第 2 章　泛代数和代数数据类型

2.1 引　　言

第 2 章到第 4 章提出一种用于对可计算函数进行编程的语言，叫做 PCF 语言。它是基于类型化λ演算的函数式语言。该语言的设计目的是便于后面各章的分析和讨论，而不是把它作为编写大程序的实际语言。PCF 语言可以看成由三部分组成：包括自然数类型和布尔类型的基本数据类型，带函数类型、积类型与和类型等的纯类型化λ演算，涉及递归函数的不动点算子。这三章分别对像 PCF 这样语言的三个组成部分进行透彻的研究，以达到全面系统了解这样的语言。本章研究像自然数类型和布尔类型这样的代数数据类型。

如果用其他的基本数据类型，如字符类型和串类型，来代替 PCF 语言的自然数类型和布尔类型，则可以得到有类似类型构造（函数和序对等）的语言，但面向的是不同种类的数据的求值。如果想对 PCF 作这样的修改，则必须确定命名基本字符和串的一种方式，就像用 0，1，2，… 命名自然数那样，并选择以字符和串作为操作对象的基本函数，代替自然数上的+和相等比较等运算以及布尔值上的运算。有了这些就可以定义这些基本数据类型的项语言的语法，然后写一组项之间的等式作为公理来刻画这些函数的性质，即给出该项语言的公理语义。一旦有了项的语法和公理语义，就可以进一步选择归约公理，编写程序，研究指称语义。

一个**代数数据类型**包括一个或多个值集，如自然数集、布尔值集、字符集或串集，再加上一组在这些集合上的函数。对代数数据类型的一个基本限制是其函数不能有函数变元，这就是“代数的”的含义。需要说明的一点是，对于属于一个代数的若干个集合，本章使用泛代数的标准术语。因此，基本“类型”符号，如 *nat*，*bool*，*char* 和 *string* 等，当它们用于代数项时，叫做**类别**（*sort*）。这样做的目的是，既维护和代数方面的文献的一致性，同时又强调由集合和相关函数组成的代数数据类型和单纯的集合是有区别的。在代数数据类型理论中，类型和类别的区别在于类型有一组运算而类别没有。

泛代数（*universal algebra*），也叫做**等式逻辑**（*equational logic*），是用于定义和研究代数数据类型的一般数学框架。在泛代数中，一个代数数据类型的公理语义由项之间的一组等式给出。代数数据类型的指称语义涉及的结构叫做**代数**，它由一组集合和一组函数组成，对应每个类别有一个集合，对应项中使用的每个函数符号有一个函数。代数项的操作语义由有向的代数等式（即归约公理）给出，习惯上称它们为**重写规则**。可以用泛代数定义和研究的数据类型有自然数、布尔值、表、有限集合、多重集合、栈、队列和树等。

本章研究泛代数和它在程序设计中定义常用数据类型时的作用，主要关心公理语义（一组等式）和指称语义（代数）之间的联系，并另用单独一节研究操作语义。本章虽只研究代数，但覆盖了大多数逻辑系统的一些公共议题。本章的主要内容有：

（1）代数项和它们在多类别代数中的解释；

（2）等式规范（也叫代数规范）和等式证明系统；

（3）等式证明系统的可靠性和完备性（公理语义和指称语义的等价）；

（4）代数之间的同态关系和初始代数；

（5）数据类型的代数理论；

（6）从代数规范导出的重写规则（操作语义）。

前四节对泛代数的数学系统提供一个简明介绍，随后一节关于数据类型的代数理论的讨

论指出数学中传统的关注点和程序设计中代数数据类型使用之间的一些区别。最后一节考虑在代数项上的归约，它有两个应用，分析等式规范的性质和为代数项上的计算建模。

在数据类型的代数理论中还有许多专题。本书所忽略的重要专题有：层次规范、参数化规范、从一个规范到另一个规范的求精、代数规范的实现的正确性等。在讨论函数应用产生错误引起的问题时，本书没有介绍诸如使用带部分函数的代数和有序类别代数等更复杂方法。对这些专题，有兴趣的读者可以参考有关文献。

本章研究的是 PCF 这样的函数式语言的代数数据类型部分，此时不会出现函数作为变元或返回值的情况，因此对它的研究比λ演算的研究要容易得多。先研究泛代数而后研究类型化λ演算的理由是，许多技术概念在这里以更简单和易于理解的形式出现，因而易于讲授和学习。

2.2 代数、基调和项

2.2.1 代数

一个代数由叫做**载体**（*carrier*）的一个或多个集合、这些载体上的一组特征元素和**一阶函数**（或叫做**代数函数**）

$$f : A_1 \times \ldots \times A_k \to A$$

组成。例如

$$\mathcal{N} = \langle \mathcal{N}, 0, 1, +, * \rangle$$

其载体$\mathcal{N}$是自然数集合，特征元素 $0, 1 \in \mathcal{N}$，函数$+, * : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$。一个载体的“特征”元素和它的其他元素的区别是，在代数项语言中每个特征元素被给出一个名字。一种简单的做法是把 $0, 1 \in \mathcal{N}$这样的特征元素叫做零元函数，以便把特征元素和函数统一。

多个载体的例子有代数

$$\mathcal{A}_{pcf} = \langle \mathcal{N}, \mathcal{B}, 0, 1, \ldots, +, true, false, Eq\ ?, \ldots \rangle$$

其中$\mathcal{N}$是自然数集合，$\mathcal{B}$是布尔值集合，0, 1, …是自然数，+是加法函数等等。它们是在写 PCF 的基本数值和布尔表达式时通常想到的数学值。在研究代数时，将把 PCF 的语法表达式和在这个代数中的语义值精确地对应起来。

2.2.2 代数项的语法

现在开始逐步给出代数的一种语法描述，有穷的语法表示在计算机科学中十分重要。首先从代数项的语法开始。

代数项的语法依赖于基本符号和它们的类型，这些信息收集起来后被叫做基调。前面已提到，在代数中通常把基本类型的符号叫做类别。

基调（*signature*） $\Sigma = \langle S, F \rangle$ 由下列两部分组成：

（1）集合 S，其元素叫做**类别**；

（2）序对$\langle f, s_1 \times \ldots \times s_k \to s \rangle$的集合 F，其中 $s_1, \ldots, s_k, s \in S$ 并且 f 不出现在两个不同的序对中。

在基调的定义中，出现在 F 中的符号 f 叫做**类型化的函数符号**（或简称**函数符号**），并且表达式 $s_1 \times \ldots \times s_k \to s$ 叫做 S **上的一阶**（或代数）**函数类型**。通常用 $f : \tau$代替$\langle f, \tau \rangle \in F$。

类别是代数载体的名称，函数符号 $f : s_1 \times \ldots \times s_k \to s$ 是代数的一个 k 元函数的名称。允许 $k = 0$，从而函数符号 $f : s$ 也可以是对应类别 s 的载体的元素的名称，通常称它们为**常量符号**。不允许有 $f : \tau$和 $f : \tau'$，即任何一个常量符号或函数符号只有唯一的类别或类型。

例 2.1 一个简单的例子是$\Sigma_{\mathcal{N}} = \langle S, F \rangle$，其中 $S = \{nat\}$仅含一个类别。F 包含 $0 : nat$, $1 : nat$,

+: $nat \times nat \to nat$ 和∗: $nat \times nat \to nat$。描述基调的习惯写法如下：

sorts : *nat*

fctns : 0, 1 : *nat*

+, ∗ : $nat \times nat \to nat$

在这种表示法中，通常把同样类型的函数符号写在同一行上。 □

定义基调的目的是用于写代数项，但还得考虑项中有变量的情况。假定有一个无穷的符号集合 V，其元素称为**变量**，并假定这些符号和所有常量符号、函数符号和类别符号都不一样。变量未指定类型是没有意义的，因此还需要列出变量的类别。**类别指派**是变量与类别序对的一个有限集合：

$$\Gamma = \{x_1 : s_1, \ldots, x_k : s_k\}$$

其中任何变量不会被指派多个类别。给定一个基调$\Sigma = \langle S, F\rangle$和类别指派Γ，Σ和Γ上类别 s 良形的代数项集合 $Terms^s(\Sigma, \Gamma)$定义如下：

（1）如果 $x : s \in \Gamma$，那么 $x \in Terms^s(\Sigma, \Gamma)$；

（2）如果 $f : s_1 \times \ldots \times s_k \to s$ 并且 $M_i \in Terms^{s_i}(\Sigma, \Gamma)$ $(i = 1, \ldots, k)$，那么 $f\ M_1 \ldots M_k \in Terms^s(\Sigma, \Gamma)$。

当 $k = 0$ 时，第二个子句应该解释成：如果 $f : s$，那么 $f \in Terms^s(\Sigma, \Gamma)$。

例如，$0 \in Terms^{nat}(\Sigma_{\mathcal{N}}, \emptyset)$，+01 也是 $Terms^{nat}(\Sigma_{\mathcal{N}}, \emptyset)$的项，因为+是$\Sigma_{\mathcal{N}}$上的二元函数符号。这两个项都不含变量，不含变量的项称为**闭项**。对任何类别指派Γ，+01 总是 $Terms^{nat}(\Sigma_{\mathcal{N}}, \Gamma)$ 的项。为了和 PCF 一致，把 0+1 看成是+01 的语法美化。如果只有一个类别，则总假定所有的变量都属于这个类别，这时类别指派可以简化为列出所有的变量即可。下面用 $Var(M)$ 表示出现在项 M 中的变量集合。

因为代数项中无约束变元，因此代换$[N/x]M$ 就是简单地把 M 中 x 的每个出现都用 N 代替。很容易看出，如果项中的一个变量由适当类别的项代替的话，那么可以得到一个良形的项，其类别和代换之前一样。这可由下面的引理精确地给出。下面用$\Gamma, x : s'$表示类别指派

$$\Gamma, x : s' = \Gamma \cup \{ x : s'\}$$

其中 x 在Γ中没有出现。

引理 2.1 如果 $M \in Terms^s(\Sigma, \Gamma, x : s')$并且 $N \in Terms^{s'}(\Sigma, \Gamma)$，那么$[N/x]M \in Terms^s(\Sigma, \Gamma)$。

证明 按 $Terms^s(\Sigma, \Gamma, x : s')$中项的结构进行归纳。这是本章第一个归纳证明，因此叙述得详细一些，以后的归纳证明仅给出要点。因为每个项都是变量、常量符号或者函数符号应用到变元的形式，因此代数项上的归纳只有一种基本情况（变量）和一种归纳步骤（函数符号）。对基本情况必须直接证明引理，而在归纳步骤中，假定引理对所有子项都成立。这个假定叫做归纳假设。

对基本情况，考虑变量 $y \in Terms^s(\Sigma, \Gamma, x : s')$。如果 y 和 x 不同，那么$[N/x]\ y$ 就是 y。在这种情况下肯定有 $y : s \in \Gamma$，因此 $y \in Terms^s(\Sigma, \Gamma)$。如果项是变量 x 本身，那么$[N/x]\ x$ 的结果是项 N。这时 x 一定有类别 s，则 N 一定属于 $Terms^s(\Sigma, \Gamma)$。这就证明了归纳的基本情况。

对于归纳步骤，假定$[N/x]M_i \in Terms^{s_i}(\Sigma, \Gamma)$（$1 \le i \le k$），并且考察形式为 $f\ M_1 \ldots M_k$ 的项。在这个项中用 N 对 x 进行代换的结果可以写成 $f\ [N/x]M_1 \ldots [N/x]M_k$。因为已经假定 $f\ M_1 \ldots M_k \in Terms^s(\Sigma, \Gamma, x : s')$，因此 f 必定有类型 $s_1 \times \ldots \times s_k \to s$。现在有$[N/x]M_i \in Terms^{s_i}(\Sigma, \Gamma)$（$1 \le i \le k$），因此 $f\ [N/x]M_1 \ldots [N/x]M_k \in Terms^s(\Sigma, \Gamma)$。这就结束了该引理的归纳证明。读者应该检验在归纳步骤中 $k = 0$ 的情况。 □

例 2.2 可以用基调$\Sigma_{stk} = \langle S, F\rangle$来写自然数和栈表达式，其中 $S = \{nat, stack\}$，F 包含下列函数符号。

sorts : *nat*, *stack*

fctns : 0, 1, 2, … : *nat*

$+, * : nat \times nat \to nat$

$empty : stack$

$push : nat \times stack \to stack$

$pop : stack \to stack$

$top : stack \to nat$

push 2 (*push* 1 (*push* 0 *empty*))是该基调的项，在这个基调的标准解释中它表示有元素 0，1 和 2 的栈。使用栈变量 *s* : *stack*，表达式 *push* 1 (*push* 0 *s*)定义把 0 和 1 压进 *s* 后的栈。 □

2.2.3 代数以及项在代数中的解释

确定了代数项的语法描述后，需要一种方式将代数项解释到相应代数上的元素。

代数是为代数项提供含义（指称语义）的数学结构。为了显式地表示从项到代数的映射，需要先把基调的类别和函数符号映射到代数中。如果Σ是一个基调，那么**Σ代数**$\mathcal{A}$包含：

（1）对每个 $s \in S$，正好有一个载体 A^s；

（2）一个解释映射$\mathcal{I}$把函数

$$\mathcal{I}(f) : A^{s_1} \times \ldots \times A^{s_k} \to A^s$$

指派给函数符号 $f : s_1 \times \ldots \times s_k \to s \in F$，并且把$\mathcal{I}(f) \in A^s$ 指派给常量符号 $f : s \in F$。

如果Σ只有一个类别，那么称它为**单类别**（*single-sorted*）代数，否则为**多类别**（*many-sorted*, *multi-sorted*）代数。

如果$\mathcal{A} = \langle \{A^s\}_{s \in S}, \mathcal{I} \rangle$是一个Σ代数，并且 *f* 是Σ的函数符号，为方便起见，通常把$\mathcal{I}(f)$写成 $f^{\mathcal{A}}$。在函数符号是有限多或可数多的情况下，一般是列出所有的函数，像代数$\mathcal{N}$ 和$\mathcal{A}_{pcf}$给出的那样。根据这两个约定，把$\Sigma_{\mathcal{N}}$代数$\mathcal{N}$ 写成

$$\mathcal{N} = \langle \mathcal{N}, 0^{\mathcal{N}}, 1^{\mathcal{N}}, +^{\mathcal{N}}, *^{\mathcal{N}} \rangle$$

它给出了解释$\Sigma_{\mathcal{N}}$上的项所需的全部信息，因为解释映射$\mathcal{I}$ 已经由这个代数的函数命名方式给出。

确定项 $M \in Terms^s(\Sigma, \Gamma)$在Σ代数中的含义相对来说是简单的事情。可能唯一不太熟悉的技术方法是对变量指派载体上的值。代数$\mathcal{A}$的**环境**η是从变量集合到$\mathcal{A}$的各载体集合的并集的一个映射 $\eta : V \to \cup_s A^s$。需要环境的原因是项 *M* 中可能含变量，在确定 *M* 的含义前必须给变量一个值。当然，一个环境必须把每个变量映射到恰当载体上的值。如果对每个 $x : s \in \Gamma$都有$\eta(x) \in A^s$，就说环境η **满足**Γ。

为$\mathcal{A}$ 给出一个满足Γ的环境η后，可以定义任何项 $M \in Terms^s(\Sigma, \Gamma)$在$\eta$下的含义$\mathcal{A}[\![M]\!]\eta$如下（双括号[![ ]!]在此用来表示求语法项的语义）：

$$\mathcal{A}[\![x]\!]\eta = \eta(x)$$

$$\mathcal{A}[\![M]\!]\eta = f^{\mathcal{A}}(\mathcal{A}[\![M_1]\!]\eta, \ldots, \mathcal{A}[\![M_k]\!]\eta)$$

若 $f : s$ 是常量符号，则$\mathcal{A}[\![f]\!]\eta = f^{\mathcal{A}}$。如果代数$\mathcal{A}$ 在上下文中是清楚的，通常省略$\mathcal{A}$而直接写$[\![M]\!]\eta$。如果 *M* 是闭项，那么$\mathcal{A}[\![M]\!]\eta$不依赖于η。此时可以省略环境，用$\mathcal{A}[\![M]\!]$表示 *M* 在代数$\mathcal{A}$ 中的含义。

例 2.3 例 2.1 的基调$\Sigma_{\mathcal{N}}$有类别 *nat* 和函数符号 0、1、+、*，可以把该基调上的项解释到 2.3.1 节的$\Sigma_{\mathcal{N}}$ 代数$\mathcal{N}$ 上。

令η是针对$\mathcal{N}$ 的一个环境，$\eta(x) = 0^{\mathcal{N}}$。$x + 1$ 在η下的含义确定如下：

$$\begin{aligned}
[\![x + 1]\!]\eta &= +^{\mathcal{N}}([\![x]\!]\eta, [\![1]\!]\eta) \\
&= +^{\mathcal{N}}(\eta(x), 1^{\mathcal{N}}) \\
&= +^{\mathcal{N}}(0^{\mathcal{N}}, 1^{\mathcal{N}}) \\
&= 1^{\mathcal{N}}
\end{aligned}$$

□

例 2.4 例 2.2 的基调Σ_{stk}=⟨*S*, *F*⟩有类别 *nat* 与 *stack*，有函数符号 0、1、…、+、*、*empty*、

push、*pop* 和 *top*。这个基调的一个代数$\mathcal{A}_{stk}$以标准的方式解释自然数，并把栈解释成自然数序列。为了描述栈上的函数，用 *n*::*s* 表示把自然数 *n* 加到序列 $s\in\mathcal{N}^*$的前面而形成的结果序列。使用这种表示法，则

$$\mathcal{A}_{stk}=\langle\, \mathcal{N}, \mathcal{N}^*, 0^{\mathcal{A}}, 1^{\mathcal{A}}, \ldots, +^{\mathcal{A}}, *^{\mathcal{A}}, empty^{\mathcal{A}}, push^{\mathcal{A}}, pop^{\mathcal{A}}, top^{\mathcal{A}} \,\rangle$$

栈函数的确定如下：

$$empty^{\mathcal{A}}=\varepsilon，\text{空序列}$$
$$push^{\mathcal{A}}(n, s)=n :: s$$
$$pop^{\mathcal{A}}(n :: s)=s$$
$$pop^{\mathcal{A}}(\varepsilon)=\varepsilon$$
$$top^{\mathcal{A}}(n :: s)=n$$
$$top^{\mathcal{A}}(\varepsilon)=0$$

需要注意的是，因为 *pop*(*empty*)是良形代数项，因此在这个代数中必须给予某种解释。这个代数中没有“*error*”元素，因而让 $pop^{\mathcal{A}}(\varepsilon)$等于空串。*top* 也有类似的情况，当栈为空时，直接返回自然数 0，这是一个随意的决定。

如果环境η把变量 *x* : *nat* 映射到自然数 3，把变量 *s* : *stack* 映射到自然数序列⟨2, 1 ⟩，其第一个元素是 2，第二个元素是 1，那么可以确定下面两个项的含义。

$$\begin{aligned}[\![pop(push\ x\ s)]\!]\eta &= pop^{\mathcal{A}}(push^{\mathcal{A}}([\![x]\!]\eta,[\![s]\!]\eta)\,)\\ &= pop^{\mathcal{A}}(push^{\mathcal{A}}(3, \langle 2, 1\rangle\,)\,)\\ &= pop^{\mathcal{A}}(\,\langle 3, 2, 1\rangle\,)\\ &= \langle 2, 1\rangle\end{aligned}$$

$$\begin{aligned}[\![top(push\ x\ s)]\!]\eta &= top^{\mathcal{A}}(push^{\mathcal{A}}([\![x]\!]\eta, [\![s]\!]\eta)\,)\\ &= top^{\mathcal{A}}(push^{\mathcal{A}}(3, \langle 2, 1\rangle\,)\,)\\ &= top^{\mathcal{A}}(\,\langle 3, 2, 1\rangle\,)\\ &= 3\end{aligned}$$

从这些例子很容易看出，对于把 *x* 映射到自然数并把 *s* 映射到栈的任何环境η，有

$$[\![pop(push\ x\ s)]\!]\eta = [\![s]\!]\eta$$
$$[\![top(\ push\ x\ s)]\!]\eta = [\![x]\!]\eta$$ □

含义函数的一个重要性质用下面的引理表达，该引理表明项含义的定义是正确的。

引理 2.2 令$\mathcal{A}$是一个Σ代数，$M\in Terms^s(\Sigma, \Gamma)$，并且$\eta$是满足Γ的环境，那么$[\![M]\!]\eta\in A^s$。

证明 这个证明根据 $Terms^s(\Sigma, \Gamma)$中项的结构进行归纳。

对于基本情况，是要证明对任何变量 $x\in Terms^s(\Sigma, \Gamma)$和满足Γ的环境$\eta$，有$[\![x]\!]\eta \in A^s$。从 $Terms^s(\Sigma, \Gamma)$的定义可知 *x* : *s* ∈ Γ。由含义的定义可知$[\![x]\!]\eta = \eta(x)$。因为η满足Γ，因此$\eta(x) \in A^s$。这就证明了归纳的基本情况。

对于归纳步骤，假定对 $1\le i \le k$，$[\![M_i]\!]\eta \in A^{s_i}$，然后考虑形式为 $f\,M_1 \ldots M_k$ 的项。根据前面有关项的含义的定义，这个项的含义是

$$[\![f\,M_1 \ldots M_k]\!]\eta = f^{\mathcal{A}}([\![M_1]\!]\eta, \ldots, [\![M_k]\!]\eta)$$

从Σ代数的定义和 $f\,M_1 \ldots M_k \in Terms^s(\Sigma, \Gamma)$这个事实，可以知道 $f^{\mathcal{A}}$是个函数，它把 $A^{s_1} \times \ldots \times A^{s_k}$ 的 *k* 元组映射到载体 A^s 上的一个元素。于是

$$f^{\mathcal{A}}([\![M_1]\!]\eta, \ldots, [\![M_k]\!]\eta) \in A^s$$

这就证明了该引理。读者应该检验在归纳步骤中当 $k = 0$ 的情况。 □

另一个直截了当的性质是，项的含义仅依赖于出现在其中的变量的值。

引理 2.3 令 $x_1, \ldots, x_k$ 是由出现在 $M \in Terms^s(\Sigma, \Gamma)$中的所有变量构成的变量表，$\eta_1$ 和η_2是满足Γ的两个环境，并且对 $i = 1, \ldots, k$ 有$\eta_1(x_i) =\eta_2(x_i)$，那么$[\![M]\!]\eta_1 = [\![M]\!]\eta_2$。

从这个引理可以知道，如果η_1 和η_2 对Γ中出现的每个变量都一致，那么对每个项 $M \in$

$Terms^s(\Sigma, \Gamma)$，都有$[\![M]\!]\eta_1 = [\![M]\!]\eta_2$，因为 M 中的每个变量必定出现在Γ中。

