arXiv:2303.17457v1 [cs.FL] 30 Mar 2023

Isabelle/HOL 中的 VDM递归函数

Leo Freitas[1] 和 Peter Gorm Larsen[2]

[1] 计算机学院，纽卡斯尔大学，
leo.freitas@newcastle.ac.uk[2] DIGIT，奥胡斯大学，
工程系，pgl@ece.au.dk

摘要。对于递归函数，需要应用归纳的一般原理。我们建议将其翻译到 Isabelle/HOL，而不是直接使用维也纳开发方法规范语言（VDM-SL）对其进行验证。本文介绍了将递归函数进行此类翻译所面临的挑战。这项工作扩展了现有的翻译和 Isabelle/HOL 中的 VDM 数学工具箱，使其能够支持递归函数。

关键词：VDM、Isabelle/HOL、翻译、递归、VSCode

1 引言

本文介绍了一种将多种 VDM 递归定义翻译为 Isabelle/HOL 的翻译策略。该策略考虑了两种形式化体系对终止性和良基性的不同表示方式。

除了克服我们讨论的技术和实践难题外，一个重要目标是创建翻译策略模板。这些模板必须涵盖多种 VDM 递归定义，并且其证明义务应高度自动化。最终成果是扩展 VDM 到 Isabelle/HOL 的翻译策略，并将其实现为 VDMJ 插件 [1] 以及 VDM-VSCode 的扩展 [8]。

Isabelle 使用文学化编程，形式化规范、证明和文档都在同一个环境中。以下省略证明脚本；完整的 VDM 和 Isabelle 源代码及证明可在 VDM 工具包代码库的 ./plugi 找到

ns/vdm2isa/src/main

下一节将介绍 VDM 和 Isabelle 中的递归与度量关系的背景。第 3 节将简要讨论 VDM 基本类型的翻译及其对递归的影响。接下来，第 4 节将说明 VDM 和 Isabelle 中的递归定义如何运作，以及二者有何不同。随后，我们将在第 5节介绍针对基本类型、集合、映射和复杂递归模式的翻译策略。最后，我们将在第 6节作出总结。

[1] 位于https://github.com/leouk/VDM_工具包

2 背景

我们的 VDM 到 Isabelle/HOL 翻译器支持范围广泛的 VDM-SL 抽象语法树。它能够处理各种表达式、多种模式、几乎所有类型、导入和导出、函数和规约、轨迹，以及部分状态和操作 [4,8]。尽管并非所有 VDM 模式都被允许使用，但翻译器能够处理其中大多数；对于无法处理的模式，也都有相应的等价形式。

我们希望扩展翻译的一个特定领域是递归定义函数。VDM 要求用户定义一个度量函数，以证明递归会终止。随后，它会生成证明义务，以确保全定义性和终止性。

最后，我们的翻译策略遵循 [7,3] 中描述的大小变化终止性（SCT）证明策略。具体而言，该策略采用 SCT 中递归定义的 SCP（多项式）和 SCNP（非多项式）子类，从而能够高效地检查终止性。实际上，如果每个无限计算都会产生一个无限递减的值序列（依据大小变化原理），那么就不可能存在无限计算。这一类中的终止性问题具有特定形式的全局秩函数，可以通过 SAT 求解找到该函数，从而提高自动化程度。

3 Isabelle 中的 VDM 基本类型

Isabelle 将自然数（Ž）表示为一种（数据）类型，包含两个构造子（0 和 Suc n）；所有数字都是这类构造的投影（例如 $2 = \text{Suc (Suc 0)}$）。Isabelle 整数（š）定义为涉及两个自然数的商类型。Isabelle 商类型是到一种构造性定义类型中的单射。与整数一样，其他 Isabelle 数值类型（例如，有理数‘、实数’等）也都基于某种复杂的自然数构造来定义。随后定义类型转换（或强制类型转换），以允许用户在不同类型空间之间转换。尽管如此，Isabelle 对 Ž 没有隐式类型扩宽规则；相反，它采用诸如 $(0::Ž) - (x::Ž) = (0::Ž)$ 这样的约定。对于涉及 š 和 Ž 类型项混合使用的表达式，可能需要显式的用户定义类型强制转换（例如，$\text{int}\ (2::Ž) - (3::š) = -\ (1::š)$）。

使用基本类型（nat、int）变量的 VDM 表达式具有特定的类型扩宽规则。例如，即使两个变量都是 nat，结果也可能是 int。（例如，在 VDM 中，0 - x:nat = - x:int）。因此，我们的翻译策略将 VDM nat 视为 Isabelle 类型 VDMNat，而 VDMNat 只是 š 的类型同义词。这样可以简化向 Isabelle 的翻译过程，无需使用类型强制转换即可编码所有 VDM 类型扩宽规则。另一方面，鉴于 VDM 的 nat 表示为 Isabelle 的 š，这一设计决策使得对 nat 上的递归函数进行编码比预期更加复杂。

