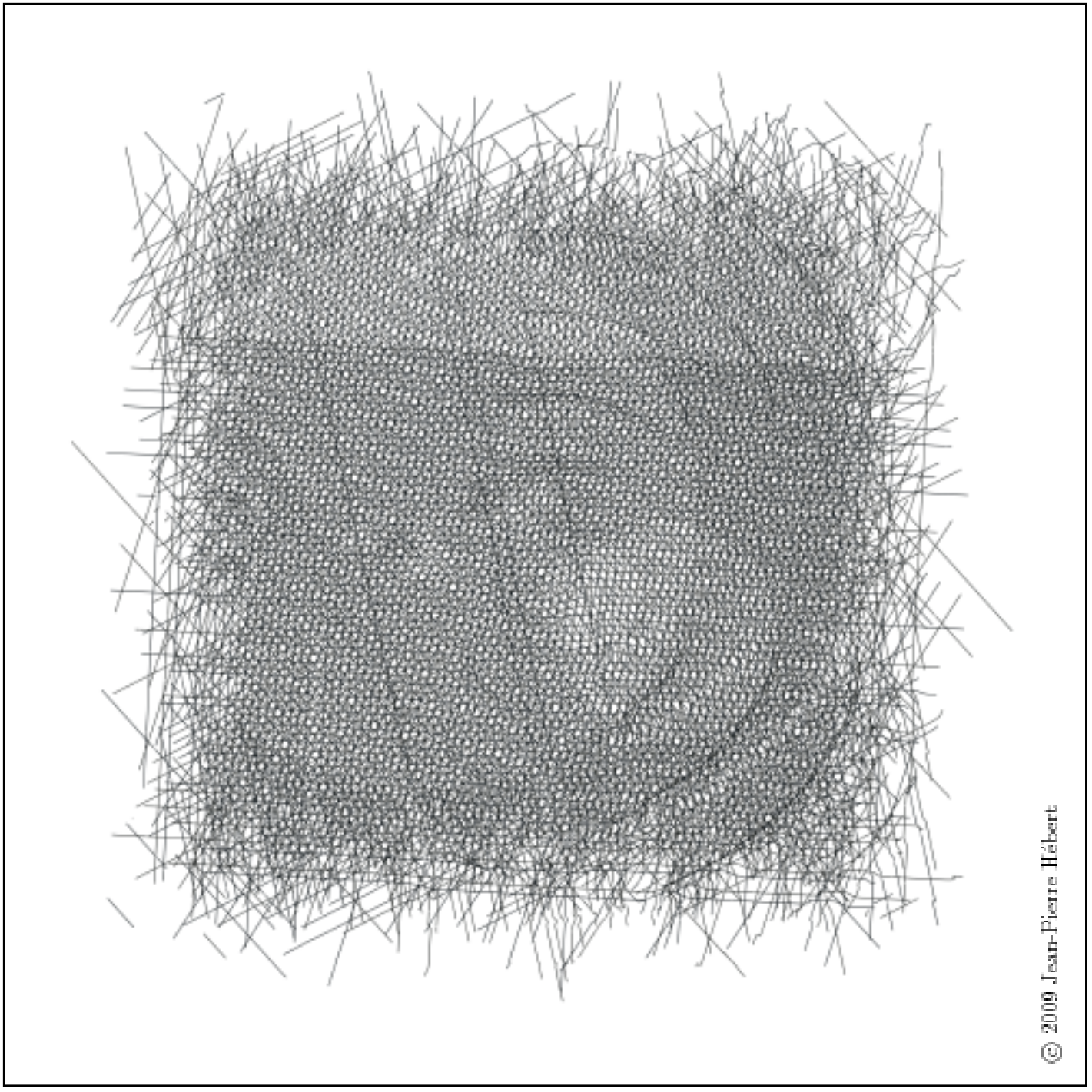

第 10 章。库和顶级程序

库和顶级程序是可移植代码的基本单元，其语言由修订的6计划报告[24]定义。顶级程序可以从一个或多个库导入，库可以从其他库导入。

库使用括号语法命名，该语法包含一系列标识符，后跟一个版本。版本本身是一个括号形式，它包含一系列表示为精确非负整数的颠覆。因此，例如，（a）、（a b）、（a b （））和（a b （1 2 3））都是有效的库名称。实现通常将名称序列视为可以找到库的源代码或目标代码的路径，可能植根于主机文件系统中的某些标准位置集。

