申请上海交通大学硕士学位论文

进程重写系统的有限性问题研究

论文作者 杨　非

学　　号 1110339027

指导教师 傅育熙教授

专　　业 计算机科学与技术

答辩日期 2014 年 1 月 16 日

Submitted in total fulfilment of the requirements for the degree of Master in Computer Science

Regularity Problems of Process Rewrite Systems

Fei Yang

Supervisor

Prof. Yuxi Fu

Depart of Computer Science And Engineering

Shanghai Jiao Tong University

Shanghai, P.R.China

Jan. 16th, 2014

上海交通大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：_____

日　期：_____年 ____月 ____日

上海交通大学
学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保　密 □，在 _____ 年解密后适用本授权书。

本学位论文属于

不保密 □。

(请在以上方框内打“✓”)

学位论文作者签名：_____　　指导教师签名：_____

日　期：_____年 ___月 ___日　　日　期：_____年 ___月 ___日

进程重写系统的有限性问题研究

摘　　要

关键词： 上海交大　饮水思源　爱国荣校

Regularity Problems of Process Rewrite Systems

ABSTRACT

KEY WORDS: SJTU, master thesis, XeTeX/LaTeX template

目　　录

插图索引

表格索引

主要符号对照表

ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率

第一章 绪论

1.1 研究背景

在计算机科学理论的发展过程中，许多不同的计算模型被相继提出。最初的计算模型所定义的都是串行计算。例如图灵机 (Turing Machine)[1], λ-演算 (λ-Calculus)[2], 递归函数 (Recursive Function)[3] 等。随着并行化计算的发展，理论计算机科学家提出了并行化的计算模型并进行了深入的研究。其中比较有代表性的是由 R.Milner 提出的 Communication Concurrency System (CCS)[4]。该模型利用进程演算的方法对可以实现并发和交互的进程模型进行了刻画。

在这些计算模型的研究中，涉及到一个计算机科学中十分重要的领域：形式化验证 (Formal Verification)。而这些计算模型，大多数都可以描述无限状态系统。对无限状态系统的形式化验证 (Verification on Infinite State Systems)，是现今理论计算机科学中的一个热门的研究方向，它包含了一系列具有重要意义的研究课题。这些问题往往可以与可计算理论 (Computability), 计算复杂性理论 (Computational Complexity)，算法 (Algorithm) 等领域中的一些经典问题相联系起来，从而得出许多振奋人心的可计算性或复杂性结论。

为了对这些模型进行验证，我们需要选择合适的等价关系 (Equivalence Relation)。人们最初研究的等价关系是语言等价 (Language Equivalence)，然而即使对于上下文无关语言 (Context Free Language)，其语言等价也是不可判定的 [5]。而不可判定的等价关系从验证角度来说是用处不大的。于是许多基于观测理论 (Observation Theory) 中互模拟 (Bisimulation) 概念的等价关系被提出，最早的是由 Park 提出的强互模拟 (Strong Bisimulation)[6]。随后，为了区别系统中内部动作和外部动作，Milner 引入了 τ 动作表示系统内部的转换，并且定义了弱互模拟 (Weak Bisimulation)[4]。为了更加精细地区分 τ 动作对系统状态的影响，van Glabbeek 和 Weijland 提出了 Branching 互模拟 (Branching Bisimilarity)[7]。这些基于互模拟的等价关系，是研究验证问题的理论基础。

在研究无限状态系统时，系统的模型可以使用进程 (Process) 进行表示。对于各种不同种类的无限状态系统，都可以利用一个统一的进程代数模型进行表示，即进程重写系统 (Process Rewrite System, 简称 PRS)[8]。PRS 是一个具有一

般性的进程模型，它提供不同模型的关于强互模拟的表达能力的层次结构。许多常见的进程代数模型都可以在这个层次结构中找到。

而对于一个具体的模型所描述的系统，确定需要研究的等价关系后，我们所关心的问题主要分 3 类，分别是关于该等价关系的等价性判定 (Equivalence Checking), 与某个给定的有限状态系统的等价性判定 (Finiteness) 和是否存在一个有限状态系统与该系统等价 (Regularity)。这 3 类问题在本质上有着一定的联系，但是在解决的方法和难度上却有着区别。本文将重点对第三类问题，即有限性问题 (Regularity Problem) 进行讨论。

从直观的角度解释，Regularity 问题是给定一个可描述无限状态系统的模型是否可表示为等价的有限系统的判定问题。在某种意义上，它也是等价判定问题和模型检测问题的一个重要条件。另一个振奋人心的事实是，Regularity 性质的成立与否，决定了系统所能到达的不同状态是否是有限的，该条件如果成立，相应的判定问题通常都存在快速解决的算法。

1.2 国内外研究现状

无限状态系统的验证近年来一直是理论计算机科学中的一个十分活跃的研究领域。在这方面最早的可判定性结论是由 Baeten, Bergstra 和 Klop 证明的上下文无关语法关于强互模拟等价的可判定性 [9]。这一结论引发了对于各种不同无限状态系统的等价性判定的研究，许多问题的可判定性和复杂性结论被提出并得到了证明。我们可以找到许多相关的调查研究 [10–13]。

这些研究所涉及到的大多数模型都可以由进程重写系统所产生 [8]，这些研究都极大的提升了我们对于一些经典无限状态系统模型的认识。这个框架下包括了许多我们熟悉的模型，如基本进程代数 (Basic Process Algebra, 简称 BPA)[14]，基本并行进程 (Basic Parallel Process, 简称 BPP)[15]，进程代数 (Process Algebra，简称 PA)[16]，下推自动机 (Pushdown Automaton，简称 PDA)[5] 以及 Petri 网 (Petri Net，简称 PN)[17]。这些模型关于互模拟等价的表达能力在进程重写系统中产生了一个严格包含关系的层次结构，这种结构使得我们的研究更加具有效率。例如一个模型的复杂性上界结论可以直接隐含其子模型上的结论，而如果对于某个模型中一个问题，我们能在其某个子模型上证明它的复杂性下界，那这个结论在该模型上显然也成立。以上即为本文中的研究所涉及到的模型。

在这个无限状态系统验证的领域中，人们最初的兴趣总是集中在对 Equivalence Checking 问题的研究。经过 20 年的发展，关于强互模拟关系的判定研究已经比较成熟。最早的 BPA 上的判定性算法是通过对进程的分解技术实现的 [9, 18]，经过对该技术的改进和对模型的进一步限定，在 Normed BPA 和 Normed BPP 上，强互模拟都有了多项式时间的判定性算法 [19, 20]，在这里，Normed 是对于进程模型的一个限定条件，它规定了模型必须可以到达空进程。同时，对于一般的 BPA 和 BPP，2-EXPTIME 和 PSPACE 完全的时间复杂性也分别被证明 [21–23]。对于 PDA，强互模拟是可判定的 [24, 25]，同时对 Normed PA 强互模拟的判定也被证明是在 2-NEXPTIME 的时间复杂度内可判定的 [26]。然而，PN 的强互模拟等价性即使在 Normed 条件的限定下也是不可判定的 [27]。

但是在实际的系统中，例如在程序分析和数据库系统中，很多系统的转换只能用内部动作来描述，所以更具有实际意义的通常是区分系统内部动作的弱互模拟和 Branching 互模拟。许多模型关于带有内部动作的互模拟等价关系判定问题都被证明是不可判定的 [28]。然而，对于 Branching 互模拟的判定问题近两年得到了很大的突破。Normed BPP 和 Normed BPA 上关于 Branching 互模拟等价的可判定性分别被证明 [29, 30]。这两个结论是十分令人兴奋的，它们为 Branching 互模拟等价的相关研究开辟了新的道路。

Finiteness 的判定是一个和 Regularity 相似但是直观上更加容易的问题。因为该问题中有限状态系统是给定的，我们需要做的仅仅是判定它和给定系统是否等价。关于各种互模拟关系的 Finitness 问题通常都有多项式时间复杂度的快速解法。例如 BPA 和 Normed BPP 中关于弱互模拟和 Branching 互模拟的 finiteness 都有多项式时间算法 [31, 32]。即使没有多项式时间的算法，关于 PDA，PA 和 PN 关于强互模拟分别是 PSPACE[33]，co-NEXPTIME[34] 和可判定的 [35]。显然，通常这些问题都比对应的 Equivalence Checking 问题有着更好的计算复杂性和可判定性。而 Regularity 问题在某种意义上提供了联系这两个问题的一个桥梁：如果某个进程满足 Regularity 性质，那么我们就可以用更快速的 Finiteness 判定的算法来进行 Equivalence Checking 的工作。

在现阶段，关于强互模拟关系的 Regularity 问题通常在可判定性上有了不错的结果。在 BPA 上关于强互模拟关系的 Regularity 问题被证明是 2-EXPTIME 的 [21, 36], 而对于 BPP 则是 PSPACE 完全的 [37]。在 PDA 上，现阶段只证明了

Normed 条件下的可判定性，而且有一个多项式时间的算法 [38]。关于 PA，也仅在 Normed 条件下有一个多项式时间的算法 [39]。而对于 PN，我们有 Regularity 的可判定性, 并且在引入内部动作后该问题变为不可判定的 [35]。

在互模拟等价关系引入内部动作之后，我们现阶段已知的 Regularity 问题的可判定结论就十分有限了。现在唯一具有实际意义的结论就是由 Fu 在 2013 年证明的，关于 Branching 互模拟，在 Normed BPA 上 Regularity 问题的可判定性 [30]。

为了简化模型，有的时候研究的模型可以加上 Totally Normed 条件。在该限定下关于弱互模拟和 Branching 互模拟的问题通常更好的可判定行结论或者算法。Hüttel 最早在 [40] 中引入了该限定，并证明了 Totally Normed BPP 的 Branching 互模拟的可判定性。Chen 也在这一限定条件下证明了 Totally Normed BPA 和 Totally Normed BPP 关于弱互模拟的可判定性 [41, 42]。

1.3 主要工作

本文主要针对关于引入内部动作的互模拟等级关系的 Regularity 问题进行了讨论。主要贡献可以分为一下几个方面:

1. 归纳总结了 PRS 上 Regularity 问题的现有结论与技术，给出了一些解决 Regularity 问题所用到的技术和引理。

2. 对 Totally Normed PA 关于弱互模拟和 Branching 互模拟关系的 Regularity 问题给出了一个多项式时间算法。证明了该算法的正确性，并做了复杂度分析。该算法的时间复杂度是 $\mathcal{O}(n^3 + mn)$ 的，其中 m 和 n 都是和输入模型相关的参数。该算法对 PA 的子模型 BPA 和 BPP 也成立。

3. 对 Normed BPP 关于 Branching 互模拟关系以及 Totally Normed PN 关于弱互模拟和 Branching 互模拟关系的 Regularity 问题进行了讨论，并为今后的进一步工作提供了一定的思路。

4. 针对 Regularity 问题的下界研究提出了一些现阶段似乎可以的问题，并且直接归约出了一些有用的推论。

本文的工作讨论的问题是属于无限状态系统验证领域中的一部分工作，和该领域中很多经典的问题都有交叉。在得到了关于 Totally Normed PA 上不错结论的同时，也提出了对后续问题的解决十分具有启发意义的研究思路。

1.4 章节安排

在本文的第二章中，将具体介绍本文讨论的问题所涉及到的进程重写系统中的各种模型以及所用到的各种互模拟等价关系；第三章中，将会介绍 Regularity 问题一些已有的结论和解决 Regularity 问题所需要用到的技术和一些引理，以及直接得到的推论；第四章中，将会给出 Totally Normed PA 关于弱互模拟和 Branching 互模拟的一个等价条件；第五章中，将会给出该问题的多项式时间算法，并做计算复杂性分析；第六章中，将对后续准备解决的问题决进行一些讨论；最后，将对全文的工作进行总结。

第二章 背景知识

在这一部分，我们首先介绍一下研究 PRS 上 Regularity 问题的背景知识。包括相关的模型和一些有趣的互模拟等价关系，以及研究无限状态系统验证的几个基本问题和目标。

2.1 进程重写系统 PRS

2.1.1 基本定义

进程重写系统 (PRS) 是一个可以用来刻画进程模型语义的一般系统，这一部分我们会给出关于进程重写系统的一些基本定义。[8]

在此之前，我们可以分析一个进程代数中的例子 [4]，以下定义了一个计数器 (Counter Machine) 的需求 *(Specification)*。

$$\begin{aligned} C_0 &= zero.C_0 + inc.C_1, \\ C_{i+1} &= dec.C_i + inc.C_{i+2}, \text{ where } i \geq 0. \end{aligned}$$