证明 该证明也是基于项结构的归纳。对 $x_1, \ldots, x_k$ 中的任何变量 x_i 有

$$[\![x_i]\!]\eta_1 = \eta_1(x_i) = \eta_2(x_i) = [\![x_i]\!]\eta_2$$

对任何项 $f\ M_1 \ldots M_k$，假定该引理对每个 M_i 都成立，即对所有的 i 有$[\![M_i]\!]\eta_1 = [\![M_i]\!]\eta_2$。然后证明$[\![f\ M_1 \ldots M_k]\!]\eta_1 = [\![f\ M_1 \ldots M_k]\!]\eta_2$。该证明如下：

$$\begin{aligned}[\![f\ M_1 \ldots M_k]\!]\eta_1 &= f^A([\![M_1]\!]\eta_1, \ldots, [\![M_k]\!]\eta_1)\\ &= f^A([\![M_1]\!]\eta_2, \ldots, [\![M_k]\!]\eta_2)\\ &= [\![f\ M_1 \ldots M_k]\!]\eta_2\end{aligned}$$

这就证明了该引理。 □

2.2.4 代换引理

代数、一阶逻辑和类型化λ演算都有的一个非常重要的性质叫做代换引理。直观上，就 M 的含义而言，用 N 代换项 M 中的 x 和改变环境使得 x 等于 N 的值，两者的效果是一样的。换句话说，当用项代换变量时，被代换项的含义是本质的，而它的语法形式不是。这就使得项中的子项可以用等价的项来替换。这是一个非常有用的性质，在推理中经常使用，它是代数项的等式证明系统的中心。

在论述此引理时用记号$\eta[x\mapsto a]$表示这样一个环境：除了 x 映射到 a 外，该环境在其他变量上和η一致。

引理 2.4（**代换引理**）令 $M \in Terms^s(\Sigma, \Gamma, x : s')$并且 $N \in Terms^{s'}(\Sigma, \Gamma)$，那么$[N/x]M \in Terms^s(\Sigma, \Gamma)$。并且对任何环境$\eta$，有

$$[\![[N/x]M]\!]\eta = [\![M]\!](\eta[x \mapsto a]),\ \text{其中}\ a = [\![N]\!]\eta\ \text{是}\ N\ \text{在}\ \eta\ \text{下的含义}$$

证明 根据项 M 的结构进行归纳。如果 M 是变量，分两种情况，即 M 是否就是 x。这两种情况的证明是直截了当的，把它们留给读者。

对于归纳步骤，考虑形式为 $f\ M_1 \ldots M_k$ 的项。该项的相应代换实例是 $f\ [N/x]M_1 \ldots [N/x]M_k$。从含义的定义知道

$$[\![f\ [N/x]M_1 \ldots [N/x]M_k]\!]\eta = f^A([\![[N/x]M_1]\!]\eta, \ldots, [\![[N/x]M_k]\!]\eta)$$

由归纳假设，$[\![[N/x]M_i]\!]\eta = [\![M_i]\!](\eta[x \mapsto a])$（$1 \le i \le k$）。再次使用含义的定义，可得

$$f^A([\![M_1]\!](\eta[x \mapsto a]), \ldots, [\![M_k]\!](\eta[x \mapsto a])) = [\![f\ M_1 \ldots M_k]\!](\eta[x \mapsto a])$$

于是引理得证。 □

2.3 等式、可靠性和完备性

2.3.1 等式

一个代数数据类型的公理语义由一个基调上项之间一组等式给出。一个基调和一组等式合在一起称为**代数规范**（*algebraic specification*）。从一个代数规范，可以使用等式证明系统去推导新的等式，或者调查什么样的代数满足这些等式强加的要求。这两个方面是紧密相关的：从规范推导出的等式应该在任何满足该规范的代数中都成立。这个性质叫做代数证明系统的**可靠性**（*soundness*）。反过来，在满足规范的所有代数中都成立的等式应该从该规范可以证明，这叫做**完备性**（*completeness*）。代数证明系统及其可靠性在 2.3.3 节给出，完备性在 2.3.5 节证明。

本书对代数或一阶逻辑的一般分析方法所做的一个小小推广是允许空载体。考虑空载体的原因是为了让每个规范都有初始代数（见 2.4.2 节），以及和类型化λ演算允许空类型这一点保持一致。和单类别一阶逻辑相比较时，值得一提的是，如果只有一个类别，假定该类别

非空是有意义的，否则任何逻辑公式在模型中都为真。

为什么说空载体提出一个技术问题？请回忆项 $M \in Terms^s(\Sigma, \Gamma)$的含义，它是对满足Γ的环境$\eta$定义的。如果 $x{:}s' \in \Gamma$并且 $A^{s'} = \varnothing$是空集，那么不可能有任何环境满足Γ。因而并不清楚，应该说 $Terms^s(\Sigma, \Gamma)$中的任何两个项都相等（因为所有项都无含义），还是说这样的等式没有什么意义。使得理论最简单的选择是：如果 $A^{s'}$是空集，那么涉及 $x : s'$的任何两个项都相等。本书采用按这种选择的理论。因为空载体会影响等式的真假，并且因为只有类别解释到非空载体时变量才可能给出值，因此证明系统将显式地掌握变量和它们所属类别的情况。后面将看到，掌握变量的情况将怎样影响命题 2.7 的证明。

等式是一个公式 $M = N\ [\Gamma]$，其中对某个 s，$M, N \in Terms^s\ (\Sigma, \Gamma)$。注意，一个等式有三部分：两个项和一个类型指派Γ。如果η满足Γ，那么只要$[\![M]\!]\eta = [\![N]\!]\eta$，就说代数$\mathcal{A}$在环境$\eta$下**满足** $M = N\ [\Gamma]$，写成

$$\mathcal{A}, \eta \vDash M = N\ [\Gamma]$$

如果$\mathcal{A}$在η下满足一个等式，也可以说该等式在η下**成立**。对于含变量的项，更感兴趣的是一个等式是否在变量所有取值情况下都成立，而不是在一个特别的环境下成立。如果对满足Γ的任何环境η，都有$\mathcal{A}, \eta \vDash M = N\ [\Gamma]$，就可以说$\mathcal{A}$ 满足 $M = N\ [\Gamma]$，写成

$$\mathcal{A} \vDash M = N\ [\Gamma]$$

可满足性也可以扩大到一组等式或一组代数。令$\mathcal{E}$是一组Σ等式，如果$\mathcal{A}$满足$\mathcal{E}$的所有等式，就说$\mathcal{A}$满足$\mathcal{E}$。类似地，如果$\mathcal{C}$是一类代数，并且对每个$\mathcal{A} \in \mathcal{C}$，都有$\mathcal{A} \vDash M = N\ [\Gamma]$，那么写成$\mathcal{C} \vDash M = N\ [\Gamma]$。

例 2.5 令Σ是有两个类别 a 和 b 的基调，令$\mathcal{A}$是一个Σ代数，其中 $A^a=\{0,1\}$并且 $A^b=\varnothing$。由于 A^a 有两个元素，则$\mathcal{A}$不可能满足 $x = y[x : a, y : a]$，即

$$\mathcal{A} \vDash x = y[x : a, y : a]$$

不可能成立。例如，如果一个环境的映射是 $x \mapsto 0$ 和 $y \mapsto 1$，那么该等式不成立。但是，如果在类型指派增加变量 $z : b$，那么

$$\mathcal{A} \vDash x = y[x : a, y : a, z : b]$$

成立。即该等式无意义地成立，因为不可能在 A^b 上对变量 z 作出任何指派。 □

逻辑中的一个标准概念是永真性。如果任何一个Σ代数都满足等式 $M = N\ [\Gamma]$，就说该等式是**永真的**（*valid*，也称**有效的**），并写成

$$\vDash M = N\ [\Gamma]$$

若等式 $x = x[x : s]$是永真的，因为在包含类别 s 的基调的任何一个代数中，它都成立，因此，即使 s 可以空也是这样。后面将看到，永真等式并非很有意义，能够引起兴趣的是在特定的一个代数或一类代数中成立的等式。

如果$\mathcal{A}$ 满足Σ上的所有等式，就说Σ代数$\mathcal{A}$是**平凡的**（*trivial*）。平凡代数的特点见习题 2.6。

2.3.2 项代数

关于代数项的一个有用事实是，任何基调Σ和变量类别指派Γ上的 $Terms(\Sigma, \Gamma)$给出了一个Σ代数，叫做**项代数**。在代数项的定义中，为每个类别 s 定义了一个项集 $Terms^s(\Sigma, \Gamma)$，在项代数中，就把它作为 s 的载体。剩下的是给每个函数符号 $f : s_1 \times \ldots \times s_k \to s$ 一个精确的解释。如果$\Sigma = \langle S, F \rangle$，Σ和变量类别指派Γ上项代数

$$Terms(\Sigma, \Gamma) = \langle \{Terms^s\ (\Sigma, \Gamma)\}, \mathcal{I} \rangle$$

的解释$\mathcal{I}$为

$$\mathcal{I}(f)\ (M_1, \ldots, M_k) = f\ M_1\ \ldots\ M_k$$

这个定义很简单，也很容易看出 $Terms(\Sigma, \Gamma)$是Σ代数，见习题 2.8。因此，下面 $Terms(\Sigma, \Gamma)$既用来表示项集，也用来表示项代数。

在项代数 $Terms(\Sigma, \Gamma)$中，项的含义很容易用代换描述。如果η是项代数的一个环境，那么η是从变量到项的映射，它也叫做一个代换（但可能涉及对多个变量的代换）。通常，如果 S 是一个代换，则用 SM 表示同时把 M 中的各个变量 x 用 Sx 替换的结果。因为项代数的环境η是一个代换，因此也可以用ηM 表示把代换η应用于 M 的结果。

例 2.6 令Σ是有类别 u 和函数符号 $f : u \to u$ 和 $g : u \times u \to u$ 的基调。若$\Gamma = \{a : u, b : u, c : u\}$，则在项代数$\mathcal{T} = Terms(\Sigma, \Gamma)$中，类别 u 的载体是基于变量 a、b、c 以及函数符号 f 和 g 的所有项的集合，即：

$$T^u = \{a, b, c, fa, fb, fc, gaa, gab, gac, gbb, \ldots, g\,(f\,(\,fa\,)\,)\,(f\,(gbc)\,), \ldots\}$$

在该项代数中，函数符号 f 的解释 $f^{\mathcal{T}}$ 是把任何项 M 映射到项 $f\ M$ 的函数，$g^{\mathcal{T}}$类似。如果环境η把变量 x 映射到 a，把 y 映射到 $f\,b$，那么有

$$\mathcal{T}\,[\![g(f\,x)\,(f\,y)]\!]\,\eta = g(f\,a)\,(f\,(f\,b))$$ □

下面的引理将含义和代换联系在一起。

引理 2.5 令 $M \in Terms(\Sigma, \Gamma)$，并且$\eta$是满足$\Gamma$的项代数 $Terms(\Sigma, \Gamma)$的环境，那么$[\![M]\!]\,\eta = \eta M$。

证明 对 M 进行结构归纳证明。对变量 x，由定义，有$[\![x]\!]\,\eta = \eta x$。对项 $f\,M_1 \ldots M_k$，由项在代数中含义的定义，有

$$[\![f\,M_1 \ldots M_k]\!]\,\eta = f^{Terms}([\![M_1]\!]\,\eta, \ldots, [\![M_k]\!]\,\eta)$$

由归纳假设，可以知道$[\![M_i]\!]\,\eta = \eta M_i$，并且因为 $Terms(\Sigma, \Gamma)$是一个项代数，因此

$$[\![f\,M_1 \ldots M_k]\!]\,\eta = f\,(\eta M_1) \ldots (\eta M_k)$$

它正好是代换η应用于 M 的结果。 □

项代数给出了一种简单的方法表明，只有 M 和 N 是语法上相同的项时，等式 $M = N\ [\Gamma]$才会永真。

2.3.3 语义蕴涵和等式证明系统

序对 $Spec = \langle \Sigma, \mathcal{E} \rangle$，包含一个基调$\Sigma$和一组等式$\mathcal{E}$，叫做**代数规范**。通常把代数规范 $Spec = \langle \Sigma, \mathcal{E} \rangle$看成是规定了一组代数，即一组满足$\mathcal{E}$的$\Sigma$代数。这就引出了语义蕴涵概念。

如果满足基调Σ上的一组等式$\mathcal{E}$的所有Σ代数$\mathcal{A}$都满足等式 $M = N\ [\Gamma]$，那么就说$\mathcal{E}$语义上蕴涵这个等式，写成

$$\mathcal{E} \vDash M = N\ [\Gamma]$$

很容易明白，在满足规范 $Spec = \langle \Sigma, \mathcal{E} \rangle$的所有代数中都成立的等式也就是由$\mathcal{E}$语义蕴涵的$\Sigma$等式。

如果一组等式在语义蕴涵下封闭，那么称它为一个语义理论。更准确地说，如果$\mathcal{E} \vDash M = N\ [\Gamma]$蕴涵 $M = N\ [\Gamma] \in \mathcal{E}$，那么等式集合$\mathcal{E}$ 叫做一个**语义理论**。**一个代数$\mathcal{A}$ 的理论** $Th(\mathcal{A})$是在$\mathcal{A}$中成立的所有等式的集合。很容易检查，一个代数的理论是一个语义理论。证明由习题 2.11 完成。

本节剩余部分致力于讨论语义蕴涵的代数证明系统。前面已提到，一个证明系统的两个重要性质是可靠性和完备性。用语义蕴涵来解释的话，可靠性是指，若一个等式从一组假设$\mathcal{E}$ 可证，那么$\mathcal{E}$ 语义上蕴涵该等式。完备性则相反，如果$\mathcal{E}$ 语义上蕴涵一个等式，那么该等式从$\mathcal{E}$ 可证。本节证明代数证明系统的可靠性，2.3.5 节证明完备性。

有关相等的某些性质是“泛”的，它们不依赖于代数的特别性质。具体说，语义相等是个等价关系。即自反公理的每个实例

$$M = M\ [\Gamma] \qquad (ref)$$

是永真的，并且语义相等是对称的和传递的。后两个性质可形式化为下面两条推理规则。

$$\frac{M = N\ [\Gamma]}{N = M\ [\Gamma]} \qquad (sym)$$

$$\frac{M = N\ [\Gamma] \quad N = P\ [\Gamma]}{M = P\ [\Gamma]} \qquad (trans)$$

下一条推理规则允许在等式中增加类别指派。这条规则不是很重要，但它不可少。需要这条规则的原因是，有了它就可以考虑没有在项中出现的变量。该规则是

$$\frac{M = N\ [\Gamma]}{M = N\ [\Gamma, x : s]} \quad x \text{ 不在}\Gamma\text{中} \qquad (add\ var)$$

它允许加一个变量到任何类别指派。重复使用该规则，可以加入有限个变量。很容易验证，一个代数$\mathcal{A}$满足这条规则的假设，那么它一定满足其结论。

最后一条规则叫做**等价代换**。如果不提及类别指派的话，该代换规则是说，如果$M = N$并且$P = Q$，那么用P和Q分别代换M和N中的x，所得的两个结果相等。该规则形式表达如下：

$$\frac{M = N\ [\Gamma, x : s] \quad P = Q\ [\Gamma]}{[P/x]M = [Q/x]N\ [\Gamma]} \quad P, Q \in Terms^{s}(\Sigma, \Gamma) \qquad (subst)$$

该规则副条件给出的限制是说，不能代换含x的P和Q到等式$M = N\ [\Gamma, x : s\ ]$中，因为$P = Q\ [\Gamma]$的类别指派Γ假定为不含x。但是从习题 2.13 可以看出，实际上这并不是一个问题。

如果从$\mathcal{E}$中的等式和公理(*ref*)及推理规则 (*sym*)、(*trans*)、(*subst*) 和(*add var*) 可以推出等式$M = N[\Gamma]$，那么就说该等式**可证**，并写成

$$\mathcal{E} \vdash M = N\ [\Gamma]$$

更形式地说，从$\mathcal{E}$到E的证明是一个等式序列，其中每个等式是公理、$\mathcal{E}$中的等式、或者是序列中早先出现的一个或多个等式的一步推导的结果，序列的最后一个等式是E。有关证明的一种有用的推理形式是基于E的证明长度的归纳。

如果$\mathcal{E}$封闭于可证明性，那么就说$\mathcal{E}$是一个**语法理论**。换种说法，如果$\mathcal{E} \vdash M = N\ [\Gamma]$蕴涵$M = N\ [\Gamma] \in \mathcal{E}$，那么$\mathcal{E}$是一个语法理论。$\mathcal{E}$的语法理论$Th(\mathcal{E})$也就是从$\mathcal{E}$可证的所有等式的集合。通过下面关于可靠性和完备性的证明，可以得知语法上的代数理论和语义上的代数理论是相同的。但是，目前仍保持两个定义是有用的。等式集合$\mathcal{E}$是**语义一致的**(*consistent*)，如果存在某个等式$M = N\ [\Gamma]$，它不是$\mathcal{E}$的语义蕴涵；等式集合$\mathcal{E}$是**语法一致的**，如果存在某个等式$M = N\ [\Gamma]$，它不能由$\mathcal{E}$证明，否则称为**不一致**的。

例 2.7 （例 2.2 和 2.4 的继续）在基调$\Sigma_{stk} = \langle S, F\rangle$上增加等式

$$top\ (push\ x\ s\ ) = x\ [s : stack, x : nat]$$

$$pop\ (push\ x\ s\ ) = s\ [s : stack, x : nat]$$

使用这些等式可以证明等式

$$top\ (push\ 3\ empty\ ) = 3$$

其证明如下：

$$\frac{\dfrac{top\ (push\ x\ s) = x\ [s{:}stack, x{:}nat] \qquad empty = empty\ [x{:}nat]}{top\ (push\ x\ empty) = x\ [x{:}nat] \qquad\qquad 3 = 3\ [\,]}}{top\ (push\ 3\ empty) = 3\ [\,]}$$

该证明两次使用 (*ref*)公理，一次栈公理和两次(*subst*)。 □

一个证明系统的**导出规则**是一条推理规则

$$\frac{\text{前提}}{\text{结论}}$$

它满足：从该前提的任何实例，通过使用该证明系统的公理和其他证明规则，可以推导出该结论的相应实例。例如，

$$\frac{M = N\ [\Gamma] \quad N = P\ [\Gamma] \quad P = Q\ [\Gamma]}{M = Q\ [\Gamma]}$$

是该代数证明系统的一个导出规则，因为从形式为 $M = N, N = P, P = Q$ 的任意三个等式，两次使用传递规则可以得到 $M = Q$。

导出规则的另一个例子是下面的同余规则，直观上说，同余是指“相等的项应用于相等的项时，产生相等的项”。

$$\frac{M_1 = N_1\ [\Gamma] \ \ldots\ M_k = N_k\ [\Gamma] \qquad f : s_1 \times \ldots \times s_k \to s \text{ and } M_i, N_i \in Terms^{s_i}(\Sigma, \Gamma)}{fM_1 \ldots M_k = fN_1 \ldots N_k\ [\Gamma]} \qquad (cong)$$

(*cong*)规则可从(*add var*), (*ref*)和(*subst*)推导。从这些假设，使用(*add var*)，可以推导出 k 个等式：

$M_i = N_i[\Gamma, x_1 : s_1, \ldots, x_{i-1} : s_{i-1}]$ （$i = 1, \ldots, k$）

其中 $x_1, \ldots, x_k$ 是不出现在Γ中的新变量（$i = 1$ 时的等式就是 $M_1 = N_1\ [\Gamma]$）。再由(*ref*)可得

$f\, x_1 \ldots x_k = f\, x_1 \ldots x_k\ \ [\Gamma, x_1 : s_1, \ldots, x_k : s_k]$

重复使用(*subst*)，第一步是先将上式两边的 x_k 分别用 M_k 和 N_k 代换，最后一步将倒数第二步代换结果中两边的 x_1 分别用 M_1 和 N_1 代换。最终得到

$f\, M_1 \ldots M_k = f\, N_1 \ldots N_k\ [\Gamma]$

例 2.8 证明等式 *top*(*pop*(*push x*(*push* 3 *empty*))) = 3 [*x* : *nat*]

为证明该等式，从栈公理和(*ref*)的实例开始，应用(*subst*)，然后再用导出规则(*cong*)得到

$$\frac{pop\ (push\ x\ s) = s\ [s{:}stack, x{:}nat] \qquad push\ 3\ empty = push\ 3\ empty\ [x{:}nat]}{pop\ (push\ x\ (push\ 3\ empty)) = push\ 3\ empty\ [x{:}nat]}$$
$$\frac{}{top\ (pop\ (push\ x\ (push\ 3\ empty))) = top\ (push\ 3\ empty)\ [x{:}nat]}$$

再从上例的结果用(*add var*)得到

top(*push* 3 *empty*) = 3 [*x* : *nat*]

最后用(*trans*)得到

top(*pop* (*push x* (*push* 3 *empty*))) = 3 [*x* : *nat*] □

例 2.9 下面的规则允许从任何等式的变量类别指派中删除多余变量。如果变量 x 在 M 和 N 中都不出现，并且从其他信息知道 s 类别的项集 $Terms^s(\Sigma, \Gamma)$非空，那么在等式 $M = N$ [Γ, $x : s$]中，$x : s$ 没有什么作用，可以把它从[Γ, $x : s$]中删除，即推理规则

$$\frac{M = N\ [\Gamma, x{:}s]}{M = N\ [\Gamma]} \qquad M \text{ 和 } N \text{ 中没有 } x，\ Terms^s(\Sigma, \Gamma) \text{非空}$$

有意义。

很容易证明这是一个导出规则。因为 $Terms^s(\Sigma, \Gamma)$非空，则存在某个 $P \in Terms^s(\Sigma, \Gamma)$，使得 $P = P$ [Γ]。结合假设 $M = N$ [Γ, $x : s$]，用(*subst*)可以证明$[P/x]M = [P/x]N$ [Γ]。因为变量 x 在 M 和 N 中不出现，代换没有影响，因此 $M = N$ [Γ]。 □

本节主要定理如下：

定理 2.6 （**可靠性**）如果$\mathcal{E} \vdash E$，那么$\mathcal{E} \models E$。

证明 可以根据该证明的长度进行归纳，即根据$\mathcal{E} \vdash E$ 的证明长度来证明$\mathcal{E} \models E$。

归纳基础是长度为 1 的证明，它是公理或$\mathcal{E}$ 的一个等式。不管哪一种情况，很容易看出，任何满足$\mathcal{E}$ 的代数一定满足该等式。

对于归纳步骤，假定证明 E 的最后一步是从 $E_1, \ldots, E_n$ 得到，并且 $E_1, \ldots, E_n$ 是由更短的证明得到。从归纳假设可以假定，如果$\mathcal{A} \models \mathcal{E}$，那么$\mathcal{A}$ 满足 $E_1, \ldots, E_n$。要证明的是，如果$\mathcal{A}$ 满足最后一条规则的这些前提，那么$\mathcal{A}$ 满足 E。这就导致根据证明规则的集合，对各种情况进行分析。下面仅给出对代换规则的证明。

