arXiv:2303.04148v1 [cs.AI] 2 Mar 2023

将建议上层合并本体转换为带类型的一阶形式

Adam Pease[1]

[1]Articulate Software
apease@articulatesoftware.com

1 引言

想象一台支持语音的家用机器人，它能够拾起物体并将其从一个房间搬到另一个房间。主人可能会说：“把我的红色手推车拿起来，放到车库里。”但机器人听到的可能是“把我的红色汽车拿起来，放到车库里。”，也可能是原句。推理系统可能会通过推理来消除这句话的歧义或进行纠正：典型汽车的重量为 1.5 吨，也就是 3000 磅，而机器人的承载能力是 100 磅，因此另一种说法才是实际说出的内容。系统必须能够针对一个从未被问过的问题构造答案（“机器人能搬运一辆汽车吗？”），执行一些涉及单位换算的简单计算，以理解 3000 磅 >100磅，并在需要时解释答案（“我让你搬我的车，你为什么没搬？”）

定理证明（或自动推理）是指通过数理逻辑中的理论来回答问题的过程。它不同于搜索和信息检索（例如使用 Google 或 Apache Lucene 进行的搜索和信息检索）：前者会结合一些简单的事实和规则，综合得出一个答案，即使这个问题以前可能从未被问过；而后者则是将查询与最相似的既往查询及其答案进行匹配。定理证明还会提供解释（即证明），说明答案是如何推导出来的。在这项工作中，我们使用建议上层合并本体（SUMO）（Niles 和 Pease，2001；Pease，2011）作为理论，即由数理逻辑中的一组陈述构成的理论。数理逻辑有许多种。其中，最常用于自动推理、且在语言表达能力与效率之间取得良好平衡的一种，称为一阶逻辑（FOL）（或谓词演算）。

通用一阶定理证明器历来不擅长进行算术证明，这与罗素悖论相关的问题有关（Baldwin 和 Lessmann，2018）。恩斯特·策梅洛发现的悖论解决方案，是将数与其他事物划分为不同的类型。自动推理领域中的这一解决方案，如今已通过一种名为

带类型的一阶形式（TFF）的语言得到了很大程度的解决（Sutcliffe 等，2012）[1]，并已在若干最优秀的现代定理证明器中实现，其中包括曼彻斯特大学的 Vampire（Kov´acs 和 Voronkov，2013）。将 SUMO 的原生形式化表示转换为 TFF 的转换器，将为使用数和算术进行实际自动推理开辟许多新机会。

要对例如机器人的能力进行有用且非平凡的推理，或进行问答，我们需要的不仅是能够进行算术计算（以及一阶逻辑推理）的语言，还需要一个内容丰富、非平凡的公理体系，其中包含关于现实世界的信息，以便据此回答这类问题。因此，我们需要使用 SUMO。它包含一套全面而多样的关于世界的逻辑陈述，约有 20,000 个概念和 80,000 条逻辑陈述，规模足以回答涉及广泛主题的有趣问题。

在早期的工作中，我们描述了如何将 SUMO 转换为 TPTP 的严格一阶语言（Pease 和 Schulz，2014；Sutcliffe，2007）。SUMO 具有丰富的类型结构，所有关系的参数都受到类型限制。转换为 TPTP 时，我们在 TPTP 中以公理化方式实现了一种带排序（带类型）的逻辑：修改 SUMO 中的所有蕴含式，使其中出现的任何变量都带有类型限制。我们还需要进行许多其他转换步骤，这些步骤已在我们早期的论文中介绍，也是转换为 TFF 的一部分；但由于它们并非 TFF 转换所特有，因此此处不再详细重述。还需注意，SUMO 中所有严格意义上的高阶内容在转换为一阶形式时都会丢失，无论目标是 TPTP 还是 TFF。简而言之，转换步骤包括：

- 扩展“行变量”，从而可以在不限定关系的参数数量的情况下表述公理，这类似于 Lisp 的 @REST 构造
- 将“谓词变量”实例化为所有可能的值。任何在关系位置使用变量的公理都需要执行此操作。
- 扩展所有可变元数关系的元数，将其表示为一组关系；这些关系根据其固定参数数量而具有不同的名称
- 重命名作为其他关系参数给出的任何关系

感兴趣的读者可参阅前文，以了解更多详细信息和示例。

2 带类型的一阶形式

与 TPTP 一样，TFF 形式也是有效的 Prolog 语法（尽管其语义显然并不相同）。TFF 有五种互不相交的类型：整数、实数、有理数、

[1]请注意，这实际上是一系列逻辑；具体来说，我们将其转换为 TF0——带类型的一阶形式，并采用 0 级多态。

布尔值以及其他所有内容。在 TFF 语法中，它们分别称为 $int、$real、$rat、$o 和 $i。逻辑语句中使用的每个变量都必须声明为其中一种类型，否则默认会被视为类型 $i。

TFF 语言内置了基本算术函数和算术比较运算符。每个函数和运算符都是多态的——实际上，它们各自都是由三个不同运算符组成的集合，可分别处理整数、有理数和实数。对于 $o 和 $i，也定义了相等关系。

TFF 的创建者计划加入定义子类型（子类型）的功能，但规范中尚未定义这一功能，任何定理证明器也尚未实现它。另一个问题是，所有类型彼此不相交，而 SUMO 支持多重继承，因此这两种类型系统并不匹配。因此，我们必须继续像在 TPTP 中那样，在 TFF 中以公理方式实现 SUMO 类型系统的大部分内容；但对于整数、有理数和实数，则采用 TFF 类型系统进行特殊处理，以便使用其算术和比较运算符。