尽管对基本类型采取了这一设计决策并由此产生了相应影响，仍然需要对 VDM 整数、集合或映射进行递归。这是因为在 Isabelle 中，这些类型并非以构造性方式定义。

4 VDM 和 Isabelle 中的递归

每个递归定义的一个重要方面是，要有一个参数来证明其具有终止性。否则，递归可能会陷入无限循环。在 VDM 中，这通过递归度量来定义：它具有与递归定义相同的输入类型签名，并返回一个自然数；该自然数必须在每次递归调用时单调递减，最终达到零。这就是在 VDM 中证明递归定义具有终止性的方法。一个简单的 VDM 递归定义示例是计算给定自然数的阶乘。

```
fact: 自然数-> 自然数fact(n) == 如果 n = 0 那么 1 否则 n * fact(n - 1) 度量n;
```

fact 递归度量将输入 n 本身用作其结果。这是可行的，因为唯一的递归调用都使用递减后的 n 值，直到到达基例 0 并终止。VDMJ 为上述定义生成三个证明义务。在 Isabelle 中，由于展开后的度量定义只是 $\forall n。\ n \neq 0 \longrightarrow 0 \leq n - 1$ 和 $\forall n。\ n \neq 0 \longrightarrow n - 1 < n$，因此这些证明义务很容易完成。

```
证明义务 1: （未证明）fact: 全函数义务（对所有n:自然数 & is_(度量_fact(n), 自然数）

证明义务 2: （未证明）fact: 子类型义务（对所有 n:自然数 & （非 （n = 0）
 => （n - 1） >= 0））

证明义务 3: （未证明）fact: 递归函数义务（对于 n:自然数 & (非 (n=0) => 度量_fact(n) > 度量_
fact((n-1)))
```

此外，尽管在 VDM 中无法定义递归类型结构上的度量，但在 Isabelle 中却很容易描述。例如，在 VDM 中无法对定义链表的递归记录编写度量，例如 `R :: v:nat n: R`。在 Isabelle 中，我们将 VDM 记录表示为数据类型时，会自动生成此度量。其他复杂的递归模式在 VDM 中很难或无法表示（参见第 5.4节）。

在 Isabelle 中，递归定义可以通过对以构造方式定义的输入进行原始递归来提供，也可以通过更通用的函数定义来提供，后者会产生证明义务。前者要求为每个类型构造子都给出定义，并且只提供化简规则；而后者允许使用更复杂的输入模式，并提供化简、消去和归纳规则，同时还考虑了部分函数。本文仅讨论函数定义。读者可在 [5] 中了解有关这些差异的更多信息。

Isabelle 的递归函数需要满足一项证明义务：其参数表示构造性且兼容的模式，并且递归调用会终止。构造性模式涵盖递归输入中所用数据类型的所有构造子（即，对 Ž 的每个构造子各有一个方程，因此一个涉及 0，另一个涉及 Suc n）。兼容模式涵盖构造模式的多种方式，最终会归结为模式完备性的各种情况（例如，$n + 2$ 仅仅是

对已定义构造子进行多次调用，例如 Suc (Suc n)）。这对于确保递归结构良好至关重要（即，递归调用不会因某些构造子不可用而陷入停滞）。例如，如果遗漏了 0 的情况，Suc n 的情况最终会到达零并失败，因为没有针对零的模式。关于终止性的证明义务用于确立递归是良基的。每当需要将所定义函数的性质证明为全函数时，都必须证明这一点。

Isabelle 的函数定义可以使用 fun 或 function 语法。前者会尝试自动证明模式构造性和兼容性证明，并为终止性证明义务寻找度量。后者要求用户通过提供度量关系手动完成这些证明。它更适用于 fun 声明无法处理的情况，这些情况通常涉及复杂或定义不明确的递归。

终止关系必须是良基的，这意味着在一个偏序关系上具有良序归纳原理，该关系定义为[2]

$$wf\ (r::('a \times 'a)\ set) =$$
$$(\forall P::'a \Rightarrow \mathbb{B}.\ (\forall x::'a.\ (\forall y::'a.\ (y, x) \in r \longrightarrow P\ y) \longrightarrow P\ x) \longrightarrow (\forall x::'a.\ P\ x))$$

Isabelle 能够发现全部三个证明的定义如下：

fun fact′:: ‹Ž ⇒Ž› where ‹fact′n =(若 n = 0 则 1 否则 n ∗(fact′(n − 1)))›

这个定义与 VDM 中的定义相当相似。不过，VDM 基本类型的扩宽规则要求我们将它们转换为 VDMNat。为 š 定义的同一版本 fact 会报错：“Could not find lexicographic termination order”（找不到字典序终止顺序）。也就是说，Isabelle 能够完成模式证明，却无法完成终止性证明。这是因为用户必须提供一个投影关系，将 š 商类型映射回构造类型 Ž。
即使我们能够避免这一翻译技术问题，对非构造性类型（如集合或映射）进行递归时，也会遇到同样的问题。它们需要对有限集合进行递归，而有限集合是通过归纳定义的。唯一容易处理的递归翻译涉及列表，因为 Isabelle 中的列表是以构造方式定义的，而且 VDM 序列可以直接映射。

