

Rheinisch-Westfälische Technische Hochschule Aachen

Bachelor's Thesis

Using Stochastic Context Free Grammars
to decide Almost Sure Termination for
Probabilistic Term Rewriting Systems

by
Arion Scheid

Lehr- und Forschungsgebiet Informatik 2
Programming Languages and Verification

1st Examiner: Prof. Dr. Jürgen Giesl
2nd Examiner: Prof. Dr. Erika Ábrahám
Supervisor: Jan-Christoph Kassing, M. Sc.

September 24, 2025

Abstract

Since term rewriting systems are Turing-complete, their termination is generally undecidable. The same holds for probabilistic term rewriting systems with respect to the property of almost sure termination (i.e., termination with probability one).

This thesis investigates a specific class of probabilistic term rewriting systems which we named GNT that is syntactically restricted to be **ground** (i.e., all rewrite rules operate on terms without variables), **non-overlapping** (i.e., each term at each position matches at most one rewrite rule, eliminating ambiguity in rule selection) and **tail-recursive** (i.e., rewrite rules produce no nested recursive subterms). We show that for GNT, the problem of deciding almost sure termination can be reduced to checking consistency in stochastic context-free grammars. Stochastic context-free grammars, originally developed in formal language theory, generate strings via probabilistic derivations. Consistency in this context means that the probability of producing a string is one. Because consistency is efficiently decidable, this reduction yields a practical decision procedure for almost sure termination in the class GNT.

Additionally we show that we can lessen the restriction of groundness to right-groundness (i.e., all rewrite rules produce terms without variables) in the context of innermost almost-sure termination (i.e., almost-sure termination using an innermost rewriting strategy, which rewrites nested reducible expression first).

We implemented all of our results in the Automated Program Verification Environment (AProVE) and were able to return new negative results for some systems out of the Termination Competition (termCOMP) 2025 that AProVE could previously not analyze, i.e., we could prove that they were not almost-surely terminating.

Statement of Originality

I, Arion Scheid, declare that I have authored this thesis independently, that I have not used other than the declared sources/resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

This work has not previously been submitted for a degree or diploma at any university

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor, Jan-Christoph Kassing, who guided me throughout the entire process of this thesis—from the initial idea to the final draft—and consistently offered support along the way. I learned a great deal from him about both my specific topic and academic work in general.

I also want to thank Prof. Jürgen Giesl for the opportunity to write this thesis in his research group, which had always piqued my interest throughout my studies.

Finally, I want to thank my friends and family, who were a constant source of support during the ups and downs of my studies. In particular, I thank my parents for giving me the freedom to pursue my interests freely and for supporting my decisions, and my sister, to whom I could always turn for encouragement and who always had the right words.

Table of Contents

Abstract	ii
Acknowledgments	iv
1 Introduction.....	1
2 Preliminaries	3
2.1 Context Free Grammars	3
2.2 Term Rewriting	12
2.3 Stochastic Context Free Grammars	20
2.4 Probabilistic Term Rewriting	23
3 Deciding Termination	30
3.1 Restricting Term Rewriting Systems	30
3.2 Deciding Right-Ground Termination	30
4 Deciding Almost-Sure Termination	34
4.1 Deciding Consistency	34
4.2 Restricting Probabilistic Term Rewriting Systems	42
4.3 Deciding Almost-Sure Termination in GNT	44
4.4 Removing Restrictions	55
5 Implementation	60
5.1 AProVE.....	60
5.2 Adaptation of the Framework	61
5.3 The New Processors	63
5.4 Evaluation	64
6 Conclusion and Future Work	66
Bibliography	69

1 Introduction

Term rewriting systems (TRSs) are a Turing-complete model of computation based on rewrite rules, closely related to functional programming and rooted in equational logic. They provide a natural framework for reasoning about computation via equations, making them widely applicable in fields such as algebra, software verification, and programming language theory. A central concern in this context is *termination*, i.e., whether every computation eventually halts—a property that is undecidable in general. Nevertheless there are several tools (e.g., AProVE [Gie+17], MuTerm [GL20], NaTT [YKS14], T_1T_2 [Kor+09]) and techniques (e.g., Dependency Pairs (DPs) [AG00], [Gie+06]) that can be employed to automatically analyze termination of TRS.

Since TRS have been thoroughly studied throughout the last decades it makes sense to expand them with probabilistic choices in the hopes of being able to employ some well established analysis techniques. This allows us to model probabilistic programs which are used to describe randomized algorithms and other computational processes involving uncertainty (see e.g., [Gor+14]) with applications in many areas. The new probabilistic extensions of TRS were introduced in [ALY20; BK02; BG05] and name *probabilistic term rewriting systems* (PTRSs). They extend TRSs by introducing probabilistic choices into rewrite steps. This leads to refined termination properties such as *almost-sure termination* (AST), which asks whether a computation terminates with probability one. As termination in the non-probabilistic setting, AST is undecidable for general PTRSs, but there are techniques for analyzing AST anyways, most of which are adaptations of techniques that originated in TRS analysis (e.g., DPs [KG23; KDG24]).

This thesis is concerned with the analysis of AST in PTRSs, but instead of adapting a TRS technique we relate a syntactically restricted class of PTRS to other stochastic models where decision procedures for termination (or related properties) already exist. More specifically, we restrict PTRSs to the following syntactic properties:

- **Ground:** rules contain no variables,
- **Non-overlapping:** two different rewrite rules cannot be simultaneously applicable to the same term at the same position,
- **Tail-recursive:** rewriting steps progress towards terms without nested function symbols.

We name the class of PTRSs that fulfill these restrictions **GNT** and show that it can be related to *stochastic context-free grammars* (SCFGs). SCFGs—well-known from natural language processing, bioinformatics, and statistical modeling—extend traditional grammars by assigning probabilities to productions. Their *consistency* problem, i.e., whether the probability of deriving a terminal string is 1, is decidable via algebraic methods that can also be employed for the analysis of other stochastic models such as *recursive markov chains* (RMCs) and *multi-type branching processes* (MT-BPs) (see [EY09]). We show that via the relation to SCFGs

these methods are also applicable for PTRSs in GNT, drawing a powerful connection between PTRSs and other stochastic models. This also results in a novel decision procedure for AST in GNT. Finally, we pose the questions of whether all restrictions are necessary and establish that groundness can be replaced with **right-groundness** (i.e., all rewrite rules produce terms without variables) for analyzing innermost AST (iAST) (i.e., AST using an innermost rewriting strategy, which rewrites innermost reducible expressions in nested reducible expression first) equivalence, giving us the class **RNT** for which iAST is decidable.

Then we implement our results within the aforementioned Automated Program Verification Environment (AProVE), making use of Microsofts Z3 SMT Solver.

The thesis will be structured as follows: In Chapter 2 we introduce the formal foundations of TRSs, PTRSs, AST, and SCFGs. Chapter 3 then looks at right-ground TRSs and reiterates a well-established decision procedure for termination. We see that it cannot be applied to PTRS and there is a necessity for another technique in the probabilistic setting. Chapter 4 then contains the main contribution of this paper, a decision procedure for GNT based on a reduction to SCFGs and using the existing algebraic SCFG consistency algorithm. Then, Chapter 5 gives an idea of how the practical implementation of the algorithm is realized. Finally, Chapter 6 recapitulates and summarizes all relevant results and gives an outlook on what future work in this field might look like.

2 Preliminaries

This chapter introduces the formal concepts required for the thesis and establishes the groundwork for all subsequent proofs. We begin with *context-free grammars* (Section 2.1), followed by a formalization of *term rewriting systems* (Section 2.2). Building on these foundations, we incorporate stochastic elements to capture uncertainty, resulting in *stochastic context-free grammars* (Section 2.3) and *probabilistic term rewriting systems* (Section 2.4).

2.1 Context Free Grammars

In order for us to introduce formal grammars we must start from the elementary notions of an alphabet, string operations and aggregate operations on sets of strings, and build to arbitrarily complex languages. The structure of this chapter closely follows that of [CBM13], where one can also find a more complete introduction to the topic in general.

To illustrate the concepts along the way we will build up a model for secondary structure analysis of RNA used in practice within the field of bioinformatics. For more information on the use of grammars in bioinformatics see [GR14].

Definition 2.1.1: Alphabet

An *alphabet* Σ is a finite set of elements called *terminal symbols* or *characters*. We denote by $|\Sigma| = n$ for $n \in \mathbb{N}$ the number of elements, i.e. the *cardinality* of an alphabet.

Example 2.1.2: Alphabet

Our alphabet will be $\Sigma_{\text{RNA}} = \{A, U, G, C\}$ for the *nucleotides* (i.e., the basic building blocks of RNA) Adenine, Uracil, Guanine and Cytosine.

Characters can now form *strings*.

Definition 2.1.3: String

A *string* s (also called a *word*) is a finite sequence $s = a_1 \dots a_n$ with $n \in \mathbb{N}$ of characters taken from an alphabet Σ (i.e., $a_i \in \Sigma$ for $1 \leq i \leq n$). The number n of characters in a string s is called *length* (denoted $|s| = n$). A special string is the empty string ε with length 0.

Example 2.1.4: String

The primary structure of RNA is a linear *sequence* of nucleotides. With our alphabet Σ_{RNA} these sequences can be represented as strings (e.g., *AUGC*).

There are many operations that can be used to manipulate strings, some of which will be introduced in the following. First we consider *concatenation*.

Definition 2.1.5: Concatenation

Let $a = a_1...a_n$ and $b = b_1...b_m$ be strings over an alphabet Σ (i.e., $a_i, b_j \in \Sigma$ for $1 \leq i, j \leq n$). Then *concatenation* is defined as $a.b = a_1...a_nb_1...b_m$ where $a.b$ is a new string over Σ . Concatenation is associative (i.e. $a.(b.c) = (a.b).c$ for strings a, b, c) and has the empty string ε as its neutral element (i.e. $a.\varepsilon = a = \varepsilon.a$ for any string a).

The dot may be dropped for readability, resulting in ab instead of $a.b$.

Example 2.1.6: Concatenation

Consider the two strings $s_1 = AUGC$ and $s_2 = GU$ representing individual RNA-sequences. We can concatenate them to $s_1.s_2 = AUGCGU$ representing a single sequence.

Going in the other direction we can also decompose any string into *substrings*.

Definition 2.1.7: Substring

Let a be a string over Σ . Then we can always deconstruct a into *substrings* u, v, w such that $a = uvw$ (u, v, w may all be the empty string here). Given such a decomposition u is called a *prefix*, v an *infix* and w a *suffix* of a . Given strings x and y , we write $x \preceq y$ if x is a substring of y . We denote by $|y|_x$ the number of times a character x occurs in y .

Example 2.1.8: Substring

In our example string $s = AUGC$ we have the substrings $AUGC, AUG, UGC, AU, UG, GC, A, U, G, C$ and ε resulting from all possible decompositions. An example for such a decomposition would be $A.UG.C$ where we can see that A is a prefix, UG an infix and C a suffix. Also we have $|AUGC|_A = 1$.

Note that now we can deconstruct any string to a concatenation of characters, which can be considered *atomic* strings. In addition to decomposition, strings can be expanded via *repetition*.

Definition 2.1.9: Repetition

The *repetition* operation is syntactic sugar for concatenating a string finitely many times with itself. We denote it as a mathematical power, where the exponent tells us how many times the string occurs. So $s^n = \underbrace{s...s}_{n \text{ times}}$ for a string s and a natural number $n \in \mathbb{N}$.

Example 2.1.10: Repetition

For $s = AUGC$ we have $s^2 = AUGCAUGC$.

Strings can now be accumulated into sets called *languages*.

Definition 2.1.11: Formal Language

A *formal language* L is a set of strings over a specified alphabet Σ . Strings s belonging to a language L (denoted $s \in L$) are called *sentences* (of L). The number of sentences in a language is called its *cardinality* and denoted $|L|$, similarly to alphabets.

Example 2.1.12: Formal Language

Nucleotides (i.e., the basic sequence building blocks in Σ_{RNA}) can form pairs with hydrogen bonds. The (stable)^a *base pairs*^b $L_{BP} = \{AU, GC\}$ are an example of a language over Σ_{RNA} . We have $|L_{BP}| = 2$.

^aTheoretically, there is also a less stable, but still common wobble pair, but we will not consider it here.

^bAs concatenation is non-commutative and base pairs have no order, we should include UA and CG . In order for us to keep the example simple, we will neglect this fact.

The same operations we used for strings can be transferred to languages.

Definition 2.1.13: Language Operations

For languages L_1 and L_2 we define their concatenation as $L_1 L_2 = \{ab \mid a \in L_1 \text{ and } b \in L_2\}$. From this, the extension of the repetition operator is straightforward, as $L^n = L^{n-1} L$ for $n > 0$ and $L^0 = \{\varepsilon\}$.

Example 2.1.14: Language Operations

Using the language L_{BP} introduced in 2.1.12, we can formulate a language with sequences consisting of exactly two stable base pairs, denoted by

$$L_{BP}^2 = \{AU AU, AUGC, GCAU, GCGC\}.$$

With the repetition operator, we can construct a new operation, the *Kleene star-operation*.

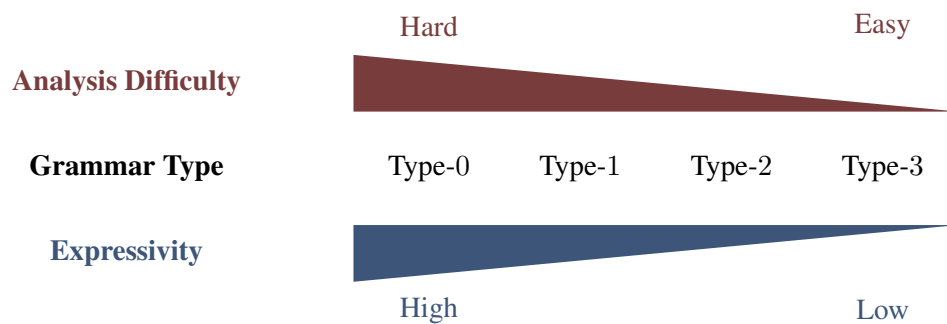
Definition 2.1.15: Closure by concatenation

For any language L we define $L^* = \bigcup_{n \in \mathbb{N}} L^n$ as *Kleene's star-operation* or *closure* by concatenation on L .

