的。）

- 若运算数4是算术加减运算符或幂集求差运算符，则运算数3的类是运算数4和子运算数3的类的结果类。

当且仅当一个运算数3是一个**常量（直接量）**运算数4，或由双双都是**常量（直接量）**的一个运算数3和一个运算数4构成时，它自己也是**常量（直接量）**。

静态条件： 如果指明运算符4，则下列相容性要求必须得到满足：

- 若运算符4是算术加减运算符，则两个运算数的类必须相容，并且它们都必须具有整数根模式；
- 若运算符4是串连接运算符，则两个运算数的类的根模式必须都是**字位**串模式或都是**字符**串模式，并且，如果两个类都是值类，它们的根模式必须具有相同的新鲜性；
- 若运算符4是幂集求差运算符，则两个运算数的类必须相容，且都必须具有幂集根模式。

动态条件： 在运算数3不是**常量**的情况下，如果加法（+）或减法(−)的结果产生一个值，它不在运算数3的类的根模式指明的界限内，则将出现OVERFLOW异常。

例子：

1.5 j (1.2)

1.5 i+j (1.2)

5.3.6 运算数4

语法：

〈运算数4〉::= (1)

〈运算数5〉 (1.1)

| 〈子运算数4〉

〈算术乘除运算符〉〈运算数5〉 (1.2)

〈子运算数4〉::= (2)

〈运算数4〉 (2.1)

〈算术乘除运算符〉::= (3)

* | / | MOD | REM (3.1)

语义： 如果指明了算术乘除运算符，则子运算数4和运算数5提供整数值，而结果整数值是两个值的乘积

(*)、商(/)、模(MOD)或余数(REM)。

模运算的定义是：I MOD J提供唯一的整数值K，0≤K<J，使得有一个整数值N，使I = N*J+K。J必须大于0。

余数运算的定义是：X REM Y = X-(X/Y)*Y对所有的整数值X和Y提供TRUE。

静态性质： 若运算数4同时又是运算数5，则运算数4的类就是运算数5的类，否则，运算数4的类是子运算数4和运算数5的类的结果类。

当且仅当一个运算数4是**常量**（**直接量**）运算数5，或由双双都是**常量**（**直接量**）的一个运算数4和一个运算数5构成时，它本身也是**常量**（**直接量**）。

静态条件： 如果指明了算术乘除运算符，则运算数5和子运算数4的类必须相容，并且都必须具有整数根模式。

动态条件： 在运算数4不是**常量**的情况下，如果乘法(*)、除法(/)、求模（MOD）或求余数(REM)运算的结果产生了一个值，它不在运算数4的类的根模式所定义的值的集合中，或者这些运算施行于某些运算符的值上，对于它们来说，该运算符是数学上无定义的，即对值为0的运算数5作除法和求余运算，或对值为非正整数的运算数5作模运算，则将出现OVERFLOW异常。

例子：

```
6.15  1_461                                    (1.1)
6.15  (4*d+3)/1_461                            (1.2)
```

5.3.7 运算数5

语法

```
〈运算数5〉::=                                  (1)
〔〈单目运算符〉〕〈运算数6〉                    (1.1)
〈单目运算符〉::=                               (2)
    -|NOT                                      (2.1)
   |〈串重复运算符〉                            (2.2)
〈串重复运算符〉::=                             (3)
    (〈整数直接量表达式〉)                      (3.1)
```

语义： 若单目运算符是变号运算符(-)，则运算数6提供一个整数值，而结果整数值等于改了正负号以后的先前的整数值。

若单目运算符是NOT，则运算数6提供一个布尔值，或一个字位串值，或一个幂集值。在头两种情况下，得到的是布尔值或字位串值的逻辑非。在最后一种情况下，得到的是集合补元素，即由那些不包含在运算数幂集值中的成员值构成的集合。

若单目运算符是串重复运算符，则运算数6是一个字符串直接量或一个字位串直接量。若整数直接量表达式的值为0，则结果是空串值。否则是由该串自己连接多次得到的串值，连接的次数是直接量表达式提供的值减1。

静态性质： 如果运算数5同时又是运算数6，则运算数5的类就是运算数6的类。

如果指明单目运算符，则运算数5的类可以确定如下：

- 若单目运算符是-或NOT，则该类是运算数6的结果类；
- 若单目运算符是串重复运算符，则该类是CHAR(n)或BIT(n)导出类（取决于直接量是字符串直接量或字位串直接量），其中n=r*L，r是整数直接量表达式提供的值，L是串直接量的**长度**。

当且仅当运算数6是**常量**（**直接量**）时，运算数5也是**常量**（**直接量**）。

静态条件： 若单目运算符是-，则运算数6的类必须具有整数根模式。

若单目运算符是NOT，则运算数6的类必须具有布尔、**字位**串或幂集根模式。

若单目运算符是串重复运算符，则运算数6必须是字符串直接量或字位串直接量。整数直接量表达式必须提供非负整数值。

动态条件： 若运算数5不是**常量**，而变号（-）运算产生一个值，它不在由运算数5的类的根模式定义的值

的集合中，则将出现OVERFLOW异常。

例子：

```
5.11  NOT  k2                    (1.1)
7.50  (6)''                      (1.1)
7.50  (6)                        (2.2)
```

5.3.8 运算数6

语法：

```
〈运算数6〉::=                    (1)
    〈原值〉                      (1.1)
  | 〈加括号表达式〉              (1.2)
〈加括号表达式〉::=               (2)
    (〈表达式〉)                 (2.1)
```

语义： 运算数6是一个原值（见5.2节），或是一个加括号表达式。

静态性质： 运算数6的类分别是原值的类或加括号表达式的类。加括号表达式的类是表达式的类。当且仅当原值或表达式分别是**常量（直接量）**时，运算数6也是**常量（直接量）**。

例子：

```
1.5  i                           (1.1)
5.11  (a1  OR  b1)               (1.2)
```

6.0 动作

6.1 概述

语法：

〈动作语句〉 ::= (1)
〔〈名字〉:〕〈动作〉〔〈处理程序〉〕
〔〈标号名字〉〕； (1.1)
〈动作〉 ::= (2)
〈加括号动作〉 (2.1)
| 〈赋值动作〉 (2.2)
| 〈调用动作〉 (2.3)
| 〈出口动作〉 (2.4)
| 〈返回动作〉 (2.5)
| 〈结果动作〉 (2.6)
| 〈转向动作〉 (2.7)
| 〈断言动作〉 (2.8)
| 〈空动作〉 (2.9)
| 〈开动动作〉 (2.10)
| 〈停止动作〉 (2.11)
| 〈延迟动作〉 (2.12)
| 〈继续动作〉 (2.13)
| 〈发送动作〉 (2.14)
| 〈引发动作〉 (2.15)
〈加括号动作〉 ::= (3)
〈条件动作〉 (3.1)
| 〈情况动作〉 (3.2)
| 〈循环动作〉 (3.3)
| 〈模块〉 (3.4)
| 〈分程序〉 (3.5)
| 〈延迟情况动作〉 (3.6)
| 〈接收情况动作〉 (3.7)

语义： 动作语句组成CHILL程序的算法部分。每个动作语句都可加标号。凡是能引发异常动作的皆可附带一个处理程序。

静态性质： 凡位于动作之前，并且后面跟一个冒号的名字(且只有这种名字)，定义为标号名字。

静态条件： 只有当动作是加括号动作，或已经指明处理程序且在动作之前有一个名字，而名字后又有冒号时，分号前才允许有标号名字。此标号名字必须和上述动作之前的名字相同。

6.2 赋值动作

语法：

〈赋值动作〉 ::= (1)
〈单个赋值动作〉 (1.1)
| 〈多重赋值动作〉 (1.2)
〈单个赋值动作〉 ::= (2)

〈地点〉{〈赋值符号〉 | 〈赋值运算符〉}
〈值〉 (2.1)

〈多重赋值动作〉::= (3)
〈地点〉{,〈地点〉}+〈赋值符号〉
〈值〉 (3.1)

〈赋值运算符〉::= (4)
〈闭式双目运算符〉〈赋值符号〉 (4.1)

〈闭式双目运算符〉::= (5)
OR | *XOR* | *AND* (5.1)
| 〈幂集求差运算符〉 (5.2)
| 〈算术加减运算符〉 (5.3)
| 〈算术乘除运算符〉 (5.4)

〈赋值符号〉::= (6)
:= | = (6.1)

导出语法： =符号是:= 符号的导出语法。

语义： 赋值动作把一个值存进一个或多个地点中。

如果使用赋值符号，则右边提供的值被存进由左边指明的（诸）地点中。

如果使用赋值运算符，则地点中的值与右边的值（按此次序）按照被指明的闭式双目运算符的语义结合起来，而其结果存回到同一地点中去。

左边（诸）地点的计算和右边的值的计算以及它们的赋值是以一种未经说明的，可能是混合的次序进行的。任何赋值，只要值和地点已经计算出来，即可实施。

若地点（或诸地点之一）是变体结构的标签场，则依赖它的诸变体场将得到一个**未定义的值**。

静态条件： 所有地点出现的模式必须是等价的，它们不得有只读性，也不得有同步性。每个模式必须与值的类相容。在涉及动态模式地点与/或具有动态类的值的情况下，检验也是动态的。

如果值是一个区域表达式（见3.2.2节），则每个地点也必须是**区域性的**。

如果在单个赋值动作中指明了赋值运算符，则被指明的值必须是表达式。

动态条件： 在动态参数化结构模式地点与/或值的情况下,如果上面提到的相容性检验的动态部分失败,则出现TAGFAIL异常。

如果任一地点具有区段模式，且由值计算所得的值位于区段模式指明的界限之外,则将出现RANGEFAIL异常。

在动态参数化串模式或数组模式地点与/或值的情况下，如果上面提到的相容性检验的动态部分失败，则将出现RANGEFAIL异常。

上面提到的条件称为值相对于模式（即地点模式）的赋值条件。

在赋值运算符的情况下，被引发的异常与计算表达式。

〈地点〉〈闭式双目运算符〉(〈表达式〉)

并把所得的值存进被指明的地点时引发的异常一样（注意，该地点只计算一次）。

例子：

```
4.11  a:=b+c                                  (1.1)
10.21 stackindex-:=1                          (2.1)
19.16 X.PREX, X.NEXT:=NULL                    (3.1)
10.21 -:=                                     (4.1)
```

6.3 条件动作

语法：

〈条件动作〉::= (1)
IF 〈布尔表达式〉〈则子句〉
〔〈否则子句〉〕FI (1.1)

〈则子句〉::= (2)

THEN〈动作语句表〉 (2.1)

〈否则子句〉::= (3)

ELSE〈动作语句表〉 (3.1)

|ELSIF〈布尔表达式〉

〈则子句〉〔〈否则子句〉〕 (3.2)

导出语法： 记号

ELSIF〈布尔表达式〉〈则子句〉〔〈否则子句〉〕是

ELSE IF〈布尔表达式〉〈则子句〉〔〈否则子句〉〕FI；的导出语法。

语义： 条件动作有两个条件分枝。若布尔表达式提供TRUE，则执行THEN后面的动作语句表，否则执行ELSE后面的动作语句表（若有的话）。

例子：

```
7.24  IF n>=10  THEN rn(r):='x';
          n-:=10;
          r+:=1;
      FI                                          (1.1)
10.46 IF last=NULL
      THEN first,last:=P;
      ELSE last->.succ:=P;
      P->.pred:=last;
      last:=P;
      FI                                          (1.1)
```

6.4 情况动作

语法：

〈情况动作〉::= (1)

CASE〈情况选择表〉OF〔〈区段表〉；〕

{〈情况选择对象〉}+

〔ELSE〈动作语句表〉〕

ESAC (1.1)

〈情况选择表〉::= (2)

〈离散表达式〉{,〈离散表达式〉}* (2.1)

〈区段表〉::= (3)

〈离散模式〉{,〈离散模式〉}* (3.1)

〈情况选择对象〉::= (4)

〈情况标号说明〉:〈动作语句表〉 (4.1)

语义： 情况动作有多重分枝。它首先指明一个或多个离散表达式(情况选择表)，然后是一组带标号的动作语句表（情况选择对象)。每个动作语句表前面有情况标号说明，这是由一些情况标号表说明组成的表(每个情况选择对应一个情况标号表说明)。每个情况标号表定义一组值。利用情况选择表中离散表达式表即可根据多重条件选出一个选择对象。

哪个情况标号说明中给定的值与情况选择表中的值相匹配，情况动作即进入与该情况标号说明相对应的动作语句表。

情况选择表中表达式的计算次序是未定义的，并可能是混合的。对计算次序的唯一要求是：必须唯一地确定一个情况选择对象。

静态条件： 情况选择条件适用于由情况标号说明的出现组成的表（见9.1.3节)。

情况选择表中离散表达式出现的个数必须与情况标号表的出现组成的表中的结果类表中类的个数相等。并且，如果有区段表的话还必须与区段表中离散模式出现的个数相等。

情况选择表中任一离散表达式的类必须与情况标号表出现的结果类表中（按位置）对应的类相容，并且，如有区段表的话，还必须与区段表中（按位置）对应的离散模式相容。该离散模式还必须与结果类表中相应的类相容。

如果有区段表，则在情况标号中由离散直接量表达式提供的值或由直接量区段或离散模式定义的值（见9.1.3节）必须位于区段表中相应离散模式的区段内。如果是强离散表达式，则其值还必须位于情况选择表中由相应离散表达式的模式定义的区段内。在后一种情况下，如果有区段表的话，由区段表中相应的离散模式定义的值还必须位于该区段中。

只有当情况标号表组成的表是完全时，任选关键词ELSE及其后面的动作语句表才可省去（见9.1.3节）。

动态条件： 如果指明区段表，而情况选择表中由离散表达式提供的值又不在区段表中由相应离散模式指明的界限内，则将出现RANGEFAIL异常。

例子：

```
4.10  CASE  order OF
      (1):a:=b+c;
          RETURN;
      (2): d=0;
           (ELSE) : d:=1;
           ESAC                                          (1.1)
11.44  Starting.P.kind, starting.p.color                 (2.1)
11.62  (rook),(*):
       IF NOT ok-rook(b,m)
          THEN
          CAUSE:illegal;
       FI;                                               (4.1)
```

6.5 循环动作

6.5.1 概述

语法：

```
〈循环动作〉::=                                           (1)
   DO〔〈控制部分〉；〕〈动作语句表〉OD                    (1.1)
〈控制部分〉::=                                           (2)
   〈步长型控制〉〔〈当型控制〉〕                          (2.1)
  |〈当型控制〉                                          (2.2)
  |〈结构型部分〉                                        (2.3)
```

语义： 循环动作有三种不同的形式：do-for形式和do-while形式用于通常循环，do-with 形式是以有效的方式访问结构场的一种方便的简化表示法。如果未指明控制部分，则每当进入循环动作时，动作语句表只进入一次。

如果把do—for和do—while两种形式结合起来，则在步长型控制之后才计算当型控制，且仅当循环动作未被步长型控制结束时才进行这种计算。

动态条件： 若存储要求不能满足，则将出现SPACEFAIL异常。

例子：

```
4.16  DO  FOR::=1  TO  c;
          OP (a,b,d, order-1);
          d:=a;
      OD                                                 (1.1)
```

```
15.48  DO  WITH  EACH;
       IF  THIS-COUNTER=COUNTER
           THEN
            STATUS:=IDLE;
           EXIT FIND-COUNTER;
       FI;
       OD                                                    (1.1)
```

6.5.2 步长型控制

语法：

〈步长型控制〉::= (1)

FOR〔〈重复〉{,〈重复〉}* | EVER〕 (1.1)

〈重复〉::= (2)

〈值枚举〉 (2.1)

| 〈地点枚举〉 (2.2)

〈值枚举〉::= (3)

〈步枚举〉 (3.1)

| 〈区段枚举〉 (3.2)

| 〈幂集枚举〉 (3.3)

〈步枚举〉::= (4)

〈循环计数器〉〈赋值符号〉

〈初值〉〔〈步值〉〕〔DOWN〕〈终值〉 (4.1)

〈循环计数器〉::= (5)

〈名字〉 (5.1)

〈初值〉::= (6)

〈表达式〉 (6.1)

〈步值〉::= (7)

BY〈整数表达式〉 (7.1)

〈终值〉::= (8)

TO〈表达式〉 (8.1)

〈区段枚举〉::= (9)

〈循环计数器〉〔DOWN〕IN〈离散模式〉 (9.1)

〈幂集枚举〉::= (10)

〈循环计数器〉〔DOWN〕IN〈幂集表达式〉 (10.1)

〈地点枚举〉::= (11)

〈循环计数器〉〔DOWN〕IN〈数组地点〉 (11.1)

语义： 按照指明的步长型控制的规定，动作语句表要反复地进入。

步长型控制可以使用多个循环计数器。每当进入动作语句表之前，各循环计数器的计算次序是不加规定的，唯一的要求是这些计算要能决定何时结束此循环动作。当诸循环计数器中至少有一个指示结束时，则整个循环语句即终止。

结束有正常和非正常之分。若循环计数器中至少有一个计算结果是指示结束时，则结束是正常的。如果有一个当型条件的计算提供FALSE，或执行出口动作或转向动作，而（目标）标号是在动作语句表之外定义的，或有一个异常被引发，其相应的处理程序是在此循环动作之外，且不附属于此循环动作的，则结束是不正常的。

1. 无穷循环：

动作表被无限制地重复，则此时只可能以非正常方式结束。

2. 值枚举：

按照循环计数器中指明的值的集合，动作语句表重复进入。该值的集合由一个离散模式（区段枚举）指明，或由一个幂集值（幂集枚举）指明，或由一个初值、一个步值和一个终值（步枚举）指明。

循环计数器总是在动作语句表内部隐含定义的。但是，如果有一个和循环计数器名字相同的访问名字在循环动作之外是可见的，则在非正常结束之前，循环计数器的值将存入被标志的地点。在正常结束情况下，存入由外部访问名字标志的地点的值是无定义的。

区段枚举：

在没有（具有）DOWN指明的区段枚举的情况下，循环计数器的初值是该离散模式定义的值的集合中最小（最大）的值。接下去在该动作语句表的执行中，“下一个值”的计算是：

SUCC(“前一个值”)(PRED(“前一个值”))如果动作语句表已经对于该离散模式定义的最大（最小）值执行完毕，则循环动作宣告结束（正常结束）。

幂集枚举：

在没有（具有）DOWN指明的幂集枚举的情况下，循环计数器的初值是被标志的幂集值的最小(最大)成员值。若该幂集值为空，则动作语句表将不被执行。在该动作语句表接下去执行时，下一个值将是幂集值中下一个较大（较小）的成员值。当动作语句表对于最大（最小）的值执行以后，整个循环语句即告结束（正常结束）。在执行循环动作时，幂集表达式只计算一次。

步枚举：

在没有（具有）DOWN指明的步枚举情况下，循环计数器的值的集合由初值、终值、可能还有步值合起来决定。在执行循环动作时，这些表达式只计算一次。其计算次序是未加指明的，可能是混合的。步值总是正的。在每次执行动作语句表之前，都要进行结束检验。开始时，首先检查并确定循环计数器的初值是否大于（小于）终值。接下去执行时，“下一个值”的计算是：

“前一个值”＋步值

(“前一个值”－步值)

以上是有步值的情况，若无步值，则为SUCC(“前一个值”)(PRED(“前一个值”))。若计算结果提供的值大于（小于）终值，则循环动作结束（正常结束），否则将引发OVERFLOW异常。

3．地点枚举：

在没有（具有）DOWN指明的地点枚举的情况下，动作语句表按照一组指明的地点重复进入，这些地点正是数组地点标志的数组地点的元素。其语义是好象一开始就遇到了下列的地点等同说明：

DCL〈循环计数器〉〈模式〉LOC:＝〈首地点〉；其中〈模式〉是数组地点模式的元素模式，而〈首地点〉是具有最小（最大）下标的元素。在进一步执行时，又好象在执行每个动作语句表之前遇到了下列地点等同说明：

DCL〈循环计数器〉〈模式〉LOC:＝〈下一地点〉；

其中〈下一地点〉是具有如下下标的数组元素：

“下一下标”＝SUCC(“前一下标”)

(PRED(“前一下标”))

如果循环计数器在下一次计算之前指出了具有最大（最小）下标的数组元素，则循环动作结束(正常结束)。在循环动作被执行时，数组地点只计算一次。

静态性质：

值枚举：

循环计数器是一个**值枚举**名字。如果在循环动作所在的范围内有一个名字是可见的，且该名字又等于循环计数器，则循环计数器是**显式的**，否则就是**隐式的**。

步枚举：

显式循环计数器的类是M值类，其中M是外部访问名字的模式（见下面：静态条件）。

隐式循环计数器的类是初值、步值（若有的话）和终值等的类的结果类。

区段枚举：

循环计数器的类是M值类，其中M是离散模式。

幂集枚举：

循环计数器的类是M值类，其中M是（强）幂集表达式的模式的成员模式。

地点枚举：

循环计数器是一个**地点枚举**名字。它的模式是数组地点的模式的**元素模式**。

如果数组地点的模式的元素布局是NOPACK，则**地点枚举**名字是（语言）可关联的。

静态条件：

步枚举：

初值、终值和步值（若有的话）的类之间应两两相容。在**显式**循环计数器的情况下，外部可见的名字必须是一个访问名字。外部访问名字的模式必须和这些类中的每一个相容，并且不得是只读模式。

幂集枚举，区段枚举

在**显式**循环计数器的情况下，外部可见的名字必须是一个访问名字。外部访问名字的模式必须与循环计数器的类相容。

幂集表达式必须是强的。

动态条件： 如果由步值提供的值不大于0，或在**显式**循环计数器的情况下，在非正常结束之前要存回到外部地点中去的值不在外部地点的模式所指明的界限之内，则将出现RANGEFAIL异常。该异常在循环动作的程序块之外出现。

例子：

```
4.16  FOR i:=1 TO c                                   (1.1)
15.27 FOR EVER                                        (1.1)
4.16  i: =1 TO c                                      (3.1)
9.11  j:=MIN (sieve) BY MIN(sieve) TO max             (3.1)
14.22 I IN INT (1:100)                                (3.2)
```

6.5.3 当型控制

语法：

```
〈当型控制〉::=                                        (1)
    WHILE〈布尔表达式〉                                (1.1)
```

语义： **布尔**表达式的计算在刚要进入动作语句表之前进行（如果有步长型控制的话，还应在此控制的计算之后进行）。若提供TRUE，即进入动作语句表，否则，循环动作即告结束（非正常结束）。

例子：

```
7.28  WHILE n>=1                                      (1.1)
```

6.5.4 结构型部分

```
〈结构型部分〉::=                                      (1)
    WITH〈结构型控制〉{,〈结构型控制〉}*                (1.1)
〈结构型控制〉::=                                      (2)
    〈结构地点〉                                       (2.1)
    〈结构表达式〉                                     (2.2)
```

注意：若结构表达式是地点，则语法结构是二义的，此时将被解释为一个结构地点。

语义： 在每个结构型控制中指明的结构地点或结构值的（可见的）场名字均可作为对场的直接访问来使用。

如果指明了结构地点，同时也就隐式地建立一些访问名字，它们和该结构地点模式的场名字相同，标志了该结构地点的各子地点。

如果指明了结构表达式，同时也就隐式地建立了一批值名字，它们和该（强）结构表达式模式的场名字相同，标志了该结构值的各子值。

在进入循环动作时，被指明的结构地点和/或结构值即被计算，但仅计算一次。计算次序是未经指明的，可能是混合的。

静态性质：

结构表达式：循环动作中可以使用的任何名字都是**值直接场**名字。它的类是M值类，其中M是该结构表达式的结构模式的某个场名字的模式，该场名字即作为**值直接场**名字而被使用。

结构地点：循环动作中可以使用的任何名字都是**地点直接场**名字。它的模式是该结构地点模式的某个场名字的模式，该场名字即作为**地点直接场**名字而被使用。若与**地点直接场**名字相结合的场名字的场布局是NOPACK，则该**地点直接场**名字是（语言）可关联的。

静态条件：结构表达式必须是强的。

例子：

15.48 WITH EACH (1.1)

6.6 出口动作

语法：

〈出口动作〉::= (1)

EXIT〈标号名字〉 (1.1)

语义 出口动作用于离开一个加括号的动作。恢复动作的地点紧接在冠以该标号名字且包含此出口动作的最内层加括号动作之后。

静态条件：出口动作必须位于冠以该标号名字的加括号动作语句之内。如果出口动作是在过程或进程定义的内部，则所离开的那个加括号动作语句也必须位于同一过程或进程定义之内（即出口动作不能用于离开过程或进程）。

出口动作不得附有处理程序。

例子：

15.52 EXIT FIND—COUNTER (1.1)

6.7 调用动作

语法：

〈调用动作〉::= (1)

〔CALL〕{〈过程调用〉 | 〈内部子程序调用〉} (1.1)

〈过程调用〉::= (2)

{〈过程名字〉 | 〈过程表达式〉}

（〔〈实在参数表〉〕） (2.1)

〈实在参数表〉::= (3)

〈实在参数〉{，〈实在参数〉}* (3.1)

〈实在参数〉::= (4)

〈值〉 (4.1)

| 〈静态模式地点〉 (4.2)

导出语法：关键词CALL是任选的。带有CALL的调用动作是从没有CALL的调用动作导出的。

语义：调用动作使过程表达式提供的值所指示的通用过程被调用，或使过程名字指示的过程被调用。在实在参数表中指明的实在值或地点被传递给过程。

静态性质：过程调用具有下列特征：一组参数说明、可能有一个结果说明、一组可能为空的异常名字、通用性、递归性以及还可能有区域性（后者只在过程名字的情况下才是可能的，见8.2.2节）。这些特征是从过程名字或与过程表达式的类相容的任何模式中继承过来的（后一种情况下，通用性总是通用的）。

当且仅当有结果说明，并且在结果说明中指明了LOC时，过程调用才是地点过程调用，否则就是值过程调用。

静态条件：过程调用中实在参数出现的个数必须与它的参数说明的个数一样多。对于过程调用的实在参数及（按位置）对应的参数说明的相容性要求是：

- 若参数说明具有IN属性(缺省)，则实在参数必须是一个值，它的类必须与相应参数说明的模式

相容。此模式不得具有同步性质。若过程调用不是区域性的，则（实在）值也不能是区域性的（见8.2.2节）。

· 若参数说明具有INOUT或OUT属性，则实在参数必须是静态模式地点，它的模式必须与M值类相容，其中M是相应的参数说明中的模式。（实在）静态模式地点既不可以有只读性质，也不可以有同步性质。若过程调用不是区域性的，（实在）地点也不能是区域性的（见8.2.2节）。

· 若参数说明具有INOUT属性，则参数说明中的模式必须与M值类相容，其中M是静态模式地点的模式。

· 若参数说明具有LOC属性，则实在参数必须是静态模式地点，它必须是可关联的，并且参数说明中的模式与这个（实在）静态模式地点的模式是读相容的。实在参数也可以是这样一个值，它不是地点，但它的类与参数说明中的模式是相容的。

动态条件： 过程调用能够引发与附属的异常名字对应的任一异常。若过程表达式提供NULL，将引发EMPTY异常，若存储要求不能得到满足，将引发SPACEFAIL异常，若过程递归地调用自己（即上一次调用仍处于活化状态）且它的递归性是非递归的，则将引发RECURSEFAIL异常。

参数传递能引发下列异常：

· 若参数说明有IN，INOUT或LOC属性，则（可能包含于实在地点中的）的（实在）值相对于参数说明模式的赋值条件在调用处适用（见6.2节），且可能的异常在过程调用之前就被引发。

· 若参数说明具有INOUT或OUT属性，则形式参数的局部值相对于（实在）地点的赋值条件在返回处适用（见6.2节），且可能的异常在过程返回后被引发。

· 若参数说明具有LOC属性，且实在参数是一个值，但不是一个地点，则（实在）值相对于参数说明模式的赋值条件在调用处适用，且可能的异常在过程被调用之前即被引发（见6.2节）。

过程表达式不得提供这样的过程，它在进程定义的内部被定义，该进程的活化又不是执行此过程调用的那个进程的活化（见8.1节），且被标志的过程的生存期不得是已经结束了的。

例子：

4.17 OP (a, b, d, order-1) (1.1)

6.8 结果和返回动作

语法：

〈返回动作〉::= (1)
RETURN〔〈结果〉〕 (1.1)
〈结果动作〉::= (2)
RESULT〈结果〉 (2.1)
〈结果〉::= (3)
〈值〉 (3.1)
| 〈静态模式地点〉 (3.2)

导出语法： 带结果的返回动作是从RESULT〈结果〉；RETURN导出的。若这样的返回动作附有一个处理程序，则该处理程序将被看作附属于该返回动作导出的结果动作。

语义： 结果动作用来建立过程调用提供的结果。这个结果可以是地点或值。返回动作用于从过程召用中返回。它就包含在这个过程的定义之中。若过程送返一个结果，则此结果是由最后执行的结果动作决定的。若无结果动作被执行过，则过程调用分别提供无定义的地点或无定义的值。

静态性质： 结果动作和返回动作均附有过程名字，它是包含该动作的最内层过程定义的名字。

静态条件： 返回动作和结果动作必须在文字上包含于过程定义之中。只有当结果动作的过程名字有结果说明时，该结果动作才能被指明。

（没有结果的）返回动作不得附有处理程序。

若在结果动作的过程名字的结果说明中指明了LOC，则结果必须是静态模式地点，使结果说明中的模式与静态模式地点的模式读相容。若结果动作的过程名字不是区域性的，则结果中的静态模式地点也不得是区域性的（见8.2.2节）。

若结果动作的过程名字的结果说明中未指明LOC，则结果必须是一个值，它的类必须与结果说明中的模式相容。若结果动作的过程名字不是区域性的，则结果中的值也不得是区域性的（见8.2.2节）。