3 方法

SUMO 的数学运算符接受通用的量类型，该类型有物理量（带计量单位的数）以及实数和整数等子类（其他子类见图 1）。因此，我们有如图 2 所示的规则

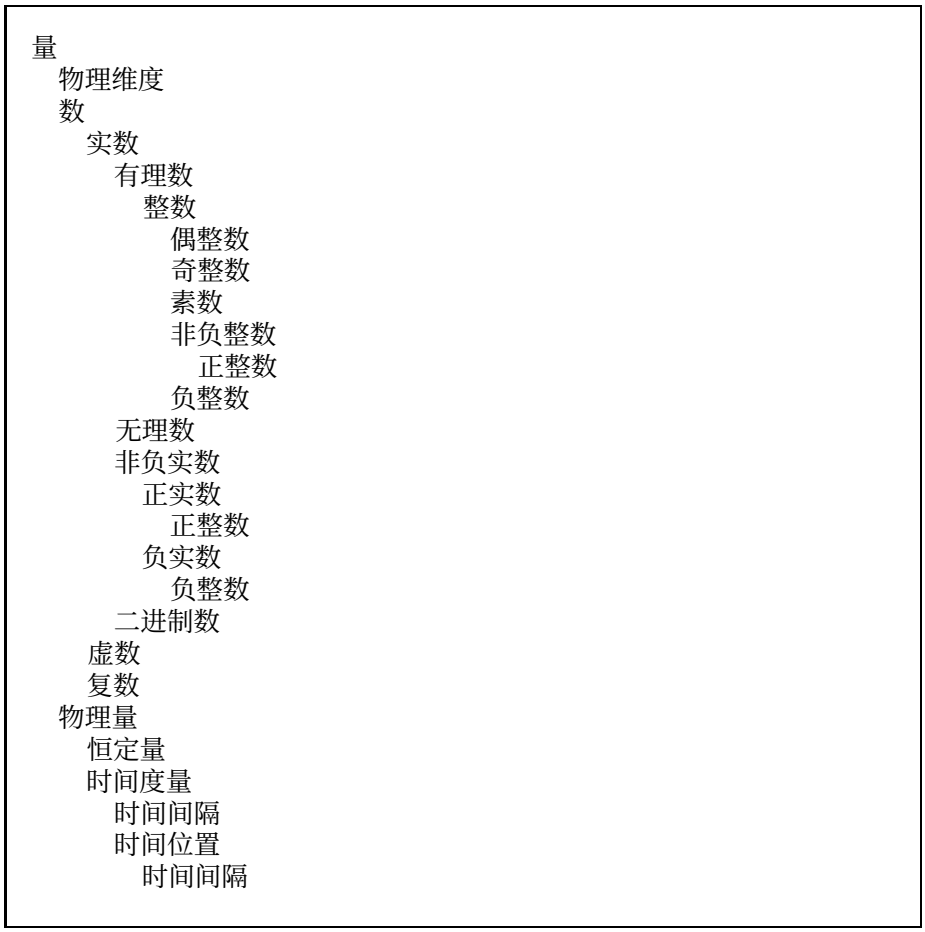


图 1：SUMO 的数量类型层级（略作简化）

```
(=>
  (and
    (实例 ?PM 颗粒物)
    (部分 ?Particle ?PM)
    (近似直径 ?Particle
      (测量函数 ?Size 微米))
    (大于 10 ?Size)
    (大于 ?Size 2.5))
  (instance ?PM 粗颗粒物)))
```

图2：包含算术比较的规则

其中，greaterThan 用于比较整数或实数，以及类似图3所示的规则

```
(=>
  (and
    (instance ?AREA 沙漠气候区)
    (subclass ?MO 月份)
    (时期平均温度 ?AREA ?MO ?TEMP)
    (大于 ?TEMP
      (测量函数 18 摄氏度)))
  (实例 ?AREA 亚热带沙漠气候区))
```

图3：比较带单位的数的规则

其中，大于用于比较量（带单位的数），例如“18 度”。

3.1 类型提升

由于 TFF 没有类型层次结构，因此关系的参数不能同时接受数和量（带单位的数）。因此，我们采用的方法是将使用数的关系与不使用数的关系分开。

取所有域规范为 Quantity 或 Quantity 子类的关系。如果域是 Integer 的子类，则将其提升为 Integer，并在前件中添加约束函数。如果是 RationalNumber，则保持类型不变。如果是 NonnegativeRealNumber 的任意子类，且同时不是 Integer 的子类，则将其提升为 RealNumber。所有包含 SUMO 关系且具有对应 TFF 关系的公理都需要复制

并使用原生 TFF 关系，前提是该 SUMO 关系属于 Quantity，但不属于 RationalNumber、RealNumber 或 Integer。

3.2 类型调整算法

算法步骤如下

1. 收集所有变量的类型。2. 如果相等关系或不等关系（即 SUMO 的“扩展到数量的关系”的一个实例）有一个参数比该关系规定的参数类型更具体，则进一步约束所有类型，除非该参数已经是实数（在这种情况下，即使另一个参数是整数，也不要对其加以约束）。3. 按照上文所述，将作为整数或实数特化类型的类型“提升”。4. 如果一个数是整数（没有小数点），且不是参数，也不处于与已定义整数类型进行比较的语句中，则在其后添加“.0”，使其成为实数，并将其类型记录为实数。5. 如果“扩展到数量的关系”的参数已被约束为这些类型（或其子类），则为该关系名称添加后缀。后缀由参数编号加上“In”“Re”或“Ra”之一组成，分别表示整数、实数或有理数。6. 如果在此阶段，公理的任何变量类型在 SUMO 层级结构中属于“量”的子类，但不属于“数”的子类，则创建该公理的两个版本：一个保留所有谓词的原始名称；另一个则为每个没有 TFF 等价项的 SUMO 谓词添加类型规范后缀（如上一步所述），并将具有 TFF 等价项的谓词转换为相应的等价项。7. 转换为 TFF 语法，包括将所有具有对应原生 TFF 关系或算术运算符且带有类型规范后缀的关系进行转换。

3.3 相关公理示例

那么，图2将变为图4，图3则变为图5。

```
!  [V_尺寸 : $实数  ，V_颗粒 : $i        ，V_PM : $i]:
        （s__实例(V_PM      ，s__颗粒物) &
         s__part(V_颗粒,V_PM) &
         s__近似直径(V_颗粒,
              s__测量函数(V_尺寸   ,s__微米)) &
         $大于(10.0,V_尺寸) &
         $大于(V_尺寸,2.5)  =>
       s__实例(V_PM      ,s__粗颗粒物))
```