Example 2.1.16: Closure by concatenation

Any alphabet Σ can also be regarded as a language, which means that language operations can be applied. Thus, we can denote all words of length n over an alphabet by Σ^n or, more generally, all words over an alphabet by Σ^* . For our alphabet Σ_{RNA} , this means $\Sigma_{\text{RNA}}^* = \{\varepsilon, A, U, G, C, AA, AU, AG, AC, \dots, AAA, \dots\}$ is the countably infinite set representing all possible RNA sequences.

At first glance, the definition of formal languages as mere sets of strings, even though they might represent RNA sequences, may seem abstract and lacking practical application. Indeed, formal language theory is rarely concerned with particular languages, but rather with different formalisms to describe languages. Examples for such formalisms would be *formal grammars* (which will be introduced in the following), *regular expressions* or some form of *automaton* (e.g. finite state automaton, Turing machine). They give structure to the languages they describe, which allows for comparison both in complexity and expressiveness. On this basis, various categories of languages were defined. These are characterized by mathematical and algorithmic properties. One such categorization, and possibly the most commonly used, is the Chomsky hierarchy (see [Cho56]), which differentiates four classes of languages by the types of grammar used to describe them. Type-0 grammars are the most expressive while Type-3 grammars can only describe comparatively simple languages. Unfortunately, the more expressive a grammar becomes, the harder solving key problems gets. This correlation is schematically captured in the following figure.



In this section we will restrict ourselves to the introduction of Type-2 or so-called *context-free grammars* (CFG), which generate the *context-free languages*. The more complex term rewriting, which is as expressive as Type-0 grammars, will then be introduced in the next section (2.2).

Despite not being the most expressive, context-free grammars have widespread practical relevance. Originally invented by linguists for natural languages in the 1950s, today they are an essential part of several fields such as computer science or molecular biology. While in computer science they are the theoretical foundation of all programming languages and make formal specifications and compilation possible, in molecular biology they are a key model for RNA secondary structure prediction (which we will see through our example). For this reason, since the 1960s, many efficient algorithms have been developed to analyze context-free languages.

Definition 2.1.17: Context-free Grammar

A *context-free grammar* (or *type-2 grammar*) G is a tuple (N, T, P, S) where

- N is the *nonterminal alphabet*, a set of characters called *nonterminals* which we will usually denote with upper case symbols
- T is the *terminal alphabet*, a set of characters called *terminals* which we will

usually denote with lower case symbols

- P is a set of *production rules*, given as a binary relation $P \subseteq N \times (N \cup T)^*$
- $S \in N$ is a special nonterminal called the start symbol

A production rule $(A, \alpha) \in P$ is usually denoted $A \rightarrow \alpha$ where A is the *left-hand* and α the *right-hand side* of the rule. Two or more rules $A \rightarrow \alpha_1, \dots, A \rightarrow \alpha_n$ with the same left-hand side can be grouped in $A \rightarrow \alpha_1 \mid \dots \mid \alpha_n$. We call $\alpha_1, \dots, \alpha_n$ the *alternatives* of A . Additionally we require for each nonterminal that there exists at least one production rule containing it on the left-hand side.

Example 2.1.18: Context-free Grammar

Our running example for context-free grammars will be the *Knudsen-Hein* grammar introduced by Bjørn Knudsen and Rasmus Hein in 1999 [KH99], see [GR14] for the formulation used here. We define it as $G_{KH} = (N, T, P, S)$ where we have nonterminals $N = \{S, L, F\}$ (where S is the start symbol), terminals $T = \{s, d\}$, and the following six production rules in P :

$$\begin{array}{lcl} S & \rightarrow & L \mid LS \\ L & \rightarrow & s \mid dFd \\ F & \rightarrow & dFd \mid LS \end{array}$$

The terminals s (resp. d) are placeholders for single (resp. paired, as in L_{BP} from 2.1.12) nucleotides A, U, G, C to simplify the grammar.

Interestingly, CFGs are not unique in their expressivity. *Push-down automata* (PDA), which emerged in automata theory, have the exact same power (i.e., they characterize the exact same language family), but as they are not relevant for this thesis we will not look at them any further. Next we introduce *derivations*, the fundamental mechanic behind the operation of CFGs. They will make use of the production rules of a CFG to rewrite nonterminals into terminal strings.

Definition 2.1.19: Derivation

Let $G = (N, T, P, S)$ be a context-free grammar and $A \rightarrow \alpha$ be a production rule in P . Also let $\beta = \delta A \eta$ and $\gamma = \delta \alpha \eta$ be strings where $\delta, \eta \in (N \cup T)^*$ is an arbitrary *context*, such that γ is obtained from β by replacing the nonterminal A with α according to the production rule. Then we say γ can be *derived* from β (or β derives γ) in G , written $\beta \Rightarrow_G \gamma$ or just $\beta \Rightarrow \gamma$ when G is clear.

Note that the rewriting of nonterminal A to string α happens without taking anything else (i.e., the context) into consideration. This is where CFGs get their name.

Example 2.1.20: Derivation

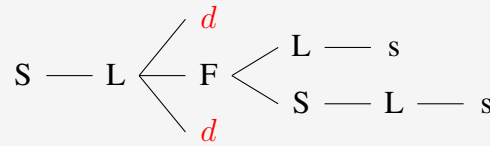
In the CFG G_{KH} introduced in 2.1.18, we can derive a single (unpaired) nucleotide s from start symbol S with

$$S \Rightarrow L \Rightarrow s$$

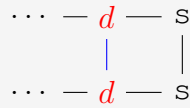
or a so called *small stem-loop* $dssd$, where we will highlight the paired nucleotides d for clarity:

$$S \Rightarrow L \Rightarrow dFd \Rightarrow dLSd \Rightarrow dsSd \Rightarrow dsLd \Rightarrow dssd.$$

A *parse tree* corresponding to this derivation gives us a better understanding of the underlying tree-like structure.



This representation shows the paired d 's that were derived at the same time as siblings (same parent node) on the same height, visually connecting them. In fact, based on this parse tree, we can interpret the derivation semantically as a secondary RNA structure, where the grammatically paired nucleotides form hydrogen bonds (represented with a blue line).



To gain an intuition for this structure, one can imagine an RNA sequence as a strand in space (that is its primary structure). Now this strand will likely not occur as a straight line, but will rather be *folded* into a three-dimensional shape. The way the string is folded is characterized by the base pairs of nucleotides we represent with d 's. These connect three-dimensionally (via hydrogen bonds) and bend the sequence in between. This folding is what we refer to as the secondary structure of the sequence. So essentially our grammar produces a primary structure as an output, and the way it is produced (i.e., the derivation) tells us about its secondary structure.

We can formally express derivations as a binary relation $\Rightarrow_G \subseteq (N \cup T)^* \times (N \cup T)^*$. A sequence (or *chain*) of derivations $\beta_0 \Rightarrow \dots \Rightarrow \beta_n$ can be shortened to $\beta_0 \Rightarrow^n \beta_n$. Often for binary relations we will require the notion of a *transitive* or *reflexive closure*.

Definition 2.1.21: Transitive and Reflexive Closure

Let $R \subseteq A \times A$ be an arbitrary binary relation on some set A . Then we define the transitive closure R^+ as the smallest set such that $R \subseteq R^+$ and if $(a, b) \in R^+$ and $(b, c) \in R^+$ then $(a, c) \in R^+$. We can extend this to a transitive, reflexive closure R^* ,

which is the smallest set such that $R^+ \subseteq R^*$ and $(a, a) \in R^*$ for all $a \in A$.

Finally, with this, we can describe the *language generated by a grammar* G .

Definition 2.1.22: Language generated by Grammar

For a CFG $G = (N, T, P, S)$, the *language generated by* G is defined as $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$, i.e., all terminal words that can be derived from start symbol S in finitely many derivation steps.

Example 2.1.23: Language generated by Grammar

Let us again consider the Knudsen-Hein grammar G_{KH} as introduced in 2.1.18. The language generated by this grammar is $L(G_{KH}) = \{s, ss, dssd, \dots\}$ (i.e., the set of all possible s, d strings, where between two d there is always at least two characters). This means, that $dd \notin L(G_{KH})$, for example.

A language is *context-free* if a context-free grammar exists that generates it. Now given a context-free language and a corresponding CFG, several different problems can be posed.

Problem 2.1.24: Word Problem

Input: A context-free language L and a terminal word w .

Question: Is $w \in L$?

This problem has generally been solved, as can be stated with the following well-known theorem.

Theorem 2.1.25: Decidability of the Word Problem

The word problem for context-free languages is decidable (in polynomial time)^a.

^aA decision procedure is the CYK-algorithm (see [GJ08])

The general idea behind finding out if a word is in a language generated by a CFG is to try to find a derivation for it. This process is called parsing, and it is essential both for our RNA-example and for compilation of programs. Unfortunately, there can be more than a single valid derivation (i.e., parse tree) for any given word.

Problem 2.1.26: Ambiguity Problem

Input: A context-free language L and a CFG G that generates L (i.e., $L(G) = L$).

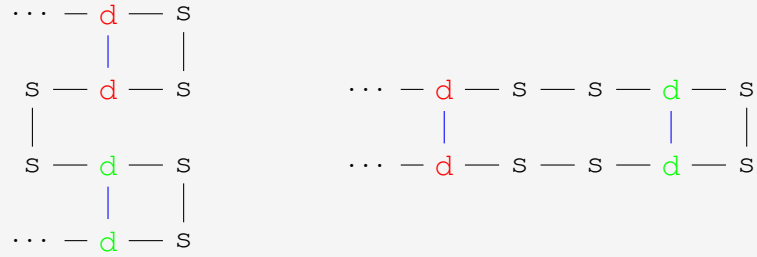
Question: Is there a terminal word w with more than one derivation?

Example 2.1.27: Ambiguity

In G_{KH} (2.1.18) consider the sequence $dssdssdssd$. Then we have two derivations that pair the occurring d 's differently, where the paired d 's will have the same color:

$$\begin{aligned} S &\Rightarrow^* LL \Rightarrow^* dssdL \Rightarrow^* dssdLssd \\ S &\Rightarrow^* L \Rightarrow^* dssLssd \Rightarrow^* dssdssdssd \end{aligned}$$

The first derivation interprets the sequence as subsequent small stem-loops, while the second derivation interprets it as a big loop with a small stem-loop nested inside. Visually we can represent the corresponding secondary RNA-structures as follows:



Just as the word problem, ambiguity has been thoroughly studied, giving us the following result.

Theorem 2.1.28: Decidability of Ambiguity

Ambiguity for context-free languages is undecidable.

For a proof of this result, see [HMu01]. In compilation, the problem of ambiguity is avoided by the use of unambiguous grammars such as $LL(1)$ or $LR(1)$. Solving this issue for grammars that parse RNA sequences will turn out to be more involved and require a probabilistic approach to select between different derivations, which we will present in 2.3. We introduce ambiguity primarily to show, that even for a simple formalism such as CFGs, some problems might be undecidable. Also, in combination with our RNA-example, we use it to motivate adding probabilities to the formalism. Nevertheless, in the context of this paper, ambiguity only plays a minor role. Our primary focus will be on the *termination* of (probabilistic) term rewriting systems. It turns out that due to the difference in applications of CFGs, no specific equivalent property emerged to be widely known and studied. This is partly also due to its simplicity and similarity to other known and more relevant properties (e.g., finiteness, which we talk about later). In section 2.3 we will then see that it is only when we add probabilities and get stochastic CFGs that we will find somewhat equivalent notions of termination emerge for term rewriting and grammars, both derived from probabilistic convergence. Here, we try to take this notion from stochastic CFGs and remove probabilities, formulating a natural notion of termination for CFGs.

Definition 2.1.29: Termination for CFGs

A CFG $G = (N, T, P, S)$ is terminating if there is no infinite derivation starting from S .

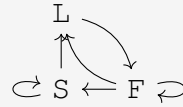
An equivalent definition of termination would be to ask whether a CFG G contains no reachable cycles (i.e., no reachable nonterminal can derive a string containing itself). In practice, we differentiate cycles, since we consider some productive and others unproductive. A cycle is considered productive, if every nonterminal in it can derive a terminal word. An unproductive cycle will contain nonterminals that cannot produce terminal strings, and must therefore result in an infinite derivation. To find any cycles, we will operate on the *dependency graph* of the grammar.

Definition 2.1.30: Dependency Graph

For a CFG $G = (N, T, P, S)$ we define its *dependency graph* $G_D = (V, E)$ with $V = N$ and $(A, \alpha_i) \in E$ for every production $A \rightarrow \alpha_1 \dots \alpha_n \in P$ and every $\alpha_i \in N$ with $i \in \{1, \dots, n\}$.

Example 2.1.31: Dependency Graph

Consider again G_{KH} from 2.1.18. Then we get the following dependency graph:



When answering the common problem of finiteness (i.e., $|L(G)| < \infty$), we build this dependency graph and look for productive cycles in it via a depth-first search (DFS). For our version of the problem, termination, we use the dependency graph and use a DFS starting at the node corresponding to the start symbol of the grammar and check for any reachable cycles. Thus we have a decision procedure and can formulate the following theorem.

Problem 2.1.32: Termination Problem

Input: A context-free grammar G .

Question: Is L terminating?

Theorem 2.1.33: Decidability of Termination

Termination for CFG is decidable.

Later, when deciding SCFG consistency (i.e., the equivalent termination property in the stochastic case), we will again differentiate productive and unproductive cycles and consider the system as non-terminating if there are any unproductive cycles, whereas termination for grammars with productive cycles will depend on the probability of staying in the cycle. Therefore, we can consider this non-probabilistic, decidable version of termination a slightly stricter version.

2.2 Term Rewriting

In this section, we will formally introduce *term rewriting*. It is similar to grammars in that we have a universe of terms, which can somewhat be compared to words over an alphabet, and rewrite rules for them, which are similar to the production rules of a grammar. Unlike words in a grammar that gain external structure through parsing, terms in a rewriting system have an inherent tree-like structure, with variables that can be substituted freely. This added complexity allows term rewriting to express any computable function (i.e., term rewriting is equivalent to *type-0 grammars* or Turing complete), in contrast to the more limited context-free grammars. The introduction to the topic is of course not complete, for a full general version see [BN98]. Once again we will be introducing all concepts alongside an example, this time exploring propositional logic, though due to the increased complexity (especially for the probabilistic version) it will be highly simplified with the unfortunate loss of immediate practical applicability¹.