假定$\mathcal{A} \models M = N$ [Γ, $x : s$]，并且$\mathcal{A} \models P = Q$ [Γ]。必须证明$\mathcal{A} \models [P/x]M = [Q/x]N$ [Γ]。令η是满足Γ的任意一个环境，必须证明$[\![[P/x]M]\!]\eta = [\![[Q/x]N]\!]\eta$。

令 $a = [\![P]\!]\eta = [\![Q]\!]\eta$，并且注意$\eta[x \mapsto a]$满足Γ, $x : s$。由代换引理：

$$[\![[P/x]M]\!]\eta = [\![M]\!](\eta[x \mapsto a])$$

并且

$$[\![[Q/x]N]\!]\eta = [\![N]\!](\eta[x \mapsto a])$$

因为$\mathcal{A} \models M = N$ [Γ, $x : s$]，因此

$$[\![M]\!](\eta[x \mapsto a]) = [\![N]\!](\eta[x \mapsto a])$$

从而$[\![[P/x]M]\!]\eta$和$[\![[Q/x]N]\!]\eta$相等。这就完成了本定理的证明。 □

使用这个定理，通过下面的命题可以说明为什么在等式中掌握变量的情况是重要的。

命题 2.7 存在一个代数理论$\mathcal{E}$ 和不含 x 的项 M 和 N，使得$\mathcal{E} \vdash M{=}N$ [Γ, $x : s$]，但是$\mathcal{E} \nvdash M{=}N$ [Γ]。

证明 令Σ是一个基调，它有类别 a 和 b，函数符号 $f : a \to b$ 和 $c, d : b$。令$\mathcal{E}$是包含能从 $f\,x = c$ [$x : a$]和 $f\,x = d$ [$x : a$]证明的所有等式，并且考虑等式 $c = d$ [$x : a$]。很清楚，从$\mathcal{E}$ 的等式通过传递性可以得到它。但是从下面的语义讨论可以看到，等式 $c = d$[∅]不可证。

考虑一个Σ代数，其对应 a 的载体是空集，但是 c 和 d 分别指称对应 b 的非空载体上的两个不同元素。等式 $f\,x = c$ [$x : a$]和 $f\,x = d$ [$x : a$]在这个模型中为真，因为 x 没有任何可能的值。然而，等式 $c = d$ [∅]在此模型中不成立。因此，由等式证明系统的可靠性，$c = d$ [∅]是不可能从$\mathcal{E}$ 证明的。 □

可靠性定理不仅表明证明系统是语义正确的，而且可以知道，如果能找出一个代数满足$\mathcal{E}$ 而不满足某个等式 E，即$\mathcal{E}$ 从语义上并不蕴涵 E，那么从$\mathcal{E}$ 不可能证明 E。在不知等式 E 是否由等式集合$\mathcal{E}$ 语义蕴涵时，可以作两种尝试：从$\mathcal{E}$ 证明 E，或者找出一个满足$\mathcal{E}$ 而不满足 E 的代数。根据完备性定理，原则上这两者之间总有一个是可能的。但是应该认识到，在寻找一个证明和寻找一个代数以表明不存在这样的证明这两者之间有一个重要区别。它就是，存在一种有效的方法，它能枚举所有的证明，因而如果 E 从$\mathcal{E}$ 可证，那么最终会找到一个证明；但是不存在这样的方法，用它可以找到一个代数以表明不存在这样的证明。

例 2.10 栈代数规范如表 2.1。满足这个规范的一个代数$\mathcal{A}_{stk}$在例 2.4 已经给出。

表 2.1 栈代数规范

sorts :	*nat*, *stack*
fctns :	0, 1, 2, … : *nat*
	+, ∗ : *nat* × *nat* → *nat*
	empty : *stack*

$push : nat \times stack \to stack$

$pop : stack \to stack$

$top : stack \to nat$

eqns : $[s : stack; x : nat]$

$0 + 0 = 0, \quad 0 + 1 = 1, \ldots$

$0 * 0 = 0, \quad 0 * 1 = 0, \ldots$

$top\ (push\ x\ s\ ) = x$

$pop\ (push\ x\ s\ ) = s$

很容易看出，等式

$$push\ (top\ s)\ (pop\ s) = s\ [s : stack]$$

是不可能从这个规范证明的，只要把 s 映射到空串就可以看出该等式在$\mathcal{A}_{stk}$中不成立。一个更加复杂的事实是，任何形式为

$$top\ empty = M\ [\Gamma]$$

的等式都不可证，假定 M 是类别为 *nat* 的项，并且不含 *empty*（包括 M 推导出的项也不含 *empty*）。该事实的证明要点如下：

（1）对任何 $n \in \mathcal{N}$，定义代数$\mathcal{A}_n$，它们和$\mathcal{A}_{stk}$几乎一样，区别仅是$top^{\mathcal{A}_n}(\varepsilon) = n$（那么，$\mathcal{A}_{stk} = \mathcal{A}_0$）。显然，它们都是该规范的代数。

（2）对类别 *nat* 任何不含 *empty* 的项 M，总能找到一个环境η（把 *stack* 类别的变量映射到足够长的串），使得对所有的 i 和 j，$\mathcal{A}_i[\![M]\!]\eta = \mathcal{A}_j[\![M]\!]\eta$。

（3）对类别 *nat* 任何不含 *empty* 的项 M，令η是使得对所有的 n，$\mathcal{A}_n[\![M]\!]\eta = \mathcal{A}_{stk}[\![M]\!]\eta$的环境。由于$\mathcal{A}_n[\![M]\!]\eta$独立于 n，而$\mathcal{A}_n[\![top\ empty]\!]\eta$依赖于 n，因此肯定存在某个 n，使得$\mathcal{A}_n[\![top\ empty]\!]\eta \neq \mathcal{A}_n[\![M]\!]\eta$。于是 $top\ empty = M[\Gamma]$是不可能从栈公理证明的。 □

2.3.4 完备性的形式

在研究完备性证明的技术前，首先讨论可以用于不同逻辑系统的三种不同的完备性。

（1）最弱的形式是，仅所有的永真公式可证。对于代数来说，永真公式是（*ref*）公理的所有实例。这种完备性太弱，以至没有什么意义。

（2）处于中间的形式叫做**演绎完备性**（*deductive completeness*），它是指每个语义蕴涵都可证。对代数而言，这是指每当$\mathcal{E} \vDash M = N[\Gamma]$，就有$\mathcal{E} \vdash M = N\ [\Gamma]$。

（3）更强形式的完备性叫做**最小模型完备性**（*least model completeness*），它是指对每个语法理论，都有某个“最小”模型，其语义理论等于该语法理论。对代数来说，最小模型完备性意味着对每个语法理论，都存在某个代数$\mathcal{A}$，$Th(\mathcal{A})$等于该语法理论。可能会引起混淆的地方是，“最小模型”是指它的理论包含的内容最少，而不是指它们载体的规模最小。

通常，当逻辑公式可以表示“分离”的信息时，最小模型完备性不成立。空载体引入了一种形式的分离。

考虑等式

$$E \triangleq x = y\ [x : a, y : a, z : b\ ]$$

只有两种模型可使 E 满足。一种模型是 a 的载体为空或只含一个元素，只含一个元素的目的是迫使 x 和 y 只能取同样的值。在这种模型中，$E_1 \triangleq x = y\ [x : a, y : a]$成立。另一种模型是 b 的载体为空，它使得 E 无意义地成立。在这种模型中，$E_2 \triangleq z = w\ [z : b, w : b]$成立。显然，$E_1$ 和 E_2 都不是从 E 可证的，所以不存在一个代数，其理论正好就是由 E 的等式推论组成的语法理论，这就意味着最小模型不存在。

2.3.5 同余、商和演绎完备性

本节主要结论是可以有空载体的多类代数的演绎完备性定理。该定理的证明要用到同余关系和商代数，因此首先研究它们。另外，由于本书主要讨论演绎完备性，因此下面简称完备性。

同余关系是指等价关系加上同余性，而同余性则是指函数保可证的相等性。

对于一个单类代数$\mathcal{A} = \langle A, f_1^{\mathcal{A}}, f_2^{\mathcal{A}}, \ldots \rangle$，同余关系是载体 A 上的一个等价关系，使得对每个 k 元函数 $f^{\mathcal{A}}$，如果 $a_i \sim b_i$ $(i = 1, \ldots, k)$，即 a_i 和 b_i 等价，那么 $f^{\mathcal{A}}(a_1, \ldots, a_k) \sim f^{\mathcal{A}}(b_1, \ldots, b_k)$。

对于多类Σ代数$\mathcal{A} = \langle \{A^s\}, \mathcal{I} \rangle$，同余关系是一簇等价关系$\sim = \{\sim_s\}$, $\sim_s \subseteq A^s \times A^s$，每个类别一个，使得对每个 $f : s_1 \times \ldots \times s_k \to s$ 及变元序列 $a_1, \ldots, a_k$ 和 $b_1, \ldots, b_k$（其中 $a_i \sim_{s_i} b_i \in A^{s_i}$），有 $f^{\mathcal{A}}(a_1, \ldots, a_k) \sim_s f^{\mathcal{A}}(b_1, \ldots, b_k)$。

给定$\mathcal{A}$上的任意同余关系~，可以构造叫做**$\mathcal{A}$模~的商**的代数$\mathcal{A}/\sim$。$\mathcal{A}/\sim$直观的含义是把$\mathcal{A}$中有关系的元素 $a \sim a'$ 压缩成$\mathcal{A}/\sim$的一个元素。

$a \in A^s$ 的等价类$[a]$定义为

$$[a]_\sim = \{a' \in A^s \mid a \sim a'\}$$

等价类的另一种记号是 $a/\sim$。

Σ商代数$\mathcal{A}/\sim$定义如下：

（1）$(\mathcal{A}/\sim)^s$ 是由 A^s 的所有等价类构成的集合

$$A^s/\sim_s = \{[a]_{\sim_s} \mid a \in A^s\}$$

（2）函数 $f^{\mathcal{A}/\sim}$由$\mathcal{A}$ 的函数 $f^{\mathcal{A}}$确定。对适当载体的 $a_1, \ldots, a_k$，

$$f^{\mathcal{A}/\sim}([a_1], \ldots, [a_k]) = [f^{\mathcal{A}}(a_1, \ldots, a_k)]$$

这个定义是否有意义并不完全明显。有意义的条件是 $f^{\mathcal{A}/\sim}([a_1], \ldots, [a_k])$必须只依赖于$[a_1], \ldots, [a_k]$，而不能依赖于所选的代表 $a_1, \ldots, a_k$。这个问题作为习题。

例如，对于单类别代数$\langle \mathcal{N}, 0, 1, + \rangle$上的同余关系“模 k 等价”，很容易看出，这个商代数是大家熟悉的整数模 k 加结构。如果 k 等于 5，那么$[3] + [4] = [2]$。

项 M 在商代数$\mathcal{A}/\sim$中的含义取决于 M 在$\mathcal{A}$中的含义。若 M 不含变量，则 M 在商代数$\mathcal{A}/\sim$中的含义是 M 在$\mathcal{A}$ 中的含义所在的等价类。若 M 含变量，则含义当然还和环境有关。

如果η是$\mathcal{A}$的一个环境，~是一个同余关系，那么$\mathcal{A}/\sim$的环境$\eta_\sim$ 定义如下：

$$\eta_\sim(x) = [\eta(x)]_\sim$$

反过来，对于$\mathcal{A}/\sim$的环境η'，对应它的$\mathcal{A}$ 的环境η有多种选择。对于变量 x，任意选择$\eta(x) \in \eta'(x)$，那么$\eta_\sim = \eta'$。使用$\mathcal{A}$ 和$\mathcal{A}/\sim$的这种对应，描述商代数中项的含义的引理如下：

引理 2.8 令~是Σ代数$\mathcal{A}$上的一个同余关系，项$M \in Terms(\Sigma, \Gamma)$并且$\eta$是一个满足Γ的环境。那么项 M 在商代数$\mathcal{A}/\sim$ 和环境$\eta_\sim$下的含义$(\mathcal{A}/\sim)[\![M]\!]\eta_\sim$由

$$(\mathcal{A}/\sim)[\![M]\!]\eta_\sim = [\mathcal{A}[\![M]\!]\eta]_\sim$$

给出。

证明 基于 M 的结构进行归纳。归纳基础很容易从$\eta_\sim$ 的定义得出。对于归纳步骤，有

$$\begin{aligned}(\mathcal{A}/\sim)[\![f\, M_1 \ldots M_k]\!]\eta_\sim &= f^{\mathcal{A}/\sim}((\mathcal{A}/\sim)[\![M_1]\!]\eta_\sim, \ldots, (\mathcal{A}/\sim)[\![M_k]\!]\eta_\sim) \\ &= f^{\mathcal{A}/\sim}([\mathcal{A}[\![M_1]\!]\eta], \ldots, [\mathcal{A}[\![M_k]\!]\eta]) \\ &= [f^{\mathcal{A}}(\mathcal{A}[\![M_1]\!]\eta, \ldots, \mathcal{A}[\![M_k]\!]\eta)] \\ &= [\mathcal{A}[\![f\, M_1, \ldots, M_k]\!]\eta]\end{aligned}$$

□

证明完备性定理前的最后一步是：任何一组等式确定项代数上的一个同余关系。先定义

项集 *Terms*(Σ, Γ)上一个关系如下：

$$M \sim_{\mathcal{E},\Gamma} N \text{ 当且仅当 } \mathcal{E} \vdash M = N\ [\Gamma]$$

引理 2.9 令$\mathcal{E}$ 是一组Σ等式集合，令 *Terms*(Σ, Γ)是基调Σ上的项集。由$\mathcal{E}$ 的可证明性确定的关系$\sim_{\mathcal{E},\Gamma}$是 *Terms*(Σ, Γ)上的一个同余关系。

证明 很容易看出，在各个类别上，$\sim_{\mathcal{E},\Gamma}$是一个等价关系，因为有公理(*ref*)以及推理规则(*sym*)和(*trans*)。为了简化记号，用$\mathcal{T}$ 表示项代数 *Terms*(Σ, Γ)。很容易从导出规则(*cong*)得出，如果 $M_i \sim_{\mathcal{E},\Gamma} N_i\ (1 \le i \le k)$ ，那么

$$f^{\mathcal{T}}(M_1, \ldots, M_k) \sim_{\mathcal{E},\Gamma} f^{\mathcal{T}}(N_1, \ldots, N_k)$$ □

定理 2.10 （**完备性**）令$\mathcal{E}$ 是一个Σ等式的集合，并且 *E* 是一个Σ等式。如果$\mathcal{E} \models E$，那么$\mathcal{E} \vdash E$。

证明 用反证法。假定等式 $M_0 = N_0\ [\Gamma_0]$不能从$\mathcal{E}$ 证明，若能找到一个代数满足$\mathcal{E}$，但是不满足该等式，则完备性得证。所找的代数就是商代数$\mathcal{A} = Terms(\Sigma, \Gamma_0)/\sim_{\epsilon,\Gamma_0}$ ，因而要证明$\mathcal{A}$满足$\mathcal{E}$ 的所有等式，但不满足 $M_0 = N_0\ [\Gamma_0]$。

为简单起见，用~和$\mathcal{T}$ 分别表示$\sim_{\epsilon,\Gamma_0}$ 和项代数 *Terms*(Σ, Γ$_0$)。

要想证明$\mathcal{A}$ 不满足 $M_0 = N_0\ [\Gamma_0]$，只要能找到一个环境使得 M_0 和 N_0 在该环境下的解释不相同就可以了。令η是$\mathcal{T}$ 的一个环境，它把变量都映射到自己。由引理 2.5，$\mathcal{T}[\![M_0]\!]\eta$就是 M_0。由引理 2.8

$$\mathcal{A}[\![M_0]\!]\eta_{\sim} = [M_0]$$

对 N_0 也一样。由假设知道$[M_0] \ne [N_0]$，因此在环境$\eta_{\sim}$中，$M_0 = N_0\ [\Gamma_0]$不成立。

剩下要证明$\mathcal{A} \models \mathcal{E}$。假定 $M = N\ [\Gamma] \in \mathcal{E}$。由引理 2.8 知道，对项代数$\mathcal{T}$ 的一个环境η，$\mathcal{A}$ 的环境可以写成$\eta_{\sim}$。具体来说，假设给定$\mathcal{A}$ 的某个环境 $\hat{\eta}$，那么对每个变量 *x*，选择某个 $P \in \hat{\eta}(x)$ 并且让$\eta(x) = P$，这样就有 $\eta_{\sim} = \hat{\eta}$。因此，只要证明对每个满足Γ的$\mathcal{T}$ 的环境η，$\mathcal{A}[\![M]\!]\eta_{\sim} = \mathcal{A}[\![N]\!]\eta_{\sim}$就可以了。

对满足Γ的任何$\mathcal{T}$ 的环境η，由引理 2.8，$\mathcal{A}[\![M]\!]\eta_{\sim} = [\mathcal{T}[\![M]\!]\eta]_{\sim}$。再由引理 2.5 得

$$\mathcal{A}[\![M]\!]\eta_{\sim} = [\mathcal{T}[\![M]\!]\eta]_{\sim} = [\eta M]_{\sim}$$

同样，$\mathcal{A}[\![N]\!]\eta_{\sim} = [\eta N]_{\sim}$。使用规则(*ref*)和(*subst*)，可以证明$\mathcal{E} \vdash \eta M = \eta N\ [\Gamma]$（见习题 2.14）。它意味着$[\eta M]_{\sim} = [\eta N]_{\sim}$，从而 *M* 和 *N* 在$\mathcal{A}$中有同样的含义。这就证明了定理。 □

完备性定理加上先前的可靠性定理表明，语法理论和语义理论是相同的。

2.3.6 非空类别和最小模型完备性

先精确定义什么是非空类别。一个类别 s 是Σ**非空的**，如果基调Σ包含类别 *s* 的一个常量符号，或者有函数符号 $f : s_1 \times \ldots \times s_k \to s$ 并且类别 $s_1, \ldots, s_k$ 都是Σ非空的。

如果 *s* 是Σ非空的，那么至少有一个项 $M \in Terms^s(\Sigma, \varnothing)$，从而在任何Σ代数中 *s* 的载体不可能为空，因为至少能命名 *s* 的一个元素 $P \in Terms^s(\Sigma, \Gamma)$，并且 *P* 肯定能解释为 *s* 载体上的一个元素。

没有空载体的代数有最小模型完备性。删除空载体有两种办法：

（1）假定没有代数有任何空载体，并加上一个删除多余变量的推理规则到 2.3.3 节的证明系统。这种方式是将讨论范围限制到没有任何空载体的代数，而不管基调中是否有空类别。

（2）基调中每个类别都有不含变量的项。这种方式由所有类别都非空来要求所有载体都非空。

在这两种情况下，在证明定理 2.10 时用的构造本质上同样可以用于证明最小模型完备性。

覆盖这两种非空情况的一个完备性定理的陈述要用到下面的推理规则：

$$\frac{M = N\,[\Gamma, x:s]}{M = N\,[\Gamma]} \qquad x \text{ 不在 } M \text{ 和 } N \text{ 中} \qquad (\textit{nonempty})$$

该规则允许将等式中不出现的变量的类别指派在环境中删除。如果知道 s 的载体非空，那么很容易看出该规则可靠。具体说，如果作为前提的等式在某个代数$\mathcal{A}$ 中得到满足，并且η是满足$\Gamma, x : s$ 的一个环境，那么存在满足Γ的一个环境η'，对于Γ中的所有变量，η'和η是一样的。由引理 2.3，项 M 和 N 在η'下有同样的含义，就像在η下那样，所以该规则可靠。

下面的定理是通过该规则来给出最小模型完备性。

定理 2.11 令$\mathcal{E}$ 是封闭于规则(*nonempty*)的语法理论，那么存在所有载体都非空的代数$\mathcal{A}$，使得$\mathcal{E} = Th\,(\mathcal{A})$。

这里封闭于规则(*nonempty*)是指，任何能用规则(*nonempty*)推出的等式都在$\mathcal{E}$ 中。

证明 假定$\mathcal{E}$ 是封闭于规则(*nonempty*)的语法理论。为了避免有关变量名字的技术细节，假定有形式为 $x : s$ 的序对的无穷集合Γ_∞，并且Σ的每个类别都有无穷多个变量。假定所关心的项都属于某个 $Terms(\Sigma, \Gamma)$，$\Gamma \subseteq \Gamma_\infty$。这并不失一般性，因为$\Gamma_\infty$有足够多的变量，重新命名一个变量不会影响一个代数是否满足一个等式。很容易看出，项代数 $Terms(\Sigma, \Gamma)$的定义不依赖于Γ是否有限，因而可以令$\mathcal{T}$ 是项代数 $Terms(\Sigma, \Gamma_\infty)$。令~是$\mathcal{T}$ 上的一个二元关系，使得对某个有穷的$\Gamma \subseteq \Gamma_\infty$，$M$~$N$ 当且仅当$\mathcal{E} \vdash M = N\,[\Gamma]$。很容易检查，~是$\mathcal{T}$ 上的一个同余关系。剩下要证明的是，对商代数$\mathcal{A} = \mathcal{T}/\sim$有$\mathcal{E} = Th\,(\mathcal{A})$。证明的步骤本质上同于定理 2.10 的证明步骤。 □

在所有类别都非空（从而所有载体也都非空）的情况下，可以用这个定理来证明 2.3.3 节没有规则(*nonempty*)的证明系统的最小模型完备性。因为当所有的类别都非空时，可以像例 2.9 中描述的那样，由代换规则删除等式中多余变量，因此(*nonempty*)成为一个导出规则。

推论 2.12 如果Σ是每个类别都Σ非空的基调，那么对Σ每个语法理论$\mathcal{E}$，存在一个代数$\mathcal{A}$，使得$\mathcal{E} = Th\,(\mathcal{A})$。

2.4 同态和初始性

2.4.1 同态和同构

同态是从一个代数到另一个代数的保结构的映射，本节将同态和同构的概念从单类代数推广到多类代数，并用同态来定义初始代数。

对多类代数来说，从Σ代数$\mathcal{A}$ 到$\mathcal{B}$ 的同态 $h : \mathcal{A} \to \mathcal{B}$ 是一簇映射 $h = \{h^s \mid s \in S\}$，使得对Σ的每个函数符号 $f : s_1 \times \ldots \times s_k \to s$，有

$$h^s(f^A(a_1, \ldots, a_k)) = f^B(h^{s_1}a_1, \ldots, h^{s_k}a_k)$$

直观上，同态看成是从$\mathcal{A}$ 的值到$\mathcal{B}$ 的值的“翻译”。

例 2.11 令$\mathcal{N} = \langle \mathcal{N}, 0, 1, + \rangle$，令~是模 k 的等价关系。那么把 $n \in \mathcal{N}$映射到它的等价类$[n]_\sim$是从$\mathcal{N}$ 到$\mathcal{N}/\sim$的一个同态，因为

$$h\,(0) = 0^{\mathcal{N}/\sim} = [0]_\sim$$

$$h\,(n + m) = h\,(n) +^{\mathcal{N}/\sim} h\,(m) = [n + m]_\sim$$

通常，任何代数到它商代数的同态都用这种方式定义。 □

例 2.12 含义函数是从项代数$\mathcal{T} = Terms(\Sigma, \Gamma)$到任意代数$\mathcal{A}$ 的一个同态 $h : \mathcal{T} \to \mathcal{A}$。如果$\eta$是$\mathcal{A}$ 的一个满足Γ的环境，该同态的定义是

$$h(M) = \mathcal{A}[\![M]\!]\eta$$

这是一个同态，因为从项的含义可知

$$h(f\,M_1 \ldots M_k) = \mathcal{A}[\![f\,M_1 \ldots M_k]\!]\eta = f^{\mathcal{A}}(\mathcal{A}[\![M_1]\!]\eta, \ldots, \mathcal{A}[\![M_k]\!]\eta)$$
$$= f^{\mathcal{A}}(h(M_1), \ldots, h(M_k))$$