动态条件： 若在过程名字的结果说明中未指明LOC，则结果动作中的值相对于它的过程名字的结果说明中的模式的赋值条件适用。

例子：

```
4.20  RETURN                                    (1.1)
1.5   RESULT  i+j                               (2.1)
5.20  C                                         (3.1)
```

6.9 转向动作

语法：

```
〈转向动作〉::=                                  ( 1 )
  GOTO〈标号名字〉                               (1.1)
```

语义： 转向动作引起控制的转移。动作在冠以标号名字的动作语句处恢复。

静态条件： 若转向动作位于过程或进程定义之内，由标号名字指示的标号也必须在此定义之内被定义（即不能跳到过程或进程召用之外）。

转向动作不得附有处理程序。

6.10 断言动作

语法：

```
〈断言动作〉::=                                  ( 1 )
  ASSERT〈布尔表达式〉                           (1.1)
```

语义： 断言动作提供了测试条件的手段。

动态条件： 若布尔表达式提供FALSE，则将出现ASSERTFAIL异常。

例子：

```
4.6  ASSERT  b>0 AND c>o AND order>0             (1.1)
```

6.11 空动作

语法：

```
〈空动作〉::=                                    ( 1 )
  〈空〉                                         (1.1)
〈空〉::=                                        ( 2 )
```

语义： 空动作不引起任何动作。

静态条件： 空动作不得附有处理程序。

6.12 引发动作

语法：

```
〈引发动作〉::=                                  ( 1 )
  CAUSE〈异常名字〉                              (1.1)
```

语义： 引发动作引发一个异常。

静态条件： 引发动作不得附有处理程序。

动态条件： 引发动作引发的异常是其名字被异常名字指出的那个异常。

例子：

4.8 CAUSE wrong-input (1.1)

6.13 开动动作

语法：

〈开动动作〉::= (1)

〈开动表达式〉〔SET〈样品地点〉〕 (1.1)

导出语法： 带SET任选的开动动作是下列单个赋值动作的导出语法：

〈样品地点〉::=〈开动表达式〉

语义： 开动动作计算开动表达式（见5.2.17节），在计算时不使用作为结果的样品值。

例子：

14.37 START CALL_DISTRIBUTOR( ) (1.1)

6.14 停止动作

语法：

〈停止动作〉::= (1)

STOP (1.1)

语义： 停止动作结束正在执行该停止动作的进程（见8.1节）。

静态条件： 停止动作不得附有处理程序。

6.15 继续动作

语法：

〈继续动作〉::= (1)

CONTINUE〈事件地点〉 (1.1)

语义： 继续动作使被指明的事件地点上被延迟的具有最高优先级的那个进程重新活化。如果具有最高优先级的进程不止一个，则根据实现定义的调度算法，从具有最高可能的优先级中选出特定的一个进程。如果被指明的事件地点上并无被延迟的进程，则继续动作，不产生其它效应（有关细节参见第8章）。

例子：

13.23 CONTINUE RESOURCE_FREED (1.1)

6.16 延迟动作

语法：

〈延迟动作〉::= (1)

DELAY〈事件地点〉〔〈优先级〉〕 (1.1)

〈优先级〉::= (2)

PRIORITY〈整数直接量表达式〉 (2.1)

语义： 延迟动作使执行它的进程被延迟。该进程可以在指明的事件地点上被一个继续动作所活化。优先级指出这个被延迟的进程相对于所有在该事件地点上被延迟的进程的优先程度。缺省和最低的优先级是0（有关细节请参见第8章）。

静态条件： 整数直接量表达式不得提供负值。

动态条件： 若事件地点的模式附有长度且在指明的事件地点上被延迟进程的个数正好等于计算完事件地点之后的长度，则将出现DELAYFAIL异常。该异常在进程被延迟前出现。

当执行延迟动作的进程正在事件地点上被延迟时，该事件地点的生存期不得结束。

例子：

13.17 DELAY RESOURCE_FREED (1.1)

6.17 延迟情况动作

语法:

〈延迟情况动作〉::= (1)
DELAY CASE〔SET〈样品地点〉;〕
〔〈优先级〉;〕{〈延迟选择对象〉}*
ESAC (1.1)

〈延迟选择对象〉::= (2)
(〈事件表〉):〈动作语句表〉 (2.1)

〈事件表〉::= (3)
〈事件地点〉{,〈事件地点〉}* (3.1)

语义: 延迟情况动作使执行它的进程被延迟。它可以在指明的事件地点之一上被继续动作所活化。在此情况下执行如下的动作语句表:它被一个事件地点所标定,使该进程重新活化的继续动作就是在该事件地点上被执行的动作(有关细节参见第8章)。每个事件地点和样品地点(若被指明了的话)的计算在进程被延迟之前即已进行。这些计算的次序是未加指明的,可能是混合的。如果两个或更多的计算提供同一个事件地点,则动作语句表的选择是非确定性的。

如果指明样品地点,则有一个样品值被存入此样品地点。这个样品值识别执行起活化作用的继续动作的那个进程。

静态条件: 样品地点的模式不得具有只读性。优先级中的整数直接量表达式不得提供负值。

动态条件: 若至少有一个事件地点的模式附有长度,使得在指明的事件地点上被延迟进程的个数正好等于计算事件地点之后的长度,则将出现DELAYFAIL异常。该异常在进程延迟之前就出现。

不管执行延迟情况动作的进程在哪个事件地点上被延迟,该事件地点的生存期都不得结束。

例子:

```
14.20  DELAY CASE
         (OPERATOR_IS_READY):/*some actions*/
         (SWITCH_IS_CLOSED):DO FOR I IN INT(1:100);
                              CONTINUE OPERATOR_IS_READY;
                              /*empty the queue*/
                              OD;
ESAC                                                        (1.1)
```

6.18 发送动作

6.18.1 概述

语法:

〈发送动作〉::= (1)
〈发送信号动作〉 (1.1)
〈发送缓冲区动作〉 (1.2)

语义: 发送动作启动同步信息的传送,从发送进程传送出去。具体的语义取决于同步对象是信号还是缓冲区。

6.18.2 发送信号动作

语法:

〈发送信号动作〉::= (1)

SEND 〈信号名字〉 〔(〈值〉{,〈值〉}*)〕
〔TO 〈样品表达式〉〕〔〈优先级〉〕 (1.1)

语义： 被指明的信号与值表及优先级（若有的话）一起发送。缺省和最低的优先级是0。若信号名字附有进程名字，则意味着只有具有该名字的进程可以接收此信号。如果指明了TO任选，则识别的是唯一能接收由发送信号动作发送的值表的进程。这种进程的识别不得与可能附属于信号名字的进程名字相矛盾。不仅该信号的可能进程名字，而且可能的样品值都是动态地附属于被发送的值表之上的（有关细节参见第8章）。

静态条件： 值出现的个数必须与信号名字的模式的个数相等。每个值的类必须与信号名字相应的模式相容。所有的值出现都不能是区域性的（见8.2.2节）。优先级中的整数直接量表达式不得提供负值。

动态条件： 每个值相对于信号名字的与其相应模式的赋值条件都必须得到遵守。

若样品表达式提供NULL，则将出现EMPTY异常。

若正当执行发送信号动作时，由样品表达式提供的值所指示的进程的生存期正好结束，则在此情况下，且仅在此情况下，出现EXTINCT异常。

如果信号名字附有进程名字，它不是由样品表达式提供的值所指示的进程的名字，则出现MODEFAIL异常。

例子：

15.68 SEND READY TO RECEIVED_USER (1.1)
15.76 SEND READOUT(COUNT) TO USER

6.18.3 发送缓冲区动作

语法：

〈发送缓冲区动作〉::= (1)
SEND 〈缓冲区地点〉(〈值〉)〔〈优先级〉〕 (1.1)

语义： 所指明的值连同优先级一起被存放入缓冲区地点（如果后者的容量能放得下它们的话）。但如缓冲区地点的模式附有长度，而在执行发送缓冲区动作之前，缓冲区中存放的值的个数已经等于这个长度，则缓冲区就放不下新来的值。其结果是，发送进程将被延迟，直到缓冲区地点再腾出容量，或发送来的值被用掉为止。缺省和最低的优先级是0（有关细节参见第8章）。

静态条件： 值的类必须与缓冲区地点的模式的缓冲区元素模式相容。值不得是区域性的（见8.2.2节）。优先级中的整数直接量表达式不得提供负值。

动态条件： 对于发送缓冲区动作，在值和缓冲区地点模式的缓冲区元素模式之间的赋值条件应得到遵守。如果有异常，则在进程延迟之前就出现。

当执行发送缓冲区动作的进程在所提供的缓冲区地点上被延迟时，此缓冲区地点的生存期不能结束。

例子：

16.115 SEND USER→ (〔READY,→COUNTER_BUFFER〕) (1.1)

6.19 接收情况动作

6.19.1 概述

语法：

〈接收情况动作〉::= (1)
〈接收信号情况动作〉 (1.1)
| 〈接收缓冲区情况动作〉 (1.2)

语义： 接收情况动作接收由发送动作传送过来的同步信息。具体的语义与同步对象有关，它可能是一个信号

或一个缓冲区。进入接收情况动作并不一定会造成执行进程的延迟（有关细节参见第8章）。

6.19.2 接收信号情况动作

语法：

〈接收信号情况动作〉::= (1)

RECEIVE CASE〔SET〈样品地点〉；〕

{〈信号接收选择对象〉}+

〔ELSE〈动作语句表〉〕ESAC (1.1)

〈信号接收选择对象〉::= (2)

(〈信号名字〉〔IN〈名字表〉〕)

:〈动作语句表〉 (2.1)

语义： 接收信号情况动作接收一个信号，可能还有一组值，它的信号名字是在信号接收选择对象中指明的。

在进入接收信号情况动作时，如果有执行该动作的进程，且在它可以接收的名字中有一个名字，则到来的信号就是属于该名字的，则该信号即被接收。如果没有这样的信号，并且未曾指明ELSE，则执行该接收信号情况动作的进程被延迟；如果指明了ELSE，则进入跟在它后面的动作语句表。

仅当下列条件满足时信号才能被进程所接收：

- 如果该信号附有进程名字，则该进程同时就是接收进程的名字。
- 如果该信号附有样品值，则此样品值确认该接收进程。

如果信号可以被接收，则进入冠以被接收信号的信号名字的动作语句表。如果有多于一个的信号可以被接收，则根据由实现定义的调度算法选出具有最高优先级的信号。如果信号名字附有一组模式，即有一组值与信号同时发送，则在IN之后必须指明一组名字。它们是引进的值名字，标志了被接收的值。如果在接收信号情况动作所在的范围中有一个可见的访问名字等于一个引进名字，则被接收的值将存入被标志的地点，存入的时刻紧接在接收信号之后，执行动作语句表之前。

如果指明SET任选，则紧接着信号被接收之后，标志发送该信号的进程的样品值将被存入指明的样品地点。

静态性质： 在信号接收选择对象的名字表中定义的任一名字都是值接收名字。它的类是M值类，其中M是在它前面的信号名字的相应模式。如果在信号接收情况动作所在的范围内有一个可见的名字等于IN后面引进的一个名字，则值接收名字是显式的，否则就是隐式的。

静态条件： 样品地点的模式不得具有只读性质。

所有的信号名字出现都必须是不同的。

当且仅当信号名字具有非空的模式集合时，在信号接收选择对象中必须指明任选的IN和名字表。名字表中名字的个数必须等于信号名字的模式个数。

若值接收名字是显式的，在外部可见的名字必须是一个访问名字，它的模式必须和值接收名字的类相容。访问名字的模式不得有只读性质。

动态条件： 若值接收名字是显式的，在被接收的值与外部访问名字的模式之间应有赋值条件成立。如果有异常，只能出现在接收信号之后，进入动作语句表之前。

当进入动作语句表时，如果存储要求不能得到满足，则将出现SPACEFAIL异常。

例子：

```
15.73  RECEIVE CASE
         (STEP):COUNT+:=1;
         (TERMINATE):
           SEND READOUT(COUNT) TO USER;
           EXIT WORK_LOOP;
       ESAC                                        (1.1)
```

6.19.3 接收缓冲区情况动作

语法：

〈接收缓冲区情况动作〉::= (1)

RECEIVE CASE〔SET〈样品地点〉；〕

{〈缓冲区接收选择对象〉}+

〔ELSE〈动作语句表〉〕

ESAC (1.1)

〈缓冲区接收选择对象〉::= (2)

(〈缓冲区地点〉IN〈名字〉)

:〈动作语句表〉 (2.1)

语义： 接收缓冲区情况动作从缓冲区地点或从缓冲区地点上被延迟的发送进程接收一个值，其地点在缓冲区接收选择对象中指出。

在进入接收缓冲区情况动作时，如果在指明的缓冲区地点之一有一个值，或有一个被延迟的发送进程，则值将被接收，一个冠以缓冲区地点的动作语句表将被执行，这个缓冲区地点提供的就是作为值的来源的那个缓冲区地点。

在进入接收缓冲区情况动作时，各缓冲区地点按未加指明的和可能是混合的次序进行计算。对计算次序的唯一要求是，必须足以选出一个选择对象。在所有被指明的缓冲区地点都不包含值，且没有一个发送进程在指明的缓冲区地点上被延迟的情况下，如果未曾指明ELSE，则执行的进程将被延迟，如果指明了ELSE，则在它之后的动作语句表将被执行。如果能被接收的值不止一个，则将按照由实现定义的调度算法选出具有最高优先级的值。如果两个或更多的缓冲区地点出现提供同一个缓冲区地点，后者就是被接收的值的来源，则动作语句表的选择是非确定性的。

值的接收发生在刚要进入冒号后面的动作语句表之前。IN后面的名字是引进的值接收名字，标志被接收的值。如果在接收缓冲区情况动作所在的范围内，有一个可见的访问名字和一个被建立的值接收名字相同，则在刚要进入动作语句表之前，被接收的值将被存入被标志的地点中去。

如果指明了SET任选，则紧接着接收值之后，指明的样品地点中即存入了样品值，后者标志发送被接收的值的进程。

静态性质： 缓冲区接收选择对象中IN后面的名字是值接收名字。它的类是M值类，其中M是标定缓冲区接收选择对象的缓冲区地点模式的缓冲区元素模式。

如果在接收缓冲区情况动作所在的范围内，有一个可见的名字和IN后面的引进名字相同，则值接收名字称作显式的，否则是隐式的。

静态条件： 样品地点的模式不得有只读性质。如果值接收名字是显式的，则外部可见的名字必须是访问名字，它的模式必须和具有相同名字的值接收名字的类相容。这个模式不得有只读性质。

动态条件： 如果值接收名字是显式的，在被接收的值和外部访问名字的模式之间应有赋值条件成立。如果有异常，只能出现在接收值之后，进入动作语句表之前。

在进入动作语句表时，如果存储要求不能得到满足，则将出现SPACEFAIL异常。

任何被提供的缓冲区地点，当它上面正有一个执行接收缓冲区情况动作的进程被延迟时，它的生存期是不能结束的。

7.0 程序结构

7.1 概述

加括号的循环动作、分程序、模块、区域、延迟情况动作、接收情况动作、过程定义和进程定义决定了程序的结构。即决定了名字的作用域和在其中建立的地点的生存期。

- 程序块这个字被用于标志：
  - 循环动作中的动作语句表，包括循环计数器和当型控制；
  - 分程序；
  - 不包括结果说明在内的过程定义；
  - 进程定义；
  - 在缓冲区接收选择对象或信号接收选择对象中的动作语句表，包括IN后面的名字或名字表在内；
  - 在接收情况动作或处理程序中ELSE后面的动作语句表；
  - 在处理程序中的异常处理选择对象；
- 模片这个字用于标志模块或区域。
- 组块这个字标志程序块或模片。
- 范围或组块范围这个字标志该组块不包含该组块内层组块中的部分（即由组块的最外嵌套层组成的部分）。组块定义了在它的范围内建立的名字的作用域。名字可有如下几种建立方式：
- 如果名字出现在说明、模式定义或异名定义的名字表中，或信号定义中，则此名字在各该说明、模式定义、异名定义或信号定义所在的范围内建立。
- 如果名字出现在形式参数表的名字表中，则此名字在相应的过程定义或进程定义的范围内建立。
- 如果名字位于冒号之前，冒号后是一个动作、区域、过程定义、入口定义或进程定义，则该名字在各该动作、区域、过程定义、包含该入口定义的过程定义和进程定义所在的范围内建立。
- 每个值枚举名字、地点枚举名字、值直接场名字和地点直接场名字在相应的循环动作的程序块范围内建立。
- 每个值接收名字在相应的信号接收选择对象或缓冲区接收选择对象的程序块的范围内建立。
- 场名字或集合元素名字在它们相应的结构模式或集合模式的定义性出现所在的范围内建立。
- 异常名字通过引发动作或异常处理选择对象建立（注意：异常名字没有特定的建立点；见第10章）。
- 语言预定义的名字看作是在标准序部模块的范围内建立（见7.8节）。

程序员引进的（建立的）名字，除异常名字外，都有唯一的建立（说明或定义）地点。这个地点称为名字的定义性出现。名字被使用的地点称为名字的应用性出现。名字约束规则把名字的每个应用性出现与唯一的定义性出现结合起来（见9.2.8节）。对于异常名字来说不区分定义性出现和应用性出现（见第10章）。

名字有一定的作用域，即程序的这样一个部分，在那里它的定义或说明是可见的，因此，也是可以自由使用的，则称该名字在该部分中是可见的。地点有一定的生存期，即程序中地点存在的部分。程序块确定了在其中建立的名字的可见性和地点的生存期。模片只确定可见性；在模片中建立地点的生存期的确切方式是：把这些地点看作好象是在包含它们的最小程序块的范围中建立的一样。模片允许限制名字的可见性。例如，在一个模块的范围中建立的名字并不能自动地在内层或外层模块中成为可见的，即便它的生存期允许这样做也罢。

7.2 范围和嵌套

语法：

〈分程序体〉::= (1)

〈数据语句表〉〈动作语句表〉 (1.1)

〈过程体〉::= (2)

〈数据语句表〉
{〈动作语句〉 | 〈入口语句〉}* (2.1)
〈进程体〉::= (3)
〈数据语句表〉〈动作语句表〉 (3.1)
〈模块体〉::= (4)
{〈数据语句〉 | 〈可见性语句〉 |
〈区域〉}*〈动作语句表〉 (4.1)
〈区域体〉::= (5)
{〈数据语句〉 | 〈可见性语句〉}* (5.1)
〈动作语句表〉::= (6)
{〈动作语句〉}* (6.1)
〈数据语句表〉::= (7)
{〈数据语句〉}* (7.1)
〈数据语句〉::= (8)
〈说明语句〉 (8.1)
| 〈定义语句〉 (8.2)
〈定义语句〉::= (9)
〈异名模式定义语句〉 (9.1)
| 〈新模式定义语句〉 (9.2)
| 〈异名定义语句〉 (9.3)
| 〈过程定义语句〉 (9.4)
| 〈进程定义语句〉 (9.5)
| 〈信号定义语句〉 (9.6)
| 〈空〉; (9.7)

语义： 在进入一个程序块的范围时，所有在进入该程序块时建立的地点的生存期初始化首先被实行。其次实行的是程序块范围中那些范围初始化和地点等同说明中可能有的动态计算，实行的次序是它们在文字上被指明的次序。

在进入一个模片的范围时，模片范围中的范围初始化和地点等同说明中可能有的动态计算按照它们在文字上指明的次序先后实行。

静态性质： 每个范围有一个唯一的直接外层组块，定义如下：

- 如果该范围是一个循环动作、分程序、过程定义、进程定义、模块或区域的范围，则它的直接外层组块就是以那个循环动作、分程序、过程定义、进程定义、模块或区域所在范围为范围的组块。
- 如果该范围是一个缓冲区接收选择对象或信号接收选择对象的动作语句表，可能包括引进的名字，或者是在一个接收缓冲区情况动作或接收信号情况动作中的ELSE后面的动作语句表，则它的直接外层组块就是以该接收缓冲区情况动作或接收信号情况动作所在范围为范围的组块。
- 如果该范围是异常处理选择对象中的动作语句表，或处理程序中ELSE后面的动作语句表，且该处理程序又是不附着于一个组块的，则直接外层组块就是以该处理程序所附着的语句所在范围为范围的组块。
- 如果该范围是异常处理选择对象，或处理程序中ELSE后面的动作语句表，且该处理程序又是附着于一个组块的，则直接外层组块就是该处理程序所附着的那个组块。

每个范围有一个唯一的直接外层范围，它就是直接外层组块的范围。每个语句有唯一的直接外层组块，它就是以该语句所在范围为范围的组块。当且仅当一个范围是一个组块（范围）的直接外层范围时，则称前者直接包含后者。

当且仅当某组块是一个语句（范围）的直接外层组块，或该语句（范围）的直接外层范围被该组块所包含时，称该语句（范围）被该组块所包含。

我们说进入一个范围，如果是：

- 模块范围：该模块作为动作被执行（即如果转向语句把控制转向定义在该模块内部的一个标号名字时，则不能说是进入了该模块）。

- 分程序范围：分程序作为一个动作被执行。
- 区域范围：正面进入该区域（即如果该区域的一个关键子程序被调用，则不能说是进入了该区域）。
- 过程范围：通过过程的主要入口进入该过程（即不是通过附带定义的入口点）。
- 进程范围：通过一个开动语句活化进程。
- 循环范围:在计算完控制部分中的表达式或地点后，循环动作被作为一个动作来执行。
- 缓冲区接收选择对象范围，信号接收选择对象范围：在接收缓冲区值或信号后，该选择对象被执行。
- 异常处理选择对象范围：在引发一个异常后，该异常处理选择对象被执行。

当且仅当动作语句表的第一个动作（若有的话），从动作语句表的外面接收控制时，则称进入了该动作语句表。

7.3 分程序

语法：

〈分程序〉::= (1)

BEGIN〈分程序体〉END (1.1)

语义： 分程序是一个动作（复合动作），可能包含局部说明和局部定义。它确定局部建立名字的可见性以及局部建立地点的生存期（见7.9节和9.2.5节）。

动态条件： 若分程序要求局部存储，而这种存储要求又不能得到满足，则将出现SPACEFAIL异常。

例子：

见15.63～15.80

7.4 过程定义

语法：

〈过程定义语句〉::= (1)

〈名字〉:〈过程定义〉

〔〈处理程序〉〕〔〈过程名字〉〕; (1.1)

〈过程定义〉::= (2)

PROC ([〈形式参数表〉])〔〈结果说明〉〕

〔EXCEPTIONS (〈异常表〉)〕〈过程属性〉;

〈过程体〉END (2.1)

〈形式参数表〉::= (3)

〈形式参数〉{,〈形式参数〉}* (3.1)

〈形式参数〉::= (4)

〈名字表〉〈参数说明〉 (4.1)

〈过程属性〉::= (5)

〔〈通用性〉〕〔RECURSIVE〕 (5.1)

〈通用性〉::= (6)

GENERAL (6.1)

|SIMPLE (6.2)

|INLINE (6.3)

〈入口语句〉::= (7)

〈名字〉:〈入口定义〉; (7.1)

〈入口定义〉::= (8)

ENTRY (8.1)

导出语法： 一个形式参数，如果它的名字表是由多于一个名字组成的，则该形式参数是从以逗号隔开的多个

形式参数出现中导出的，其中每个名字对应一个出现，每个出现都有相同的参数说明。例如，I，J INT LOC是从I INT LOC，J INT LOC中导出的。

语义： 过程定义定义了（可能是参数化的）动作序列，在程序的不同地点上可以调用这些动作。通过执行返回动作，或通过到达过程体的末尾，或通过到达一个附着于该过程定义的异常处理选择对象（调用失败），均可使控制回到调用点。过程复杂程度的不同可作如下说明：

简单过程(SIMPLE）是不能被动态地处理的过程。它们并不当作值来处理，即它们不能存入一个过程地点，也不能作为参数传给一个过程调用或作为结果被一个过程调用送返。

通用过程(GENERAL)无简单过程那种限制，并可作为过程值处理。

直接插入过程(INLINE)具有和简单过程同样的限制，且不能是递归的。它们的语义和正常过程相同，但编译程序将把生成的目标码直接插入过程召用点，而不是生成实际上调用该过程的代码。

只有简单和通用过程可被指明为（相互）递归的。如未指明过程属性，则采用由实现定义的缺省属性。

过程可以送返一个值或一个地点（由结果说明中的LOC属性指示）。

过程定义前面的名字定义了该过程的名字。若过程名字是通用的，它是被定义的过程值的一个过程直接量。它的类由形式参数表和结果说明中的模式和属性确定。

使用入口语句可使过程具有多个入口点。这些语句可看作是附加的过程定义。入口语句中的名字定义以它所在范围为范围的过程的入口点的名字。入口点由入口语句在文字上的位置确定。

参数传递：

基本上有两种参数传递机制：**按值传递**和**按地点传递**(LOC属性)。OUT和INOUT属性表示按值传递机制的变种。

按值传递：

用按值方式传递参数时，值被作为参数传给过程，并存入具有指明的参数模式的一个局部地点中。其效果是，好象在过程调用开始时就遇上了地点说明：

DCL 〈形式参数名字〉〈模式〉:=〈实在参数〉;

但是，初始化不能在过程体内部引发异常。可用任选关键字IN显式地指明按值传递机制。

如果指明了属性INOUT，则实在参数值将在返回前从一个地点中取得。形式参数的当前值被重新存入实在地点中。

OUT的效果与INOUT相同，唯一的例外是在进入过程后，实在地点的初始值不被复制入形式参数地点中，因此形式参数有一个未定义的初始值。如果过程在调用点引发一个异常，则回存操作不需实行。

按地点传递：

在按地点传递参数时，地点被作为参数传给过程体。不可关联的地点和动态模式地点都不能按此方式传递。其效果是，好象在过程的入口点遇上了下列地点等同说明语句：

DCL 〈形式参数名字〉〈模式〉LOC :=〈实在参数〉;但是，这种说明不能在过程体内部引发异常。

如果指明的值不是静态模式地点，将隐式地建立起一个包含被指明的值的地点，并在调用点传送此地点。被建立的地点的生存期是整个过程调用期。

结果传送：

过程可以送返一个值或一个地点。在前一种情况下，可在任一结果动作中指明这个值，而在后一种情况下，则指明静态模式地点（见6.8节）。送返的值或地点由返回前最近一次执行的结果动作决定。如果带有结果说明的过程在返回时并未执行结果动作，则该过程送返一个未定义的值或未定义的地点。在此情况下，该过程调用不能用作地点过程调用（见4.2.10节），也不能用作值过程调用（见5.2.15节），而只能用作调用动作（见6.7节）。

寄存器说明：

寄存器说明可以在过程的形式参数和结果说明中给出。在按值传递的情况下，意味着实在值包含在被指明的寄存器中；在按地点传递的情况下，意味着指向实在地点的（隐含的）指针包含在被指明的寄存器中。如果是在结果说明中指明的，则意味着送返的值或指向送返地点的（隐含的）指针包含在被指明的寄存器中。

静态性质： 当且仅当一个名字是在过程定义语句或入口语句中定义时（即位于一个冒号和过程定义或入口定义之前），则该名字是过程名字。

每个过程名字附有一个过程定义，其规定如下：

- 若过程名字是在一个过程定义语句中定义的，则它就是该语句中的过程定义。
- 若过程名字是在一个入口语句中定义的，则它就是以该入口语句所在的范围为范围的过程定义。

每个过程名字附有如下的，由其过程定义所定义的性质：

- 它有一组参数说明，它们是由形式参数表中的参数说明出现定义的，每个参数由一个模式，可能还由一个参数属性和/或寄存器名字组成。
- 它可能有一个结果说明，由一个模式，可能还由一个LOC属性和/或寄存器名字组成。
- 它有一个可能为空的异常名字的集合，即异常表中提到的那些名字。
- 它有通用性。如果指明了GENERAL、SIMPLE或INLINE，则通用性分别是通用或简单或直接插入。否则就由实现定义的缺省指明通用或简单。如果过程名字是在区域内部定义的，则它的通用性是简单的。
- 它有递归性。如果指明了RECURSIVE，则是递归的，否则就由实现定义的缺省指明递归或非递归。但是，若通用性是直接插入，或过程名字是关键的（见8.2节），则递归性是非递归的。

一个通用的过程名字是一个过程直接量。每个通用过程名字附有一个过程模式，它的形式是：

PROC ([〈参数表〉])[〈结果说明〉]
[EXCEPTIONS (〈异常表〉)][RECURSIVE]

其中〈结果说明〉，若有的话，和〈异常表〉与过程定义中的一样，〈参数表〉就是形式参数表中以逗号隔开的〈参数说明〉出现的序列。

当且仅当形式参数中的参数说明不包含LOC属性时，在形式参数的名字表中定义的名字是一个地点名字。如果它包含LOC属性，则是一个地点等同名字。任何这样的地点名字或地点等同名字都是（语言）可关联的。

静态条件： 如果过程名字是区域性的（见8.2.2节），则它的过程定义不能指明GENERAL。

如果过程名字是关键的（见8.2节），则它的定义既不能指明GENERAL，也不能指明RECURSIVE。

任何过程定义都不能同时指明INLINE和RECURSIVE。

如果指明分号前的任选过程名字，则必须和过程定义前面的名字一致。

仅当在参数说明或结果说明中指明了LOC时，其中的模式才可以具有同步属性。

例子：

```
1.3  add:
       PROC(i, j INT)(INT)EXCEPTIONS(OVERFLOW);
       RESULT  i+j;
     END add;                                          (1.1)
```

7.5 进程定义

语法：

〈进程定义语句〉::= (1)

〈名字〉:〈进程定义〉
[〈处理程序〉][〈进程名字〉]; (1.1)

〈进程定义〉::= (2)

PROCESS ([〈形式参数表〉]);
〈进程体〉END (2.1)