标准库机制的实现与语法大小写的可移植实现一起可用，http://www.cs.indiana.edu/syntax-case/。

第 10.1 节.标准库

修订后的[6]报告[24]描述了一个基本库

```
（RNRS 基地 （6））
```

定义了语言最常用的功能。单独的标准库文档 [26] 描述了下面列出的库。

```
（rnrs 算术按位 （6））
（rnrs arithmetic fixnums （6））
（rnrs arithmetic flonums （6））
（rnrs bytevectors （6））
（rnrs conditions （6））
（rnrs control （6））
（rnrs enums （6））
（rnrs eval （6））
（rnrs exceptions （6））
（rnrs files （6））
（rnrs hashtables （6））
（rnrs io ports （6））
（rnrs io simple （6））
（rnrs lists （6））
（rnrs mutable-pairs （6））
```

```
(rnrs 可变字符串 (6))
(rnrs programs (6))
(rnrs r5rs (6))
(rnrs records procedure (6))
(rnrs records syntaxtactic (6))
(rnrs records inspection (6))
(rnrs sorting (6))
(rnrs syntax-case (6))
(rnrs unicode (6))
```

标准库文档中描述了另一个库，即复合库

```
(rnrs (6))
```

导出所有 (rnrs base (6)) 绑定以及上面列出的其他库的绑定，但 (rnrs eval (6)))、(rnrs 可变对 (6))、(rnrs mutable-strings (6)) 和 (rnrs r5rs (6))) 的绑定除外。

尽管这些库中的每一个都有版本(6)，但对它们的引用可以并且在大多数情况下应该省略版本，例如，复合库应该被简单地引用为(rnrs)。

第 10.2 节. 定义新库

新库是使用库窗体定义的，该窗体具有以下语法。

```
(库库名称
(导出导出规范...)
(导入进口规格...)
库体)
```

库名指定名称和可能的版本，通过该名称和版本，库由另一个库或顶级程序的导入形式标识。它还充当实现用于定

位库的路径，通过某些特定于实现的进程，每当需要加载库时。库名称具有以下两种形式之一：

（标识符标识符...）
（标识符标识符 ...版本）

其中版本具有以下形式：

（颠覆...）

每个 Subversion 表示一个精确的非负整数。没有版本的库名称与具有空版本 （） 的库名称相同。例如，（列表工具 setops） 和 （list-tools setops （）） 是等效的，并指定一个没有版本的库名称，而 （list-tools setops （1 2）） 指定一个版本化的库名称，这可以被认为是（列表工具 setops）库的版本 1.2。

导出子窗体命名导出，还可以选择在库外部识别导出的名称。每个导出规范采用以下两种形式之一：

标识符
（重命名（内部名称导出名称）...）

其中每个内部名称和导出名称都是一个标识符。第一个表单命名单个导出标识符，其导出名称与其内部名称相同。第二个名称是一组导出，每个导出名称都显式给出，并且可能与其内部名称不同。

导入子窗体命名新库所依赖的其他库，并可能命名要导入的标识符集以及在新库中应通过的名称来识别这些库。它还可以指定何时应为需要此类信息的实现提供绑定。每个导入规范采用以下两种形式之一：

导入集
（用于导入集导入级别 ...）

其中导入级别是以下之一：

运行
展开
（元级别）

和级别表示一个精确的整数。

for 语法声明导入库何时可以使用导入的绑定，从而声明实现何时必须使绑定可用。run 和 （meta 0） 是等效的，并指定从库导入的绑定可能由导入库的运行时表达式（定义右侧表达式和初始化表达式）引用。expand 和 （meta 1） 是等效的，并指定从库导入的绑定可能由导入库的转换器表达式（定义语法、let 语法或 letrec 语法右侧表达式）引用。（meta 2） 指定从库导入的绑定可能由出现在导入库的转换器表达式中的转换器表达式引用，对于更高的元级别，依此类推。也可以指定负元级别，并且在某些情况下，当转换器扩展到在较低元级别使用的另一个关键字绑定的转换器时，需要负元级别。

库导出可能具有非零导出元级别，在这种情况下，有效导入级别是 for 和 导出级别指定的级别之和。除 （rnrs base） 和 （rnrs） 之外，每个标准库的导出都具有零导出级别。对于 （rnrs base），除语法规则、标识符语法及其辅助关键字 _、...和 set！ 之外，所有导出都具有零级导出。设置！具有导出级别 0 和 1，而其他导出级别为 1。（rnrs） 库的所有导出都有导出级别 0 和 1。