还可以定义从商代数 $Terms(\Sigma, \Gamma)/\sim_{\mathcal{E},\Sigma}$到满足$\mathcal{E}$ 的代数的同态。 □

为了精确论述同态和含义间的联系，需要环境之间的一个“翻译”。如果 $h : \mathcal{A} \to \mathcal{B}$，并且$\eta$是$\mathcal{A}$ 的一个环境，那么定义$\mathcal{B}$ 的环境η^h 为：

$$\eta^h(x) = h(\eta(x)) \qquad \text{对任何变量 } x$$

如果η满足某个类别指派Γ，那么η^h 也满足。

引理 2.13 令 $h : \mathcal{A} \to \mathcal{B}$ 是任意同态，并且η是满足类别指派Γ的任意$\mathcal{A}$ 环境。那么对任何项 $M \in Terms(\Sigma, \Gamma)$，有

$$h(\mathcal{A}[\![M]\!]\eta) = \mathcal{B}[\![M]\!]\eta^h$$

当 M 中不含变量时，$h(\mathcal{A}[\![M]\!]) = \mathcal{B}[\![M]\!]$

证明 该证明直截了当基于项的归纳。当项是变量时，直接从η^h 的定义可得。对于复合项 $f\,M_1 \ldots M_k$，从归纳假设 $h(\mathcal{A}[\![M_i]\!]\eta) = \mathcal{B}[\![M_i]\!]\eta^h$，由同态的定义得

$$h(\mathcal{A}[\![f\,M_1 \ldots M_k]\!]\eta) = f^{\mathcal{B}}(h(\mathcal{A}[\![M_1]\!]\eta), \ldots, h(\mathcal{A}[\![M_k]\!]\eta))$$
$$= f^{\mathcal{B}}(\mathcal{B}[\![M_1]\!]\eta^h, \ldots, \mathcal{B}[\![M_k]\!]\eta^k)$$

这就证明了该引理。 □

如果 $h : \mathcal{A} \to \mathcal{B}$ 和 $k : \mathcal{B} \to \mathcal{C}$ 都是Σ代数的同态，那么它们的合成 $k \circ h : \mathcal{A} \to \mathcal{C}$ 是通过合成各类别映射而得到的一族映射$(k \circ h)^s = k^s \circ h^s$。

引理 2.14 如果 $h : \mathcal{A} \to \mathcal{B}$ 和 $k : \mathcal{B} \to \mathcal{C}$ 都是Σ代数的同态，那么合成 $k \circ h : \mathcal{A} \to \mathcal{C}$ 也是Σ代数的同态。

这个证明作为一个习题。

一个双射（单射并且满射）的同态叫做**同构**。若从$\mathcal{A}$ 到$\mathcal{B}$ 有一个同构，就说$\mathcal{A}$ 和$\mathcal{B}$是同构的，写成$\mathcal{A} \cong \mathcal{B}$。直观上说，同构就是重新命名元素而不改变代数结构，因而它们本质上是相同的。

2.4.2 初始代数

初始代数在研究代数数据类型时是重要的，因为它们通常和代数规范“预期”的或“标准”的实现一致。如果$\mathcal{C}$ 是一类Σ代数并且$\mathcal{A} \in \mathcal{C}$，若对每个$\mathcal{B} \in \mathcal{C}$，都存在唯一的同态 $h : \mathcal{A} \to \mathcal{B}$，那么$\mathcal{A}$ 在$\mathcal{C}$ 中叫做**初始代数**（*initial algebra*）。如果把同态 $h : \mathcal{A} \to \mathcal{B}$ 看成是从$\mathcal{A}$ 到$\mathcal{B}$ 的一个翻译，那么初始代数是“典型”的，因为可以把初始代数“翻译”到该类中的所有其他代数。初始代数有尽可能少的非空载体，即其他代数的空载体个数不会多于初始代数的空载体个数，因为初始代数的每一个元素必须映射到其他代数（经过某个同态）的一个元素。再具体一点，不存在从非空集合到空集的函数，但是从空集到任何集合都有唯一的函数，把函数看成序对集合的话，该函数是一个空集。另外，初始代数满足尽可能少的闭等式（由引理 2.13），闭等式指由闭项构成的等式。本节的主要结论是，闭项代数的商是初始的（命题 2.17），另外还描述了初始代数的等式理论（命题 2.18）。

例 2.13 考虑基调Σ_0，它有一个类别 *nat*，有函数符号 $0 : nat$ 和 $S : nat \to nat$。令$\mathcal{C}$ 是所有Σ_0 代数构成的代数类。闭项代数$\mathcal{T} = Terms(\Sigma_0, \varnothing)$是$\mathcal{C}$ 的初始代数，它的载体是所有闭项 $0, S(0), S(S(0)), \ldots, S^k(0), \ldots$。该代数的函数 S 把 $S^k(0)$映射到 $S^{k+1}(0)$。该代数的元素少到能解释所有的函数符号，并且该代数满足项之间尽可能少的等式。

要证明$\mathcal{T}$ 在$\mathcal{C}$ 中是初始的，第一步是证明从闭项代数到$\mathcal{C}$ 的任何代数都有一个同态。正好，例 2.12 说明含义函数$\mathcal{A}[\![\cdot]\!]$是从$\mathcal{T}$ 到任何Σ代数$\mathcal{A}$ 的一个同态。剩下是证明该同态的

唯一性。如果 h 是从$\mathcal{T}$ 到Σ代数$\mathcal{A}$ 的任意同态，那么由引理 2.13，h 和含义函数是相同的。注意，对于含变量的项，这个议论是不成立的。□

下面先证明任何代数类的所有初始代数都是同构的。

引理 2.15 假定 $h : \mathcal{A} \to \mathcal{B}$ 和 $k : \mathcal{B} \to \mathcal{A}$ 都是同态，并且 $h \circ k = Id_{\mathcal{B}}$，$k \circ h = Id_{\mathcal{A}}$，那么$\mathcal{A}$ 和$\mathcal{B}$ 是同构的，其中 Id 是恒等函数。

证明 根据同构的定义，直接证明对每个类别 s，同态 h^s 是双射便可以了。显然，h^s 一定是满射的，因为 k^s 把每一个 $x \in B^s$ 映射到某个 $y \in A^s$ 使得 $h^s(y)=x$。基于类似的推理，可以看出 h^s 是单射的。其实该证明仅使用 h 和 k 是一簇以类别为索引的函数这个事实，而并不依赖于 h 和 k 是同态。□

因为同构代数本质上是相同的，这就意味着任何代数类的初始代数如果存在，那么它唯一到同构。

命题 2.16 如果$\mathcal{A}$ 和$\mathcal{B}$ 在代数类$\mathcal{C}$ 中都是初始代数，那么$\mathcal{A}$ 和$\mathcal{B}$ 是同构的。

证明 因为$\mathcal{A}$ 和$\mathcal{B}$ 在代数类$\mathcal{C}$ 中都是初始代数，那么存在唯一同态 $h : \mathcal{A} \to \mathcal{B}$ 和 $k : \mathcal{B} \to \mathcal{A}$。由引理 2.15，只要证明 $h \circ k = Id_{\mathcal{B}}$并且 $k \circ h = Id_{\mathcal{A}}$就可以了。

由引理 2.14，$h \circ k$ 和 $k \circ h$ 分别是$\mathcal{A}$ 到$\mathcal{A}$ 和$\mathcal{B}$ 到$\mathcal{B}$ 的同态。因为 $Id_{\mathcal{A}}$和 $Id_{\mathcal{B}}$也分别是$\mathcal{A}$ 到$\mathcal{A}$ 和$\mathcal{B}$ 到$\mathcal{B}$ 的同态，由初始同态的唯一性，$h \circ k = Id_{\mathcal{B}}$，$k \circ h = Id_{\mathcal{A}}$。□

在例 2.13 中，在写符号 $nat, 0 : nat$ 和 $S : nat \to nat$ 时，通常把它们解释成自然数 0 和后继函数，其实这是该基调的“预期模型”。该例中，基调Σ_0 的初始代数和该预期模型同构。

如果$\mathcal{E}$ 是一组Σ等式，如果在满足$\mathcal{E}$ 的所有Σ代数构成的代数类中$\mathcal{A}$ 是初始的，那么就说$\mathcal{A}$ 对$\mathcal{E}$ 是初始的。

命题 2.17 令$\mathcal{E}$ 是一组Σ等式，并且令$\mathcal{A} = Terms(\Sigma, \varnothing)/\sim_{\mathcal{E}, \varnothing}$是只有闭项并且模可证的相等性的代数。那么，$\mathcal{A}$ 对$\mathcal{E}$ 来说是初始代数。

在$\mathcal{E}$ 是空集的情况下，很容易看出 $Terms(\Sigma, \varnothing)/\sim_{\mathcal{E}, \varnothing}$和 $Terms(\Sigma, \varnothing)$同构。因此，从这个命题知道，闭项代数在所有的Σ代数构成的类中是初始的。

证明 由项代数和商的性质，从$\mathcal{E}$ 可证的等式在$\mathcal{A}$ 中都成立。剩下要证明的是从$\mathcal{A}$ 到任何满足$\mathcal{E}$ 的代数有唯一的同态。

令$\mathcal{B}$ 是满足$\mathcal{E}$ 的一个代数。对$\mathcal{A}$ 中的任何等价类$[M]$，令 $h([M])$是不含变量的项 M 在$\mathcal{B}$ 中的含义$\mathcal{B}[\![M]\!]$。因为$\mathcal{B}$ 满足$\mathcal{E}$，因此这个映射是很好地定义了的。很容易检查 h 是个同态。由引理 2.13，任何其他同态 $h' : \mathcal{A} \to \mathcal{B}$ 和 h 一定有相同的值，因此 h 是从$\mathcal{A}$ 到$\mathcal{B}$ 的唯一同态。□

由此可以得出，任何代数规范都有初始代数。

下面的例子是自然数作为初始代数的延伸讨论，2.6 节还有一些其他例子。

例 2.14 考虑只有一个类别 nat 的基调Σ_1，它有函数符号 $0 : nat$，$S : nat \to nat$ 和$+ : nat \times nat \to nat$。令$\mathcal{E}$是等式集合

$$x + 0 = x$$

$$x + (Sy) = S(x + y)$$

在此省略类别指派，因为这里只有一个类别。令$\mathcal{C}$ 是满足$\mathcal{E}$ 的所有Σ_1 代数构成的代数类。

基于这组$\mathcal{E}$，可以证明有如下事实：

（1）$S^k0 + S^l0 = S^{k+l}0$；

（2）对任何闭项 M，存在某个自然数 k，使得 $M = S^k0$；

（3）等式 $S^k0 = S^l0$ 是不可证的，除非 $k = l$；

（4）每个等价类正好包含一个形式为 S^k0 的项；

（5）等价类和形式为 S^k0 的项集间有一个双射。

可以把这个项代数的载体看成由闭项 $0, S0, \ldots, S^k0, \ldots$构成的集合，并且在这个初始代数

中，函数 S 映射 $S^k0 \mapsto S^{k+1}0$，+映射$(S^k0, S^l0) \mapsto S^{k+l}0$。于是，这个初始代数和该基调的标准模型（有后继算子和加法的自然数）同构。

该规范的初始代数可以和其他有更多或较少元素的代数相对照。一般来说，如果一个代数有更多元素的话，那么这些多余元素不能由项定义，并且不会出现在该代数上的计算中，这些元素被称为“垃圾”。 如果一个代数有较少元素的话，那么就有一些不能被证明为相等的有区别的元素被等同，这种情况被称为“混淆”。先用例子来说明这两种可能性，而把证明这些例子的非初始性作为习题。

比初始代数含更多元素的代数有整数代数$\mathcal{Z}$ 。另一个例子由加“无穷个”整数到$\mathcal{N}$来构造。后者更一般，因为也可以用这种方法来构造表、集合、树和其他规范的非初始代数。对Σ_1，非初始代数$\mathcal{A} = \langle A^{nat}, 0^{\mathcal{A}}, S^{\mathcal{A}}, +^{\mathcal{A}}\rangle$有

$$A^{nat} = (\{0\} \times \mathcal{N}) \cup (\{1\} \times \mathcal{Z})$$

其中$\mathcal{N}$是自然数集合，$\mathcal{Z}$是整数集合。直观上，A^{nat} 含一个自然数“拷贝”，其中每个元素是序对$\langle 0, n\rangle$的形式，A^{nat} 还含一个整数“拷贝”，其中每个元素是序对$\langle 1, n\rangle$的形式。若把所有这些元素列成一行，可以让自然数出现在整数的左边：

$$0, 1, 2, 3, \ldots \quad \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$$

然后把左边的自然数想象成“小”或“有限”的数，把右边的整数想象成“大”或“无限”的数，因为右边的整数被看成大于左边所有的有限数。$0^{\mathcal{A}}$ 被解释为最左边的零，任何数的后继是上面序列中右邻它的数。如果同一部分的两个数相加，将得到标准的结果（有限或无限）。如果一个有限数加到无限数上，结果将是无限数。用形式的定义给出如下：

$$\begin{aligned} 0^{\mathcal{A}} &= \langle 0, 0\rangle \\ S^{\mathcal{A}}\langle i, n\rangle &= \langle i, n+1\rangle \\ \langle i, n\rangle +^{\mathcal{A}} \langle j, m\rangle &= \langle max(i, j), n+m\rangle \end{aligned}$$

可以检查，该规范的两个公理在这些代数中都成立。子集$(\{1\} \times \mathcal{Z})$的元素在这儿是“垃圾”，它们不能由闭项定义。因为初始代数所含的元素都可由闭项定义，因此这些“垃圾”不出现在初始代数中。

比初始代数含元素少的代数有模 k 的自然数（k 是任意自然数），例 2.11 已讨论了这个代数。很容易检查，该规范的两个公理在这个代数中都成立。认为它有混淆的原因是，在该规范中不能证明为相等的不同数码在该代数中有同样的值。而在初始代数中，只有在能证明两个闭项相等时，它们才有同样的值。 □

初始代数的一个重要特点是，它可能满足不能由$\mathcal{E}$ 证明的额外等式。直观上，增加元素到一个代数可能引起带变量的等式变得不能成立，因为一个新元素可能和所有其他元素共享的一个性质冲突。由于初始代数的载体尽可能小，初始代数满足的等式可能在其他满足$\mathcal{E}$ 的，有更多元素的代数中不成立。一个简单的例子是只有一个类别和两个常量 a 和 b 的基调。满足 $a = b$ 的初始代数只有一个元素。该初始代数满足 $x = y$，但是它不是等式 $a = b$ 的语义蕴涵，因为满足 $a = b$ 的含更多元素的代数不满足 $x = y$。一个更加自然的例子如下：

例 2.15 例 2.14 的初始代数满足+的交换性，虽然它不可能从那些等式证明。很容易看出，在初始代数中加法是可交换的，因为对任何不含变量的项 M 和 N，有$\mathcal{E} \vdash M = S^k0$ 和$\mathcal{E} \vdash N = S^l0$（对某个 k 和 l），并且 $\mathcal{E} \vdash M + N = S^{k+l}0 = N + M$。但是等式 $x + y = y + x$ 从$\mathcal{E}$ 不可证。可以例举一个满足$\mathcal{E}$ 的代数，其+的解释无交换性。

令 X 是任意至少含两个元素的集合，令 A^{nat} 是所有 $f : X \to X$ 的函数的集合。令 $0^{\mathcal{A}}$是 X 上的恒等函数，令 $S^{\mathcal{A}}$是 A^{nat} 上函数之间的恒等映射，并且令$+^{\mathcal{A}}$是 A^{nat} 上的函数合成。很容易检查，代数$\mathcal{A} = \langle A^{nat}, 0^{\mathcal{A}}, S^{\mathcal{A}}, +^{\mathcal{A}}\rangle$满足$\mathcal{E}$ 。举例说，

$$S^{\mathcal{A}}(f +^{\mathcal{A}} g) = f \circ g = f \circ S^{\mathcal{A}}(g) = f +^{\mathcal{A}} S^{\mathcal{A}}(g)$$

因为 $S^{\mathcal{A}}$是函数上的恒等映射。在这个代数中$+^{\mathcal{A}}$ 没有交换性，因为函数合成没有交换性。具

体说，因为 X 至少含两个元素，那么可以找到两个函数 $f, g : X \to X$，使得 $f \circ g \neq g \circ f$。 □

不含变量的项又叫做**基项**（*ground term*）。如果 S 是一个代换，它把Γ中的变量映射到Σ上的基项，就说 S 是Σ，Γ的一个**基代换**。如果 $M \in Terms^s(\Sigma, \Gamma)$并且 S 是一个Σ，Γ基代换，就说 SM 是 M 的**基实例**。类似地，如果 $M = N$ [Γ]是 $Terms^s(\Sigma, \Gamma)$的项之间的等式，并且 S 是一个Σ，Γ基代换，那么就把 $SM = SN$ [∅]称为 $M = N$ [Γ]的**基实例**。下面的命题刻画在初始代数中成立的等式的特征。

命题 2.18 令$\mathcal{E}$ 是一组Σ等式，并且$\mathcal{A}$ 对$\mathcal{E}$ 来说是初始代数。对任何Σ等式 $M = N$ [Γ]，下面三个条件等价：

(1) $\mathcal{A}$ 满足 $M = N$ [Γ]；

(2) $\mathcal{A}$ 满足 $M = N$ [Γ]的每一个基实例；

(3) $M = N$ [Γ]的每一个基实例都可以从$\mathcal{E}$ 证明。

证明 不失一般性，假定初始代数$\mathcal{A}$ 是 $Terms(\Sigma,\varnothing)/{\sim_{\mathcal{E},\varnothing}}$。下面将使用事实：一个Σ，Γ基代换正好和$\mathcal{T} = Terms(\Sigma, \varnothing)$的一个满足Γ的环境$\eta$相同。

考虑等式 $M = N$ [Γ]，这个等式在$\mathcal{A}$中可满足，当且仅当对每个满足Γ的$\mathcal{T}$ 环境η，有$\mathcal{A}[\![M]\!]\eta_\sim = \mathcal{A}[\![N]\!]\eta_\sim$。由引理 2.5 和 2.8，它等价于

$$[\eta M]_\sim = [\mathcal{T}[\![M]\!]\eta]_\sim = [\mathcal{T}[\![N]\!]\eta]_\sim = [\eta N]_\sim$$

因为这些项的等价类由$\mathcal{E}$ 的可证性确定，因而有$\mathcal{A}, \eta_\sim \vDash M = N$ [Γ]当且仅当$\mathcal{E} \vdash \eta M = \eta N$。这就证明了(1)和(3)等价。

如果 S 是任意Σ，Γ基代换，那么$\mathcal{A}[\![SM]\!]$不依赖于任何环境。于是$\mathcal{A}[\![SM]\!] = [SM]_\sim$，对 N 也同样。根据类似上面的推理可以得到，$\mathcal{A}$满足 $M = N$ [Γ]的每个基实例，当且仅当每个基等式 $SM=SN$ 从$\mathcal{E}$ 可证。即(2)和(3)等价。 □

2.5 代数数据类型

2.5.1 代数数据类型

对于像自然数和布尔值这样的标准数学结构，大家已经非常清楚。但是对于程序设计中的其他许多数据类型就不是这样，例如对于优先队列和符号表，它们不存在标准的数学构造，没有单一和标准的计算机实现。对于这样的数据类型，通常是列出它们的操作并描述这些操作的行为和性质，它们是公理化地定义而不是由数学构造来定义。不管是否用一种形式语言来描述反映操作性质的公理，公理化方法在实践中的优点是，可以精确说明对所有实现的共同要求，而不偏向任何一个特定实现。

本节剩余部分讨论数据类型公理化方法的一般特征。

一个简单的例子是有限集合数据类型，它有从属关系测试、并集运算和插入运算。它的规范在表 2.2 给出，这是通过增加类别 *set* 来把集合加到 *nat* 和 *bool* 的多类代数中。

表 2.2 *set, nat* 和 *bool* 的规范

sorts:	*set*, *nat*, *bool*
fctns:	0, 1, 2, … : *nat*
	$+ : nat \times nat \to nat$
	$Eq? : nat \times nat \to bool$
	$true, false \to bool$
	empty : *set*
	$insert : nat \times set \to set$
	$union : set \times set \to set$

ismem? : *nat* × *set* → *bool*
$cond_n$: *bool* × *nat* × *nat* → *nat*
$cond_b$: *bool* × *bool* × *bool* → *bool*
$cond_s$: *bool* × *set* × *set* → *set*

eqns: [*x*, *y* : *nat*, *s*, *s*′ : *set*, *u*, *v* : *bool*]
0 + 0 = 0, 0 +1= 1, 1 +1 = 2, …
Eq? *x x* = *true*
Eq? 0 1 = *false*, *Eq*? 0 2 = *false*, …
ismem? *x empty* = *false*
ismem? *x* (*insert y s*) = **if** *Eq*? *x y* **then** *true* **else** *ismem*? *x s*
union empty s = *s*
union (*insert y s*) *s*′ = *insert y* (*union s s*′)
$cond_n$ *true x y* = *x*
$cond_n$ *false x y* = *y*
$cond_b$ *true u v* = *u*
$cond_b$ *false u v* = *v*
$cond_s$ *true s s*′ = *s*
$cond_s$ *false s s*′ = *s*′

在表 2.2 中，*x* + *y* 仍然看成+*xy* 的语法美化，表达式 **if** *M* **then** *N* **else** *P* 看成是 *cond MNP* 的语法美化。集合类型没有标准的打印表示，通常也不作为输入或输出值。这儿并不关心集合的内部实现方式（如比特向量、数组、链表或其他），关心的是其最终结果为自然数或布尔值的一个操作序列的行为。从“公理化地定义”数据类型这个概念上看，这些数据类型称为是**抽象的**。数据抽象的一般原理如下：

（1）抽象数据类型由它的规范定义。使用这样数据类型的程序应该只依赖于由它的规范保证的性质，而不依赖于它的任何特定实现。

（2）在很多普通的例子中，一个数据类型的函数可以划分成构造算子、运算算子和观察算子。**构造算子**是一个函数，它产生该数据类型的一个新元素。**运算算子**是该数据类型上的函数，但它不产生新的元素。**观察算子**应用于该数据类型的元素，但返回其他类型的元素，如自然数或布尔值。对于这样的数据类型的推理，可以使用基于构造算子的归纳。