语义： 进程定义定义了可能是参数化的动作序列，它可以在程序的不同地点开动和并发执行（见第8章）。

静态性质： 当且仅当一个名字在进程定义语句的内部被定义时（即位于冒号和进程定义之前），它是一个进程名字。

进程名字可以附有一个由实现定义的异常名字的集合。

静态条件： 如果在分号前面指明了任选的进程名字，它必须和进程定义前面的名字一致。

进程定义语句不得包含在一个区域中，也不得包含在程序块中，除非后者是虚拟的最外层进程定义（见7.3节）。

形式参数表中的参数属性不能是INOUT，也不能是OUT。

仅当在形式参数表中形式参数的参数说明指明了LOC时，其中的模式才可以具有同步属性。

例子：

```
14.12  PROCESS( );
         DO FOR EVER;
           WAIT(10/*秒*/);
           CONTINUE OPERATOR_IS_READY;
         OD;
         END                                                    (2.1)
```

7.6 模块

语法：

```
〈模块〉::=                                                      (1)
   MODULE〈模块体〉END                                           (1.1)
```

语义： 模块是可能包含局部说明和定义的动作。模块是限定名字可见性的手段；它对局部建立地点的生存期没有影响。

模块的具体可见性规则在9.2节中给出。

静态性质： 当且仅当一个名字是通过放在冒号和MODULE前面而被定义时，它是模块名字。

例子：

```
7.42  MODULE
        SEIZE convert;
        DCL n INT INIT:=1979;
        DCL rn CHAR(20) INIT:=(20)'';
        GRANT n,rn;
        convert( );
        ASSERT rn='MDCCCCLXXVIIII'//(6)";
      END                                                       (1.1)
```

7.7 区域

语法：

```
〈区域〉::=                                                      (1)
   〔〈名字〉:〕REGION〈区域体〉END
   〔〈处理程序〉〕〔〈区域名字〉〕;                               (1.1)
```

语义： 区域是为进程并发执行提供的一种手段，可用来对它的局部说明的数据对象实行互斥性访问（见第8章）。它确定局部生成名字的可见性的方式和模块一样。

静态性质： 当且仅当一个名字是通过放在REGION前的冒号前面而被定义时，该名字是一个区域名字。

静态条件： 分号前的任选区域名字必须和区域的名字一致。

一个区域不得包含于一个程序块中，除非后者是虚拟的最外层进程定义。

例子：

见13.1—13.25

7.8 程序

语法：

〈程序〉::= (1)

{〈模块动作语句〉 | 〈区域〉}+ (1.1)

语义： 程序由一组包含在一个虚拟的最外层进程定义中的模块或区域组成。这个进程定义被认为是在它的范围中包含了一个标准CHILL序部模块。该模块包含下述一些对象的定义：CHILL预定义名字以及实现预定义的内部子程序、模式和寄存器名字。

静态性质： 语言和实现定义的名字(见附录C2)被认为是在虚拟的最外层进程定义的范围中的模块中生成的。该模块保证了这些名字的PERVASIVE性质（见9.2.6.2节)。

7.9 存储分配和生存期

地点或过程在它的程序中存在的时间称为它的**生存期**。

地点是通过说明或通过执行GET-STACK内部子程序调用而生成的。

在分程序的范围中说明的地点的生存期就是控制留在该分程序中的时间，除非该地点的说明带有STATIC属性。在一个模片范围中说明的地点的生存期等于该模片最小外层分程序范围中说明的地点的生存期。如果被说明的地点带有STATIC属性，则它的生存期就好象它是在虚拟的最外层进程定义的范围中被说明的一样。这意味着，对于具有STATIC属性的地点说明，存储分配只进行一次，即在开动虚拟的最外层进程时进行。如果这种说明在过程定义或进程定义内部出现，则对于所有的召用或活化，存在的地点只有一个。

通过执行GETSTACK内部子程序调用而建立的地点的生存期始于调用执行，终于离开最小的外层分程序。如果在计算一个过程调用的实在参数或开动表达式时执行GETSTACK内部子程序调用，则被建立的地点的生存期是过程调用期或被建立的进程的生存期。

在地点等同说明中建立的访问的生存期是包含该地点等同说明的最小程序块。

过程的生存期是包含该过程定义的最小程序块。

静态性质： 当且仅当一个地点是下列几种静态模式地点之一时，该地点称为是**静态**的：

- 被说明为带有STATIC属性的一个地点名字，或其定义不是包含在一个非虚拟进程定义的程序块中的地点名字。
- 一个地点等同名字，在它的定义中出现的静态模式地点是**静态**的。
- 一个串元素或子串，其中的串地点是**静态**的，并且左元素和右元素是**常量**，或定位是**常量**。
- 一个数组元素或子数组，其中数组地点是**静态**的，并且表达式是**常量**，或下元素和上元素是**常量**，或在其中出现的整数表达式是**常量**。
- 一个结构场，其结构地点是**静态**的。如果结构地点不是参数化的结构地点，则场名字不得是变体场名字。
- 一个地点转换，在其中出现的地点是**静态**的。

8.0 并发执行

8.1 进程和它们的定义

进程是一系列语句的顺序执行，它的顺序执行可以与其它的进程并发进行。进程的**行为**由**进程定义**描述(见7.5节)。它描述局限于进程的对象以及需要顺序执行的一系列动作语句。

进程通过开动表达式的计算而**建立**（见5.2.17节)。它被**活化**（即处于执行中）并被认为是与其它进程并发执行的。被建立的进程是由该进程定义的**进程**名字所指出的定义的活化。可按同一定义建立几个进程，其个数没有规定，并且可以并发地执行。每个进程被一个**样品值**唯一地确认，该**样品值**是开动表达式的结果，或THIS运算符的计算结果。进程的建立将引起它的局部说明的地点的建立，但那些带有STATIC属性的地点除外（见7.9节)，同时也引起了被局部定义的值和过程的建立。被局部说明的地点、值和过程被认为和它们所从属的、被建立的进程具有同一个活化。虚拟的最外层进程(见7.8节),即正在执行中的整个CHILL程序，被认为是由一个开动表达式建立的，开动表达式由系统执行，而程序是在系统控制下执行的。在建立进程时，它的形式参数（若有的话）是由开动表达式中相应的实在参数提供的值和地点。

执行停止动作，或达到进程体的末尾，或达到进程定义尾部指明的处理程序的一个异常处理选择对象的末尾(执行失败)，都可以使进程**终止**。如果虚拟的最外层进程执行一个停止动作或执行失败,则当且仅当所有它的下属进程（即由其中的开动表达式建立的进程）都结束时，它才被完全地终止。

在CHILL编程这一级上，进程总是处于两个状态之一：是**活化**的（即正在执行中),或**被延迟**的（即等待一个条件被满足)。从活化转向延迟称为该进程**被延迟**，从被延迟转向活化称为该进程的**重新活化**。

8.2 互斥和区域

8.2.1 概述

区域（见7.7节）是一种手段，可使进程以互斥方式访问区域中说明的地点。根据静态的上下文条件（见8.2.2节)，进程（不是虚拟的最外层进程）要访问区域内说明的地点，只能通过调用在该区域内部定义并被该区域开放的过程来实现。

在下列两种情况下，可以说一个**过程**名字标志了一个**关键过程**（此时它是一个**关键过程**名字)：当且仅当它是在一个区域内部定义的，并且是该区域所开放的，或具有同样过程定义的**过程**名字（见7.4节）是**关键的**(只当涉及入口定义时，后者才有意义）。

当且仅当控制不处于一个区域的任何**关键**过程中，又不在该区域本身中（当范围初始化正在进行时),则可以说该区域是**自由**的。

在下列情况时，区域被**封锁**（以防止并发执行）：

- 该区域被进入（注意：由于区域不包含在分程序中，不可能并发地进入这个区域)。
- 该区域的一个**关键**过程被调用。
- 在该区域中被延迟的进程被重新活化。

在下列情况时，区域将被**释放**，并重新成为自由的：

- 离开该区域。
- 从**关键**过程返回。
- **关键**过程执行一个动作，它使正在执行的进程被延迟(见8.3节)。在动态地嵌套的关键过程调用的情况下，只有最近封锁的区域被释放。

如果正当区域被封锁时，有一个进程企图调用它的一个**关键**过程,或企图进入该区域，则该进程被挂起，直到该区域被释放为止。（注意，在CHILL意义上，该进程仍然是处于活化状态的。）

当一个区域被释放时,可能有多于一个的进程已被挂起。挂起的原因是由于企图进入该区域，或由于企图调用它的一个**关键**过程，或由于企图在它的一个**关键**过程中被重新活化。此时将根据实现定义的调度算法，只能

有一个进程被选中进入该区域。

8.2.2 区域性

在一个区域内说明的地点只能通过调用关键过程或通过在进入区域时实现范围初始化而被访问。为了能静态地检验这一点，下列静态上下文条件必须满足：

- 在各相应的节中提到的区域性要求（赋值动作、过程调用、发送动作、结果动作）；
- 区域性过程不是通用的（见7.4节）。
- 关键过程既非通用的又非递归的(见7.3节)。

一个地点、值或过程名字可以是区域性的。这个性质的定义如下：

1. 地点

当且仅当下列条件中有任何一项满足时，一个地点称为是区域性的：

- 它是下列几种访问名字之一：
  - 文字上在一个区域内说明的地点名字，但不是在关键过程的形式参数中被定义的。
  - 一个地点等同名字，在它说明中的静态模式地点是区域性的，或者它是在一个区域性过程的形式参数中被定义的。
  - 一个有基名字，在它的说明中的约束或自由关联地点名字是区域性的。
  - 一个地点枚举名字，在相应的循环动作中的数组地点是区域性的。
  - 一个地点直接场名字，在相应的循环动作中的结构地点是区域性的。
- 它是一个非关联化约束关联，其中约束关联表达式是区域性的。
- 它是一个非关联化自由关联，其中自由关联表达式是区域性的。
- 它是一个非关联化行，其中行表达式是区域性的。
- 它是一个数组元素、子数组或数组切片，其中数组地点是区域性的。
- 它是一个串元素、子串或串切片，其中串地点是区域性的。
- 它是一个结构场，其中结构地点是区域性的。
- 它是一个地点过程调用，在地点过程调用中指明了过程名字，该过程名字是区域性的。
- 它是一个地点内部子程序调用，实现指明它是区域性的。
- 它是一个地点转换，其中的静态模式地点是区域性的。

2. 值

当且仅当一个值或表达式是区域性的原值，或包含区域性表达式的带括号表达式时，它们本身也是区域性的。

当且仅当下列条件之一得到满足时，一个原值是区域性的：

- 它是一个地点内容，该地点内容是区域性的，它的模式具有关联性质。
- 它是下列两种值名字之一：
  - 异名名字，在它的定义中的常量值是区域性的。
  - 值直接场名字，在相应的循环动作中的结构表达式是区域性的，它的模式具有关联性质。
- 它是包含一个数组多元组或结构多元组的多元组，其中至少有一个被指明的值出现是区域性的。
- 它是一个值数组元素、一个值子数值，或一个值数组切片，其中数组表达式是区域性的，且数组表达式的模式的元素模式具有关联性质。
- 它是一个值结构场，其中结构表达式是区域性的，并且该场的模式具有关联性质。
- 它是一个被关联地点，其中地点是区域性的。
- 它是一个表达式转换，其中表达式是区域性的。
- 它是一个值过程调用，在值过程调用中指明了过程名字，它是区域性的，它的结果模式具有关联性质。
- 它是一个值内部子程序调用，这可能是一个实现值内部子程序调用，该调用送返一个值，该值的类与模式相容，模式具有关联性质，并且实现指明它是区域性的。也可能是 ADDR (〈地点〉)，其中地点是区域性的。

3. 过程名字

当且仅当一个过程名字是在一个区域内部定义，但不是**关键**的（即不是被该区域开放的）时，该过程名字是**区域性**的。

8.3 进程的延迟

如果一个进程是活化的，它可以通过执行下列动作之一或计算下列表达式之一而变成被延迟的：

延迟动作（见6.16节）。若一个进程执行延迟动作，就变成了被延迟的。它带着一个优先级，作为一个成员加入到被延迟进程构成的集合中去，该集合附着于指明的事件地点。

延迟情况动作（见6.17节）。如果一个进程执行延迟情况动作，就变成了被延迟的，它带着指明的优先级，作为一个成员加入到每个由延迟进程构成的集合中去，每个这样的集合附着于在该延迟情况动作的延迟选择对象中指明的一个事件地点。

接收表达式（见5.2.18节）。在一个进程计算接收表达式时，当且仅当指明的缓冲区地点上既没有值，也没有被延迟的发送进程时，该进程将变成被延迟的。它作为一个成员加入到被延迟的接收进程构成的集合中去，该集合附着于指明的（空）缓冲区地点。

接收缓冲区情况动作（见6.19.3节）。在进程执行一个接收缓冲区情况动作时，当且仅当任何被指明的缓冲区地点上都没有值，也没有被延迟的发送进程，又未指明 ELSE 时，该进程将变成被延迟的。它作为一个成员加入到被延迟的接收进程构成的每个集合中去，每个这样的集合附着于该接收缓冲区情况动作的一个缓冲区接收选择对象中指明的一个缓冲区地点。

接收信号情况动作（见6.19.2节）。在进程执行一个接收信号情况动作时，当且仅当没有任何执行该接收信号情况动作的进程所接收的信号被挂起，并且仅当未指明 ELSE 时，该进程将变成被延迟的。它作为一个成员加入到被延迟进程构成的每个集合中去，每个这样的集合附着于信号接收选择对象中指明的一个**信号**名字上。

发送缓冲区动作（见6.18.3节）。在进程执行发送缓冲区动作时，当且仅当缓冲区地点的模式附有长度，且在发送动作之前缓冲区中值的个数已经等于该长度时，该进程将变成被延迟的。它带着指明的优先级，作为一个成员加入到被延迟进程组成的集合中去，该集合附着在缓冲区地点上。

当进程的控制处于一个**关键**过程之内时，如果该进程执行一个动作，该动作使进程被延迟，则相应的区域将被释放。过程的动态上下文条件将被保存起来，直到该进程在区域中被延迟的地方重新活化，此时区域又被封锁。

8.4 进程的重新活化

如果进程被延迟，则当且仅当另一个进程执行下列动作之一时，被延迟的进程将被重新活化。

继续动作（见6.15节）。当进程执行继续动作时，它重新活化另一个进程的充分必要条件是：在指明的事件地点上被延迟进程的集合是非空的。此时将根据实现定义的调度算法选出具有最高优先级的进程并使之活化。这个被重新活化的进程即被从所有的延迟进程的集合中除去。

发送缓冲区动作（见6.18.3节）。当进程执行发送缓冲区动作时，它重新活化另一个进程的充分必要条件是：在指明的缓冲区地点上被延迟的接收进程的集合是非空的。此时将根据实现定义的调度算法选出一个进程并使之活化。重新活化的进程即被从所有的延迟进程的集合中除去。如果在指明的缓冲区地点上被延迟接收进程的集合是空的，则只要缓冲区容量放得下，被送来的值将和指明的优先级一起存入缓冲区（见8.3节）。

发送信号动作（见6.18.2节）。当进程执行发送信号动作时，它重新活化另一个进程的充分必要条件是：对于指明的**信号**名字来说，被延迟进程的集合中有一个进程，它能接收该信号。此时将根据实现定义的调度算法选出一个进程并使之活化。重新活化的进程即从所有的延迟进程的集合中除去。如果没有一个被延迟的进程能接收该信号，则该信号将被挂起，同时被挂起的还有指明的优先级、可能的值表、进程名字和/或样品值。

接收缓冲区情况动作（见6.19.3节）。当进程执行接收缓冲区情况动作时，它重新活化另一个进程的充分必要条件是：至少有一个指明的缓冲区地点上被延迟发送进程的集合是非空的。在此情况下，它从缓冲区地点的值中，或从被延迟的发送进程的值中接收一个具有最高优先级的值。从缓冲区接收一个值时，该进程要把该值从缓冲区中除去，并根据实现定义的调度算法选出一个被延迟的发送进程（它的值具有最高的优先级），使之活

化。重新活化的进程即被从所有被延迟的发送进程的集合中除去，它的值和指明的优先级一起被存入缓冲区。直接从一个被延迟的发送进程接收一个值时，根据实现定义的调度算法选出一个带有该值的具有最高优先级的被延迟的进程，并使之活化。重新活化的进程即被从所有的延迟发送进程的集合中除去，它的值则被接收。

如果正当进程在一个关键过程内部活化时，它执行一个动作，该动作使另一个进程活化，则原来的进程仍然保持活化，即在该点不释放那个区域。

8.5 信号定义语句

语法：

〈信号定义语句〉::= (1)

SIGNAL 〈信号定义〉{,〈信号定义〉}*; (1.1)

〈信号定义〉::= (2)

〈名字〉〔=(〈模式〉{,〈模式〉}*)〕

〔TO 〈进程名字〉〕 (2.1)

语义： 信号定义定义了进程之间传递的值的组合函数和分解函数。在发送一个信号时，也传送被指明的值表。如果没有一个进程正在接收情况动作中等待这个信号，则这些值将被保留，直到有一个进程接收它们为止。

静态性质： 当且仅当一个名字是在一个信号定义中定义时，它是一个**信号**名字。**信号**名字具有如下的性质：

- 它附有一个任选的模式表，它们是在信号定义中提到的那些模式。
- 它附有一个任选的进程名字，就是在TO后面指明的进程名字。

静态条件： 信号定义中的任何模式都不能具有同步信质。

例子：

15.16 SIGNAL INITIATE=(INSTANCE),
TERMINATE; (1.1)

9.0 一般语义性质

9.1 模式检验

9.1.1 模式和类的性质

9.1.1.1 新鲜性

非形式定义

模式的新鲜性表明它是否是通过一个新模式定义语句定义的。模式的新鲜性可以是nil，即：它是一个(基)模式，不是通过一个新模式定义的；也可以是用来定义本模式的一个新模式名字。

定义

模式的新鲜性定义如下：

- 若该模式被一个新模式名字所标志，则它的新鲜性就是该新模式名字。
- 否则，若该模式被一个异名模式名字所标志，则它的新鲜性就是在它的定义中的定义模式的新鲜性。
- 否则，若该模式被一个参数化数组模式、参数化串模式或参数化结构模式所标志，则它的新鲜性分别是其中的原始数组模式名字、原始串模式名字或原始变体结构模式名字的新鲜性。
- 否则，若该模式被一个区段模式所标志，则它的新鲜性就是它的父本模式的新鲜性。
- 否则，若该模式被一个虚拟引进的父本模式所标志，它的新鲜性就是造成上述引进的新模式名字（见3.2.3节）。
- 否则，若该模式被READ〈模式〉所标志，则它的新鲜性就是〈模式〉的新鲜性。
- 否则，新鲜性为**nil**。

9.1.1.2 只读模式

非形式定义

如果某模式的一个地点作为整体来说是只读的，即无论是它自己，还是它的任何一部分都不能被写入新内容，则该模式称为是只读的。

定义

模式具有下列继承性质：当且仅当下列条件中有任何一项成立时，它是只读模式：

- 它被一个形式为READ〈模式〉的模式所标志。
- 它被一个参数化数组模式、参数化串模式或参数化结构模式所标志，其中原始数组模式名字、原始串模式名字或原始变体结构模式名字分别标志一个只读模式。

9.1.1.3 只读性质

非形式定义

如果一个模式的地点是只读的或包含只读的分量或子分量等，则该模式具有只读性质。

定义

当且仅当下列条件之一成立时，模式具有只读性质：

- 该模式是一个**模式名字**，它被一个具有**只读性质**的模式所定义。
- 该模式是一个**数组模式**，其**元素**模式具有**只读性质**，或它是一个结构模式，其中至少有一个**场**模式具有**只读性质**。
- 该模式是一个**只读模式**。

9.1.1.4 关联性质

非形式定义

如果一个模式的地点有关联模式或包含有关联模式的分量或子分量等，则该模式具有**关联性质**。

定义

当且仅当下列条件之一成立时，模式具有**关联性质**：
- 该模式是一个**模式名字**，它被一个具有**关联性质**的模式所定义。
- 该模式是一个**数组模式**，其**元素**模式有**关联性质**，或是一个结构模式，其中至少有一个**场**模式有**关联性质**。
- 该模式是一个**关联模式**。

9.1.1.5 带标签的参数化性质

非形式定义

如果一个模式的地点具有**带标签**的参数化结构模式，或包含有**带标签**参数化结构模式的分量或子分量等，则该模式有**带标签的参数化性质**。

定义

当且仅当下列条件之一成立时，模式具有**带标签的参数化性质**：
- 该模式是一个**模式名字**，它被一个具有**带标签参数化性质**的模式所定义。
- 该模式是一个**数组模式**，其**元素**模式具有**带标签参数化性质**，或是一个结构模式，其中至少有一个**场**模式具有**带标签参数化性质**。
- 该模式是一个**带标签**的参数化结构模式。

9.1.1.6 同步性质

非形式定义

如果一个模式的地点具有同步模式，或包含有同步模式的分量或子分量等，则该模式具有**同步性质**。

定义

当且仅当下列条件之一得到满足时，模式具有**同步性质**；
- 该模式是一个**模式名字**，它被一个具有**同步性质**的模式所定义。
- 该模式是一个**数组模式**，其**元素**模式具有**同步性质**，或者是一个结构模式，其中至少有一个**场**模式具有**同步性质**。
- 该模式是一个事件模式或缓冲区模式。

9.1.1.7 根模式

若M不是一个组合模式，则任一M值类或M导出类均有一**根**模式，其定义如下：
- 若M不是区段模式，则**根**模式为M。
- 若M是区段模式，则**根**模式是M的**父本**模式。

9.1.1.8 结果类

给定两个相容的类(见10.1.2.6节)，它们是完全类、M值类或M导出类，其中M是离散模式、幂集模式或串模式，则结果类定义如下：

- M导出类和N导出类的结果类是M导出类；
- 如果M不是区段模式，则M值类和N导出类的结果类是M值类，否则是P值类，其中P是M的父本模式；
- 如果M不是区段模式，则M值类和N值类的结果类是M值类，否则是P值类，其中P是M的父本模式；
- 完全类和任何其它类的结果类是后者（即其它类）。

给定一组两两相容的类 Ci (i = 1,…, n)，这组类表的结果类可递归地定义如下：若n>1，则先求 Ci (i = 1,…, n-1) 的结果类，再求此结果类和类 Cn的结果类，否则是C_1和C_1的结果类。

（注意，根据CHILL的定义，取类 Ci 的次序是无关紧要的，即所有这样的结果类都是相容的。）

9.1.2 模式和类的关系

在以下几节中定义了模式之间、类之间，以及模式和类之间的相容性关系。在整个文本中这些关系被用来定义静态条件。

相容性关系本身是用一些其它关系定义的，它们主要为了上述目的而在第9章中被使用。

9.1.2.1 “被定义”关系

非形式定义

一个模式名字被说成是被它的定义模式所定义的，并且可以类推，即如果后者也是一个模式名字，则前者也是被它的定义模式的定义模式所定义。

定义

说模式名字N被模式M所定义的充分必要条件是：

- M是N的定义模式。
- N的定义模式是被M定义的模式名字。

9.1.2.2 模式的等价关系

概述

非形式定义

在描述相容性关系时，要用到下列等价关系：

- 如果两个模式属于同一种，即如它们具有相同的继承性质，则称它们是相似的。
- 如果两个模式是相似的且具有相同的新鲜度，则称它们是**V**等价的（值等价）。
- 如果两个模式是V等价的，并且还考虑到存储中值的表示方面或最小存储量方面的可能差别，则称它们是等价的。
- 若两个模式是等价的，并且具有相同的只读说明，则称它们是**I**等价的(地点等价)。

定义

下面各节中，将以（部分）关系集合的形式给出模式的等价关系。如果求关系集合的对称、自反和传递闭包，即可得到完全的等价算法。在关系中提到的模式可以是虚拟地引进的或动态的。在后一种情况下，完全的等价性检验只能在运行时进行。如果动态部分检验失败，将导致RANGEFAIL或TAGFAIL异常（见相应各节）。

对两个递归模式进行任何等价性检验，都要求对定义它们的递归模式集合中相应通路的相应模式进行检验。如果没有发现矛盾，则两个模式是等价的（因此，如果在比较中遇到了两个过去已经比较过的模式，则检验算法的通路被成功地结束）。

“相似”关系

当且仅当下列条件之一成立时，两个模式是**相似的**：

- 它们是整模式；
- 它们是布尔模式；
- 它们是字符模式；
- 它们是集合模式，并且所定义的值的个数相同，还定义了相同的**集合元素**名字。对于相同的名字，NUM内部子程序调用提供相同的值；
- 它们是具有**相似的父本**模式的区段模式；
- 其中一个是区段模式，它的**父本**模式**相似**于另一模式；
- 其中一个是布尔模式，另一个是长度为1的**字位**串模式；
- 其中一个是字符模式，另一个是长度为1的**字符**串模式；
- 它们是幂集模式，它们的**成员**模式是**等价的**；
- 它们是约束关联模式，它们的**被关联**模式是**等价的**；
- 它们是自由关联模式；
- 它们是行模式，它们的**被关联原始**模式是**等价的**；
- 它们是过程模式，并且
  1. 它们有相同个数的参数说明，（按位置）对应的参数说明具有**I等价**模式，相同的参数属性，以及相同的寄存器说明，若有的话。
  2. 它们都有或者都没有结果说明。如果有，则两个结果说明应有**I等价**模式、有相同的属性和相同的寄存器说明，若有的话。
  3. 它们有相同的**异常**名字集合；
  4. 它们有相同的递归性；
- 它们是样品模式；
- 它们是事件模式，都没有长度或有相同的长度；
- 它们是缓冲区模式，并且：
  1. 它们都没有长度或有相同的长度；
  2. 它们有**I等价**的**缓冲区元素**模式；
- 它们是串模式，并且：
  1. 它们都是**字位**串模式或都是**字符**串模式；
  2. 它们有相同的长度。如果有一个（或两个）模式，是动态的，则检验也是动态的。检验失败将导致RANGEFAIL异常；
- 它们是数组模式，并且：
  1. 它们的**下标**模式是**V等价的**；
  2. 它们的**元素**模式是**等价的**；
  3. 它们的元素布局是等价的（见9.1.2.2节）；
  4. 它们的元素个数相同，如果有一个（或两个）模式是动态的，则检验也是动态的。检验失败将导致RANGEFAIL异常；
- 它们是结构模式，但不是参数化结构模式，并且；
  1. 它们的场的个数相同 （按位置）相应的场是**等价的**（见9.1.2.2节）；
  2. 如果它们都是**可参数化变体**结构模式，则它们的类表必须是相容的；
- 它们是参数化的结构模式，并且：
  1. 它们的**原始变体**结构模式是**相似的**；
  2. 它们的（按位置）对应值必须是相同的。如果有一个（或两个）模式是动态的，则检验也是动态的。检验失败将导致TAGFAIL异常。

“V等价”关系

当且仅当两个模式是类似的，并且具有相同的新鲜性时，它们是**V等价的**。

“等价”关系

当且仅当两个模式是V等价的，并且下列条件也满足时，它们是等价的：

- 如果其中一个模式是布尔模式，则另一模式也必须是布尔模式；
- 如果其中一个模式是字符模式，则另一模式也必须是字符模式；
- 如果其中一个模式是区段模式，则另一模式也必须是区段模式，且两个上界必须相等，两个下界也必须相等。

“I等价”关系

当且仅当两个模式是等价的，并且如果其中一个模式具有只读性质，则另一模式也必须有只读性质，且下列条件也满足时，它们是I等价的：

- 如果两者都是约束关联模式，则它们的被关联模式必须是I等价的；
- 如果两者都是行模式，则它们的被关联原始模式必须是I等价的；
- 如果两者都是数组模式，则它们的元素模式必须是I等价的；
- 如果两者都是结构模式，则（按位置）对应的场必须是I等价的；

场的“等价”和“I等价”关系

在两个给定的结构模式的上下文范围内，如果有两个场，它们都是固定场，并且1.是等价的，2.是I等价的；或者它们都是选用场，并且1.是等价的，2.是I等价的，则这两个场也是1.等价的，2.I等价的。反之亦然。

对于（相应的）固定场、变体场、选用场、和变体选择对象，其“等价”关系和“I等价”关系分别递归地定义如下：

1. **固定场和变体场**
   a. 两个场必须有等价的布局。
   b. 两个场模式必须是1.等价的，2.I等价的。
2. **选用场**
   a. 两个选用场都有标签或都没有标签。在前一种情况下，双方标签必须有相同数量的标签场名字，且（按位置）相应的标签场名字必须标志相应的固定场。
   b. 双方必须有相同数量的变体选择对象，且（按位置）对应的变体选择对象必须是1.等价的，2.I等价的。
   c. 双方都未指明ELSE，或双方都指明了ELSE。在后一种情况下，后面必须跟有相同数量的变体场，且（按位置）对应的变体场必须是1.等价的，2.I等价的。
3. **变体选择对象**
   a. 两个变体选择对象必须有相同数量的情况标号表，且（按位置）对应的情况标号表必须都是无关紧要，或都是（ELSE），或定义有相同的值集。
   b. 两个变体选择对象必须有相同数量的变体场，且（按位置）对应的变体场必须是1.等价的，2.I等价的。

布局的“等价”关系

下面假定每个地位的形式是：

POS（〈字号〉，〈起始位〉，〈长度〉）

并且每个步的形式是：

STEP（〈地位〉，〈步长〉，〈图长〉）

3.10.6 节给出了把地位或步变成所要求的形式的相应规则。

1. **场布局**

如果两个场布局都是NOPACK，或都是PACK，或都是地位，则它们是等价的。在后一种情况下，其中一个地位必须等价于另一个（见下面）。

2. **元素布局**

如果两个元素布局都是NOPACK，或都是PACK，或都是步，则它们是**等价的**。在后一种情况下，两个步中的地位必须等价(见下面)，两个元素布局的NUM(步长)必须提供同一个值，它们的NUM(图长)也必须提供同一个值。