因此，对非构造性类型定义递归函数需要证明相容性和完备性。此外，由于 Isabelle 无法判断终止性是否一目了然，这也会导致部分函数定义。然而，在 VDM 中，对集合（以及映射域）进行递归很常见，因此有必要扩展我们的翻译策略。

5 VDM 递归翻译策略

我们希望找到一种翻译策略，既能处理上述问题，也能适用于基本类型、集合、序列、映射以及相互递归。
VDM-SL 抽象语法树会标记所有递归函数，包括没有显式度量的函数。所有这类函数都会使用 Isabelle 的 fun 语法进行翻译，该语法会尝试

[2] 关于良序归纳的详细说明，参见 Isabelle 的 Wellfounded.thy 理论。

寻找相容性和终止性的证明。在我们的 fact 示例设置中，Isabelle 会找到终止性证明。

如果失败了，用户可以为 VDM `@IsaMeasure`注解作出定义。VDM 注解是注释，会根据特定实现进行处理[2,8]。如果用户未提供 `@IsaMeasure` 注解，而 *fun* 失败了，那么用户就需要自行确定所需的证明设置。

@IsaMeasure 注解定义了一种良基测度关系，该关系将参与 Isabelle 终止性证明的设置。例如，对于 fact 示例，用户需要在 VDM 测度前写一个注解。

```
--@IsaMeasure( { (n -1, n) | n : 自然数 & n <> 0 } )
```

此测度关系对应于递归调用（`fact(n-1)`）与其定义方程（`fact(n)`）之间的关系，其中筛选条件决定了该关系适用于 `n` 的哪些取值。第 5.4 节中定义了更有意思的测度关系。

在翻译过程中，vdm2isa 插件将对 @IsaMeasure 注释进行类型检查（即，它是在函数签名上的类型正确关系）。接下来，它将把该注释及一些自动化引理翻译为一系列 Isabelle 定义，以便在证明已翻译 VDM递归函数的终止性时使用。若未提供注释，则该插件将遵循与 Isabelle 类似的原则，尝试根据递归函数定义的结构自动推断度量关系应为何种关系。若此操作失败，将通知用户。即使测度关系综合成功，用户仍必须在 Isabelle 的终止性证明中恰当地使用它。

下文将详细说明每种相关 VDM 类型的翻译策略。有关整体翻译策略的详细信息，请参见发行版中的示例[3] 和 [4]。该翻译策略将各种隐式 VDM 检查作为显式谓词施加。例如，VDM 集合始终是有限的，并且集合、序列和映射元素上的类型不变量必须对每个元素都成立。

5.1 对 VDM 基本类型（自然数、整数）进行递归

按照通用翻译策略 [4], ，我们首先对隐式前置条件进行编码。该条件要求给定参数 n 为 VDMNat，并附带一组有助于证明策略合成的定义常量。

定义 pre-fact :: ‹VDMNat ⇒ ›其中 ‹pre-fact n ≡ VDM 自然数不变式 n› 引理 pre-fact-defs = pre-fact-def

接下来，我们递归地定义阶乘。当前置条件不成立时，我们返回未定义；在 Isabelle 中，未定义项无法用于推理。domintros 标签会告诉 Isabelle 生成定义域谓词，以防此函数不是全函数。定义域谓词是

[3] https://github.com/leouk/VDM_工具包

我们策略中的重要部分，因为每个 VDM 函数在其前置条件之外被调用时都会未定义。它还会生成依赖定义域谓词的证明规则，如下所列。

function(domintros) fact :: ‹VDMNat ⇒ VDMNat› where ‹fact n = (若 pre-fact n，则（若 n = *0* 则为 1，否则为 n * (fact (n − *1*))），否则未定义)›

模式兼容性和完备性的证明义务，使用 Isabelle 针对简单递归模式的常规证明策略，并通过 *pat-completeness* 方法得以解决。在第 5.4 节讨论的一般情况下，用户可能需要证明一些目标。Isabelle 会证明有关分类讨论以及用于消去、归纳和化简的（部分）规则的各种定理。

部分规则要求将定义域谓词 fact-dom 作为假设。它表示一种确保终止性的良基关系。也就是说，如果用户不想（或不知道如何）证明终止性，那么每次应用定义时都会带有这样的定义域谓词，从而记录仍缺少这种良基关系这一要求。

如果终止性证明得以完成，这些 p-规则可以化简为不依赖定义域谓词的全规则，前提是已经提供了良基关系。定义域谓词会使用户的证明复杂化，也会使证明策略的自动合成更加困难。