下面是 Busi, Gabbrielli 和 Zavattaro 给出的实现 *(Implementation)*[43]:

$$\begin{aligned} Counter &= zero.Counter + inc.(d)(O \mid d.Counter), \\ O &= dec.\overline{d} + inc.(e)(E \mid e.O), \\ E &= dec.\overline{e} + inc.(d)(O \mid d.E). \end{aligned}$$

用 BPA 来编码 *(Programming)* 就是:

$$Z \xrightarrow{inc} XZ, \quad Z \xrightarrow{zero} Z, \quad X \xrightarrow{inc} XX, \quad X \xrightarrow{dec} \epsilon.$$

通过这个例子，我们可以直观的看出 PRS 中的 BPA 模型可以编码一个计数器。当然我们也可以通过更加复杂的编码来实现更加复杂工作的验证。我们下面给出 PRS 语法的定义和语义的规则。

定义 2.1 (进程项 Process Term)**.** 令 $Act = \{a, b, \ldots\}$ 是一个*原子动作 (Atomic Actions)* 的集合;$Const = \{\epsilon\} \cup \{X, Y, Z, \ldots\}$ 是一个*进程常量 (Process Constants)* 的集合。$S = \{\alpha_1, \alpha_2, \ldots\}$ 被称为*进程项 (Process Terms)* 的集合，它被用来刻画系统的状态，可以由一下的 BNF 产生:

$$\alpha ::= \epsilon \mid X \mid \alpha_1.\alpha_2 \mid \alpha_1 \mid \alpha_2$$

其中

- ϵ 被称为*空进程 (Empty Process)*;
- $\alpha_1.\alpha_2$ 是一个*串行 (Sequential)* 进程;
- $\alpha_1 \mid \alpha_2$ 是一个*并行 (Parallel)* 进程。

我们这里用小写希腊字母 $\alpha, \beta, \gamma, \ldots$ 来表示进程项。

有了进程项的定义，对于一个进程演算系统，就定义它的*操作语义 (Operational Semantics)*。这里，我们使用*标号迁移系统 (Labeled Transition System 简称 LTS)* 来定义 PRS 中的模型所遵循的语义规则。

定义 2.2 (标号迁移系统 LTS)**.** 一个*标号迁移系统 (LTS)* 是一个五元组 $(S, Act, \longrightarrow, \alpha_0, F)$，其中

- S 是一个*状态 (States)* 的有限集合，
- Act 是一个*标号 (Labels)* 的有限集合，
- $\longrightarrow \subseteq S \times Act \times S$ 是一个*转换关系 (Transition Relation)*，
- $\alpha_0 \in S$ 是一个给定的*初始状态 (Start State)*，
- $F \subseteq S$ 是一个*终结状态 (Final States)* 的有限集合，这意味着对于任何 $\alpha \in F$ 不存在 $a \in Act$ 和 $\beta \in S$ 使得 $\alpha \xrightarrow{a} \beta$。

我们通常将 $(\alpha, a, \beta) \in \longrightarrow$ 记做 $\alpha \xrightarrow{a} \beta$。

接下来就可以利用语义推导规则 *(Inference Rules)* 来得到 PRS 模型的操作语义。LTS 所定义的语义转换关系是由形如 $\alpha \xrightarrow{a} \beta$ 的规则的所构成的有限集合 Δ 所生成的。对于任意 $a \in Act$，语义迁移关系 $\xrightarrow{a}$ 是从以下的语义推导规则构造的最小的转换关系：

$$\frac{\alpha \xrightarrow{a} \beta \in \Delta}{\alpha \xrightarrow{a} \beta} \qquad \frac{\alpha \xrightarrow{a} \alpha'}{\alpha.\beta \xrightarrow{a} \alpha'.\beta}$$

$$\frac{\alpha \xrightarrow{a} \alpha'}{\alpha \mid \beta \xrightarrow{a} \alpha' \mid \beta} \qquad \frac{\beta \xrightarrow{a} \beta'}{\alpha \mid \beta \xrightarrow{a} \alpha \mid \beta'}$$

2.1.2 层次结构

PRS 利用对语义规则中进程项类型的分类，分成了几个子模型。这些子模型大多可以和一些常用的进程模型所对应，它们之间有着一个关于互模拟关系表达能力包含关系的层次结构。这里我们首先先根据连结符，将进程项分为四类：

1. **1**: 仅仅由单个 *(Single)* 进程常量构成的项，形如 X。
2. **S**: 单个进程常量或者串行连结 *(Sequential Composition)* 的进程常量构成的项，形如 $X.Y.Z$。
3. **P**: 单个进程常量或者并行连结 *(Parallel Composition)* 的进程常量构成的项，形如 $X \mid Y \mid Z$.
4. **G**: 由任意 *(General)* 串行或者并行连结的进程常量构成的项，如 $(X.(Y \mid Z)) \mid W$。

下面我们将给出 PRS 的严格定义：

定义 2.3 (进程重写系统 PRS)**.** 令 $\Xi, \Pi \in \{\mathbf{1}, \mathbf{S}, \mathbf{P}, \mathbf{G}\}$. 一个$(\Xi, \Pi)$-*PRS* 是一个满足如下条件的规则集合 Δ：对于每条规则 $\alpha \xrightarrow{a} \beta \in \Delta$ 有

- $\alpha \in \Xi \setminus \{\epsilon\}$,
- $\beta \in \Pi$,
- 系统的初始状态由一个进程项 $\alpha_0 \in \Xi$ 给定。

一个 $(\mathbf{G},\mathbf{G})$-PRS 即为一个一般的 PRS.

不失一般性，本文假定一个 PRS 系统的初始状态 α_0 均为一个单独常量进程项。

分类了 PRS 中各种模型之后，我们可以得到图2–1中的*层次结构 (Hierarchy)*。接下来，将介绍本文研究中所涉及到的一些子模型。

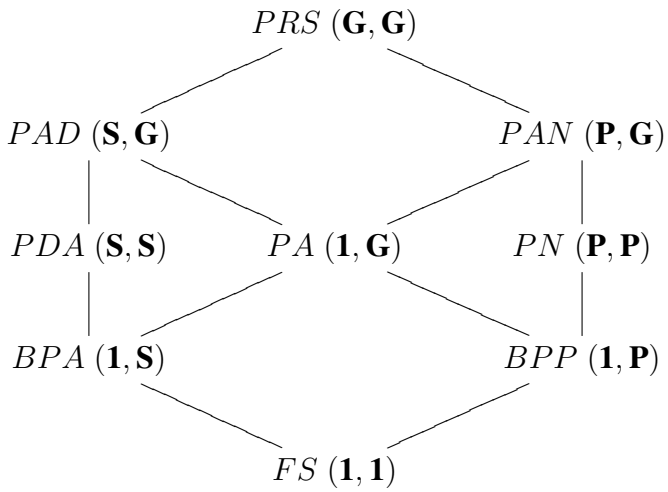

图 2–1 PRS *层次结构* PRS-Hierarchy

2.1.3 PRS 中的模型

本小节中将按照 PRS 层次结构中自地向上的顺序依次介绍几个模型。

1. $(\mathbf{1},\mathbf{1})$-PRS: *有限状态自动机 (Finite State Automaton 简称 FS)*，也称作*正则进程 (Regular Process)*，产生的语言为*正则语言 (Regular Language)*。

2. $(\mathbf{1},\mathbf{S})$-PRS: *基本进程代数 (Basic Process Algebra 简称 BPA)*，和*上下文无关语法 (Context-free Grammar)* 等价。

3. $(\mathbf{S},\mathbf{S})$-PRS: 被证明和*下推自动机 (Pushdown Automaton 简称 PDA)* 等价，BPA 是它的子模型，该模型也是程序分析领域中一个重要模型。以上两种为只允许串行连结符的进程模型。

4. $(\mathbf{1},\mathbf{P})$-PRS: 基本并行进程 *(Basic Parallel Process* 简称 *BPP)*，和可交换的上下文无关语法 *(Commutative Context-free Grammar)* 等价。这里进程项虽然是用并行连结符连结的，但是并不允许进程间的交互操作。

5. $(\mathbf{P},\mathbf{P})$-PRS: 和 *Petri* 网 *(Petri Net* 简称 *PN)* 等价，BPP 是它的子模型，该模型在验证领域有着广泛的应用。以上两种为仅允许并行连结符的进程模型。

6. $(\mathbf{1},\mathbf{G})$-PRS: 进程代数 *(Process Algebra* 简称 *PA)*，该模型同时允许两种连结符在进程项中出现，BPA 和 BPP 都是它的子模型。在本文中将着重对其 Regularity 问题进行研究，所得的可判定性结论和算法对 BPA 和 BPP 都是适用的。

7. 对于 PRS 中的其它几种模型，由于其模型复杂性比较高，且等价性通常被证明是不可判定的。所以现阶段仅仅对其可达性 *(Reachability* 进行了研究，这里由于篇幅所限，就不加以深入讨论了。

在本文中余下的部分，为了方便表示，如果不加说明，都将使用简称来代替相应的模型。

2.2 互模拟等价关系

在使用形式化方法进行无限状态系统验证问题的研究中，一个十分基本的问题就是对等价关系 *(Equivalence Relation* 的选取。等价关系是可以用来地刻画两个系统的相等性的关系。一个合适的等价关系应该关于人们对系统的要求是可靠 *(Sound)* 且完备 *(Complete)* 的，这两条性质是模型对等价关系正确性的基本要求。除此以外，等价关系的在计算上的可判定性也在其选择中扮演了重要角色。我们知道，尽管同构 *(Isomophsm)* 或者语言等价 *(Language Equivalence)* 都是十分直观的等价关系。但是同构并不能包含我们所需要的所有互相等价的进程对，即不满足完备性，另一方面，它在计算复杂性上也是 NP 完全的；而语言等价虽然是很严格的等价关系，但是它是不可判定的 [5]。

一个看似不错的选择是语义等价 *(Semantic Equivalence)*，van Glabbeek 在 [44] 对语义等价进行了详细的总结，互模拟等价 *(Bisimulation Equivalence)* 是语义等价的一种。在观测理论中，人们通常利用互模拟的概念来定义等价关系，

互模拟等价关系所关心的是从观测者的角度是否能区分进程间的不同。我们选择它进行研究，还因为互模拟等价关系的判定在计算上通常都有更好的可行性 *(Feasibility)*，同时它也找到一个十分美妙的博弈论 *(Game Theoretical)* 的刻画。这些良好的性质都为我们的理论研究和实际应用提供了方法上的启发和方向上的引导。

2.2.1 强互模拟关系

最初的互模拟等价关系是由 Park 提出的强互模拟 *(Strong Bisimilarity)* 关系 [6]。强互模拟关系是一个通过归纳法进行定义的关系，在博弈论的角度，它假设有一个防守者 *(Duplicator)* 和一个破坏者 *(Spoiler)*，是否能在有限轮的游戏中区分出两个进程的不同。

下面给出强互模拟关系的定义：

定义 2.4 (强互模拟关系 Strong Bisimilarity)**.** 一个关于标号迁移系统中状态的二元关系 $\mathcal{R}$ 是一个强互模拟关系 *(Strong Bisimulation)* 当且仅当对任意 $(\alpha, \beta) \in \mathcal{R}$，我们有：

- 如果 $\alpha \xrightarrow{a} \alpha'$ 那么存在 β'，使得 $\beta \xrightarrow{a} \beta'$，且有 $(\alpha', \beta') \in \mathcal{R}$。
- 如果 $\beta \xrightarrow{a} \beta'$ 那么存在 α'，使得 $\alpha \xrightarrow{a} \alpha'$，且有 $(\alpha', \beta') \in \mathcal{R}$。

如果对于某些强互模拟关系 $\mathcal{R}$，我们有 $(\alpha, \beta) \in \mathcal{R}$，那么称 α 和 β 是强互模拟等价 (Strong Bisimulation Equivalent) 的或强互模拟 (Strong Bisimilar) 的, 记做 $\alpha \sim \beta$。

$\sim = \bigcup\{\mathcal{R} : \mathcal{R}$ 是一个强互模拟关系$\}$，是最大的强互模拟关系 (Strong Bisimilarity)。

在小节1.2中，我们提到了强互模拟关系验证的相关问题通常有着不错的可判定性或者算法复杂性结论。在观测理论中，由于强互模拟关系的简洁性，也是人们在最初的研究中最感兴趣的等价关系。不过我们注意到，在强互模拟关系的定义中，我们并不考虑系统中的动作是否能被外界观测到。从某种意义上来说，强互模拟关系有点太“强”了，因为每一个动作都会被用来区分进程的等价性，即使这个动作是外界观测不到的甚至是不改变系统状态的。