3. **地位**

当且仅当两个NUM(字号)出现提供同一个值、两个NUM(起始位)出现提供同一个值、两个NUM(长度)出现也提供同一个值时，这两个地位也是等价的。

9.1.2.3 “读相容”关系

非形式定义

当且仅当模式M和模式N是等价的，且M和它的可能的(子)分量有更严格的只读说明时，则称M和N**读相容**。这种关系是非对称的。

例子：

READ REF READ CHAR和REF CHAR是**读相容的**。

定义

当且仅当M和N是等价的，且若N是只读模式，则M也必须是只读模式，并且在下列条件成立时，M被称为和N**读相容的**。

- 若M和N是约束关联模式，则M的被关联模式与N的被关联模式**读相容**；
- 若M和N是行模式，则M的被关联原始模式必须与N的被关联原始模式**读相容**；
- 若M和N是数组模式，则M的元素模式必须与N的元素模式**读相容**；
- 若M和N是结构模式，则M的每个场模式必须与N的相应场模式**读相容**；

9.1.2.4 “可限制为”关系

非形式定义

“可限制为”关系涉及具有关联性质的等价模式。我们说模式M**可限制为**模式N，如果模式M或它的可能的子分量所关联的地点比N所关联的地点在只读说明方面的限制相同或稍少，因此这个关系是非对称的。该关系用于赋值中(见9.1.2.5节)。

例子：

REF INT **可限制为** REF READ INT

STRUCT(P REF BOOL) **可限制为** STRUCT(Q REF READ BOOL)

定义

当且仅当下列条件之一成立时，模式M是**可限制为**模式N的(一种非对称的关系)：

- M不具有关联性质且M**等价于**N。
- M和N是约束关联模式，且N的被关联模式与M的被关联模式**读相容**。
- M和N是自由关联模式且M和N**等价**。
- M和N是行模式，且N的被关联原始模式与M的被关联原始模式**读相容**。
- M和N是数组模式，且M的元素模式**可限制为**N的元素模式。
- M和N是结构模式，且M的每个场模式**可限制为**N的相应的场模式。

9.1.2.5 模式和类的相容性

- 任何模式M都和**完全类相容**；
- 当且仅当M是一个关联模式，或一个过程模式，或一个样品模式时，M和**空类相容**；
- 当且仅当模式M是一个关联模式，且有下列条件之一成立时，M和N关联类**相容**：
  1. N是静态模式，M是约束关联模式，它的被关联模式与N**读相容**；
  2. N是静态模式，M是自由关联模式；
  3. M是行模式，其被关联原始模式是V，并且：

- 若V是一个串模式，N也必须是串模式，并且V(P)和N是**读相容**的。其中P是N的（可能为动态的）长度；
- 若V是一个数组模式，N也必须是数组模式，且V(P)应与N**读相容**，其中P是N的（可能为动态的）上界；
- 若V是一个**变体结构**模式，则N必须是参数化的结构模式，且V(P_1，…，Pn)与N**读相容**，其中P_1，…，Pn标志N的值表；

- 当且仅当M和N是**相似**模式时，模式M与N导出类相容；
- 当且仅当有下列条件之一成立时，模式M与N值类**相容**：
  1. 若M不具有关联性质，则M和N必须是**V等价**的；
  2. 若M具有关联性质，则N必须**可限制为**M。

9.1.2.6 类的相容性

- 任何类与自己**相容**。
- **完全**类与任何其它类**相容**。
- **空**类与任何M关联类**相容**。
- 当且仅当M是一个关联模式、过程模式或样品模式时，**空**类与M导出类或M值类**相容**。
- 当且仅当M关联类与N关联类**等价**时，这两个类**相容**。若M与/或N（都）是动态模式，则等价检验的动态部分不起作用，即不会出现异常。
- 当且仅当N是一个关联模式，且有下列条件之一成立时，M关联类与N导出类或N值类**相容**：
  1. M是静态模式，N是约束关联模式，它的**被关联模式等价于**M。
  2. M是静态模式地点，N是自由关联模式。
  3. N是行模式，它的**被关联原始**模式V满足下列条件：
     - 若V是串模式，M也必须是串模式，且V(P)**等价于**M，其中P是（可能为动态的）M的长度；
     - 若V是数组模式，M也必须是数组模式，且V(P)**等价于**M，其中P是M的（可能为动态的）上界；
     - 若V是**变体**结构模式，则M必须是参数化结构模式，且V(P_1，…，Pn)**等价于**M，其中P_1，…，Pn标志M的值表。
- 当且仅当M和N**相似**时，M导出类和N导出类或N值类相容。
- 当且仅当M和N为**V等价**时，M值类和N值类相容。

当且仅当两个类表具有相同数目的类，并且（按位置）对应的类是**相容**时，这两个类表也是**相容**的。

9.1.3 情况选择

语法：

```
〈情况标号说明〉::=                                    (1)
    〈情况标号表〉{,〈情况标号表〉}*                    (1.1)
〈情况标号表〉::=                                      (2)
    (〈情况标号〉{,〈情况标号〉}*)                      (2.1)
  | 〈ELSE〉 | 〈无关紧要〉                             (2.2)
〈情况标号〉::=                                        (3)
    〈离散直接量表达式〉                               (3.1)
  |〈直接量区段〉                                      (3.2)
  | 〈离散模式名字〉                                   (3.3)
〈无关紧要〉::=                                        (4)
    (*)                                               (4.1)
```

语义： 情况选择是从选择对象表中选出选择对象的手段。这个选择是以一个指明的选择值表为基础的。

情况选择可以应用于：

- 选用场（见3.10.4节），在此情况下选出的是一组变体场。

- 有标号的数组多元组（见5.2.5节），在此情况下选出的是一个数组元素值。
- 情况动作（见6.4节），在此情况下选出的是一个动作语句表。

在第一种和最后一种情况下，每个选择对象被标以一个情况标号说明；在有标号的数组多元组中，每个值被标以一个情况标号表。为便于描述起见，在本节中把有标号的数组多元组看成是只有一个情况标号表出现的情况标号说明。

被情况选择选出的选择对象是其前面的情况标号说明与选择值表匹配的那一个。(选择值的个数恒等于情况标号说明中情况标号表出现的个数。)当且仅当值表中的每个值与情况标号说明中(按位置)对应的情况标号表匹配时，则称该值表与该情况标号说明相匹配。

值与情况标号表相匹配的充分必要条件是：

- 情况标号表由情况标号组成，值是由情况标号之一显式地说明的值之一。
- 情况标号表由（ELSE）组成，值是由（ELSE）隐式地说明的值之一。
- 情况标号表由无关紧要组成。

被一个情况标号**显式地**说明的值是由任何离散表达式提供的值，或由直接量区段或离散模式名字定义的值。被（ELSE）**隐式地**说明的值是所有符合下列条件的可能的选择值：它们不是任何情况标号说明中的任何相应的情况标号表（即属于同一个选择值）所指示的值。

静态性质：

- 带情况标号说明的选用场、有标号的数组多元组或情况动作，都附有一个情况标号说明表。该表由每个变体选择对象、值或情况选择对象前面的情况标号说明所组成。
- 每个情况标号附有一个类。如果它是离散直接量表达式，则这个类就是该离散直接量表达式的类；如果它是直接量区段，则这个类就是该直接量区段中所有离散直接量表达式的类的结果类；如果它是离散模式名字，则这个类是M值类的结果类，其中M即该离散模式名字。
- 每个情况标号表附有一个类。若它是（ELSE）或〈无关紧要〉，则该类是**完全**类，否则是所有情况标号的类的结果类。
- 每个情况标号说明有一个类表，其中的类就是情况标号表中的那些类。
- 每个情况标号说明表附有一个结果类表（前提是：各情况标号说明要有相同数量的类；这总是成立的）。构造结果类表的方法是：对表中的每个位置，构造占有此位置的所有类的结果类。

当且仅当对每一个可能的选择值的表都存在一个情况标号说明，且与该选择值的表匹配时，则情况标号说明表是**完全的**。所有可能的选择值组成的集合可由上下文按如下方式确定：

- 对于**带标签变体**结构模式，它是相应的**标签**场的模式定义的值的集合。
- 对于**无标签变体**结构模式，它是相应结果类（此类决不可能是**完全**类，见3.10.4节）的**根**模式定义的值的集合。
- 对于数组多元组，它是该数组多元组模式的**下标**模式定义的值的集合。
- 对于带区段表的情况动作，它是该区段表中相应离散模式定义的值的集合。
- 对于不带区段表的情况动作，它是M定义的值的集合，其中相应选择的类是M值类或M导出类。

静态条件： 对于每个情况标号说明来说，情况标号表出现的个数必须相同。

对任意两个情况标号说明出现，它们的类表必须相容。

情况标号说明出现的表必须是**一致的**，即每个可能的选择值的表至多与一个情况标号说明匹配。

例子：

```
11.7   (occupied)                                    (3.1)
11.62  (rook),(*)                                    (1.1)
8.24   (ELSE)                                        (2.2)
```

9.1.4 语义范畴的定义和概况

本节给出所有语义范畴的概况。它们已在语法描述中通过下划线指明。如果这些语法范畴未在适当的节中定义，则它们的定义在这里给出，否则只提示有关的节号。

9.1.4.1 名字

模式名字

数组模式名字：　由数组模式定义的名字。
布尔模式名字：　由布尔模式定义的名字。
约束关联模式名字：　由约束关联模式定义的名字。
缓冲区模式名字：　由缓冲区模式定义的名字。
字符模式名字：　由字符模式定义的名字。
离散模式名字：　由离散模式定义的名字。
事件模式名字：　由事件模式定义的名字。
自由关联模式名字：　由自由关联模式定义的名字。
样品模式名字：　由样品模式定义的名字。
整数模式名字：　由整数模式定义的名字。
模式名字：　见3.2.1节。
新模式名字：　见3.2.3节。
参数化数组模式名字：　由参数化数组模式定义的名字。
参数化串模式名字：　由参数化串模式定义的名字。
参数化结构模式名字：　由参数化结构模式定义的名字。
幂集模式名字：　由幂集模式定义的名字。
过程模式名字：　由过程模式定义的名字。
区段模式名字：　由区段模式定义的名字。
行模式名字：　由行模式定义的名字。
集合模式名字：　由集合模式定义的名字。
串模式名字：　由串模式定义的名字。
结构模式名字：　由结构模式定义的名字。
异名模式名字：　见3.2.2节。
变体结构模式名字：　由变体结构模式定义的名字。

访问名字

有基名字：　见4.1.4节。
地点名字：　见4.1.2,7.4节。
地点直接场名字：　见6.5.4节。
地点枚举名字：　见6.5.2节。
地点等同名字：　见4.1.3,7.4节。

值名字

异名名字：　见5.1节。
值直接场名字：　见6.5.2节。
值枚举名字：　见6.5.4节。
值接收名字：　见6.19.2，6.19.3节。

其它名字

约束或自由关联地点名字：　具有约束关联模式或自由关联模式的地点名字。
内部子程序名字：　指示实现定义的内部子程序的任何实现定义的名字。
场名字：　见3.10.4节。
通用过程名字：　其通用性为通用的过程名字。
标号名字：　见6.1节。
模块名字：　见7.6节。
非保留名字：　一个名字，它不是附录C1中提到的保留名字之一。
过程名字：　见7.4节。
进程名字：　见7.5节。

区域名字：　　见7.7节。
寄存器名字：　　由实现定义的机器寄存器的名字。
保留名字表：　　仅由保留名字组成的名字表（见附录C1）。
集合元素名字：　　见3.4.5节。
信号名字：　　见8.5.2节。
标签场名字：　　见3.10.4节。
未定义异名名字：　　见5.1节。

9.1.4.2 地点

数组地点：　　具有数组模式的地点。
缓冲区地点：　　具有缓冲区模式的地点。
事件地点：　　具有事件模式的地点。
样品地点：　　具有样品模式的地点。
串地点：　　具有串模式的地点。
结构地点：　　具有结构模式的地点。

9.1.4.3 表达式

数组表达式：　　其类与数组模式相容的表达式。
布尔表达式：　　其类与布尔模式相容的表达式。
约束关联表达式：　　其类与约束关联模式相容的表达式。
离散表达式：　　其类与离散模式相容的表达式。
离散直接量表达式：　　离散表达式是一个**直接量**。
自由关联表达式：　　其类与自由关联模式相容的表达式。
样品表达式：　　其类与样品模式相容的表达式。
整数表达式：　　其类与整数模式相容的表达式。
整数直接量表达式：　　整数表达式是一个**直接量**。
幂集表达式：　　其类与幂集模式相容的表达式。
过程表达式：　　其类与过程模式相容的表达式。
行表达式：　　其类与行模式相容的表达式。
串表达式：　　其类与串模式相容的表达式。
结构表达式：　　其类与结构模式相容的表达式。

静态条件：　布尔表达式或离散表达式（如在语法中指出了这样的表达式）都不得具有动态类。即对表达式是否与布尔模式或离散模式相容的检验可以静态进行。

9.1.4.4 其它语义范畴

实现值内部子程序调用：　　见11.1.3节。
地点过程调用：　　见6.7节。
模块动作语句：　　直接包含模块动作的动作语句。
非撇号字符：　　不是撇号的字符。
值过程调用：　　见6.7节。

9.2 可见性和名字的约束

9.2.1 概述

在7.1节中提到的特殊的CHILL结构在程序内部建立了新的名字。程序结构语句和可见性语句确定名字在整个程序中的可见性。本节讨论名字的可见性，但异常名字除外。即每个名字都被认为是不在异常名字的上

下文之中出现的。关于**异常名字**见第10章。

为了能精确地描述程序的可见性结构，在名词使用方面作了如下进一步的规定，它们仅对9.2节有效：

- (一个名字的)**名字串**是字符串(用作该名字的标志)，它被看作是孤立于任何上下文之外的一个词法元素。

一个**名字**是与一个名字串的定义相联系的那个名字串（定义性出现，见9.2.2节)。

例子

```
B:  BEGIN
    MODULE DCL I INT; END;
    MODULE DCL I PTR; END;
    END B;
```

在以B为标号的程序块的分程序体中引进了**两个名字**，它们的名字串都是I。

在一个范围内，每个名字具有下列四种可见性等级之一：

表 I 可 见 性 等 级

可　　见　　性	性　　质（非　形　式　的）
直 接 强 可 见	名字对于建立点、开放点或引进点都是可见的
间 接 强 可 见	名字通过分程序嵌套或它的渗透属性来继承
弱　　可　　见	被强可见名字隐含的名字
不　　可　　见	不能应用的名字

直接强可见的名字和**间接强可见**的名字统称**强可见**名字。**弱可见**的名字和**强可见**名字统称**可见的**名字，其它名字是**不可见**的。程序结构语句和可见性语句唯一地确定每个名字属于哪一个可见性类。可见性类的确切性质将在下列各节中解释。

名字约束是使每个名字串与一个唯一的名字相联系的机制，即赋予名字串以唯一的意义。

9.2.2 可见性和名字的建立

名字通过 7.1 节中提到的语言结构来建立。除了**场名字**和**集合元素**名字外，所有的名字都有一个唯一的**定义性出现**，这就是引入该名字的语言结构。为了统一处理名字的可见性的确立和名字的约束，规定了下面的机制，它给每个被建立的名字以唯一的定义性出现：

- 在一个组块的范围内，每个模式出现被认为是在该范围内定义的一个虚拟的**异名模式**名字的应用性出现。

对于过程定义来说，结果模式的虚拟异名模式定义位于包含此过程的组块的范围内。形式参数模式的虚拟异名模式定义位于该过程的范围中。

在应用可见性和名字约束规则时，必须考虑这些虚拟定义。

例子：

```
DCL  I SET (A, B),
     K INT,
     J ARRAY (SET (A, B)) INT;
```

可替换成：

```
SYNMODE  &1 = SET (A, B),  &2 = INT;
         &3 = ARRAY (&1) &2;
DCL I &1, K &2, J &3;
```

&1，&2和&3是虚拟的**异名模式**名字。可见性规则被应用于这些虚拟的置换。虚拟置换的结果是，建立名字的模式（SET，STRUCT）在一个范围内只出现一次，即在虚拟的异名模式定义的右边。这个异名模式定义被认为是各**集合元素**名字或各场名字的**唯一的定义性出现**。

场名字的可见性和名字约束的性质不同于其它名字的相应性质(比它们简单)。因此，在 9.2 节的剩余部分

中，“名字”一词不包括场名字，除非是特别指明。

9.2.3 隐含名字

一个范围内的每个强可见名字都有一个（可能为空的）隐含名字的集合，定义如下：

- 每个模式有一个（可能为空的）隐含名字的集合，如表2中所示。

一个（强可见）名字的隐含的名字是：

- 如果该名字是访问名字，则隐含的名字是由该访问名字的模式所隐含的那些名字。
- 如果该名字是**模式**名字，则隐含的名字是由该定义模式所隐含的那些名字。
- 如果该名字是**过程**名字，则隐含的名字是由结果说明的模式所隐含的那些名字。
- 如果该名字是**信号**名字，则隐含的名字是由附属于它的那些模式所隐含的所有名字。
- 否则，没有隐含的名字。

表2 模 式 的 隐 含 名 字

模　　式	隐 含 名 字 的 集 合
INT, BIN, CHAR INSTANCE, PTR BOOL, EVENT CHAR(n), BIN(n) BIT(n), RANGE(…)	空
模式名字	它的定义模式所隐含名字的集合
M（m:n）	M隐含名字的集合
REF M, ROW M READ M, POWERSET M PROC(…)(M) BUFFER M	M隐含名字的集合
ARRAY (M) N	M和N隐含名字的集合的并集
STRUCT (N_1M_1, …, N_nM_n)	M_1到M_n所隐含名字的集合的并集。对于变体结构，它是变体结构的所有场隐含名字的集合的并集。
参数化的VM（…）	VM隐含名字的集合
SET（……）	集合元素名字的集合

（注意，隐含名字总是集合元素名字，它们自己不会再隐含别的名字。）

9.2.4 范围内的可见性

在一个范围内强可见的名字是**直接强可见的**，或**间接强可见的**。

只有在下列情况时，一个名字才在一个范围内是**直接强可见的**：

- 该名字在此范围中有定义性出现。
- 该名字被引进此范围中（见9.2.6.3节）。
- 该名字被向此范围开放（见9.2.6.2节）。

只有在下列情况时，一个名字才在一个范围内是**间接强可见的**：

- 此范围是一个程序块范围，该名字是被继承的（见9.2.5节）。
- 该名字在外层范围中是强可见的，并在该外层范围中具有**渗透性质**（见9.2.6.2节），并且在该范围中不存

在具有相同名字串的**直接强可见**的名字。在此情况下，该名字在此范围中也具有**渗透性质**。

只有在下列情况时，一个名字才在一个范围内是**弱可见**的。

- 该名字被此范围中的一个强可见名字所隐含。

可见性规则是这样定义的，它使得在每个范围中，所有强可见的名字都有不同的名字串。但是，两个或更多的弱可见名字可以有相同的名字串。因此，在某些情况下，这样的名字是不能应用的(见9.2.8节)。

9.2.5 可见性和程序块

下列的可见性规则适用于程序块。

- 如果一个名字在一个组块的范围内是**强可见**的，则在每个直接包含于其中的程序块的范围内，只要该程序块内不存在具有相同名字串的直接强可见名字，则该名字是**间接强**可见的（名字通过程序块**继承**)。

9.2.6 可见性和模片

9.2.6.1 概述

语法：

〈可见性语句〉::= (1)

〈开放语句〉 (1.1)

| 〈引进语句〉 (1.2)

语义： 可见性语句只能在模片的范围中出现，它控制着在这些范围中显示地提到的名字（也隐式地控制它们隐含的名字)。

9.2.6.2 开放语句

语法：

〈开放语句〉::= (1)

GRANT〈开放窗口〉[PERVASIVE]; (1.1)

〈开放窗口〉::= (2)

〈开放元素〉{,〈开放元素〉}* (2.1)

| ALL (2.2)

〈开放元素〉::= (3)

〈非保留名字〉 (3.1)

| 〈新模式名字〉〈禁止子句〉 (3.2)

〈禁止子句〉::= (4)

FORBID{〈禁止名字表〉| ALL} (4.1)

〈禁止名字表〉::= (5)

(〈场名字〉{,〈场名字〉}*) (5.1)

语义： 开放语句是用于把一个模片范围中的名字的可见性扩展到它的直接外层范围中去的手段。只有当**新模式名字**是结构模式时，才能为它们指明FORBID。这意味着，该模式的所有地点和值的场只能在开放模块的内部选出，而不是在它的外部选出。

下列可见性规则是适用的：

- 在一个模片的范围中**可见**的名字，如果在该模片范围的开放语句中被提到，则此名字在直接外层组块的范围中是**直接强**可见的。该名字称为是向外层范围开放的。
- 记号FORBID ALL表示禁止新模式名字的所有场名字的语法简写（见9.2.7节)。
- 记号GRANT ALL [PERVASIVE] 是一个语法简写，表示开放所有这样的名字（并具有渗透性质，如果指明了的话)，这些名字在开放模片的范围中是强可见的，它们的定义性出现就在开放模片之中。

静态性质： 一个具有属性PERVASIVE的被开放的名字，在外层范围中有**渗透性质**。

静态条件： 任何非保留名字的定义性出现必须位于开放模片之中。

带有FORBID说明的新模式名字的定义性出现必须在开放模片的范围内，该新模式名字必须是一个结构模式，并且禁止名字表中的每个场名字必须是新模式名字的一个场名字。

如果开放语句位于一个区域的范围中，它不得开放区域性的值名字，或区域性的访问名字。

例子：

```
1.11  GRANT add, mult;                                    (1.1)
```

9.2.6.3 移入语句

语法：

```
〈引进语句〉::=                                            (1)
    SEIZE 〈引进窗口〉;                                    (1.1)
〈引进窗口〉::=                                            (2)
    〈引进元素〉{, 〈引进元素〉}*                           (2.1)
  | ALL                                                   (2.2)
〈引进元素〉::=                                            (3)
    〈模片名〉 ALL                                         (3.1)
  | 〈非保留名字〉                                         (3.2)
〈模片名字〉::=                                            (4)
    〈模块名字〉                                           (4.1)
  | 〈区域名字〉                                           (4.2)
```

语义： 引进语句是用于把组块范围中的名字的可见性扩展到直接包含于组块中的模片的范围中去的手段。

以下的可见性规则是适用的：

- 在一个组块的范围中可见的名字，如果在直接包含于该组块中的模片范围中的引进语句中被提到，则该名字在此模片范围中是直接强可见的，也就是说该名字被引进了此模片范围之中。
- 如果外层范围中具有渗透性质的名字被引进，则它在引进模块的范围中是直接强可见的，并且还保持其渗透性质。
- 记号SEIZE ALL是一个语法简写，用于引进所有这样的名字，它们在外层组块的范围中是强可见的，并且它们的定义性出现在引进模片之外。
- 记号 SEIZE〈模片名字〉ALL是一个语法简写，用于引进所有这样的名字，它们在外层组块的范围中是强可见的，并且被模片名字标志的模块或区域所开放。

静态条件： 任何非保留名字或模片名字的定义性出现必须在引进模片之外。

在引进元素中提到的名字不得是值直接场名字，也不得是地点直接场名字。

例子：

```
15.14  SEIZE/* external signals*/
           ACQUIRE, RELEASE, CONGESTED, STEP, READOUT;    (1.1)
```

9.2.7 场名字的可见性

只有在下列上下文中，场名字才可以在它们的定义性出现之外出现：

- 结构地点或结构值的场选择。
- 带标号的结构多元组。
- 开放语句中的禁止子句。

在头两条上下文中，那些附着于结构地点、(强)结构值或多元组的模式的场名字是可见的，除非该模式的新鲜性是一个新模式名字，并且此新模式名字是在附有禁止子句的情况下被一个模片所开放的。在后一种情况下，只有那些未在禁止子句中提到的场名字是在开放模片外可见的。

在最后一条的上下文中，被开放的新模式名字的全部场名字并且只有这样场名字是可见的。

9.2.8 名字的约束

名字约束是用来使单个名字与名字串的每个出现相结合的一种机制。

约束规则取决于该名字串出现在下列哪一种上下文中：

1. 命令名字。
2. 异常名字。
3. 保留名字。
4. 场名字。
5. 引进元素中的非保留名字、模块名字或区域名字。
6. 任何其它名字。

约束规则

1. 命令名字串遵守实现定义的约束方式且不影响CHILL的约束规则（见2.6节）。
2. 异常名字串是按照11.3节中给出的处理程序识别规则来处理的。
3. 如果保留名字串不在它所出现的编译单位中被释放命令所释放，则仍有其保留的意义。如果被释放，则遵守6.中的规则。但即使在一个编译单元中被释放也不能向该单位之外开放。
4. 场名字串的约束方式如下，它与10.2.7节中提到的上下文有关。
   - 约束于结构地点或（强）结构表达式的结构模式的可见的场名字；
   - 约束于（强）多元组的结构模式的可见的场名字；
   - 约束于新模式名字的可见的场名字。

   如果该名字串不能约束于这样的一个场名字，则程序是错的。
5. 在引进元素的上下文中出现的名字串是根据上述6项中提到的规则被约束的。它被约束于该引进语句所在范围的直接外层范围中。
6. 对于名字串在一个组块范围内的任何其它出现来说：
   a. 如果在此范围中有多于一个的强可见名字都具有此名字串，则程序是错误的；
   b. 否则，如果在此范围中有一个强可见名字具有此名字串，则此名字串约束于该名字；
   c. 否则，如果在此范围中恰有一个弱可见名字具有此名字串，则此名字串约束于该名字；
   d. 否则，如果在此范围中有多于一个的弱可见名字具有此名字串，且所有那些（集合元素）名字的类都相容，则此名字串约束于那些名字中的（任意的）一个；
   e. 否则，程序是错误的。

除了上面提到的规则之外，出现在引进语句或开放语句中的名字串应该约束于这样的一个名字，它的定义性出现分别位于开放模片之内或引进模片之外（如果根据规则d.选择的话）。

10.0 异常处理

10.1 概述

异常可以是语言定义的异常，它可以具有语言定义的名字，可以是用户定义的异常，也可以是实现定义的异常。语言定义的异常是由动态条件的动态违例而引发的。所有带名字的异常都可由执行引发动作而被引发。

当异常被引发后，可以被处理，即执行相应的处理程序的动作语句表。

异常处理是这样定义的：在每个语句处可以静态地知道哪些异常可能出现（即静态时就可以知道哪些异常不会出现），也可以知道哪些异常可以找到适当的处理程序，或哪些异常可以被传递到一个过程的调用点上去。如果出现异常而不能找到它的处理程序，则程序是错误的。

10.2 处理程序

语法：

〈处理程序〉::= (1)

ON{〈异常处理选择对象〉}*

[ELSE〈动作语句表〉] END (1.1)

〈异常处理选择对象〉::= (2)

(〈异常表〉):〈动作语句表〉 (2.1)

语义： 如果在一个语句中出现了异常，该语句附有一个处理程序，且在异常处理选择对象的异常表中提到了该异常的名字，则进入该异常处理选择对象中的动作语句表。如果在一个语句中出现了异常，该语句附有一个处理程序，在任何直接包含该处理程序的异常表中都未指明该异常的名字，但指明了ELSE，则进入ELSE后面的动作语句表。

如果该处理程序附着于一个动作，则当到达异常处理选择对象的动作语句表的末尾时，控制将转到该处理程序所在的动作语句之后的那个动作语句上去。

如果该处理程序附着于一个过程定义，则当到达动作语句表的末尾时，控制将返回到调用点。如果该处理程序附着于一个进程定义，则当到达异常处理选择对象的动作语句表的末尾时，正在执行的进程将终止。

静态条件： 所有异常表出现的所有名字都必须是不同的。

动态条件： 如果在进入动作语句表时，存储要求不能被满足，则将出现SPACEFAIL异常。

例子：

```
10.43  ON
       (no-space):CAUSE overflow;
       END                                    (1.1)
```

10.3 处理程序的识别

如果在动作A处，或在数据语句或区域D处出现了异常E，则该异常可以被适当的处理程序所处理，即处理程序中的动作语句表将被执行，或该异常将被传递到一个过程的调用点上去，如果这两者都办不到，则程序是错误的。

对于任何动作A，或数据语句或区域D，对其中所给出的异常E是否可以找到一个适当的处理程序，或者该异常是否能被传递到调用点去，要能够静态地确定。

对于A或D，处理E的适当处理程序可按如下方法确定：

1. 如果有处理程序附着于A或D，且在一个异常表中提到了E，或指明了ELSE，则它就是E的适当的处理程序。

2. 否则，如果A或D直接包含在一个加括号的动作中，则适当的处理程序（如果有的话）就是加括号动作的相对于E的适当的处理程序。
3. 否则，如果A或D位于一个过程定义的范围中，则：
   - 如果在过程定义之后指明了一个处理程序，该处理程序在异常表中指明了E，或指明了ELSE，则该处理程序就是适当的处理程序；
   - 如果在过程定义的异常表中提到了E，则E在调用点被引发；
   - 否则，不存在处理程序。
4. 否则，如果A或D位于一个（可能是虚拟的）进程定义的范围中，则：
   - 如果在进程定义之后指明处理程序，且该处理程序在异常表中指明了E，或ELSE，则该处理程序就是适当的处理程序。
   - 否则，不存在处理程序；
5. 否则，如果A是处理程序动作语句表中的一个动作，则考察该处理程序所附着的那个动作 A′ 或定义 D′，并且想象A′或D′是不带处理程序的，在此假设下，相对于A′或D′，处理E的适当的处理程序就是相对于A的适当的处理程序。

当一个异常被引发时，如果由于控制转向适当处理程序而导致从分程序退出时，则在从分程序退出时局部存储将被释放。

11.0 实现任选

11.1 实现定义的内部子程序

语法：

〈内部子程序调用〉::= (1)
〈内部子程序名字〉
(〔〈内部子程序参数表〉〕) (1.1)
〈内部子程序参数表〉::= (2)
〈内部子程序参数〉
{，〈内部子程序参数〉}* (2.1)
〈内部子程序参数〉::= (3)
〈值〉 (3.1)
|〈地点〉 (3.2)
|〈非保留名字〉 (3.3)