程序员可能很难指定允许库或顶级程序正确编译或运行的导入级别。此外，通常不可能使库的绑定在需要时可

用，而不导致它们在某些情况下在不需要时可用。例如，不能说在展开库 B 时需要库 A 的运行时绑定，而不在展开导入代码 B 时同时提供 A 的运行时绑定。使绑定可用涉及为绑定的右侧执行代码，并且可能还要执行初始化表达式，因此无法精确指定何时需要绑定可能会给程序增加编译和运行时开销。

因此，允许实现忽略导入集上的导出级别和 for 包装器，而是在扩展导入库或顶级程序时，根据对导入库导出的实际显示位置，自动确定何时必须使导入库的绑定可用。当使用这样的实现时，永远不需要使用for包装器，即所有导入规范都可以是导入集。但是，如果代码旨在用于不自动确定何时必须使库的绑定可用，则在导入的库的绑定在适当时间不可用时，必须使用 for。

导入集采用以下形式之一：

```
库规范
（仅限导入集标识符...）
（导入集标识符除外...）
（前缀导入集前缀）
（重命名导入集（导入名称内部名称）...）
```

其中前缀、导入名称和内部名称是标识符。导入集是要从库中导入的标识符的递归规范，也可能是导入库中应通过的名称来识别这些标识符的递归规范。递归结构的底部必须有一个库规范，它标识一个库并从该库导入所有标识符。仅包装器将封闭导入集的导入标识符限制为列出的标识符，except 包装器将封闭导入集的导入标识符限制为未列出的标识符，前缀包装器为封闭导入集的每个导入标识符添加前缀，重命名包装器为封闭的导入集的选定标识符指定内部名称导入集，同时保留其他导入的名称。因此，例如，导入集

（前缀
（仅
（重命名 （列表工具 setops） （差异差异））
合并
差异）
设置：）

仅从（列表工具 setops）库中导入并集和差异，将差异重命名为 diff，同时不保留联合，并将前缀集：添加到两个名称中，以便在导入库中已知两个导入的名称是 set：union 和 set：diff。

库规范采用以下形式之一：

库-参考
（库库-参考）

其中库引用采用以下两种形式之一：

（标识符标识符...）
（标识符标识符 ...版本参考）

当库引用的第一个标识符为 for、library（仅）（除了）、前缀或重命名（前缀或重命名）时，有必要将库引用包含在库包装器中，以将其与由这些关键字之一标识的导入规范或导入集区分开来。

版本引用标识库的特定版本或一组可能的版本。版本引用具有以下形式之一：

（颠覆参考$_1$ ...颠覆参考）
（和版本参考...）
（或版本参考...）
（不是版本参考）

如果每个 subversion 引用都与版本对应的 subversion 匹配，则第一种形式的版本引用与至少具有 n 个元素的版本匹配。如果 和 版本引用窗体的每个版本引用子窗体都与版本匹配，则该窗体与某个版本匹配。如果一个或版本引用表单的任何版本引用子窗体与版本匹配，则该表单与某个版本匹配。如果非版本引用子窗体与版本不匹配，则该窗体与版本匹配。

颠覆引用采用以下形式之一：

```
subversion
（>=subversion）
（<= subversion）
（和 subversion-reference ...）
（或颠覆参考...）
（不是颠覆参考）
```

第一种形式的 Subversion 引用与 Subversion 匹配，如果它与 Subversion 相同。如果版本的 subversion 大于或等于出现在 >= 表单中的 subversion，则 >= subversion 引用与该版本的 subversion 匹配。类似地，如果版本的 subversion 小于或等于出现在 <= 形式中的 subversion，则 <= subversion 引用与该版本的 subversion 匹配该版本的 subversion。如果和 subversion 引用表单的每个 subversion-reference 子形式都与该版本的 subversion 匹配，则该表单与该版本的 subversion 匹配。如果某个版本的 subversion 引用的任何子形式与该版本的 subversion 匹配，则该或 subversion 引用与该版本的 subversion 匹配。如果非颠覆引用子形式与版本的颠覆性与该版本的颠覆不匹配，则该版本与该版本的颠覆相匹配。

例如，如果库的两个版本可用，一个版本为版本 （1 2），另一个版本版本为 （1 3 1）），则版本引用 （）

和 (1) 同时匹配，(1 2) 与第一个版本匹配，但不与第二个版本匹配，(1 (>= 2)) 同时匹配，并且 (和 (1 (>= 3)) (不是 (1 3 1))) 两者都不匹配。

当库引用标识多个可用库时，将以某种依赖于实现的方式选择其中一个可用库。

库和顶级程序不应直接或间接指定导入两个名称相同但版本不同的库。为了避免诸如不兼容类型和复制状态之类的问题，鼓励实现（尽管不是必需的）禁止程序导入同一库的两个版本。

