GSOS 语言的声明式验证器

Matteo Cimini马萨诸塞大
学洛厄尔分校 美国马萨诸塞州洛
厄尔 matteo_cimini@uml.edu

规则格式可以快速确立进程代数的元理论性质。因此，我们希望找出能够轻松表达规则格式的领域特定语言（DSL）。在此前的工作中，我们开发了 LANG-N-CHANGE，这是一种领域特定语言，提供了便捷的功能，用于浏览语言定义并从中检索信息。本文中，我们使用 LANG-N-CHANGE 编写了一个 GSOS 规则格式验证器，并在此过程中为 LANG-N-CHANGE 增添了合适的宏。我们的 GSOS 验证器简洁明了，只需几行代码。我们已用它验证了多个并发算子，确认它们符合 GSOS 格式。此外，我们的代码以声明式方式表达了该格式的限制。

1 引言

创建进程代数之后，语言设计者的工作还没有结束。理想情况下，语言设计者会努力证明所构建的进程代数具备所需的性质。根据进程代数的不同，值得探究的问题包括：互模拟等价是否为同余、语言是否具有确定性，或者其中某些算子是否满足交换律和结合律等代数定律，等等。

语言验证领域旨在开发方法和工具，以语言定义为输入，对其进行静态分析，并确定某项性质是否对该语言成立。在并发理论领域，基于结构操作语义（SOS）规范的规则格式 [34]最能体现语言验证的做法。规则格式指出，如果用于编写某种语言的结构操作语义规范的规则符合某些语法限制，那么就能保证某项语义性质成立。这种方法已用于自动推导强互模拟等价 [12, 25, 26, 40], 和弱互模拟等价 [11, 21, 23], 的同余性，并确立算子的代数定律 [4, 5, 18, 35], ; 它也用于推导确定性 [1]和有界非确定性 [22], 等全局性质，以及研究概率转移 [9, 19, 28] 和带绑定子的上下文 [6, 20, 41], , 等等。

因此，有必要找出合适的领域特定语言（DSL），以便规则格式的设计者更轻松地表达其格式并对其进行自动测试。这样，他们就能快速测试新想法，同时在一套测试中检验多个进程代数，并找到快速进行新格式原型设计的途径。在设计者仍在构建理论结果时，这些测试有助于调试新的规则格式。

遗憾的是，现有文献中没有专门用于表达规则格式的领域特定语言。本文聚焦于一种名为 LANG-N-CHANGE [30, 32], 的现有领域特定语言。它的设计目的并非语言验证，但其操作可以重新用于编写规则格式。LANG-N-CHANGE 是一种用于表达语言变换的领域特定语言：输入是语言定义和变换指令，输出则是对该语言的修改

IlariaCastellani 和AlcesteScalas（编）：第14届并发与通信中心软件编程语言方法研讨会（PLACES2023）。EPTCS 378，2023年，第14–25页，doi:10.4204/EPTCS.378.2

© M. Cimini 本作品根据知识共享署名许可协议授权。

（作为输入给出）。请看下方左侧的函数应用类型规则，以及右侧带有子类型的版本。

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 \qquad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\ e_2 : T_2}\ (\text{T-APP}) \quad \Longrightarrow \quad \frac{\Gamma \vdash e_1 : T_{11} \to T_2 \qquad \Gamma \vdash e_2 : T_{12} \qquad T_{12} <: T_{11}}{\Gamma \vdash e_1\ e_2 : T_2}\ (\text{T-APP'})$$

（T-APP'）是对（T-APP）的一种变换。LANG-N-CHANGE 提供了用于表达这种变换的语言特性。例如，LANG-N-CHANGE 可以表达这样的测试：检测 T_1 是否在（T-APP）中被使用了两次，并为这两个出现位置分别分配新的不同变量 T_{11} 和 T_{12} 。LANG-N-CHANGE 还包括创建前提 $T_{12} <: T_{11}$ 的操作。LANG-N-CHANGE 已应用于多个能够自动转换结构操作语义规范的案例研究，例如添加子类型 [16, 30], 、模式匹配 [32], 、引用 [32], 和渐进式类型 [31], , 以及从小步风格 [30], 和 CK 机器 [16] 自动推导大步语义。

为了表达语言变换，LANG-N-CHANGE 提供了用于浏览结构操作语义规则、其前提以及语言其他组成部分的细粒度操作。在本文中，我们探讨了使用 LANG-N-CHANGE 表达规则格式的想法。其思路是设计一种语言变换，最终不修改输入语言，而是使用 LANG-N-CHANGE 的操作来检验规则格式所规定的语法约束，并在不满足这些约束时抛出运行时错误。我们已使用 LANG-N-CHANGE 编写了一个 GSOS 格式 [12] 检查器。这是一种众所周知的规则格式，可以为许多具有常见算子的进程代数建立互模拟等价的同余性。为了更好地表达该格式所规定的某些测试，我们为 LANG-N-CHANGE 增加了适用的宏。这些宏既不是对核心语言的扩展，也不是对其求值器的扩展；它们会被解析并转换为 LANG-N-CHANGE 的普通操作。