（3）有一些标准可用于确定一个实现是否正确地满足规范，但这个课题不在此讨论。另一方面，对于任意一个数据类型，很难断定是否给出了“足够”的公理。

在表 2.2 的集合例子中，*empty* 和 *insert* 是构造符，*union* 是运算符，函数 *ismem*?是一个观察符。

2.5.2 初始代数语义和数据类型归纳

代数规范是代数数据类型的公理化方式的有穷语法描述，即代数规范 *Spec* = ⟨Σ, $\mathcal{E}$ ⟩中的 $\mathcal{E}$ 是Σ确定的代数项语言的公理语义。从介绍代数项在代数中的解释开始，就不断在讨论代数项语言的指称模型。本小节研究选择什么样的代数作为代数规范的指称语义。

在文献中，代数规范有几种不同的“语义”形式。最一般的语义叫做**宽松语义**（*loose semantics*），即满足一个代数规范的所有代数所构成的代数类称为该规范的语义。这是从普通的数学观点出发的语义，其问题是这些代数可能划分成若干个同构类。另一种最常用的语义叫做**初始代数语义**，即满足一个代数规范的所有初始代数所形成的同构类称为该规范的语义。因为仅一个同构类，因此初始代数语义本质上为规范给出“唯一”的标准模型。先前已讨论过的初始代数的两个重要性质总结在下面两个口号中：

- **没有垃圾**，即不存在额外的不能由代数项命名的元素。也可以说仅包含由基项定义的元素，或者说包含尽可能少的元素。
- **没有混淆**，只有在规范有要求时元素才会等同。也可以说初始代数仅满足基项之间可证的等式，或者说不能证明为相等的项必定解释到不同的元素。

初始代数的一个重要结果是支持基于数据类型构造子的归纳，叫做**数据类型归纳**。在举例之前，先区别代数规范中的构造符和其他函数符号。形式地说，$Spec = \langle \Sigma, \mathcal{E} \rangle$的函数符号子集 F_0 是**构造符集合**，如果 $Terms(\Sigma, \varnothing)/\sim_{\mathcal{E}, \varnothing}$的每个等价类正好只含一个由 F_0 的函数符号所构成的基项。很容易明白，可以基于对构造符的归纳来证明初始代数的性质，因为按这种方式，初始代数的每个元素都被覆盖。虽然基于构造符集合的归纳可以简化证明，但是基于所有的项来证明初始代数的性质也不会带来什么害处。

例 2.16 对于表 2.2 的规范，*empty* 和 *insert* 是 *set* 类型的构造符，*union* 是个运算符，*ismem*? 是个观察符。若要证明 *empty* 和 *insert* 是一个构造符集合，必须证明在初始代数里面类别为 *set* 的项集中，每个等价类只有一个项仅用 *empty*，*insert* 和类别为 *nat* 的常量构成。这可以基于项的结构来归纳证明，在此省略。

可以用一个简单的例子来说明数据类型的归纳：证明表 2.2 规范的初始代数中的每个集合仅含有限多个元素。为叙述简单，在此对规范中的符号和它在初始代数中的含义不加区分。集合有限是指，如果 *ismem*? *x s* = *true* 仅对有限多个原子 *x* 成立，那么集合 *s* 有限。因为该初始代数中任何集合 *s* 都可由 *empty* 和 *insert* 构造的项表示，因此基于这种项的结构进行归纳，以证明对任何集合 *s*，表达式 *ismem*? *x s* 仅对有限多个原子 *x* 的求值为 *true*。对于归纳基础，*ismem*? *x empty* 对所有的 *x* 为 *false*。对于归纳步骤，假定 *ismem*? *x s* 最多对 *n* 个原子 *x* 为 *true*；那么 *ismem*? *x* (*insert a s*)最多对 *n*+1 个原子 *x* 为 *true*。这就完成了证明。 □

再简单介绍代数规范的另两种语义形式，以作为和初始语义的对比，对所提到的性质，在此不作证明。

除了宽松语义和初始语义外，还有其他的语义形式，一种叫做**生成语义**或**可到达语义**（也有人称这种形式为宽松语义）。在这种方式下，一个规范的模型是所有满足该规范的生成代数。一个Σ代数$\mathcal{A}$ 是**生成的**（或叫做**可到达的**，或叫做**项生成的**），如果对每一个元素 $a \in A^s$，存在一个基项 M 使得 $a = \mathcal{A}[\![M]\!]$。生成语义和初始语义间的联系如下：

（1）如果Σ生成代数$\mathcal{A}$ 满足$\mathcal{E}$，那么从初始代数$\mathcal{T}$=$Terms(\Sigma,\varnothing)/\sim_{\mathcal{E}, \varnothing}$ 到生成代数$\mathcal{A}$ 有一个满射。

（2）如果Σ生成代数$\mathcal{A}$ 满足$\mathcal{E}$，那么 $Th(\mathcal{A})$包含规范$\langle \Sigma, \mathcal{E} \rangle$的初始代数的理论。

（3）等式 $M = N$ [Γ]在满足$\mathcal{E}$ 的所有Σ生成代数中都成立当且仅当该等式在规范$\langle \Sigma, \mathcal{E} \rangle$的初始代数中成立。

另一种语义形式叫做**终结代数语义**（*final algebra semantics*）。终结代数与初始代数的概念相反：在一个代数类$\mathcal{C}$ 中，如果从每个$\mathcal{B} \in \mathcal{C}$ 到$\mathcal{A} \in \mathcal{C}$，都存在唯一的同态，那么代数$\mathcal{A}$是$\mathcal{C}$ 的终结代数。一个代数类中的所有终结代数都同构。若用满足一个规范的终结代数作为该规范的语义，则称之为终结语义。

若$\mathcal{C}$ 是满足某个规范的所有代数构成的代数类，如果所有载体都是单元素集合的代数也在$\mathcal{C}$ 中，则这个代数一定是$\mathcal{C}$ 的终结代数，因为从任何集合到单元素集合都正好一个函数。

如果把某些类别的解释固定，而其他类别的解释用终结语义，则它是一种有用的方法。因为用初始语义时隐含的意思是，不能证明为相等的就是不相等的，即前面已经说过的初始代数满足尽可能少的闭等式。而用终结语义时则反过来，不能证明为不相等的则是相等的。在此不深入讨论，只举一个简单的例子来说明两者的区别。

从初始语义角度看，表 2.2 规范中类别 *set* 其实还不是一般集合的规范，而是表的规范。因为不能证明

$$insert\ x\ (insert\ y\ z) = insert\ y\ (insert\ x\ z)\ [x : nat, y : nat, z : set]$$

因此元素的插入次序不同，则得到的是不同的集合。还因为不能证明

$$insert\ x\ (insert\ x\ z) = insert\ x\ z\ [x : nat, z : set]$$

因此元素可以在集合中出现多次。所以说这是表的规范而不是集合的规范。要想得到集合的规范，必须增加描述插入运算可交换的等式和消除多重元素的等式。

但是若用终结语义，并且 *nat* 和 *bool* 的解释固定到自然数和布尔值，那么表 2.2 的规范就是集合的规范，无需增加描述插入运算可交换的等式和消除多重元素的等式。

2.5.3 解释没有意义的项

在程序设计时经常会碰到表达式没有意义的情况，举两个例子如下：

（1）除法是一个部分函数，除数为零的表达式没有意义。解决办法是在求商之前，先测试除数是否为零。

（2）调用不会终止的函数也构成一个没有意义的表达式。没有算法可用来测试任意函数对任意变元是否终止。

如果想在代数规范中表示这些情况，如说明除数为零时产生一个错误，那么必须在基调中增加表示错误的项（简称错误值）。

在代数规范中，导致错误的运算会引起一些问题。因为这样的项也是良形项，而良形项在基调的任何代数中都有值，那么怎样规定这些项的值。这里调查三种可能的方式，粗略地讲，它们是：“什么也不规定”、“任意做一个决定”和“非常仔细地说明什么是所需要的”。

作为例子，将表的规范列在表 2.3。所用的名字遵从 Lisp 的术语。

表 2.3　*list*，*atom* 和 *bool* 的一个规范

sorts:	*list*, *atom*, *bool*
fctns:	*a*, *b*, *c*, *d*, … : *atom*
	true, *false* : *bool*
	nil : *list*
	$cons : atom \times list \to list$
	$car : list \to atom$
	$cdr : list \to list$
	$isempty? : list \to bool$
	$cond_a : bool \times atom \times atom \to atom$
	$cond_b : bool \times bool \times bool \to bool$
	$cond_l : bool \times list \times list \to list$
eqns:	[*x*, *y* : *atom*, *l*, *l′* : *list*, *u*, *v* : *bool*]
	$car(cons\ x\ l) = x$
	$cdr(cons\ x\ l) = l$
	$isempty?\ nil = true$
	$isempty?\ (cons\ x\ l) = false$
	$cond_a\ true\ x\ y = x$
	$cond_a\ false\ x\ y = y$
	$cond_b\ true\ u\ v = u$
	$cond_b\ false\ u\ v = v$
	$cond_l\ true\ l\ l' = l$
	$cond_l\ false\ l\ l' = l'$

针对该规范的问题是，求空表的第一个表元是没有意义的，因此 *car nil* 没有“自然”

的值。运算 *cdr* 是求删除第一个表元后的剩余表元构成的表，*cdr nil* 也没有意义。这两个问题在表 2.3 中的体现是没有为 *car nil* 和 *cdr nil* 准备等式。可以这样解释：因为这些项没有意义，所以不想让它们等于任何有意义的值。但是，如果把初始代数作为规范的“标准语义”，马上可以看出对这些项没有规定函数值的一些缺点，这也成为反对强调初始代数的一个理由。在初始代数 $\mathcal{A}$ 中，因为证明不了 *car nil* 等于任何原子常量 *a*, *b*, *c*, …，也证明不了 *cdr nil* 等于任何仅用 *cons* 和 *nil* 构造的表，因此在载体中必须有专门对应 *car nil* 的元素属于 A^{atom} 并且有专门对应 *cdr nil* 的元素属于 A^{list}。然而，这些“额外”的元素并不就此停止。一旦有了原子 *car nil*，则可以把它加到表中，如 *cons* (*car nil*) *nil*。类似地，也可以测试额外的表 *cdr nil* 是否非空，即 *isempty*? (*cdr nil*)。这又要产生一个额外的布尔值，因为不可能证明它等于 *true* 或 *false*。这样，通过把函数应用到 *car nil* 和 *cdr nil*，会产生该初始代数的无数多个元素。它们使得所期望的在表构造符 *nil* 和 *cons* 上的归纳推理成为无效，对 *atom* 和 *bool* 也有类似的问题。

另一种方便但也存在问题的办法是，任意做一个规定。例如，对任意的 *a*，令 *car nil* = *a*，令 *cdr nil* = *nil*。这样做可恢复数据类型在所期望的构造符上的归纳。但是由于 *a* 的随意性，它破坏了“有相同的 *car* 和 *cdr* 的表相同”这个性质。因为 *car nil* = *car* (*cons a nil*)并且 *cdr nil* = *cdr* (*cons a nil*)，但是 *nil* 和 *cons a nil* 不相同。因此任意规定总会引起这样或那样的问题。

有关错误情况的更直接的办法是在代数规范中包含一个显式的错误值及与它们有关的公理。这似乎可以得到更加合理的初始代数并允许包含“错误管理”的函数，这些函数测试错误情况并试图修复它们。但是下面可以看到，这些公理将比所预想的要复杂得多。本节剩余部分致力于错误值的公理化处理。

一种朴素的想法是增加错误值到表数据类型，即加上常量符号 $error_a$: *atom* 和 $error_l$: *list* 和等式 *car nil* = $error_a$ 和 *cdr nil* = $error_l$。但是，一旦加了这两个错误值，就会发现还得加入第三个，因为判断一个错误表是否为空又引出一个新的错误值 $error_b$，用于等式 *isempty*? $error_l$ = $error_b$。加上这些常量后，必须考虑函数在这些值上的行为。对于这个例子中的大多数函数，可以让函数应用于错误值的结果是适当类型的错误值。条件表达式是一个例外，如果不取错误分支的话，可以返回一个有意义的值。这里有一点点随意，因为看起来 $cond_a$ *true a* $error_a$ = $error_a$ 也是合理的。上述增加错误值的想法引起函数符号和等式公理的增加，它们列在表 2.4 中。

表 2.4 *list*, *atom* 和 *bool* 错误值的朴素处理

fctns:	$error_l$: *list*
	$error_a$: *atom*
	$error_b$: *bool*
eqns:	[*x*, *y* : *atom*, *l*, *l*′ : *list*, *u*, *v* : *bool*]
	car nil = $error_a$
	cdr nil = $error_l$
	cons $error_a$ *l* = $error_l$
	cons x $error_l$ = $error_l$
	car $error_l$ = $error_a$
	cdr $error_l$ = $error_l$
	isempty? $error_l$ = $error_b$
	$cond_a$ $error_b$ *x y* = $error_a$
	$cond_b$ $error_b$ *u v* = $error_b$
	$cond_l$ $error_b$ *l l*′ = $error_l$

表 2.3 和表 2.4 的基调和等式组合在一起后，第一眼看去，它似乎是合理的。但是仔细

观察，会发现有严重问题。在起初写非空表的 *car* 和 *cdr* 的公理时，没有考虑错误值的可能性，换句话说，等式公理

$$car\ (cons\ x\ l) = x\ [x : atom, l : list]$$

隐含地假定任何由原子 *x* 和表 *l* 合起来的表 *cons x l* 的第一个元素是 *x*。但是公理

$$cons\ x\ error_l = error_l\ [x : atom]$$

违反了这个假定。事实上，把这两个等式公理放在一起会推出

$$x = car\ (cons\ x\ error_l) = car\ error_l = error_a\ [x : atom]$$

这个等式从语义上说，满足表 2.3 和表 2.4 构成的规范的每个代数只有一个原子。类似的推理表明只存在一张表和一个布尔值。因此这种朴素的错误处理办法导致不一致的规范。

上述问题可以通过对函数变元作显式的假设而克服。如果允许在描述代数的语言中加入逻辑否定和其他逻辑连接词的话，那么就可以写一个公理

$$x \neq error_a \wedge l \neq error_l \Rightarrow car(cons\ x\ l) = x\ [x : atom, l : list]$$

它显式地规定，函数变元不能是错误值。

现在通过在基调中加入少数函数符号，可以在代数规范的框架中得到同样的结果。对于每个类别 *s*，加入一个函数符号 $OK_s : s \to bool$，它用于判断变元是否为 $error_s$。即用公理来规范哪些元素是 *OK* 的，并且为防止错误值，在等式中增加条件测试。对于表 2.3 的基调，可以加入函数符号 OK_a，OK_b，OK_l 和下面的公理

$$OK_a\ a = true, OK_a\ b = true, \ldots, OK_a\ error_a = false$$

来使每一个非错误值“*OK*”，并且每一个错误值不“*OK*”。然后使用 *OK* 和条件来修订非空表的 *car* 公理如下：

$$car(cons\ x\ l) = cond_a\ (OK_a\ x)\ (cond_a\ (OK_l\ l\ )\ x\ error_a)\ error_a \quad [x : atom, l : list]$$

注意，因为要维持 $cons\ x\ error_l = error_l$，因此从这个修改后的带条件的公理可以导出等式 $car\ error_l = error_a$。完整的规范在表 2.5 中，其初始代数的每个载体有一个错误元素，它和其他元素有区别。

表 2.5　有错误值的 *list*，*atom* 和 *bool* 的规范

sorts:	*list*, *atom*, *bool*
fctns:	*a*, *b*, *c*, *d*, … : *atom*
	true, *false* : *bool*
	nil : *list*
	$cons : atom \times list \to list$
	$car : list \to atom$
	$cdr : list \to list$
	$isempty? : list \to bool$
	$cond_a : bool \times atom \times atom \to atom$
	$cond_b : bool \times bool \times bool \to bool$
	$cond_l : bool \times list \times list \to list$
	$error_a : atom, error_b : bool, error_l : list$
	$OK_a : atom \to bool, OK_b : bool \to bool, OK_l : list \to bool$
eqns:	[*x*, *y* : *atom*, *l*, *l′* : *list*, *u*, *v* : *bool*]
	$car\ error_l = error_a$
	$car\ nil = error_a$
	$car(cons\ x\ l) = cond_a\ (OK_a\ x)\ (cond_a\ (OK_l\ l\ )\ x\ error_a)\ error_a$
	$cdr\ error_l = error_l$
	$cdr\ nil = error_l$

$cdr(cons\ x\ l) = cond_l\ (OK_a\ x)\ (cond_l\ (OK_l\ l\ )\ l\ error_l)\ error_l$

$isempty?\ error_l = error_b$

$isempty?\ nil = true$

$isempty?\ (cons\ x\ l) = cond_b\ (OK_a\ x)\ (cond_b\ (OK_l\ l)\ false\ error_b)\ error_b$

$cons\ x\ error_l = error_l$

$cons\ error_a\ l = error_l$

$cond_a\ true\ x\ y = x$

$cond_a\ false\ x\ y = y$

$cond_a\ error_b\ x\ y = error_a$

（对 $cond_b$ 和 $cond_l$ 类似）

$OK_a\ a = true, OK_a\ b = true, OK_a\ c = true, \ldots$

$OK_a\ error_a = false$

$OK_b\ true = true, OK_b\ false = true$

$OK_b\ error_b = false$

$OK_l\ nil = true$

$OK_l\ (cons\ x\ l\ ) = cond_b\ (OK_a\ x)\ (cond_b\ (OK_l\ l\ )\ true\ false)\ false$

$OK_l\ error_l = false$

2.5.4 错误值的其他解决方法

对付错误值的另一种完全不同的方法是用部分函数而不用全函数。例如，不是把 *car nil* 看成类别 *atom* 的表达式，并且不要求它在任何代数中必须给出一个值。在这儿认为 *car* 对 *nil* 没有定义（这种方式不同于指称语义中引入一个额外的叫做“未定义”的元素⊥）。使用部分函数的基本好处是只存在由 *nil* 和 *cons* 生成的表。缺点是不能讨论错误管理函数，因为一个没有定义的表达式不可能产生一个值传给一个函数。此外，不能有不依赖于变元是否有定义的“惰性”函数。这是因为在采用部分函数方式时，若表达式含没有定义的子表达式，那么该表达式不可能以任何合理的方式（例如合成）给出一个值。最后，有部分函数的代数比普通的只有全函数的代数要复杂得多。因而不清楚用部分函数的方法是否值得去探索。

还有采用有序类别代数（*order-sorted algebra*）方式的，它也有一些优点，在此不介绍。

2.6 重写系统

2.6.1 基本定义

代数项语言的操作语义是用于代数项的归约系统，通常叫做**项重写系统**，简称**重写系统**。重写系统由一组叫做重写规则的有向等式组成。形式地说，Σ上**重写系统** $\mathcal{R}$ 是一组重写规则 $L \to R$，其中 L，$R \in Terms^s(\Sigma, \Gamma)$ 是同一类别的项，但 L 不能是变量，并且 $Var(R) \subseteq Var(L)$。规则 $L \to R$ 允许把一个项中 L 的实例 SL 化简成 SR。

细心的读者可能对变量上的限制 $Var(R) \subseteq Var(L)$ 感到奇怪。在用规则 $L \to R$ 去化简项时，如果某个变量仅出现在 R 中，则它可以被代换成任何项，因而很可能不是一种化简。另一个技术上的原因是为了让重写保项的类别，没有对变量的这个限制时，做不到这一点。

由 $\mathcal{R}$ 确定的一步归约关系 $\to_{\mathcal{R}}$ 是项上的最小关系，使得对每条规则 $(L \to R) \in \mathcal{R}$，含 x 一次出现的项 M 以及使得 $[SL/x]M \in Terms(\Sigma, \Gamma)$ 的代换 S，有

$$[SL/x]M \to_R [SR/x]M$$

关系 $\to\!\!\to_{\mathcal{R}}$ 是 $\to_{\mathcal{R}}$ 的自反传递闭包。当不会出现混淆时，下标可省略。很容易看出，系统地重

新命名任何规则$(L \to R) \in \mathcal{R}$ 中的变量不会改变关系$\to_{\mathcal{R}}$。因此可以假定没有一个变量会出现在不同的规则中。在比较一对规则时，这是很有用的。

例 2.17 描述包含 0，一元减和加的自然数表达式的基调如下：

sorts : *nat*

fctns : 0 : *nat*

$- : nat \to nat$

$+ : nat \times nat \to nat$

在这个基调上的一些归约规则如下：

$x + 0 \to x$

$x + (-x) \to 0$

$(x + y) + z \to x + (y + z)$

使用这些规则，可以执行下列重写（归约）步骤，其中每次被重写的部分就是加下划线的子项，它们被称为可归约式（*redex*）。

$\underline{(x + y) + (-y)} \to x + \underline{(y + (-y))} \to \underline{x + 0} \to x$

注意，一步归约的执行是通过把完整的项或它的一个子项同一条规则的左部进行匹配。这个例子中每一步正好只有一种重写的可能，但是经常会出现几个子项都可以和规则的左部匹配的情况。例如项$(x + 0) + y$ 可以由第一条规则归约成 $x + y$，也可以由第三条规则归约成 $x + (0 + y)$。注意，所得到的项都不能再归约。 □

重写系统有两种重要的应用。其一是对代数项提供一种计算模型，如 2.17 见到的那样，通过重写来化简一个项，一直到最简形式为止。第二种重要应用是自动定理证明。如果有一组等式，那么它们可以被定向为一组重写规则。如果所得到的重新系统是合流的（稍后解释），那么这组等式假设的推论可以用该重写系统来证明。这种方式的优点是，可以用简单的归约来判断一个等式是否可证。虽然归约并非总会终止，但对终止的重写系统（稍后解释）而言，归约过程肯定是终止的。如果归约终止而等式左右两项的归约结果不一样，那么该等式就不是这组等式假设的逻辑推论。

本节以代数项的重写系统研究为例，来说明非代数项重写系统的一般问题。一些定理的证明被略去，以避免读者陷入冗长的证明细节。

一个直截了当的事实是归约保类别。

引理 2.19 令$\mathcal{R}$ 是Σ上的重写系统。如果 $M \in Terms^S(\Sigma, \Gamma)$，并且 $M \to\!\!\!\!\to_{\mathcal{R}} N$，那么 $N \in Terms^S(\Sigma, \Gamma)$。

该引理的证明作为习题。

合流性和终止性是重写系统的重要性质。引入一些记号来描述合流性：如果 $M \to\!\!\!\!\to N$，则可把它写成 $N \leftarrow\!\!\!\!\leftarrow M$；如果存在一个项 P 使得 $M \to\!\!\!\!\to P \leftarrow\!\!\!\!\leftarrow N$，则可写成 $M \to\!\!\!\!\to \circ \leftarrow\!\!\!\!\leftarrow N$。如果 $N \leftarrow\!\!\!\!\leftarrow M \to\!\!\!\!\to P$ 蕴涵 $N \to\!\!\!\!\to \circ \leftarrow\!\!\!\!\leftarrow P$，则关系$\to\!\!\!\!\to$是**合流的**。图 2.1 是合流性的示意图。如果不存在一步归约的无穷序列 $M_0 \to M_1 \to M_2 \ldots$，则关系$\to$是**终止的**。如果$\to\!\!\!\!\to_{\mathcal{R}}$是合流的，则称重写系统$\mathcal{R}$ 是合流的。如果$\to_{\mathcal{R}}$是终止的，则称重写系统$\mathcal{R}$ 是终止的。不能再归约的项称为**范式**。