终止性证明通过建立与该 function 的递归调用相对于其声明相关联的良基关系来完成。@IsaMeasure 注释被翻译为 Isabelle 缩写。用于筛选函数前置条件成立的条件：这对于确保终止性证明永远不会到达未定义情况非常重要。另一个筛选条件来自 if 语句中被否定的测试。更复杂的定义将包含更复杂的筛选条件。我们使用缩写而不是定义，以避免在证明中需要展开已定义的项。

缩写 fact-wf :: ‹(VDMNat × VDMNat) set› where ‹ fact-wf ≡ { (n − *1*, n) | n 。 pre-fact n ∧ n ≠ *0* }›

由于 fact 定义是简单的（非相互递归、单一调用点且易于选择度量关系）递归，因此其设置很容易建立良基性。对于此类递归，我们可以借助 Isabelle 机制，通过使用 gen-VDMNat-term 和 int-ge-less-than 这些项来证明良基性。

第一个项是根据第二个项定义的，而第二个项是我们的良基关系 fact-wf 的子集。Isabelle 已经有关于 int-ge-less-than 良基性的证明。因此，对于 sledgehammer 来说，证明我们的项是良基的是很简单的。作为翻译策略的一部分，我们定义（并尝试发现其证明）以下引理。这遵循 [6] 中描述的策略。

引理 l-fact-term-wf: ‹wf (gen-VDMNat-term fact-wf)›

最后，我们使用基于该关系的先前已证明引理来证明终止性。这会将目标化简为证明终止关系是良基的，以及前置条件蕴含该关系；在本例中，这两点都可以通过化简轻松证明。

在这个例子中，子目标由 sledgehammer 证明。一般来说，用户必须自行找到证明，或者处理涉及递归调用的定义域谓词。完成终止性证明后，Isabelle 会提供用于消去、归纳和化简的规则版本；这些规则是全函数的，且不依赖定义域谓词。

为了确保我们的选择不会导致空关系，我们通过证明下一个目标来确保终止关系实际上与良基谓词相同。用户可能会想要这样做，但这并非翻译策略的一部分。如果度量关系为空，那么递归调用简化规则无论如何都派不上用场。

lemma l-fact-term-valid: ‹(gen-VDMNat-term fact-wf) = fact-wf ›

5.2 对 VDM 集合进行递归

接下来，我们将翻译策略扩展到 VDM 集合。为此，我们使用一个函数来计算集合中各元素的和。

```
sumset: 自然数集合 -> 自然数sumset(s) == 如果 s = {} 则 0 否则令 e 属于集合 s 中
sumset(s - {e}) + e --@IsaMeasure({(x - { let e in set x in e }, x) | x : set of nat& x <> {}}) --
@Witness(sumset({ 1 }))度量 card s;
```

最常见的 VDM 集合递归会通过任意选取一个集合元素来消耗集合，然后在不包含所选元素的集合上继续递归，直到集合为空。VDM 测度说明递归基于输入参数的基数。由于 Isabelle 要求使用关系，VDM 测度不适用于 Isabelle 证明；因此，VDM 测度基本上会被忽略。不过，在证明过程中，它们仍可能作为存在量化目标的潜在 Witness。

隐式的 VDM 检查被定义为前置条件，以确保给定集合只包含自然数且为有限集合，具体定义见 inv-VDMSet′。

定义 pre-sumset :: ‹VDMNat VDMSet ⇒ ›，其中 ‹
pre-sumset s ≡ inv-VDMSet′ inv-VDMNat s› 引理
pre-sumset-defs = pre-sumset-def inv-VDMSet′-defs

接下来，我们在 Isabelle 中定义 VDM 递归函数。它会检查给定集合是否满足函数前置条件；如果不满足，则返回未定义。之后，每种情况基本上都按照 VDM 逐一编码。针对 VDM 集合中的 let-in 模式，翻译策略使用 Isabelle 的希尔伯特选择算子（SOME x. x ∈ s）。需要注意的是，这种方式自然也适用于 VDM 的 let-be-st 模式。

function (domintros) sumset :: ‹VDMNat VDMSet ⇒ VDMNat›
where ‹sumset s = (if pre-sumset s then (if s = {} then 0 else let e =
(ϵ x . x ∈ s) in sumset (s − {e}) + e) else 未定义)›

模式完备性和兼容性目标都显而易见。接下来定义由 @IsaMeasure 给出的度量关系。它定义为选取 e 后得到的较小集合，以及入口调用中使用的集合，因此产生这些序对 (s − {SOME e. e ∈ s}, s)。最后，我们确保所有关系元素都满足函数前置条件，并且 if 测试为否定。

缩写 sumset-wf-rel :: ‹(VDMNat VDMSet × VDMNat VDMSet) set› 定义为 ‹sumset-wf-rel ≡ {(s − {(ϵ e . e ∈ s)}, s)| s . pre-sumset s ∧s ≠ {}}›

鉴于这是一个简单的递归，我们再次可以借助 Isabelle 机制，使用 gen-set-term 和 finite-psubset。这些内容确立了这样一种关系：在关系对中，第一个元素严格小于第二个元素的集合关系是良基的。这使得 sledgehammer 很容易完成良基性证明。