2.2.2 考虑内部动作的互模拟关系

基于上一节定义的强互模拟关系，为了更精确的刻画进程间的等价，Milner 在等价关系的定义中引入了系统内部动作 *(Silent Actions)*，定义了*弱互模拟关系 (Weak Bisimilarity)*[4]。它忽略了系统内部动作对观测的影响，使得等价关系的对进程间的区分更 “弱” 了。而 van Glabbeek 和 Weijland 为了更精确地区分系统内部动作对系统状态的影响，定义了 *Branching 互模拟关系 (Branching Bisimilarity)*[7]。它仅仅忽略了不影响系统状态的内部动作，是一个更加精确和合理的等价关系。

为了给出这两种等价关系的形式化定义，我们做如下规定：

- 我们用符号 τ 来表示所有系统内部动作，令 $Act = \{a, b, c, \ldots\} \cup \{\tau\}$。$\ell$ 属于 Act，ℓ^* 属于 Act^*。
- 我们用 $\Longrightarrow$ 来表示 $\xrightarrow{\tau}$ 的自反传递闭包 (Reflexive Transitive Closure)。$\stackrel{\widehat{\ell}}{\Longrightarrow}$ 用来表示 $\Longrightarrow\xrightarrow{\ell}\Longrightarrow$，如果 $\ell \neq \tau$，否则表示 $\Longrightarrow$。

接下来给出弱互模拟和 Branching 互模拟的定义：

定义 2.5 (弱互模拟关系 Weak Bisimilarity)**.** 一个关于标号迁移系统中状态的二元关系 $\mathcal{R}$ 是一个*弱互模拟关系 (Weak Bisimulation)* 当且仅当对任意 $(\alpha, \beta) \in \mathcal{R}$，我们有：

- 如果 $\alpha \xrightarrow{a} \alpha'$，那么存在 β'，使得 $\beta \stackrel{\widehat{a}}{\Longrightarrow} \beta'$，且有 $(\alpha', \beta') \in \mathcal{R}$。
- 如果 $\beta \xrightarrow{a} \beta'$，那么存在 α'，使得 $\alpha \stackrel{\widehat{a}}{\Longrightarrow} \alpha'$，且有 $(\alpha', \beta') \in \mathcal{R}$。

最大的弱互模拟关系 (Weak Bisimilarity) 记做 $\approx$。

定义 2.6 (Branching 互模拟关系 Branching Bisimilarity)**.** 一个关于标号迁移系统中状态的二元关系 $\mathcal{R}$ 是一个 *Branching 互模拟关系 (Branching Bisimulation)* 当且仅当对任意 $(\alpha, \beta) \in \mathcal{R}$，我们有：

- 如果 $\alpha \xrightarrow{a} \alpha'$，那么以下两个命题之一成立：
  - $a = \tau$，且 $(\alpha', \beta) \in \mathcal{R}$；

  - 存在 β''，使得 $\beta \Longrightarrow \beta''$，满足 $(\alpha, \beta'') \in \mathcal{R}$，且存在 β'，有 $\beta'' \xrightarrow{a} \beta'$，满足 $(\alpha', \beta') \in \mathcal{R}$。

- 如果 $\beta \xrightarrow{a} \beta'$，那么以下两个命题之一成立：

  - $a = \tau$，且 $(\alpha, \beta') \in \mathcal{R}$；
  - 存在 α''，使得 $\alpha \Longrightarrow \alpha''$，满足 $(\alpha'', \beta) \in \mathcal{R}$，且存在 α'，有 $\alpha'' \xrightarrow{a} \alpha'$，满足 $(\alpha', \beta') \in \mathcal{R}$。

最大的 Branching 互模拟关系 (Branching Bisimilarity) 记做 $\simeq$。

可以看出，弱互模拟在模拟的过程中忽略了所有内部动作对观测造成的影响，要求相对“宽松”。而 Branching 互模拟从某种意义上来说更加精确地分析了不同内部动作的区别，系统在模拟的过程中可以先经过一系列不改变状态的内部动作，然后进行模拟。这种 Branching 互模拟关系在模拟路径上除最后一步都需要满足不改变系统状态，更加符合实际应用中我们遇到的系统，例如程序优化的过程中，我们也仅能忽略不改变程序状态的内部语句，而不能忽略会对程序结果造成影响的语句。

下面给出一个例子，以说明 Branching 互模拟与强互模拟在验证问题中的应用场景, 如图2–2

$$\begin{array}{ccc} Spec1 & \overset{\sim}{\text{——}} & Spec2 \\ \simeq \Big| & & \Big| \simeq \\ Prog1 & \overset{\simeq}{\text{——}} & Prog2 \end{array}$$

图 2–2 验证问题中的互模拟关系

- $\sim$ 被用来验证两个需求 *(Specifications)* 是否等价。
- $\simeq$ 被用来验证一个程序 *(Program)* 是否是某个需求的正确实现 *(Implementation)*。
- $\simeq$ 被用来验证两个程序 *(Programs)* 是否等价。

在为等价关系的刻画带来精确性的同时，引入内部动作也会增加相应的验证问题的难度，但是如果要使得形式化验证的理论真正能应用在实际的系统中，对内部动作的考虑是不可避免的。

2.3 无限状态系统验证问题

本节将给出无限状态系统验证问题的严格定义，我们进行这方面研究的目标。

定义 2.7 (问题定义)**.** 这里我们给出 3 类问题的严格定义

- $Equivalence\,Checking\ \cong$
  输入：两个进程 α, β，一个等价关系 $\cong$
  问题：$\alpha \cong \beta$?

- $Finitness\ \cong_{FS}$
  输入：一个进程 α，一个 FS γ，一个等价关系 $\cong$
  问题：$\alpha \cong \gamma$?

- $Regularity\ \cong_{REG}$
  输入：一个进程 α，一个等价关系 $\cong$
  问题：是否存在 FS γ，满足 $\alpha \cong \gamma$?

举几个在实际应用中的例子，Equivalence Checking 解决的是一个实现 (Implementation) 是否能满足需求 (Specification)，因为往往这两者都是由某种无限状态系统表示的；而 Finiteness 解决了一个硬件系统 (Hardware Design) 是否满足需求，因为硬件系统都是有限的。而 Regularity 则解决了需求到底能不能被硬件系统实现。

通过第一章中的介绍，我们知道这 3 类问题是有着紧密的内部联系的。而 Regularity 问题也是联系另外两类问题的一个桥梁。如果能有效地判定一个进程的 Regularity 性质，那么往往能根据 Finiteness 问题的已有结论，得到更快速的 Equivalence Checking 算法。

本章最后我们将介绍一下我们研究等价验证问题的目标，给定一个问题，我们希望能证明

可判定性 (Decidable)?

- 可判定，那么我们寻找它的相关算法 (Algorithm)，证明其计算复杂性 (Complexity) 的上界 (Upper Bound) 和下界 (Lower Bound)。
- 不可判定，那么我们探究其不可判定的难度。

我们对于这些问题，在可计算性和算法复杂性上总是希望能得到完备 (Completeness) 的结论。

第三章 相关结论和技术

为了能更好的研究 PRS 上的 Regularity 问题，本章中我们首先将引入一个无限状态系统验证问题中常常用到的重要限定条件，Normed 进程，同时我们会对该限定做进一步的规定，并给出 Totally Normed 进程。之后我们将介绍 PRS 上 Regularity 问题现阶段我们已知的结论，并进行归纳和总结，并得到一些推论。最后将介绍一些 PRS 上证明 Regularity 问题所用到的一些技术和引理。

3.1 Normed 条件

在无限状态系统验证问题中，我们常常要对所研究的模型做一些合理的限定，以便使该模型满足一些我们所需要的性质，从而可以得到更好的可判定行或者算法复杂性的结论。从研究语言理论的角度出发，我们经常要求模型是 *Normed*。从直观上来说，一个 Normed 进程可以通过有限步转换到达一个 ϵ 进程。如果做进一步限定，Totally Normed 进程则不仅要满足 Normed 性质，而且到达 ϵ 进程至少要经过一个可见动作。(Totally) Normed 进程的定义为:

定义 3.1 ((Totally) Normed 进程 (Totally)Normed Process)**.** 一个进程项 α 的 Norm，记做 $\|\alpha\|$，表示从 α 能达到 ϵ 的转换序列的最短长度，如果考虑弱互模拟，则不考虑 τ 动作。

我们称一个进程项 α 是 Normed，如果满足 $0 \leq \|\alpha\| < \infty$，称它是 Totally Normed，如果满足 $0 < \|\alpha\| < \infty$。一个 Normed PRS 进程 Δ 中所有的进程变量均是 Normed。类似的，一个 Totally Normed PRS 进程 Δ 中所有的进程变量均是 Totally Normed。

Normed 条件规定了一个进程只能做有限个减少 Norm 的动作，满足该条件的进程往往会满足一些美妙的分解性质。而 Totally Normed 条件则进一步限定了进程做 τ 动作的限制，没有进程项能只做 τ 动作就变空。Totally Normed 限定也是十分合理的，在现实系统中通常也不需要定义，或者通过一些技巧来规避那种 “悄无声息” 就消失的进程。这些限定在研究考虑内部动作的互模拟关系的验证问题时，发挥了很大的作用。

为了表达的简洁性，我们在相关模型前加上 n 和 tn 分别作为 Normed 和 Totally Normed 的简称。例如 Normed BPA 简称为 nBPA，而 Totally Normed PA 简称为 tnPA。

3.2 PRS 上 Regularity 问题现有结论总结

本节将根据不同的 PRS 子模型，分别总结其现有的关于强互模拟，弱互模拟和 Branching 互模拟的 Regularity 问题的结论，并根据问题的相似性，直接给出一些推论。根据定义2.7中给出的记号，我们将三类问题分别及做 $\sim_{REG}$, $\approx_{REG}$ 和 $\simeq_{REG}$。

表3–1为 BPA 上的结论总结，表中如果出现两行则分别表示该问题已知的复杂度上界和下界。“?” 表示该问题还未被解决。

	BPA	nBPA
$\sim_{REG}$	Decidable [21, 36] PSPACE-hard [45]	NL-complete [45][39]
$\simeq_{REG}$	? EXPTIME-hard [46]	Decidable [30]
$\approx_{REG}$	? EXPTIME-hard [46]	? NP-hard [47, 48]

表 3–1 BPA Regularity 问题现有结论

其中由 Fu 证明的 $nBPA$ 上 $\simeq_{REG}$ 问题的可判定行结论是现在唯一已知的考虑系统内部动作的互模拟等价关系的 Regularity 问题结论的可判定性结论。这一结论利用到了 nBPA 中的 $\simeq$ 的可判定性证明的相关技术。

表3–2总结了 BPP 上的已知结论:

在 BPP 上，我们已经有了 $\sim_{REG}$ 的 PSPACE 完全性结论。这里利用到了一种在 BPP 中常用的 DD-Function 的技术，来对 BPP 的等价类进行划分，但是，对 $\approx$ 和 $\simeq$，DD-Function 并不适用，所以我们还需要寻找其他技术来证明其可判定性。

而对与 nBPP，在 [29]，中，给出了 $\simeq$ 的可判定性结论，而相关的 $\simeq_{REG}$ 问题却一直未被解决。而由于下文中将要提到的引理3.5，我们可知该问题至少

	BPP	nBPP
$\sim_{REG}$	PSPACE-complete [37, 49, 50]	NL [39] NL-hard [50]
$\simeq_{REG}$	?	?
$\approx_{REG}$	? PSPACE-hard [47]	? PSPACE-hard [47]

表 3–2 BPP Regularity 问题现有结论

是半可判定的，所以引起包括本文作者在内的许多研究人员的极大兴趣。但是要完全的解决该问题，还需要客服许多技术上的难题。这一问题还将在本文的第六章中的未解决问题中进一步讨论。

表3–3为 PDA 上的现有结论：

	PDA	nPDA
$\sim_{REG}$	? EXPTIME-hard [33, 45]	P [38] NL-hard [45]
$\simeq_{REG}$	?	?
$\approx_{REG}$	? EXPTIME-hard [33, 45]	? EXPTIME-hard [33, 45]

表 3–3 PDA Regularity 问题现有结论

在 PDA 上，我们似乎还没有找到一些好的方法来处理 Regularity 问题，即使对 $\sim_{REG}$。我们只是关于 nPDA 的 $\sim_{REG}$ 问题，有多项式时间的算法。

表3–4总结了 PN 上的现有结论：

PN 这类并行进程的研究中经常会用到 Presburg 算数来刻画其中的一些问题，从而能得到一些直接的判定性结论。同时，对于 PN 中的进程，我们有一个等价的向量表示，称为 Vector Addition System(VAS)，其中的很多技术可以用于 PN 中相关问题的证明，如可达性问题 (Reachability)。同时这种向量表示在相关验证的问题的表述中也会带来很大的方便性。