语义： 除了由语言定义的内部子程序外，实现还可提供一组由实现定义的内部子程序。

一个值、一个地点或任何一个非保留名字的由程序定义的名字，都可作为参数传递。内部子程序调用可以送返一个值或一个地点。其参数传递机制由实现定义。

内部子程序可以是普遍适用的，即它的类（若它为值内部子程序调用），或它的模式（若它为地点内部子程序调用）可以不仅依赖于内部子程序的名字，而且还依赖于被传递的实在参数的静态性质和调用的静态上下文。

静态性质： 内部子程序名字是由实现定义的名字，它被看作是在标准序部模块中定义的(见7.8节)。它可以附有由实现定义的异常名字的集合。当且仅当实现指明了：在选定参数的静态性质和调用的静态上下文后，内部子程序调用提供一个值（地点），此时称此内部子程序调用为值（地点）内部子程序调用。

11.2 实现定义的整数模式

除了由INT定义的整数模式外，实现还可定义其它的整数模式，例如短整数、长整数、无符号整数。这些整数模式必须由实现定义的整数模式名字标志。这些名字被看作是新模式名字，并类似于INT。它们的值区段是由实现定义的。这些整数模式可以被定义为适当类的根模式。

11.3 实现定义的寄存器名字

实现可以定义一组预定义的寄存器名字（见3.7和7.8节）。

11.4 实现定义的进程名字和异常名字

实现可以定义一组由实现定义的进程名字，即这些进程名字的定义不是用CHILL说明的。这个定义被认为是位于标准序部模块的范围之中。以它为名字的各进程可以被开动，标志这些进程的样品值可以被处理。

对于每个进程名字或每组进程名字，实现均可定义一组异常名字。当开动此进程时，这些异常可以被引发(见6.14节)。

11.5 实现定义的处理程序

实现可以将实现定义的处理程序附着于假想的最外层进程定义上(见7.8节)。由实现定义的处理程序中的异

常名字和动作可以指明任何合法的CHILL异常名字或动作。注意，只有最外层进程(而不是任何内层进程)引发的异常才能进入这类处理程序的处理选择对象。

11.6 语法任选

在某些地方，CHILL允许对同一个语义有多于一个的语法描述。至于在下列任选中选择那一个，在整个程序中是固定的。

赋值符号

赋值符号是:=或=

ARRAY

保留名字ARRAY要末必须采用，要末不准采用。

RETURNS

在带有结果说明的过程定义中，保留名字RETURNS要末必须采用，要末不准采用。

结构模式

结构模式要末采用嵌套结构表示法，要末采用层次编号表示法。

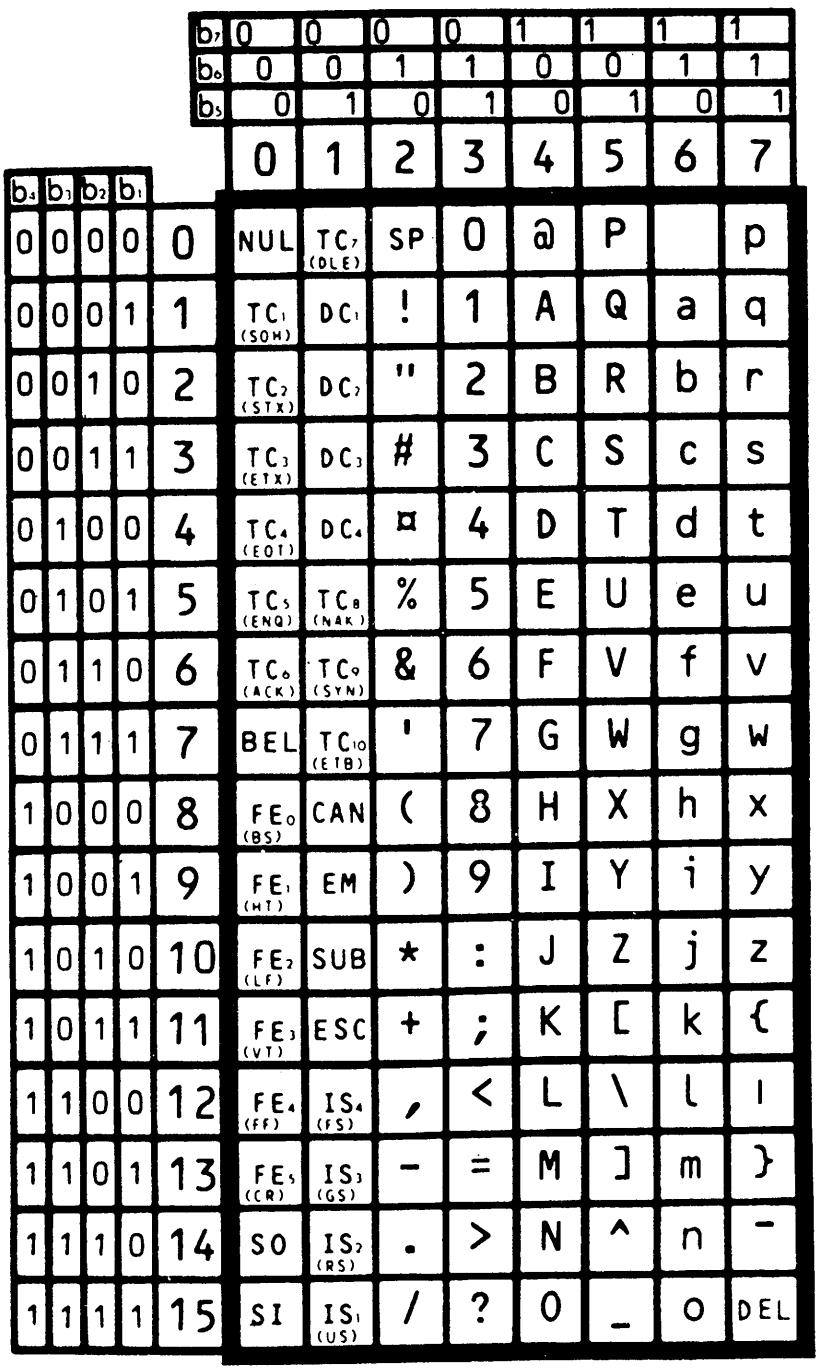
直接量和多元组括号

如果在表示字母表中有方括号可供使用，则可用括号〔和〕分别取代(:和:)。

附录A： CHILL程序的字符集

A．1 CCITT第5号字母表国际参考版

建议书V3(内部表示是从b7到b1各字位构成的二进制数，其中b1是最小有效位)。



b_4 b_3 b_2 b_1		b_7 0	0	0	0	1	1	1	1
		b_6 0	0	1	1	0	0	1	1
		b_5 0	1	0	1	0	1	0	1
		0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	TC_7 (DLE)	SP	0	@	P		p
0 0 0 1	1	TC_1 (SOH)	DC_1	!	1	A	Q	a	q
0 0 1 0	2	TC_2 (STX)	DC_2	"	2	B	R	b	r
0 0 1 1	3	TC_3 (ETX)	DC_3	#	3	C	S	c	s
0 1 0 0	4	TC_4 (EOT)	DC_4	¤	4	D	T	d	t
0 1 0 1	5	TC_5 (ENQ)	TC_8 (NAK)	%	5	E	U	e	u
0 1 1 0	6	TC_6 (ACK)	TC_9 (SYN)	&	6	F	V	f	v
0 1 1 1	7	BEL	TC_{10} (ETB)	'	7	G	W	g	w
1 0 0 0	8	FE_0 (BS)	CAN	(	8	H	X	h	x
1 0 0 1	9	FE_1 (HT)	EM	)	9	I	Y	i	y
1 0 1 0	10	FE_2 (LF)	SUB	*	:	J	Z	j	z
1 0 1 1	11	FE_3 (VT)	ESC	+	;	K	[	k	{
1 1 0 0	12	FE_4 (FF)	IS_4 (FS)	,	<	L	\	l	|
1 1 0 1	13	FE_5 (CR)	IS_3 (GS)	-	=	M	]	m	}
1 1 1 0	14	SO	IS_2 (RS)	.	>	N	^	n	~
1 1 1 1	15	SI	IS_1 (US)	/	?	O	_	o	DEL

A．2　表示CHILL程序的最小字符集

本文件使用CCITT第5号字母表基本码的下列子集表示CHILL程序。

b_7				0	0	0	0	1	1	1	1
b_6				0	0	1	1	0	0	1	1
b_5				0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1	0	1	2	3	4	5	6	7
0	0	0	0	0			SP	0		P		
0	0	0	1	1				1	A	Q		
0	0	1	0	2				2	B	R		
0	0	1	1	3				3	C	S		
0	1	0	0	4				4	D	T		
0	1	0	1	5				5	E	U		
0	1	1	0	6				6	F	V		
0	1	1	1	7			'	7	G	W		
1	0	0	0	8			(	8	H	X		
1	0	0	1	9			)	9	I	Y		
1	0	1	0	10			*	:	J	Z		
1	0	1	1	11			+	;	K			
1	1	0	0	12			,	<	L			
1	1	0	1	13			-	=	M			
1	1	1	0	14			.	>	N			
1	1	1	1	15			/	?	O	_		

附录B　专用符号

	名　　字	用　　途
;	分　号	语句结束符等
,	逗　号	各类语言结构的分隔符
(	左圆括号	各类语言结构的开括号
)	右圆括号	各类语言结构的闭括号
[	左方括号	多元组的开括号
]	右方括号	多元组的闭括号
(:	左多元组括号	多元组的开括号
:)	右多元组括号	多元组的闭括号
:	冒　号	标号指示符，区段指示符
.	圆　点	场选择符号
:=	赋值符号	赋值，赋初值
<	小　于	关系运算符
<=	小于等于	关系运算符
=	等　于	关系运算符，赋值，赋初值
/=	不　等	关系运算符
>=	大于等于	关系运算符
>	大　于	关系运算符
+	加	加运算符
−	减	减运算符
*	星　号	乘运算符，未定义值，未命名值，无关紧要符号
/	斜　线	除运算符
//	双斜线	连接运算符
→	箭　头	关联和非关联化
〈〉	方　块	命令子句的开始或结尾
/*	注释开始	注释的开括号
*/	注释结束	注释的闭括号
′	撇　号	各类直接量的开始或结束符号
″	双撇号	字符或字符串直接量内部的撇号
——	下划线	名词或直接量内部的空格符

附录C： CHILL专用名字

C.1 保留名字

```
ALL
ARRAY
ASSERT

BASED
BEGIN
BUFFER
BY

CALL
CASE
CAUSE
CONTINUE

DCL
DELAY
DO
DOWN

ELSE
ELSIF
END
ENTRY
ESAC
EVENT
EVER
EXCEPTIONS
EXIT

FI
FOR
FORBID

GENERAL
GOTO
GRANT

IF
IN
INIT
INLINE
INOUT

LOC

MODULE

NEWMODE
NOPACK

OD
OF
ON
OUT

PACK
PERVASIVE
POS
POWERSET
PRIORITY
PROC
PROCESS

RANGE
READ
RECEIVE
RECURSIVE
REF
REGION
RESULT
RETURN
RETURNS
ROW

SEIZE
SEND
SET
SIGNAL
SIMPLE
START
STATIC
STEP
STOP
STRUCT
SYN
SYNMODE

THEN
TO

UP

WHILE
WITH
```

C.2 预定义名字

ABS	FALSE	NOT	SIZE
ADDR		NULL	SUCC
AND		NUM	
	GETSTACK		
			THIS
BIN		OR	TRUE
BIT	INSTANCE		
BOOL	INT		
		PRED	UPPER
		PTR	
CARD	MAX		
CHAR	MIN		XOR
	MOD	REM	

C.3 CHILL异常名字

ASSERTFAIL
DELAYFAIL
EMPTY
EXTINCT
MODEFAIL
OVERFLOW
RANGEFAIL
RECURSEFAIL
SPACEFAIL
TAGFAIL

C.4 CHILL 命令

FREE

附录 D: 程序例子

1. 整数操作

```
integer_operations:
MODULE
   add:
   PROC (i,j INT)(INT) EXCEPTIONS (OVERFLOW);
      RESULT i+j;
   END add;
   mult:
   PROC (i,j INT)(INT) EXCEPTIONS (OVERFLOW);
      RESULT i*j;
   END mult;
   GRANT add, mult;
   SYNMODE operand_mode=INT;
   GRANT operand_mode;
   SYN neutral_for_add=0,
      neutral_for_mult=1;
   GRANT neutral_for_add,
         neutral_for_mult;
END integer_operations;
```

2. 分数上的同类操作

```
fraction_operations:
MODULE
   NEWMODE fraction=STRUCT (num,denum INT);
   add:
   PROC (f1,f2 fraction)(fraction) EXCEPTIONS (OVERFLOW);
      RETURN [f1.num*f2.denum+f2.num*f1.denum,
         f1.denum*f2.denum];
   END add;
   mult:
   PROC (f1,f2 fraction)(fraction) EXCEPTIONS (OVERFLOW);
      RETURN [f1.num*f2.num,f2.denum*f1.denum];
   END mult;

   GRANT add, mult;
   SYNMODE operand_mode=fraction;
   GRANT operand_mode;
   SYN neutral_for_add fraction=[0,1],
      neutral_for_mult fraction=[1,1];
   GRANT neutral_for_add,
        neutral_for_mult;

END fraction_operations;
```

3　复数上的同类操作

```
complex_operations
MODULE
    NEWMODE complex=STRUCT (re,im INT);
    add:
    PROC (c1,c2 complex)(complex) EXCEPTIONS (OVERFLOW);
       RETURN [c1.re+c2.re,c1.im+c2.im];
    END add;
    mult:
    PROC (c1,c2 complex)(complex) EXCEPTIONS (OVERFLOW);
       RETURN [c1.re*c2.re-c1.im*c2.im ,
               c1.re*c2.im+c1.im*c2.re];
    END mult;

    GRANT add, mult
    SYNMODE operand_mode=complex;
    GRANT operand_mode;
    SYN neutral_for_add=complex [0,0],
       neutral_for_mult=complex [1,0];
    GRANT neutral_for_add,
         neutral_for_mult;

END complex_operations;
```

4. 广阶算术

```
general_order_arithmetic: /*from collected algorithms from CACM no.93*/
MODULE
   op:
   PROC (a INOUT, b,c,order INT) EXCEPTIONS (wrong_input) RECURSIVE;
      DCL d INT;
      ASSERT b>0 AND c>0 AND order>0
         ON (ASSERTFAIL):
            CAUSE wrong_input;
         END;
      CASE order OF
         (1): a :=b+c;
            RETURN;
         (2): d :=0;
         (ELSE): d :=1;
      ESAC;
      DO FOR i :=1 TO c;
         op (a,b,d,order-1);
         d :=a;
      OD;
      RETURN;
   END op;

   GRANT op;

END general_order_arithmetic;
```

5. 按位相加並检验其结果

```
add_bit_by_bit:
MODULE
        adder:
        PROC (a STRUCT(a2,a1 BOOL) IN, b STRUCT(b2,b1 BOOL) IN)
            RETURNS(STRUCT(c4,c2,c1 BOOL));

        DCL c STRUCT (c4,c2,c1 BOOL);
        DCL k2,x,w,t,s,r BOOL;
        DO WITH a,b,c;
            k2 :=a1 AND b1;
            c1 :=NOT k2 AND (a1 OR b1);
            x :=e2 AND b2 AND k2;
            w :=a2 OR b2 OR k2;
            t :=b2 AND k2;
            s :=a2 AND k2;
            r :=a2 AND b2;
            c4 :=r OR s OR t;
            c2 :=x OR (w AND NOT c4);
        OD;
        RETURN c;
    END adder;
    GRANT adder;
END add_bit_by_bit;

exhaustive_checker:
MODULE
    SEIZE adder;
    DCL a STRUCT (a2,a1 BOOL),
        b STRUCT (b2,b1 BOOL),
    SYNMODE res=ARRAY (1:16) STRUCT (c4,c2,c1 BOOL);
    DCL r INT, results res;
    DO WITH a,b;
        r :=0;
        DO FOR a2 IN BOOL;
            DO FOR a1 IN BOOL;
                DO FOR b2 IN BOOL;
                    DO FOR b1 IN BOOL;
                        r+ :=1;
                        results (r) :=adder (a,b);
                    OD;
                OD;
            OD;
        OD;
        ASSERT result=res [[FALSE,FALSE,FALSE],[FALSE,FALSE,TRUE],
                           [FALSE,FALSE,TRUE],[FALSE,TRUE,TRUE],
                           [FALSE,FALSE,TRUE],[FALSE,TRUE,FALSE],
                           [FALSE,TRUE,TRUE],[TRUE,FALSE,FALSE],
                           [FALSE,TRUE,FALSE],[FALSE,TRUE,FALSE],
                           [TRUE,FALSE,FALSE],[TRUE,FALSE,TRUE.],
                           [FALSE,TRUE,TRUE],[TRUE,FALSE,FALSE],
                           [TRUE,FALSE,TRUE],[TRUE,TRUE,FALSE]];
END exhaustive_checker;
```

6. 日历游戏

```
playing_with_dates:
MODULE /* from collected algorithms from CACM no. 199 */
   SYNMODE month=SET(jan,feb,mar,apr,may,jun,
                     jul,aug,sep,oct,nov,dec);
   NEWMODE date=STRUCT (day INT (1:31), mo month, year INT);

   gregorian-date:
   PROC (julian_day_number INT)(date);
      DCL j INT :=julian_day_number,
          d,m,y INT;
      j-:=1_721_119;
      y :=(4 * j - 1) / 146_097;
      j :=4 * j - 1 - 146_097 * y;
      d :=j / 4;
      j :=(4 * d + 3) / 1_461;
      d :=4 * d + 3 - 1_461 * j;
      d :=(d + 4) / 4;
      m :=(5 * d - 3) / 153;
      d :=5 * d - 3 - 153 * m;
      d :=(d + 5) / 5;
      y :=100 * y + j;
      IF m<100 THEN m+:=3;
               ELSE m-:=9;
                    y+:=1;
      FI;
      RETURN [d,month (m+1), y];
   END gregorian_date;

   julian_day_number
   PROC (d date)(INT);
      DCL c,y,m INT;
      DO WITH d;
         m :=NUM (mo)+1;
         IF m>2 THEN m-:=3;
                ELSE m+:=9;
                     year-:=1;
         FI;
         c :=year/100;
         y :=year-100*c;
         RETURN (146_097*c)/4+(1_461*y)/4
            +(153+m+c)/5+day+1_721_119;
      OD;
   END julian_day_number;
   GRANT gregorian_date, julian_day_number;
END playing_with_dates

test:
MODULE
  SEIZE gregorian_date, julian_day_number;
  ASSERT julian_day_number ([10,dec,1979])=julian_day_number(
       gregorian_date(julian_day_number([10,dec,1979])));
END test;
```

7. 罗马数字

```
Roman:
MODULE
    SEIZE n,rn;
    convert:
    PROC () EXCEPTIONS (string_too_small);
        DCL r INT :=0;
        DO WHILE n>=1_000;
            rn(r):='M';
            r+:=1;
            n-:=1_000;
        OD;
        IF n>500 THEN rn(r):='D';
                      n-:=500;
                      r+:=1;
        FI;
        IF n>=100 THEN rn(r):='C';
                       n-:=100;
                       r+:=1;
        FI;
        IF n>=50 THEN rn(r):='L';
                      n-:=50;
                      r+:=1;
        FI;
        IF n>=10 THEN rn(r):='X';
                      n-:=10;
                      r+:=1;
        FI;
        DO WHILE n>=1;
            rn(r):='I';
            r+:=1;
            n-:=1;
        OD;
        RETURN;
    END ON (RANGEFAIL): DO FOR i :=0 TO UPPER (rn);
                          rn(i) :='.';
                        OD;
                        CAUSE string_too_small;
    END convert;
END Roman;
test:
MODULE
    SEIZE convert;
    DCL n INT INIT :=1979;
    DCL rn CHAR (20) INIT :=(20)' ';
    GRANT n,rn;

    convert ();

    ASSERT rn='MDCCCCLXXVIIII'//(6)' ';

END test;
```

8. 任意长字符串的字母计数

```
letter_count:
MODULE
   SEIZE max;
   DCL letter POWERSET CHAR INIT :=['A' : 'Z'];
   count:
   PROC (input ROW CHAR (max) IN, output ARRAY('A':'Z') INT OUT);
      DO FOR i :=0 TO UPPER (input ->);
         IF input -> (i) IN letter
            THEN
               output (input -> (i))+:=1;
         FI;
      OD;
   END count;
   GRANT count;
END letter-count;
test:
MODULE
   DCL c CHAR (10) INIT :='A-B<ZAA9K''';
   DCL output ARRAY ('A' : 'Z') INT;
   SYN max=10_000;
   GRANT max;
   SEIZE count;
   count (-> c,output);
   ASSERT output=[('A') : 3,('B','K','Z') : 1, (ELSE) : 0];

END test;
```

9. 素数

```
prime:
MODULE
   SEIZE max;
   NEWMODE number_list =POWERSET INT(2:max);
   SYN  empty = number_list [];
   DCL sieve number_list INIT := [2:max];
      primes number_list INIT :=empty;
   GRANT primes;
   DO WHILE sieve/=empty;
      primes OR :=[MIN (sieve)];
      DO FOR j :=MIN (sieve) BY MIN (sieve) TO max;
         sieve-:=[j];
      OD;
   OD;
END prime;
```

10. 用两种不同方法实现栈，对用户透明

```
1   stacks_1:
2   MODULE
3       SEIZE element
4       SYN max=10_000,min=1;
5       DCL stack ARRAY (min : max) element,
6          stackindex INT INIT :=min;
7       push:
8       PROC (e element) EXCEPTIONS (overflow);
9           IF stackindex=max
10              THEN CAUSE overflow;
11          FI;
12          stackindex+:=1;
13          stack (stackindex) :=e;
14          RETURN;
15      END push;
16      pop:
17      PROC () EXCEPTIONS (underflow);
18          IF stackindex=min
19              THEN CAUSE underflow;
20          FI;
21          stackindex-:=1;
22          RETURN;
23      END pop;
24
25      elem:
26      PROC (i INT)(element LOC) EXCEPTIONS (bounds);
27          IF i<min OR i>max
28              THEN CAUSE bounds;
29          FI;
30          RETURN stack (i);
31      END elem;
32
33      GRANT push,pop,elem;
34  END stacks-1;
```

```
35  stacks_2:
36  MODULE
37     SEIZE element;
38     NEWMODE cell=STRUCT (pred,succ REF cell,
39                          info element);
40     DCL p,last,first REF cell INIT :=NULL;
41     push:
42     PROC (e element) EXCEPTIONS (overflow);
43        p :=allocate (cell) ON
44                            (nospace) : CAUSE overflow;
45                            END;
46        IF last=NULL
47           THEN first,last :=p;
48           ELSE last ->. succ :=p;
49              p ->. pred :=last;
50              last :=p;
51        FI;
52        last ->. info :=e;
53        RETURN;
54     END push;
55     pop:
56     PROC () EXCEPTIONS (underflow);
57        IF last=NULL;
58           THEN CAUSE underflow;
59        FI;
60        last :=last ->. pred; IF last = NULL THEN first := NULL FI;
61        last ->. succ :=NULL;
62        RETURN;
63     END pop;
64     elem:
65     PROC (i INT) (element LOC) EXCEPTIONS (bounds);
66        IF first=NULL
67           THEN CAUSE bounds;
68        FI;
69        p :=first;
70        DO FOR j=2 TO i;
71           IF p ->. succ=NULL
72              THEN CAUSE bounds;
73           FI;
74           p :=p ->. succ;
75        OD;
76        RETURN p ->. info;
77     END elem;
78
79     GRANT push,pop,elem;
80
81  END stacks_2;
```

11. 下棋片断

```
NEWMODE piece=STRUCT(color SET(white,black),
                     kind SET(pawn,rook,knight,bishop,queen,king));
NEWMODE column=SET (a,b,c,d,e,f,g,h);
NEWMODE line=INT (1 : 8);
NEWMODE square=STRUCT (status SET (occupied,free),
                       CASE status OF
                       (occupied) : p piece,
                       (free) :
                       ESAC);
NEWMODE board=ARRAY (line) ARRAY (column) square;
NEWMODE move=STRUCT (lin_1,lin_2 line,
                     col_1,col_2 column);

initialise:
PROC (bd board INOUT);
   bd :=[(1) : [(a,h):[.status: occupied, .p : [white,rook]],
        (b,g):[.status: occupied, .p : [white,knight]],
        (c,f):[.status: occupied, .p : [white,bishop]],
          (d):[.status: occupied, .p : [white,queen]],
          (e):[.status: occupied, .p : [white,king]]],
  (2):[(ELSE) : [.status: occupied, .p : [white,pawn]]],
 (3:6):[(ELSE) : [.status: free]],
   (7):[(ELSE) : [.status: occupied, .p : [black,pawn]]],
   (8):[(a,h) : [.status: occupied, .p : [black,rook]],
     (b,g):[.status: occupied, .p : [black,knight]],
           (e,f) : [.status: occupied, .p : [black,bishop]],
             (d) : [.status: occupied, .p : [black,king]],
             (e) : [.status: occupied, .p : [black,queen]]]];
   RETURN;

END initialise;
```

```
register_move:
PROC (b board LOC,m move) EXCEPTIONS (illegal);
   DCL starting square LOC :=b (m.lin_1)(m.col_2),
      arriving square LOC :=b (m.lin_1)(m.col_2);

   DO WITH m;
      IF starting.status=free
         OR (lin_2<1 OR lin_2>8 OR col_2<a OR col_2>h)
         OR (arriving.status/=free AND arriving.p.kind=king)
         THEN
            CAUSE illegal;
      FI;
      CASE starting.p.kind, starting.p.color OF

       (pawn),(white):
          IF col_1 = col_2 AND (arriving.status/=free
             OR NOT (lin_2=lin_1+1 OR lin_2=lin_1+2 AND lin_2=2))
             OR (col_2=PRED(col_1) OR col_2=SUCC(col_1))
             AND arriving.status=free OR arriving.p.color=white
          THEN
              CAUSE illegal; /*capturing en passant not implemented*/
          FI;
       (pawn),(black):
          IF col_1=col_2 AND (arriving.status/=free
             OR NOT (lin_2=lin_1-1 OR lin_2=lin_1-2 AND lin_1=7))
             OR (col_2=PRED(col_1) OR col_2=SUCC(col_1))
             AND arriving.status=free OR arriving.p.color=black
             THEN
                CAUSE illegal; /* same remark */
          FI;
       (rook),(*):
          IF NOT ok_rook (b,m)
             THEN
                CAUSE illegal;
          FI;
       (bishop),(*):
          IF NOT ok_bishop (b,m)
             THEN
                 CAUSE illegal;
          FI;
       (queen),(*):
          IF NOT ok_rook (b,m)
             THEN
                IF NOT ok_bishop (b,m)
                   THEN
                      CAUSE illegal
                FI;
         FI;
       (knight,*):
         IF ABS(ABS(NUM(col_2)-NUM(col_1))
            -ABS(lin_2-lin_1)) /= 1
            OR ABS(NUM(col_2)-NUM((col_1))
                +ABS(lin_2-lin_1) =/ 3
            OR arriving.status/=free AND
```

```
86              arriving.p.color=starting.p.color
87          THEN CAUSE illegal;
88          FI;
89            (king),(*):
90             IF ABS(NUM(col_2)-NUM(col_1)) > 1
91                OR ABS(lin_2-lin_1) > 1
92                OR lin_2=lin_1 AND col_2=col_1
93                OR arriving.status/=free AND
94                   arriving.p.color=starting.p.color
95             THEN CAUSE illegal;
96             FI; /*checking king moving to check not implemented*/
97        ESAC;
98     OD;
99     arriving :=starting;
100    RETURN;
101 END register_move;
102 ok_rook:
103 PROC (b board,m move)(BOOL);
104    DO WITH m;
105       IF NOT (col_2=col_1 OR lin_1=lin_2)
106          OR arriving.status/=free AND
107          arriving.p.color=starting.p.color
108       THEN RETURN FALSE;
109       FI;
110       IF col_1=col_2
111          THEN IF lin_1<lin_2
112                  THEN DO FOR l := lin_1+1 TO lin_2-1;
113                          IF board (l)(col_1).status/=free
114                             THEN RETURN FALSE;
115                          FI;
116                       OD;
117                  ELSE DO FOR l := lin_1-1 DOWN TO lin_2+1;
118                          IF board (l)(col_1).status/=free
119                             THEN RETURN FALSE;
120                          FI;
121                       OD;
122               FI;
123          ELSE IF col_1<col_2
124                  THEN DO FOR c := SUCC(col_1) TO PRED(col_2);
125                          IF board (lin_1)(c).status/=free
126                             THEN RETURN FALSE;
127                          FI;
128                       OD;
129                  ELSE DO FOR c := SUCC(col_2) DOWN TO PRED(col_1);
130                          IF board (lin_1)(c).status/=free
131                             THEN RETURN FALSE;
132                          FI;
133                       OD;
134               FI;
135       FI;
136       RETURN TRUE;
137    OD;
138 END ok_rook;
```

```
139  ok_bishop:
140  PROC (b board,m move)(BOOL);
141     DO WITH m;
142        CASE lin_2>lin_1,col_2>col_1 OF
143           (TRUE),(TRUE): c := col_1
144                       DO FOR  l := lin_1+1 TO lin_2-1;
145                          c := SUCC(c);
146                          IF board (l)(c).status/=free
147                             THEN RETURN FALSE;
148                          FI;
149                       OD;
150                       IF SUCC(c)/=col_2
151                          THEN RETURN FALSE;
152                       FI;
153           (TRUE),(FALSE): c := col_1
154                        DO FOR l := lin_1+1 TO lin_2-1;
155                           c := PRED(c);
156                           IF board (l)(c).status/=free
157                              THE RETURN FALSE;
158                           FI;
159                        OD;
160                        IF PRED(c)/=col_2
161                           THEN RETURN FALSE;
162                        FI;
163           (FALSE),(TRUE): c := col_1
164                       DO FOR l := lin_1-1 DOWN TO lin_2+1;
165                          c := SUCC(c)
166                          IF board (l)(c).status/=free
167                             THEN RETURN FALSE;
168                          FI;
169                       OD;
170                       IF SUCC(c)/=col_2
171                          THEN RETURN FALSE;
172                       FI:
173           (FALSE),(FALSE): c := col_1;
174                        DO FOR l := lin_1-1 DOWN TO lin_2+1;
175                           c := PRED(c);
176                           IF board (l)(c).status/=free
177                              THEN RETURN FALSE;
178                           FI;
179                        OD;
180                        IF PRED (c)/=col_2
181                           THEN RETURN FALSE;
182                        FI;
183        ESAC;
184        RETURN arriving.status=free OR
184              arriving.p.color/=starting.p.color;
186     OD;
187  END ok_bishop;
```