引理 l-sumset-rel-wf: ‹wf (gen-set-term sumset-wf-rel)›

接下来，我们处理终止性证明，仍然采用相同的关系设置。

幸运的是，对于大多数简单情况，这通常很容易分解。翻译策略会获取 @IsaMeasure 表达式并分解其各个部分，使筛选谓词成为假设，并使关系中的元素属于所选的良基度量。下一条引理对此进行了定义；在 Isabelle 的 sledgehammer 完成证明之前，需要进行一些手动干预。

引理 l-pre-sumset-sumset-wf-rel:
‹pre-sumset s ⟹ s ≠ {} ⟹ (s − {(ϵ x. x ∈ s)}, s) ∈ (gen-set-term sumset-wf-rel)›

这个引理的直观含义是，在函数前置条件以及进行递归调用的筛选情形（s ≠ ∅）下，度量关系中的元素满足良基性。也就是说，前置条件和筛选条件有助于建立终止关系。对于这个特定证明，前置条件中唯一需要的性质是集合有限。基于此，我们尝试进行终止性证明，sledgehammer 找到了所有子目标的证明。

注意，我们省略了 *VDMNat* 情形下关于终止性和前置条件的此类引理，详见第 5.1 节。翻译策略确实按照相同的步骤定义了该引理，随后**sledgehammer** 再次找到了证明。

引理 l-pre-fact-wf-rel: ‹⟦pre-fact n; n ≠ 0⟧ ⟹ (n − 1, n) ∈ gen-VDMNat-term fact-wf ›

最后，尽管对于此证明而言这并非必要，我们仍建议用户始终为顶层递归调用提供一个 Witness。这可以通过使用 `@Witness`注解 [8]来完成：它为函数输入参数提供了一个具体示例。对于所涉及的终止性证明中的存在量词谓词而言，这个 Witness 很有用（见第 5.4节）。

5.3 对 VDM 映射进行递归

映射上的递归函数是集合的一个特例，因为映射递归通常会遍历映射的定义域。例如，对映射值域中的元素求和的函数定义为

sum_elems: map 自然数到自然数 -> 自然数 sum_elems(m) == 若 m = {|->} 则为 0，否则令 d 属于 domm 中，使 m(d) + sum_elems({d}<-:m) --@IsaMeasure({{{d} <-: m, m) | m : map 自然数到自然数, d: 自然数 & m <> {} 且 d 属于 dom m}) --@Witness(sum_elems({ 1 |-> 1 })) 度量 card dom m;

与集合的情况一样，它通过选取一个定义域元素来遍历映射，执行必要的计算，然后对移除了所选元素的过滤后映射进行递归，直到映射为空，函数终止。度量关系遵循相同的模式：递归调用点与定义点相关联，其中 if 测试和 let-in-setchoice 都是过滤谓词的一部分。

遵循映射的一般翻译策略 [4], ，我们使用 *inv-Map*定义函数的前置条件。它要求映射的定义域和值域都是有限的，并且定义域和值域中的所有元素都满足相应的类型不变式。请注意，如果递归是在定义域和值域以外的集合上定义的，那么 Isabelle 要求证明该集合是有限的。由于定义域和值域集合都是有限的，因此如有需要，这应该很容易证明。

定义pre-sum-elems :: ‹(VDMNat ⇀ VDMNat) ⇒ ›，其中 ‹
pre-sum-elems m ≡ inv-VDMMap inv-VDMNat inv-VDMNat m› 引理
pre-sum-elems-defs =pre-sum-elems-def inv-VDMMap-defs

Isabelle 中的 VDM 映射（VDMNat ⇀ VDMNat）被定义为一种映射到可选结果的 HOL 函数。也就是说，如果元素位于定义域中，则映射会得到一个非**nil** 值；而如果元素不属于定义域，则映射会得到一个**nil** 值。这样，所有映射都是全函数，定义域之外的值会映射到**nil**。Isabelle 的翻译和兼容性证明遵循之前使用的模式。

function(domintros) sum-elems :: ‹(VDMNat ⇀ VDMNat) ⇒
VDMNat› where ‹sum-elems m =(若前置条件 pre-sum-elems m 成立，
则如果 m = Map.empty，则为 0；否则令 d = (ϵ e . e ∈ 定义域 m)，则
为(the(m d) + (sum-elems ({d} −◁ m)))；否则为未定义)›

同样，接下来会翻译良基关系，其中前置条件作为关系过滤器的一部分包含在内。{d} −◁ m 对应于 VDM 定义域反过滤算子{d}<-:m。

缩写
sum-elems-wf :: ‹((VDMNat ⇀ VDMNat) × (VDMNat ⇀VDMNat))
VDMSet› 其中 ‹sum-elems-wf ≡