我们使用该工具验证了已知符合 GSOS 格式的常见并发算子，例如 CCS 并行算子 [29], CSP 中的同步并行组合[27], 以及 ACP 的投影算子 [10]。（第 4 节提供了我们的完整测试列表。）总计而言，我们验证了 18 个并发算子。我们还进行了一系列负面测试。具体来说，我们将不符合 GSOS 格式的语言作为输入提供给该工具，并确认该工具确实拒绝了这些语言。

我们的 GSOS 验证器仅包含 6 行代码，因此十分简洁。此外，我们的代码以声明式方式表达了 GSOS 的语法限制。本文的工作表明，LANG-N-CHANGE 可能是表达规则格式的有用工具。

本文的组织结构如下。第 2 节概述 LANG-N-CHANGE。第 3 节介绍我们新增的宏以及 GSOS 验证器。第 4 节讨论我们的评估结果。第 5 节讨论相关工作，第 6 节总结全文。

2 LANG-N-CHANGE 概述

本节重复介绍 LANG-N-CHANGE [30, 32] 的相关背景。图 1 展示了 LANG-N-CHANGE 的工具流程。输入由两个要素组成：语言定义和语言转换。输出要么是语言定义，要么是错误消息。

哪些语言定义？LANG-N-CHANGE 适用于用结构操作语义定义的语言。输入是结构操作语义的转移系统规范（含负迁移）的文本表示 [13]。

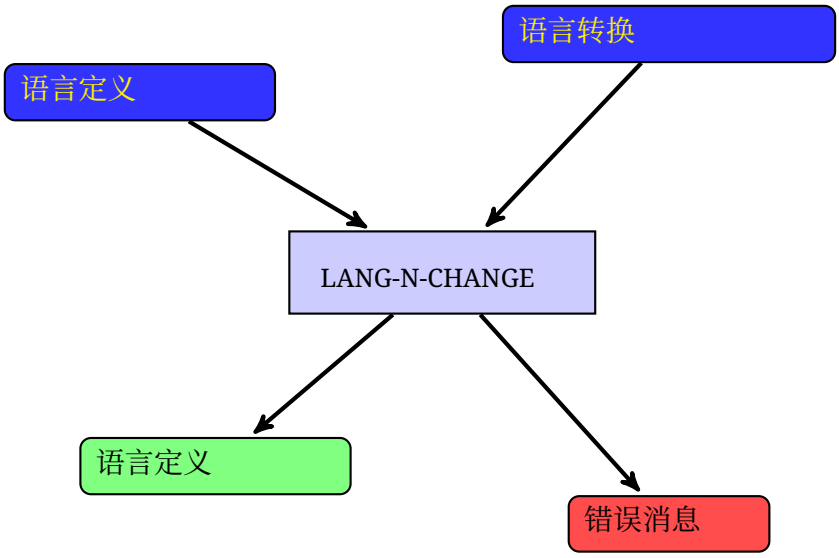


图 1：LANG-N-CHANGE 的工具流程

以下是输入 LANG-N-CHANGE 的语言示例：一种包含前缀算子、交错算子和顺序算子的进程代数。

标签 L ::= (a) | (b) | (c) 进程 P ::= (null) | (a P) | (b P) | (c P) | (par P P) | (sequence PP)

```
(a P) --(a)--> P.
```

（并行组合 P1 P2）--(a)-->（并行组合 P1’P2） <== P1 --(a)--> P1’。
（并行组合 P1 P2）--(a)-->（并行组合 P1 P2’） <== P2 --(a)--> P2’.

(顺序组合 P1P2) --(a)--> (顺序组合P1’ P2) <== P1--(a)--> P1’.(顺序组合P1 P2)--(a)--> P2’ <== P2--(a)--> P2’ /\P1-/-(a)-->/\P1 -/-(b)-->/\ P1 -/-(c)-->.

...其余规则（与上述规则相同，只是针对其他标签）

也就是说，语法声明进程和标签，而一系列推理规则定义带标签的迁移。直观地说，<== 表示“前提是”，该符号之后的公式就是规则的前提。诸如“P2 -/-(a)—>”这样的公式表示 P_2 不会执行 a-迁移。与其他 结构操作语义 的文本表示相比，这种语法并无新意，实际上它受到了 Ott 工具 [38]所采用语法的启发。

关于示例语言的一些说明：GSOS 格式只适用于有限标签集 [12]。为求简便，我们选择了动作集 {a,b,c}。此外，GSOS 中算子的统一设置是将函数符号应用于进程。这通常按上文所示的方式实现：为每个动作设置一个前缀算子。

什么语言变换？以下是 LANG-N-CHANGE 语法中与本文[1]相关的子集。

表达式 e ::= x | str | t | [e. ..e] | 表头 e | 表尾e | e@e | e − e | 映射(e, e) | e(e)| 规则 | 前提 | 结论 | 自身| e[p] : e | uniquefy(e) ⇒ (x, x) : e | getVars(e)| 如果 b 那么e 否则 e | 跳过 | 错误 str 布尔表达式。 b::= e = e | isVar(e) | b且 b | b或 b | 非b 模式 p ::= x | [p . . . p] | predname p | opname p | x p