这里我们注意表3–4中 * 所标的结论，即 PN 上 $\simeq_{REG}$ 的不可判定性并未直

	PN	nPN
$\sim_{REG}$	Decidable [49] PSPACE-hard [50]	EXPSAPCE [51] EXPSPACE-hard [52]
$\simeq_{REG}$	Undecidable*	?
$\approx_{REG}$	Undecidable [49] EXPSPACE-hard [52]	? EXPSPACE-hard [52]

表 3–4 PN Regularity 问题现有结论

接在引用论文中出现，而是本文作者经过与合作者的严格论证，将 $\approx_{REG}$ 不可判定性的证明技术，应用到 $\simeq_{REG}$ 上，得出的一个引文 [49] 的推论。

推论 3.1 (PN $\simeq_{REG}$ 的不可判定性)**.** *PN 的 $\simeq_{REG}$ 问题是不可判定 (Undecidable) 的。*

另外我们注意到，BPP 实际是一种特殊的 Petri 网，在**??**中定义了这种 PN 的子类，称为 Communication Free Petri 网，它其实就是一种进程间通信的并发模型。利用这一等价模型，我们可以利用 PN 中的技术和结论，通过条件的加强，来证明 BPP 上相关问题的结论。

最后表3–5将总结 PA 上的相关结论，因为我们将要研究的问题是 tnPA 上的，所以这里同时还总结了 tnPA 上的结论。

	PA	nPA	tnPA
$\sim_{REG}$	? PSPACE-hard [50]	NL [39] NL-hard [50]	NL [39] NL-hard [50]
$\simeq_{REG}$	? EXIPTIME-hard [46]	? PSPACE-hard [47]	P* NL-hard [50]
$\approx_{REG}$	? EXPTIME-hard [46]	? PSPACE-hard [47]	P* NL-hard [50]

表 3–5 PA Regularity 问题现有结论

PA 模型的进程项中由于同时允许串行和并行连结符的存在，所以为判定

性带来了很大的难度。很大程度上我们现阶段的证明只能在 nPA 的限制下，通过一些巧妙的技术来规避这些连结符的复杂性带来的困扰。

表3–5中 * 所标的两个多项式时间判定算法为本文的主要结论，该结论同样适用于 PA 的子模型 tnBPA 和 tnBPP。同时该问题的时间复杂度也将在本文中给出。

3.3 相关技术路线和引理

本节中，我们将对 PRS 下 Regularity 问题证明的技术路线进行讨论，并给出一些与具体模型无关的，关于 Regularity 问题的一般性引理。

我们通常用 (α, Δ) 来表示初始状态为 α，转换规则集合为 Δ 的某个 PRS 进程。

Regularity 问题是在一个可以表示无限状态的系统中，探究它的有限状态的性质。从进程进行转换的角度出发，我们可以给出一个进程不满足 Regularity 性质的等价条件。

从进程行为的角度出发，为了寻找进程中状态的无限性，我们需要寻找一个如引理3.2中的进程无限长的动作序列。

引理 3.2 (Infinite $\not\cong$ Path). *一个进程系统 (α, Δ)，关于等价关系 $\cong$ 不是 Regular 的，当且仅当存在一个无限长的动作序列*

$$\alpha \xrightarrow{a_0} \alpha_1 \xrightarrow{a_1} \alpha_2 \xrightarrow{a_2} \dots$$

满足对任意 $i \neq j$，有 $\alpha_i \not\cong \alpha_j$。

证明. 我们分别证明充分性和必要性：

- $\Rightarrow$ 假设 (α, Δ) 不是 $\cong_{REG}$ 的，那么我们将所有从 α 出发的动作序列看成一棵以 α 为根的树 (Tree)。由于所有转换规则构成一个有限集合，所以这个图也是有限分枝的 (Finite Branching)。由假设，(α, Δ) 不是 $\cong_{REG}$ 的，所以这颗树上有无限个互补关于 $\cong$ 等价的不同的状态。那么，就一定存在一条满足条件的无限长的序列。否则，这样的路径就会变成一个环，由于该树只有有限分枝，就不存在无限的关于 $\cong$ 互不相同的状态了。这样就证明了该序列一定存在。

- $\Leftarrow$ 存在这样的无限长动作序列，则 (α, Δ) 满足 Regularity 的定义。

□

相对于特定的互模拟等价 $\cong$ 的判定，一个更容易的任务是判定两个进程项在表示上是否完全相同 (Lexically Equivalent)。即两个进程是否是完全相同的表达式。我们用 $\equiv$ 来表示。我们下面给出一个 $\ncong_{REG}$ 的必要条件。

引理 3.3 (Infinite $\neq$ Path)**.** *一个进程系统 (α, Δ)，关于等价关系 $\cong$ 不是 Regular 的，那么存在一个无限长的动作序列*

$$\alpha \xrightarrow{a_0} \alpha_1 \xrightarrow{a_1} \alpha_2 \xrightarrow{a_2} \ldots$$

满足对任意 $i \neq j$，有 $\alpha_i \neq \alpha_j$。

证明. 和引理3.2证明类似，与之同理，我们考虑以 α 为根的一棵转换动作的树。如引理3.2中的无限长路径一定存在，而 $\ncong$ 则一定有 $\neq$。证毕。 □

关于 Normed 进程，我们显然有 Norm 不相等的进程是不互模拟等价的。这样我们就有了一个关于 Normed 进程的 Regularity 问题的充分条件。

引理 3.4 (Infinite Norm Increasing Path)**.** *一个进程系统 (α, Δ)，如果存在一个无限长的动作序列*

$$\alpha \xrightarrow{a_0} \alpha_1 \xrightarrow{a_1} \alpha_2 \xrightarrow{a_2} \ldots$$

满足对任意 i，存在 j，有 $\|\alpha_i\| < \|\alpha_j\|$。那么该进程系统不是 Regular 的。

证明. 如果存在这样的动作序列，那么 (α, Δ) 显然满足不是 Regular 的定义。 □

另一个重要的引理是，如果一个进程系统 (α, Δ) 的某个等价关系 $\cong$ 或 $\cong_{FS}$ 是可判定的 *(Decidable)*，那么相应的 $\cong_{REG}$ 是半可判定 *(Semi-Decidable)* 的。这是一个对于 Regularity 问题十分重要的结论，因为这样在很多情况下，我们就只需要再去证明另一半问题 $\ncong_{REG}$ 的半可判定性了。

引理 3.5 (Semi-Decidability)**.** *若 (α, Δ) 的 $\cong$ 或者 $\cong_{FS}$ 是可判定的 (Decidable)，那么 $\cong_{REG}$ 是半可判定的 (Semi-Decidable)。*

证明. 考虑以下过程，我们通过对有限状态系统 FS 的某种编码，可以有效地枚举 (Enumerate) 其中的进程 (γ, Δ')。接着我们利用 $\cong$ 或者 $\cong_{FS}$ 的相应算法判定其是否与 (α, Δ) 等价。如果等价，则该过程终止，该进程系统满足 Regularity 性质，否则继续重复枚举过程。该过程即是一个 $\cong_{REG}$ 的半可判定算法。 □

而对于另一半问题 $\ncong_{REG}$ 的研究，由于我们不可能直接去验证路径的无限长度这样的性质，所以我们只能去寻找一些进程生成的状态树中的一些有限的子结构。例如一些可能通过重复作用而生成出无限状态的动作串。这类子结构在各种不同的模型中会满足不同的性质，常常会在 Regularity 的判定算法中起到关键作用。对于 tnPA，本文将在第四章中具体研究这种子结构。

第四章　Totally Normed PA Regularity 的充要条件

在这一章中，我们将具体讨论和证明 tnPA 的 $\approx_{REG}$ 和 $\simeq_{REG}$ 问题的充分必要条件，定理4.6。这里将 [39] 中用于判定 nPA 的 $\sim_{REG}$ 的一些技术应用到我们讨论的问题中，起到了很好的效果。这里证明的充要条件将会在第五章中的多项式算法中被判定。

这里需要说明的是，由于这些条件都需要是可判定的，在判定的过程中我们需要一个计算复杂性上可行的算法。这就意味着我们所给出条件的判定过程需要有一个确定的，有限的，可以有效计算的上界。否则判定算法上会发生状态数量的爆炸，整个过程就丧失了计算上的可行性。这个上界会在接下来的分析中一步步变得清晰，我们要遵守的一个原则是这个上界只能和输入进程的语法规则和初始状态的规模相关。

4.1　PA 模型定义

为了问题研究的严谨性，我们这里在 PRS 的一般性定义2.3的基础上，给出 PA 的严格定义。

定义 4.1 (PA 定义)**.** 一个 PA 系统是一个形如 $X \xrightarrow{a} \alpha$ 的有限转换规则的集合 Δ，一个 PA 表达式的语法定义为:

$$\alpha, \beta ::= X \mid \alpha.\beta \mid \alpha \parallel \beta \mid \alpha \between \beta$$

其中用大写英文字母表示进程变量, 希腊字母表示进程项 (状态)。相应的操作语义可以定义为一下规则

$$\frac{\alpha \xrightarrow{a} \beta \in \Delta}{\alpha \xrightarrow{a} \beta} \qquad \frac{\alpha \xrightarrow{a} \alpha'}{\alpha.\beta \xrightarrow{a} \alpha'.\beta} \qquad \frac{\alpha \xrightarrow{a} \alpha'}{\alpha \| \beta \xrightarrow{a} \alpha' \| \beta}$$

$$\frac{\beta \xrightarrow{a} \beta'}{\alpha \| \beta \xrightarrow{a} \alpha \| \beta'} \qquad \frac{\alpha \xrightarrow{a} \alpha'}{\alpha \between \beta \xrightarrow{a} \alpha' \| \beta}$$

一个 PA 进程可以具体表示为 (α, Δ)。

我们用函数 $Var(\_)$ 表示 _ 中所包含的进程变量。

我们要求一个 PA 中定义的所有进程变量 $X_i \in Var(\Delta)$ 存在一个可达的状态 α 满足 $_i \in Var(\alpha)$。

另外，我们用 $Length(\alpha)$ 表示 α 中出现的进程变量的个数。

同时，我们限制在章里我们所关心的 PA 进程都是 Totally Normed 的。同时如果不特殊说明，我们证明中所用到的等价关系都是弱互模拟关系 $\approx$。这些证明对于 $\simeq$ 都是成立的。

4.2 进程状态的增长

给定一个 tnPA 的进程 (α, Δ)，我们希望能找到导致它产生无限中不同状态的进程变量的集合。这些变量导致了进程状态所能产生的增长，是导致进程非 Regularity 的原因。

我们首先针对一个 tnPA 进程，定义两个十分有用的函数。

定义 4.2 (Fire 函数)**.** 给定一个 tnPA 进程 (α, Δ)

$$Fire(\alpha) = \begin{cases} \emptyset & \text{if } \alpha = \epsilon \\ \{X\} & \text{if } \alpha = X \\ Fire(\beta_1) & \text{if } \alpha = \beta_1.\beta_2 \text{ or } \alpha = \beta_1 \between \beta_2 \\ Fire(\beta_1) \cup Fire(\beta_2) & \text{if } \alpha = \beta_1 \| \beta_2 \end{cases}$$

$Fire()$ 函数会返回那些当前状态下潜在可以被激发 (Active) 的进程变量的集合。

定义 4.3 (Tail 函数)**.** 给定一个 tnPA 进程 (α, Δ)

$$Tail(\alpha) = \begin{cases} \{X\} & \text{if } \alpha = X \\ \emptyset & \text{if } \alpha = \epsilon \text{ or } \alpha = \beta_1 \| \beta_2 \text{ where } \beta_1 \neq \epsilon \neq \beta_2 \\ Tail(\beta_2) - Var(\beta_1) & \text{if } \alpha = \beta_1.\beta_2 \text{ or } \alpha = \beta_1 \between \beta_2 \text{ where } \beta_1 \neq \epsilon \neq \beta_2 \end{cases}$$

$Tail()$ 函数返回所有在当前状态下被阻塞 (Block)，即必须先将别的变量转换成 ϵ，最后才能激发的进程变量的集合。

通过观察这两个函数的性质，我们能得到以下性质，引理4.1:

引理 4.1. *假设我们有 $X \in Var(\alpha)$ 满足 $X \notin Tail(\alpha)$，那么必然存在 α'，使得 $\alpha \rightarrow^* \alpha'$，且 $X \in Fire(\alpha')$ 且 $Length(\alpha') \geq 2$。*

证明. 用反证法，如果 $X \in Fire(\alpha')$ 且 $Length(\alpha') = 1$。那么 X 可以会在别的进程变量都消失后再被激发，就有 $X \in Tail(\alpha)$，矛盾。 □

同时，引理4.2也是十分自然的：

引理 4.2. *给定 tnPA 进程 (α, Δ)。那么对于所有 $X \in Var(\alpha)$，存在 (β, Δ)，满足 $\alpha \rightarrow^* \beta$，且 $X \in Fire(\beta)$。*