12. 建立和处理一个循环链表

```
1   CIRCULAR_LIST:
2   MODULE

3       HANDLE_LIST:
4       MODULE
5           GRANT INSERT, REMOVE, NODE;
6           NEWMODE NODE=STRUCT(PRED, SUC REF NODE, VALUE INT);
7           DCL POOL ARRAY(1:1000)NODE;
8           DCL HEAD NODE:=(: NULL,NULL,0 :);
9           INSERT:
            PROC(NEW NODE);
10              /* INSERT ACTIONS */
11          END INSERT;

12          REMOVE:
            PROC();
13              /* REMOVE ACTIONS */
14          END REMOVE;

15          INITIALIZE_LIST:
16          BEGIN
17              DCL LAST REF NODE:= ->HEAD;
18              DO FOR NEW IN POOL;
19                  NEW.PRED := LAST;
20                  LAST->.SUC:= ->NEW;
21                  LAST:= ->NEW;
22                  NEW.VALUE:=0;
23              OD;
24              HEAD.PRED:=LAST;
25              LAST->.SUC:= ->HEAD;
26          END INITIALIZE_LIST

27      END HANDLE_LIST;

28      DCL NODE_A NODE:=(: NULL,NULL,536 :);
29      REMOVE();
30      REMOVE();
31      INSERT(NODE_A);
32  END CIRCULAR_LIST;
```

13. 用于管理对资源竞争访问的一个区域

```
1   ALLOCATE_RESOURCES:
2   REGION
3       GRANT ALLOCATE, DEALLOCATE;
4       NEWMODE RESOURCE_SET = INT(0:9);
5       DCL ALLOCATED ARRAY(RESOURCE_SET)BOOL :=
                    (: (RESOURCE_SET): FALSE :);
6       DCL RESOURCE_FREED EVENT;

7       ALLOCATE:
8       PROC()(INT);
9           DO FOR EVER;
10              DO FOR I IN RESOURCE_SET;
11                  IF NOT ALLOCATED(I)
12                  THEN
13                      ALLOCATED(I) := TRUE;
14                      RETURN I;
15                  FI;
16              OD;
17              DELAY RESOURCE_FREED;
18          OD;
19      END ALLOCATE;

20      DEALLOCATE:
21      PROC(I INT);
22          ALLOCATE(I) := FALSE;
23          CONTINUE RESOURCE_FREED;
24      END DEALLOCATE;

25  END ALLOCATE_RESOURCES;
```

14. 对一个电话总机的排队呼叫

```
1  SWITCHBOARD:
2  MODULE
3   /* This example illustrates a switchboard which queues incoming calls
4      and feeds them to the operator at an even rate. Every time the
5      operator is ready one and only one call is let through. This is
6      handled by a call distributor which lets calla through at fixed
7      intervals. If the operator is not ready or there are other calls
8      waiting, a new call must queue up to wait for its turn. */

9      DCL OPERATOR_IS_READY,
10        SWITCH_IS_CLOSED EVENT;

11     CALL_DISTRIBUTOR:
12     PROCESS();
13        DO FOR EVER;
14           WAIT(10 /*seconds*/);
15           CONTINUE OPERATOR_IS_READY;
16        OD;
17     END CALL_DISTRIBUTOR;

18     CALL:
19     PROCESS();
20        DELAY CASE
21        (OPERATOR_IS_READY): /* some actions */
22        (SWITCH_IS_CLOSED): DO FOR I IN INT(1:100);
23                               CONTINUE OPERATOR_IS_READY;
24                               /*empty the queue*/
25                            OD;
26        ESAC;
27     END CALL;

28     OPERATOR:
29     PROCESS();
30        DO FOR_EVER;
31           IF TIME = 1700
32              THEN
33                 CONTINUE SWITCH_IS_CLOSED;
34           FI;
35        OD;
36     END OPERATOR;

37     START CALL_DISTRIBUTOR():
38     START OPERATOR();
39     DO FOR I INT(1:100);
40        START CALL();
41     OD;
42  END SWITCHBOARD;
```

15. 对一组资源的分配和回收

```
1   <> FREE (STEP);
2   COUNTER MANAGER:
3   MODULE
4   /* To illustrate the use of signals and the receive case, (buffers
5     might have been instead) we will look at an example where an
6     ALLOCATOR manages a set of resources, in this case a set of
7     COUNTERs. The module is part of a larger system where there are
8     USERs, that can request the services of the COUNTER_MANAGER. The
9     module is made to consistof two process definitions, one for the
10    ALLOCATION and one for the COUNTERS. INITIATE and TERMINATE
11    are internal signals sent from the ALLOCATOR
12    to the COUNTERs. All the other signals are external, being sent
13    from or to the USERs. */

14     SEIZE /* external signals */
15        ACQUIRE, RELEASE, CONGESTED,STEP,READOUT;
16     SIGNAL INITIATE = (INSTANCE),
17           TERMINATE;

18     ALLOCATOR:
19     PROCESS();
20        NEWMODE NO_OF_COUNTERS = INT(1:100);
21     DCL COUNTERS ARRAY (NO_OF_COUNTERS)
22              STRUCT (COUNTER INSTANCE,
23                      STATUS SET (BUSY,IDLE));
24     DO FOR EACH IN COUNTERS;
25     EACH:= (: START COUNTER(), IDLE :);
26     OD;

27     DO FOR EVER;
28     BEGIN
29        DCL USER INSTANCE;
30        AWAIT_SIGNALS:
31        RECEIVE CASE SET USER;
32        (ACQUIRE):
33           DO FOR EACH IN COUNTERS;
34              DO WITH EACH;
35                 IF STATUS = IDLE
36                 THEN
37                    STATUS:=BUSY;
38                    SEND INITIATE (USER) TO COUNTER;
39                    EXIT AWAIT_SIGNALS;
40                 FI;
41              OD;
42           OD;
43           SEND CONGESTED TO USER;
44        (RELEASE IN THIS_COUNTER);
45           SEND TERMINATE TO THIS_COUNTER;
```

```
46              FIND_COUNTER:
47              DO FOR EACH IN COUNTERS;
48                 DO WITH EACH;
49                    IF THIS_COUNTER = COUNTER
50                    THEN
51                       STATUS:= IDLE;
52                       EXIT FIND_COUNTER;
53                    FI;
54                 OD;
55              OD FIND_COUNTER;
56           ESAC AWAIT_SIGNALS;
57        END;
58        OD;
59        END ALLOCATOR;

60        COUNTER:
61        PROCESS();
62           DO FOR EVER;
63           BEGIN
64              DCL USER INSTANCE;
65                 COUNT:= 0;
66              RECEIVE CASE
67                 (INITIATE IN RECEIVED_USER):
68                    SEND READY TO RECEIVED_USER;
69                    USER:= RECEIVED_USER;
70              ESAC;
71              WORK_LOOP:
72              DO FOR EVER;
73                 RECEIVE CASE
74                    (STEP): COUNT +:=1;
75                    (TERMINATE):
76                       SEND READOUT(COUNT) TO USER;
77                       EXIT WORK_LOOP;
78                 ESAC;
79              OD WORK_LOOP;
80           END;
81           OD;
82        END COUNTER;

83        START ALLOCATOR();
84     END COUNTER_MANAGER;
```

16. 利用缓冲区对一组资源进行分配和回收

```
<> FREE(STEP);
USER_WORLD:
MODULE
/* This example is the same as no.15 except that buffers are
   used for communication in stead of signals.
   The main difference is that processes are now identified
   by means of references to local message buffers rather than
   by instance values. There is one message buffer declared
   local to each process. There is one set of message types
   for each process definition. When started each process must
   identify its buffer address to the starting process.
   The USER_WORLD module sketches some of the environment in
   which the COUNTER_MANAGER is used. */

GRANT USER_BUFFERS,
      ALLOCATOR_MESSAGES, ALLOCATOR_BUFFERS,
      COUNTER_MESSAGES, COUNTER_BUFFERS;
NEWMODE
  USER_MESSAGES =
      STRUCT(TYPE SET(CONGESTED, READY,
                      READOUT, ALLOCATOR_ID),
             CASE TYPE OF
             (CONGESTED)   :
             (READY)       : COUNTER REF COUNTER_BUFFERS,
             (READOUT)     : COUNT INT,
             (ALLOCATOR_ID): ALLOCATOR REF ALLOCATOR_BUFFERS
             ESAC),
  USER_BUFFERS = BUFFER(1) USER_MESSAGES,
  ALLOCATOR_MESSAGES =
      STRUCT(TYPE SET(ACQUIRE, RELEASE, COUNTER_ID),
             CASE TYPE OF
             (ACQUIRE)   : USER REF USER_BUFFERS,
             (RELEASE,
              COUNTER_ID): COUNTER REF COUNTER_BUFFERS
             ESAC),
  ALLOCATOR_BUFFERS = BUFFER(1) ALLOCATOR_MESSAGES,
  COUNTER_MESSAGES =
      STRUCT(TYPE SET(INITIATE, STEP, TERMINATE),
             CASE TYPE OF
             (INITIATE) : USER REF USER_BUFFERS,
             (STEP,
              TERMINATE):
             ESAC,
  COUNTER_BUFFERS = BUFFER(1) COUNTER_MESSAGES;
DCL USER_BUFFER USER_BUFFERS,
    ALLOCATOR_BUF REF ALLOCATOR_BUFFERS,
    COUNTER_BUF REF COUNTER_BUFFERS;
START ALLOCATOR(->USER_BUFFER);
ALLOCATOR_BUF := (RECEIVE USER_BUFFER).ALLOCATOR;
END_USER_WORLD;
```

```
 51  COUNTER_MANAGER:
 52  MODULE
 53  SEIZE USER_BUFFERS,
 54        ALLOCATOR_MESSAGES, ALLOCATOR_BUFFERS,
 55        COUNTER_MESSAGES, COUNTER_BUFFERS;
 56
 57  ALLOCATOR:
 58  PROCESS(STARTER REF USER_BUFFERS);
 59     DCL ALLOCATOR_BUFFER ALLOCATOR_BUFFERS;
 60     NEWMODE NO_OF_COUNTERS = INT(1:10);
 61     DCL COUNTERS ARRAY(NO_OF_COUNTERS)
 62           STRUCT(COUNTER REF COUNTER_BUFFERS,
 63                  STATUS SET(BUSY, IDLE)),
 64        MESSAGE ALLOCATOR_MESSAGES;
 65     SEND STARTER->([ALLOCATOR_ID, ->ALLOCATOR_BUFFER]);
 66     DO FOR EACH IN COUNTERS;
 67        START COUNTER(->ALLOCATOR_BUFFER);
 68        EACH := [(RECEIVE ALLOCATOR_BUFFER).COUNTER, IDLE];
 69     OD;
 70     DO FOR EVER;
 71        BEGIN
 72        DCL USER REF USER_BUFFERS;
 73        MESSAGE := RECEIVE ALLOCATOR_BUFFER;
 74        HANDLE_MESSAGES:
 75        CASE MESSAGE.TYPE OF
 76        (ACQUIRE):
 77           USER := MESSAGE.USER;
 78           DO FOR EACH IN COUNTERS;
 79              DO WITH EACH;
 80                 IF STATUS= IDLE
 81                 THEN STATUS := BUSY;
 82                      SEND COUNTER->([INITIATE, USER]);
 83                      EXIT HANDLE_MESSAGES;
 84                 FI;
 85              OD;
 86           OD;
 87           SEND USER->([CONGESTED]);
 88        (RELEASE):
 89           SEND MESSAGE.COUNTER([TERMINATE]);
 90           FIND_COUNTER:
 91           DO FOR EACH IN COUNTERS;
 92              DO WITH EACH;
 93                 IF MESSAGE.COUNTER = COUNTER
 94                 THEN STATUS := IDLE;
 95                      EXIT FIND_COUNTER;
 96                 FI;
 97              OD;
 98           OD FIND_COUNTER;
 99        ESAC HANDLE_MESSAGES;
100        END;
101     OD;
102  END ALLOCATOR;
```

```
103 COUNTER:
104 PROCESS(STARTER REF ALLOCATOR_BUFFERS);
105    DCL COUNTER_BUFFER ALLOCATOR_BUFFERS;
106    SEND STARTER->([COUNTER_ID, ->COUNTER_BUFFER]);
107    DO FOR EVER;
108       BEGIN
109         DCL USER REF USER_BUFFERS,
110             COUNT INT := 0,
111             MESSAGE COUNTER_MESSAGES;
112         MESSAGE := RECEIVE COUNTER_BUFFER;
113         CASE MESSAGE.TYPE OF
114         (INITIATE): USER := MESSAGE.USER;
115                     SEND USER->([READY, ->COUNTER_BUFFER]);
116         ELSE /* some error action */
117         ESAC;
118         WORK_LOOP:
119         DO FOR EVER;
120            MESSAGE := RECEIVE COUNTER_BUFFER;
121            CASE MESSAGE.TYPE OF
122            (STEP)      : COUNT +:= 1;
123            (TERMINATE):SEND USER->([READOUT, COUNT]);
124                        EXIT WORK_LOOP;
125            ELSE /* some error action */
126            ESAC;
127         OD WORK_LOOP;
128    END;
129    OD;
130 END COUNTER;
131 END COUNTER_MANAGER;
```

17. 串扫描程序 1

```
1  string_scanner1: /* This program implements strings by means
2                      of packed arrays of characters.*/
3  MODULE
4  SYN
5    blanks ARRAY(0:9)CHAR PACK = [(*):' '], linelength = 132;
6  SYNMODE
7    stringptr = ROW ARRAY(lineindex)CHAR PACK,
8    lineindex = INT(0:linelength-1);
9
10 scanner:
11 PROC(string stringptr, scanstart lineindex INOUT,
12      scanstop lineindex, stopset POWERSET CHAR)
13     RETURNS(ARRAY(0:9)CHAR PACK);
14     DCL count INT:=0,
15         res ARRAY(0:9)CHAR PACK:=blanks;
16     DO
17       FOR c IN string->(scanstart:scanstop)
18       WHILE NOT (c IN stopset);
19         count+:=1;
20     OD;
21     IF count>0
22       THEN
23           IF count>10
24             THEN
25               count:=10;
26           FI;
27           res(0:count-1):=string->(scanstart:scanstart+count-1);
28     FI;
29     RESULT res;
30     IF scanstart+count < scanstop
31       THEN
32           scanstart:=scanstart+count+1;
33     FI;
34     END scanner;
35
36 GRANT
37     scanner;
38
39 END string_scanner;
```

18. 串扫描程序 2

```
string_scanner2: /* This example is the same as no.18 but it uses
                    character string in stead of packed arrays.*/
MODULE
  SYN
    blanks = (10)' ', linelength = 132;
  SYNMODE
    stringptr = ROW CHAR(linelength),
    lineindex = INT(0:linelength-1);

  scanner:
    PROC(string stringptr, scanstart lineindex INOUT,
        scanstop lineindex, stopset POWERSET CHAR)
      RETURNS (CHAR(10));
     DCL count INT:=0;
     DO FOR i := scanstart TO scanstop
       WHILE NOT (string->(i) IN stopset);
         count+:=1;
     OD;
     IF count>0
       THEN
         IF count>=10
           THEN
             RESULT string->(scanstart UP 10);
           ELSE
             RESULT string->(scanstart:scanstart+count-1)
                   //blanks(count:9);
         FI;
       ELSE
         RESULT blanks;
     FI;
     IF scanstart+count < scanstop
       THEN
         scanstart:=scanstart+count+1;
     FI;
   END scanner;

  GRANT
    scanner;

END string_scanner;
```

19. 从双重链表中除去一个项

```
1  QUEUE_REMOVAL:
2  MODULE
3     SEIZE INFO;
4     GRANT REMOVE;
5     REMOVE: PROC(P PTR) RETURNS(INFO) EXCEPTIONS(EMPTY);
6     /* This procedure removes the item referred to by
7        P from a queue and returns the information contents
8        of that queue element.*/
9          DCL 1 X BASED (P),
10              2 I INFO POS(0,8:31),
11              2 PREV PTR POS(1,0:15),
12              2 NEXT PTR POS(1,16:31);
13         DCL PREV, NEXT PTR;
14         PREV := X.PREV;
15         NEXT := X.NEXT;
16         X.PREV, X.NEXT := NULL;
17         RESULT X.INFO;
18         P := PREV;
19         X.NEXT := NEXT;
20         P := NEXT;
21         X.PREV := PREV;
22    END REMOVE;
23
24 END QUEUE_REMOVAL;
```

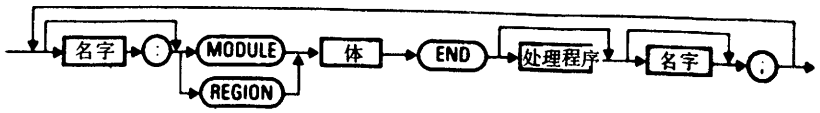
附录E　语法图

本附录中的图是描述CHILL语法的。

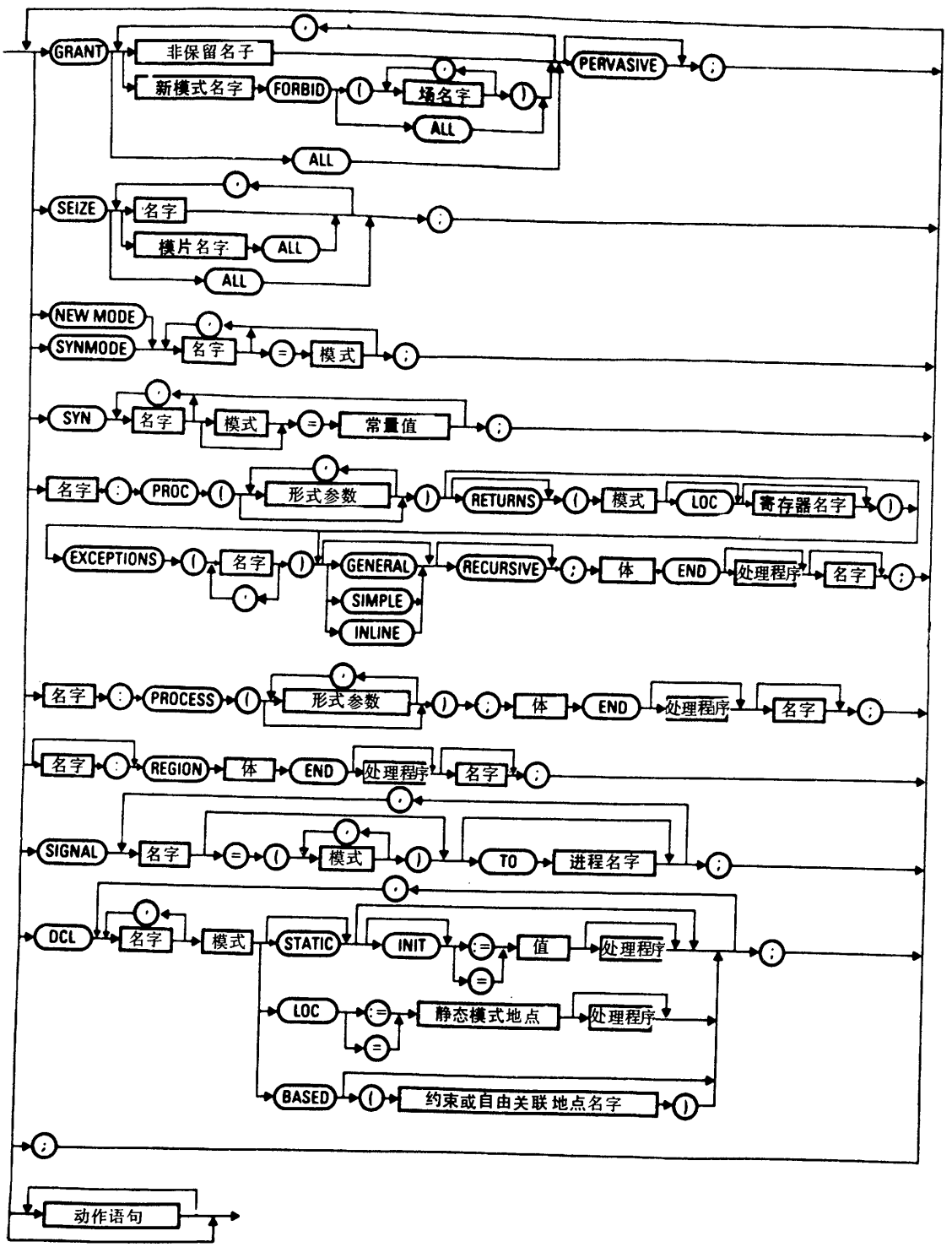
这些图是为便于阅读而设计的，并非供语法分解使用。

为了提高可读性，作了一些简化。因此，它们不能看成是定义性的，只是用于帮助理解CHILL。在本文件的其它地方，上下文无关语法的定义是用巴科斯-瑙尔范式给出的。

程序


名字
:
MODULE
REGION
体
END
处理程序
名字
;

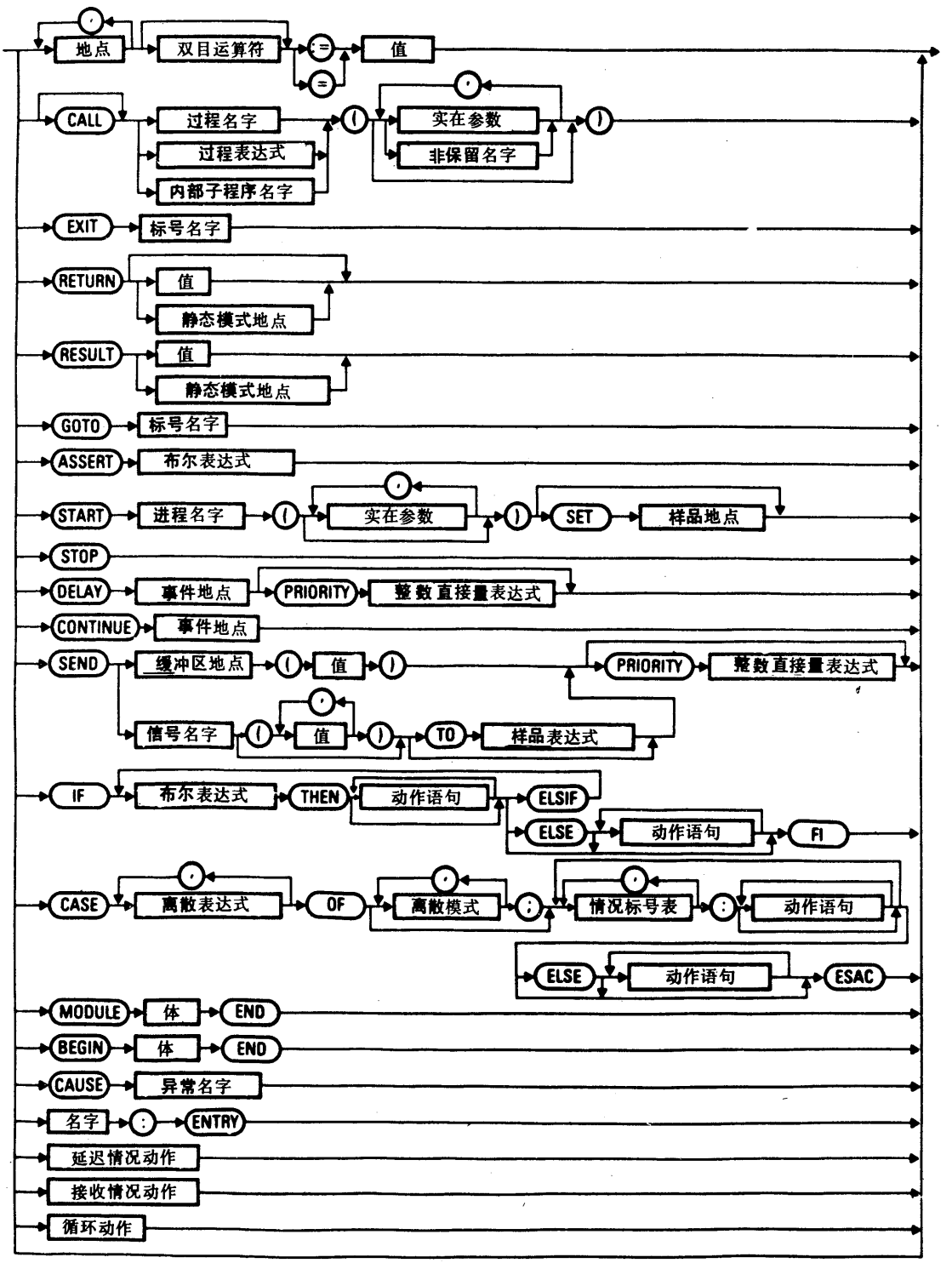
体


GRANT
非保留名子
新模式名字
FORBID
(
)
场名字
ALL
PERVASIVE
;
SEIZE
名字
模片名字
NEW MODE
SYNMODE
=
模式
SYN
常量值
PROC
形式参数
RETURNS
LOC
寄存器名字
EXCEPTIONS
GENERAL
SIMPLE
INLINE
RECURSIVE
PROCESS
REGION
SIGNAL
TO
进程名字
DCL
STATIC
INIT
:=
值
静态模式地点
BASED
约束或自由关联地点名字
动作语句

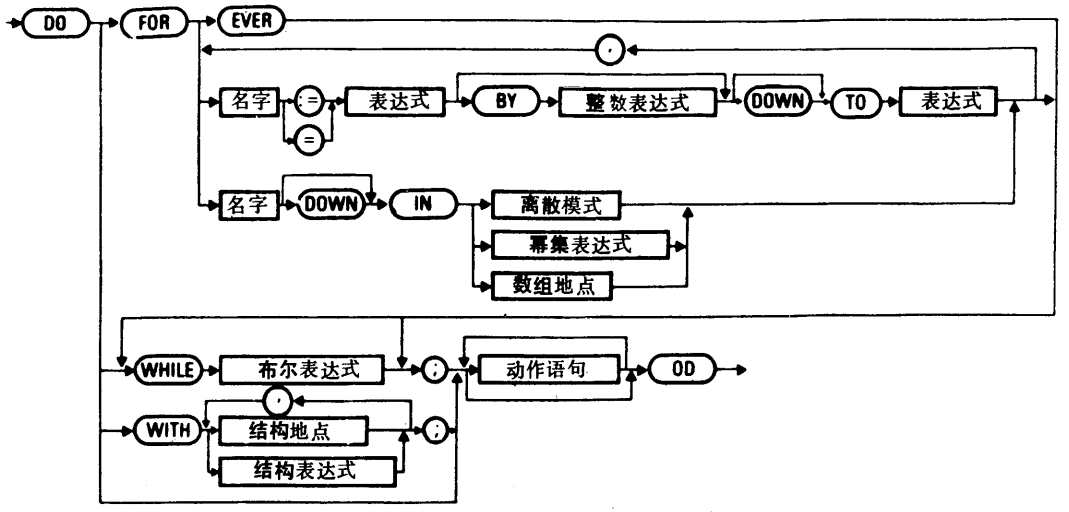
动作语句


名字
:
动作
处理程序
名字
;

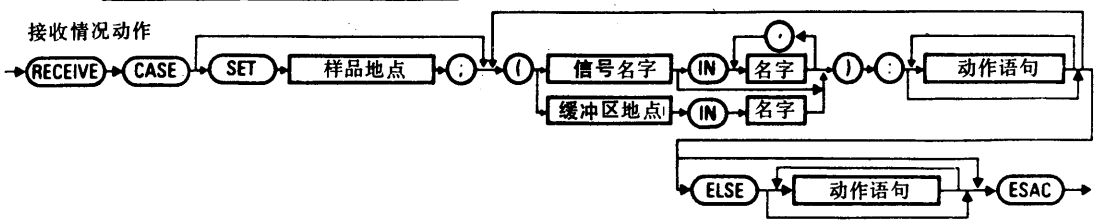
动作


地点
双目运算符
:=
=
值
CALL
过程名字
过程表达式
内部子程序名字
(
实在参数
非保留名字
)
EXIT
标号名字
RETURN
值
静态模式地点
RESULT
值
静态模式地点
GOTO
标号名字
ASSERT
布尔表达式
START
进程名字
(
实在参数
)
SET
样品地点
STOP
DELAY
事件地点
PRIORITY
整数直接量表达式
CONTINUE
事件地点
SEND
缓冲区地点
(
值
)
PRIORITY
整数直接量表达式
信号名字
(
值
)
TO
样品表达式
IF
布尔表达式
THEN
动作语句
ELSIF
ELSE
动作语句
FI
CASE
离散表达式
OF
离散模式
;
情况标号表
:
动作语句
ELSE
动作语句
ESAC
MODULE
体
END
BEGIN
体
END
CAUSE
异常名字
名字
:
ENTRY
延迟情况动作
接收情况动作
循环动作