我们假设存在一个由运算符名称组成的集合 OPNAME，其元素由 opname 表示。例如，OPNAME 包含 par 和 sequence 等元素。我们还假设存在一个由谓词名称组成的集合 PREDNAME，其元素由 predname 表示。PREDNAME 包含 $\longrightarrow$ 和 $\not\longrightarrow$ 等元素。LANG-N-CHANGE 以统一的抽象语法（predname $\text{arg}_1 \cdots \text{arg}_n$）接收公式，因为它不对语言作任何假设。然而，为方便用户，该工具仍会读取诸如 $P \longrightarrow^a P'$ 这样的转移，其语法为 P –(a)–> P'（参见上面的示例语言）。

表达式是主要的句法类别。给定一个结构操作语义规范，即语言 $\mathscr{L}$，以及一个表达式 e，e 包含将应用于 $\mathscr{L}$ 的操作。表达式可以是变量、字符串（str）、项（t）（例如 (par P1 P2) 和 (sequence P1 P2)），也可以是列表；列表支持用于提取表头和表尾、追加两个列表（e @ e）以及执行列表差（e − e）的常规操作。表达式也可以是映射 map(e_1,e_2)，其中 e_1 和 e_2 是列表。e_1 的第一个元素是 e_2, 的第一个元素的键，其余元素也依此类推[2]。给定一个映射 m，m(k) 会检索 m 中与键 k 关联的值。

LANG-N-CHANGE 包含特殊关键字 rules、premises、conclusion 和 self。关键字 rules 会返回作为输入给定的语言的所有推理规则组成的列表。我们将在以下运算符的上下文中介绍其他关键字。

选择器运算符 e_1[p]: e_2 会逐个选择满足模式 p 的列表元素，并对每个元素执行主体 e_2。选择器会返回一个列表，其中包含每次评估 e_2所产生的值。在 e_2, 中使用时，关键字 self 会返回该次迭代中选中的列表元素。模式 p 可以是一个变量；也可以尝试匹配一个列表（模式 [p . . . p]）、匹配使用特定谓词名称的公式（模式 predname p）、匹配具有特定顶层运算符的项（模式 opname p），或者尝试匹配顶层名称未指定的公式或项（模式 x p）。与模式匹配通常的做法一样，模式 p 中使用的变量会在 e_2, 中绑定，并在运行时实例化。举例来说，考虑上文示例语言（包含前缀、并行组合和顺序组合）中的规则[P $\longrightarrow^L$ P′]: e_2。rules 会评估为一个包含该语言所有规则的列表。当列表中包含规则时，选择器的模式会应用于匹配这些规则的结论。因此，模式 P $\longrightarrow^L$ P′ 会选出所有归约规则。e_2 的第一次迭代会以 P =(a P)、L =(a) 和 P′ =P 执行，依此类推。（注意，模式变量与规则的元变量不会发生冲突，因为它们在 LANG-N-CHANGE 中彼此独立。）为方便程序员，只需编写 rules[$\longrightarrow$]: e，即可选出定义 $\longrightarrow$ 的规则，而无需指定完整模式。此外，e[p] 是 e[p] : self 的简写，也就是说，它会返回一个由模式选中的元素构成的列表。因此，rules[$\longrightarrow$] 会选出所有归约规则。

[1]我们请读者参阅 [30] 和 [32] 以了解 LANG-N-CHANGE 的语法。[2]该模式的动机见于 [30]。例如，它可以根据子类型规则的结论（如结论$T_1 \to T_2 <: T'_1 \to T'_2$和 $T_1 \times T_2 <: T'_1 \times T'_2$），快速将 T_i映射为 T'_i。

当选择器运算符作用于规则时，可以使用 premises 关键字，它会返回所选规则的前提列表。例如，rules[⟶]: premises 会返回一个列表，其中每个元素都是某条规则的前提列表，例如我们这个进程代数示例中的 [[], [P1 —(a)—> P1'], [P2 —(a)—> P2'], ...]。类似地，conclusion 关键字会返回所选规则的结论。

uniquefy(e_1) ⇒ (x, y) : e_2 接收一个公式列表 e_1，并返回这些公式的一个版本，其中同一元变量的多个出现都被赋予不同的元变量。随后，计算会继续执行 e_2。新公式列表作为 x 传递给 e_2。uniquefy 还会计算一个映射，用于概述原始公式列表 e_1所发生的更改。该映射作为 y 传递给 e_2。此操作适用于将 (T-APP) 转换为 (T-APP') 等变换，正如我们在第1节中所述。假设 l 包含 (T-APP) 的前提列表，那么 uniquefy(l) ⇒(newPremises, mapOfChanges) : e 将执行 e，其中 newPremises 是列表 $[\Gamma \vdash e_1 : T_{11} \rightarrow T2, \Gamma \vdash e_2 : T_{12}]$，而 mapOfChanges 是映射 $\{T_1 \mapsto [T_{11}, T_{12}]\}$，表示 T_1 的各个出现已被拆分为 T_{11} 和 T_{12}[3]。

getVars(e) 返回 e 求值后其中使用的元变量列表。我们还有一个if语句，以及一个不执行任何操作的 skip操作。当 if 没有 else 分支时，如 if b then e，这表示if b thene else skip。错误error 会抛出运行时错误，并携带一个字符串作为错误消息。if语句的布尔条件可以检查语法相等性，检查e 是否为元变量（使用 isVar(e)），并且可以使用布尔运算组合这些检查。