证明. 由 $Fire$ 函数的定义直接显然成立。 □

在我们考虑解决 tnPA 进程的 Regularity 问题时，希望找到一个充分必要条件。由引理3.3中介绍的必要条件，我们首先希望能找到那种有潜在生成无限种不同语言 (Lexically) 上不同的进程表达式能力的进程变量。然后我们再去证明这些表达式间的互不互模拟性，从而证明其充分性。

根据这种思路，我们给出关于能使进程*增长 (Growing)* 的进程变量的定义4.4：

定义 4.4 (增长变量 Growing Variable)**.** 给定一个 tnPA 进程的规则集合 Δ，对于一个变量 $X \in Var(\Delta)$，如果存在 α，满足 $X \rightarrow^* \alpha$，$X \in Fire(\alpha)$ 且 $(\alpha) \geq 2$。那么我们称 X 是*增长的 (Growing)*。

4.3 转换关系树

接下来，我们将会证明在 tnPA 中，Growing 的进程变量就是导致进程状态无限性的原因。再回顾一下引理3.2中的提供的思路，当我们希望验证状态的无限性时，我们只需要考虑那种无限长的形如 $\alpha \xrightarrow{a_0} \alpha_1 \xrightarrow{a_1} \alpha_2 \xrightarrow{a_2} \ldots$，且满足 $\alpha_i \not\approx \alpha_j$ for $i \neq j$ 的动作路径 P。为了考察路径中那些进程表达试的长度，我们将使用一个树状的结构对其进行表示，我们称之为*转换关系树 (Transition Tree)*，记做 $T_{\mathcal{P}}$，在定义4.5中利用归纳法，我们给出其严格的定义。利用 $T_{\mathcal{P}}$，我们可以准确刻画路径 $\mathcal{P}$ 中连续出现的变量之间的关系。这些信息对我们接下来的证明是有很大用处的。

定义 4.5 (转换关系树)**.** 给定一个 tnPA 进程 (α, Δ) 和一个形如 $\alpha \xrightarrow{a_0} \alpha_1 \xrightarrow{a_1} \alpha_2 \xrightarrow{a_2} \ldots$ 的状态转换路径 $\mathcal{P}$ 一个**转换关系树** *(Transition Tree)*，$T_{\mathcal{P}} = (V, E)$ 定义如下：

1. V 是结点的集合，它们由 $Var(\Delta) \cup \{R\}$ 进行标号，其中 R 表示树根的特殊标号。

2. E 是树中边的集合

3. 路径 $\mathcal{P}$ 中每一个状态，例如 α_j，被树里的第 $j+1$ 层记录，记做 $Level_{j+1}$。这层中的节点被 α_j 中的变量所标记，如果重复出现，则重复标记。

$T_{\mathcal{P}}$ 的拓扑结构 (Topological Structure) 可以利用树的层数 i 进行归纳定义。

1. 当 $i = 0$：为树根，标记为 R。

2. 当 $i = 1$：这一层对应了进程的初始状态 α_0。

3. 当 $i > 1$: 我们假设 $Level_i$ 上所有节点都已经被定义，接着我们试图定义 $Level_{i+1}$ 上的所有节点。我们现在定义 $Level_i$ 上所有节点的子节点的规则：

   根据定义4.1中 PA 的操作语义，我们知道对于一步转换 $\alpha_{i-1} \xrightarrow{a_i} \alpha_i$，一定存在一个变量，假如是 $A \in Var(\alpha_{i-1})$ 和一条转换规则 $A \xrightarrow{a_i} \gamma_i \in \Delta$ 被激发。我们称这个变量 A 为 α_{i-1} 中的**激发变量** *(Active Variable)*，对应的树中的节点为 $Level_i$ 中的**激发节点** *(Active Node)*。

   我们根据此将 $Level_i$ 中所有节点 $N \in V$ 分成两类分别定义其子节点的标号规则：

   (a) 如果 N 不是一个激发节点。那么 N 只有一个儿子节点，标号和 N 相同。而且，一条在它们之间连一条边来表示他们之间的标号继承关系。

   (b) 如果 N 是一个激发节点。我们假设它对应的进程变量为 A，而相应的转换规则为 $A \xrightarrow{a_i} \gamma_i$。令 $n = Length(\gamma_i)$。如果 $n > 0$，那么节点 N 有 n 个子节点，否则它是一个叶子节点。它的第 k 个子节点的标号就是 γ_i 中从左到右的第 k 个变量。

观察这个定义，我们可以看出进程转换中标号变量的继承关系的信息可以在树中表示出，而且保留了一部分的进程的结构信息。下面的引理4.3告诉了我们这样的子节点的排列给我们带来的一些信息。

引理 4.3. 给定一个转换关系树 $T_{\mathcal{P}}$，N 是 $_{i+1}$ 层中的一个标号为 A 的激发节点，其对应的规则是 $A \xrightarrow{a_i} \gamma_i$。它的儿子节点从左到右分别为 $B_1, B_2, \ldots, B_n$。假设 $Tail(\gamma_i) = \{B\}$，那么 N 的最后一个子节点的标号就是被阻塞的进程变量 B。

证明. 很明显，这个性质可以由定义4.3和定义4.5中阻塞变量以及转换关系树的定义中得出。

从 $Tail(\gamma_i)$ 的定义我们知道：

- 如果 $Length(\gamma_i) = 1$ 那其中唯一的变量即为阻塞变量，
- 否则，γ_i 的结构有两种可能性：$\gamma_i = \beta_1.\beta_2$ 或者 $\gamma_i = \beta_1 \| \beta_2$. 在这两种情况下，阻塞的变量只能出现在右边的进程中。所以显然被阻塞的变量只能出现在最右边的儿子的标号中。 □

更进一步分析，从 $T_{\mathcal{P}}$ 中我们还可以的到那些潜在会增长的变量的信息。这些节点有两个或两个以上的子节点，我们称它们为**分枝节点** *(Branching Nodes)*。这些节点在分析增长变量的过程中起到了十分重要的作用。为了分析的方便，在定义4.6，我们为每个节点定义一个辅助的集合。

定义 4.6 (Finite 集合). 给定一个转换关系树 $T_{\mathcal{P}}$，对于每个节点 N，集合 $Finite(N) \subseteq Var(\Delta)$ 由在树的层次归纳定义如下：

1. 如果 N 是 $T_{\mathcal{P}}$ 的树根，那么 $Finite(N) = Var(\Delta)$。
2. 否则，我们假设 N 的父节点是 M：
   (a) 如果 M 不是分枝节点，那么 $Finite(N) = Finite(M)$。
   (b) 如果 M 是分枝节点，而且 N 在 M 中未被阻塞，那么 $Finite(N) = Finite(M) - \{Label(M)\}$。
   (c) 如果 M 是分枝节点，而且 N 在 M 中被阻塞，那么 $Finite(N) = Finite(M)$。

$Finite(N)$ 集合刻画了使得以 N 为根节点的子树能保持有限性质的那些 N 的后代。这一性质在引理4.4中得到了应用。

引理 4.4. *给定一个转换关系树 $T_{\mathcal{P}}$，对于每一个节点 N，若 $Label(N) \notin Finite(N)$，那么 $Label(N)$ 是一个增长变量。*

证明. 我们根据 $Finite(N)$ 的生成过程来证明这个结论。假设 $Label(N) = X$。由定义4.6，我们可以发现从这个集合中去掉变量 X 的唯一可能是，存在另一个节点 M，满足以下条件：

1. M 是 N 在树中的一个*祖先 (Ancestor)*。
2. $Label(M) = X$。
3. M 是一个分支节点且有一个子节点 O，满足 O 在 M 中未被阻塞，且 N 由 O 被继承。

由于 O 在 M 中未被阻塞，那么所有 O 子孙，包括 N，不会在 M 中被阻塞。即 $X \notin Tail(X)$，于是存在 α，满足 $X \rightarrow {}^{*}\alpha$，$X \in Fire(\alpha)$ 且 $Length(\alpha) \geq 2$。

由定义4.4，我们可以得到 X 是一个增长变量。 □

为了证明定理4.6中本章的主要结论，接下来我们针对转换关系树 $T_{\mathcal{P}}$，给出关于它的概念，用来刻画它的一些细节，以方便更精细的分析。在定义4.7中我们将树*分段 (Segment)* 进行研究。

定义 4.7 (树的分段 Breaking Levels and Segments)**.** 给定一棵转换关系树 $T_{\mathcal{P}}$，

- 我们称那些只包含一个节点的层为*分割层 (Breaking Levels)*，这些层中的节点为*分割节点 (Breaking Nodes)*。
- 通过分割节点，我们可以将树 $T_{\mathcal{P}}$ *分段 (Segments)*。

  一个段 S 是 $T_{\mathcal{P}}$ 中的一个子图，由分割层进行分割。每个分割节点都是一个段的*起始节点 (Starting Node)*，直到下一个分割节点，*结束节点 (Ending Node)*，以前的所有节点和边都属于这个段。如果不存在这样的结束节点，那么该段包含从起始节点向下的所有节点和边。

  这里我们用 $Seg(N)$ 表示一个以 N 为起始节点的段。

下面我们在定义4.8中给出另一个用来衡量的树的宽度结构的函数，我们可以通过类似的定义来衡量段的相关性质。

定义 4.8 (树的宽度 Width)**.** 给定一个以 N 为根节点的转换关系树 $T_{\mathcal{P}}$，树的宽度函数 $WidthT(N)$ 定义如下:

$$WidthT(N) \quad = \quad \begin{cases} w & card(Level_i) \text{ 的最大值}, i \in \mathbb{N} \cup \{0\} \\ \infty & \text{如果不存在最大值} \end{cases}$$

其中函数 $card(S)$ 表示集合 S 中互不相同的元素的个数，即集合的基数 (Cardinality)。

类似的我们可以为一个段 $Seg(N)$ 定义它的宽度函数 $WidthS(N)$，或者为 $T_{\mathcal{P}}$ 的一个以 N 为根节点的子树定义它的宽度函数 $WidthT(N)$。

下面在定义4.9中，我们为转换关系树定义了一个*分枝度 (Branching Degree)*，它的最大值被输入的 tnPA 的规则所限定。

定义 4.9 (Branching Degree)**.** 给定一个转换关系树 $T_{\mathcal{P}}$，它的*分枝度 (Branching Degree)*，为它节点的子节点个数的最大值，即其中分枝数的最大值。我们用 **D** 表示这个数。

每个节点的分枝数由它对应被激活的转换规则所决定，所以 **D** 的最大值由输入中进程的规则 Δ 所限定，不会超过输入的最长的规则的长度，所以它的上界 (Upper Bound) 是*有限 (Finite)* 的且可以*有效计算 (Effectively Computable)* 的。

利用转换关系树的辅助，我们可以对进程的行为作出更细致的分析。这是一个关于进程转换路径的语法上的表示，它可以更精确地刻画其中进程表达式的“长度”，并用子节点的顺序来记录部分进程变量执行顺序的信息。通过本节的分析，我们定义了一些转换关系树上有用的结构和概念，接下来，我们就可以试图证明造成进程状态增长的原因。

4.4 等价条件证明

本节中我们将利用第4.2节中的增长变量和第4.3节中的转换关系树，来证明一个 tnPA 进程非 Regular 的等价条件。

首先我们需要证明引理4.5，它给了不包含增长变量的进程所对应的转换关系树中的所有子树的宽度一个上界。

引理 4.5. 给定一个 *tnPA* 进程 (α, Δ) 和它的一个转换关系树 $T_{\mathcal{P}}$。如果 $Var(\Delta)$ 不包含任何增长变量，那么 $T_{\mathcal{P}}$ 中的每个节点 N，满足 $WidthT(N) \leq D^{n-1}$，其中 $n = card(Finite(N))$。

证明. 我们显然可以得到 $n > 0$，如果假设 $n = 0$，那么 $Finite(N) = \emptyset$。由引理4.4，由于 $Label(N) \notin (N) = \emptyset$，可知 $Lable(N)$ 是一个增长变量，和假设矛盾。所以我们之考虑 $n > 0$ 的情况。

现在我们可以利用关于 n 的归纳法进行证明：

1. $n = 1$：这种情况我们只要证明 $WidthT(N) = \mathbf{D}^{n-1} = 1$。

   假设以 N 为跟的子树中有一个分枝节点 N_1。不妨设 $Finite(N) = \{X\}$，那么有 $Finite(N_1) = \{X\}$。这是因为 N_1 是 N 的一个后代节点，而且 N_1 不可以是一个增长节点，根据引理4.4，可知 $Label(N_1) = X$。

   由假设 N_1 是一个分枝节点，那么 $N1$ 至少有一个子节点 N_1' 在 N_1 中不被阻塞，于是有 $Finite(N_1') = \emptyset$。且有 $Label(N_1') = X_1 \notin Finite(N_1')$，则 X_1 只能是一个增长节点。产生矛盾。