图4：采用原生比较运算符的 TFF 公理

```
!  [V_区域:  $i  ，V_MO : $i  ，V_TEMP : $i  ]
         s__instance(V_AREA, 沙漠气候区) &
         s__子类(V_MO      ，月份) &
         s__时期平均温度(V_AREA, V_MO, V_TEMP) &
         s__大于(V_TEMP,
              s__测量函数(18,s__摄氏度)))  =>
    s__实例(V_AREA, s__亚热带沙漠气候区))
```

图 5：使用 SUMO 比较运算符的 TFF 公理

我们已经有了类似图6所示的转换公理。

```
(=>
   (equal ?AMOUNT
      (MeasureFn ?X Joule))
   (equal ?AMOUNT
      (MeasureFn
         (乘法函数 0.0002778 ?X) 瓦特)))
```

图 6：单位换算公理

因此，我们需要为每个相等关系和不等关系添加一条新公理来比较量，例如图7所示。

```
(=>
  (and
    (相等 ?Q1 (测量函数 ?I1 ?U))
    (相等 ?Q2 (测量函数 ?I2 ?U))
    (大于 ?I1 ?I2))
  (大于 ?Q1 ?Q2))
```

图7：单位比较公理

不过，我们可以像图8那样使用谓词变量，更简洁地实现这一点。

```
(=>
  (and
    (实例 ?REL 扩展到数量的关系)
    (相等 ?Q1 (测量函数 ?I1 ?U))
    (相等 ?Q2 (测量函数 ?I2 ?U))
    (?REL ?I1 ?I2))
  (?REL ?Q1 ?Q2))
```

图8：适用于所有运算符的单位比较公理

这将生成前述公理以及所有其他相关公理（图 9）。

```
(实例 相等 扩展到数量的关系)
(实例 大于 扩展到数量的关系)
(实例 大于或等于 扩展到数量的关系)
(实例 小于 扩展到数量的关系)
(实例 小于或等于 扩展到数量的关系)
```

图 9：所有接受数和量的关系

图8中的公理加上图 9 中的关系语句，将使 Sigma 中的谓词变量实例化过程生成图7中的公理（以及许多其他公理）。随后，当 SUMO 的 greaterThan 带有严格为整数或实数的参数时，Sigma 中后续的 TFF 转换步骤会将其转换为 $greater。

在调用处理整个 SUMO 到 TFF 的转换（而非子集）时，我们需要将数值类型多态视为一种“宏”——即每个使用 TFF 内置数学运算符的语句。

3.4 算法的详细示例

让我们以图 10 中的公理及其所含函数的参数类型规范为起点，进行逐步翻译

```
(=>
    (相等。
      (余数函数 ?NUMBER1 ?NUMBER2)
      ?NUMBER)
   (相等
      (符号函数 ?数2)
      (符号函数 ?数)))

(domain 余数函数 1 量)
(domain 余数函数 2 量)
(range 余数函数 量)

(domain 符号函数 1 实数)
(range SignumFn 整数)
```

图 10：包含函数及其类型签名的公理

对于步骤 1，当前的 Sigma 转换会知道要将每种类型约束为公理中最具体的值。在本例中，这意味着 ?NUMBER 和 ?NUMBER2 由于都是 SignumFn 的参数，因此必须是实数，因为 SignumFn 接受一个必须为实数的参数——(domain SignumFn 1 RealNumber)。我们将得到一个列表：[?NUMBER=RealNumber, ?NUMBER1=Quantity, ?NUMBER2=RealNumber]

对于步骤 2，我们也有 RelationExtendedToQuantitiesRemainderFn 和 equal，此时必须将其参数约束为相同的最低公共类型。在 [?NUMBER1=Quantity, ?NUMBER2=RealNumber] 中，该类型是实数。因此，变量及其类型的列表变为 [?NUMBER=RealNumber, ?NUMBER1=RealNumber, ?NUMBER2=RealNumber]。

步骤 3 不适用，因为变量列表中没有 Integer 或 RealNumber 的特化。步骤 4 不适用，因为没有字面数字。由于我们已修改 RemainderFn 的参数，因此需要在步骤 5 中将其重命名为 RemainderFn 0Re1Re2Re。然后，我们需要检查 equal。请注意，对于每个返回某种数值类型的函数，我们都必须跟踪其修改后的返回类型。在本例中，虽然 RemainderFn 返回的是

量，但它的参数已将其约束为返回实数。因此，我们需要检查 equal 的参数是否相同，并在必要时对其进行约束。在本例中，?NUMBER 已经是实数，无需进一步约束。

由于不存在类型为数的变量，因此不适用步骤 6。现在我们有图 11

```
(=>
    （相等
     (余数函数      __0Re1Re2Re ?NUMBER1 ?NUMBER2)
     ?NUMBER)
   相等
     (符号函数 ?NUMBER2)
     (符号函数 ?NUMBER)))
```

图 11：函数名称展开

最后一步是将其转换为 TFF 语法，并在适用时将关系和函数转换为其原生 TFF 名称。结果如图12所示。

```
!  [V_数 : $实数          ，V_NUMBER1 : $real   ，
    V_数2 : $实数] :
          ($equal(s__余数函数__0Re1Re2Re(V_NUMBER1,V_NUMBER2),
                  V_数)  =>
          ($equal(s__符号函数(V_NUMBER2),
                  s__符号函数(V_NUMBER))))
```

图12：TFF中带有原生比较运算符的公理

我们来看一个映射到某些 TFF 内置函数或关系的公理示例。对于 greaterThan，我们可能会遇到在 SUMO 公理中与单位一起使用的情况，也可能遇到不带单位使用的情况。不带单位的情况将映射到 TFF 内置的 $greater 关系，如图 13 所示。

```
(=>
  (and
    (实例 ?X ?Y)
    (子类 ?Y 纯物质)
    (沸点 ?Y
      (测量函数 ?BOIL 开尔文度))
    (熔点 ?Y
      (测量函数 ?MELT 开尔文度))
    (测量 ?X
      (测量函数 ?TEMP 开尔文度))
    (大于 ?TEMP ?MELT)
    (小于 ?TEMP ?BOIL))
  (属性 ?X 液体))
```