我们在此不讨论LANG-N-CHANGE 的类型检查器，该检查器已在 [30]中介绍，并且例如当 e 不是列表时可以拒绝 e[p]: e′，以及其他类型错误。

3 AGSOS 验证器

GSOS规则格式 我们回顾一下 GSOS 格式 [12]。以下是 GSOS规则的形式：

$$\frac{\{x_i \longrightarrow^{l_{ij}} y_{ij} \mid i \in I, 1 \le j \le m_i\} \cup \{x_j \not\longrightarrow^{l'_{jk}} \mid j \in J, 1 \le k \le n_j\}}{(op\ x_1 \ldots x_h) \longrightarrow^{l} t}$$

注意，xs 和 ys 是表示元变量的元变量，因此可以在不同元变量之间陈述某种关系。换句话说，xs 和 ys 都表示诸如 P、P_1, P_2, 等元变量。我们有 x_i 和 y_i 彼此不同。I 和 J 是 {1, ..., h} 的子集，也就是说，前提中的 xs 来自结论，并且每个都可以多次作为正前提的主体，也可以多次作为负前提的主体。t 中出现的元变量只能来自 xs 和 ys。最后，ls 是常量标签。

符合这些限制的规则称为 GSOS 规则。例如，上一节中的进程代数（包含前缀、并行组合和顺序组合）的所有规则都是 GSOS规则。

考虑下面这条定义复制算子行为的规则。

$$\frac{(P \mid !P) \longrightarrow^{a} P'}{!P \longrightarrow^{a} P'}$$

这条规则不是 GSOS 规则，因为前提的源是 (P |!P)，而不是一个变量。

以下是 SOS 元理论的一个经典结果：如果该语言的所有规则都是 GSOS规则，那么互模拟等价对于语言 [12] 是同余。

[3]唯一 fy 可以采用更精细的风格，如 [30], 所示，但本文不需要这种风格。

3.1 LANG-N-CHANGE 的新宏

我们在 LANG-N-CHANGE 中定义以下宏。

- e 必须匹配 $p_1 \mid p_2 \mid \ldots \mid p_n$，否则执行 e′；如果 not((e $-$ e[p_1] $-$ e[p_2] $\ldots$ $-$ e[p_n]) $=$ []) 不成立，则执行 e′。其中，e 是一个列表。其思想是，e 的每个元素都必须匹配模式 p1、p2、.....、p_n 中的一个，否则执行 e′。为此，我们逐步从 e 中减去由这些模式筛选出的子列表，并检查所得列表是否为空。这个宏可用于检查前提和结论是否具有正确的形式。我们使用同样的“空列表”测试，通过以下方式检查一个列表是否为另一个列表的子列表：e sublistOf e′，(e $-$ e′) $=$ []。这个宏可用于检查规则某一部分中所使用的元变量是否全部来自正确的元变量列表。

- 使用 p 匹配 e $\rightarrow$ e′ 否则 e″，如果 [e][p] $=$ [] 则 e″ 否则 e′。
  这里，我们检查 e 是否匹配模式 p；如果匹配，则执行 e′，否则执行 e″。为此，我们创建仅包含一个元素 [e] 的列表，并使用选择器按照模式 p 对其进行筛选。如果结果列表为空，则模式 p 未能匹配 e[4]。在此宏中省略“$\rightarrow$ e′”时，表示“$\rightarrow$ skip”。省略“otherwise e″”时，表示“otherwise skip”。与上文针对列表的情况相反，此宏适用于检查单个元素是否匹配某个模式，并指定 then 与 otherwise 两种反应。

- 以下宏 可用于快速访问迁移公式的源和目标：

  前提.LT源 $\triangleq$ (前提[P –-L–-> P′]:P) @ (前提[P –/-L–->]:P) 前提.LT目标 $\triangleq$ 前提[P –-L–-> P′]:P′结论.LT源 $\triangleq$ 表头 ([结论][P –-L–-> P′]:P) 结论.LT目标 $\triangleq$ 表头 ([结论][P –-L–-> P′]:P′)

  请注意，premises.LTsources 会提取正、负标记迁移公式的源。LTsources 中的“LT”代表标记迁移。我们还为（无标记）迁移 P $\longrightarrow$P 引入了类似的宏，例如 Tsources、Ttargets 等。我们认为，这些并非语言设计中的临时宏。标记迁移和（无标记）迁移十分常见，因此提供一些操作是合理的，例如“将此前提作为标记迁移公式处理，并返回其源”。给定其他形式的公式时，premises.LTsources 和 premises.LTtargets 会返回空列表，而 conclusion.LTsource 和 conclusion.LTtarget 在运行时会因表头而失败。

- 如果 distinctVars(e)，否则为 e′，
  uniquefy([(pname e)]) $\Rightarrow$ (new, m)：如果非(m $=$ 映射([], []))，则 e′
  这里，e 是一个元变量列表。我们创建公式 (pname e)，其中 pname 是一个未使用的谓词名称，这样就能将它传递给 uniquefy。如果 m 是空映射 map([], [])，那么 uniquefy 就没有检测到任何元变量被使用多次，即 e 中的所有元变量都是不同的。distinctVars 会执行 e′ otherwise。

3.2 LANG-N-CHANGE 中的 GSOS 验证器

现在我们使用 LANG-N-CHANGE 编写 GSOS 验证器。我们将任务分为 5 个部分。这 5 项检查应按其出现的顺序执行，即先执行第 1 部分，再执行第 2 部分，依此类推。

[4] [31] 对这个宏的简化版本采用了此方法。

如果相应的检查成功，它们都会返回 skip；否则就会抛出运行时错误。得益于 LANG-N-CHANGE 的操作和我们定义的宏，这些检查很容易理解，因此有些检查无需额外注释。下文中，我们用数学字体表示 LANG-N-CHANGE 的模式变量。

- 第1部分: 所有前提都是正向或负向迁移公式，并且使用常量标签 .