   这就意味着，这棵子树中不可能包含任何分枝节点，即 $WidthT(N) = \mathbf{D}^{n-1} = 1$。

2. $n > 1$：这种情况我们只要证明以 N 为根节点的子树中的每一个段的宽度都不会超过 $\mathbf{D}^{n-1}$，其中 $n = card(Finite(N))$。另 S 为该子树中一个以 N_1 为起始节点的段。我们显然有 $card(Finite(N_1)) \leq n$。

   下面我们考虑 N_1 的四种可能情况，从而找到 $WidthS(N_1)$ 的上界：

   (a) 若 N_1 是一个叶子节点。
   那么显然 $WidthS(N_1) = 1$

   (b) 若 N_1 不是叶子节点，也不是一个分枝节点。
   那么它唯一的儿子节点就是段 S 的结束节点，可知 $WidthS(N_1) = 1$。

   (c) 若 N_1 是一个分枝节点，而且 N_1 中没有子节点被阻塞。
   令 N_1' 为 N_1 的一个子节点，我们有 $Finite(N_1') = Finite(N_1) - \{Label(N_1)\}$。
   所以有 $card(Finite(N_1')) \leq n - 1$。
   由归纳假设，我们有 $WidthT(N_1') \leq \mathbf{D}^{n-2}$。$N_1$ 最多有 $\mathbf{D}$ 个子节点，所以有 $WidthS(N_1) \leq WidthT(N_1) \leq \mathbf{D} \times \mathbf{D}^{n-2} = \mathbf{D}^{n-1}$。

(d) 若 N_1 是一个分枝节点而且有一个被阻塞的子节点 N_b。
我们首先考虑除去 N_b 以外的那些子节点。不妨设 N_1' 为这样的节点，根据上一种情况的证明，我们有 $WidthT(N_1) \leq \mathbf{D}^{n-2}$
然后对于那个被阻塞的节点 N_b，我们试图证明它在直到段 S 之前的子孙节点都不会有任何分枝节点。
如果 N_B 的子孙节点中有一个分枝节点 N_b' 它属于段 S 但它不是结束节点 N_e。由于 N_b' 是分枝节点，那么他是一个激活的节点。另一方面，我们有 N_b 是被阻塞的节点，所以它的子孙节点只能在其他节点所对应的变量都消失之后才能被激活。我们可以断言 N_b' 必须是 N_e 或者是它的一个子孙，这是由于 N_e 是 N_b 下面第一个分割节点。所以 N_b' 不可能属于 S，推出矛盾。
这就意味着 N_b 在 N_e 之前的所有子孙节点都不是分枝节点, 所以 N_b 对 $WidthS(N_1)$ 的贡献只有 1。
最后我们可以总结这种情况下 $WidthS(N_1) \leq (\mathbf{D}-1) \times \mathbf{D}^{n-2} + 1 \leq \mathbf{D}^{n-1}$。

于是我们有 $WidthT(N) \leq \mathbf{D}^{n-1}$。

本引理得证。 □

有了之前的准备，我们现在就开始证明本章中的主要结论，定理4.6。它为 tnPA 的 $\approx_{REG}$ 和 $\simeq_{REG}$ 问题证明了一个充分必要条件。这是进行 Regularity 判定算法设计的理论基础，它确保了第五章中算法的正确性。

定理 4.6 (tnPA $\approx_{REG}$ 的等价条件)**.** *一个 tnPA 进程 (α, Δ) 满足 $\approx_{REG}$ 当且仅当$Var(\Delta)$ 不包含任何增长变量。*

证明. 我们分别证明充分性和必要性。

- $\Rightarrow$ 假设 tnPA 进程中存在增长变量，为 $X \in Var(\Delta)$。我们需要证明利用 Δ 中的规则可以到达无限个互不弱互模拟的的状态。
  有假设中 (α, Δ) 的定义，可知存在某个状态 α_x，满足 $\alpha \rightarrow^* \alpha_x$，且 $X \in Var(\alpha_x)$。由引理4.2可知，存在一个状态 β_x，满足 $\alpha \rightarrow^* \alpha_x \rightarrow^* \beta_x$，且 $X \in Fire(\beta_x)$。

我们知道，互模拟的进程的 Norm 必然想等，由引理3.4可知，我们只需证明对于任意 $k \in \mathcal{N}$，存在一个可达的状态 β_k 满足 $\|\beta_k\| \geq k$。

这里我们选择状态 β_x 进行分析，令 X 为激活变量。由于 X 是一个增长变量，所以存在一个可达的状态 β_1'，满足 $X \rightarrow {}^*\beta_1'$，$X \in Fire(\beta_1')$，且 $Length(\beta_1') \geq 2$。

所以存在一个状态 β_1，形如 $\beta_x \rightarrow {}^*\beta_1$，我们将其中被激活的变量 X 转换为 β_1'。另外，我们有 $X \in Fire(\beta_1)$，因为 $X \in Fire(\beta_1')$，且 $Fire(\beta_1') \subseteq Fire(\beta_1)$。

这样我们可以重复这一过程，从 β_1 中构造出 $\beta_2, \beta_3 \ldots$，这些状态满足 $X \in Fire(\beta_i)$，且 $Length(\beta_{i+1}) > Length(\beta_i)(i = 2, 3, \ldots)$。即满足 $Length(\beta_k) \geq k$。

由假设，我们有 (α, Δ) 是 Totally Normed，所以有对于每一个变量 $Y \in Var(\beta_k)$，满足 $\|Y\| \geq 1$。结合之前 $Length(\beta_k) \geq k$ 的结论，我们就可以从 Totally Normed 性质得出 $\|\beta_k\| \geq k$。

必要性得证。

- $\Leftarrow$ 我们只需要证明，如果一个 tnPA 进程 (α, Δ) 不满足 $\approx_{REG}$，那么一定存在一个增长变量 $X \in Var(\Delta)$。

  我们基本的证明策略是找到一个不包含增长变量的进程所能产生语言上互不相同的表达式数量的上界。假设一个 tnPA 进程 (α, Δ) 不包含任何一个增长变量，那么我们只需要证明它是 $\approx_{REG}$ 的。或者我们证明如果它不是 $\approx_{REG}$ 的，那么必然存在矛盾。

  由引理3.2，这个目标等价于证明如果存在一个形如 $\alpha \xrightarrow{a_0} \alpha_1 \xrightarrow{a_1} \alpha_2 \xrightarrow{a_2} \ldots$ 的无限长度转换动作序列 $\mathcal{P}$，那么存在 $i \neq j$ 且 $\alpha_i \approx \alpha_j$。

  现在令 $T_{\mathcal{P}}$ 为 $\mathcal{P}$ 的转换关系树，我们进一步对树的宽度进行分析，以找到该进程潜在的能生成的进程表达式的长度的上限。

  令 N 为 $T_{\mathcal{P}}$ 的根节点。应用引理4.5，可知 $WidthT(N) \leq \mathbf{D}^{n-1}$，其中 $n = card(Finite(N))$，且 $\mathbf{D}$ 为 $T_{\mathcal{P}}$ 的分枝度。

  显然，我们有 $card(Finite(N)) \leq card(Var(\Delta))$。那么有 $Length(a_i) \leq WidthT(N) \leq \mathbf{D}^{card(Var(\Delta))-1} = m$ 其中，$i = 1, 2, \ldots$。这里 m 是一个由输入进程规则决定的常数，也就是进程从初始状态所可能生成的进程的长度

由这个常数所限定。所以在 $\mathcal{P}$ 中最多只有有限 (Finite) 个语言 (Lexically) 上互不相同的表达式。

最后，必然存在 $i \neq j$，满足 $\alpha_i \equiv \alpha_j$，蕴含了 $\alpha_i \approx \alpha_j$。产生矛盾。

充分性得证。

综上，本定理得证。 □

这个证明中有两处涉及到了 $\approx$，首先在必要性的证明中如两个 tnPA 进程 Norm 不相等，那么它们必然不是 $\approx$ 的。同理，它们也不是 $\simeq$ 的。

在充分性的证明中，我们用到了如果两个进程是 $\equiv$ 那么必然是 $\approx$ 的，同理，显然对于 $\equiv$ 的两个进程，也满足 $\simeq$。综上所述，该定理对与 $\simeq_{REG}$ 一样成立，我们有推论4.7:

推论 4.7 (tnPA $\simeq_{REG}$ 的等价条件)**.** *一个 tnPA 进程 (α, Δ) 满足 $\simeq_{REG}$ 当且仅当$Var(\Delta)$ 不包含任何增长变量。*

需要注意的是，该证明对更加一般的 nPA 模型并不成立。因为我们在必要性的证明中用到了 tnPA 中所有进程变量的 Norm 至少为 1 的限定条件。如果去掉这一限定，那么对于 Norm 为 0 的那些进程，我们就不能用进程表达式语言上的长度限制来得到 Norm 上的下界了。如果要克服这一问题，我们需要对进程的行为进行更精细的分析，以得到更强大的性质。

在第五章中，我们将集中讨论如何找到一个有效的算法对该条件进行判定，并分析该算法的时间复杂度，从而找到解决 tnPA 的 Regularity 问题的具体算法。

第五章　Totally Normed PA Regularity 的算法

本章中，我们将利用第四章中定理4.6中给出的充分必要条件，设计判定 tnPA 的 $\approx_{REG}$ 和 $\simeq_{REG}$ 的算法。我们将证明该问题可以在多项式时间内判定。

5.1　增长变量判定算法

在定理4.6中，我们给出了 tnPA 的 Regularity 性质的一个充分必要条件，即增长变量的性质。本节我们对一个给定的 tnPA 进程 (α, Δ)，将给出一个多项式时间算法。由我们 PA 进程的定义4.1中的假设，对于每个进程变量 $X \in Var(\Delta)$，必定存在一个可达的状态，使他可以被激活。所以，我们只需要对输入中给定的规则，判断其中是否包含一个增长变量。

我们现在证明引理5.1。

引理 5.1 (增长变量的可判定性 Decdability for Growing Variables)**.** *给定一个 tnPA 进程 (α, Δ)，其中是否存在一个增长变量 $X \in Var(\Delta)$，是可判定的。*

证明. 见算法1 *GROW*。 □

其中 $Transition1$ 和 $Transition2$ 分别由算法2 *TRANSITION1*，和算法3 *TRANSITION2* 计算：

1. 算法1 *GROW* 的第一步中，对于函数 *Transition1(S)* 的迭代调用计算了关系 (X, Y) 的自反传递闭包，其中 (X, Y) 满足存在一条规则 $X \xrightarrow{a} \alpha \in \Delta$，且 $Y \in Var(\alpha)$。
   得到的集合恰好包含了由 X 生成的所有可能被激活的变量。
2. 下一步中，对 *Transition2(S)* 的调用，计算了关系 (X, Y)。(X, Y) 满足存在一条规则 $X \xrightarrow{a} \alpha \in \Delta$，且有 $Y \in Var(\alpha)$，$Y \notin Tail(\alpha)$，以及 $Lenth(\alpha) \geq 2$。
   这一步结束后，集合中所有变量都是被激活的，且集合包含了所有满足不被阻塞，而且经过一步转换后至少还存在另一个变量的变量。

Algorithm 1 GROW

Grow(Δ)

for all $X \in Var(\Delta)$ **do**
  $S \leftarrow \{X\}$
  while $S \neq Transition1(S)$ **do**
    $S \leftarrow Transition1(S)$
  end while
  $S \leftarrow Transition2(S)$
  while $S \neq Transition1(S)$ **do**
    $S \leftarrow Transition1(S)$
  end while
  if $X \in S$ **then**
    return TRUE
  end if
end for
return FALSE

Algorithm 2 TRANSITION1

Transition1(S)

for all $Y \in Var(\Delta)$ **do**
  if $\exists X \in S$ and a rule $X \xrightarrow{a} \alpha \in \Delta$ and $Y \in Var(\alpha)$ **then**
    $S \leftarrow S \cup \{Y\}$
  end if
end for
return S

Algorithm 3 TRANSITION2

Transition2(S)

1: $S' \leftarrow \emptyset$
2: **for** all $Y \in Var(\Delta)$ **do**
3: **if** $\exists X \in S$ and a rule $X \xrightarrow{a} \alpha \in \Delta$ with $Y \in Var(\alpha)$, $Length(\alpha) \geq 2$ and $Y \notin Tail(\alpha)$ **then**
4: $S' \leftarrow S' \cup \{Y\}$
5: **end if**
6: **end for**
7: **return** S'

3. 算法的最后一步，我们再次迭代调用函数 *Transition1(S)*，它将生成所有包含之前性质的变量的集合。然后我们可以检测变量 X 是否还在所得的集合中。