Our first goal in this section will be to construct terms. To this end we define a *signature* of (uninterpreted) function symbols.

Definition 2.2.1: Signature

A *signature* $\Sigma = \biguplus_{n \in \mathbb{N}} \Sigma_n$ is a union of pairwise disjoint finite sets. An element $f \in \Sigma_n$ is called a function symbol of *arity* $n \in \mathbb{N}$. The elements of Σ_0 are called *constants*. We will always assume that Σ is finite and that $\Sigma_0 \neq \emptyset$.

Example 2.2.2: Signature

Propositional logic is a formal language whose elements are called *propositional formulae*. These formulae will either be *propositional variables* p_1 and p_2 , truth values *true* and *false*, which we will represent as constants $\top$ and $\perp$, or (recursively) composed with *logical connectives*. Such connectives are denoted with a function symbol, where the arity matches the number of formulae it connects. Hence we get our signature $\Sigma_P = \biguplus_{n \in \mathbb{N}} \Sigma_n$ with $\Sigma_0 = \{p_1, p_2, \top, \perp\}$ (constants), $\Sigma_1 = \{\neg\}$ (negation), $\Sigma_2 = \{\wedge\}$ (and) and $\Sigma_n = \emptyset$ for any other $n \geq 3$.

Terms will now be tree-like constructions built with the signature.

Definition 2.2.3: Term

Let Σ be a signature and $\mathcal{V}$ be a non-empty, countable set of *variables* such that $\mathcal{V} \cap \Sigma = \emptyset$. The set of *terms* $\mathcal{T}(\Sigma, \mathcal{V})$ over a signature Σ is inductively defined as

- $\mathcal{V} \subseteq \mathcal{T}(\Sigma, \mathcal{V})$

¹The avid reader who enjoyed the application of grammars in bioinformatics introduced earlier might be interested to know that there are also applications of rewriting in that field, the most closely related of which is RNA graph rewriting introduced in 2023 (see [GG23]), that requires algebraic graph transformations alongside variants of finite state machines and model checking. But all this far surpasses the scope of this thesis.

- For $f \in \Sigma_n$ and $t_1, \dots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$ we have $f(t_1, \dots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$.

For later use, we also define the set of variables $\mathcal{V}(t)$ in a term t inductively as

- $\mathcal{V}(x) = \{x\}$ for $x \in \mathcal{V}$
- $\mathcal{V}(f(t_1, \dots, t_n)) = \bigcup_{1 \leq i \leq n} \mathcal{V}(t_i)$ for $f \in \Sigma_n$ and $t_1, \dots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$.

A term t is called a *ground term* if we have $\mathcal{V}(t) = \emptyset$. We often write $\mathcal{T}(\Sigma)$ instead of $\mathcal{T}(\Sigma, \emptyset)$ for the set of ground terms.

Example 2.2.4: Term

With our signature Σ_P from 2.2.2, and with a set of variables $\mathcal{V} = \{x_1, x_2, \dots\}$, we now have the set of terms $\mathcal{T}(\Sigma_P, \mathcal{V})$ (i.e., the set of propositional formulae) containing elements such as x_1 , $\neg p_1$, and $\neg(x_2 \wedge p_2)$ (where only $\neg p_1$ is a ground term). We will denote the binary connective $\wedge$ in infix notation (i.e., for any two formulae $\varphi, \psi \in \mathcal{T}(\Sigma_P, \mathcal{V})$ we write $\varphi \wedge \psi$ for $\wedge(\varphi, \psi)$) and remove unnecessary brackets.

Additionally we want to be able to access the function symbol at the root of a term.

Definition 2.2.5: Root Symbol of Term

We define $\text{rootymb} : \mathcal{T}(\Sigma, \mathcal{V}) \setminus \mathcal{V} \rightarrow \Sigma$ where

$$\text{rootymb}(t) = f$$

for $t = f(t_1, \dots, t_n)$, $f \in \Sigma_n$, $t_1, \dots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$.

Example 2.2.6: Root Symbol of Term

Consider the term examples from before. We have $\text{rootymb}(\neg p_1) = \neg$ and $\text{rootymb}(\neg(p_1 \wedge p_2)) = \neg$.

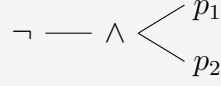
Going with the intuitive understanding of terms as trees, it makes sense to address *subterms* (i.e., subtrees) on different branches.

Definition 2.2.7: Subterm

Let $t, q \in \mathcal{T}(\Sigma, \mathcal{V})$ be two terms. Then q is a *subterm* of t (denoted by $q \trianglelefteq t$) if $q = t$ or $t = f(t_1, \dots, t_n)$ and $q \trianglelefteq t_i$ for some $1 \leq i \leq n$. We call q a *proper* subterm (denoted by $q \triangleleft t$) if $q \trianglelefteq t$ and $q \neq t$.

Example 2.2.8: Subterm

Consider again the formula $\neg(p_1 \wedge p_2)$, which is a ground term as defined in 2.2.4. Its structure can be represented with the following tree.



Now we can easily imagine subterms as subtrees (i.e., trees resulting in cutting a branch to a parent). This gives us the proper subterms $p_1 \wedge p_2$, p_1 and p_2 .

Until now there is no real difference between variables and constants. *Substitutions* extend variables to terms, allowing matching and controlled term replacement. They are fundamental to how term rewriting systems operate.

Definition 2.2.9: Substitution

A *substitution* is a function $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\Sigma, \mathcal{V})$ with $\sigma(x) = x$ for all but finitely many $x \in \mathcal{V}$, which will be called the (finite) *domain of* σ (denoted $Dom(\sigma) := \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$). Thus σ be identified with the finite set $\{x/\sigma(x) \mid x \in Dom(\sigma)\}$. This can be simplified further by writing $x\sigma$ instead of $\sigma(x)$. The set of all substitutions for a signature Σ and a set of variables $\mathcal{V}$ is denoted by $Sub(\Sigma, \mathcal{V})$. Any substitution $\sigma \in Sub(\Sigma, \mathcal{V})$ can also be applied to terms $t = f(t_1, \dots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ as defined by $t\sigma = f(t_1\sigma, \dots, t_n\sigma)$.

A special type of substitution is a *ground* substitution, where every variable is mapped to a ground term (i.e., $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\Sigma)$). Although this technically violates the original definition of $\sigma(x) = x$ for all but finitely many $x \in \mathcal{V}$ for infinite $\mathcal{V}$, we allow this since we will always only be interested in a finite amount of variable instantiations.

Example 2.2.10: Substitution

Consider the terms from 2.2.4 and the ground substitution σ identified with $\{x_1/\neg p_2, x_2/p_1\} \in Sub(\Sigma_P, \mathcal{V})$. Then $x_1\sigma = \neg p_2$, $\neg p_1\sigma = \neg p_1$ and lastly $\neg(x_2 \wedge p_2)\sigma = \neg(p_1 \wedge p_2)$.

Further, as with any other function, substitutions can be composed. This is specifically possible since we have extended their definition to include the domain of all terms. Using this, we can now define different relations between terms.

Definition 2.2.11: Matching and Unification

For terms $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$, we say s *matches* t if there exists a substitution σ such that $s\sigma = t$. s and t are *unifiable* if there exists a substitution σ such that $s\sigma = t\sigma$. Such a substitution is then called a *unifier*. As two terms might have (infinitely) many unifiers,

we introduce a *most general unifier* (mgu) as a unifier σ such that for every other unifier σ' there exists a substitution δ with $\sigma' = \delta \circ \sigma$ (i.e., any other unifier can be derived from the mgu by refining it). It is well-known that for any two unifiable terms a unique mgu can always be found in polynomial time.

Example 2.2.12: Matching and Unification

Consider the term $s = \neg(x_2 \wedge p_2)$ and substitution $\sigma = \{x_1/\neg p_2, x_2/p_1\}$ from 2.2.10. Also consider the term $t = \neg(p_1 \wedge p_2)$ we used in 2.2.8. Then we can easily see that s matches t (i.e., $s\sigma = t$). Since σ acts as an identity on t , we also see that σ is a unifier for s and t (i.e., $s\sigma = t\sigma$), though it is not the most general. The most general unifier of s and t is $\sigma' = \{x_2/p_1\}$, and we see that with $\delta = \{x_1/\neg p_1\}$ we get $\delta \circ \sigma' = \sigma$.

To get to full rewriting, we now want a formal way of addressing the specific subterms of a term. To this end, we introduce the notion of *positions* within terms. If we go with the intuitive notion of terms being trees, positions are navigate from the root to specific nodes.

Definition 2.2.13: Position

For a term $t \in \mathcal{T}(\Sigma, \mathcal{V})$ we inductively define the set of *positions* $Occ(t) \subseteq \mathbb{N}^*$ denoted as number-strings with

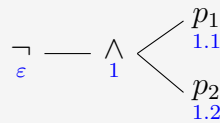
- $\varepsilon \in Occ(t)$
- $\{i.\pi \mid 1 \leq i \leq n, \pi \in Occ(t_i)\} \subseteq Occ(t)$ for $t = f(t_1, \dots, t_n)$ where $f \in \Sigma_n$ and $t_1, \dots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$

ε is also called the *root position* of a term t .

Note that we understand numbers with more than 1 digit as unique characters, which can therefore not be deconstructed into their digits. As a result they are different from the concatenation of single digit numbers into the same string. To avoid confusion, the concatenation operator will be explicitly added when needed.

Example 2.2.14: Position

Once again consider the term $t = \neg(p_1 \wedge p_2)$ from 2.2.8. We number the nodes of the tree representation in blue such that the number at the root of each subtree gives us the position of the corresponding subterm.



We see that t has the set of positions $Occ(t) = \{\varepsilon, 1, 1.1, 1.2\}$.

To structure positions more clearly, we introduce a partial order on $\mathbb{N}^*$ which reflects the idea that one position is an initial segment of another.

Definition 2.2.15: Prefix Order

The prefix order $\leq$ on $\mathbb{N}^* \times \mathbb{N}^*$ is a reflexive binary relation where $(\pi, \tau) \in \leq$ (also written $\pi \leq \tau$) if there exists a $\rho \in \mathbb{N}^*$ such that $\pi.\rho = \tau$.

The prefix order will always implicitly be referred to when the order of positions is mentioned. As the order is only partial, we can also find positions $\pi, \tau \in \mathbb{N}^*$ that are not comparable (i.e., $\pi \not\leq \tau$ and $\tau \not\leq \pi$) which will be called *orthogonal* (denoted by $\pi \perp \tau$).

Example 2.2.16: Prefix Order

Given the positions from 2.2.14, the positions are (partially) ordered with the paths going from the root node. Thus we have $\varepsilon \leq 1$ and $1 \leq 1.1$ for example, whereas we have $1.1 \perp 1.2$, i.e., 1.1 and 1.2 are orthogonal.

Orthogonal positions will become relevant when choosing leftmost innermost terms later, thus we also introduce an order on them.

Definition 2.2.17: Order on Orthogonal Positions

We define the *order on orthogonal positions* as a binary relation $\prec$ on positions $\mathbb{N}^*$ as the smallest set such that for orthogonal positions τ, π we have $\tau \prec \pi$ if $i <_{\mathbb{N}} j$ for the unique $i, j \in \mathbb{N}$ such that $\chi.i \leq \tau$ and $\chi.j \leq \pi$, where χ is the longest common prefix of τ and π .

Example 2.2.18: Order on Orthogonal Positions

Consider the orthogonal positions 1.1 and 1.2 from the example before. We have $1 \leq_{\mathbb{N}} 2$ with $\chi.1 \leq 1.1$ and $\chi.2 \leq 1.2$ for the longest common prefix $\chi = 1$ of 1.1 and 1.2. Thus we have $1.1 \prec 1.2$.

Now using positions we can formally denote specific subterms of a term.

Definition 2.2.19: Position of Subterms

Let $t \in \mathcal{T}(\Sigma, \mathcal{V})$ be a term. Then $t|_{\pi}$ refers to the subterm of t at position $\pi \in \text{Occ}(t)$, inductively defined as

- $t|_{\varepsilon} = t$
- $t|_{\pi} = t_i|_{\pi'}$ for $t = f(t_1, \dots, t_n)$ where $f \in \Sigma_n$, $t_1, \dots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$ and $\pi = i.\pi'$

Example 2.2.20: Position of Subterms

Considering $t = \neg(p_1 \wedge p_2)$ and its tree depiction from 2.2.14 again, we see that we can imagine referencing a subterm as referencing the node label at the root of the corresponding subtree. Thus we have $t|_1 = p_1 \wedge p_2$ or $t|_{1.2} = p_2$.

We can now leverage this notation and define the replacement of subterms by other terms.

Definition 2.2.21: Replacement of Subterms

Let $t, r \in \mathcal{T}(\Sigma, \mathcal{V})$ be terms and $\pi \in \text{Occ}(t)$ a position. Then $t[r]_\pi$ denotes the term that result from replacing the subterm $t|_\pi$ with the new term r in t . Once again this can be expressed inductively, as we have

- $t[r]_\varepsilon = r$
- $t[r]_\pi = t_i[r]_{\pi'}$ for $t = f(t_1, \dots, t_n)$ where $f \in \Sigma_n$, $t_1, \dots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$ and $\pi = i.\pi'$

Example 2.2.22: Replacement of Subterms

With our usual term $t = \neg(p_1 \wedge p_2)$, position $1.1 \in \text{Occ}(t)$ and $s = \neg p_1$ from 2.2.4, we have $t[s]_{1.1} = \neg(\neg p_1 \wedge p_2)$. Given our intuitive understanding of t as a tree, what this does is replace the subtree (i.e., subterm) p_1 whose root node is labeled 1.1 (i.e., at position 1.1) with a new subtree corresponding to s .

Finally, we are ready to define *term rewriting systems*.

Definition 2.2.23: Term Rewriting System

For terms $l, r \in \mathcal{T}(\Sigma, \mathcal{V})$ we call $l \rightarrow r$ a *rewrite rule* if $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ and $l \notin \mathcal{V}$. A *term rewriting system* (TRS) $\mathcal{R}$ is a finite set of rewrite rules.