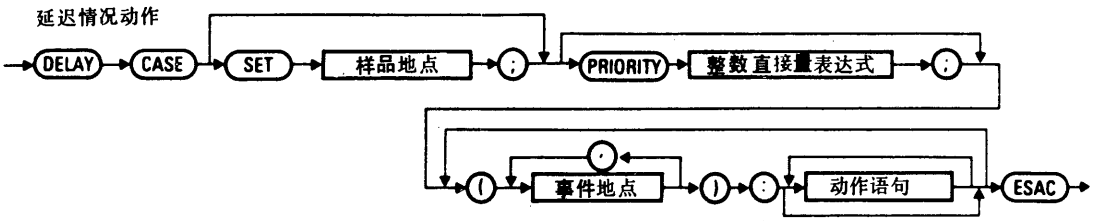
循环动作



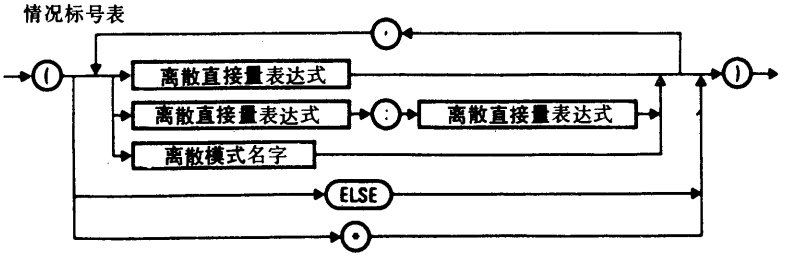
接收情况动作



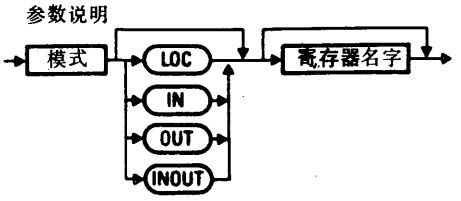
延迟情况动作



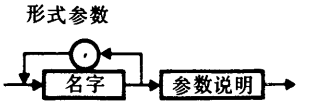
情况标号表



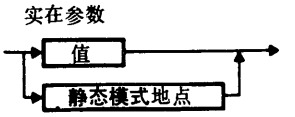
参数说明

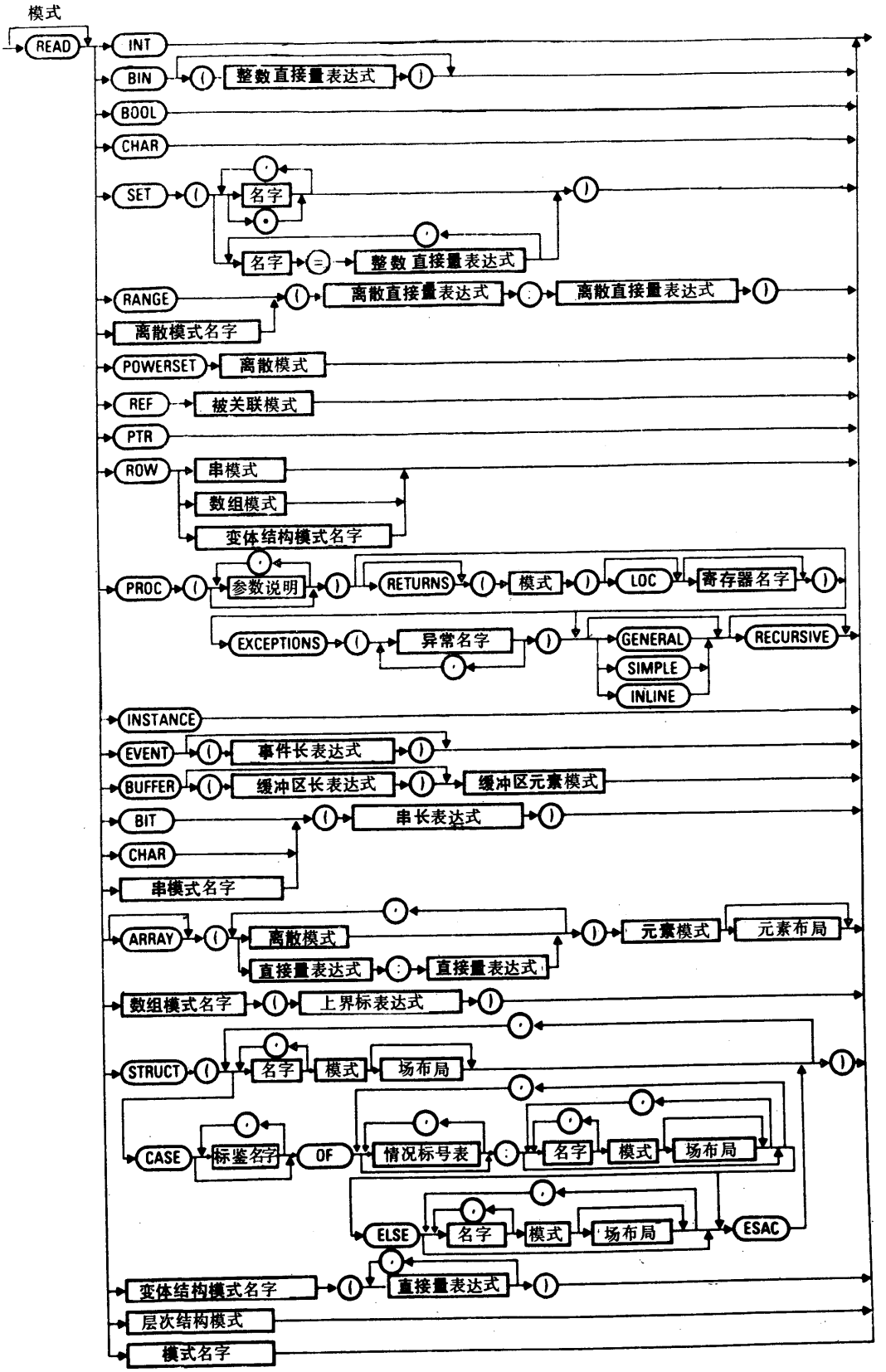


形式参数

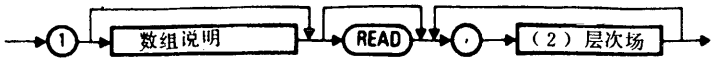


实在参数

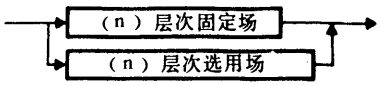



模式
READ
INT
BIN
整数直接量表达式
BOOL
CHAR
SET
名字
名字
=
整数直接量表达式
RANGE
离散直接量表达式
离散直接量表达式
离散模式名字
POWERSET
离散模式
REF
被关联模式
PTR
ROW
串模式
数组模式
变体结构模式名字
PROC
参数说明
RETURNS
模式
LOC
寄存器名字
EXCEPTIONS
异常名字
GENERAL
SIMPLE
INLINE
RECURSIVE
INSTANCE
EVENT
事件长表达式
BUFFER
缓冲区长表达式
缓冲区元素模式
BIT
串长表达式
CHAR
串模式名字
ARRAY
离散模式
直接量表达式
直接量表达式
元素模式
元素布局
数组模式名字
上界标表达式
STRUCT
名字
模式
场布局
CASE
标鉴名字
OF
情况标号表
名字
模式
场布局
ELSE
名字
模式
场布局
ESAC
变体结构模式名字
直接量表达式
层次结构模式
模式名字

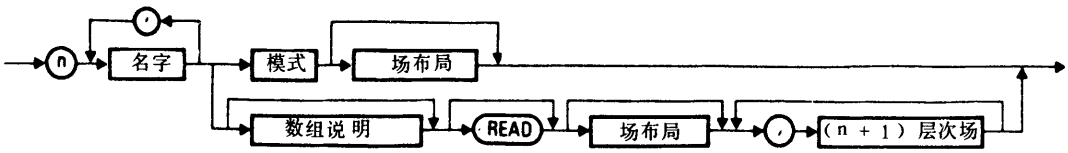
层次结构模式



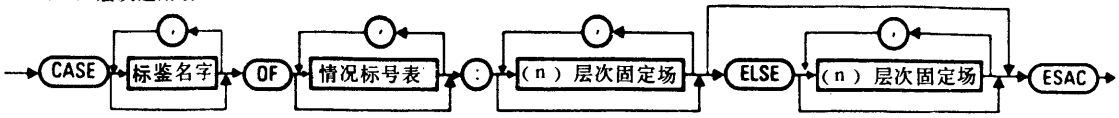
(n) 层次场



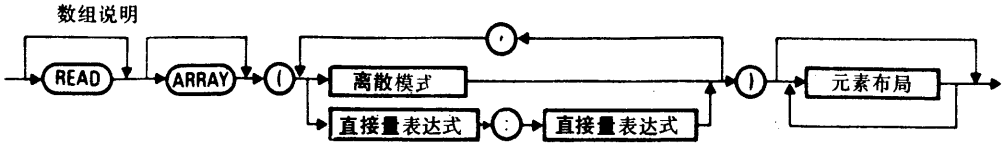
(n) 层次固定场



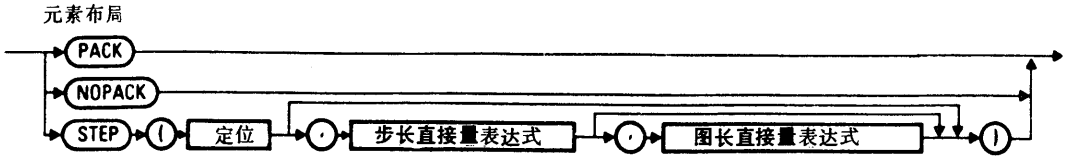
(n) 层次选用场



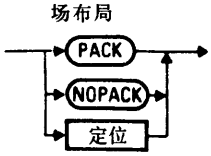
数组说明



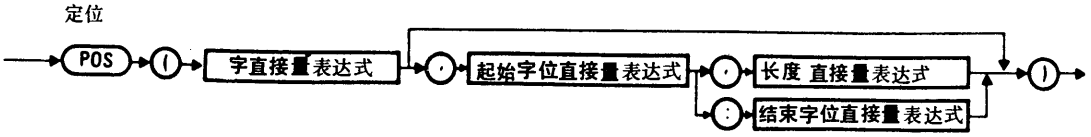
元素布局



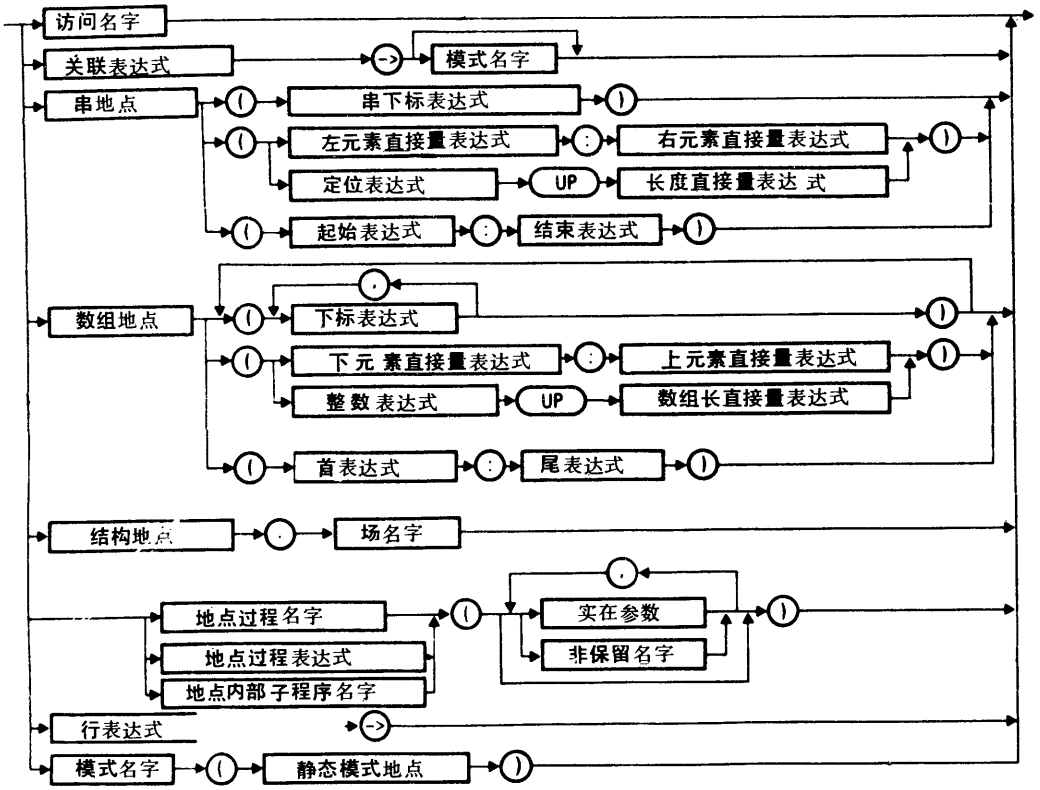
场布局



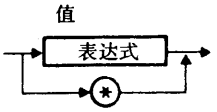
定位



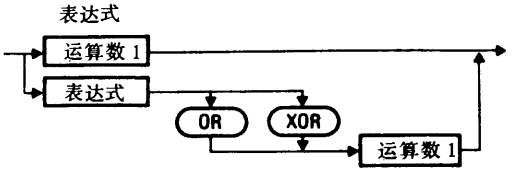
地点



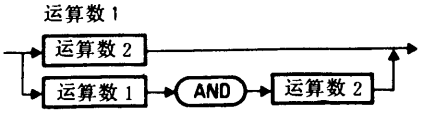
值



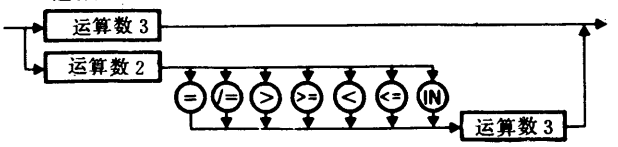
表达式



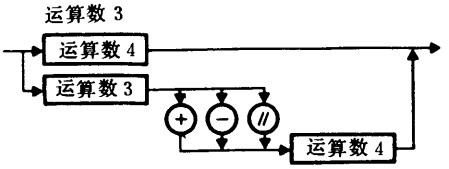
运算数 1



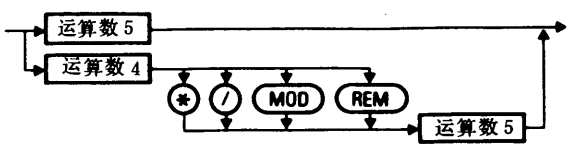
运算数 2



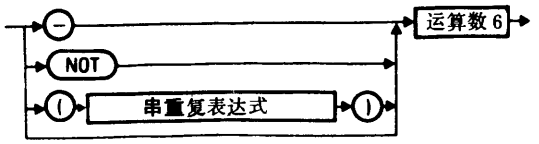
运算数 3



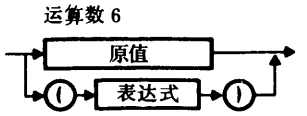
运算数 4



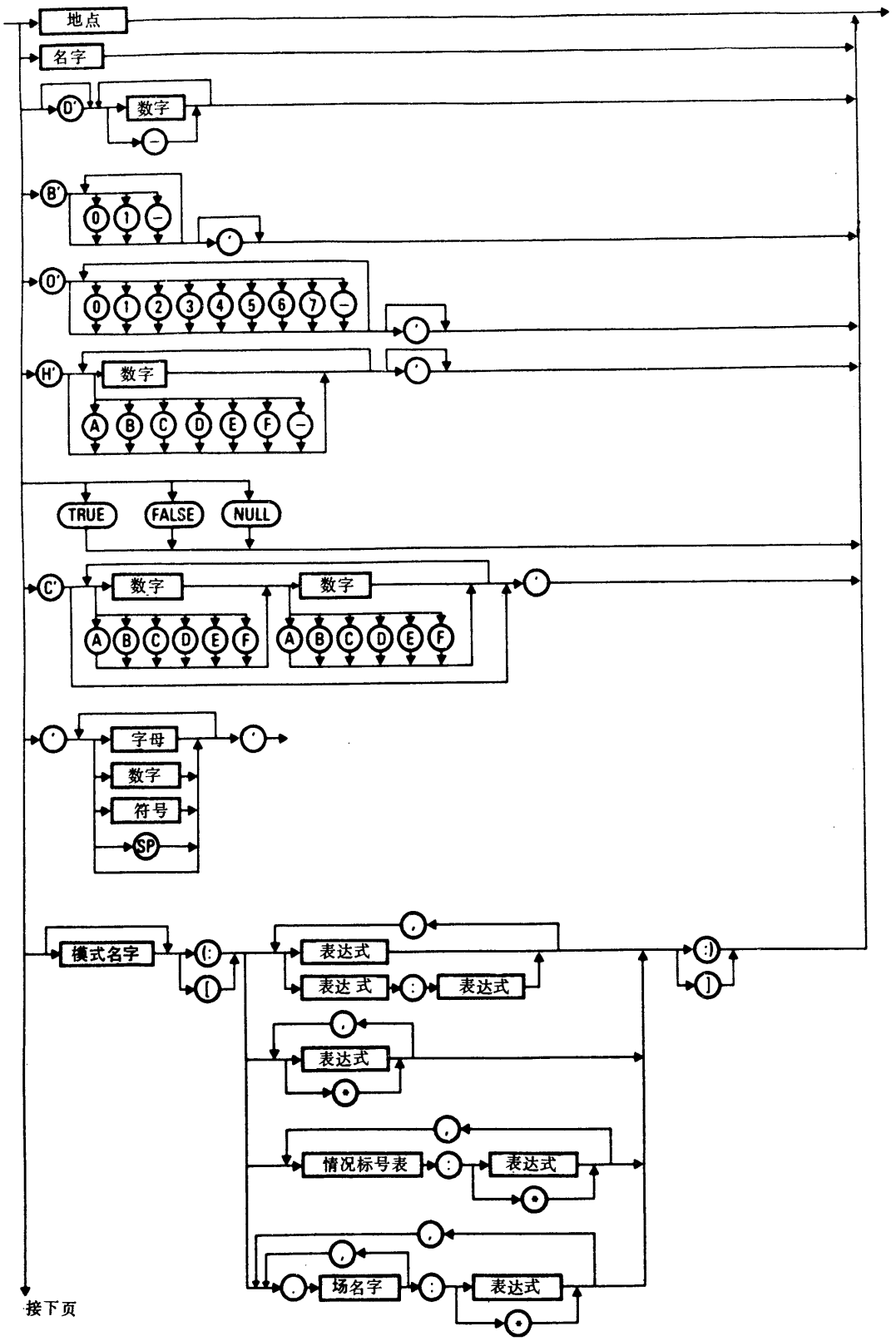
运算数 5



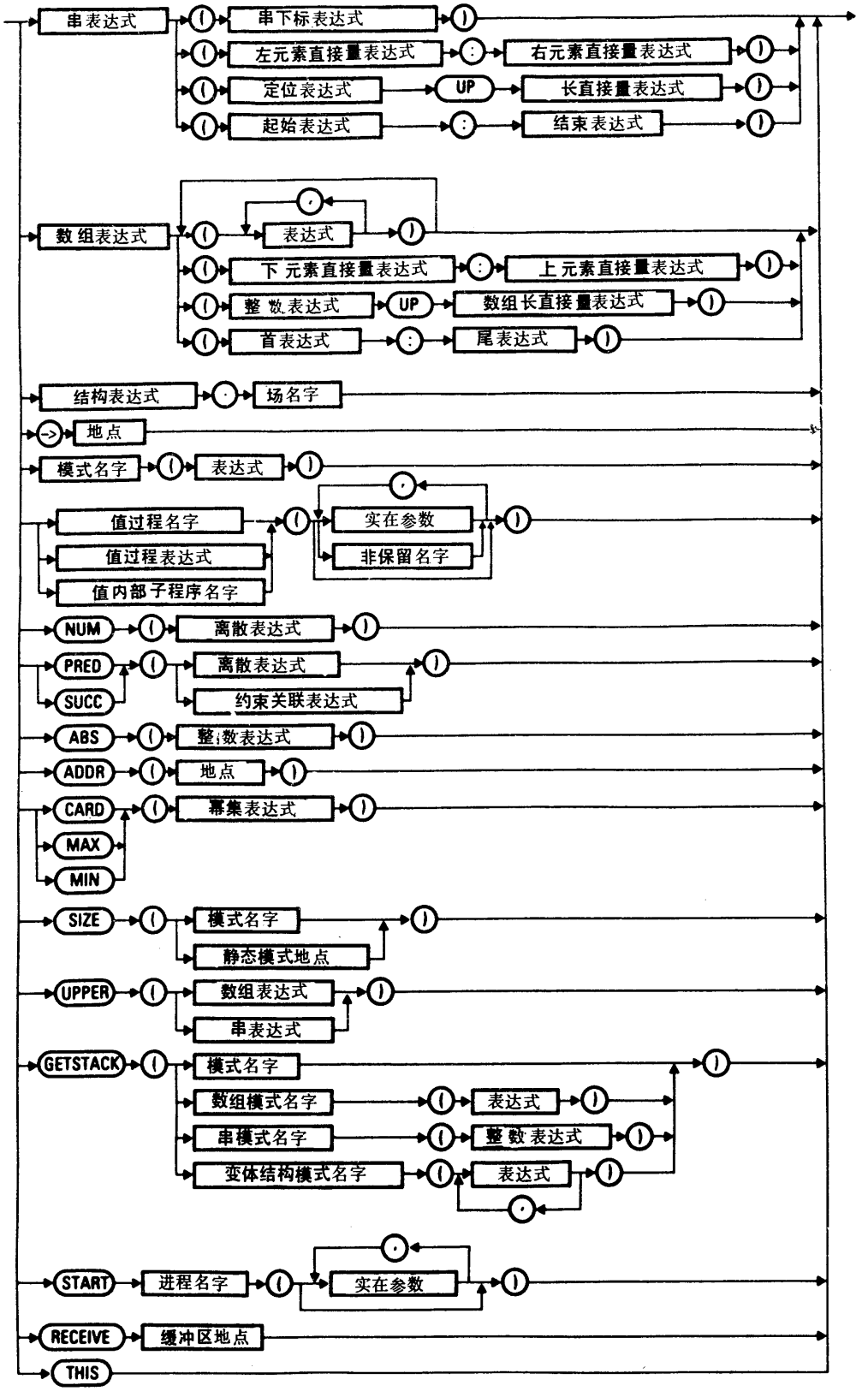
运算数 6



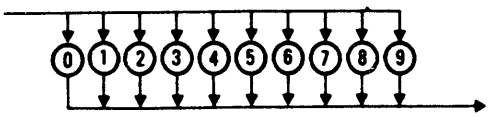
原值


地点
名字
D'
数字
B'
0
1
O'
2
3
4
5
6
7
H'
A
B
C
D
E
F
TRUE
FALSE
NULL
C'
字母
符号
SP
模式名字
表达式
情况标号表
场名字
接下页

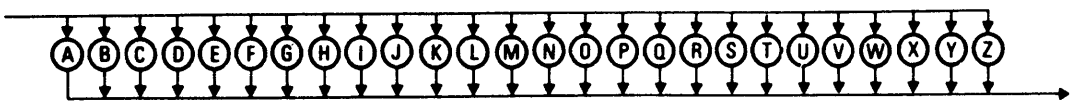
原值（接上页）


串表达式
串下标表达式
左元素直接量表达式
:
右元素直接量表达式
定位表达式
UP
长直接量表达式
起始表达式
:
结束表达式
数组表达式
表达式
下元素直接量表达式
:
上元素直接量表达式
整数表达式
UP
数组长直接量表达式
首表达式
:
尾表达式
结构表达式
.
场名字
->
地点
模式名字
表达式
值过程名字
值过程表达式
值内部子程序名字
实在参数
非保留名字
NUM
离散表达式
PRED
SUCC
离散表达式
约束关联表达式
ABS
整数表达式
ADDR
地点
CARD
MAX
MIN
幂集表达式
SIZE
模式名字
静态模式地点
UPPER
数组表达式
串表达式
GETSTACK
模式名字
数组模式名字
表达式
串模式名字
整数表达式
变体结构模式名字
表达式
START
进程名字
实在参数
RECEIVE
缓冲区地点
THIS

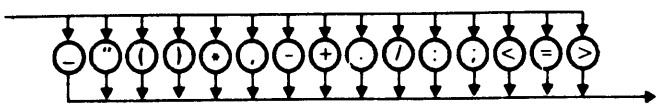
数字



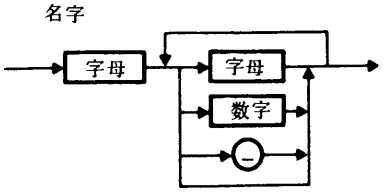
字母



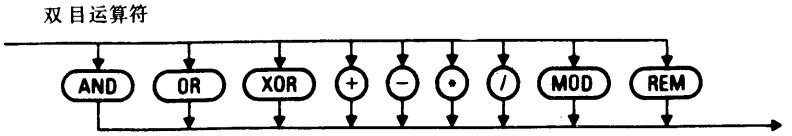
符号



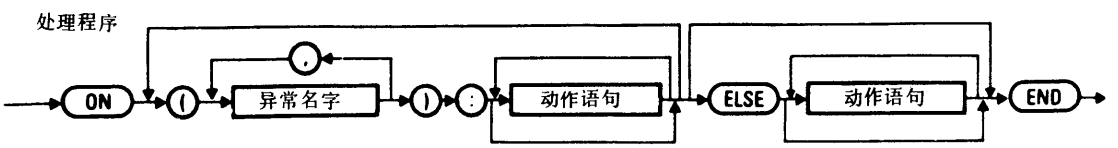
名字



双目运算符



处理程序



附录F　产生式规则索引

非终结符	定义处		应用处
	节号	页号	页号
〈*access name*〉〈访问名字〉	4.2.2		
〈*action*〉〈动作〉	6.1		
〈*action statement*〉〈动作语句〉	6.1		
〈*action statement list*〉〈动作语句表〉	7.2		
〈*actual parameter*〉〈实在参数〉	6.7		
〈*actual parameter list*〉〈实在参数表〉	6.7		
〈*alternative fields*〉〈选用场〉	3.10.4		
〈*apostrophe*〉〈撇号〉	5.2.4.7		
〈*arithmetic additive operator*〉〈算术加减运算符〉	5.3.5		
〈*arithmetic multiplicative operator*〉〈算术乘除运算符〉	5.3.6		
〈*array element*〉〈数组元素〉	4.2.7		
〈*array length*〉〈数组长度〉	4.2.7		
〈*array mode*〉〈数组模式〉	3.10.3		
〈*array slice*〉〈数组切片〉	4.2.14		
〈*array specification*〉〈数组说明〉	3.10.5		
〈*array tuple*〉〈数组多元组〉	5.2.5		
〈*assert action*〉〈断言动作〉	6.10		
〈*assigning operator*〉〈赋值运算符〉	6.2		
〈*assignment action*〉〈赋值动作〉	6.2		
〈*assignment symbol*〉〈赋值符号〉	6.2		
〈*based declaration*〉〈有基说明〉	4.1.4		
〈*begin-end block*〉〈分程序〉	7.3		
〈*begin-end body*〉〈分程序体〉	7.2		
〈*binary bit string literal*〉〈二进位串直接量〉	5.2.4.8		
〈*binary integer literal*〉〈二进整数直接量〉	5.2.4.2		
〈*bit string literal*〉〈字位串直接量〉	5.2.4.8		
〈*boolean literal*〉〈布尔直接量〉	5.2.4.3		
〈*boolean mode*〉〈布尔模式〉	3.4.3		
〈*bound reference mode*〉〈约束关联模式〉	3.6.2		
〈*bracketed action*〉〈加括号动作〉	6.1		
〈*buffer element mode*〉〈缓冲区元素模式〉	3.9.3		
〈*buffer length*〉〈缓冲区长度〉	3.9.3		
〈*buffer mode*〉〈缓冲区模式〉	3.9.3		
〈*buffer receive alternative*〉〈缓冲区接收选择对象〉	6.19.3		
〈*built-in routine call*〉〈内部子程序调用〉	11.1		
〈*built-in routine parameter*〉〈内部子程序参数〉	11.1		
〈*built-in routine para-meter list*〉〈内部子程序参数表〉	11.1		
〈*call action*〉〈调用动作〉	6.7		
〈*case action*〉〈情况动作〉	6.4		
〈*case alternative*〉〈情况选择对象〉	6.4		
〈*case label*〉〈情况标号〉	9.1.3		
〈*case label list*〉〈情况标号表〉	9.1.3		
〈*case label specification*〉〈情况标号说明〉	9.1.3		
〈*case selector list*〉〈情况选择表〉	6.4		
〈*cause action*〉〈引发动作〉	6.12		
〈*character*〉〈字符〉	5.2.4.7		
〈*character mode*〉〈字符模式〉	3.4.4		