图 13：另一个包含数值比较的规则

步骤 1 的结果如下，其中 '+' 表示类而非实例 - [?X=量，?Y=纯物质+，?BOIL=实数，?TEMP=实数，?MELT=实数]。对于步骤 2，我们有 greaterThan 和 lessThan，但它们的参数全都是实数，因此无需更改。步骤 3 不适用。对于步骤 4，没有数，因此也不适用。步骤 5 中，我们有 greaterThan 和 lessThan，它们的参数都是实数，因此将分别重命名为 greaterThan 0Re1Re2Re 和 lessThan 0Re1Re2ReReal。重命名结果如图 14 所示。步骤 6 不适用。对于步骤 7，我们将 greaterThan 0Re1Re2Re 重命名为 $greater，将 lessThan 0Re1Re2Re 重命名为 $less，并转换为 TFF 语法，结果如图 15 所示。

```
(=>
  (and
    (实例 ?X ?Y)
    (子类 ?Y 纯物质)
    (沸点 ?Y
      (测量函数 ?BOIL 开尔文度))
      (熔点 ?Y
        (测量函数 ?MELT 开尔文度))
      (测量 ?X
        (测量函数 ?TEMP 开尔文度))
      (大于__0Re1Re2Re ?TEMP ?MELT)
    (小于      __0Re1Re2Re ?TEMP ?BOIL))
  (属性 ?X 液体))
```

图 14：用实数重命名后包含数值比较的规则

```
!  [V_X : $i  ,  V_Y : $i   ,  V_BOIL : $real  ,
    V_TEMP : $real  ,  V_MELT:  $实数]:
       (s__实例(V_X,V_Y) &
        s__子类(V      _Y,s__纯物质) &
        s__沸点(V_Y,
                          s__测量函数(V_BOIL,s__开尔文度)) &
        s__熔点(V_Y,
                          s__测量函数(V_MELT,s__开尔文度)) &
        s__测量(V     _X,
                      s__测量函数(V_TEMP,s__开尔文度)) &
        大于(V_TEMP,V_MELT) &
        小于(V_TEMP,V_BOIL))  =>
       s__属性(V_X,s__液体)
```

图 15：重命名运算符后包含数值比较的 TFF 规则

另一个案例是图 16

```
(<=>
  (and
    (实例 ?LD 液体滴)
    (近似直径 ?LD
      (测量函数 ?Size 微米))
    (小于 500 ?Size))
  (实例 ?LD 小液滴))
```

图 16：包含字面数字的公理

对于步骤 1，我们有 [?LD=小液滴，?尺寸=实数]。对于步骤 2，由于 ?尺寸 已经是实数，因此不需要将其更改为实数，例如“500.0”，因为它们都是小于的参数。步骤 3 不适用。在步骤 4 中，我们将“500”提升为“500.0”。在步骤 5 中，我们将小于更改为小于 0Re1Re2Re，以符合其参数类型。步骤 6 不适用。对于步骤 7，我们将小于 0Re1Re2Re 转换为 $less，并将其翻译为 TFF 语法，结果如图 17 所示。

```
!  [V_尺寸 : $实数  ，V_LD：$i]:
  ((s__实例(V_LD,s__液体滴) &
    s__近似直径(V_LD,
                        s__测量函数(V_尺寸  ,s__微米)) &
    小于(500.0,V_尺寸))  =>
  s__instance(V_LD,s__小液滴)))

!  [V_Size : $实数  ，V_LD：$i]:
  (s__instance(V_LD,s__小液滴)  =>
   (s__instance(V_LD,s__液体滴) &
    s__近似直径(V_LD,
                        s__测量函数(V_尺寸  ,s__微米)) &
    小于(500.0,V_尺寸)) out
```

图 17：含字面数字的 TFF 公理

```
(=>
    (instance ?X NegativeInteger)
    (greaterThan 0 ?X))
```

图18：包含负整数的公理

对于图18，转换步骤依次如下

1. [?X=NegativeInteger]2. 不适用 3. 将负整数提升为整数，因此得到 [?X=Integer]4. 不适用 5. 我们将 greaterThan 转换为 greaterThan 0In1In2In 6. 不适用 7. TFF 语法转换结果见图19

```
! [V_X : $int] :
    (s__instance(V_X,s__负整数) =>
     $大于(0,V_X))
```

图19：带有负整数的 TFF 公理

但这行不通，因为 instance 接受的是 $i 类型，而不是 V X 的类型 $int。

4 对翻译算法的修改

实际上，我们需要抑制所有涉及 Integer 和 RealNumber 下级类型的公理，但仅限于这些公理是对这些术语的定义时！否则，我们需要让这些定义成为对整数和实数的限制。因此，我们为 Integer 和 RealNumber 下的所有类型建立一个定义性约束表。该表将由图20和图21组成。

```
(=>
    (实例 ?NUMBER 偶整数)
    （相等
        (余数函数 ?NUMBER 2) 0))

(=>
    (实例 ?NUMBER 奇整数)
    （相等
        (余数函数 ?NUMBER 2) 1))

(=>
    (实例 ?素数 素数)
    (对于所有 (?数)
        (=>
            （相等
                (余数函数 ?素数 ?数) 0)
            (or
                (相等 ?数 1)
                (相等 ?数 ?素数)))))

(=>
    (实例 ?X 非负整数)
    (大于 ?X -1))

(=>
    (实例 ?X 负整数)
    (大于 0 ?X))

(=>
    (实例 ?X 正整数)
    (大于 ?X 0))

(<=>
    (实例 ?NUMBER 正实数)
    (and
        (大于 ?NUMBER 0)
        (实例 ?NUMBER 实数)))

(=>
    (实例 ?NUMBER 正实数)
    （相等
        (符号函数 ?NUMBER) 1))
```