  规则 [-->]: 前提必须匹配 P --(op []-->P′ | P -/-(op []--> 否则返回错误消息

  其中，msg =“前提必须是正标记迁移或负标记迁移，且其标签必须是常量”。常量是一个顶层运算符的参数为空列表的项。

- 第2 部分：所有结论都是使用常量标签的迁移公式，并且定义在一个以元变量作为参数应用的算子上。（第 4 部分稍后将检查这些元变量彼此不同，因为它们还需要与 ys 不同。）

  规则 [-->]：将结论与 (op_1 Ps)--(op_2 []--> P′ ->Ps [P] 匹配：如果不是 (isVar(P))，则报错 msg_1 否则报错 msg_2

  其中 msg_1 = “作为结论主语的算子必须以所有元变量为参数”，以及 msg_2 = “结论公式必须是标签为常量且应用于算子的正向标记迁移”。

- 第三部分：所以 前提的来源必须来自结论中的 xs，而 ys 必须是元变量 .

  规则 [-->]:
  如果 not(premises.LTsources sublistOf getVars(conclusion.LTsource))，则 error msg_1，否则 premises.LTtargets [P]：如果 not(isVar(P))，则 error msg_2

  其中，msg_1 = “前提的源必须是结论的源中运算符的参数”，而 msg_2 = “前提的目标必须是元变量”。这里，premises.LTsources、conclusion.LTsource 和 premises.LTtargets 是在第 1 部分和第 2 部分确认存在标记迁移公式后使用的。

- 第 4 部分：结论的源中的 xs 与前提中的 ys 必须各不相同。

  规则 [-->]: distinctVars (getVars(conclusion.LTsource) @ premises.LTtargets) 否则 errormsg

  其中 msg =“结论源中算子的参数和前提的目标必须都是互不相同的元变量”。

- 第 5 部分：结论目标中的元变量来自 xs 和 ys。

  规则 [-->]：如果 not(getVars(conclusion.LTtarget) sublistOf (getVars(conclusion.LTsource) @ premises.LTtargets))，则返回错误消息

其中，msg ＝“结论目标中的元变量必须来自结论的源或前提的目标”。

4 评估

我们实现了本文中所描述的宏 [14][5]。更准确地说，我们没有更改 LANG-N-CHANGE 的核心语言，而是为了方便程序员，将这些构造子添加到了表层语言中。这些构造子在解析时会被直接消去。

我们为 GSOS 验证器创建了一组测试用例。我们定义了一种只包含前缀算子 l.P 的基础语言定义。该语言作为基础，我们在此基础上添加其他特性。以此为起点，我们为以下每个并发算子分别创建了一种语言：CCS 的交错并行算子 [29] （不包含进程通信）、带通信的 CCS 完全并行算子 [29], 、CSP 的同步并行组合 [27], 、CCS 的外部选择、CSP 的内部选择、ACP 的投影 [10], 、CSP 的隐藏、左合并算子、CCS 的重命名算子、CCS 的限制算子、来自 [2], 信号传递 [8], 的“沙漏”算子、扰乱算子、中断算子、顺序算子 ;、优先级算子，以及 while 循环算子。已知这些算子均满足 GSOS 限制。

我们的代码仓库包含 18 个并发算子，我们确认我们的 GSOS 验证器成功对所有这些语言执行了第 3.2 节（第 1–5 部分）中的检查。也就是说，我们的工具验证了上述算子均遵循 GSOS 格式。我们还进行了一系列负面测试。具体来说，我们创建了不符合第 3 节所述 GSOS 限制的语言。我们确认我们的 GSOS 验证器在这些情况下会失败，并提供相应的错误消息。

尽管为了便于阅读，第 3.2 节的大多数检查都分多行展示，但实际上每项检查都可以写成一行。总的来说，我们只需用 6 行 LANG-N-CHANGE 代码就能编写一个 GSOS 验证器。这是一个极为简洁的验证器。此外，我们认为，我们的代码以声明式方式表达了 GSOS 格式的语法限制。我们的工具网站报告了所有测试结果 [14]。

5 相关工作

我们尚未发现有专门用于表达规则格式而设计的领域特定语言。

能够验证规则格式的工具并不多。PREG 公理化工具 [3] 包含一个 GSOS 格式检查器，约有四百行 Maude 代码[6]。除了格式检查之外，这部分还实现了从语言中检索信息的方法。例如，它实现了用于检索规则、搜索前提以及获取公式中所用变量的函数，仅举几例。事实证明，大多数语言工具 [15, 17, 24, 37, 39] 都需要使用这些功能。看来，每种工具都使用一种特定的编程语言，将语言存储为该编程语言的一种数据类型，并重新实现这些检索函数。LANG-N-CHANGE 可以通过提供一种用于简洁、声明式地表达这些功能的领域特定语言来缓解这一问题。若能将 LANG-N-CHANGE 嵌入编程语言中，让实现者调用其中的基本操作，将会很有意义。