The TRS $\mathcal{R}$ induces a *rewrite relation* $\rightarrow_{\mathcal{R}} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma, \mathcal{V})$ via

$$s \rightarrow_{\mathcal{R}} t :\Leftrightarrow s|_\pi = l\sigma \text{ and } t = s[r\sigma]_\pi$$

for a position $\pi \in \text{Occ}(s)$, a rule $l \rightarrow r \in \mathcal{R}$, and a substitution $\sigma \in \text{Sub}(\Sigma, \mathcal{V})$. A subterm $s|_\pi$ that is matched by the left-hand side of a rule is called a *redex* (for *reducible expression*).

The two restrictions to rewrite rules made here are common practice, but might not seem immediately obvious. To understand their purpose, we consider problems that arise when removing them. Removing $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ allows rewrite rules, where new variables are produced, that can then be substituted completely freely. This means, that we could, for example, simply substitute the new variables with the term that was just rewritten, resulting in a pointless rewrite cycle. The same holds true if we removed the restriction $l \notin \mathcal{V}$. A single variable on the left-hand side of a rewrite rule could match any term and create a similar rewrite cycle. Analyzing whether

infinite rewrite sequences are possible is a central component of TRS analysis, thus it makes sense to get such trivial cases out of the way, especially since it does not have any impact on expressivity.

Example 2.2.24: Term Rewriting System

We define a term rewriting system $\mathcal{R}$ over $\mathcal{T}(\Sigma_P, \mathcal{V})$ as follows.

$$\begin{array}{ll}
\neg \top \rightarrow \perp & (\top\text{-neg}) \\
\neg \perp \rightarrow \top & (\perp\text{-neg}) \\
\top \wedge \top \rightarrow \top & (\wedge\text{-base}) \\
\perp \wedge x_1 \rightarrow \perp & (\wedge\text{-rec}) \\
x_1 \wedge x_2 \rightarrow x_2 \wedge x_1 & (\wedge\text{-comm})
\end{array}$$

To understand these rules, remember that we introduced $\top$ and $\perp$ as placeholders for *true* and *false* in 2.2.2. Our final goal is to compute the truth value for formulae over propositional variables $\{p_1, p_2\}$. Since for this these variables need to be assigned truth values, we add rewrite rules that represent this with the following TRS $\mathcal{P}$:

$$\begin{array}{ll}
p_1 \rightarrow \top & (p_1\text{-assign}) \\
p_2 \rightarrow \perp & (p_2\text{-assign})
\end{array}$$

Then $\mathcal{R}_{\mathcal{P}} = \mathcal{R} \cup \mathcal{P}$ is our full exemplary rewrite system, that can compute the truth value for basic formulae over the propositional variables p_1 and p_2 (i.e., ground terms over Σ_P), given the corresponding truth assignments $p_1 \mapsto \text{true}$ and $p_2 \mapsto \text{false}$. For example, we can now compute the truth value of $t = \neg(p_1 \wedge p_2)$ from 2.2.4 as follows:

$$\begin{array}{ll}
t \rightarrow_{\mathcal{R}_{\mathcal{P}}} \neg(\top \wedge p_2) & (p_1\text{-assign}) \\
\rightarrow_{\mathcal{R}_{\mathcal{P}}} \neg(\top \wedge \perp) & (p_2\text{-assign}) \\
\rightarrow_{\mathcal{R}_{\mathcal{P}}} \neg(\perp \wedge \top) & (\wedge\text{-comm}) \\
\rightarrow_{\mathcal{R}_{\mathcal{P}}} \neg \perp & (\wedge\text{-rec}) \\
\rightarrow_{\mathcal{R}_{\mathcal{P}}} \top & (\perp\text{-neg})
\end{array}$$

Behind each derivation step is denoted the rule which was used to match the corresponding subterm. The only time this matching requires a non-empty substitution is when applying $(\wedge\text{-comm})$. Here for $s = \neg(\top \wedge \perp)$ we use $\sigma = \{x_1/\top, x_2/\perp\}$ and position $1 \in \text{Occ}(s)$ so that $s|_1 = \top \wedge \perp = (x_1 \wedge x_2)\sigma$. Thus we can rewrite to $t = s[(x_2 \wedge x_1)\sigma]_1 = s[\perp \wedge \top]_1 = \neg(\perp \wedge \top)$.

Just as the production rules of grammars where the motor for derivations, rewrite rules to match terms in the rewriting process are fundamental to the continuation of the rewriting process. If we cannot match any further rules to a term t and our rewriting process therefore halts, we have

reached what we call a *normal form*.

Definition 2.2.25: Normal Form

Let $\mathcal{R}$ be a TRS and $t \in \mathcal{T}(\Sigma, \mathcal{V})$ a term over the same signature. Then t is a *normal form* (w.r.t $\rightarrow_{\mathcal{R}}$, or sometimes just $\mathcal{R}$) if there is no $s \in \mathcal{T}(\Sigma, \mathcal{V})$ with $t \rightarrow_{\mathcal{R}} s$. An element $t' \in \mathcal{T}(\Sigma, \mathcal{V})$ is a normal form *of* t if t' is a normal form and $t \rightarrow_{\mathcal{R}}^* t'$. Here, $\rightarrow_{\mathcal{R}}^*$ denotes the transitive and reflexive closure of $\rightarrow_{\mathcal{R}}$. This means we can also write $t \rightarrow_{\mathcal{R}}^+ r$ if t rewrites to r in *at least* 1 step. A term t is *normalizing* if there exists a normal form t' of t . A term t is *strongly normalizing* if every rewrite sequence starting with t will reach a normal form and is therefore finite. By $\text{NF}_{\rightarrow_{\mathcal{R}}} \subseteq \mathcal{T}(\Sigma, \mathcal{V})$ we will denote the set of all normal forms w.r.t $\rightarrow_{\mathcal{R}}$ (or just $\mathcal{R}$).

Example 2.2.26: Normal Form

With our TRS $\mathcal{R}_{\mathcal{P}}$ from 2.2.24 we have $\text{NF}_{\mathcal{R}_{\mathcal{P}}} = \{\perp, \top\}$. Since we have seen, that $t = \neg(p_1 \wedge p_2) \rightarrow_{\mathcal{R}_{\mathcal{P}}}^* \top$ we know that t is normalizing, but not strongly normalizing. The latter will be explained further in the next example.

By viewing rewriting as a computation, normal forms serve as the analog of final computed results. Termination guarantees that every computation halts in such a state.

Definition 2.2.27: Termination

Let $\mathcal{R}$ be a TRS. We call it *terminating* if there is no infinite rewrite sequence $t_0 \rightarrow_{\mathcal{R}} t_1 \rightarrow_{\mathcal{R}} t_2 \rightarrow_{\mathcal{R}} \dots$. In other words, a TRS is terminating if every rewrite sequence always reaches a normal form within a finite number of steps, which means that every term is strongly normalizing.

Example 2.2.28: Termination

As mentioned in 2.2.26, $t = \neg(p_1 \wedge p_2)$ is not strongly normalizing in $\mathcal{R}_{\mathcal{P}}$. This is because with $(\wedge\text{-comm})$ we can create an infinite rewrite sequence like the following

$$\neg(p_1 \wedge p_2) \rightarrow_{\mathcal{R}_{\mathcal{P}}} \neg(p_2 \wedge p_1) \rightarrow_{\mathcal{R}_{\mathcal{P}}} \neg(p_1 \wedge p_2) \rightarrow_{\mathcal{R}_{\mathcal{P}}} \dots$$

and thus never achieve a normal form. From this we can conclude that $\mathcal{R}_{\mathcal{P}}$ is not terminating.

As before, we can formulate a corresponding problem.

Problem 2.2.29: Termination Problem

Input: A term rewriting system $\mathcal{R}$.

Question: Is $\mathcal{R}$ terminating?

The following theorem is well-known, a full proof can be found in [Der87].

Theorem 2.2.30: Decidability of Termination

The Termination Problem of TRSs is generally undecidable.

This thesis and much of the research in term rewriting therefore is concerned with finding useful and applicable restrictions where termination becomes manageable and can either be decided or approximated.

2.3 Stochastic Context Free Grammars

While with context-free grammars we can express many different languages with underlying structure, they do not provide any mechanism for expressing uncertainty or preference among derivations. But in practical applications such as natural language processing, biological sequence modeling, and uncertain data generation, a stochastic framework naturally arises. Specifically, at the end of 2.1 we noticed that our example, the Knudsen-Hein grammar for RNA secondary structure analysis, was ambiguous. This corresponds to the fact that there can be multiple biologically plausible secondary structures, the likelihood of which can be drastically different. To capture this, *stochastic context-free grammars* (SCFGs) give each derivation a probability (i.e., likelihood) that will allow us to score them and choose the most likely to circumvent ambiguity. This is done by extending CFGs via an assignment of probabilities to production rules, thereby treating derivations as random processes. Since in section 4.1, where we show how SCFG consistency can be determined, we follow [EY09], we will also use their definitions to define SCFGs in this section.

We begin with some probability theory. The key notion required will be a *discrete probability distribution*, which will be the foundation for the probabilistic structure of production rules.

Definition 2.3.1: Discrete Probability Distribution

A *discrete probability distribution* over a countable set X is a function $\mu : X \rightarrow [0, 1]$ such that

$$\sum_{x \in X} \mu(x) = 1$$

Thus, a discrete probability distribution assigns to each element of X a probability, and the total probability mass across all elements is exactly 1.

Example 2.3.2: Discrete Probability Distribution

The classical example when it comes to probabilities is throwing a 6-sided die. This means $X = \{1, 2, 3, 4, 5, 6\}$ is the set of outcomes, and we can formalize a discrete probability distribution μ with $\mu(x) = \frac{1}{6}$ for each $x \in X$.

In a *stochastic context-free grammar*, for each nonterminal, the set of production rules expand-

ing it will form a discrete probability distribution over right-hand sides.

Definition 2.3.3: Stochastic Context-Free Grammar

A *stochastic context-free grammar* (SCFG) G is a tuple (N, T, P, S) where

- N is the *nonterminal alphabet*, a finite set of nonterminal symbols,
- T is the *terminal alphabet*, a finite set of terminal symbols,
- $P \subseteq N \times [0, 1] \times (N \cup T)^*$ is a finite set of *production rules* where for each $A \in N$, the set $X_A = \{(A, p, \alpha) \in P\}$ (i.e., the set of rules with left-hand side A) forms a discrete probability distribution μ_A by assigning to each rule its associated probability $p \in [0, 1]$,
- $S \in N$ is the *start symbol*.

As a shorthand, we will write a rule $(A, p, \alpha) \in P$ as $A \xrightarrow{p} \alpha$ indicating that nonterminal A rewrites to α with probability p .

Example 2.3.4: SCFG

For the rest of this section we will use the probabilistic variant G_{KH}^p of G_{KH} from 2.1.18 with the following rules

$$\begin{array}{lll} S & \xrightarrow{p_1} & L \\ L & \xrightarrow{p_3} & s \\ F & \xrightarrow{p_5} & dFd \end{array} \quad \begin{array}{lll} S & \xrightarrow{p_2} & LS \\ L & \xrightarrow{p_4} & dFd \\ F & \xrightarrow{p_6} & LS \end{array}$$

where we see probabilities as parameters and capture them in a vector $p = (p_1, p_2, p_3, p_4, p_5, p_6)$ with $p_n + p_{n+1} = 1$ for all $n \in \{1, 3, 5\}$.

For simplicity we will additionally restrict derivations to *leftmost derivations*, where we always rewrite the leftmost nonterminal (if there is any).

Definition 2.3.5: Probabilistic Leftmost Derivation

Let $G = (N, T, P, S)$ be a stochastic context-free grammar and let $A \xrightarrow{p} \alpha$ be a production rule in P . Let $\beta = \delta A \eta$ be a string with $\delta \in T^*$ and $\eta \in (N \cup T)^*$. Then the string $\gamma = \delta \alpha \eta$ is obtained from β via a *leftmost derivation* with probability $\mu_A(A \xrightarrow{p} \alpha) = p$ (denoted $\beta \xrightarrow{p}_l \gamma$). Since in the probabilistic context we will always use leftmost derivations we will leave the l out of the notation.

Example 2.3.6: Probabilistic Leftmost Derivation

Consider the string $dLSd$ in G_{KH}^p . Then $dLSd \xrightarrow{p_3} dsSd$ is a possible leftmost derivation. Since rule $L \xrightarrow{p_3} s$ is used, the derivation has probability $\mu_L(L \xrightarrow{p_3} s) = p_3$.

Since we can have several rules for any nonterminal A we will also have just as many possible leftmost derivations to choose from (choice according to the discrete probability distribution μ_A). A derivation is therefore nothing but a sequence of probabilistic choices and we can assign it a probability, the so-called *derivation probability*.

Definition 2.3.7: Derivation Probability

Let $G = (N, T, P, S)$ be a SCFG and $d = (S \xRightarrow{p_1} \beta_1 \xRightarrow{p_2} \beta_2 \xRightarrow{p_3} \dots \xRightarrow{p_n} \beta_n)$ a finite leftmost derivation starting in S . The probabilities p_i associated with deriving β_i in each step give us the *derivation probability* $\mu_G(d) = \prod_{i \in \{1, \dots, n\}} p_i$. Again, for the sake of readability, G will not be denoted if it is clear from the context.

Example 2.3.8: Derivation Probability

As in the non-probabilistic case, consider the string $dssd$. Then in G_{KH}^p we have the leftmost derivation

$$d = S \xRightarrow{p_1} L \xRightarrow{p_4} dFd \xRightarrow{p_6} dLSd \xRightarrow{p_3} dsSd \xRightarrow{p_1} dsLd \xRightarrow{p_3} dssd.$$

which has probability $\mu(d) = p_1 \cdot p_4 \cdot p_6 \cdot p_3 \cdot p_1 \cdot p_3 = p_1^2 \cdot p_3^2 \cdot p_4 \cdot p_6$.

For any SCFG G we denote by $L(G)$ the language generated by G (i.e., the set of all derivable terminal words), just as before. But in the probabilistic case, we are not only concerned with whether a word can be derived, but rather with which probability. Thus we extend our probability measure μ in G from sequences to words.

Definition 2.3.9: Word Probability

Let $G = (N, T, P, S)$ be a SCFG and $w \in T^*$ a word. Then $\mu(w)$ is the *probability of w* in G , which we define as

$$\mu(w) = \sum_{d \in D_G(w)} \mu(d)$$

where $D_G(w) = \{S \xRightarrow{p_1} \beta_1 \xRightarrow{p_2} \dots \xRightarrow{p_n} \beta_n \mid \beta_n = w\}$ denotes the set of all finite leftmost derivations yielding w . As usual, G can be omitted and we write $D(w)$ for $D_G(w)$.