图20：数的定义 15

```
(<=>
     (实例 ?NUMBER 负实数)
     (and
          (小于 ?NUMBER 0)
          (实例 ?NUMBER 实数)))

(=>
     (实例 ?NUMBER 负实数)
      (相等
          (符号函数 ?NUMBER) -1))

(<=>
     (instance ?NUMBER 负实数)
     (and
          (大于或等于 ?NUMBER 0)
          (instance ?NUMBER 实数)))
(=>
     (instance ?NUMBER 负实数)
     (or
           (相等
               (符号函数 ?NUMBER) 1)
           (相等
               (符号函数 ?NUMBER) 0)))
```

图 21：数的定义（续）

这样一来，还剩下一些数类型，它们除了定义之外没有任何公理，也不会用作参数类型：无理数、二进制数、虚数和复数。至少目前，我们将不对这些类型进行转换。还要注意，有两个子类型分别继承自实数和整数。它们是正整数（继承自正实数）和负整数（继承自负实数）。我们稍后会再讨论这个问题。

我们重新表述其中一些公理，使其保持一致；当双向蕴含和类型断言出现在两侧时，将其移除。这样便得到了一些修订后的公理，如图22所示。

```
(=>
    (实例 ?NUMBER 正实数)
    (大于 ?NUMBER 0))

(=>
    (实例 ?NUMBER 负实数)
    (小于 ?NUMBER 0))

(=>
    (实例 ?NUMBER 非负实数)
    (大于或等于 ?NUMBER 0))
```

图22：修订后的数的定义

我们对所有公理进行预处理，查找指定某个数的实例的前件，然后将后件缓存到一个按类型索引的映射中。我们还提取这些前件中用于约束类型的变量名。最终，我们得到分别如图23所示的前件和变量列表。

```
{正整数  =
    (大于 ?X 0),
 奇整数  =
    (相等 (余数函数 ?NUMBER 2) 1),
 负实数  =
    (相等 (符号函数 ?NUMBER) -1),
 偶整数  =
    (相等 (余数函数 ?NUMBER 2) 0),
 非负整数  =
    (大于 ?X -1),
 负整数  =
    (大于 0 ?X),

 素数  =
    (对所有 (?数)
      (=>
        (相等 (余数函数 ?素数 ?数) 0)
        (or
          (相等 ?数 1)
          (相等 ?数 ?素数)))),

 有理数  =
    (存在 (?INT1 ?INT2)
      (and
        (instance ?INT1 整数)
        (instance ?INT2 整数)
        (equal ?NUMBER (除法函数 ?INT1 ?INT2)))),

 非负实数  =
    (or
      (equal (符号函数 ?NUMBER) 1)
      (相等 (符号函数 ?数) 0))}

{正整数  =                  X,
 奇整数  =                  数,
 负实数  =                  数,
 偶整数  =                  数,
 非负整数  =                X,
 负整数  =                  X,
 素数  =                    素数 ,
 有理数  =                  数    ,
 非负实数 = 数}
```

图23：所有数值类型条件

例如，我们会得到 [PositiveInteger=(greaterThan ?NUMBER 0)]。索引的值将作为前件添加到任何将该类型用于变量的公理中。例如，参见图24，其中第一个公理变为第二个公理，而所得的 TFF 版本则是第三个。

```
(=>
    (equal ?W
        (WeekFn ?N ?Y))
    (during ?W ?Y))

(=>
  (greaterThan ?N 0)
  (=>
    (equal ?W
        (周函数 ?N ?Y))
    (期间 ?W ?Y)))

! [V_N : $int , V_W : $i , V_Y : $i]:
  ($大于(V_N,0) =>
    ((V_W = s __周函数(V_N,V_Y)) =>
      s__期间(V_W,V_Y)))
```

图24：添加数字类型条件

对于规则后件中出现的数字子类型，我们采用不同的替换过程。如果后件中包含涉及某个数字子类型的实例陈述，则将其替换为图23列表中该子类型及其所有父类型的定义条件。例如，见图25，其中一个变量被判定为正实数，因此必须将其替换为相应的定义条件

```
(=>
     (测量 ?QUAKE
          (测量函数 ?VALUE 里氏震级))
     (实例 ?VALUE 正实数))

(=>
     (测量 ?QUAKE
          (测量函数 ?VALUE 里氏震级))
     (大于 ?VALUE 0))

!  [V_地震 : $i,       V_VALUE : $实数] :
   (s__测量(V_QUAKE,
                  s__测量函数(V_VALUE,s__里氏震级))  =>
   $大于(V_VALUE,0)
```

图25：在后件中添加数值类型

5 函数的类型传播

遗憾的是，如上所述，我们的翻译算法仍不完整。类型变化也会因函数而沿公式传播。以图26为例，其中有一个包含多个嵌套函数的公理，这些函数进而为其下方的变量赋予类型。但函数的返回类型也应对变量的类型构成约束。

```
(<=>
   相等
      (余数函数 ?NUMBER1 ?NUMBER2) ?NUMBER)
    （相等
      (AdditionFn
         (乘法函数
            (向下取整函数
               (除法函数 ?NUMBER1 ?NUMBER2))
               ?NUMBER2)
            ?NUMBER)
         ?NUMBER1))

{?NUMBER1=[量]，?NUMBER2=[量],?NUMBER=[量]}
```

图26：嵌套函数

这些函数的返回类型如图27所示。虽然只有向下取整函数的类型比量更受限制，但这些限制实际上会传播到整个变量集合，从而施加约束。

```
(domain RemainderFn 1 Quantity)
(domain RemainderFn 2 Quantity)
(range RemainderFn Quantity)

(domain AdditionFn 1 Quantity)
(domain AdditionFn 2 Quantity)

(domain 除法函数 1 量)
(domain 除法函数 2 量)

(domain 向下取整函数 1 实数)
(range 向下取整函数 整数)

(domain 除法函数 1 量)
(domain 除法函数 2 量)
```