[5]LANG-N-CHANGE 的旗舰实现是 [32] ，但它采用的语法比 [30] 中最初提出的语法更为冗长。因此，我们实现了一个轻量级的 LANG-N-CHANGE 领域特定语言求值器。LANG-N-CHANGE 的旗舰实现仍然是 [32]。[6]我们感谢 Eugen-Ioan Goriac 通过电子邮件在私人通信中慷慨提供了这一估算，也感谢他澄清了该 GSOS 检查器最初是 PREG 公理化工具的一部分，而不是 Meta SOS。请注意，如果不计算解析代码，该检查器估计约有 150 行 Maude 代码。

Meta SOS [7] 实现了 GSOS 格式以外的其他规则格式，因此无法直接比较。Mousavi 和 Reniers 的工具 [33] 在 Maude 中提供了 GSOS 验证器，其实现方法与 [3]不同。进程代数以 Maude 重写规则的形式提供。随后，该工具利用 Maude 的内省式反射功能来探查规则、前提等的结构。语言设计者当然可以采用这种方法来表达他们接下来的规则格式，但这要求熟悉 Maude、其反射库，以及一种非常特殊的元编程风格，而这种风格的学习曲线可能相当陡峭。一些从业者可能会觉得 LANG-N-CHANGE 的语言特性更加直观、易于使用。

6 结论

规则格式可以快速确立进程代数的元理论性质。因此，我们希望找到能够轻松表达规则格式并自动对其进行测试的领域特定语言。本文发现，LANG-N-CHANGE 提供了便捷的操作来检查操作语义。我们在 LANG-N-CHANGE 之上创建了宏，以便更好地表达规则格式中常见的一些检查。随后，我们使用 LANG-N-CHANGE 和这些宏实现了 GSOS 规则格式。总之，我们仅用 6 行代码就编写了一个完整的 GSOS 验证器，并用它验证了若干并发算子。此外，我们的代码以声明式方式表达了 GSOS 限制。

未来，我们希望将这种方法应用于其他规则格式 [34]。若干格式，包括 tyft、ntyft、path 和 panth [25, 26, 40], , 会检查变量是否互不相同、规定迁移公式的特定形式，并提取源和目标以供分析。我们相信，LANG-N-CHANGE 和我们的宏在这些场景中会很有用。不过，我们的方法也可能存在一些挑战。例如，LANG-N-CHANGE 可能尚未实现某些操作，而在尝试描述其他规则格式时，我们或许会发现需要这些操作。我们在使用 LANG-N-CHANGE 自动为某些纯函数式语言添加引用时，就遇到了这种情况 [32]。这项工作需要将归约规则的形式从 $e \longrightarrow e$ 提升为 $e;\mu \longrightarrow e;\mu$，其中 μ 是堆。在这种情况下，我们扩展了 LANG-N-CHANGE，加入了一个名为 lift 的算子，用于指定关系形式的变化。也就是说，我们添加了 LANG-N-CHANGE 原本缺少的新算子。（结果发现，这个算子对自动化渐进式类型也很有用，相关内容见 [31], 。）另一个挑战是 LANG-N-CHANGE 存在一些局限。例如，GSOS 规则格式的一个特点是，其检查始终是“局部的”，仅限于所选规则本身。而有些规则格式则需要对相关进程代数保持更全局的视角。一个例子是用于确立算子交换律（在互模拟等价意义下）的规则格式 [35], , 它会同时比较多条规则。LANG-N-CHANGE 似乎最适合一次选择一条规则。如果其他语言设计更合适，我们将探索开发其他领域特定语言。

我们的工具已公开提供，所有测试均记录在其 GitHub 代码仓库 [14] 中。

引用

[1] Luca Aceto、Arnar Birgisson、Anna Ingolfsdottir、MohammadReza Mousavi 和 Michel A. Reniers（2009）：确定性与幂等性的规则格式。载于：第三届 IPM 国际软件工程基础会议论文集（FSEN’09），施普林格出版社，柏林、海德堡，第 146–161 页，doi:10.1007/978-3-642-11623-0 8。_