   如果结果是肯定的，那么变量 X 是一个增长变量。

另一方面，如果一个变量是增长的，那么由增长变量的定义4.4，由于我们检测了所有变量，所以它的增长性一定能被该算法检测出来。

5.2 时间复杂度分析

在引理5.2中，我们给出了算法的时间复杂度。

引理 5.2 (算法时间复杂度 Complexity of the Algorithm)**.** *该判定算法1 GROW 的时间复杂度为 $\mathcal{O}(n^3 + mn)$，其中 n 为输入的规则的个数，m 为其中最长规则的长度。*

证明. 假设输入的进程 (α, Δ) 中，一共有 n 条规则，其中最长规则的长度为 m。

那么最多有 n 个互不相同的进程变量。

1. 对于每条规则 $X \xrightarrow{a} \alpha \in \Delta$ 中的进程表达式 α，我们首先计算 $Tail(\alpha)$。这一步的时间复杂度为 $\mathcal{O}(mn)$。因为最多只有个 n 集合，对于每个集合，我们最多需要迭代 m 次。

2. 同样，我们也可以对于每个变量，对两个 *TRANSITION* 函数进行预处理。这一步，我们个一对每个变量建立一个表，保存满足条件的变量。这一步计算的复杂度为 $\mathcal{O}(mn)$，因为我们一共需要处理最多 n 条规则，长度最多为 m。
3. 接下来我们继续对两个 *TRANSITION* 函数进行简单的分析。最大的循环次数，对于任何变量，做多有 n 条不同的规则，所以最多为 n。对于每条规则，我们对条件的检测需要 $\mathcal{O}(1)$ 的时间，所以这两个函数的时间复杂度为 $\mathcal{O}(n)$。
4. 最后，我们考虑主函数 *GROW*。外层循环需要最多执行 n 次，而内层循环在最多执行 n 次后也会到达不动点 (Fixpoint)。

综上，算法总共的时间复杂度为 $\mathcal{O}(mn) + \mathcal{O}(n^2) \cdot \mathcal{O}(n) = \mathcal{O}(n^3 + mn)$。 □

由定理4.6和引理5.1，引理5.2。我们可以得到 tnPA 的 $\approx_{REG}$ 问题的主要结论。

定理 5.3 (tnPA 的 $\approx_{REG}$)**.** *给定一个 tnPA 进程 (α, Δ)，它的 $\approx_{REG}$ 问题可以在多项式时间内被判定。时间复杂度为 $\mathcal{O}(n^3 + mn)$，其中 n 为输入的规则的个数，m 为其中最长规则的长度。*

由推论4.7我们可知，该结论对于 tnPA 的 $\simeq_{REG}$ 同样成立。我们有推论5.4

推论 5.4 (tnPA 的 $\simeq_{REG}$)**.** *给定一个 tnPA 进程 (α, Δ)，它的 $\simeq_{REG}$ 问题可以在多项式时间内被判定。时间复杂度为 $\mathcal{O}(n^3 + mn)$，其中 n 为输入的规则的个数，m 为其中最长规则的长度。*

有第2.1节中介绍的 PRS 层次中的模型包含关系可知，BPA 和 BPP 都是 PA 的子模型，所以该结论在 tnBPA 和 tnBPP 上都成立。我们有推论5.5

推论 5.5 (tnBPA 和 tnBPP 的 $\approx REG$ 和 $\simeq_{REG}$)**.** *给定一个 tnBPA 或 tnBPP 进程 (α, Δ)，它的 $\approx_{REG}$ 和 $\simeq_{REG}$ 问题均可以在多项式时间内被判定。时间复杂度为 $\mathcal{O}(n^3 + mn)$，其中 n 为输入的规则的个数，m 为其中最长规则的长度。*

第六章 后续问题研究讨论

在第四章和第五章中，我们给出了一个用于判定 tnPA 进程 $\approx_{REG}$ 和 $\simeq_{REG}$ 问题的算法，并分析了它的时间复杂度。同时这个问题对于 tnBPA 和 tnBPP 依然成立。本章我们将对一些我们十分感兴趣，但是并未被解决的 Regularity 问题进行讨论。

有了 tnPA 上的结论，一个非常自然的问题，就是如果将 Totally Normed 条件去掉，那我们是否还有可判定性的结论。即考虑 Normed PA 的 $\approx_{REG}$ 和 $\simeq_{REG}$ 问题。由于 PA 模型的复杂性，所以首先，我们可以分别考虑串行和并行的子模型 nBPA 和 nBPP 上的相关问题。我们考虑这两个模型的另一个原因是，他们的 $\simeq$ 等价性验证问题已经分别被解决 [29, 30]，而且 nBPA 上的 $\simeq_{REG}$ 也有了可判定性结论 [30]。所以我们有理由相信，nBPP 的 $\simeq_{REG}$ 问题有希望在近期内被攻克。

另外，对于 tnPN 上，我们也可以利用本文中的一些直观，在对模型进行一些限定之后，找到 $\approx_{REG}$ 和 $\simeq_{REG}$ 问题的一些思路。

我们在进行无限状态系统验证问题的研究时，另一个十分重要的研究目标就是给定一个问题，我们需要设法证明他的算法复杂性的下界或者不可判定性。我们统称这两类问题为下界问题。我们将在本章最后，讨论 Regularity 相关的下界问题。

6.1 nBPP 的 $\simeq_{REG}$ 问题

本节我们将对 nBPP 的 $\simeq_{REG}$ 的解决思路做一些简单的讨论。我们将在着重讨论 $\not\simeq_{REG}$ 的半可判定性。

Czerwiński, Hofman 和 Lasota 在 2011 年在 [29] 中给出了 nBPP 的 $\simeq$ 等价的可判定性证明。自从有了这一结论之后，nBPP 上的 $\simeq_{REG}$ 问题就成为了等待解决的公开问题。

引理 6.1 (nBPP 的 $\simeq$ 可判定性 nBPP $\simeq$ Decidability)**.** *Normed BPP 的 $\simeq$ 是可判定的。*

这样，我们利用引理6.1和引理3.5可知

推论 6.2 (nBPP 的 $\simeq_{REG}$ 的半可判定性 nBPP $\simeq_{REG}$ Semi-Decidability)**.** *Normed BPP 的 $\simeq$ 是半可判定的。*

有了 [30] 中 nBPA 的 $\simeq$ 和 $\simeq_{REG}$ 问题的可判定性结论，我们有理由相信，nBPP 的 $\simeq_{REG}$ 也是可判定的。有了推论6.2，现在我们只需要证明 $\not\simeq_{REG}$ 是半可判定的。一个最直接的思路是找到一个 $\not\simeq_{REG}$ 的充分必要条件，然后证明这个条件是半可判定的。

为了证明无限性的半可判定，我们要去寻找 nBPP 生成无限个互不 $\simeq$ 等价的状态的原因。

我们知道，我们可以用一个整数向量来表示并行进程，其中每个整数表示对应进程变量上的指数。定一个 nBPP 进程 (α, Δ)，我们将其中能指数上能到达无穷大的进程变量的集合记为 $Inf(\alpha, \Delta)$。我们有引理6.3：

引理 6.3. *给定一个 nBPP 进程 (α, Δ)，$Inf(\alpha, \Delta)$ 是半可判定的。*

证明. 这个集合是一个半线性集合 (Semilinear Set)，证明略。 □

为了简化问题，我们提出引理6.4，从而将研究的注意力集中在 $Inf(\alpha, \Delta)$ 上：

引理 6.4. *给定一个 nBPP 进程 (α, Δ)，如果 $Inf(\alpha, \Delta) = \emptyset$ 那么它是 Regular 的。*

证明. 假设给定进程满足条件，那么它只能在语言上生成有限个不同的进程表达式。 □

如果我们需要在语言上能得到无限个互不相同的进程项，那么我们必须能使得向量中某个整数可以无限增大。另外，BPP 是一个上下文无关的模型，所以我们对于进程的行为每次只需要考虑一个进程变量。我们这里仅仅需要考虑那些可以在指数上无限增大的进程变量，即集合 $Inf(\alpha, \Delta)$ 中进程变量的性质。

然而，这仅仅是一个 $\simeq_{REG}$ 的必要条件，事实表明，并非所有的 $Inf(\alpha, \Delta)$ 中的进程都会产生无限的状态。为了进一步我们所考虑的进程，我们有引理6.5

引理 6.5. *给定一个 nBPP 进程 (α, Δ)，如果存在进程变量 $X \in Inf(\alpha, \Delta)$，而且 $\|X\| > 0$，那么该进程是 $\not\simeq_{REG}$ 的。*

证明. 由引理3.2，该进程的 Norm 显然可以任意增大 (但不是 ∞)，得证。 □

由引理6.3和6.5可知这我们无需考虑那些 $Norm > 0$ 的进程，因为这种情况可以简单的判定，即检测是否 $Inf(\alpha, \Delta)$ 中有 $Norm > 0$ 的进程变量。

我们就将问题简化为，只需要对 $Inf(\alpha, \Delta)$ 中 Norm 为零的进程进行讨论即可。

由这一思路，我们将定义一种稳定 *(Stable)* 的进程变量。

定义 6.1 (稳定的进程变量 Stable Variable)**.** 给定一个 nBPP 进程 (α, Δ)，一个进程变量 $X \in Var(\Delta)$ 是稳定 (Stable) 的如果它满足如下性质:

- 对于所有 $Y \in Reach(X)$，我们有 $\|Y\| = 0$。
- 存在 $k \in \mathbb{N}, k \geq 1$，满足 $X^{k-1} \not\simeq X^k \simeq X^{k+1}$。

我们无需考虑那些 $Norm > 0$ 的进程显然，我们有引理6.6

引理 6.6. *给定一个 nBPP 进程 (α, Δ)，它是 $\simeq_{REG}$ 的，如果所有 $X \in Var(\Delta)$ 都是稳定的进程变量。*

证明. 根据 $\simeq$ 的同余 (Congurence) 性，我们显然可证。 □

这样，对于半可判定性的最后一种情况，我们只需要证明所有 $Inf(\alpha, \Delta)$ 中的进程变量，如果是稳定的，那么必然存在可以 *(*有效计算 (Effectively Computable) 的常数 C，使得 $k < C$。

引理 6.7 (nBPP $\not\simeq_{REG}$ 半可判定算法终止性 Termination of Semi-Decidability Procedure for nBPP $\not\simeq_{REG}$)**.** *给定一个 nBPP 进程 (α, Δ)，如果存在可有效计算 (Effectively Computable) 的常数 C，使得如果 $V \in Var(\Delta)$ 是稳定的进程变量，那么存在满足 $X^{k-1} \not\simeq X^k \simeq X^{k+1}$，且 $k \leq C$。*

则 $nBPP$ 的 $\not\simeq_{REG}$ 是半可判定的。

证明. 我们根据进程变量的 Norm 分情况讨论:

- 给定一个 nPP 进程 (α, Δ)，对于 $Inf(\alpha, \Delta)$ 中的 $Norm > 0$ 的进程，容易判定。

- 对于 $Norm = 0$ 的进程，如果存在这样可以有效计算的 C，那么对于每个进程变量，我们都可以有效得判定其是否是稳定的。假设对于某个进程变量 X，超过 C 仍然找不到对应的 k，那么由假设可知 X 不是稳定的。我们很容易得到无限个互不 $\simeq$ 等价的状态 $X^1 \not\simeq X^2 \not\simeq \ldots$。如果这样，过程终止，该进程是 $\not\simeq_{REG}$ 的。

于是，如果能有效计算这样的 C，那么 $\not\simeq REG$ 是半可判定的。 □

结合推论6.2和引理6.7，我们可知，如果能有效的计算出这样的常数 C，那么 $\simeq_{REG}$ 是可判定的。

从直观上来分析，这个常数 C 表示了单个进程变量组成的表达式能产生不同状态的上限。为了有效计算这个常数，一个思路是给进程变量确定一个合理的序关系，然后从这个序关系上，自底向上地归纳地求解。需要注意的是，为了能够有效地计算，这个界也只能是和输入的进程规则相关的。

尽管迄今为止，有效的计算 C 的方法还没有被发现。但是种种迹象表明，沿着这个思路继续探究，必定能够完全的解决 nBPP 上的 $\simeq_{REG}$ 问题。

6.2 tnPN 的 $\approx_{REG}$ 和 $\simeq_{REG}$ 问题

Petri Net 是一个在程序分析，软件工程和协议验证方面常用的模型。它是 PRS 中的完全并行化的一个进程模型。由 [49] 和推论3.1可知，该模型的 Regularity 问题在仅考虑强互模拟的情况下，是可判定的，相关的 $\approx_{REG}$ 和 $\simeq_{REG}$ 问题都是不可判定的。所以我们在研究这两个问题时，至少需要加上 Normed 限制。