{ (({d} −◁ m), m) | m d 。pre-sum-elems m ∧ m ≠ Map。empty ∧d ∈dom m }›

对于递归度量关系上的良基引理，Isabelle 没有提供可用的帮助，而且投影关系中映射的域元素十分棘手。因此，我们必须证明该良基引理，而这通常不会自动完成。这是关于集合和映射上的 VDM 递归函数翻译的一个区别。

幸运的是，这类情况的证明策略已有一定了解：它与 finite-psubset 良基性的证明策略类似。该策略使用给定的 VDM度量表达式，提取所关注的正确投影，随后沿用 finite-psubset 的证明，而 sledgehammer 可以找到最后几步。

前置条件子目标和终止性证明沿用与之前相同的模式。它们的证明是通过 sledgehammer 找到的，但一般情况下并非如此。

引理 l-sum-elems-wf : ‹wf sum-elems-wf ›

引理 l-pre-sum-elems-sum-elems-wf : ‹⟦
pre-sum-elems m; m ≠ Map。empty⟧ ⟹ ({(ϵ e。
e ∈dom m)} −◁ m, m) ∈ sum-elems-wf ›

引理 l-sum-elems-wf-valid: ‹sum-elems-wf ≠{}›

最后，我们还证明了这个良基终止关系并非空集，就像我们对集合和自然数递归所做的那样。请注意，在这里，@Witness注解有助于确定要使用的实际值，以此作为证明该关系非空的见证。

5.4 涉及复杂度量的 VDM 递归

目前展示的递归示例涵盖了广泛的情况，并且具有较高程度的自动化。不过，同样的策略也可以应用于更复杂的递归定义。对于 VDM用户而言，代价是需要编写复杂的 `@IsaMeasure` 定义，并且很可能还需要额外的用户定义的引理。这些引理可以在 VDM 中使用 `@Lemma` 注解来定义。为说明这一点，我们在 VDM 中定义著名的阿克曼函数[4], ，这是复杂递归的典型示例。

```
ack: 自然数* 自然数 -> 自然数ack(m,n) == 若 m = 0 则 n+1
否则若 n = 0 则 ack(m-1, 1) 否则 ack(m-1, ack(m, (n-1))) --
@IsaMeasure( pair_less_VDMNat) --@Witness(
ack(2, 1) )度量尚未指定;
```

[4] https://en.wikipedia.org/wiki/Ackermann_函数

请注意，VDM 测度尚未定义，而 `@IsaMeasure` 使用了一种名为 *pair-less* 的 Isabelle 构造。它是 Isabelle 针对 SCNP 问题的具体序工具体系的一部分 [6]。它考虑涉及多个参数的递归，其中某些参数可能会增加调用次数（例如大小变化）。我们尚不了解在 VDM 中定义此类度量的方法。

我们将 pair-less 实例化为 VDMNat，作为输入之间全序关系的传递闭包上的字典序积[5]。如果 VDM 测度定义在关系之上，那么在假设传递闭包采用标准定义的情况下，就可以在 VDM 中定义阿克曼度量[6]。

```
pair_less_VDMNat: () -> 集合 ((自然数*自然数) *(自然数*自然数)) pair_less_VDMNat() == 字典序_积[
自然数, 自然数](less_than_VDMNat(), less_than_VDMNat()); less_than_VDMNat: () -> 集合 (自然数*自
然数) less_than_VDMNat() == 传递_闭包[自然数]({ mk_(z', z) | z', z : 自然数 & z' <z }); 字典序_积[
@A,@B]: 集合 (@A*@A) * 集合 (@B*@B) ->集合 ((@A*@B) * (@A*@B)) 字典序_积(ra,rb) == { mk_(
mk_(a, b), mk_(a', b')) |a, a': @A, b, b': @B & mk_(a,a') 属于集合 ra 或 a = a' 且 mk_(b, b') 属于集合
rb };
```

这表示按参数排序的可能性的字典序积。接下来，翻译结果为：

```
定义 pre-ack :: ‹VDMNat ⇒ VDMNat ⇒ ,›，其中 ‹
pre-ack m n ≡ inv-VDMNat m ∧ inv-VDMNat n› 引理
pre-ack-defs = pre-ack-def

函数(domintros) ack :: ‹VDMNat ⇒ VDMNat ⇒ VDMNat›,
其中 ‹ack m n = (if pre-ack m n then if m = 0 then n+1else
if n = 0 then ack (m−1) 1 else ack (m−1) (ack m (n−1)) else
undefined)›

缩写 ack-wf :: ‹((VDMNat × VDMNat) ×(VDMNat ×VDMNat)) VDMSet›，其中
‹ack-wf ≡pair-less-VDMNat›
```

尽管度量更复杂，但由于 Isabelle 自动化的支持，相容性和终止性证明仍然相似。我们还展示了这个使用 VDMNat 的阿克曼版本与 Isabelle 中使用 Ž 的常规定义等价。此处省略细节，但我们已通过归纳证明二者等价。

```
定理 ack-correct: ‹ack′ m n = ack m n› 应用 (归纳 ‹m› ‹n› 规则:
ack′.induct) 由 (simp add: pre-ack-defs)+
```