图27：函数参数和返回类型

向下取整函数使除法函数的返回类型成为实数，进而使 ?NUMBER1 和 ?NUMBER2 成为实数；但由于 ?NUMBER2 是整数，

?NUMBER1 也必须是整数，因此除法函数的返回类型变为整数，这意味着我们需要将其重命名为 DivisionFn 0In1In2In，然后向下取整函数必须变为 FloorFn 1In2。由于 MultiplicationFn 0In1In2In 返回整数，因此 ?NUMBER 也必须是整数。余数函数随后也变为 RemainderFn 0In1In2In。结果就是图28所示的变量集和 TFF 翻译。

```
{?NUMBER1=[整数],?NUMBER2=[整数]，?NUMBER=[整数]}

! [V__NUMBER1:$整数,V__NUMBER2:$整数,V__NUMBER:$整数]:
  ((s__余数函数__0In1In2In(V__NUMBER1，V__NUMBER2) =
   V__数 =>
     $sum(
       $product(
         s__向下取整函数_1In(
           $商_e(V__数1, V__数2)),
         V__数2)      ,
       V__数) =
       V__数1) &
  ($求和(
    $乘积(
      s__向下取整函数_1In(
        $商_e(V__数1, V__数2)),
      V__数2)      ,
    V__数) =
    V__数1 =>
      s__余数函数__0在1在2在(V__数1      ，V__数2) =
        V__数))
```

图28：变量类型传播结果

接下来，我们还需要将原始术语上的所有公理复制到具有修改后的后缀的新术语上。

凡是参数类型介于量和整数之间的关系，都需要额外提供一个整数版本。参数介于量和实数之间的关系，则需要整数和有理数版本。参数为量、物理维度或数的关系，则需要整数、实数和有理数这三个版本，以及原始版本。

5.1 排序冲突

在 TFF 中，不等式必须用于单一排序。然而，在 SUMO 中，我们有图29所示的情况

```
(=>
   相等
      (上取整函数 ?NUMBER)
      ?INT)
   (not
      (存在 (?OTHERINT)
         (and
            (实例 ?OTHERINT 整数)
            (大于或等于 ?OTHERINT ?NUMBER)
            (小于 ?OTHERINT ?INT)))))
```

图29：类型冲突

CeilingFn 接受一个实数并返回大于或等于该参数的最小整数。greaterThanOrEqualTo 必须比较实数和整数，但这在 TFF 中是不允许的。

我们希望能够在图30中陈述以下内容

```
(! [V__数 : $实数,V__整数 : $int] : (s__上取整函数__0
In1ReFn(V__数) = V__整数 =>~ ? [V__其他整数:$i
nt]: ($greatereq(V__其他整数,V__数) & $小于(V__其他
整数 ,V__整数))))).
```

图30：TFF类型冲突

但由于 $greatereq 的参数类型不同，因此会产生错误。SUMO 中 FloorFn 的公理也存在类似问题。我们可以像图31那样使用 TFF 的类型强制转换函数来解决这个问题

```
(! [V__数 : $real,V__整数 : $int] : (s__上取整函数__0
In1ReFn(V__数) = V__整数 =>~ ? [V__OTHERINT:$int]:
($greatereq(V__OTHERINT,$to_ 实数 (V__ 数 )) & $l
ess(V__OTHERINT ,V__整数))))).
```

图31：通过类型强制转换消除TFF类型冲突

对于这些特定的公理，我们也可以使用 TFF 内置的 $ceiling 和 $floor 函数，从而彻底省去这些公理，因为它们有助于定义这些 TFF 内置函数。

混用量与基本数值类型时，也会出现另一种形式的问题，例如：

```
(=>
  (直径 ?圆 ?长度)
  (存在 (?一半)
    (and
      (半径 ?圆 ?一半)
      (相等 (乘法函数 ?一半 2) ?长度))))
```

图3：TFF类型冲突原文

例如，我们希望将“two feet”乘以 2，得到“four feet”。这等价于以下形式——

```
(=>
  (直径 ?圆 ?长度)
  (存在 (?N __一半 ?U __一半)
    (and
      (半径 ?圆 (测量函数 ?N __一半 ?U __一半))
      相等
        (测量函数 (乘法函数 ?N__一半 2) ?U__一半)
        ?长度))))
```

图33：类型冲突 TFF 修改

其中，带单位的量的数值部分乘以数值参数。

6 结论

我们希望，将一个庞大而全面的理论转换为 TFF 时所遇到的大量实际挑战，能够推动未来版本的 TFF 得以改进，从而使涉及算术的定理证明更加直接。尤其是，若能具备类型层次结构，将大大简化所需的转换步骤。

这套转换软件以 Java 编写，是 Sigma 知识工程环境的一部分；该环境与建议上层合并本体（SUMO）一同发布在 GitHub 上，网址为 https://github.com/ontologyportal。该网站还提供了一份扩展版技术报告，详细介绍了更多转换细节；由于篇幅所限，本文无法纳入这些内容。

7 附录：SampleProof

这里展示了引言中示例问题的一个算术证明。我们断言，机器人只能搬运重量小于 100 磅的物体（公理 f1），随后断言存在一个事件，其中机器人搬运了一个重 1.5 吨的物体（公理 f2 和 f4）。我们将此查询提交给 Vampire 定理证明器。可以看到，在步骤 f71 中，它应用了 f8 中陈述的单位换算公理，将 100 磅换算为 0.05 吨；而 1.5 并不小于 0.05。由此便得出矛盾。

~$ /home/apease/workspace/vampire/vampire --mode casc -t 300 /home/apease/.sigmakee/KBs/Robot-small.tff

% 已找到反驳证明。感谢 Tanya！ % SZS 状态:
Robot-small 不可满足 % SZS 输出开始:
Robot-small 的证明

tff(func_def_0, 类型, s__测量函数__1ReFn: ($real * $i) > $i).

tff(f1,公理,(s__最大有效载荷容量(s__Robot1, s__测量函数__1ReFn(100.0, s__磅质量))), file('/home/apease/.sigmakee/KBs/Robot-small.tff',kb_SUMO_3179)).

tff(f2,公理,(s__测量(s__我的汽车,s__测量函数__1ReFn(1.5,s__吨质量))), file('/home/apease/.sigmakee/KBs/Robot-small.tff',kb_SUMO_3180)).