然而，经过第6.1节中的研究，我们知道即使是 nPN 的一个子模型 nBPP，它的 $\simeq_{REG}$ 问题尚未完全解决。

为了研究 PN 下关于包含内部动作的互模拟关系的 Regularity 问题，在第四章和第五章中 PA 工作的启发下，我们很自然的可以想到，可以先试图解决 Totally Normed 限制下的 PN 的 $\approx_{REG}$ 和 $\simeq_{REG}$ 问题。在 tnPA 的模型中，该条件可以有效地将进程所生成的表达式的语言的数量和其状态的数量建立一定的约束关系。在 tnPN 的模型中，我们也可以尝试沿着这个思路进行研究。

我们知道，类似 PN 的并行进程模型都可以用 Vector Addition System(VAS)

进行表示。由 [51, 52]，我们有一个关于 VAS 的有限性 (Boundedness) 的复杂度结论：

引理 6.8 (VAS Boundedness)**.** *给定一个 VAS，判定是否其是否只能达到有限的数值 (Boundedness) 的问题是 EXPSPACE 完全的。*

我们知道由于 PN 和 VAS 是等价的模型，所以我们有引理6.9

引理 6.9. *给定一个 PN 进程，其中是否有一个变量的指数能达到无穷大的问题是 EXPSPACE 完全的。*

我们发现，引理6.9中的条件，很明显是 $\not\sim_{REG}$ 和 $\not\simeq_{REG}$ 的一个必要条件。原因是只有能生成语言上互不相同的无穷多个进程，才可能生成无穷多个互不互模拟的进程。

一个很自然的思路就是我们可以试图去证明这个条件也是 tnPN$\not\sim_{REG}$ 和 $\not\simeq_{REG}$ 的一个充分条件。从而利用引理6.9中的结论来对该问题进行判定。

现在遇到的困难是，由于 tnPN 并没有类似于上下文无关模型 (BPA，BPP 和 PA) 中的 Norm 对加法的同余性，所以我们没有办法直接通过简单的 Norm 的分析，来得到 Norm 可以不停增加的进程项序列。

现在该问题有两种可能的结果：

1. 通过进一步分类 tnPN 中的进程变量，更精细地分析它的行为和规则。可以使用类似与归纳法的思路，从而证明 Norm 增加的序列所需满足的条件。如果成立，那么我猜想这个问题可能最终是 EXPSPACE 完全的。
2. 通过对 [49] 和推论3.1中的归约进一步分析，证明该问题是不可判定的。

我个人更倾向与相信第一个猜想。因为，Totally Normed 的条件从某种程度上限制了进程变量 “消失” 的能力，“消失” 之前至少要留下 “痕迹”。这就为进程表达式的语言上的无穷性和进程状态的语义上的无穷性提供了一个联系，使得 Regularity 的判定问题更加有可能被解决。而且，第一个猜想中的结论也是更有实际意义的，正如第二章中提到的，我们更希望能得到复杂性上完全的结论。

6.3 Regularity 问题的下界

不仅限于无限状态系统的验证问题，对于计算机理论中的所有问题，人们最感兴趣的都是是否能得到一个完备 (Complete) 的结论。

而完备的结论包括两个方面，一个是复杂性上界或者可判定性，前文中大部分篇幅所讨论的都是对这类结论的探究；另一方面则是复杂性下界或者不可判定性，接下来我们就在本文的最后部分对 PRS 的 Regularity 问题的下界进行一些讨论。

全文总结

这里是全文总结内容。

参考文献

[1] Turing A M. On computable numbers, with an application to the Entscheidungsproblem[J]. Proceedings of the London mathematical society, 1936, 42(2):230–265.

[2] Church A. The Calculi of Lambda Conversion.(AM-6)[M].[S.l.]: Princeton University Press, 1985.

[3] Rogers H. Theory of recursive functions and effective computation[M].[S.l.]: McGraw-Hill, 1967.

[4] Milner R. Comunication and Concurrency[M].[S.l.]: Prentice Hall, 1989.

[5] Hopcroft J, Motwani R, Ullman J. Introduction to automata theory, languages and computation[M], Vol. 2.[S.l.]: Addison-Wesley Publishing Company, 1979.

[6] Park D. Concurrency and automata on infinite sequences[J]. Theoretical computer science, 1981:167–183.

[7] Van Glabbeek R, Weijland W. Branching time and abstraction in bisimulation semantics[J]. Journal of the ACM (JACM), 1996, 43(3):555–600.

[8] Mayr R. Process rewrite systems[J]. Information and Computation, 2000, 156(1):264–286.

[9] Baeten J, Bergstra J, Klop J. Decidability of bisimulation equivalence for process generating context-free languages[J]. Journal of the ACM (JACM), 1993, 40(3):653–682.

[10] Burkart O, Caucal D, Moller F, et al. Verifcation on infinite structures[M]//Handbook of Process Algebra.[S.l.]: Cambridge University Press, 2001:545–623.

[11] Kučera A, Jančar P. Equivalence-checking on infinite-state systems: Techniques and results[J]. Theory and Practice of Logic Programming, 2006, 6(3):227–264.

[12] Moller F, Smolka S, Srba J. On the computational complexity of bisimulation, redux[J]. Information and Computation, 2004, 194(2):129–143.

[13] Srba J. Roadmap of infinite results[J]. Bullitin in EATCS, 2002(78):163–175.

[14] Bergstra J A, Klop J W. Algebra of communicating processes with abstraction[J]. Theoretical computer science, 1985, 37:77–121.

[15] Christensen S. Decidability and decomposition in process algebras[J]. 1993.

[16] Baeten J, Weijland W. Process algebra[M].[S.l.]: Cambridge University Press, 1990.

[17] Peterson J L. Petri nets[J]. ACM Computing Surveys (CSUR), 1977, 9(3):223–252.

[18] Christensen S, Hüttel H, Stirling C. Bisimulation equivalence is decidable for all context-free processes[J]. Lecture Notes in Computer Science, 1992, 630:138–147.

[19] Hirshfeld Y, Jerrum M, Moller F. A polynomial algorithm for deciding bisimilarity of normed context-free processes[J]. Theoretical Computer Science, 1996, 158(1):143–159.

[20] Hirshfeld Y, Jerrum M, Moller F. A polynomial-time algorithm for deciding bisimulation equivalence of normed basic parallel processes.[J]. Mathematical Structures in Computer Science, 1996, 6(3):251–259.

[21] Burkart O, Caucal D, Steffen B. An elementary bisimulation decision procedure for arbitrary context-free processes[J]. Mathematical Foundations of Computer Science, 1995, 969:423–433.

[22] Jančar P. Bisimilarity on Basic Process Algebra is in 2-ExpTime (an explicit proof)[J]. Logical Methods in Computer Science, 2012, 9(1).

[23] Jančar P. Strong bisimilarity on basic parallel processes in PSPACE-complete[C]//Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on. .[S.l.]: [s.n.] , 2003:218–227.

[24] Sénizergues G. Decidability of bisimulation equivalence for equational graphs of finite out-degree[C]//Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on. .[S.l.]: [s.n.] , 1998:120–129.

[25] Stirling C. Decidability of bisimulation equivalence for normed pushdown processes[J]. Theoretical Computer Science, 1998, 195(2):113–131.

[26] Hirshfeld Y, Jerrum M. Bisimulation equivalence is decidable for normed process algebra[J]. Automata, Languages and Programming, 1999:72–73.

[27] Jančar P. Undecidability of bisimilarity for Petri nets and some related problems[J]. Theoretical Computer Science, 1995, 148(2):281–301.

[28] Jančar P, Srba J. Undecidability of bisimilarity by defender's forcing[J]. Journal of the ACM (JACM), 2008, 55(1):5.

[29] Czerwiński W, Hofman P, Lasota S. Decidability of branching bisimulation on normed commutative context-free processes[J]. CONCUR 2011–Concurrency Theory, 2011:528–542.

[30] Fu Y. Checking equality and regularity for normed BPA with silent moves[J]. 2013.

[31] Kučera A, Mayr R. Weak bisimilarity between finite-state systems and BPA or normed BPP is decidable in polynomial time[J]. Theoretical Computer Science, 2002, 270(1):677–700.

[32] Fu H. Branching bisimilarity between finite-state systems and BPA or normed BPP is polynomial-time decidable[M]//Programming Languages and Systems.[S.l.]: Springer, 2009:327–342.

[33] Kučera A, Mayr R. On the complexity of semantic equivalences for pushdown automata and BPA[M]//Mathematical Foundations of Computer Science.[S.l.]: Springer, 2002:433–445.

[34] Göller S, Lin A. The complexity of verifying ground tree rewrite systems[C]//Logic in Computer Science (LICS), 2011 26th Annual IEEE Symposium on. .[S.l.]: [s.n.] , 2011:279–288.

[35] Jančar P, Moller F. Checking regular properties of Petri nets[M]//CONCUR'95: Concurrency Theory.[S.l.]: Springer, 1995:348–362.

[36] Burkart O, Caucal D, Steffen B. Bisimulation collapse and the process taxonomy[M]//CONCUR'96: Concurrency Theory.[S.l.]: Springer, 1996:247–262.

[37] Kot M. Regularity of BPP is PSPACE-complete[C]//Proceedings of the 3rd Ph. D. Workshop of Faculty of Electrical Engineering and Computer Science (WOFEX'05). .[S.l.]: [s.n.] , 2005:393–398.

[38] Esparza J, Hansel D, Rossmanith P, et al. Efficient algorithms for model checking pushdown systems[C]//Computer Aided Verification. .[S.l.]: [s.n.] , 2000:232–247.

[39] Kučera A. Regularity is decidable for normed PA processes in polynomial time[C]//Foundations of Software Technology and Theoretical Computer Science. .[S.l.]: [s.n.] , 1996:111–122.

[40] Hüttel H. Silence is golden: Branching bisimilarity is decidable for context-free processes[C]//Computer Aided Verification. .[S.l.]: [s.n.] , 1992:2–12.

[41] Chen H. Decidability of weak bisimilarity for a subset of BPA[J]. Electronic Notes in Theoretical Computer Science, 2008, 212:241–255.

[42] Chen H. More on weak bisimilarity of normed basic parallel processes[J]. Theory and Applications of Models of Computation, 2008:192–203.

[43] Busi N, Gabbrielli M, Zavattaro G. Replication vs. recursive definitions in channel based calculi[M]//Automata, Languages and Programming.[S.l.]: Springer, 2003:133–144.

[44] Van Glabbeek R. The linear time-branching time spectrum[M].[S.l.]: Springer, 1990.

[45] Srba J. Strong bisimilarity and regularity of basic process algebra is PSPACE-hard[M]//Automata, Languages and Programming.[S.l.]: Springer, 2002:716–727.

[46] Mayr R. Weak bisimilarity and regularity of BPA is EXPTIME-hard[C]//Proccedings of the 10th International Workshop on Expressiveness in Concurrency (EXPRESS¡¯03). .[S.l.]: [s.n.] , 2003:160–143.

[47] Srba J. Complexity of weak bisimilarity and regularity for BPA and BPP[J]. Electronic Notes in Theoretical Computer Science, 2003, 39(1):79–93.

[48] StříAbrná J. Hardness results for weak bisimilarity of simple process algebras[J]. Electronic Notes in Theoretical Computer Science, 1998, 18:179–190.

[49] Jančar P, Esparza J. Deciding finiteness of Petri nets up to bisimulation[J]. Automata, Languages and Programming, 1996:478–489.

[50] Srba J. Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard[J]. Lecture Notes in Computer Science, 2002(2285):535–546.

[51] Rackoff C. The covering and boundedness problems for vector addition systems[J]. Theoretical Computer Science, 1978, 6(2):223–231.

[52] Cardoza E, Lipton R, Meyer A R. Exponential space complete problems for Petri nets and commutative semigroups (Preliminary Report)[C]//Proceedings of the eighth annual ACM symposium on Theory of computing. .[S.l.]: [s.n.] , 1976:50–54.

致 谢

感谢所有测试和使用交大硕士学位论文 LaTeX 模板的同学！

感谢那位最先制作出博士学位论文 LaTeX 模板的交大物理系同学！

感谢 William Wang 同学对模板移植做出的巨大贡献！

攻读学位期间发表的学术论文目录

[1] CHEN H, CHAN C T. Acoustic cloaking in three dimensions using acoustic metamaterials[J]. Applied Physics Letters, 2007, 91:183518.

[2] CHEN H, WU B I, ZHANG B, et al. Electromagnetic Wave Interactions with a Metamaterial Cloak[J]. Physical Review Letters, 2007, 99(6):63903.

攻读学位期间参与的项目

[1] 973 项目“XXX”

[2] 自然基金项目“XXX”

[3] 国防项目“XXX”