Example 2.3.10: Word Probability

We again consider the word $dssd$ we derived with d in 2.3.8. Since there is no other leftmost derivation of it in G_{KH}^p we have $D(dssd) = \{d\}$ and thus $\mu(dssd) = \mu(d) = p_1^2 \cdot p_3^2 \cdot p_4 \cdot p_6$.

This definition implies $\mu(w) = 0$ if $w \notin L(G)$. Furthermore we can consider the probability for sets of words, by summing up the probabilities of all words in the set. A particularly interesting probability is $\mu(L(G)) = \sum_{w \in L(G)} \mu(w)$, that of the entire language $L(G)$.

Definition 2.3.11: Consistency

A SCFG $G = (N, T, P, S)$ is *consistent* if $\mu(L(G)) = 1$. In that case, the grammar G defines a proper discrete probability distribution over terminal strings with μ .

We note that if $\mu(L(G)) < 1$ there is some probability mass lost to infinite derivations, which do not yield terminal strings.

Example 2.3.12: Consistency

Given the probabilistic Knudsen-Hein grammar as defined in 2.3.4, the consistency is dependent on the probability parameter vector p , which denotes the rule probabilities. In practice, these parameters will be learned via an algorithm (e.g., Inside-Outside, Viterbi in [GR14]) from training data consisting of annotated RNA structures. Generally, with reasonable assumptions about the probabilities, the resulting grammar will be consistent, but if learned poorly (e.g., via naive estimation from a biased sample) inconsistency can arise. We will look at examples for both cases in section 4.1.

Once again, we formulate a problem concerning this property.

Problem 2.3.13: Consistency Problem

Input: A stochastic context-free grammar G .

Question: Is G consistent?

As mentioned above, SCFGs have been studied extensively, which allows us to formulate the following well-known theorem for this section.

Theorem 2.3.14: Decidability of Consistency

The consistency of SCFGs is decidable.

We will prove this theorem in the upcoming section 4.1.

2.4 Probabilistic Term Rewriting

In this section, we expand upon the definition of TRS and get *probabilistic term rewriting systems* (PTRS), which were introduced in [ALY20; BK02; BG05]. Just as with SCFG, by introducing a notion of uncertainty into our model we can capture a broader variety of properties. PTRS form a useful bridge between computation theory and probabilistic modeling. To formally realize this connection, similarly to adding a discrete probability distribution (as we did with SCFG), we will introduce probabilistic multidistributions. This will elevate term rewriting to the rewriting of multidistributions. Of course, this results in a more complex rewriting process, where probabilistic properties arise. As an example, the former notion of termination (i.e., whether every computation terminates) is expanded with the probabilistic *almost sure*

termination, where we now care about the probabilistic convergence of our rewriting process rather than single, likely irrelevant, infinite rewrite sequences (computations).

Definition 2.4.1: Multiset

A *multiset* can be formally expressed as an ordered pair (A, m) where A is the *universe* or *underlying set* and $m : A \rightarrow \mathbb{N}$ is a function that maps each $a \in A$ to its *multiplicity*. To simplify the notation, we will identify a multiset with a set M which can contain multiple instances of each element $a \in A$, where the number of instances matches the multiplicity of the element. The *support* of a multiset $\text{Supp}(A, m)$ will be the set of elements $a \in A$ with $m(a) > 0$, i.e., the elements occurring in M at least once.

Example 2.4.2: Multiset

A classical example for multisets are prime factors of a number. The underlying set in this case will be the set of prime numbers. Choose number 120 for example, then we can denote its prime factors with a multiset $(\mathbb{P}, m_{120})$ where $m_{120}(2) = 3$, $m_{120}(3) = 1$, $m_{120}(5) = 1$ and $m_{120}(p) = 0$ for any other prime p , which means that $\text{Supp}(\mathbb{P}, m_{120}) = \{2, 3, 5\}$. With our simplified notation, we have $M_{120} = \{2, 2, 2, 3, 5\}$.

Definition 2.4.3: Multi-Distribution

A *multi-distribution* on a non-empty set A is a countable multiset μ of pairs $(p : a)$, where $0 < p \leq 1$ is a probability and $a \in A$ is an element, such that $\sum_{(p:a) \in \mu} p = 1$. The set of all multi-distributions on A is denoted by $\text{Dist}(A)$. The *support* of a multi-distribution μ is defined as the multiset $\text{Supp}(\mu) := \{a \mid (p : a) \in \mu \text{ for some } p\}$. The set of finitely supported multi-distributions is $\text{FDist}(A) := \{\mu \mid \mu \in \text{Dist}(A), \text{Supp}(\mu) \text{ is finite}\}$. Every multi-distribution μ induces a mapping from A to $[0, 1]$ with $\mu(a) = \sum_{(p:a) \in \mu} p$.

Example 2.4.4: Multi-Distribution

Consider $\mu = \{\frac{1}{2} : \top, \frac{1}{2} : \top \wedge \top\} \in \text{FDist}(\mathcal{T}(\Sigma_{\mathcal{P}}, \mathcal{V}))$. We have $\text{Supp}(\mu) = \{\top, \top \wedge \top\}$. The induced mapping of μ can be interpreted as saying that $\top$ and $\top \wedge \top$ will occur with probability $\frac{1}{2}$.

As mentioned in the introduction to this chapter, *probabilistic term rewriting systems* will rewrite such multidistributions on terms instead of just terms. Nevertheless our rewrite system will only be able to produce finitely supported multi-distributions with each rewrite rule.

Definition 2.4.5: Probabilistic Term Rewrite System

A *probabilistic term rewrite rule* is a pair $l \rightarrow \mu \in \mathcal{T}(\Sigma, \mathcal{V}) \times \text{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ such that $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ for any $r \in \text{Supp}(\mu)$ and $l \notin \mathcal{V}$. A *probabilistic term rewriting systems* (PTRS) is a finite set $\mathcal{R}$ of probabilistic term rewrite rules.

Similar to a TRS, the PTRS $\mathcal{R}$ induces a *probabilistic rewrite relation* denoted by $\rightarrow_{\mathcal{R}} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \text{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ via

$$s \rightarrow_{\mathcal{R}} \{p_1 : t_1, \dots, p_k : t_k\} :\Leftrightarrow s|_{\pi} = l\sigma \text{ and } t_j = s[r_j\sigma]_{\pi} \text{ for all } 1 \leq j \leq k,$$

for a position $\pi \in \text{Occ}(s)$, a rule $l \rightarrow \{p_1 : r_1, \dots, p_k : r_k\} \in \mathcal{R}$, and a substitution $\sigma \in \text{Sub}(\Sigma, \mathcal{V})$. A subterm $s|_{\pi}$ that is matched by the left-hand side of a rule is once again called a *redex*. We call $s \rightarrow_{\mathcal{R}} \mu$ an innermost rewrite step (denoted $s \xrightarrow{i}_{\mathcal{R}} \mu$) if all proper subterms of the used redex $l\sigma$ are in normal form w.r.t. $\mathcal{R}$ and a leftmost innermost rewrite step (denoted $s \xrightarrow{li}_{\mathcal{R}} \mu$) if additionally π is leftmost, i.e., there is no redex at a position τ with $\tau \prec \pi$.

We note that a term is in normal form w.r.t. $\xrightarrow{i}_{\mathcal{R}}$ if and only if it is in normal form w.r.t. $\rightarrow_{\mathcal{R}}$, which means we can speak about normal forms w.r.t. $\mathcal{R}$ for both cases.

Example 2.4.6: Probabilistic Term Rewrite System

For this section, we will adapt the term rewriting system $\mathcal{R}_{\top}$ we introduced in 2.2.24 to the probabilistic setting. To keep this simple, we will only consider the rule $\top \wedge \top \rightarrow \top \in \mathcal{R}_{\top}$ and modify it. To this end, note that with this rule we can rewrite any term of the form $\top \wedge (\top \wedge \dots (\top \wedge \top) \dots)$ where $\top$ occurs $n \in \mathbb{N}$ times (which we will denote $\wedge^n(\top)$) to $\top$. In other words, $\wedge^n(\top)$ always normalizes to $\top$. Our resulting new probabilistic term rewriting system $\mathcal{R}_{\top}$ will add some *noise* to the rewriting process, consisting only of the following rule:

$$\wedge^2(\top) \rightarrow \{\tfrac{1}{2} : \top, \tfrac{1}{2} : \wedge^3(\top)\}$$

This means that whenever we want to finish our computation and derive the normal form $\top$, we have a chance of $\frac{1}{2}$ that this computation fails and instead we add another $\top$ to the term.

As in the non-probabilistic case, we want to be able to express rewrite sequences. This is not possible if our rewrite relation only maps single terms to multi-distributions on terms. Thus we need to *lift* the probabilistic rewrite relation to be a rewrite relation between multidistributions which can capture multiple rewrite steps at once.

Definition 2.4.7: Lifting

Let $\mathcal{R}$ be a PTRS and $\rightarrow_{\mathcal{R}} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \text{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ the probabilistic rewrite relation induced by $\mathcal{R}$. The *lifting* $\Rightarrow_{\mathcal{R}} \subseteq \text{FDist}(\mathcal{T}(\Sigma, \mathcal{V})) \times \text{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ is the smallest relation that satisfies

- If $t \in \text{NF}_{\rightarrow_{\mathcal{R}}}$ then $\{1 : t\} \Rightarrow_{\mathcal{R}} \{1 : t\}$,

- If $t \rightarrow_{\mathcal{R}} \mu$ then $\{1 : t\} \Rightarrow_{\mathcal{R}} \mu$,
- If $\mu_i \Rightarrow_{\mathcal{R}} \nu_i, 0 < p_i \leq 1$ and $\sum_{1 \leq i \leq k} p_i = 1$ for all $1 \leq i \leq k$, then $\bigcup_{1 \leq i \leq k} p_i \cdot \mu_i \Rightarrow_{\mathcal{R}} \bigcup_{1 \leq i \leq k} p_i \cdot \nu_i$ where $p \cdot \mu = \{(p \cdot q : t) \mid (q : t) \in \mu\}$ for any multi-distribution μ

A finite $\Rightarrow_{\mathcal{R}}$ -rewrite sequence μ is a finite family $\mu = (\mu_k)_{k \in \{1, \dots, N\}}$ of multi-distributions for some $N \in \mathbb{N}$ such that $\mu_k \Rightarrow_{\mathcal{R}} \mu_{k+1}$ for all $1 \leq k < N$. An infinite $\Rightarrow_{\mathcal{R}}$ -rewrite sequence μ is an infinite countable family $\mu = (\mu_k)_{k \in \mathbb{N}}$ of multi-distributions such that $\mu_k \Rightarrow_{\mathcal{R}} \mu_{k+1}$ for all $k \in \mathbb{N}$. Let $\overset{i}{\Rightarrow}_{\mathcal{R}}$ be the lifting of innermost rewriting and $\overset{li}{\Rightarrow}_{\mathcal{R}}$ be the lifting of leftmost innermost rewriting.

Example 2.4.8: Lifting

Consider the PTRS $\mathcal{R}_{\top}$ as defined in 2.4.6 and let $\Rightarrow_{\mathcal{R}_{\top}}$ be the lifting of $\rightarrow_{\mathcal{R}_{\top}}$. Then, starting from term $\top^2$, we get the following rewrite sequence:

$$\begin{aligned}
\mu_0 &= \{1 : \wedge^2(\top)\} \\
\Rightarrow_{\mathcal{R}_{\top}} \mu_1 &= \{\tfrac{1}{2} : \top, \tfrac{1}{2} : \wedge^3(\top)\} \\
\Rightarrow_{\mathcal{R}_{\top}} \mu_2 &= \{\tfrac{1}{2} : \top, \tfrac{1}{4} : \wedge^2(\top), \tfrac{1}{4} : \wedge^4(\top)\} \\
\Rightarrow_{\mathcal{R}_{\top}} \mu_3 &= \{\tfrac{1}{2} : \top, \tfrac{1}{8} : \top, \tfrac{1}{8} : \wedge^3(\top), \tfrac{1}{8} : \wedge^3(\top), \tfrac{1}{8} : \wedge^5(\top)\}
\end{aligned}$$

Note that even if two terms rewrite to the same term, we leave them separate. If we had a nondeterministic choice of more than one rule for any such reduct they could be rewritten differently, which is not possible if they are summed up. Keeping individual paths separate will make a lot more sense later on when we consider computation trees.

Since by each rule application the multi-distribution on the right-hand side can give us many different rewriting paths, it makes sense to introduce another formalism to capture each individual path separately. *Rewrite sequence trees*, in contrast to rewriting (flat) multi-distributions, give the rewriting process a tree-like structure and separate single rewriting threads both formally and visually. This will also make them better suited for proofs later on.

Definition 2.4.9: Rewrite Sequence Tree

A *rewrite sequence tree* (RST) is a directed, labelled tree $\mathfrak{T} = (V^{\mathfrak{T}}, E^{\mathfrak{T}}, L^{\mathfrak{T}})$ with

- $V^{\mathfrak{T}}$ is a possibly infinite, non-empty set of vertices
- $E^{\mathfrak{T}} \subseteq V^{\mathfrak{T}} \times V^{\mathfrak{T}}$ is the set of directed edges
- $L^{\mathfrak{T}} : V^{\mathfrak{T}} \rightarrow (0, 1] \times \mathcal{T}(\Sigma, \mathcal{V})$ labels every vertex by a probability and a term.

such that the following properties are satisfied:

1. $G^{\mathfrak{T}} = (V^{\mathfrak{T}}, E^{\mathfrak{T}})$ is a finitely branching, directed tree. Let $\text{Leaf}^{\mathfrak{T}}$ be the set of all leaves in $G^{\mathfrak{T}}$ and let $\mathfrak{r}^{\mathfrak{T}}$ be the root node of $G^{\mathfrak{T}}$.

2. For all $x \in V^{\mathfrak{T}}$ with $L^{\mathfrak{T}}(x) = (p_x : t_x)$ and $x E^{\mathfrak{T}} \neq \emptyset$ we have

$$\sum_{y \in x E^{\mathfrak{T}}, L^{\mathfrak{T}}(y) = (p_y : t_y)} p_y = p_x$$

Here, by $x E^{\mathfrak{T}} = \{y \mid (x, y) \in E^{\mathfrak{T}}\}$ we denote the set of all direct successors of x .