[5] 有关此定义的详细信息，请参阅该发行版中的 VDMToolkit.thy。[6] Relations.vdmsl 在 VDM 工具包发行版中提供了此类定义。

一般来说，每个复杂递归函数都需要这样的设置。幸运的是，Isabelle 提供了多种可用选项。不过，通常递归越复杂，用户就越需要提供额外的自动化。

5.5 更复杂的示例

我们可以处理 [7]中的所有示例。这里展示一些需要复杂设置的示例。例如，排列函数展示了如何通过复杂的度量来对递减参数进行置换。要完成终止性证明，必须提供前置条件，这也说明了为什么针对 ℤ 的证明可能更困难。这些示例表明，Isabelle 中失败的证明如何帮助改进 VDM 规约。

```
perm: 整数 * 整数 * 整数 -> 整数 perm(m,n,r) == 若 0 < r 则 perm(m, r-1, n), 否则若 0 <
n 则 perm(r, n-1, m), 否则返回 m --@IsaMeasure({mk_(mk_(m, r-1, n), mk_(m,n,r)) | ... &
  0 < r} 并集 -- {mk_(mk_(r, n-1, m), mk_(m,n,r)) | ... & not 0 < r and 0 < n}) 前置条件 ((
0 < r or 0 < n) => m+n+r > 0) 度量 maxs({m+n+r, 0}); tak: 整数 * 整数 * 整数 -> 整数
tak(x,y,z) == 若 x <= y 则返回 y, 否则 tak(tak(x-1,y,z), tak(y-1,z,x), tak(z-1,x,y)) 度量尚未指定;
```

```
定义 pre-perm :: ‹VDMInt ⇒ VDMInt ⇒ VDMInt ⇒ ›, 其中 ‹
pre-perm m n r ≡ ((0 < r ∨ 0 < n) ⟶m+n+r > 0)› 引理
pre-perm-defs =pre-perm-def inv-VDMInt-def inv-True-def
```

```
function(domintros) perm :: ‹VDMInt ⇒ VDMInt ⇒ VDMInt ⇒
VDMInt›, 其中 ‹perm m n r = (if pre-perm m n r then if 0 < r then
perm m (r−1) n else if 0 < n then perm r (n−1) m else m else
undefined)›
```

```
定义 ‹perm-wf-rel ≡
  {((m, r−1, n), (m, n, r)) | m r n 。 pre-perm m n r ∧ 0 < r} ∪ {((r,
  n−1, m), (m, n, r)) | m r n 。 pre-perm m n r ∧ ¬ 0 < r ∧ 0 < n}›
```

其度量关系包含每个递归调用对应的元素，并按相应的 if-then 情况进行筛选。证明这种包含多个递归调用的度量关系具有良基性，需要进一步的证明工程，下一条引理将对此作出说明。

```
引理 l-perm-wf-rel-VDM-measure: ‹perm-wf-rel ⊆
measure (λ(m, r, n) 。 nat (max 0 (m+r+n)))›
```

```
引理 l-perm-wf-rel: ‹良基 perm-wf-rel›
```

Isabelle 的 度量 定义将给定函数的严格小于关系的逆像投影为递归度量关系[7] 。此处，VDM 定义的度量以这种度量函数投影的形式给出。这向 VDM用户凸显了 VDM 与 Isabelle 递归度量之间的联系（及其差异）。

如果 pre-perm 专门抑制参数的负和，那么这里的 SCNP设置即可正常工作。这一点起初并不明显。加入前置条件后，sledgehammer 便能找到终止性证明。

最后定义竹内函数[8], ，其中同时包含置换和内部递归；接下来将介绍该函数，而关键部分是使用多重集 [6] 的 SCNP设置。这是必要的，因为有序字典序积不足以描述这种递归类型。它没有隐式前置条件。翻译策略适用于这些定义，但几乎不可能自动找到证明。

function(domintros) tak :: ‹VDM 整数 ⇒ VDM 整数 ⇒ VDM 整数 ⇒ › where ‹tak x y z = (if x ≤ y then y else tak (tak (x−1) y z) (tak (y−1) z x) (tak (z−1) x y))›

下面是一个示例，说明在进行终止性证明之前，必须将定义域谓词保留为假设。

引理 tak-pcorrect:
‹tak-dom (x, y, z) ⟹tak x y z = (if x ≤ y then y else if y ≤ z then z else x)›

每种情况（包括非递归调用）都在 SCNP设置中表示，然后使用它们的度量字典序组合，作为终止性证明的度量关系。

定义 tak-m1，其中 ‹tak-m1 = (λ(x, y, z)。 if x ≤ y then 0 else 1)› 定义 tak-m2，其中 ‹tak-m2 = (λ(x, y, z)。 nat (Max {x, y, z} − Min {x, y, z}))› 定义 tak-m3，其中 ‹tak-m3 = (λ(x, y, z)。 nat (x − Min {x, y, z}))›