续表

非终结符	定义处		应用处
	节号	页号	页号
〈*character string*〉〈字符串〉	2.4		
〈*character string literal*〉〈字符串直接量〉	5.2.4.7		
〈*CHILL directive*〉〈CHILL命令〉	2.6		
〈*CHILL value built-in routine call*〉〈CHILL值内部子程序调用〉	5.2.16		
〈*closed dyadic operator*〉〈闭式双目运算符〉	6.2		
〈*comment*〉〈注释〉	2.4		
〈*composite mode*〉〈组合模式〉	3.10.1		
〈*continue action*〉〈继续动作〉	6.15		
〈*control part*〉〈控制部分〉	6.5.1		
〈*data statement*〉〈数据语句〉	7.2		
〈*data statement list*〉〈数据语句表〉	7.2		
〈*decimal integer literal*〉〈十进整数直接量〉	5.2.4.2		
〈*de claration*〉〈说明〉	4.1.1		
〈*declaration statement*〉〈说明语句〉	4.1.1		
〈*defining mode*〉〈定义模式〉	3.2.1		
〈*definition statement*〉〈定义语句〉	7.2		
〈*delay action*〉〈延迟动作〉	6.16		
〈*delay alternative*〉〈延迟选择对象〉	6.17		
〈*delay case action*〉〈延迟情况动作〉	6.17		
〈*dereferenced bound reference*〉〈非关联化约束关联〉	4.2.3		
〈*dereferenced free reference*〉〈非关联化自由关联〉	4.2.4		
〈*dereferenced row*〉〈非关联化行〉	4.2.15		
〈*digit*〉〈数字〉	5.2.4.2		
〈*directive*〉〈命令〉	2.6		
〈*directive clause*〉〈命令子句〉	2.6		
〈*discrete mode*〉〈离散模式〉	3.4.1		
〈*do action*〉〈循环动作〉	6.5.1		
〈*dynamic mode location*〉〈动态模式地点〉	4.2.1		
〈*element layout*〉〈元素布局〉	3.10.6		
〈*element mode*〉〈元素模式〉	3.10.3		
〈*else clause*〉〈否则子句〉	6.3		
〈*emptiness literal*〉〈空直接量〉	5.2.4.5		
〈*empty*〉〈空〉	6.11		
〈*empty action*〉〈空动作〉	6.11		
〈*end*〉〈结束〉	4.2.13		
〈*end bit*〉〈结束位〉	3.10.6		
〈*end value*〉〈终值〉	6.5.2		
〈*entry definition*〉〈入口定义〉	7.4		
〈*entry statement*〉〈入口语句〉	7.4		
〈*event length*〉〈事件长度〉	3.9.2		
〈*event list*〉〈事件表〉	6.17		
〈*event mode*〉〈事件模式〉	3.9.2		
〈*exception list*〉〈异常表〉	3.7		
〈*exception name*〉〈异常名字〉	3.7		
〈*exit action*〉〈出口动作〉	6.6		
〈*expression*〉〈表达式〉	5.3.2		
〈*expression conversion*〉〈表达式转换〉	5.2.14		
〈*expression list*〉〈表达式表〉	4.2.7		
〈*field layout*〉〈场布局〉	3.10.6		
〈*field name list*〉〈场名字表〉	5.2.5		

续表

非终结符	定义处		应用处
	节号	页号	页号
〈*fields*〉〈场〉	3.10.4		
〈*first*〉〈首〉	4.2.14		
〈*fixed fields*〉〈固定场〉	3.10.4		
〈*forbid clause*〉〈禁止子句〉	9.2.6.2		
〈*forbid name list*〉〈禁止名字表〉	9.2.6.2		
〈*for control*〉〈步长型控制〉	6.5.2		
〈*formal parameter*〉〈形式参数〉	7.4		
〈*formal parameter list*〉〈形式参数表〉	7.4		
〈*free directive*〉〈释放命令〉	2.6		
〈*free reference mode*〉〈自由关联模式〉	3.6.3		
〈*generality*〉〈通用性〉	7.4		
〈*getstack argument*〉〈取栈顶变元〉	5.2.16		
〈*goto action*〉〈转向动作〉	6.9		
〈*granted element*〉〈开放元素〉	9.2.6.2		
〈*grant statement*〉〈开放语句〉	9.2.6.2		
〈*grant window*〉〈开放窗口〉	9.2.6.2		
〈*handler*〉〈处理程序〉	10.2		
〈*hexa decimal bit string literal*〉〈十六进位串直接量〉	5.2.4.8		
〈*hexadecimal digit*〉〈十六进数字〉	5.2.4.2		
〈*hexadecimal integer literal*〉〈十六进整数直接量〉	5.2.4.2		
〈*if action*〉〈条件动作〉	6.3		
〈*implementation directive*〉〈实现命令〉	……		
〈*index mode*〉〈下标模式〉	3.10.3		
〈*initialisation*〉〈初始化〉	4.1.2		
〈*instance mode*〉〈样品模式〉	3.8		
〈*integer literal*〉〈整数直接量〉	5.2.4.2		
〈*integer mode*〉〈整数模式〉	3.4.2		
〈*irrelevant*〉〈无关紧要〉	9.1.3		
〈*iteration*〉〈重复〉	6.5.2		
〈*labelled array tuple*〉〈有标号数组多元组〉	5.2.5		
〈*labelled structure tuple*〉〈有标号结构多元组〉	5.2.5		
〈*last*〉〈尾〉	4.2.14		
〈*left element*〉〈左元素〉	4.2.6		
〈*length*〉〈长度〉	3.10.6		
〈*letter*〉〈字母〉	5.2.4.7		
〈*level structure mode*〉〈层次结构模式〉	3.10.5		
〈*lifetime-bound initialisation*〉〈生存期初始化〉	4.1.2		
〈*literal*〉〈直接量〉	5.2.4.1		
〈*literal expression list*〉〈直接量表达式表〉	3.10.4		
〈*literal range*〉〈直接量区段〉	3.4.6		
〈*location*〉〈地点〉	4.2.1		
〈*location built-in routine call*〉〈地点内部子程序调用〉	4.2.11		
〈*location contents*〉〈地点内容〉	5.2.2		
〈*location conversion*〉〈地点转换〉	4.2.12		
〈*location declaration*〉〈地点说明〉	4.1.2		
〈*location enumeration*〉〈地点枚举〉	6.5.2		
〈*location procedure call*〉〈地点过程调用〉	4.2.10		
〈*loc-identity declaration*〉〈地点等同说明〉	4.1.3		
〈*loop counter*〉〈循环计数器〉	6.5.2		
〈*lower bound*〉〈下界〉	3.4.6		

续表

非终结符	定义处		应用处
	节号	页号	页号
〈*lower element*〉〈下元素〉	4.2.8		
〈*member mode*〉〈成员模式〉	3.5		
〈*membership operator*〉〈成员运算符〉	5.3.4		
〈*mode*〉〈模式〉	3.3		
〈*mode definition*〉〈模式定义〉	3.2.1		
〈*module*〉〈模块〉	7.6		
〈*module body*〉〈模块体〉	7.2		
〈*modulion name*〉〈模片名字〉	9.2.6.3		
〈*monadic operator*〉〈单目运算符〉	5.3.7		
〈*multiple assignment action*〉〈多重赋值动作〉	6.2		
〈*name*〉〈名字〉	2.2		
〈*name list*〉〈名字表〉	2.6		
〈*nested structure mode*〉〈嵌套结构模式〉	3.10.4		
〈*newmode definition statement*〉〈新模式定义语句〉	3.2.3		
〈(*n*)*level alternative*〉〈(*n*)层次选择对象〉	3.10.5		
〈(*n*)*level alternative fields*〉〈(*n*)层次选用场〉	3.10.5		
〈(*n*)*level fields*〉〈(*n*)层次场〉	3.10.5		
〈(*n*)*level fixed fields*〉〈(*n*)层次固定场〉	3.10.5		
〈(*n*)*level variant fields*〉〈(*n*)层次变体场〉	3.10.5		
〈*non-composite mode*〉〈非组合模式〉	3.3		
〈*numbered set element*〉〈编号集合元素〉	3.4.5		
〈*numbered set list*〉〈编号集合表〉	3.4.5		
〈*octal bit string literal*〉〈八进位串直接星〉	5.2.4.8		
〈*octal integer literal*〉〈八进整数直接量〉	5.2.4.2		
〈*on-alternative*〉〈异常处理选择对象〉	10.2		
〈*operand-1*〉〈运算数 1〉	5.3.3		
〈*operand-2*〉〈运算数 2〉	5.3.4		
〈*operand-3*〉〈运算数 3〉	5.3.5		
〈*operand-4*〉〈运算数 4〉	5.3.6		
〈*operand-5*〉〈运算数 5〉	5.3.7		
〈*operand-6*〉〈运算数 6〉	5.3.8		
〈*operator-3*〉〈运算符 3〉	5.3.4		
〈*operator-4*〉〈运算符 4〉	5.3.5		
〈*origin array mode name*〉〈原始数组模式名字〉	3.10.3		
〈*origin string mode name*〉〈原始串模式名字〉	3.10.2		
〈*origin variant structure mode name*〉〈原始变体结构模式名字〉	3.10.4		
〈*parameter attribute*〉〈参数属性〉	3.7		
〈*parameterised array mode*〉〈参数化数组模式〉	3.10.3		
〈*parameterised string mode*〉〈参数化串模式〉	3.10.2		
〈*parameterised structure mode*〉〈参数化结构模式〉	3.10.4		
〈*parameter list*〉〈参数表〉	3.7		
〈*parameter spec*〉〈参数说明〉	3.7		
〈*parenthesized expression*〉〈加括号表达式〉	5.3.8		
〈*pattern size*〉〈图长〉	3.10.6		
〈*pos*〉〈地位〉	3.10.6		
〈*position*〉〈定位〉	4.2.6		
〈*powerset difference operator*〉〈幂集求差运算符〉	5.3.5		
〈*powerset enumeration*〉〈幂集枚举〉	6.5.2		
〈*powerset inclusion operator*〉〈幂集蕴含运算符〉	5.3.4		
〈*powerset mode*〉〈幂集模式〉	3.5		

续表

非终结符	定义处		应用处
	节号	页号	页号
〈*powerset tuple*〉〈幂集多元组〉	5.2.5		
〈*primitive value*〉〈原值〉	5.2.1		
〈*priority*〉〈优先级〉	6.16		
〈*proc body*〉〈过程体〉	7.2		
〈*procedure attributes*〉〈过程属性〉	7.4		
〈*procedure call*〉〈过程调用〉	6.7		
〈*procedure definition*〉〈过程定义〉	7.4		
〈*procedure definition statement*〉〈过程定义语句〉	7.4		
〈*procedure literal*〉〈过程直接量〉	5.2.4.6		
〈*procedure mode*〉〈过程模式〉	3.7		
〈*process body*〉〈进程体〉	7.2		
〈*process definition*〉〈进程定义〉	7.5		
〈*process definition statement*〉〈进程定义语句〉	7.5		
〈*program*〉〈程序〉	7.3		
〈*range*〉〈区段〉	5.2.5		
〈*range enumeration*〉〈区段枚举〉	6.5.2		
〈*range list*〉〈区段表〉	6.4		
〈*range mode*〉〈区段模式〉	3.4.6		
〈*reach-bound initialisation*〉〈范围初始化〉	4.1.2		
〈*receive buffer case action*〉〈接收缓冲区情况动作〉	6.19.3		
〈*receive case action*〉〈接收情况动作〉	6.19.1		
〈*receive expression*〉〈接收表达式〉	5.2.18		
〈*receive signal case action*〉〈接收信号情况动作〉	6.19.2		
〈*rererence mode*〉〈关联模式〉	3.6.1		
〈*referenced location*〉〈被关联地点〉	5.2.13		
〈*referenced mode*〉〈被关联模式〉	3.6.2		
〈*region*〉〈区域〉	7.7		
〈*region body*〉〈区域体〉	7.2		
〈*relational operator*〉〈关系运算符〉	5.3.4		
〈*result*〉〈结果〉	6.8		
〈*result action*〉〈结果动作〉	6.8		
〈*result spec*〉〈结果说明〉	3.7		
〈*return action*〉〈返回动作〉	6.8		
〈*right element*〉〈右元素〉	4.2.6		
〈*row mode*〉〈行模式〉	3.6.4		
〈*seized element*〉〈引进元素〉	9.2.6.3		
〈*seize statement*〉〈引进语句〉	9.2.6.3		
〈*seize window*〉〈引进窗口〉	9.2.6.3		
〈*send action*〉〈发送动作〉	6.18.1		
〈*send buffer action*〉〈发送缓冲区动作〉	6.18.3		
〈*send signal action*〉〈发送信号动作〉	6.18.2		
〈*set element*〉〈集合元素〉	3.4.5		
〈*set list*〉〈集合表〉	3.4.5		
〈*set literal*〉〈集合直接量〉	5.2.4.4		
〈*set mode*〉〈集合模式〉	3.4.5		
〈*signal definition*〉〈信号定义〉	8.5		
〈*signal definition statement*〉〈信号定义语句〉	8.5		
〈*signal receive alternative*〉〈信号接收选择对象〉	6.19.2		
〈*single assignment action*〉〈单个赋值动作〉	6.2		
〈*space*〉〈空格〉	5.2.4.7		

续表

非终结符	定义处		应用处
	节号	页号	页号
〈*start*〉〈起始〉	4.2.13		
〈*start action*〉〈开动动作〉	6.13		
〈*start bit*〉〈起始位〉	3.10.6		
〈*start expression*〉〈开动表达式〉	5.2.17		
〈*start value*〉〈初值〉	6.5.2		
〈*static mode location*〉〈静态模式地点〉	4.2.1		
〈*step*〉〈步〉	3.10.6		
〈*step enumeration*〉〈步枚举〉	6.5.2		
〈*step size*〉〈步长〉	3.10.6		
〈*step value*〉〈步值〉	6.5.2		
〈*stop action*〉〈停止动作〉	6.14		
〈*string concatenation operator*〉〈串连接运算符〉	5.3.5		
〈*string element*〉〈串元素〉	4.2.5		
〈*string length*〉〈串长度〉	3.10.2		
〈*string mode*〉〈串模式〉	3.10.2		
〈*string repetition operator*〉〈串重复运算符〉	5.3.7		
〈*string slice*〉〈串切片〉	4.2.13		
〈*string type*〉〈串类型〉	3.10.2		
〈*structure field*〉〈结构场〉	4.2.9		
〈*structure mode*〉〈结构模式〉	3.10.4		
〈*structure tuple*〉〈结构多元组〉	5.2.5		
〈*sub-array*〉〈子数组〉	4.2.8		
〈*sub expression*〉〈子表达式〉	5.3.2		
〈*sub operand-1*〉〈子运算数 1〉	5.3.3		
〈*sub-operand-2*〉〈子运算数 2〉	5.3.4		
〈*sub-operand-3*〉〈子运算数 3〉	5.3.5		
〈*sub-operand-4*〉〈子运算数 4〉	5.3.6		
〈*sub string*〉〈子串〉	4.2.6		
〈*symbol*〉〈符号〉	5.2.4.7		
〈*synchronisation mode*〉〈同步模式〉	3.9.1		
〈*synmode definition statement*〉〈异名模式定义语句〉	3.2.2		
〈*synonym definition*〉〈异名定义〉	5.1		
〈*synonym definition statement*〉〈异名定义语句〉	5.1		
〈*tags*〉〈标签〉	3.10.4		
〈*then clause*〉〈则子句〉	6.3		
〈*tuple*〉〈多元组〉	5.2.5		
〈*undefined value*〉〈未定义值〉	5.3.1		
〈*unlabelled array tuple*〉〈无标号数组多元组〉	5.2.5		
〈*unlabelled structure tuple*〉〈无标号结构多元组〉	5.2.5		
〈*unnamed value*〉〈未命名值〉	3.4.5		
〈*unumbered set list*〉〈未编号集合表〉	3.4.5		
〈*upper bound*〉〈上界〉	3.4.6		
〈*upper element*〉〈上元素〉	4.2.8		
〈*upper index*〉〈上界标〉	3.10.3		
〈*value*〉〈值〉	5.3.1		
〈*value array element*〉〈值数组元素〉	5.2.9		
〈*value array slice*〉〈值数组切片〉	5.2.11		
〈*value built-in routine call*〉〈值内部子程序调用〉	5.2.16		
〈*value enumeration*〉〈值枚举〉	6.5.2		
〈*value name*〉〈值名字〉	5.2.3		

续表

非终结符	定义处		应用处
	节号	页号	页号
〈*value procedure call*〉〈值过程调用〉	5.2.15		
〈*value string element*〉〈值串元素〉	5.2.6		
〈*value string slice*〉〈值串切片〉	5.2.8		
〈*value structure field*〉〈值结构场〉	5.2.12		
〈*value sub-array*〉〈值子数组〉	5.2.10		
〈*value substring*〉〈值子串〉	5.2.7		
〈*variant alternative*〉〈变体选择对象〉	3.10.4		
〈*variant fields*〉〈变体场〉	3.10.4		
〈*visibility statement*〉〈可见性语句〉	9.2.6.1		
〈*while control*〉〈当型控制〉	6.5.3		
〈*with control*〉〈结构型控制〉	6.5.4		
〈*with part*〉〈结构型部分〉	6.5.4		
〈*word*〉〈字〉	3.10.6		
〈*zero-adic operator*〉〈零目运算符〉	5.2.19		

附录G 索 引

ABS
access 访问
access name 访问名字
action 动作
action statement 动作语句
action statement list 动作语句表
active 活化的
actual parameter 实在参数
actual parameter list 实在参数表
addition 加法
ADDR
ALL
all class 完全类
atternative fields 选用场
AND
and 与
apostrophe 撇号
applied occurrence 应用性出现
arithmetic additive operator 算术加减运算符
arithmetic multiplicative operator 算术乘除运算符
ARRAY
array element 数组元素
array expression 数组表达式
array location 数组地点
array mode 数组模式
array mode name 数组模式名字
array slice 数组切片
array tuple 数组多元组
ASSERT
assert action 断言动作
ASSERTFAIL
assigning operator 赋值运算符
assignment action 赋值动作
assignment conditions 赋值条件
assignment symbol 赋值符号

Backus-Naur Form 巴科斯——瑙尔范式
BASED
based declaration 有基说明
based name 有基名字
BEGIN
begin-end block 分程序
BIN
binary bit string literal 二进位串直接量
binary integer literal 二进整数直接量
BIT
bit string 字位串
bit string literal 字位串直接量
bit string mode 字位串模式
block 程序块
BOOL

boolean expression 布尔表达式
boolean literal 布尔直接量
boolean mode 布尔模式
boolean modename 布尔模式名字
bound or free reference location name 约束或自由关联地点名字
bound reference expression 约束关联表达式
bound reference mode 约束关联模式
bound reference mode name 约束关联模式名字
bracketed action 加括号动作
BUFFER
buffer element mode 缓冲区元素模式
buffer length 缓冲区长度
buffer location 缓冲区地点
buffer mode 缓冲区模式
buffer mode name 缓冲区模式名字
buffer receive alternative 缓冲区接收选择对象
built-in routine call 内部子程序调用
built-in rou tine name 内部子程序名字
built-in routine parameter 内部子程序参数
built-in routine parameter list 内部子程序参数表
built-in routines 内部子程序
BY

CALL
call action 调用动作
CARD
CASE
case action 情况动作
case alternative 情况选择对象
case label 情况标号
case label list 情况标号表
case selection 情况选择
case selector 情况选择式
case selector list 情况选择表
CAUSE
cause action 引发动作
change-sign operator 变号运算符
CHAR
character 字符
character mode 字符模式
character mode name 字符模式名字
character set 字符集合
character string 字符串
character string literal 字符串直接量
character string mode 字符串模式
CHILL directive CHILL 命令
CHILL value built-in routine call CHILL值内部子程序调用
class 类
comment 注释
compatible 相容的
complement 补
complete 完全的
composite mode 组合模式
concatenation operator 连接运算符
consistent 一致的

constant value 常量值
CONTINUE
continue action 继续动作
critical procedure 关键过程

DCL
decimal integer literal 十进整数直接量
declaration 说明
declaration statement 说明语句
defined by 被定义
defining mode 定义模式
defining occurence 定义性出现
DELAY
delay action 延迟动作
delay alternative 延迟选择对象
delay case action 延迟情况动作
DELAYFAIL
delaying 延迟
dereferenced bound reference 非关联化约束关联
dereferenced free reference 非关联化自由关联
dereferenced row 非关联化行
dereferencing 非关联化
derived class 导出类
derived syntax 导出语法
digit 数字
directive 命令
directive clause 命令子句
directly strongly visible 直接强可见
discrete expression 离散表达式
discrete literal expression 离散直接量表达式
discrete mode 离散模式
discrete mode name 离散模式名字
division 除法
DO
do action 循环动作
DOWN
dynamic array mode 动态数组模式
dynamic class 动态类
dynamic conditions 动态条件
dynamic mode 动态模式
dynamic mode location 动态模式地点
dynamic parameterised structure mode 动态参数化结构模式
dynamic properties 动态性质
dynamic string mode 动态串模式

element layout 元素布局
element mode 元素模式
ELSE
ELSIF
emptiness literal 空直接量
EMPTY
empty action 空动作
END
enter 进入

ENTRY
entry statement 入口语句
equality 相等
equivalent 等价
ESAC
EVENT
event length 事件长度
event location 事件地点
event mode 事件模式
event mode name 事件模式名字
EVER
examples 例子
exception 异常
exception handling 异常处理
exception list 异常表
exception name 异常名字
EXCEPTIONS
exclusive or 异或
EXIT
exit action 出口动作
expression 表达式
expression conversion 表达式转换
EXTINCT

FALSE
FI
field 场
field layout 场布局
field name 场名字
fixed field 固定场
fixed structure mode 固定结构模式
FOR
FORBID
forbid clause 禁止子句
for control 步长型控制
formal parameter 形式参数
format effector 格式作用符
FREE
free directive 释放命令
free reference expression 自由关联表达式
free reference mode 自由关联模式
free reference mode name 自由关联模式名字

GENERAL
general 通用的
generality 通用性
general procedure 通用过程
general procedure name 通用过程名字
GETSTACK
GOTO
goto action 转向动作
GRANT
granted 开放的
grant statement 开放语句

grant window 开放窗口
greater than 大于
greater than or equal 大于等于
group 组块

handler 处理程序
handler identification 处理程序的识别
hereditary property 继承性质
hexadecimal bit string literal 十六进位串直接量
hexadecimal integer literal 十六进整数直接量
holes 孔

IF
if action 条件动作
implementation directive 实现命令
implementation options 实现任选
implementation value built-in routine call 实现值内部子程序调用
implied name 隐含的名字
IN
index mode 下标模式
indirectly strongly visible 间接强可见的
inequality 不等
INIT
initialisation 初始化
INLINE
inline 直接插入
INOUT
INSTANCE
instance expression 样品表达式
instance location 样品地点
instance mode 样品模式
instance mode name 样品模式名字
INT
integer expression 整数表达式
integer literal 整数直接量
integer literal expression 整数直接量表达式
integer mode 整数模式
integer mode name 整数模式名字

labelled array tuple 有标号数组多元组
labelled structure tuple 有标号结构多元组
label name 标号名字
layout des cription 布局描述
l-equivalent l 等价
less than 小于
less than or equal 小于等于
level structure mode 层次结构模式
level number 层次号
lexical element 词法元素
lifetime 生存期
lifetime-bound initialisation 生存期初始化
literal 直接量
literal expression 直接量表达式

literal range 直接量区段
LOC
location 地点
location built-in routine call 地点内部子程序调用
location contents 地点内容
location conversion 地点转换
location declaration 地点说明
location do-with name 地点直接场名字
location enumeration 地点枚举
location enumeration name 地点枚举名字
location equivalence 地点等价
location name 地点名字
location procedure call 地点过程调用
loc-identity declaration 地点等同说明
loc-identity name 地点等同名字
loop counter 循环计数器
lower bound 下界
lower case 小写

mapped mode 被映射模式
MAX
member mode 成员模式
membership operator 成员运算符
metalanguage 元语言
MIN
MOD
mode 模式
mode checking 模式检验
mode definition 模式定义
MODEFAIL
mode name 模式名字
MODULE
module 模块
module action statement 模块动作语句
module name 模块名字
modulion 模片
modulo operator 求模运算符
multiple assignment action 多重赋值动作
multiplication 乘法
mutual exclusion 互斥

name 名字
name binding 名字约束
name creation 名字建立
name string 名字串
negation 非
nested structure mode 嵌套结构模式
NEWMODE
newmode definition statement 新模式定义语句
newmode name 新模式名字
non-apostrophe character 非撇号字符
non-composite mode 非组合模式
non-reserved name 非保留名字
NOPACK

NOT
novelty 新鲜性
NULL
null class 空类
NUM
numbered set element 编号集合元素
numbered set list 编号集合表

octal bit string literal 八进位串直接量
octal integer literal 八进整数直接量
OD
OF
ON
on dlternative 异常处理选择对象
OR
or 或
origin variant structure mode 原始变体结构模式
OUT
OVERFLOW

PACK
parameter attribute 参数属性
parameterisable 可参数化的
parameterised array mode 参数化数组模式
parameterised array mode name 参数化数组模式名字
parameterised string mode 参数化串模式
parameterised string mode name 参数化串模式名字
parameterised structure mode 参数化结构模式
parameterised structure mode name 参数化结构模式名字
parameter list 参数表
parameter spec 参数说明
parameter passing 参数传递
parent mode 父本模式
pass by location 按地点传递
pass by value 按值传递
path 通路
PERVASIVE
pervasive 渗透的
POS
POWERSET
powerset difference operator 幂集求差运算符
powerset enumeration 幂集枚举
powerset expression 幂集表达式
powerset inclusion operator 幂集蕴含运算符
powerset mode 幂集模式
powerset mode name 幂集模式名字
powerset tuple 幂集多元组
PRED
predefined name 预定义名字
primitive value 原值
PRIORITY
priority 优先级
PROC
procedure attributes 过程属性

procedure call 过程调用
procedure definition 过程定义
procedure definition statement 过程定义语句
procedure expression 过程表达式
procedure literal 过程直接量
procedure mode 过程模式
procedure mode name 过程模式名字
procedure name 过程名字

PROCESS

process 进程
process creation 进程建立
process definition 进程定义
process definition statement 进程定义语句
process name 进程名字
program 程序
program structure 程序结构

PTR

RANGE

range enumeration 区段枚举

RANGEFAIL

range mode 区段模式
range mode name 区段模式名字
reach 范围
reach-bound initialization 范围初始化
re-activation 重新活化

READ

read-compatible 读相容
read-only mode 只读模式
read-only property 只读性质

RECEIVE

receive buffer case action 接收缓冲区情况动作
receive case action 接收情况动作
receive expression 接收表达式
receive signal case action 接收信号情况动作

RECURSEFAIL

RECURSIVE

recursive definition 递归定义
recursive mode 递归模式
recursive procedure 递归过程
recursivity 递归性

REF

referability 可关联性
referable 可关联的
reference class 关联类
referenced location 被关联地点
referenced mode 被关联模式
referenced origin mode 被关联原始模式
reference mode 关联模式
reference value 关联值
referencing property 关联性质

REGION

region 区域
regional 区域性的
regionality 区域性

region name 区域名字
register name 寄存器名字
register specification 寄存器说明
relational operator 关系运算符
relations on modes 模式关系
REM
remainder operator 求余运算符
reserved name 保留名字
reserved name list 保留名字表
restrictable to 可限制为
RESULT
result 结果
result action 结果动作
result spec 结果说明
resulting class 结果类
result transmission 结果传送
RETURN
return action 返回动作
RETURNS
root mode 根模式
ROW
row expression 行表达式
row mode 行模式
row mode name 行模式名字

scope 作用域
SEIZE
seized 引进的
seize statement 引进语句
seize window 引进窗口
semantic categories 语义范畴
semantic description 语义描述
semantics 语义
SEND
send action 发送动作
send buffer action 发送缓冲区动作
send signal action 发送信号动作
SET
set element name 集合元素名字
set list 集合表
set literal 集合直接量
set mode 集合模式
set mode name 集合模式名字
SIGNAL
signal definition 信号定义
signal definition statement 信号定义语句
signal name 信号名字
signal receive alternative 信号接收选择对象
similar 相似的
SIMPLE
simple 简单的
single assignment action 单个赋值动作
SIZE
size 大小
space 空格

SPACEFAIL
special name 专用名字
special symbol 专用符号
START
start action 开动动作
start expression 开动表达式
STATIC
static 静态的
static conditions 静态条件
static mode 静态模式
static mode location 静态模式地点
static properties 静态性质
STEP
step enumeration 步枚举
step value 步值
STOP
stop action 停止动作
storage allocation 存储分配
strict syntax 严格语法
string concatenation operator 串连接运算符
string element 串元素
string expression 串表达式
string length 串长度
string location 串地点
string mode 串模式
string mode name 串模式名字
string repetition operator 串重复运算符
string slice 串切片
string type 串类型
strongly visible 强可见的
strong value 强值
STRUCT
structure field 结构场
structure expression 结构表达式
structure location 结构地点
structure mode 结构模式
structure mode name 结构模式名字
structure tuple 结构多元组
sub-array 子数组
substring 子串
substraction 减法
SUCC
SYN
synchronisation mode 同步模式
synchronisation property 同步性质
SYNMODE
synmode definition statement 异名模式定义语句
synmode name 异名模式名字
synonym definition 异名定义
synonym name 异名名字
synonymous with 与……同义
syntax description 语法描述
syntax options 语法任选

TAGFAIL

tag field 标签场
tag field name 标签场名字
tagged parameterised property 带标签参数化性质
tagged parameterised structure mode 带标签参数化结构模式
tagged variant structure mode 带标签变体结构模式
tagless parameterised structure mode 无标签参数化结构模式
tagless variant structure mode 无标签变体结构模式
termination 结束，终止
THEN
THIS
TO
TRUE
tuple 多元组

undefined location 无定义地点，未定义地点
undefined synonym name 无定义异名名字，未定义异名名字
undefined value 无定义值，未定义值
underline symbol 下划线符号
unlabelled array tuple 无标号数组多元组
unlabelled structure tuple 无标号结构多元组
unnamed value 未命名值
unnumbered set list 未编号集合表
UP
UPPER
upper bound 上界

value 值
value array element 值数组元素
value array slice 值数组切片
value built-in routine call 值内部子程序调用
value class 值类
value do-with name 值直接场名字
value enumeration 值枚举
value enumeration name 值枚举名字
value equivalence 值等价
value name 值名字
value procedure call 值过程调用
value receive name 值接收名字
value string element 值串元素
value string slice 值串切片
value structure field 值结构场
value sub-array 值子数组
value substring 值子串
variant alternative 变体选择对象
variant field 变体场
variant structure mode 变体结构模式
variant structure mode name 变体结构模式名字
v-equiva lent V 等价
visibility 可见性
visibility statement 可见性语句
visible 可见的
weakly visible 弱可见的

WHILE
while control　当型控制
WITH
with control　结构型控制

XOR

zero-adic operator　零目运算符

北京印刷-中国·统一书号：15045·总3044-有5415