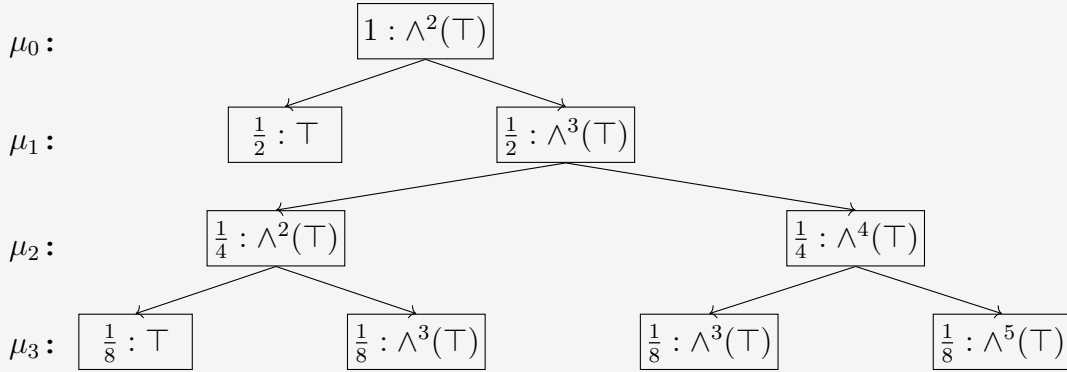
3. $L^{\mathfrak{T}}(\mathfrak{r}) = (1 : t)$ for some $t \in \mathcal{T}(\Sigma, \mathcal{V})$.

4. $t_x \rightarrow_{\mathcal{R}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_n}}{p_x} : t_{y_n}\}$ for $x E = \{y_1, \dots, y_n\}$.

For a node $x \in V^{\mathfrak{T}}$ with $L^{\mathfrak{T}}(x) = (p : t)$, we will also write $p_x^{\mathfrak{T}}$ for p and $t_x^{\mathfrak{T}}$ for t . If the RST is clear from the context, we may omit it in the notations for readability (e.g., we write V for $V^{\mathfrak{T}}$). For a node $x \in V$, we write $d(x)$ for the depth of x , that is, the number of edges in the path from the root to x .

Example 2.4.10: Rewrite Sequence Tree

The multi-distribution rewrite sequence from 2.4.8 can also be represented as the following rewrite sequence tree.



We see that in this representation each edge represents a rewriting step between the labels of the connected nodes. Instead of carrying normal forms with us throughout the entire computation (rewriting sequence) as before, as soon as we reach a normal form, it becomes a leaf in the tree and is not relevant in the next computation steps. As a result, the terms in each layer of the RST only form a subset of the terms in the corresponding multi-distribution for the same computation.

Next we define several important properties of probabilistic term rewriting systems, the most important of which is going to be termination. In contrast to the non-probabilistic setting, we now have multiple different notions of termination. The most similar to the classical notion of termination is *certain termination*.

Definition 2.4.11: Certain Termination

Let $\mathcal{R}$ be a PTRS. We call it *certainly terminating* if and only if there is no infinite $\xrightarrow{i}_{\mathcal{R}}$ -rewrite sequence $(\mu_k)_{k \in \mathbb{N}}$ such that for all $i \in \mathbb{N}$, the support of the multi-distribution μ_i contains at least one term that is not in normal form w.r.t $\mathcal{R}$.

Example 2.4.12: Certain Termination

$\mathcal{R}_{\top}$ from 2.4.6 is not certainly terminating. The rewrite sequence from 2.4.8 could be continued infinitely such that every set contains a term that is not in normal form.

This notion of termination just ignores the probabilities we added and checks for normal termination. If we want to consider the probabilities, we need a more involved notion of termination. To this end, we first formalize the probability of already having reached a normal form (i.e., an end of the computation) within a multi-distribution.

Definition 2.4.13: Termination Probability $|\mu|_{\mathcal{R}}$

Let $\mathcal{R}$ be a PTRS. As before, $\text{NF}_{\mathcal{R}} \subseteq \mathcal{T}(\Sigma, \mathcal{V})$ denotes the set of all normal forms w.r.t $\mathcal{R}$. Then if $\mu \in \text{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$, let $|\mu|_{\mathcal{R}} \in [0, 1]$ be defined as $|\mu|_{\mathcal{R}} = \sum_{(p:t) \in \mu, t \in \text{NF}_{\mathcal{R}}} p$. We will refer to it as the *termination probability* of a multi-distribution μ .

Example 2.4.14: Termination Probability

Consider the last multidistribution $\mu = \{\frac{1}{2} : \top, \frac{1}{8} : \top, \frac{1}{8} : \top^3, \frac{1}{8} : \top^3, \frac{1}{8} : \top^5\}$ of the $\Rightarrow_{\mathcal{R}_{\top}}$ -rewrite sequence from 2.4.8. then $|\mu|_{\mathcal{R}_{\top}} = \frac{1}{2} + \frac{1}{8} = \frac{5}{8}$, since $\top$ is the only normal form w.r.t $\mathcal{R}_{\top}$ in μ .

Lemma 2.4.15: Termination Probability Convergence

Let $\mathcal{R}$ be a PTRS and $(\mu_k)_{k \in \mathbb{N}}$ be an infinite $\Rightarrow_{\mathcal{R}}$ -rewrite sequence. Then we see that

- $\mu_k(t) \leq \mu_{k+1}(t) \leq 1$ for any $k \in \mathbb{N}$ and $t \in \text{NF}_{\mathcal{R}}$,
- $|\mu_k|_{\mathcal{R}} \leq |\mu_{k+1}|_{\mathcal{R}} \leq 1$ for any $k \in \mathbb{N}$.

By the monotone convergence theorem both $\lim_{k \rightarrow \infty} \mu_k(t)$ and $\lim_{k \rightarrow \infty} |\mu_k|_{\mathcal{R}}$ exist. We say $(\mu_k)_{k \in \mathbb{N}}$ converges with probability $q \in [0, 1]$ if $\lim_{k \rightarrow \infty} |\mu_k|_{\mathcal{R}} = q$ (we will also denote this limit by $|\mu_{\infty}|_{\mathcal{R}}$).

Now this form of probabilistic convergence can be leveraged to define a probabilistic variant of termination, namely *almost-sure termination*.

Definition 2.4.16: Almost-Sure Termination

Let $\mathcal{R}$ be a PTRS. We call it *almost-surely terminating* (AST) if and only if every $\Rightarrow_{\mathcal{R}}$ -rewrite sequence converges with probability one, i.e., for every infinite $\Rightarrow_{\mathcal{R}}$ -rewrite sequence $(\mu_k)_{k \in \mathbb{N}}$ we have $|\mu_{\infty}|_{\mathcal{R}} = 1$.

Example 2.4.17: Almost-Sure Termination

Consider again $\mathcal{R}_{\top}$ from 2.4.6 and rewrite sequence $(\mu_n)_{n \in \mathbb{N}}$ from 2.4.8. Then $|\mu_{\infty}|_{\mathcal{R}_{\top}} = \frac{1}{2} + \frac{1}{8} + \cdots = \sum_{n=0}^{\infty} \frac{1}{2^{2n+1}} = \frac{1}{2} \cdot \sum_{n=0}^{\infty} \left(\frac{1}{4}\right)^n = \frac{2}{3}$ via a geometric series argument, thus $|\mu_{\infty}|_{\mathcal{R}_{\top}} < 1$ and $\mathcal{R}_{\top}$ is not AST.

While in this thesis we will restrict ourselves to this notion of termination, it is important to mention there are other types of termination for PTRS (e.g., positive almost-sure termination), which we will not regard in this work.

3 Deciding Termination

This chapter will take a closer look at termination for TRSs. Specifically we will see what known restrictions to TRSs make termination decidable (right-groundness) and see a decision procedure for these restricted TRS.

3.1 Restricting Term Rewriting Systems

Although general termination is undecidable, certain restrictions on rewrite rules allow termination to become decidable. An example of such a restriction, which will also appear in the probabilistic setting later, is *right-groundness*.

Definition 3.1.1: Right-Ground TRS

Let $\mathcal{R}$ be a TRS. We call $\mathcal{R}$ *right-ground* if for every rule $l \rightarrow r \in \mathcal{R}$ the right-hand side r is a ground term (i.e., $\mathcal{V}(r) = \emptyset$).

Example 3.1.2: Right-Ground TRS

Consider again $\mathcal{R}_{\mathcal{P}}$ from 2.2.24 for this section. Since rule $(\wedge\text{-comm})$ (i.e., $x_1 \wedge x_2 \rightarrow x_2 \wedge x_1$) contains variables on the right-hand side, $\mathcal{R}_{\mathcal{P}}$ is not right ground. We replace it by

$$x_1 \wedge \perp \rightarrow \perp \quad (\wedge\text{-rec-2})$$

and call the resulting right-ground TRS $\mathcal{R}_{rg}$.

Intuitively this restriction removes the possibility of reproducing arbitrary terms. This is because any arbitrary term that can be matched with variables in the left-hand-sides of rules can only be rewritten to a fixed finite set of terms. In the following section we will see how this tames the complexity of rewriting systems, such that termination becomes manageable.

3.2 Deciding Right-Ground Termination

Before we can proof the decidability of right-ground termination, we must first introduce the concept of *well-foundedness*.

Definition 3.2.1: Well-foundedness

A binary relation R on a set A is well founded if every $S \subseteq A$ has a minimal element m with respect to R , i.e., there exists an $m \in S$ such that, for every $s \in S$, one does not have $(m, s) \in R^a$. In other words, there are no infinite chains $a_0, a_1, \dots$ where $(a_n, a_{n+1}) \in R$ for every $n \in \mathbb{N}$.

^aNormally this is defined the other way around and we require no infinite descending chains, but for our purposes this inverted definition is better suited.

Example 3.2.2: Ordering of Natural Numbers

A classic example of a well-founded binary relation is the natural ordering $>^{\mathbb{N}}$ on natural numbers $\mathbb{N}$. Every chain (i.e., sequence of numbers getting smaller) is bounded by 0.

We can immediately apply this concept to the rewriting relation.

Lemma 3.2.3: Termination is Well-foundedness

Let $\rightarrow_{\mathcal{R}}$ be the rewrite relation induced by some TRS $\mathcal{R}$. $\mathcal{R}$ is terminating if and only if $\rightarrow_{\mathcal{R}}$ is well-founded.

Proof. Rewriting sequences in $\mathcal{R}$ are chains in its induced rewrite relation, so the Lemma trivially holds. $\square$

The proper subterm relation introduced in 2.2.7 also turns out to have this property.

Lemma 3.2.4: Well-foundedness of Proper Subterm Relation

The proper subterm relation $\triangleleft$ on terms $t \in \mathcal{T}(\Sigma, \mathcal{V})$ is well-founded.

Proof. Consider any term t . Then, any chain of proper subterms of t in $\triangleleft$ is bounded by the finite nesting depth of the term, or, considering the tree representation of t , by the height of tree. $\square$

Example 3.2.5: Proper Subterm Relation

Consider again the formula $\neg(p_1 \wedge p_2)$ we introduced in Example 2.2.8. We have the following sequences resulting from the proper subterm relation:

$$p_1 \triangleleft p_1 \wedge p_2 \triangleleft \neg(p_1 \wedge p_2)$$

$$p_2 \triangleleft p_1 \wedge p_2 \triangleleft \neg(p_1 \wedge p_2)$$

We see that p_1 and p_2 are minimal elements with respect to $\triangleleft$.

Note that we can have several orthogonal minimal elements. Using the well-foundedness of the proper subterm relation, we can prove the following lemma.

Lemma 3.2.6: Right-Ground Termination

Let $\mathcal{R}$ be a right-ground TRS. $\mathcal{R}$ is terminating if and only if there exists no rule $l \rightarrow r \in \mathcal{R}$ such that $r \rightarrow_{\mathcal{R}}^+ r'$ where $r \sqsubseteq r'$ (i.e., r is a subterm of r').

Proof. To prove direction $\Rightarrow$ of the statement, we prove its contraposition. Therefore, assume there exists a rule $l \rightarrow r \in \mathcal{R}$ such that $r \rightarrow_{\mathcal{R}}^+ r'$ where $r \not\leq r'$. Let π be the position in r' such that $r'|_{\pi} = r$. Then we get the infinite rewrite sequence $r \rightarrow_{\mathcal{R}}^+ r' = r'[r]_{\pi} \rightarrow_{\mathcal{R}}^+ r'[r']_{\pi} = r'[r'[r]_{\pi}]_{\pi} \rightarrow_{\mathcal{R}}^+ \dots$

To prove direction $\Leftarrow$ of the statement, assume there is no rule $l \rightarrow r \in \mathcal{R}$ such that $r \rightarrow_{\mathcal{R}}^+ r'$ and $r \not\leq r'$. We will show that $\mathcal{R}$ is terminating with an induction on the size of $\mathcal{R}$.

- $\mathcal{R} = \emptyset$:

$\mathcal{R}$ is clearly terminating, since every term is already in normal form w.r.t. $\mathcal{R}$ and therefore also strongly normalizing.

- $\mathcal{R} \neq \emptyset$:

We prove that $\mathcal{R}$ is terminating by contradiction. To this end, assume that $\mathcal{R}$ is not terminating. Then the set of terms t with infinite rewrite sequences is non-empty. By Lemma 3.2.4 (the well-foundedness of the proper subterm relation), this set must contain a minimal term t_0 with a corresponding infinite rewrite sequence $t_0 \rightarrow_{\mathcal{R}} t_1 \rightarrow_{\mathcal{R}} \dots$. In particular, all proper subterms of t_0 must be strongly normalizing (if there are any), otherwise we would have a contradiction to t_0 being a minimal term in the set of terms with infinite rewrite sequences. Also, at some point in the rewrite sequence we must rewrite a term at its root position ε (i.e., we rewrite the whole term). If this were not the case, then this would mean that a proper subterm of t_0 was sufficient for an infinite rewrite sequence, which would again be a contradiction. Thus there exists some $i \in \mathbb{N}$ such that $t_i = l\sigma$ and $t_{i+1} = r\sigma$ for some rule $l \rightarrow r \in \mathcal{R}$. By the right-groundness of $\mathcal{R}$ the right-hand side of the rule r contains no variables, so we have $t_{i+1} = r$. This means, there is an infinite rewrite sequence $r \rightarrow_{\mathcal{R}} t_{i+2} \rightarrow_{\mathcal{R}} \dots$ starting in r . If the rule $l \rightarrow r$ is not used in this sequence, it means $\mathcal{R} \setminus \{l \rightarrow r\}$ is already terminating, which contradicts the induction hypothesis. Thus $l \rightarrow r$ must be used in the sequence, i.e., we have $r \rightarrow_{\mathcal{R}}^* t_j = t_j[l\sigma']_{\pi} \rightarrow_{\mathcal{R}} t_j[r]_{\pi} = t_{j+1}$ for some $j \geq i + 1$. But then $r \rightarrow_{\mathcal{R}}^+ t_{j+1}$ and $r \not\leq t_{j+1}$, which contradicts the initial assumption.