tff(f3,公理,(！ [X0,X1,X2,X3] : ((大于(X2,X3) & s__最大有效载荷容量(X1,X3) & s__测量(X0,X2)) =>~? [X4] : (s__实例(X4,s__搬运) & s__受事(X4,X0) & s__工具(X4,X1)))), file('/home/apease/.sigmakee/KBs/Robot-small.tff',kb_SUMO_268)). tff(f4,公理,(s__实例(s__Carry1,s__搬运) & s__受事(s__Carry1,s__我的汽车) & s__工具(s__Carry1,s__Robot1)), file('/home/apease/.sigmakee/KBs/Robot-small.tff',assert)). tff(f8,公理,(！ [X3 : $实数] : s__测量函数__1ReFn(X3,s__磅质量) =s__测量函数__1ReFn($商(X3,2000.0),s__吨质量)), file('/home/apease/.sigmakee/KBs/Robot-small.tff',kb_SUMO_3175b2)). tff(f9,公理,(！ [X6,X7 : $实数,X8 : $实数] : ($大于(X7,X8) =>大于(s__测量函数__1ReFn(X7,X6),s__测量函数__1ReFn(X8,X6)))), file('/home/apease/.sigmakee/KBs/Robot-small.tff',kb_SUMO_228b)). tff(f10,公理,(！ [X6,X7 : $实数,X8 : $实数]: (大于(s__测量函数__1ReFn(X7,X6),s__测量函数__1ReFn(X8,X6)) =>$大于(X7,X8))), file('/home/apease/.sigmakee/KBs/Robot-small.tff',kb_SUMO_228c)). tff(f11,plain,(！ [X6,X7 : $实数,X8 : $实数] : ($小于(X8,X7) =>大于(s__测量函数__1ReFn(X7,X6),s__测量函数__1ReFn(X8,X6)))), inference(求值,[] , [f9])). tff(f12,plain,(！ [X6,X7 : $实数,X8 : $实数]: (大于(s__测量函数__1ReFn(X7,X6),s__测量函数__1ReFn(X8,X6)) =>$小于(X8,X7))), inference(求值,[] , [f10])). tff(f14,plain,(！ [X0 : $实数] : s__测量函数__1ReFn(X0,s__磅质量) =s__测量函数__1ReFn($商(X0,2000.0),s__吨质量)),

inference(rectify,[],[f8])). tff(f16,plain,(! [X0,X1 : $real,X2 : $real] : ($小于(X2,X1) =>大于(s__测量函数__1ReFn(X1,X0),s__测量函数__1ReFn(X2,X0)))), inference(rectify,[],[f11])). tff(f17,plain,(! [X0,X1 : $real,X2 : $real]: (大于(s__测量函数__1ReFn(X1,X0),s__测量函数__1ReFn(X2,X0)) =>$小于(X2,X1))), inference(rectify,[],[f12])). tff(f20,plain,(! [X0,X1 : $real,X2 : $real]: (大于(s__测量函数__1ReFn(X1,X0),s__测量函数__1ReFn(X2,X0)) ⊢$小于(X2,X1))), inference(ennf_transformation,[],[f16])).

tff(f21,plain,(! [X0,X1 : $实数,X2 : $实数] : ($小于(X2,X1) ⊢大于(s__测量函数__1ReFn(X1,X0),s__测量函数__1ReFn(X2,X0)))), inference(ennf_transformation,[],[f17])).

tff(f22,plain,(! [X0,X1,X2,X3]: (! [X4] : (~s__实例(X4,s__搬运) ⊢s__受事(X4,X0) ⊢s__工具(X4,X1)) ∦(~大于(X2,X3) ⊢s__最大有效载荷容量(X1,X3) ⊢s__测量(X0,X2)))), inference(ennf_transformation,[],[f3])).

tff(f23,plain,(! [X0,X1,X2,X3]: (! [X4] : (~s__实例(X4,s__搬运) ⊢s__受事(X4,X0) ⊢s__工具(X4,X1)) ⊢大于(X2,X3) ⊢s__最大有效载荷容量(X1,X3) ⊢s__测量(X0,X2))), inference(flattening,[],[f22])).

tff(f25,plain,(s__最大有效载荷容量(s__Robot1,s__测量函数__1ReFn(100.0,s__磅质量))),

inference(cnf_转换,[],[f1])).

tff(f26,plain,(s__测量(s__我的汽车,s__测量函数__1ReFn(1.5,s__吨质量))), inference(cnf_transformation,[],[f2])).

tff(f27,plain,(s__工具(s__Carry1,s__Robot1)),
inference(cnf_transformation,[],[f4])).

tff(f28,plain,(s__受事(s__Carry1,s__我的汽车)),
inference(cnf_transformation,[],[f4])).

tff(f29,plain,(s__实例(s__Carry1,s__搬运)),
inference(cnf_transformation,[],[f4])).

tff(f30,plain,((! [X0:$real] : (s__测量函数__1ReFn(X0,s__磅质量) = s__测量函数__1ReFn($商(X0,2000.0),s__吨质量)))), inference(cnf_transformation,[],[f14])).

tff(f32,plain,((! [X2:小于,X0,X1:小于]: (大于(s__测量函数__1ReFn(X1,X0),s__测量函数__1ReFn(X2,X0)) | 小于(X2,X1)))), inference(cnf_transformation,[],[f20])).

tff(f33,plain,((! [X2:小于,X0,X1:小于]: (小于(X2,X1) | 大于(s__测量函数__1ReFn(X1,X0),s__测量函数__1ReFn(X2,X0))))), inference(cnf_transformation,[],[f21])).

tff(f34,plain,((! [X4,X2,X0,X3,X1]: (~实例(X4,s__搬运) | 受事(X4,X0) | 工具(X4,X1) | 大于(X2,X3) | 最大有效载荷容量(X1,X3) | 测量(X0,X2)))), inference(cnf_transformation,[],[f23])).

tff(f35,plain,((! [
X4,X0,X1]:

(~s__实例(X4,s__搬运) ⊢s__受事(X4,X0) ⊢
s__工具(X4,X1) ⊧sP0(X1,X0)))), inference(cnf_
transformation,[] ,[f35_D])).