[2] Luca Aceto、Bard Bloom & Frits Vaandrager（1994）：将 SOS 规则转化为方程。Information and Computation111(1)，第 1–52 页，doi:10.1006/inco.1994.1040。[3] Luca Aceto、Georgiana Caltais、Eugen-Ioan Goriac & Anna Ingolfsdottir（2011）：PREG 公理化工具——带谓词的 GSOS 闭项互模拟检查器。载于 Andrea Corradini、Bartek Klin & Corina Cîrstea 编：计算机科学中的代数与余代数国际会议（CALCO 2011），Springer Berlin Heidelberg，柏林，海德堡，第 378–385 页，doi:10.1007/978-3-642-22944-2_27。[4] Luca Aceto、Matteo Cimini、Anna Ingólfsdóttir、Mohammad Reza Mousavi & Michel A. Reniers（2011）：零元和单位元的 SOS 规则格式。Theoretical Computer Science 412(28)，第 3045–3071 页，doi:10.1016/j.tcs.2011.01.024。[5] Luca Aceto、Matteo Cimini、Anna Ingolfsdottir、MohammadReza Mousavi & Michel A. Reniers（2012）：分配律规则格式。TheoreticalComputerScience458，第 1–28 页，doi:10.1016/j.tcs.2012.07.036。[6]Luca Aceto、Ignacio Fábregas、Álvaro García-Pérez、Anna Ingólfsdóttir & Yolanda Ortega-Mallén（2019）：名义进程演算的规则格式。Logical Methods in Computer Science 15(4)，doi:10.23638/LMCS-15(4:2)2019。[7] Luca Aceto、Eugen-Ioan Goriac & Anna Ingólfsdóttir （ 2013 ）：Meta SOS—— 基于 Maude 的 SOS 元理论框架。载于 Johannes Borgström & Bas Luttik 编：并发表达能力第 20 届联合国际研讨会与结构操作语义第 10 届研讨会会议论文集，EXPRESS/SOS 2013，阿根廷布宜诺斯艾利斯，2013 年 8 月 26 日，EPTCS 120，第 93–107 页，doi:10.4204/EPTCS.120.8。[8] J. C. M. Baeten & J. A. Bergstra（1992）：带信号和条件的进程代数。载于 Manfred Broy 编：程序设计与数学方法，Springer Berlin Heidelberg，柏林，海德堡，第 273–323 页，doi:10.1007/978-3-642-77572-7_13。[9] Falk Bartels（2002）：概率转移系统的 GSOS：（扩展摘要）。Electronic Notes in TheoreticalComputerScience65(1)，第 29–53 页，doi:10.1016/S1571-0661(04)80358-X。CMCS'2002，计算机科学中的余代数方法（ETAPS 2002 卫星会议）。[10] J.A. Bergstra & J.W. Klop（1984）：同步通信进程代数。InformationandControl 60(1)，第 109–137 页，doi:10.1016/S0019-9958(84)80025-X。[11] Bard Bloom（1995）：弱双模拟的结构操作语义。TheoreticalComputerScience 146(1)，第 25–68 页，doi:10.1016/0304-3975(94)00152-9。[12] Bard Bloom、Sorin Istrail & Albert R. Meyer（1995）：互模拟不可追踪。Journal of the ACM 42(1)，第 232–268 页，doi:10.1145/200836.200876。[13] Roland Bol & Jan Friso Groote（1996）：转换系统规范中负前提的含义。JournaloftheACM43(5)，第 863–914 页，doi:10.1145/234752.234756。[14] Matteo Cimini（2022）：GSOS 验证器。https://github.com/mcimini/gsos-validator。[15] Matteo Cimini、Dale Miller & Jeremy G. Siek（2020）：函数式语言的外置类型操作语义。载于：第 13 届 ACM SIGPLAN 软件语言工程国际会议会议论文集，SLE 2020，美国线上会议，2020 年 11 月 16–17 日，第 108–125 页，doi:10.1145/3426425.3426936。[16] Matteo Cimini & Benjamin Mourad（2021）：课堂中的语言变换。载于 Ornela Dardha & Valentina Castiglioni 编：并发表达能力第 28 届联合国际研讨会与结构操作语义第 18 届研讨会会议论文集，EXPRESS/SOS 2021，以及结构操作语义第 18 届研讨会，法国巴黎（线上会议），2021 年 8 月 23 日，EPTCS339，第 43–58 页，doi:10.4204/EPTCS.339.6。[17] Matteo Cimini & Jeremy G. Siek（2016）：渐进化器：生成渐进类型系统的方法论与算法。载于：第 43 届 ACM SIGPLAN-SIGACT 编程语言原理年度研讨会会议论文集，POPL '16，计算机协会，纽约，纽约州，美国，第 443–455 页，doi:10.1145/2837614.2837632。