□

Using this lemma, we get the following important theorem.

Theorem 3.2.7: Right-Ground Termination Decidability

Termination for right-ground TRSs is decidable.

Proof. To prove this theorem we give an algorithm $\mathcal{A}$ that decides right-ground termination. Given a right-ground TRS $\mathcal{R} = \{l_1 \rightarrow r_1, \dots, l_n \rightarrow r_n\}$ it does the following steps:

1. Initialize $T_i = \{r_i\}$ for each $i \in \{1, \dots, n\}$.
2. Construct $T_i = \{t \mid s \in T_i, s \rightarrow_{\mathcal{R}} t\}$ for each $i \in \{1, \dots, n\}$.
3. If $T_i = \emptyset$ holds for all $i \in \{1, \dots, n\}$, stop and return `true`.

4. If $r_i \trianglelefteq t$ for any $i \in \{1, \dots, n\}$ and $t \in T_i$, stop and return `false`.

5. Go to step 2.

Obviously, every step in $\mathcal{A}$ is computable, making it a valid decision procedure. We now prove its correctness, i.e., given a right-ground TRS $\mathcal{R}$, $\mathcal{A}$ returns `true` (resp. `false`) if and only $\mathcal{R}$ terminates (resp. does not terminate).

Assume $\mathcal{R}$ terminates. By Lemma 3.2.6 there does not exist a term t with $r_i \rightarrow_{\mathcal{R}}^+ t$ and $r_i \triangleleft t$. Thus $\mathcal{A}$ can never return `false`. Additionally, termination means that no term r_i can have infinite rewrite sequences. Specifically, each set of reducts T_i that is continuously constructed in step 2. of the algorithm, will eventually not have any further reducts and be set to $\emptyset$. The duration of the algorithm is thus bounded by the length of the longest finite rewrite sequence starting in any r_i . Then, all T_i will be set to $\emptyset$ and $\mathcal{A}$ returns `true`.

Assume $\mathcal{R}$ does not terminate, then by Lemma 3.2.6 there exists an $i \in \{1, \dots, n\}$ with $r_i \rightarrow_{\mathcal{R}}^+ t$ and $r_i \trianglelefteq t$. After finitely many steps $t \in T_i$ must then hold and $\mathcal{A}$ returns `false`. $\square$

Example 3.2.8: Deciding Right-Ground Termination

We consider $\mathcal{R}_{rg}$ as introduced in Example 3.1.2. To check whether it terminates, we employ the algorithm $\mathcal{A}$ used in the proof of Theorem 3.2.7. To reiterate, we have $\mathcal{R}_{rg} = \{\neg\top \rightarrow \perp, \neg\perp \rightarrow \top, \top \wedge \top \rightarrow \top, \perp \wedge x_1 \rightarrow \perp, x_1 \wedge \perp \rightarrow \perp, p_1 \rightarrow \top, p_2 \rightarrow \perp\}$ which we will numerate according to the order in which they are written down here (e.g., $\top \wedge \top \rightarrow \top$ is rule number 3).

1. We have the following initial sets T_i for $i \in \{1, \dots, 7\}$:

$$\begin{aligned} T_1 &= T_4 = T_5 = T_7 = \{\perp\} \\ T_2 &= T_3 = T_6 = \{\top\} \end{aligned}$$

2. After applying step 2., we have:

$$\begin{aligned} T_1 &= T_4 = T_5 = T_7 = \emptyset \\ T_2 &= T_3 = T_6 = \emptyset \end{aligned}$$

3. Since $T_i = \emptyset$ for all $i \in \{1, \dots, 7\}$, $\mathcal{A}$ terminates and returns `true`.

Since $\mathcal{A}$ returns `true` $\mathcal{R}_{rg}$ terminates.

4 Deciding Almost-Sure Termination

This chapter contains the main result of this thesis. We introduce the class GNT of PTRSs, characterized by being ground, nonoverlapping and tail-recursive, and give a reduction from AST to SCFG consistency. Using an existing SCFG consistency algorithm, which we introduce first, we will then define an AST-decision procedure for our restricted PTRS.

4.1 Deciding Consistency

As stated in Theorem 4.1.19, the consistency of SCFG is decidable. In this chapter we will take a closer look at the decision procedure underlying this result. A more general version can be found in [EY09].

The basic idea behind the consistency algorithm is that instead of calculating $\mu(L(G))$ and checking whether it is exactly 1 or smaller, which would involve reasoning about the word probabilities of all derivable words, we try to determine termination probabilities for all the nonterminals that are on right-hand sides of production rules. Then, we check whether those are equal to 1, which can be done with some algebraic tricks as we will see. But before all that, the first step in the algorithm is to clean up the grammar.

Definition 4.1.1: Clean SCFG

A SCFG $G = (N, T, P, S)$ is called *clean* if it contains no *unreachable* nonterminals. A nonterminal X is unreachable, if there is no derivation $S \rightarrow^* \alpha$ with $\alpha \in (N \cup T)^*$ such that X is a substring of α ($X \preceq \alpha$).

It turns out that any SCFG can easily be turned to a clean SCFG, and that it does not influence consistency. To remove unreachable nonterminals one can iteratively expand a set M of nonterminals by all nonterminals that can be reached in one derivation step from the nonterminals in M , starting with M only containing the start symbol. Then as soon as an iteration does not expand M , all nonterminals that are not in M are removed. This is equivalent to a depth first-search as mentioned toward the end of section 2.1 to decide termination (or finiteness). As also mentioned there, unproductive cycles will cause inconsistent grammars, since we will loose probability mass to them. Therefore, to check for inconsistency, we will check for such unproductive cycles first. They are produced by useless nonterminals (i.e., nonterminals that cannot derive any terminal string). To search for useless nonterminals, a similar algorithm to the marking algorithm to check satisfiability of horn clauses can be employed. We first mark all terminal symbols. Then for any fully marked right hand side of a production rule, mark all occurrences of its left hand side. Again, we terminate if no new symbols are marked, and all unmarked nonterminals are useless. If there are any useless nonterminals, then the grammar is inconsistent.

For the next step of the algorithm, assume a clean SCFGs with no useless nonterminals left, also assume the $n \in \mathbb{N}$ nonterminals are named S_i where $i \in \{1, \dots, n\}$. As stated above, our

goal is to determine the termination probabilities x_i of the nonterminals S_i . First, we combine all termination probabilities into a vector $x = (x_1, \dots, x_n)^T$. Then, for each nonterminal S_i we create a polynomial $P_i(x)$ that defines its termination probability in terms of the other nonterminals termination probabilities, specifically the ones it derives in one step, as follows:

$$P_i(x) = \sum_{S_i \xrightarrow{p} \alpha} p \cdot \prod_{S_j \in N} x_j^{|\alpha|_{S_j}}$$

We set $x_i = P_i(x)$ for each $i \in \{1, \dots, n\}$, this means in total we get the polynomial equation system $x = P(x)$ where the termination probabilities form the least fixed-point of the system.

Example 4.1.2: Constructing the System of Polynomial Equations

Consider the Knudsen-Hein grammar introduced in 2.3.4. We then rename S to S_1 , L to S_2 and F to S_3 as per our convention. The resulting grammar looks like the following:

$$\begin{array}{lll} S_1 & \xrightarrow{p_1} & S_2 \\ S_2 & \xrightarrow{p_3} & s \\ S_3 & \xrightarrow{p_5} & dS_3d \end{array} \quad \begin{array}{lll} S_1 & \xrightarrow{p_2} & S_2S_1 \\ S_2 & \xrightarrow{p_4} & dS_3d \\ S_3 & \xrightarrow{p_6} & S_2S_1 \end{array}$$

Since the grammar is already clean, no nonterminals need to be removed. For termination probability vector $x = (x_1, x_2, x_3)^T$ we then get the following polynomial:

$$P(x) = \begin{pmatrix} p_1 \cdot x_2 + p_2 \cdot x_1 \cdot x_2 \\ p_3 + p_4 \cdot x_3 \\ p_5 \cdot x_3 + p_6 \cdot x_1 \cdot x_2 \end{pmatrix} \begin{bmatrix} P_1(x) \\ P_2(x) \\ P_3(x) \end{bmatrix}$$

The termination probabilities are for each nonterminal the least-fixed point of the system $x = P(x)$.

Computing the least fixed-point of $x = P(x)$ is certainly possible, but rather involved. Thus, we instead create something called the *(first) moment matrix* $B(1)$, that aims to capture the expected number of occurrences of nonterminals on the right-hand sides of rules, and analyze whether 1 is the least fixed-point of the system using that. To this end, we look at derivatives of $P_i(x)$ with respect to x_j for $i, j \in \{1, \dots, n\}$. We can capture all such derivatives with a so-called Jacobian matrix $B(x) = \frac{\partial P}{\partial x}$, which is normally defined as follows:

$$B(x) = \begin{pmatrix} \frac{\partial P_1}{\partial x_1} & \dots & \frac{\partial P_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial P_n}{\partial x_1} & \dots & \frac{\partial P_n}{\partial x_n} \end{pmatrix}$$

This means, for row i and column j with $i, j \in \{1, \dots, n\}$ the corresponding entry of the Jacobian matrix is

bian $B(x)_{i,j}$ is defined as

$$B(x)_{i,j} = \frac{\partial P_i}{\partial x_j} = \sum_{S_i \xrightarrow{p} \alpha} p \cdot |\alpha|_{S_j} \cdot x_j^{|\alpha|_{S_j} - 1} \cdot \prod_{\substack{S_k \in N \\ k \neq j}} x_k^{|\alpha|_{S_k}}$$

Example 4.1.3: Computing the Jacobian

Consider $P(x)$ from 4.1.2. Then the corresponding Jacobian $B(X)$ is given by

$$B(x) = \begin{pmatrix} p_2 \cdot x_2 & p_1 + p_2 \cdot x_1 & 0 \\ 0 & 0 & p_4 \\ p_6 \cdot x_2 & p_6 \cdot x_1 & p_5 \end{pmatrix}.$$

The last step to obtaining the moment matrix is to evaluate $B(x)$ at $x = 1$. Since $1^l = 1$ for any $l \in \mathbb{N}$, entries $B(1)_{i,j}$ for $i, j \in \{1, \dots, n\}$ now are of the form

$$B(1)_{i,j} = \sum_{S_i \xrightarrow{p} \alpha} p \cdot |\alpha|_{S_j}$$

Example 4.1.4: The (First) Moment Matrix

We evaluate $B(x)$ from 4.1.3 at $x = 1$ and get

$$B(1) = \begin{pmatrix} p_2 & p_1 + p_2 & 0 \\ 0 & 0 & p_4 \\ p_6 & p_6 & p_5 \end{pmatrix} = \begin{pmatrix} p_2 & 1 & 0 \\ 0 & 0 & p_4 \\ p_6 & p_6 & p_5 \end{pmatrix}.$$

Note how by the definition of our parameter vector p from 2.3.4 we can set $B(1)_{1,2} = p_1 + p_2 = 1$. We summarize our results by formally defining the moment matrix of a SCFG.

Definition 4.1.5: Moment Matrix of a SCFG

Let $G = (N, T, P, S)$ be a clean SCFG and let $N = \{S_1, \dots, S_n\}$ with $S = S_1$ for some $n \in \mathbb{N}$ without loss of generality. Then the (first) moment matrix $B(1) \in \mathbb{Q}_{\geq 0}^{n \times n}$ corresponding to G satisfies that

$$B(1)_{i,j} = \sum_{S_i \xrightarrow{p} \alpha \in P} p \cdot |\alpha|_{S_j}$$

for each $i, j \in \{1, \dots, n\}$.

Now, before moving on with moment matrices, we define some basic algebraic notions.

Definition 4.1.6: Eigenvectors and Eigenvalues

Let A be a $n \times n$ matrix with dimension $n \in \mathbb{N}$. Also let v be a nonzero vector of length n (i.e., a $n \times 1$ matrix). We call v an *eigenvector* of A , if

$$Av = \lambda v$$

for some scalar λ . Then λ is called the corresponding *eigenvalue*. An equivalent formulation is

$$(A - \lambda I)v = 0$$

which will be useful later on.

Next we introduce a *determinant*.

Definition 4.1.7: Determinant

Let A be a $n \times n$ matrix with dimension $n \in \mathbb{N}$. We define the *determinant* of A recursively with

$$\det(A) = \sum_{j=1}^n (-1)^{i+j} A_{i,j} M_{i,j} \text{ for some row } i \in \{1, \dots, n\}$$

where $M_{i,j}$ is defined as the determinant of the $(n-1) \times (n-1)$ matrix that results from A by removing the i -th row and j -th column. For A with dimension 1, we set $M_{i,j} = 1$. This recursive definition is also known as the Laplace (row)^a expansion.

^aWe could also fix the column $j \in \{1, \dots, n\}$ and iterate over rows $i \in \{1, \dots, n\}$ in the sum, which is called the column expansion.

Example 4.1.8: Determinant

Consider a 2×2 matrix $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$. Then we have $\det(A) = 2 \cdot 2 - 1 \cdot 1 = 3$.

With eigenvalues and determinants defined, we can now state a central theorem that will help us find the eigenvalues of square matrices.

Lemma 4.1.9: Determining Eigenvalues with the Characteristic Polynomial

Consider again an $n \times n$ matrix A . The eigenvalues of A are the values λ such that

$$\det(A - \lambda I) = 0$$

where $\det(A - \lambda I)$ is a polynomial of degree n in λ that we call the *characteristic*

polynomial $\mathcal{X}_A$ of A .