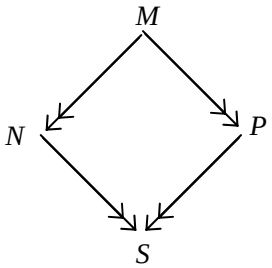

图 2.1 合流性示意图

合流并且终止的重写系统通常又叫做**典范**系统。虽然经常谈论的是典范系统，但是非典范系统也是非常有用的，尤其是当这些规则对项的某个子集合流时。在 2.6.6 节将看到，终止可以使得判定重写系统的合流性变得容易。

一种有用的关系是归约的无向版本，叫做**可变换性**。项 $M, N \in Terms^S(\Sigma, \Gamma)$是**可变换的**，写成 $M \leftrightarrow_{\mathcal{R}} N\ [\Gamma]$，如果 M 和 N 相同，或者存在某个项 $M' \in Terms^S(\Sigma, \Gamma)$，使得 $M \to\!\!\!\!\to_{\mathcal{R}} M'$或

者 $M' \twoheadrightarrow_{\mathcal{R}} M$，并且 $M' \leftrightarrow_{\mathcal{R}} N$ [Γ]。换句话说，$M \leftrightarrow N$ [Γ] 当且仅当存在一个项序列 $M_1, \ldots, M_k \in Terms^s(\Sigma, \Gamma)$使得

$$M \twoheadrightarrow M_1 \twoheadleftarrow M_2 \twoheadrightarrow M_3 \ldots M_k \twoheadleftarrow N$$

其中箭头的方向并没有什么意义。

一个项的两个子项可能在语法上完全一样，有时需要引用子项的某个特定出现，因此下面关于子项位置的定义是非常有用的。直观上，项中的一个**位置**是某个子项的在项中的场所。位置用自然数的有穷序列来标识，它告知如何从项的分析树根部来寻找该位置。举例来说明这种方式，$f\,(gx(hab))(gba)\;x$ 的分析树见图 2.2，为了找到由序列 1, 2 命名的位置，从树根开始。数 1 表示寻找由树的第一个（最左）分支代表的子树，它引导到 $gx(hab)$子树的根部。数 2 表示再寻找该子树的第二个分支。所以位置 1, 2 由子项 hab 占据。更精确地，可以对序列 $\vec{n} = n_1 \ldots n_k$ 的长度进行归纳来定义 M 中位置为 $\vec{n}$ 的子项。如果ε 是空串，那么在位置ε 的子项就是完整的 M。如果 M 的形式是 $f\,(M_1, \ldots, M_k)$并且 $\vec{n} = i, n_2 \ldots n_l\ (1 \le i \le k)$，那么在位置 $\vec{n}$ 的子项是 M_i 中命名为 $n_2 \ldots n_l$ 的子项。

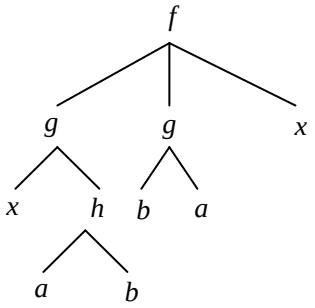

图 2.2 $f\,(gx(hab))(gba)\;x$ 的分析树

如果 $\vec{n}$ 是 M 中的一个位置，就用 $M\,|\,\vec{n}$ 表示 M 在 $\vec{n}$ 的子项，用$[N/\vec{n}\,]M$ 表示 M 在 $\vec{n}$ 的子项由 N 代换的结果。使用这种记号，一步归约可表示为：$M \to_{\mathcal{R}} N$ 当且仅当在 M 中有一个位置 $\vec{n}$ 使得 $M\,|\,\vec{n} \equiv SL$ 并且 $N \equiv [SR/\ \vec{n}\,]M$。

2.6.2 合流性和可证的相等性

合流的重写系统对分析可证的相等性是非常有用的。主要原因是，对于合流的重写系统，当一个等式的两个项都能归约到同一个项时，该等式一定可证。虽然只对代数系统讨论这个定理，但是本质上，同样的证明可以用到其他许多等式系统中，包括类型化λ演算和无类型的λ演算。

如果$\mathcal{R}$ 是一组重写规则，$\mathcal{E}_{\mathcal{R}}$ 用来表示对应的无向等式集合。使用这种记号，这里不加证明地给出下面的定理。

定理 2.20 对任何重写系统$\mathcal{R}$，$M \leftrightarrow_{\mathcal{R}} N$ [Γ]当且仅当$\mathcal{E}_{\mathcal{R}} \vdash M = N$ [Γ]。对任何合流的重写系统$\mathcal{R}$ ，$\mathcal{E}_{\mathcal{R}} \vdash M = N$ [Γ]当且仅当 $M \twoheadrightarrow_{\mathcal{R}} \circ \twoheadleftarrow_{\mathcal{R}} N$。

从这个定理，可以得到下面的推论。

推论 2.21 如果$\twoheadrightarrow_{\mathcal{R}}$是合流的，并且类别 s 存在两个不同的范式，那么$\mathcal{E}_{\mathcal{R}}$是一致的。

2.6.3 终止性

有很多方法可用于证明重写系统的终止性，其中一些方法给项以一种“权值”。例如，假定给每个项 M 一个自然数权值 w_M ，并且能证明每当 $M \to N$，则有 $w_M > w_N$ 。那么，由于不存在自然数的无穷递减序列，因此也就不存在一步归约的无穷递减序列。

这种方法看起来很好，但是设计一个能达此要求的权值计算函数是非常困难的。事实上，不存在测试任意重写系统是否终止的算法；但是在一些简单的场合，很多办法还是管用的。本节介绍基于有序载体代数的一种简单方法。该方法使用禁止产生无穷序列的二元关系。

先给出表示上的约定。如果$\prec$是集合 A 上的一个二元关系，那么用$\preceq$表示$\prec$关系和相等关系的并，换句话说，$x \preceq y$ 当且仅当 $x \prec y$ 或 $x = y$，并且 $y \succ x$ 当且仅当 $x \prec y$。注意，在使用$\prec$作为关系名字时，不必假定$\prec$是传递的或者具有序关系的其他性质。由于所使用的大多数例子是序关系，因此使用这个符号有直观上的帮助。在第 1 章已经定义过，集合 A 上的一个

关系$\prec$是**良基的**，如果不存在 A 上元素的无穷递减序列 $a_0 \succ a_1 \succ a_2 \succ \ldots$。良基关系在这里的重要性是这样，如果能在项和有良基关系的集合 A 的元素间建立起一个对应，那么可以利用它去证明不存在无穷的归约序列 $M_0 \to M_1 \to M_2 \to \ldots$。

有几种方式可把项映射到有良基关系的集合。因为已经研究了项在代数中的解释，因而可以利用代数的语义结构。一个代数$\mathcal{A} = \langle A^{s_1}, A^{s_2}, \ldots, f_1^{\mathcal{A}}, f_2^{\mathcal{A}}, \ldots\rangle$是良基的，如果：

（1）在每个载体 A^s 上有一个良基关系$\prec_s$，并且

（2）对每个 n 元函数符号 f，如果 $x_1 \preceq y_1, \ldots, x_n \preceq y_n$ 并且对某个 i（$1 \le i \le n$）有 $x_i \prec y_i$，那么 $f^{\mathcal{A}}(x_1, \ldots, x_n) \prec f^{\mathcal{A}}(y_1, \ldots, y_n)$。

一个简单的例子是让每个载体都是自然数集合$\mathcal{N}$，使用通常的序，并且用正系数的多项式来定义每个函数 $f^{\mathcal{A}}$。

若$\mathcal{A}$是一个良基代数，并且 M 和 N 都是类别 s 上的项，如果$[\![M]\!]\eta \prec_s [\![N]\!]\eta$，那么写成

$$A, \eta \vDash M \prec N$$

类似地，如果对任何环境η都有$\mathcal{A}, \eta \vDash M \prec N$，那么写成$\mathcal{A} \vDash M \prec N$。

定义良基代数的理由在下面的定理中给出：

定理 2.22 令$\mathcal{R}$是Σ项上的一个重写系统，并且令$\mathcal{A}$是一个良基的Σ代数。如果对$\mathcal{R}$ 中每条规则 $L \to R$ 都有$\mathcal{A} \vDash L \succ R$，那么$\mathcal{R}$ 是终止的。

证明 该证明使用代换引理和有关项的解释的其他事实。必须证明的基本事实仅是：对任何含变量 x 的项 $P \in Terms(\Sigma, \Gamma, x : s)$，在任何环境$\eta$下，如果 $a \succ b \in A^s$，那么

$$[\![P]\!]\eta[x \mapsto a] \succ [\![P]\!]\eta[x \mapsto b]$$

这很容易基于 P 的归纳来证明，把它留给读者。

为证明该定理，还需证明，如果 $M \to N$，那么对任何环境η有$[\![M]\!]\eta \succ [\![N]\!]\eta$。证明它也就足够了，因为良基载体不会有无穷的“递减”元素序列。通常，一步归约可以写成 $[SL/x]P \to [SR/x]P$ 的形式，其中 x 在 P 中正好只出现一次。于是必须证明，对任何环境η，

$$[\![[SL/x]P]\!]\eta \succ [\![[SR/x]P]\!]\eta$$

令η是任意环境，$a = [\![SL]\!]\eta$ 并且 $b = [\![SR]\!]\eta$。由代换引理，对某个由η和 S 决定的环境η'，有 $a = [\![L]\!]\eta'$并且 $b = [\![R]\!]\eta'$。于是，由该定理的假设，$a \succ b$。由本证明第一段提到的归纳，可以证明

$$[\![P]\!]\eta[x \mapsto a] \succ [\![P]\!]\eta[x \mapsto b]$$

再由代换引理可得$[\![[SL/x]P]\!]\eta \succ [\![[SR/x]P]\!]\eta$，从而得到本定理。 □

注意，当用项在良基代数上的解释来证明重写系统的终止性时，所用的代数并不满足和该重写系统相对应的那些等式。这是因为归约项时必须“减少”它在这个良基代数上的值。而对于满足该重写系统相对应的那些等式的代数来说，重写不改变项的解释。

把表 2.1 中关于栈的两个等式从左到右地定向，就得到一个非常简单的重写系统。很容易看出，这个系统是终止的，因为每条规则都减少项中符号的个数。使用一个良基代数，它的载体都是自然数。然后把每个函数符号都解释为数值函数，函数的值是其变元的和加 1。这样，一个项的含义就是它的函数符号的个数，很容易检查，每条重写规则都减小项的值。

很容易看出，当定理 2.22 的条件得到满足，并且每个载体都是自然数时，一个项的数值含义是关于该项最长归约序列的一个上界。如果重写的步数是关于项长度的一个快速增长函数，那么至少需要把一个函数符号解释为增长速度快于多项式的数值函数。下面的例子使用指数函数来说明到析取范式的重写总是终止的。

例 2.18 考虑下面命题逻辑公式的重写系统。通常把一元函数符号$\neg : bool \to bool$ 写成前缀算符，把二元函数符号$\wedge, \vee : bool \times bool \to bool$ 写成中缀算符。

$$\neg\neg x \to x$$
$$\neg (x \vee y) \to (\neg x \wedge \neg y)$$

$\neg (x \wedge y) \to (\neg x \vee \neg y)$

$x \wedge (y \vee z) \to (x \wedge y) \vee (x \wedge z)$

$(y \vee z) \wedge x \to (y \wedge x) \vee (z \wedge x)$

这个系统把一个公式变换成析取范式。

使用自然数良基代数能够证明这个重写系统是终止的。令$\mathcal{A} = \langle A^{bool}, \vee^{\mathcal{A}}, \wedge^{\mathcal{A}}, \neg^{\mathcal{A}}\rangle$是这样的一个代数，其载体 $A^{bool} = \mathcal{N} - \{0, 1\}$，即大于 1 的自然数，并且

$\vee^{\mathcal{A}}(x, y) = x + y + 1$

$\wedge^{\mathcal{A}}(x, y) = x * y$

$\neg^{\mathcal{A}}(x) = 2^x$

很容易看出，这些函数在自然数上是严格单调的（按照通常的序）。于是$\mathcal{A}$ 是一个良基代数。

为了证明该重写系统是终止的，必须检查对于自由变量的任意取值，每条重写规则左部定义的值都大于其右部定义的值。第一条规则是明显的，因为对任何自然数 $x > 1$，有 $2^{2^x} > x$。再看第二条规则，对任意 $x, y > 1$，有 $2^{(x+y+1)} = 2^x 2^y 2 > 2^x 2^y$。剩下的规则通过类似的计算可得。 □

例 2.19 本例证明从表 2.6 的整数多重集合公理导出的重写系统的终止性。为简单起见，省略多重集合和布尔值上的条件规则。因为这些规则和其他规则相互不影响。考虑的基调有类别 *mset*, *nat* 和 *bool*，常量有自然数、空多重集合和 *true* 及 *false*，函数符号有+, *Eq*?, *insert*, *count* 和 $cond_n$。重写规则是

$0 + 0 \to 0, 0 + 1 \to 1, \ldots$

Eq? *x x* → *true*

Eq? 0 1 → *false*, *Eq*? 0 2 → *false*, …

count x empty → 0

count x (*insert y m*) → **if** *Eq*? *x y* **then** (*count x m*) + 1 **else** *count x m*

$cond_n$ *true x y* → *x*

$cond_n$ *false x y* → *y*

该基调在代数$\mathcal{A}$ 中进行解释，其中 $A^{nat} = A^{mset} = A^{bool} \subseteq \mathcal{N}$。对每个函数符号，必须选择严格单调的函数，使得每条规则左部的含义保证大于其右部的含义。对于大多数函数符号，这种选择是简单的；由于第二条 *count* 规则右部函数符号的个数多于左部的，因此必须更加仔细考虑 *count* 的数值解释。

表 2.6 多重集合、*nat* 和 *bool* 的代数规范

sorts :	*mset*, *nat*, *bool*
fctns :	0, 1, 2, … : *nat*
	+ : *nat* × *nat* → *nat*
	Eq? : *nat* × *nat* → *bool*
	true, *false* : *bool*
	empty : *mset*
	insert : *nat* × *mset* → *mset*
	count : *nat* × *mset* → *nat*
	$cond_n$: *bool* × *nat* × *nat* → *nat*
	$cond_b$: *bool* × *bool* × *bool* → *bool*
	$cond_m$: *bool* × *mset* × *mset* → *mset*
eqns :	[*x*, *y* : *nat*; *m*, *m*′ : *mset*; *u*, *v* : *bool*]
	0 + 0 = 0, 0 + 1 = 1, …, 5 + 7 =12, …
	Eq? *x x* = *true*

$Eq?\ 0\ 1 = Eq?\ 0\ 2 = \ldots = false$

$count\ x\ empty = 0$

$count\ x\ (insert\ y\ m) = \textbf{if}\ Eq?\ x\ y\ \textbf{then}\ (count\ x\ m) + 1\ \textbf{else}\ count\ x\ m$

$cond_n\ true\ x\ y = x$

$cond_n\ false\ x\ y = y$

$cond_b\ true\ u\ v = u$

$cond_b\ false\ u\ v = v$

$cond_m\ true\ m\ m' = m$

$cond_m\ false\ m\ m' = m'$

给每个项指派整数的一种直观方式是给出最长可能归约序列的一个上界。例如，对于形式为 $cond_n\ M_1\ M_2\ M_3$ 的项，最长的归约序列是尽可能地归约所有这 3 个子项，然后（假定 M_1 归约到 *true* 或 *false*）再按条件规则归约，因此合理解释 $cond_n$ 的数值函数可以是

$$cond_n^{\mathcal{A}}\ x\ y\ z = 1 + x + y + z$$

按照同样的理由，除 *count*?以外，其他函数直截了当地解释如下：

$$+^{\mathcal{A}}\ x\ y = 1 + x + y$$

$$Eq?^{\mathcal{A}}\ x\ y = 1 + x + y$$

$$insert^{\mathcal{A}}\ x\ m = 1 + x + m$$

A^{nat} 不使用所有的自然数：

$$A^{nat} = A^{mset} = A^{bool} = \mathcal{N} - \{0, 1\}$$

其技术上的原因在下面可以明白。于是，每个常量符号都解释为 2。

对于 *count*，必须选择一个函数，使得每条 *count* 规则左部给出的值大于右部给出的值。第一条 *count* 规则仅要求函数值一定大于 2。从第二条规则可以得到条件

$$count^{\mathcal{A}} x\ (y + m) > 5 + x + y + 2 \times (count^{\mathcal{A}}\ x\ m)$$

通过检查可以猜想，$count^{\mathcal{A}}\ x\ \ m$ 的值必须指数依赖于它的第二个变元，因为对任何 $y \geq 2$，$count^{\mathcal{A}} x\ (y + m)$的值必须大于 $count^{\mathcal{A}}\ x\ m$ 的两倍。可以通过取

$$count^{\mathcal{A}}\ x\ m = x \times 2^m$$

来满足这个数值要求。

用这种解释，检验定理 2.22 的条件是否得到满足是容易的，从而可得出多重集合的重写规则是终止的。其中仅有的复杂情况是 *count*，但搞清楚

$$x \times 2^{(y+m)} - (5 + x + y + x \times 2^{(m+1)}) > 0$$

并不是十分困难的事。

2.6.4 临界对

下面将讨论两类合流的重写系统，一类必须是终止的，另一类可以不是。在 2.6.5 节讨论第一类，其重写规则相互之间无实质性的影响，因而讨论合流性时不受系统是否终止的影响。在 2.6.6 节讨论第二类，用终止性来分析规则之间的相互影响。在这两种情况下都挺重要的一个概念是**临界对**（*critical pair*），它是一对项，代表了重写规则之间的一种相互影响。另一个常用的概念是**局部合流**（*local confluence*），它是合流的一种较弱形式。因为局部合流用来启发临界对概念，因此先定义局部合流。

关系$\rightarrow$是局部合流的，如果 $N \leftarrow M \rightarrow P$ 蕴涵 $N \rightarrow\!\!\!\!\rightarrow \circ \leftarrow\!\!\!\!\leftarrow P$。这个关系严格弱于合流关系，因为仅假定当 M 由一步重写到 N 或 P 时有 $N \rightarrow\!\!\!\!\rightarrow \circ \leftarrow\!\!\!\!\leftarrow P$。2.6.6 节将表明，对于终止的重写系统来说，局部合流等价于合流。

例 2.20 局部合流但不合流的重写系统的一个简单例子如下：

$$a \to b, b \to a$$
$$a \to a_0, b \to b_0$$

a 和 b 都能归约到 a_0 和 b_0。但是不管是 a_0 还是 b_0 都不能归约到对方，因此合流性不成立。然而直观的分析可以看出该系统是局部合流的。该系统的一个图形表示在图 2.3。

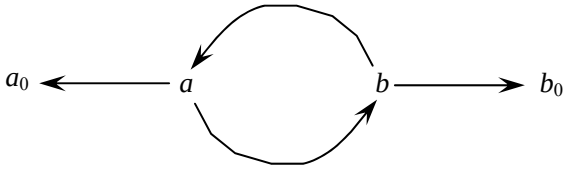

图 2.3 局部合流但不合流的归约

在本节的其余部分，考虑怎样判定一个重写系统$\mathcal{R}$是否局部合流。对于规则有限的重写系统，可以通过检查叫做“临界对”的有限种情况来说明局部合流。三种重要情况的项和归约的图形表示在图 2.4，2.5 和 2.6 分别给出。

假定在重写规则的有限集合$\mathcal{R}$ 下，项 M 可用两种不同的方式归约，即存在重写规则 $L \to R$ 和 $L' \to R'$及代换 S 和 S'，使得 M 在位置 $\vec{n}$ 的子项是 SL 并且 M 在位置 $\vec{m}$ 的子项是 $S'L'$（$L \to R$ 和 $L' \to R'$并非一定是不同的规则）。如果重新命名变量，使得 L 和 L'没有公共的变量，那么可以进一步假定 S 和 S'是包括了 L 和 L'中所有变量的同样代换。$\vec{n}$ 和 $\vec{m}$ 之间有几种可能的联系。

最简单的情况说明在图 2.4 中，$\vec{n}$ 和 $\vec{m}$ 不互为子序列，因而 SL 和 SL'不互为子项。此时，不管先归约哪一个都没有关系，因为总可以紧接着归约另一个而得到相同结果。

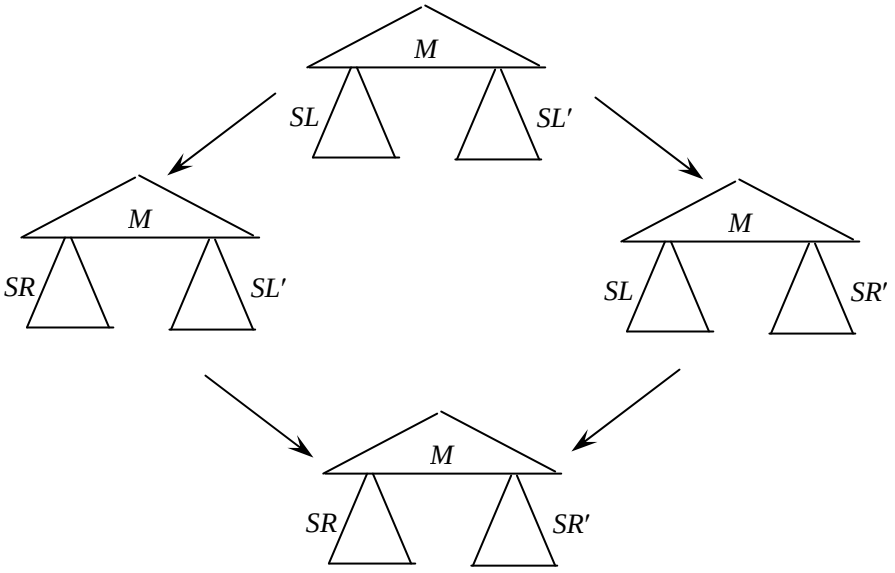

图 2.4 不相交的归约

第二种情况是一个子项包含另一个子项。为了表示上的简单，假设较大的子项就是 M 本身，即假定 $\vec{n}$ 是空串，使得 $M \equiv SL$，并假定 SL'是 SL 在位置 $\vec{m}$ 的子项。这时又可分成两种子情况。