*库体*包含导出标识符的定义、不打算导出的标识符的定义以及初始化表达式。它由一个（可能是空的）定义序列组成，后跟一个（可能是空的）初始化表达式序列。当在第一个表达式之前的库主体中出现开始、let-syntax 或 letrec 语法形式时，它们将被拼接到主体中。 任何体形式都可以通过句法扩展产生，包括定义、刚才提到的拼接形式或初始化表达式。库主体的扩展方式与 lambda 或其他主体相同（第 292 页），并且它扩展为等效的 letrec* 形式，以便从左到右计算主体中的定义和初始化形式。

在库的导出表单中列出的每个*导出*都必须从另一个库导入或在*库体*中定义，如果两者不同，则在任一情况下使用内部名称而不是导出名称。

导入库或在库中定义的每个标识符必须只有一个绑定。如果导入到库中，则不得在库正文中定义它，如果在库正文中定义，则只能定义一次。如果从两个库导入，则在这两种情况下它必须具有相同的绑定，仅当绑定源自两个库中的一个并由另一个库重新导出，或者如果绑定

源自第三个库并由两个库重新导出时，才会发生这种情况。

在库中定义但未由库导出的标识符在库外部显示的代码中不可见。但是，在库中定义的语法扩展可以扩展为对此类标识符的引用，因此扩展的代码确实包含对标识符的引用;这称为间接导出。

库的导出变量在库内部和外部都是不可变的，无论它们是显式导出的还是隐式导出的。如果显式导出的变量出现在导出库内部或外部的 `set!` 表达式的左侧，则存在语法冲突。如果库定义的任何其他变量出现在 `set!` 表达式的左侧并被间接导出，则这也是语法冲突。

加载库，并根据需要“由实现”评估其中包含的代码，由库之间的导入关系确定。库的转换器表达式（库主体的`定义`语法窗体右侧的表达式）可以在与库的主体表达式（主体的`定义`窗体右侧的表达式以及初始化表达式）的不同时间进行计算。至少，当（如果不是之前）在扩展另一个库或顶级程序时（如果不是之前）找到对库的导出关键字之一的引用时，必须计算库的转换器表达式，并且当（如果不是之前）计算对库的导出变量之一的引用时，必须计算正文表达式。 当使用该库的程序运行时，或者当另一个库或顶级程序正在扩展时，如果引用由在扩展过程中调用的转换器计算，则可能会发生这种情况。在扩展其他库的过程中，实现可以根据需要多次评估库的转换器和正文表达式。特别是，如果实际上并不需要表达式，它可以计算零次表达式，正好一次，或者对于扩展的每个元级别计算一次。对于库的转换器或主体表达式的评估，通常涉及外部可见的副作用（例如，弹出一个窗口）是一个坏主意，因为这些副作用发生的

时间或时间是未指定的。通常可以只影响库初始化的本地化效果，例如，创建库使用的表。

示例在第 10.4 节中给出。

第 10.3 节. 顶级计划

顶级程序本身不是语法形式，而是一组通常仅由文件边界限定的形式。顶级程序可以被视为没有库包装器、库名称和导出窗体的库窗体。另一个区别是，定义和表达式可以在顶级程序的主体中混合，但不能在库的主体中混合。因此，顶级程序的语法只是一个导入形式，后跟一系列定义和表达式：

```
（导入进口规格...）
定义或表达式
...
```

在一个或多个定义之前出现在顶级程序主体中的表达式被视为出现在虚拟变量定义的右侧，该变量在程序中的任何位置都不可见。

过程：（命令行）
返回：表示命令行参数
库的字符串列表：（rnrs 程序）、（rnrs）

可以在顶级程序中使用此过程来获取传递给该程序的命令行参数的列表。

过程：（退出）
过程：（退出 obj）
返回：不返回
库：（rnrs 程序），（rnrs）

此过程可用于从顶级程序退出到操作系统。如果未给出 obj，则返回给操作系统的退出值应指示正常退出。如果 obj 为 false，则返回到操作系统的退出值应指示异常退出。否则，obj 将转换为适合操作系统的退出值。

第 10.4 节. 例子

下面的示例演示了库语法的几个功能。它定义了（列表工具 setops）库的“版本 1”，该库导出两个关键字和几个变量。该库导入（rnrs base）库，该库提供除成员过程之外所需的所有内容，成员过程是从（rnrs列表）导入的。库导出的大多数变量都绑定到过程，这是典型的。