终止性证明使用度量关系 tak-m1 <*mlex*> tak-m2 <*mlex*>tak-m3 <*mlex*> ∅。四种情况都需要用户定义的引理。

有了全定义版本后，归纳和化简规则便可用来证明它更简单的等价形式。

定理 tak-correct: ‹tak x y z = (if x ≤ y then y else if y ≤ z then z else x)› by (归纳 x y z rule: tak.induct) auto

[7] 也就是说，度量 = inv-image less-than。逆像定义为 inv-image r f = {(x, y) | (f x, f y) ∈ r}。[8] https://isabelle.in.tum.de/library/HOL/HOL-Examples/Functions.html[9] 度量字典序积 f <*mlex*> R 表示为字典序积 f <*mlex*> R 的逆像，即 = inv-image (less-than <*lex*> R)(λx. (f x, x))。

5.6 相互递归

最后，我们处理相互递归。VDM 对相互递归定义几乎没有限制。我们复用 VDM 类型检查器的循环度量发现算法，以推导相关的递归调用。请注意，此算法并不完备，可能无法发现某些递归调用，例如那些位于某个 λ 项结果中的调用。如果递归调用图发现失败，用户就必须至少为一个相互递归定义添加 @IsaMutualRec 注解，并列出其中涉及的所有预期函数名。

相互递归定义的证明义务会引用彼此的度量函数。Isabelle 要求所有相关函数都在单个定义中给出，并在证明设置中使用 Isabelle 的和类型（或并类型）。必要时，需要进行终止性证明设置，而自动化能力有限。下面是一个简单示例及其转换，它使用 fun 语法，因此不需要进一步的证明义务。请注意，在此例中使用 fun 语法时，我们将隐式前置条件声明为化简规则。

```
--@IsaMutualRec({odd}) 偶数: 自然数-> 布尔偶数(n) == 如果 n
 = 0 则为真 否则奇数(n-1) 度量 n; --@IsaMutualRec({even}) 奇数:
自然数 -> 布尔奇数 (n) == 如果 n = 0 则为假 否则偶数 (n-1) 度量
n;
```

定义pre-even :: ‹VDMNat ⇒ ›其中 [终止性化简]: ‹pre-even n ≡ VDM 自然数不变式 n› 定义 pre-odd :: ‹VDMNat ⇒ ›其中 [终止性化简]: ‹pre-odd n ≡ VDM 自然数不变式 n› fun (domintros) 偶数 :: ‹VDMNat ⇒ ›和 奇数 :: ‹VDMNat ⇒ ›其中
‹偶数 n = (if pre-even n then (if n = *0* then Trueelse 奇数 (n− *1*)) else 未定义)›| ‹ 奇数 n = (if pre-odd n then (if n = *0* then False else 偶数 (n− *1*)) else 未定义)›

6 讨论与结论

本文介绍了将 VDM 中针对基本类型、集合和映射的递归函数以及相互递归转换为 Isabelle 的翻译策略。我们展示了该策略在复杂递归中的工作方式。

复杂递归示例表明，VDM 递归度量扩展可能需要结合度量关系和函数。完整的 VDM 和 Isabelle 源码及证明可在 VDM 工具包代码库 `RecursiveVDM*.thy`[10] 中找到。

后续工作。我们正在 vdm2isa 插件中实现这一翻译策略，该插件预计很快就会推出。

致谢。我们感谢 Stephan Merz 与我们讨论在 Isabelle 中进行复杂良基递归证明的要点，也感谢 Nick Battle 与我们讨论 VDM 递归度量的局限性。Peter Gorm Larsen 也想向 Poul 致谢

[10] https://github.com/leouk/VDM_工具包

感谢 Jensen基金会为“网络物理系统数字孪生”（DiT4CPS）项目提供资金。

参考文献

1. Battle, N.：VDMJ用户指南。技术报告，富士通服务有限公司，英国（2009） 2. Battle, N.：VDMJ注释指南（2022） 3. Ben-Amram, A.M., Codish, M.：基于SAT的方法：通过全局秩函数实现大小变化终止。在：TACAS。第218–232页。Springer（2008） 4. Freitas, L.：VDM翻译策略：翻译至Isabelle（Nill Full游戏）（2016） 5. Krauss, A.：在Isabelle/HOL中定义递归函数。慕尼黑工业大学（2021） 6. Krauss, A., Heller, A.：针对SCNP终止性的机械化证明重构。技术报告，慕尼黑工业大学（2012） 7. Lee, C.S., Jones, N.D., Ben-Amram, A.M.：程序终止性大小变化原理。在：POPL。第81–92页。POPL ’01，ACM，美国纽约（2001） 8. Rask, J.K. 等：Visual Studio Code中的高级VDM支持。在：第20届International Overture Workshop。第35–50页（2022）