第一种是“平凡”情况，SL'是 SL 的一个子项，并且 S 把 L 中的某个变量 x 用包含 SL' 的一个项代换即可。换句话说，如果把 SL 分成从 L 中来的符号和由 S 引入的符号，那么在位置 $\vec{m}$ 的子项仅包含由 S 引入的符号。这种情况描绘在图 2.5，此时如果 R 不含 x，那么从 SL 到 SR 的归约删除了子项 SL'，否则生成的项包含一个或多个 SL'的出现（取决于 R 中有多少个 x）。因为可以把每个 SL'的出现都归约到 SR'，因而可得到局部合流。

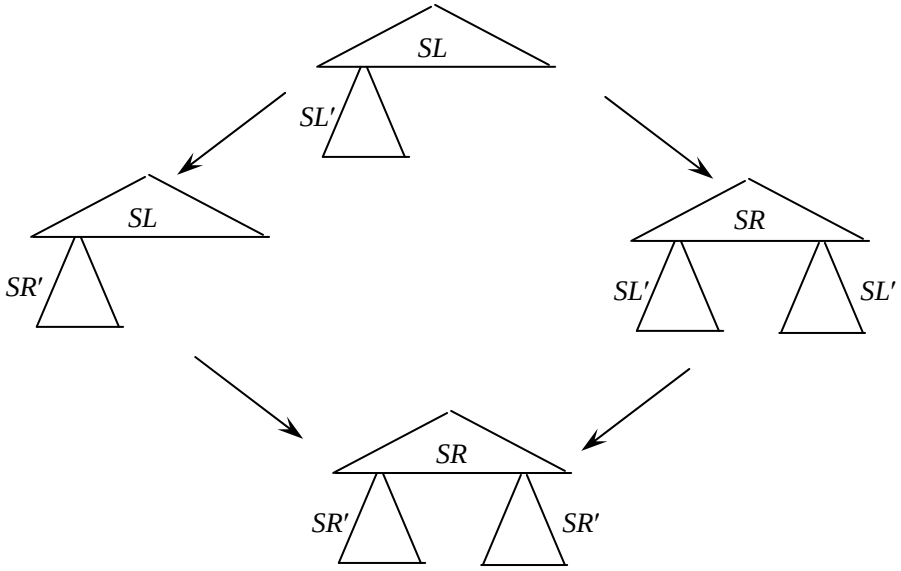

图 2.5 平凡的重叠

剩下的子情况必须仔细考虑。SL'在 SL 的位置 $\vec{m}$ 处并且 $L|\vec{m}$ 不是变量。显然，这种情况只有在 L'的主函数符号出现在 L 中并且和 $L|\vec{m}$ 的主函数符号一样时才会发生。如果使用第一条规则，可以得到 SR。如果使用第二条规则，可以得到项$[SR'/\ \vec{m}\ ]SL$。这可以用图 2.6 描绘。虽然碰巧会有 $SR \twoheadrightarrow \circ \twoheadleftarrow [SR'/\ \vec{m}\ ]SL$，但是没有理由认为这是一种普遍情况。

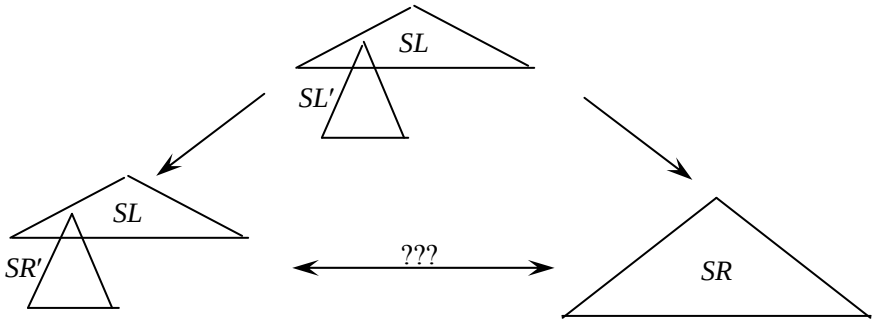

图 2.6 非平凡的重叠

例 2.21 本例使用关于表的两条重写规则

$cdr(cons\ x\ l) \to l$

$cons(car\ l')(cdr\ l') \to l'$

来举例说明三种重叠情况。

让一个项同时包含这两条规则的左部，就可以得到不相交的归约（图 2.4）。例如，对于项

$cond\ B\ \underline{(cdr(cons\ s\ l))}\ \underline{(cons(car\ l')(cdr\ l')\ )}$

可以任意选择这两条规则之一进行归约，可以独立地归约的两个子项都加了下划线。

对于在变量的代换实例处的重叠（图 2.5），可以使用项

$$\underline{cdr(cons\ x(\underline{cons(car\ l')(cdr\ l')}))}$$

把第一条规则左部的变量 l 用第二条规则的左部代换即可得到这个项。注意，如果归约较大的子项（即整个项），加下划线的较小的子项将原封不动。

不相交的归约和平凡重叠的归约差不多在每个重写系统中都可能有。但是，并非每个重写系统都会出现非平凡重叠（图 2.6），本例特地选择会出现这种情况的规则。把第一条规则

中的 x 和 l 分别用 $car\ l'$和 $cdr\ l'$代替，就得到含非平凡重叠的项

$$\underline{cdr(\underline{cons(car\ l')(cdr\ l')})}$$

如果把第一条规则用于较大的可归约子项（即完整的项），那么在较小的可归约子项中的 *cons* 被该规则删掉，因而第二条规则不再能使用。如果先把第二条规则用于较小的可归约子项，虽然在归约结果中 *cdr* 仍存在，但是第一条规则也不再能使用。幸好，这两个归约的结果是一样的，因而不难看出这两条规则是局部合流的。

由于规则出现非平凡重叠现象而使得系统不具备局部合流性的例子见例 2.17。项$(x + 0) + y$ 可以归约到 $x + y$ 或者 $x + (0 + y)$，这两个项都不能再归约。 □

对于两条规则 $L \to R$ 和 $L' \to R'$（可以是相同的规则），为了寻找非平凡的重叠，看起来似乎要考虑无穷多的代换 S。其实不然，对于每个 L，L 中的位置 $\vec{m}$，以及 L'，如果存在代换 S'使得 $S'L' \equiv S'L|_{\vec{m}}$，那么存在最一般的代换 S。最一般是基于这样的序定义：如果存在 S''使得 $S' = S''\circ S$，那么 S 比 S'更一般。使用合一算法，可以为每个 L'和非变量子项 $L|_{\vec{m}}$ 计算最一般的代换，或者确定不存在这样的代换。对很多重写规则对来说，这种计算是直截了当和简单的。

可以说非平凡重叠是一个三元组$\langle SL, SL', \vec{m}\rangle$，使得 SL'出现在 SL 的 $\vec{m}$ 处，并且 $L|_{\vec{m}}$ 不是变量。如果 S 是最一般的代换，使得$\langle SL, SL', \vec{m}\rangle$是一个非平凡重叠，那么序对$\langle SR, [SR'/\vec{m}]SL\rangle$叫做**临界对**（*critical pair*）。临界对的第一元是用规则 $L \to R$ 归约非平凡重叠三元组$\langle SL, SL', \vec{m}\rangle$的第一元 SL 的结果，临界对的第二元是将规则 $L' \to R'$用于 SL 在位置 $\vec{m}$ 的子项的结果。

例 2.21 的两条规则之间会产生两个临界对。将第一条规则左部的子项 *cons x l* 和第二条规则的左部 $cons(car\ l')(cdr\ l')$合一，得到的 SL（非平凡重叠三元组的第一元）是 $cdr(cons(car\ l)(cdr\ l))$，用这两条规则归约该项得到的临界对是$\langle cdr\ l, cdr\ l\rangle$。将第二条规则左部的子项 $cdr\ l'$和第一条规则的左部 $cdr(cons\ x\ l)$合一，得到的 SL 是 $cons(car\ (cons\ x\ l))(cdr(cons\ x\ l))$，用这两条规则归约该项得到的临界对是$\langle cons\ x\ l, cons\ (car\ (cons\ x\ l))\ l\rangle$。

应该记住的一个有用事实是，对于规则 $L \to R$ 和 $L' \to R'$，只有 L'的主函数符号 f 出现在 L 中，临界对才有可能出现。而且，如果在 L'中 f 应用到非变量的项，并且在 L 中的相应位置 f 也应用到非变量项，那么这两个非变量项的主函数符号必须相同。例如，如果有规则 $f(gxy) \to R'$，那么只有当规则 $L \to R$ 左部 L 中有子项 $f\,z$ 或形式为 $f(gPQ)$的子项时，才可能产生临界对。形式为 $f(h\ldots)$的子项不会引起临界对，因为不可能存在代换使得分别以 h 和 g 为主函数符号的两个项在语法上一样。

考虑同一条规则的两次使用引起的重叠是重要的。例如，只有一条重写规则 $f(f\,x) \to a$ 的系统不是局部合流的，因为有非平凡三元组$\langle f(f(f\,x)), f(f\,x), 1\rangle$，引起临界对$\langle fa, a\rangle$（项 $f(f(f\,x))$既可归约到 fa，又可归约到 a）。另外，任何规则 $L \to R$ 都引起重叠$\langle L, L, \varepsilon\rangle$，其中ε是空串，因此任何规则都引起相同项构成的临界对$\langle R, R\rangle$。这种从一条规则的两次应用引起的相同项构成的临界对，叫做**平凡临界对**（*trivial critical pair*）。

考虑规则左部完全相同的情况也是重要的，因为重写系统可以有两条规则 $L \to R$ 和 $L \to R'$，这时$\langle R, R'\rangle$是临界对。

命题 2.23 重写系统$\mathcal{R}$ 是局部合流的，当且仅当对每个临界对$\langle M, N\rangle$有 $M \to\!\!\!\!\to_{\mathcal{R}} \circ \leftarrow\!\!\!\!\leftarrow_{\mathcal{R}} N$。

如果一个有限的重写系统$\mathcal{R}$是终止的，那么该命题就给出一个可用于判定$\mathcal{R}$是否局部合流的算法。2.6.6 节将讨论这个算法。

证明 从左到右的蕴涵是简单明了的。另一个方向的证明仿照用于启发临界对定义的推理。假定 M 可以用 $L \to R$ 和 $L' \to R'$归约到两个不同的子项 N 和 P，那么在 M 的相关子项 SL 和 SL'间存在三种可能的关系。如果这些子项不相交或是平凡的重叠，那么由本小节前面

的讨论可以知道 $N \twoheadrightarrow \circ \twoheadleftarrow P$。剩下的情况是 N 和 P 含一个临界对的代换实例。因为 M 中没有参与 $M \to N$ 和 $M \to P$ 归约的符号在确定是否有 $N \twoheadrightarrow \circ \twoheadleftarrow P$ 时不起作用，因此可以简化记号，假定$\mathcal{R}$有一个临界对$\langle SR, [SR'/\ \vec{m}\ ]SL\rangle$，使得 $N \equiv S'(SR)$并且 $P \equiv S'([SR'/\ \vec{m}\ ]SL)$。由该命题的假设，有 $SR \twoheadrightarrow \circ \twoheadleftarrow [SR'/\ \vec{m}\ ]SL$。于是，因为 N 和 P 的归约都是应用到 SR 和$[SR'/\ \vec{m}\ ]SL$ 的代换实例，因此有 $N \twoheadrightarrow \circ \twoheadleftarrow P$。 □

2.6.5 左线性无重叠重写系统

如果一个重写系统没有非平凡的临界对，则称它是无重叠的。换句话说，形成临界对的唯一方式是同一条规则 $L \to R$ 用两次以得到$\langle R, R\rangle$。因为这种平凡的临界对不会影响合流，也许会认为无重叠的重写系统是合流的。但实际不是这样，下面的例 2.22 是一个反例。直观上，这个问题的出现是因为重写规则允许用一种形式的项代替另一种形式的项，因而可能会有这样的规则，它们在经过几步重写后才出现重叠。但是，如果对规则的左部做适当的限制，那么无临界对时可以保证合流。表示这种限制的术语是左线性：一条规则 $L \to R$ 是**左线性**（*left-linear*）的，如果任何变量在 L 中的出现不超过一次的话。一个重写系统是左线性的，如果它的每一条规则都是左线性的。

本小节的主要结论是，左线性、无重叠的重写系统是合流的。该事实的最早证明可能是为了组合逻辑（见例 2.24），它可以推广到任意的左线性和无重叠的系统。先看一个无重叠的非左线性系统不合流的例子。

例 2.22 下面关于有穷数和“无穷”数的重写系统没有临界对，因此它局部合流，但它不是合流的。

$$\infty \to S\ \infty$$
$$Eq\ ?\ x\ x \to true$$
$$Eq\ ?\ x\ (S\ x) \to false$$

直观上，除了“无穷数” ∞ 是它本身的后继外，两条 *Eq* ?规则是很精美的。但是，对于∞，

$$Eq\ ?\ \infty\ \infty \to true$$
$$Eq\ ?\ \infty\ \infty \to Eq\ ?\ \infty\ (S\ \infty) \to false$$

出现一个项有两个不同的范式，*true* 和 *false*。

合流在这种情况下会失败的直观原因有点难以捉摸。该系统中唯一可能出现临界对的情况是，存在代换 R 使得 $R\ (Eq\ ?\ x\ x) \equiv R\ (Eq\ ?\ x\ (S\ x))$。任何这样的代换 $R = [M/x]$必须使 $M \equiv SM$。因为不存在项 M 在语法上会等于它的后继 SM，因此这样的代换 R 不存在。可是，因为项∞可以归约到它自己的后继，于是这条规则的出现就相当于代换实例 $Eq\ ?\ \infty\ \infty$与 $Eq\ ?\ \infty\ (S\ \infty)$有一个重叠。 □

下面的命题不加证明地给出。

命题 2.24 如果$\mathcal{R}$是左线性的且无重叠，那么$\twoheadrightarrow_{\mathcal{R}}$是合流的。

例 2.23 下面关于表项的重写系统是左线性的且无重叠，因而是合流的。

$$car\ (cons\ x\ l) \to x$$
$$cdr\ (cons\ x\ l) \to l$$
$$isempty?\ nil \to true$$
$$isempty?\ (cons\ x\ l) \to false$$

然而，如果加入非左线性规则

$$cons\ (car\ l)\ (cdr\ l) \to l$$

则左线性被破坏。从例 2.21 已经明白，这条非线性规则和 *cdr* 规则的相互影响并非很糟糕，它和 *car* 的重叠也是良性的。但是，它和第二条 *isempty*?规则的相互作用引起局部合流失败。因为 *isempty*? (*cons* (*car l*) (*cdr l*))可归约到 *false*，也可归约到 *isempty*? (*l*)，它们是不同的范

式。 □

例 2.24 无类型的组合逻辑有下面的代数规范

sorts : β

fctns : $S, K, I : \beta$

$ap : \beta \to \beta \to \beta$

eqns : $ap\ (ap(ap\ S\ x)\ y)z = ap\ (ap\ x\ z)(ap\ y\ z)$

$ap\ (ap\ K\ x)\ y = x$

$ap\ I\ x = x$

很容易看出，如果从左到右定向这三个等式公理，就得到一个左线性且无重叠的重写系统。因此组合逻辑是合流的。为方便起见，把 *ap* 写成中缀算符，甚至省略 *ap*。在本例的其余部分，用 $x \bullet y$ 代替 $ap\ x\ y$，并且在省略括号时，•是左结合的。

不难看出，这个重写系统是不终止的。例如，考虑项 $S \bullet I \bullet I$，很容易看出

$$(S \bullet I \bullet I) \bullet x \twoheadrightarrow x \bullet x$$

于是有

$$(S \bullet I \bullet I) \bullet (S \bullet I \bullet I) \twoheadrightarrow (S \bullet I \bullet I) \bullet (S \bullet I \bullet I)$$

研究组合逻辑的一个重要动力是，任何无类型的λ项都可以翻译成组合逻辑的项。满足（无类型）组合逻辑公理的代数叫做（无类型）组合代数。 □

2.6.6 局部合流、终止和合流之间的联系

在上一小节已经看到，每个左线性且无重叠的重写系统是合流的，与它终止与否无关。本节考虑非左线性但终止的系统的合流性的一个充分条件，：终止且局部合流的系统是合流的。这个事实完全不依赖于代数项的结构，因而可以用到其他系统，包括λ演算归约和很多图重写规则。

命题 2.25 令$\mathcal{R}$ 是终止的重写系统，那么$\mathcal{R}$ 是合流的当且仅当它是局部合流的。

证明 已经知道，如果每当 $N \leftarrow_{\mathcal{R}} M \twoheadrightarrow_{\mathcal{R}} P$ 就有 $N \twoheadrightarrow_{\mathcal{R}} \circ \leftarrow_{\mathcal{R}} P$，那么$\mathcal{R}$ 对 M 是合流的。因为$\mathcal{R}$是终止的，可令项 M 的定额（*norm*）$|M|$是从 M 开始的最长归约序列。然后基于定额来归纳证明$\mathcal{R}$对每个项都是合流的。下面用$\twoheadrightarrow$代替$\twoheadrightarrow_{\mathcal{R}}$。归纳基础（$|M|=0$）是简单的。

假定 $M \twoheadrightarrow N$，$M \twoheadrightarrow P$ 并且对每个 $|Q|<|M|$的 Q，$\mathcal{R}$ 对 Q 是合流的。如果归约 $M \twoheadrightarrow N$ 或 $M \twoheadrightarrow P$ 的长度为 0，那么显然 $N \twoheadrightarrow \circ \twoheadleftarrow P$。于是假定 $M \to N_1 \twoheadrightarrow N$ 并且 $M \to P_1 \twoheadrightarrow P$。由局部合流，存在某个项 Q 使得 $N_1 \twoheadrightarrow Q$ 并且 $P_1 \twoheadrightarrow Q$。由归纳假设，存在某个项 R 使得 $N \twoheadrightarrow R$ 并且 $Q \twoheadrightarrow R$。因为 $P_1 \twoheadrightarrow Q \twoheadrightarrow R$ 并且 $P_1 \twoheadrightarrow P$，再次应用归纳假设可以得到项 S 使得 $R \twoheadrightarrow S$ 并且 $P \twoheadrightarrow S$。把这些已列出的归约组合在一起，可以看出 N 和 P 都归约 S，见图 2.7。命题得证。

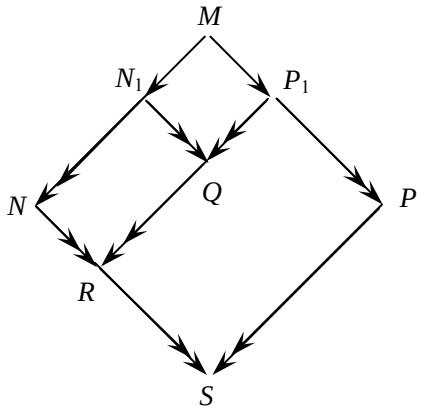

图 2.7 由局部合流和终止证明合流的示意图

例 2.25 下面条件表达式的重写规则是局部合流的并且是终止的，但不是左线性的。

if *true* **then** x **else** $y \to x$

if *false* **then** x **else** $y \to y$

if u **then** x **else** $x \to x$

该系统的终止性是直截了当可看出来的，因为所有的归约都缩短项的长度。这儿有两个临界对，它们都和第三个非左线性规则有关。第一个临界对是通过用第一和第三条规则来归约项

if *true* **then** *x* **else** *x*

而得到，是$\langle x, x\rangle$，显然它不会引起任何问题。第二个临界对还是$\langle x, x\rangle$。 □

从命题 2.19 知道，通过检查每个临界对是否合流可以判断一个重写系统是否局部合流。这个检查通常叫做 *Knuth-Bendix* 测试，它是 *Knuth* 和 *Bendix* 提出的一个算法的一部分，该算法叫做 *Knuth-Bendix* 完备化过程。叫做完备化过程的原因是，它不仅可用来测试一组终止的重写规则$\mathcal{R}$是否局部合流，而且在$\mathcal{R}$不合流时它会添加规则，添加的规则对原来的规则集是守恒的。守恒是指，如果 *Knuth-Bendix* 算法产生$\mathcal{R}' \supseteq \mathcal{R}$，那么$\mathcal{E}_{\mathcal{R}'}$和$\mathcal{E}_{\mathcal{R}}$确定同样的代数理论。该完备化过程需要一个算法来确定：在某个良基代数中，任何项 M 和 N 之间是否有形式为 $M \succ N$ 的不相等性。

该完备化过程由测试每条重写规则 $L \to R$ 是否 $L \succ R$ 开始。如果是，则该重写系统终止，然后可以测试局部合流性。通过对每个临界对$\langle M, N\rangle$，寻找从 M 和 N 开始的所有可能归约，以发现是否有 $M \twoheadrightarrow \circ \twoheadleftarrow N$，由此完成局部合流性的测试。因为该重写系统终止，因而可能的归约仅有限多种。如果所有的临界对都收敛，那么该算法终止，并且得知该系统是合流的和终止的。

如果对某个临界对$\langle M, N\rangle$不具备 $M \twoheadrightarrow \circ \twoheadleftarrow N$，那么由临界对的定义知道，存在一个项，它既可归约到 M，又可归约到 N。因为由等式推理，基于$\mathcal{E}_{\mathcal{R}}$是可以证明 M 和 N 相等，因而增加 $M \to N$ 或 $N \to M$ 为一个重写规则是有逻辑意义的。如果 $M \succ N$ 或 $N \succ M$，那么该完备化过程加上相应的规则（$M \to N$ 或 $N \to M$）并且继续进行局部合流的测试。但是，如果 M 和 N 不可比较（既没有 $M \succ N$，也没有 $N \succ M$），那么就不能保证加入的规则保终止性，这就导致完备化过程失败。另外，完备化过程可能不终止。下面的例子既用来解释完备化过程，同时也用来说明完备化过程可能不终止。

例 2.26 本例在加入任意有穷条规则后，既不能产生一个合流的系统，也不会到达不可以再加入规则的局面。

考虑下面由等式定向得到的两条重写规则，它们分别叫做同态性（用 H 表示）和结合性（用 A 表示）。

$$fx + fy \to f(x + y) \qquad (H)$$

$$(x + y) + z \to x + (y + z) \qquad (A)$$

通过构造一个良基代数$\mathcal{A}$，可以看出由这两条规则构成的重写系统是终止的。对$\mathcal{A}$的唯一载体，可以使用大于 2 的自然数。为了使规则(A)减小（即右部在$\mathcal{A}$中的解释比左部的解释小），必须为+选择某个无结合性的解释。指数运算是最简单而又熟悉的算术运算之一，可以使用

$$+^{\mathcal{A}}(x, y) = y^x$$

$$f^{\mathcal{A}}(x) = x^2$$

简单的计算表明，在这个代数中，两条规则都是减小的。

由(H)的左部 $fx + fy$ 和(A)左部的子项$(x + y)$合一得到$(fx + fy) + z$，然后用两种方式归约它得到仅有的临界对：

$$\langle f(x + y) + z,\quad fx + (fy + z)\rangle$$

它们都是范式但又不相同，为了保证完备性，需要加入一条重写规则。由于它们的权重分别是

$$f^{\mathcal{A}}(x +^{\mathcal{A}} y) +^{\mathcal{A}} z = \quad z^{(y^x)^2} \quad = \quad z^{(y^{2x})}$$

$$f^{\mathcal{A}}x +^{\mathcal{A}} (f^{\mathcal{A}}y +^{\mathcal{A}} z) = \quad \left(z^{y^2}\right)^{x^2} \quad = \quad z^{(y^2)(x^2)}$$