tff(f35_D,plain,((! [X0,X1]: ((! [X4] : (~s__实例(X4,s__Carrying) ⊢s__受事(X4,X0) ⊢s__工具(X4,X1))) <=>~sP0(X1,X0)))), introduced(general_splitting_component_introduction,[new_symbols(naming,[sP0])])).

tff(f36,plain,((! [X2,X0,X3,X1]: (~大于(X2,X3) ⊢s__
最大有效载荷容量(X1,X3) ⊢s__测量(X0,X2) ⊢
sP0(X1,X0)))), inference(general_splitting,[] ,[f
34,f35_D])).

tff(f37,plain,((! [X2,X0,X3]: (sP1(X3,X0) ⊢s__测
量(X0,X2) ⊢大于(X2,X3)))), inference(cnf_
transformation,[] ,[f37_D])).

tff(f37_D,plain,((! [X0,X3]: ((! [X2] : (~s__测量(X0,X2) ⊢大于(X2,X3))) <=>~sP1(X3,X0)))), introduced(一般_拆分_组件_引言,[新_符号(命名,[sP1])])).

tff(f38,plain,((! [X0,X3,X1]: (~sP0(X1,X0) ⊢s__最大
有效载荷容量(X1,X3) ⊢sP1(X3,X0)))), inference(一般
_拆分,[] ,[f36,f37_D])).

tff(f39,plain,((! [X0,X1]: (~s__受事
(s__Carry1,X0) ⊧

~s__工具(s__Carry1,X1) ∣sP0(X1,X0)))),
inference(resolution,[] , [f35,f29])).

tff(f40,plain,((! [X0:$实数,X1:$实数]: (~小于($商(X0,2000.0),X1) ∣
大于(s__测量函数__1ReFn(X1,s__吨质量), s__测量函数__1
ReFn(X0,s__磅质量))))), inference(superposition,[] , [f32,f30])).

tff(f44,plain,((! [X0]: (sP0(X0,s__我的汽
车) ⊢s__工具(s__Carry1,X0)))),
inference(resolution,[] , [f39,f28])).

tff(f45,plain,((! [X2,X0:$实数,X1:$实数]: (大于(s__测量函数__1ReFn(X0,s__
吨质量), s__测量函数__1ReFn(X1,s__磅质量)) ⊢大于(s__测量函数__1
ReFn(X0,X2), s__测量函数__1ReFn($商(X1,2000.0),X2))))),
inference(resolution,[] , [f40,f33])).

tff(f47,plain,((! [X0,X1]: (~sP1(X1,s__我的
汽车) ⊢s__最大有效载荷容量(X0,X1) ⊢s__工
具(s__Carry1,X0)))), inference(resolution,
[] , [f44,f38])).

tff(f50,plain,((! [X2,X0,X1]: (~s__工具(s__
Carry1,X0) ⊢s__最大有效载荷容量(X0,X1) ⊢
s__测量(s__我的汽车,X2) ⊢大于(X2,X1)))),
inference(resolution,[] , [f47,f37])).

tff(f55,plain,((! [X0,X1]: (~s__最大有效载荷容量(s__
Robot1,X0) ⊢s__测量(s__我的汽车,X1) ⊢大于
(X1,X0)))), inference(resolution,[] , [f50,f27])).

tff(f60,plain,((! [X0]: (~s__测量(s__我的汽车,X0) ⊢大于(X0,s__测量函数__1ReFn(100.0,s__磅质量))))), inference(resolution,[] , [f55,f25])).

tff(f65,plain,(~大于(s__测量函数__1ReFn(1.5,s__吨质量), s__测量函数__1ReFn(100.0,s__磅质量))), inference(resolution,[] , [f60,f26])).

tff(f71,plain,((! [X1]: (~大于(s__测量函数__1ReFn(1.5,X1), s__测量函数__1ReFn($quotient(100.0,2000.0),X1))))), inference(resolution,[] , [f65,f45])).

tff(f72,plain,((! [X1]: (~大于(s__测量函数__1ReFn(1.5,X1), s__测量函数__1ReFn(0.05,X1))))), inference(求值,[] , [f71])).

tff(f78,plain,(~小于(0.05,1.5)),
inference(resolution,[] , [f72,f32])).

tff(f95,plain,($false), inference(evaluation,[] , [f78])). % SZS 输出结束：Robot-small 的证明 % ------------------------------ % 版本：Vampire 4.2.2（提交 6588b35，时间为 2018-07-1913:39:17 +0200） % 终止原因：反驳证明

% 已用内存 [KB]：9722 % 已用时间：0.003 秒 % ------------------------------ % ------------------------------
% 在 99.856 秒内成功

参考文献

Baldwin, J. T. 和 Lessmann, O.（2018）。什么是罗素悖论? https://www.scientificamerican.com/article/what-is-russells-paradox/。

Kov´acs, L. 和 Voronkov, A.（2013）。一阶定理证明与 Vampire。载于《第25届计算机辅助验证国际会议论文集》，CAV 2013 第8044卷，第1–35页，美国纽约州纽约市：施普林格纽约出版社。Niles, I. 和 Pease, A.（2001）。迈向标准上层本体。载于 Welty, C. 和 Smith, B. 编，《第二届信息系统形式本体国际会议论文集（FOIS-2001）》，第2–9页。

Pease, A.（2011）。《本体：实践指南》。Articulate Software 出版社，加利福尼亚州安格温。

Pease, A. 和 Schulz, S.（2014）。使用 Sigma KEE 3.0 为大型本体构建知识工程。载于《国际自动推理联合会议》。Sutcliffe, G.（2007）。TPTP、TSTP、CASC 等。载于《第三届计算机科学国际会议论文集：理论与应用》（CSR'07），第6–22页，柏林、海德堡：施普林格出版社。Sutcliffe, G.、Schulz, S.、Claessen, K. 和 Baumgartner, P.（2012）。带算术的一阶类型化TPTP形式。载于程序设计、人工智能与推理逻辑国际会议（LPAR 2012），第406–419页。