语法扩展集扩展为对变量 list->set 和 member? 的引用，同样也扩展为对变量 member-help? 的引用。虽然显式导出了 list->set，但成员帮助？这使得成员帮助？成为一种间接出口。过程 u-d-help 未显式导出，并且由于导出的语法扩展都不会扩展为对 u-d-help 的引用，因此也不会间接导出。这意味着可以分配它，但在此示例中未分配它。

```
（库 （列表工具 setops （1））
（导出集空集空集？ 列表>集集>列表
联合交集差异成员？）
（import （rnrs base） （only （rnrs lists） member））

（define-syntax set
（syntax-rules ()
[（_ x ...）
（list->set （list x ...））））

（定义空集 ’（））

（定义空集？ 空？）
```

```
(define list->set
(lambda (ls)
(cond
[(null? ls) '()]
[(member (car ls) (cdr ls)) (list->set (cdr ls))]
[else (cons (car ls) (list->set (cdr ls)))])))

(define set->list (lambda (set) set))

(define u-d-help
(lambda (s1 s2 ans)
(let f ([s1 s1])
(cond
[(null? s1) ans]
[(成员? (car s1) s2) (f (cdr s1))]
[else (cons (car s1) (f (cdr s1)))])))))

(define union
(lambda (s1 s2)
(u-d-help s1 s2 s2))))

(define intersection
(lambda (s1 s2)
(cond
[(null? s1) '()]
[(member? (car s1) s2)
(cons (car s1) (intersection (cdr s1) s2))]
[else (intersection (cdr s1) s2)]))))

(define difference
(lambda (s1 s2)
(u-d-help s1 s2 '()))))

(define member-help?
(lambda (x s)
(和 (成员 x s) #t))))
```

```
(定义语法成员?
(syntax-rules ()
[(_ elt-expr set-expr)
(let ([x elt-expr] [s set-expr])
(and (not (null? s)) (member-help? x s)))))
```

下一个库(more-setops)根据(列表工具 setops)操作定义了一些额外的集合操作。库引用中没有包含任何版本(列表工具 setops);(列表工具 setops);(列表工具设置);这等效于与任何版本匹配的空版本引用。引用集关键字很有趣，因为它的转换器在扩展时引用 list->set from (list-tools setops)。因此，如果从(more-setops)导入的另一个库或顶级程序引用引用集，则在扩展其他库或顶级程序时，必须评估(列表工具 setops)库的运行时表达式。另一方面，当(更多 setops)库本身展开时，不需要计算(列表工具 setops)库的运行时表达式。

```
(library (more-setops)
(export quoted-set-cons set-remove)
(import (list-tools setops) (rnrs base) (rnrs
syntax-case))

(define-syntax quoted-set
(lambda (x)
(syntax-case x ()
[(k elt ...)
#'(引用
#, (datum->syntax #'k
(list->set
(syntax->datum #'(elt ...)))))))

(define set-cons
(lambda (opt optset)
(union (set opt) optset)))

(define set-remove
```

```
(lambda (opt optset)
(difference optset (set opt)))))
```

如果实现没有自动推断何时需要使绑定可用，则必须修改(more-setops)库中的导入表单，以指定通过 for import-spec 语法在哪些元级别使用它导入的绑定，如下所示：

```
(导入
(对于(列表工具 setops) 展开运行)
(对于 (rnrs base) 展开运行)
(对于 (rnrs 语法大小写) 展开))
```

为了完成该示例，下面的简短顶级程序练习了几个(列表工具 setops)和(更多 setops)导出。

```
(import (list-tools setops) (more-setops)
(rnrs))
(define-syntax pr
(syntax-rules ()
[(_ obj)
(begin
(write 'obj)
(display " => ")
(write obj)
(newline))]))
(define get-set1
(lambda ()
(quoted-set a b c d))))
(define set1 (get-set1))
(define set2 (quoted-set a c e))

(pr (list set1 set2))
(pr (eq? (获取设置1) (获取设置1)))
(pr (eq? (获取设置1) (设置'a 'b 'c 'd)))
(pr (union set1 set2))
(pr (intersection set1 set2))
```

```
(pr (difference set1 set2))
(pr (set-cons 'a set2))
(pr (set-cons 'b set2))
(pr (set-remove 'a set2))
```

运行此程序应打印的内容留给读者作为练习。

第 12 章给出了其他库和顶级程序示例。

ISBN 978-0-262-51298-5 / LOC QA76.73.S34D93
订购本书 / 关于这本书

http://www.scheme.com