比较当 $x, y, z \geq 3$ 时这些表达式的数值可以看出，如果加上从第一个项到第二个项的重写规则，系统仍能保证终止。该规则是通用模板

$$f^n(x+y)+z \to f^n x + (f^n y + z) \qquad (A)_n$$

的一个实例。$(A) \equiv (A)_0$。

继续完备化过程，通过将(H)的左部 $fx+fy$ 和$(A)_{n-1}$（$n>1$）的子项 $x+y$ 合一得到 $f^{n-1}(fx+fy)+z$，然后得到形式为

$$\langle f^n(x+y)+z,\quad f^n x + (f^n y + z)\rangle$$

的临界对。它们仍然都是范式但又不相同，为了保证完备性，仍然需要加入一条重写规则。由于它们的权重分别是

$$(f^{\mathcal{A}})^n (x +^{\mathcal{A}} y) +^{\mathcal{A}} z = \quad z^{((y^x)^{2n})} \quad = \quad z^{(y^{2nx})}$$

$$(f^{\mathcal{A}})^n x +^{\mathcal{A}} ((f^{\mathcal{A}})^n y +^{\mathcal{A}} z) = \quad \left(z^{(y^{2n})}\right)^{(x^{2n})} \quad = \quad z^{(y^{2n})(x^{2n})}$$

导致加上重写规则$(A)_n$，即通用模板的下一个实例。

由此看出，完备化过程不会终止。 □

2.6.7 代数数据类型的应用

在代数规范的设计、分析和应用中，重写规则至少有两种不同方式的应用。如果一个等式系统能够定向（完备化）成一个合流且终止的重写系统，那就给出了一种非常有用的方式来确定两个项是否可证明为相等。如果它们被归约到不同的范式，那么它们就不能证明为相等。即重写技术可用于自动定理证明。重写系统的另一种潜在使用是作为数据类型的快速原型实现。需要注意的是，虽然归约可直截了当用于对规范进行调试，但是归约方式的实现对实际的程序设计来说效率太低。另外，因为对初始代数的理解主要是基于语法的（基于可证明性），因此重写系统对初始代数的分析和理解也是非常有用的。

下面用带错误值的表的代数规范来说明重写系统的这两种使用。首先，通过对第一个规范测试局部合流性可以发现不一致性。其次，可以通过完备化有向等式到一个合流的重写系统来刻画修改后的规范的初始代数。

2.5.3 节的表 2.3 和表 2.4 给出了错误值的朴素处理办法，结果出现了不一致。考虑对它们进行从左到右定向得到的重写系统。很容易看出，该系统是终止的，因为每条规则都缩短表达式的长度。在为完整的系统检查合流性前，先分别查看表 2.3 和表 2.4 的临界对。在表 2.4 中，只有规则

$$cons\ error_a\ l = error_l \quad 和$$

$$cons\ x\ error_l = error_l$$

会给出一个临界对。因为只要把 x 和 l 用错误值代替就可使左部合一，因此临界对是$\langle error_l, error_l \rangle$。显然它不会导致局部合流失败。

现在考察表 2.4 中可能和表 2.3 的规则重叠的规则，第一个选择是 $cons\ error_a\ l \to error_l$，因为表 2.3 有含 *cons* 的规则 $car(cons\ x\ l) \to x$。使用这两条规则得到的第一个非平凡临界对是$\langle error_a, car\ error_l \rangle$。这个临界对不会引起问题，因为 $car\ error_l \to error_a$。

下一个临界对由规则 $cons\ error_a\ l \to error_l$ 和规则 $cdr(cons\ x\ l) \to l$ 的子项合一得到。合一得到的项是 $cdr(cons\ error_a\ l)$，它可以分别归约到表变量 l 和 $cdr\ error_l$。后者只能归约到 $error_l$，从而合流失败。从上面为合流所做的归约可以知道，表 2.3 和表 2.4 给出的等式规范是可以证明等式 $error_l = l\ [l : list]$。该等式表明，在任何满足该规范的代数中，所有的表必须相等。不难看出，其他类别的类似等式也可证，因而知道该规范是不一致的。

有错误元素并经修改后的表规范在表 2.5。将等式从左到右定向，得到一个重写系统。

在检查临界对之前，先检查系统的终止性，它将为完备化过程增加规则时的定向建立基础。

可以使用良基代数$\mathcal{A}$来证明该重写系统的终止性：$\mathcal{A}$的载体都是正整数集合，并且将每个常量都解释为 1。该解释是对各种形式的项可执行的最大归约步数的一种估计，如例 2.19 所描述的那样。

$cons^{\mathcal{A}}\ x\ l = x + l$

$car^{\mathcal{A}}\ l = 2 \times l + 5$

$cdr^{\mathcal{A}}\ l = 2 \times l + 5$

$isempty?^{\mathcal{A}}\ l = l + 8$

$cond^{\mathcal{A}}\ u\ x\ y = u + x + y + 1$

$OK^{\mathcal{A}}\ x = x + 8$

检查从表 2.5 规范导出的每条规则都满足 $L > R$ 是直截了当的。这里有几个临界对，它们都涉及到有关 *cons x error* 和 *cons error l* 的规则。要检查的第一个有重叠的项是

car(*cons error l*)

如果归约较外的项，将得到

cond (*OK error*)(*cond* (*OK l*) *error error*) *error*

而归约较内的项将得到 *error*。这是一个非平凡的临界对，但是两个项都可归约到 *error*，因而不引起问题。

下一个有重叠的项是

car(*cons x error*)

它给出一个看起来差不多的临界对

⟨*cond* (*OK x*)(*cond* (*OK error*) *x error*) *error*, *car error*⟩

第二个项显然可以归约到 *error*，而对于第一个项，通过归约较内的那个 *cond* 得到

cond (*OK x*) *error error*

在良基代数$\mathcal{A}$中，很容易看出

cond (*OK x*) *error error* > *error*

为了保终止性，加上规则

cond (*OK x*) *error error* → *error*

因为从该规范可证该规则的左部和右部相等，因此加上该规则不会改变项之间可证的相等性。

有关 *cdr* 和 *isempty*?的临界对的情况类似。对这两个函数，第一个临界对很容易收敛，使用为 *car* 加入的新规则，第二个临界对也收敛。

最后的临界对出现在把 *OK* 用于 *cons* 的情况。对于 *OK*(*cons error l*)，两种情况下都得到 *false*。然而，对于

OK(*cons x error*)

必须加上规则

cond (*OK x*) *false false* → *false*

于是，在表 2.5 的规范定向后的重写系统上添加两条新规则就可得到一个合流的重写系统。

一个规范的初始代数由基项的等价类组成，模可证的相等性。如果该规范有对应的合流且终止的重写系统，那么可通过归约来检查两个项的相等是否可证，因此可以使用重写系统来理解规范的初始代数。特别是，因为每个等价类正好只有一个范式，因此初始代数同构于这样一个代数，它的载体正好是每个类别的基范式集合。于是对表 2.5 的规范来说，在这样的初始代数中，原子是 *a*, *b*, *c*, *d*, …, $error_a$，布尔值是 *true*, *false*, $error_b$，表的所有可能形式是 *nil*, $error_l$ 或 *cons A L*，其中 *A* 和 *L* 都不含 *error*。于是该规范正好为每个载体加了一个 *error* 元素。

习　　题

2.1 用归纳法证明，如果 $M \in Terms^s(\Sigma, \Gamma)$并且$\Gamma'$是包含$\Gamma$的类别指派，那么 $M \in Terms^s(\Sigma, \Gamma')$。

2.2 对于例 2.3，让 $M*N$ 作为$*MN$ 的语法美化。

(a) 假定$\eta(x) = 1$ 并且$\eta(y) = 2$，计算$[\![(x+1)*2]\!]\eta$。

(b) 证明对任何环境η，有$[\![(x+ y)* z]\!]\eta = [\![(x*z) +(y*z)]\!]\eta$。

2.3 对于例 2.4，

(a) 假定$\eta(x) = 3$ 并且$\eta(s) = \langle 2, 1\rangle$，计算$[\![push\ x\ (pop\ s)]\!]\eta$。

(b) 证明对任何把 $s : stack$ 映射到非空串的环境η，有$[\![push\ (top\ s)(pop\ s)]\!]\eta = [\![s]\!]\eta$。如果$\eta(s) = \varepsilon$ 会有什么变化？

2.4 证明代换规则

$$\frac{M = N\ [\Gamma, x{:}s]}{[P/x]M = [P/x]N\ [\Gamma]} \quad P \in Terms^s(\Sigma, \Gamma)$$

是语义上可靠的。换句话说，假定 $M, N \in Terms^{s'}(\Sigma, \Gamma, x : s)$，$P \in Terms^s(\Sigma, \Gamma)$，并且对满足 $\Gamma, x : s$ 的任何环境η'，有$[\![M]\!]\eta' = [\![N]\!]\eta'$。使用代换引理证明，对满足$\Gamma$的任何环境$\eta$，有$[\![[P/x]M]\!]\eta = [\![[P/x]N]\!]\eta$。

2.5 证明等式 $x = x\ [x : s]$是永真的。具体说，令$\Sigma = \langle S, F\rangle$是含 $s \in S$ 的任何基调，令$\mathcal{A}$是任意的Σ代数，证明，对任何有类别指派$[x : s]$的环境η，有

$$\mathcal{A}, \eta \vDash x = x\ [x : s]$$

别忘了考虑 A^s=∅的可能性。

2.6 证明代数$\mathcal{A}$是平凡的，当且仅当任何载体都是空集或是仅有一个元素的集合。

2.7 令$\mathcal{A} = \langle A, \bullet^{\mathcal{A}}, K^{\mathcal{A}}\rangle$是一单类别代数，对应的基调有一个二元算符$\bullet$和一个常量符号 K，该二元算符将写成中缀形式。假定$\mathcal{A} \vDash (K\bullet x)\bullet y = x$，其中变量的类别不必指明，因为只有一个类别。

(a) 证明，如果$\mathcal{A}$多于一个元素，那么$\mathcal{A}$必定无限（提示：证明，如果$\mathcal{A}$有限，那么存在某个 $a \in A$ 使得 $K^{\mathcal{A}} \bullet^{\mathcal{A}} a = K^{\mathcal{A}}$）。

(b) 证明，如果$\mathcal{A}$ 多于一个元素，那么$\bullet$不是可结合的。

2.8 解释 $Terms(\Sigma, \Gamma)$作为代数的定义，并检查它确实满足作为代数的条件。

2.9 使用引理 2.5 证明，对于项代数 $Terms(\Sigma, \Gamma)$，$Terms(\Sigma, \Gamma) \vDash M = N\ [\Gamma]$当且仅当 M 和 N 语法上完全相同。

2.10 在应用多个变量的代换 S 到项 P 时，可以用 Sx_i 同时代换 P 中的每个 x_i。具体讲，如果 P 有变量 $x_1, \ldots, x_k$。不是先用 Sx_1 代换 x_1，然后再用 Sx_2 代换 x_2，等等，而是同时完成代换。同时代换映射 $x_i \mapsto M_i$ 通常写成$[M_1, \ldots, M_k/x_1, \ldots, x_k]$。找出一组表达式 M，N 和 P，使得$[M, N/x,y]P \neq [M/x]\ ([N/y]P)$。

2.11 证明，对任何代数$\mathcal{A}$，所有在代数$\mathcal{A}$ 中成立的等式集合 $Th\ (\mathcal{A})$是一个语义理论。

2.12 证明，等式集合$\mathcal{E}$ 是语义上一致的，当且仅当存在一个非平凡代数$\mathcal{A} \vDash \mathcal{E}$。

2.13 证明

$$\frac{M\ =\ N\ [\Gamma, x{:}s] \quad P\ =\ Q\ [\Gamma']}{[P/x]M\ =\ [Q/x]N\ [\Gamma \cup \Gamma']} \quad P, Q \in Terms^s(\Sigma, \Gamma')$$

是一个导出规则。注意，要考虑 x 出现在Γ'，P和Q中的情况。注意，在写$[P/x]M = [Q/x]N\ [\Gamma \cup \Gamma']$时，假定$\Gamma \cup \Gamma'$是良形的类别指派，使得没有任何变量会同时出现在两个不同的类别中。

2.14 如果 S 把每个 $x : s \in \Gamma$ 映射到 $Sx \in Terms^s(\Sigma, \Gamma')$，则称 S 叫做(Γ, Γ')代换。证明，规则

$$\frac{M = N\ [\Gamma]}{SM = SN\ [\Gamma']} \qquad S \text{ 是一个}(\Gamma, \Gamma')\text{代换}$$

是个导出规则（提示：见习题 2.10）。

2.15 考虑表 2.6 的多重集合、自然数和布尔值的规范。下面(b)和(c)使用变量 $x, y : nat$ 和 $m : mset$。

(a) 证明等式

$count\ 3\ (insert\ 5\ (insert\ 3\ (empty))) = 1$

(b) 对任何数码 a, b 和 c，下面的等式可证。

$count\ a\ (insert\ b\ (insert\ c\ m)\ ) = count\ a\ (insert\ c\ (insert\ b\ m)\ )$

(c) 找出一个代数，它满足该规范的所有等式，但是不满足等式

$insert\ x\ (insert\ y\ m) = insert\ y\ (insert\ x\ m)$

由此得出该等式不可证。

2.16 树的代数规范由表 2.7 给出。直观上，类别 *tree* 由二权树构成，原子存在叶结点上（如果 l 是叶结点，那么 *lable* l 是存放在该结点上的原子）。证明该规范的下列推论。

(a) $lsub\ (node\ (rsub\ (node\ (leaf\ a)\ (leaf\ b)))(leaf\ c)) = leaf\ b$

(b) $label\ (lsub\ M) = label\ (rsub\ M)$，其中 $M \equiv node\ (leaf\ a)\ (leaf\ a)$

(c) 试说明，如果 M 是表 2.7 基调上的不含变量的项，它的类别是 *tree*，并且由函数符号 *leaf* 或 *node* 开头，那么

$is\_leaf?\ M = is\_leaf?\ (cond\ (is\_leaf?\ M)\ (leaf\ a)\ M)$

可证。

(d) 构造满足本规范的一个代数，用来说明下面的等式是不可能从本规范证明的。

$label\ (node\ t\ t') = label\ (node\ t'\ t)$　　其中 $t, t' : tree$

(e) 试说明，如果项 M: *tree* 不包含函数符号 *leaf*，那么等式

$lsub\ (leaf\ x) = M\ [\Gamma, x : atom]$

不可证（提示：见例 2.10）。

表 2.7 树的代数规范

sorts :	$atom, tree, bool$
fctns :	$a, b, c, d, \ldots : atom$
	$leaf : atom \to tree$
	$label : tree \to atom$
	$node : tree \times tree \to tree$
	$true, false : bool$
	$is\_leaf : tree \to bool$
	$lsub : tree \to tree$
	$rsub : tree \to tree$
	$cond_a : bool \times atom \times atom \to atom$
	$cond_b : bool \times bool \times bool \to bool$
	$cond_t : bool \times tree \times tree \to tree$
eqns :	$[x, y : atom; t, t' : tree; u, v : bool]$
	$is\_leaf?\ (leaf\ x) = true$
	$is\_leaf?\ (node\ t\ t'\ ) = false$

$label\ (leaf\ x) = x$

$lsub\ (node\ t\ t') = t$

$rsub\ (node\ t\ t') = t'$

$cond_a\ true\ x\ y = x$

$cond_a\ false\ x\ y = y$

$cond_b\ true\ u\ v = u$

$cond_b\ false\ u\ v = v$

$cond_t\ true\ t\ t' = t$

$cond_t\ false\ t\ t' = t'$

2.17 证明，最小模型完备性蕴涵演绎完备性。由此得出，在有规则（*nonempty*）的情况下，没有空载体的多类别代数有可靠性和演绎完备性。

2.18 完成定理 2.11 的证明。

2.19 证明两个同态的合成是一个同态。

2.20 使用引理 2.13 证明同态和等式之间的下列联系。

(a) 从$\mathcal{A}$到$\mathcal{B}$ 的满射同态 h 是指，对任意 $b \in B^s$，存在 $a \in A^s$，使得 $h^s(a) = b$。证明，如果 h 是这样的满射，那么 $Th(\mathcal{A}) \subseteq Th(\mathcal{B})$。

(b) 如果$\mathcal{A}$和$\mathcal{B}$同构，那么 $Th(\mathcal{A}) = Th(\mathcal{B})$。

2.21 证明，从$\mathcal{A}$到$\mathcal{B}$有一个满射同态，当且仅当对某个同余关系~，$\mathcal{B}$和$\mathcal{A}$/~同构。使用习题 2.20 的结果，可以得出对任何$\mathcal{A}$上的同余~，有 $Th(\mathcal{A}) \subseteq Th(\mathcal{A}/\sim)$。

2.22 令$\mathcal{A}_1$ 是例 2.14 自然数基调Σ_1 标准的和初始的代数，令$\mathcal{A}_2$ 是加上无数个整数的代数（描述在例 2.14 中），令$\mathcal{A}_3$ 是自然数模 k 的代数（对某个 k，也描述在例 2.14 中）。证明，对于$\mathcal{A}_2 \to \mathcal{A}_1$ 和$\mathcal{A}_3 \to \mathcal{A}_1$，存在多个同态，或者不存在同态，由此证明$\mathcal{A}_2$ 和$\mathcal{A}_3$ 都不是初始的。

2.23 考虑单类别 *nat* 的自然数基调Σ_2，函数符号有 $0 : nat, S : nat \to nat$ 和$+, * : nat \times nat \to nat$。令$\mathcal{E}$是等式集合

$x + 0 = x$

$x + S(y) = S(x + y)$

$x * 0 = 0$

$x * (Sy) = x * y + x$

在此省略了类别指派，因为只有一个类别。令$\mathcal{C}$是满足$\mathcal{E}$的所有Σ_2 代数构成的代数类。

(a) 对该基调上的任何闭项 M，从$\mathcal{E}$证明，对某个自然数 k，$M = S^k0$。

(b) 用可靠性（定理 2.6）和(a)的结论来证明，如果把从$\mathcal{E}$ 可证明为相等的项看成等价，那么闭项的每个等价类正好只有一个形式为 S^k0 的项。

2.24 在表 2.2 说明的 *set* 数据类型中，证明等式

$union\ s\ s' = union\ s'\ s\ [s : set, s' : set]$

在初始代数中不成立。可以假定重写规则都是将等式公理从左到右定向而得到，并且它们是合流的（也就是说不用在此证明这一点）。合流性蕴涵：两个项可证明为相等当且仅当它们归约到同一个项。

2.25 本习题分析表 2.6 给出的代数规范，增加了一个交集运算，并由下面两个公理定义该运算的性质。

$intersect\ empty\ m = empty$

$intersect(insert\ x\ m)\ m' =$ **if** $ismem?\ x\ m'$ **then** $insert\ x\ (intersect\ m\ m')$ **else** $intersect\ m\ m'$

(a) 证明 *empty* 和 *insert* 是多重集合的构造符。

(b) 证明，等式

$$insert\ x\ (insert\ y\ m) = insert\ y\ (insert\ x\ m)$$

在初始代数中不成立。你可以像习题 2.24 那样假定重写规则及合流性。

(c) 通过构造一个代数来证明，如果加入等式

$$insert\ x\ (insert\ y\ m) = insert\ y\ (insert\ x\ m)$$

作为公理，该规范仍然是一致的。

2.26 像 2.5.3 节那样，用 *OK* 函数设计有显式错误处理的树的规范。没有错误值的规范在表 2.7。

2.27 考虑代数项 $f\ (gxy)(f\ (gu(huv)))(f\ (g\ (gxy)z))$

(a) 写出在位置 2,1,2 的子项。

(b) 用自然数序列描述 gxy 的位置。

2.28 证明，如果$\mathcal{R}$含规则 $L \to R$，其中 L 是一个变量或者 R 含的一个变量在 L 中不出现，那么系统$\mathcal{R}$不终止。

2.29 令$\mathcal{R}$是任意的重写系统，令$\mathcal{R}^{op}$是反向规则 $R \to L$（$L \to R$ 在$\mathcal{R}$中）的集合。证明重写系统$\mathcal{R} \cup \mathcal{R}^{op}$合流但不终止。

2.30 证明引理 2.15。

2.31 考虑下面关于命题逻辑公式的重写系统:

$$\neg\neg\, x \to x$$
$$\neg\, (x \vee y) \to (\neg x \wedge \neg y)$$
$$\neg\, (x \wedge y) \to (\neg x \vee \neg y)$$
$$x \vee (y \wedge z) \to (x \vee y) \wedge (x \vee z)$$
$$(y \wedge z) \vee x \to (y \vee x) \wedge (z \vee x)$$

该系统类似于例 2.18 的系统，不过是把公式翻译成合取范式而不是析取范式。证明该系统终止。可以定义和使用一个良基代数，它的载体是自然数的某个子集。

2.32 使用一个良基代数，证明下面的重写系统终止。

(a) $0 + x \to x$

$(Sx) + y \to S\ (x + y)$

(b) 把下面规则加到(a)的规则中。

$0 * x \to 0$

$(Sx) * y \to (x * y) + y$

(c) 把下面规则加到(a)和(b)的规则中。

$fact\ 0 \to 1$

$fact\ (Sx) \to (Sx) * fact\ (x)$

2.33 用例 2.17 的基调，考虑重写系统

$0 + x \to x$

$(-x) + x \to 0$

$(x + y) + z \to x + (y + z)$

(a) 找出该系统的所有临界对。

(b) 对每个临界对$\langle SR\ ,\ [SR'/\ \vec{m}\ ]SL\rangle$，确定是否有 $SR \twoheadrightarrow \circ \twoheadleftarrow\ [SR'/\ \vec{m}\ ]SL$。

2.34 命题逻辑公式的一个重写系统在例 2.18 给出。

(a) 找出该系统的所有临界对。

(b) 对每个临界对$\langle SR\ ,\ [SR'/\ \vec{m}\ ]SL\rangle$，确定是否有 $SR \twoheadrightarrow \circ \twoheadleftarrow\ [SR'/\ \vec{m}\ ]SL$。

2.35 考虑由例 2.26 的规则(H)和形式为$(A)_n$的所有规则构成的无穷重写系统，证明，该无穷系统合流且终止，或者给出一个反例。

2.36 单条规则

$$f(g(fx)) \to g(fx)$$

构成的重写系统是终止的，因为这条规则减少项中的函数符号。执行完备化过程若干步，并对每个临界对，加上减少函数符号个数的重写规则。所加重写规则的一般形式是怎样的？

2.37 习题 2.26 要求用和表 2.5 的表规范同样的风格，写一个带错误值的树的代数规范。完备化这个规范以得到一个合流的重写规则集合，并且使用这些规则来刻画该规范的初始代数的特征。