[18] Sjoerd Cranen、MohammadReza Mousavi 和 Michel A. Reniers（2008）：结合律规则格式。载于 Franck van Breugel 和 Marsha Chechik（编）：CONCUR 2008—并发理论，Springer Berlin Heidelberg，柏林，海德堡，第447–461页，doi:10.1007/978-3-540-85361-9_35.[19] Pedro R. D’Argenio、Matias David Lee 和 Daniel Gebler（2015）：凸概率双模拟和抽象概率双模拟的 SOS 规则格式。载于 Silvia Crafa 和 Daniel Gebler（编）：并发表达能力第22届国际研讨会与第12届结构操作语义研讨会联合会议论文集，EXPRESS/SOS 2015，马德里，西班牙，2015年8月31日，EPTCS 190，第31–45页，doi:10.4204/EPTCS.190.3.[20] Marcelo Fiore 和 Sam Staton（2009）：名称传递进程演算的同余规则格式。信息与计算207(2)，第209–236页，doi:10.1016/j.ic.2007.12.005.[21] Wan J. Fokkink（2000）：作为同余的根分支双模拟。计算机与系统科学杂志60(1)，第13–37页，doi:10.1006/jcss.1999.1663.[22] Wan J. Fokkink 和 Thuy Duong Vu（2003）：结构操作语义与有界非确定性。Acta Informatica 39，第501–516页，doi:10.1007/s00236-003-0111-1.[23] Rob J. van Glabbeek（2005）：弱双模拟的 Cool 同余格式。载于 Dang Van Hung 和 Martin Wirsing（编）：计算理论方面——ICTAC 2005，Springer Berlin Heidelberg，柏林，海德堡，第318–333页，doi:10.1007/11560647_21.[24] Sylvia Grewe、Sebastian Erdweg、Pascal Wittmann 和 Mira Mezini（2015）：面向大众的类型系统：推导可靠性证明和高效检查器。载于：2015年ACM编程与软件新思想、新范式和反思国际研讨会（Onward!），Onward! 2015，ACM，纽约，美国纽约州，第137–150页，doi:10.1145/2814228.2814239.[25] Jan Friso Groote（1993）：带负前提的转移系统规范。Theoretical Computer Science 118(2)，第263–299页，doi:10.1016/0304-3975(93)90111-6.[26] Jan Friso Groote 和 Frits Vaandrager（1992）：结构操作语义与作为同余的双模拟。信息与计算100(2)，第202–260页，doi:10.1016/0890-5401(92)90013-6.[27] C. A. R. Hoare（1978）：通信顺序进程。ACM 通讯21(8)，第666–677页，doi:10.1145/359576.359585.[28] Ruggero Lanotte 和 Simone Tini（2009）：作为同余的概率双模拟。ACM 计算逻辑汇刊10(2)，第9:1–9:48页，doi:10.1145/1462179.1462181.[29] Robin Milner（1980）：通信系统演算。计算机科学讲义92，Springer Berlin Heidelberg，柏林，海德堡，doi:10.1007/3-540-10235-3.[30] Benjamin Mourad 和 Matteo Cimini（2020）：语言转换演算。载于：第46届信息学理论与实践当前趋势国际会议（SOFSEM 2020），施普林格，第547–555页，doi:10.1007/978-3-030-38919-2_44.[31] Benjamin Mourad 和 Matteo Cimini（2020）：基于语言转换的声明式渐进式处理器。载于 Olaf Chitil（编）：IFL 2020：第32届函数式语言实现与应用研讨会，线上会议／坎特伯雷，英国，2020年9月2日至4日，ACM，第44–54页，doi:10.1145/3462172.3462190.[32] Benjamin Mourad 和 Matteo Cimini（2020）：系统描述：Lang-n-Change——语言转换工具。载于 Keisuke Nakano 和 Konstantinos Sagonas（编）：函数式与逻辑编程——第15届国际研讨会，FLOPS 2020，日本秋田，2020年9月14日至16日，会议论文集，计算机科学讲义12073，施普林格，第198–214页，doi:10.1007/978-3-030-59025-3_12.[33] Mohammad Reza Mousavi 和 Michel A. Reniers（2006）：在 Maude 中为 SOS 元理论构建原型。Theoretical Computer Science 电子札记156(1)，第135–150页，doi:10.1016/j.entcs.2005.09.030。第二届结构操作语义研讨会（SOS 2005）会议论文集。[34] Mohammad Reza Mousavi、Michel A. Reniers 和 Jan F. Groote（2007）：SOS 格式与元理论：20年后。Theoretical Computer Science 373(3)，第238–272页，doi:10.1016/j.tcs.2006.12.019。

[35] MohammadReza Mousavi、Michel Reniers 与 Jan Friso Groote（2005）：SOS 的语法交换格式。《Information Processing Letters》93(5)，第 217–223 页，doi:10.1016/j.ipl.2004.11.007。[36] Gordon D. Plotkin（2004）：操作语义的结构化方法。《Journal of Logic and Algebraic Programming》60-61，第 17–139 页，doi:10.1016/j.jlap.2004.05.001。[37] Michael Roberson、Melanie Harries、Paul T. Darga 与 Chandrasekhar Boyapati（2008）：类型系统可靠性的高效软件模型检测。载于 Gail E. Harris 编：《面向对象编程系统、语言与应用第23届 ACM SIGPLAN 会议论文集》，OOPSLA '08，计算机协会，纽约，美国纽约州，第 493–504 页，doi:10.1145/1449764.1449803。[38] Peter Sewell、Francesco Zappa Nardelli、Scott Owens、Gilles Peskine、Thomas Ridge、Susmit Sarkar 与 Rok Strniša（2010）：Ott：为语义学研究者提供有效的工具支持。《Journal of Functional Programming》20(1)，第 71–122 页，doi:10.1017/S0956796809990293。[39] Andrei Stef˘anescu、Daejun Park、Shijiao Yuwen、Yilong Li 与 Grigore Ro¸su（2016）：适用于所有语言的基于语义的程序验证器。载于：2016 年 ACM SIGPLAN 面向对象编程、系统、语言与应用国际会议论文集，OOPSLA 2016，SPLASH 2016 的一部分，阿姆斯特丹，荷兰，2016年10月30日至11月4日，第 74–91 页，doi:10.1145/2983990.2984027。[40] C. Verhoef（1994）：带谓词和负前提的结构操作语义的同余定理。载于 Bengt Jonsson 与 Joachim Parrow 编：《CONCUR'94：并发理论》，Springer Berlin Heidelberg，柏林，海德堡，第 433–448 页，doi:10.1007/978-3-540-48654-1_32.[41] Axelle Ziegler、Dale Miller 与 Catuscia Palamidessi（2006）：名称传递演算的同余格式。《Electronic Notes in Theoretical Computer Science》156(1)，第 169–189 页，doi:10.1016/j.entcs.2005.09.032。第二届结构操作语义研讨会（SOS 2005）会议论文集。