Proof. Consider the equation $(A - \lambda I)v = 0$ we used to define eigenvectors and eigenvalues in 4.1.6. It is well known that this equation now has a nonzero solution v if and only if $(A - \lambda I)$ is not invertible which is the case if and only if the $\det(A - \lambda I) = 0$. $\square$

Example 4.1.10: Determining Eigenvalues with the Characteristic Polynomial

Let $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ as before. Then

$$\begin{aligned} \det(A - \lambda I) &= \det \left(\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} - \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} \right) \\ &= \det \begin{pmatrix} 2 - \lambda & 1 \\ 1 & 2 - \lambda \end{pmatrix} \\ &= (2 - \lambda)^2 - 1^2 \\ &= \lambda^2 - 4\lambda + 3 \end{aligned}$$

The eigenvalues are the roots of the characteristic polynomial $\mathcal{X}_A = \lambda^2 - 4\lambda + 3$. Considering its factorized form $(\lambda - 3) \cdot (\lambda - 1)$, the eigenvalues $\lambda_1 = 3$ and $\lambda_2 = 1$ of A are immediately obvious.

Note that with this theorem the question of finding eigenvalues for a square matrix of dimension $n \in \mathbb{N}$ is transformed to finding the roots of a polynomial of degree n . Now that we can compute all the eigenvalues of a matrix, we define important quantities associated with them.

Definition 4.1.11: Spectrum and Spectral Radius

The *spectrum* of a matrix A is the list of its eigenvalues^a. The *spectral radius* $\rho(A)$ of a matrix A is the maximum absolute value of any eigenvalue (i.e., element of the spectrum).

^aRepeated according to their multiplicity, but this is not important to us.

Example 4.1.12: Spectrum and Spectral Radius

Consider again matrix A from 4.1.10. Its spectrum is $\{1, 3\}$ and its spectral radius is $\rho(A) = 3$.

Finally we can express an important connection between the moment matrix of an SCFG, which we defined earlier, and its consistency.

Lemma 4.1.13: Consistency via the Spectral Radius of the Moment Matrix[EY09]

Given a clean SCFG G with no useless nonterminals and its moment matrix $B(1)$, G is consistent if and only if the spectral radius of $B(1)$ is bounded by 1 (i.e., $\rho(B(1)) \leq 1$).

So essentially, if we want to check consistency for an SCFG, we must simply compute its moment matrix (which is possible in polynomial time) and determine its spectral radius. This means, that the decidability of consistency is solely dependent on whether we can decide if the spectral radius of a nonnegative rational matrix is greater than 1. Fortunately, this is indeed decidable, which will result mainly from the following theorems.

Theorem 4.1.14: Perron-Frobenius [LT85]

Let A be a nonnegative $n \times n$ matrix with dimension $n \in \mathbb{N}$. Then the following statements are true:

- The spectral radius $\rho(A)$ is real and nonnegative.
- $\rho(A)$ is an eigenvalue of A called the *Perron root* (or *Perron-Frobenius eigenvalue*).

By this theorem, we know that to check whether $\rho(B(1)) > 1$, we simply have to check whether the characteristic polynomial $\mathcal{X}_{B(1)}$ of $B(1)$ has a real root greater than 1. To this end we use the *existential theory of the reals*.

Definition 4.1.15: Existential Theory of the Reals ($ExTh(\mathbb{R})$)

$ExTh(\mathbb{R})$ is a formal language. A sentence in $ExTh(\mathbb{R})$ is a *prenex*^a form sentence: $\exists x_1, \dots, x_n R(x_1, \dots, x_n)$, where R is a boolean combination of “atomic predicates” of the form $f_i(x)\Delta 0$, where f_i is a multi variate polynomial with rational coefficients over the variables $x = x_1, \dots, x_n$, and Δ is a comparison operator ($=, \neq, \geq, \leq, <, >$).

In other words one can simply take the language of propositional logic introduced earlier and allow for the propositional variables to be replaced by our new atomic predicates $f_i(x)\Delta 0$ to get $ExTh(\mathbb{R})$.

^aQuantifier $\exists$ appears only at the front

Example 4.1.16: Existential Theory of the Reals

Consider the characteristic polynomial $x^2 - 4x + 3$ (we replaced λ with the typical variable x). Then

$$\exists x(x^2 - 4x + 3 = 0)$$

is a sentence in $ExTh(\mathbb{R})$ that claims that the polynomial has real roots. Here $R(x)$ consists of a single atomic predicate $f_1(x)\Delta 0$ where $f_1(x)$ is $x^2 - 4x + 3$ and Δ is $=$.

Now we formalize the problem of checking the validity of such sentences about real numbers.

Problem 4.1.17: $ExTh(\mathbb{R})$ Problem

Input: A sentence φ .

Question: Does φ belong to $ExTh(\mathbb{R})$?

In other words, if a sentence passes the basic syntactic checks, we want to decide whether it is a true statement about the real numbers.

Lemma 4.1.18: Existential Theory of the Reals[Col98]

Let φ be an existential first-order sentence in prenex form in $ExTh(\mathbb{R})$, which uses m distinct polynomials f_i in atomic predicates, over n variables $x_1, \dots, x_n$, each f_i having degree $\leq d$, and with all rational coefficients in φ describable in at most L bits.

There is an algorithm that decides whether φ is true over the real numbers, and that runs in PSPACE and in time: $O((L \log L \log \log L)(md)^{O(n)})$.

This leads us to the final theorem of this chapter.

Theorem 4.1.19: Decidability of Consistency

Consistency for SCFGs is decidable.

Proof. Given a SCFG G , we give a decision procedure $\mathcal{B}$ such that it returns true if and only if G is consistent.

1. Construct the corresponding clean grammar G' of G by removing unreachable nonterminals
2. Check for unproductive cycles, if there are any return `false`
3. Construct the corresponding moment matrix $B(1)$ of G'
4. Compute the characteristic polynomial $\mathcal{X}_{B(1)}$ of $B(1)$
5. Construct the logical sentence $\varphi = \exists x(x > 1 \wedge \mathcal{X}_{B(1)}(x) = 0)$
6. Decide the answer to $\varphi \in ExTh(\mathbb{R})$ and return its negation

All but the last step are trivially computable. The last step is then computable by Lemma 4.1.18.

Next we prove that the procedure is correct. G is consistent if and only if G' is consistent, since removing unreachable nonterminals has no effect on the produced language. Now G' is consistent if and only if it contains no unproductive cycles and $\rho(B(1)) \leq 1$ by Lemma 4.1.13. By Lemma 4.1.9 and Theorem 4.1.14 $\rho(B(1)) \leq 1$ if and only if the maximal real root of $\mathcal{X}_{B(1)}$ is bounded by 1. Then again, this is equivalent to there not being a real root of $\mathcal{X}_{B(1)}$ greater than 1, which is the negation of $\varphi \in ExTh(\mathbb{R})$. $\square$

Example 4.1.20: Deciding Consistency

Consider again G_{KH} as introduced in 2.3.4 and used in Example 4.1.2, 4.1.3 and 4.1.4. We want to determine consistency for G_{KH} using the procedure we introduced in the proof of Theorem 4.1.19. In the mentioned examples, we already computed the corresponding moment matrix $B(1)$ of G_{KH} , so step 1. and 2. are completed.

4. We have

$$\begin{aligned} \mathcal{X}_{B(1)}(\lambda) &= \det(B(1) - \lambda I) \\ &= \det \begin{pmatrix} p_2 - \lambda & 1 & 0 \\ 0 & -\lambda & p_4 \\ p_6 & p_6 & 1 - p_6 - \lambda \end{pmatrix} \end{aligned} \quad (1)$$

$$= -\lambda^3 + (1 + p_2 - p_6)\lambda^2 + (p_2 \cdot p_6 + p_4 \cdot p_6 - p_2)\lambda + p_4 \cdot p_6 \cdot (1 - p_2) \quad (2)$$

(1) $p_1 + p_2 = 1$, $p_5 = 1 - p_6$ by definition of parameter vector p .

(2) Laplace expansion and simplification.

5. Now we construct the sentence

$$\varphi = \exists x(x > 1 \wedge \mathcal{X}_{B(1)}(x) = 0)$$

6. Since the validity of φ in $ExTh(\mathbb{R})$ depends on the parameter vector p we can only verify it with concrete instantiations. Thus let us consider the two following:

- $p_i = \frac{1}{2}$ for $i \in \{1, \dots, 6\}$:
We have $\mathcal{X}_{B(1)}(\lambda) = -\lambda^3 + \lambda^2 + \frac{1}{8}$ which has a maximal real root $\lambda \approx 1.1028$. Thus $\varphi \in ExTh(\mathbb{R})$ is true and we return false. G_{KH} is not consistent for the chosen parameter vector. Intuitively this result makes sense, since now for S and L the probability for terminating and continuing are roughly balanced, but then F shifts that by only allowing infinite recursion or going back to the start.
- $p_2 = p_4 = p_6 = 0$ (we then must have $p_1 = p_3 = p_5 = 1$):
We have $\mathcal{X}_{B(1)}(\lambda) = -\lambda^3 + \lambda^2$ which has maximal real root $\lambda = 1$. Now $\varphi \in ExTh(\mathbb{R})$ is false and we return true. G_{KH} is consistent for the chosen parameter vector. Once again, the intuition is that with this parameter assignment there only is one possible derivation, $S \xrightarrow{1} L \xrightarrow{1} s$.

A visual representation of the resulting characteristic polynomials created with www.desmos.com/calculator can be found in Fig. 4.1. To find more parameters such that the grammar is consistent one can input the general polynomial and play around with the parameters to see when the maximal root remains below 1.

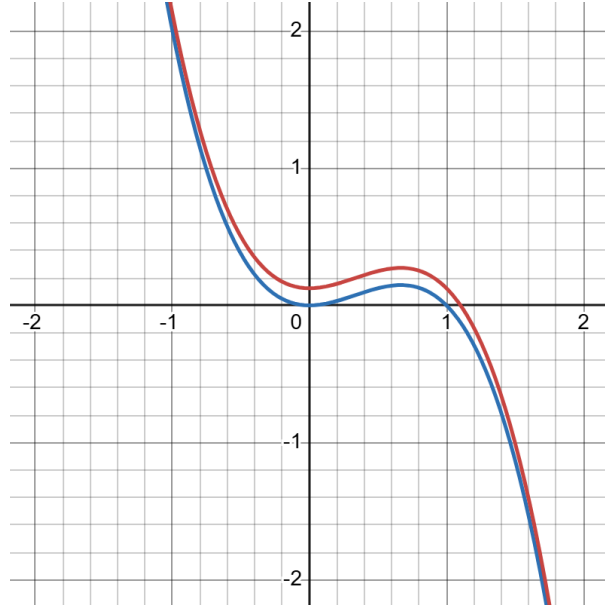


Figure 4.1: Desmos plot of characteristic polynomials

4.2 Restricting Probabilistic Term Rewriting Systems

Since PTRSs are much more expressive than SCFGs, we must restrict PTRSs heavily to enable a reduction. In contrast to the non-probabilistic setting, right-ground PTRSs are still far too expressive. Therefore, we go even further and define (fully) *ground* PTRS.

Definition 4.2.1: Ground PTRS

We call a PTRS $\mathcal{R}$ *ground*, if for every rule $l \rightarrow \mu \in \mathcal{R}$ and $r \in \text{Supp}(\mu)$ we have $\mathcal{V}(l) = \emptyset = \mathcal{V}(r)$, i.e., none of the rules contain any variables.

Example 4.2.2: Ground PTRS

Consider again $\mathcal{R}_\top = \{\wedge^2(\top) \rightarrow \{\frac{1}{2} : \top, \frac{1}{2} : \wedge^3(\top)\}\}$ from 2.4.6. Since $\mathcal{R}_\top$ consists of only one rule which contains no variables $\mathcal{R}_\top$ is ground.

Additionally, since our end goal with the restrictions is to achieve probabilistic rewriting systems that are comparable in their expressivity to stochastic context-free grammars, we want to eliminate nested right-hand sides in rules. More specifically, no right-hand side should contain a term that can wait for a subterm to be rewritten (recursively) before being rewritten itself. If we interpret the rewriting of a left-hand side of a rule as a recursive call to a function with the same name, then what we are doing is disallowing nested recursion. To define this formally, we must first properly define which symbols we consider as recursive calls.

Definition 4.2.3: Defined Function Symbols

The set of *Defined function symbols* of a PTRS $\mathcal{R}$ is denoted by

$$\mathcal{D}_{\mathcal{R}} = \{\text{rootshmb}(l) \mid l \rightarrow \mu \in \mathcal{R}\}.$$

where $\text{rootshmb}(t)$ for a term t is its head as defined in 2.2.5

Example 4.2.4: Defined Function Symbols

Consider $\mathcal{R}_{\top}$ from 2.4.6 and term $t = \wedge^2(\top)$ (syntactic sugar for $\wedge(\top, \top)$). Then t contains the function symbols $\wedge$ and $\top$, where $\wedge$ is a generative function symbol.

With this we can formalize the restriction to non-nested recursion.

Definition 4.2.5: Tail-recursive PTRS

Let $\mathcal{R}$ be a PTRS. A term t is *tail-recursive* if no generative function symbol occurs inside the argument(s) of another generative function symbol. Formally we require that there exist no two positions $p, q \in \text{Occ}(t)$ with $p \neq q$ and $p \leq q$ such that $\text{rootshmb}(t|_p), \text{rootshmb}(t|_q) \in \mathfrak{F}_{\mathcal{R}}$ where $\leq$ is the prefix order from 2.2.15.

We call $\mathcal{R}$ *tail-recursive*, if for every rule $l \rightarrow \mu \in \mathcal{R}$ every $r \in \text{Supp}(\mu)$ is tail-recursive.

Example 4.2.6: Tail-recursive PTRS

Consider again $\mathcal{R}_{\top}$ from 2.4.6 with the single rule $\wedge^2(\top) \rightarrow \{\frac{1}{2} : \top, \frac{1}{2} : \wedge^3(\top)\}$. If we take a closer look at $r = \wedge^3(\top) = \wedge(\top, \wedge(\top, \top))$ we see that the generative function symbol $\wedge$ occurs as the head of r (which is the subterm at the root position, ε), and at position 2 with $\varepsilon \leq 2$, where we have the subterm $\wedge(\top, \top)$ and $\wedge$ is once again the head. By our definition $\mathcal{R}_{\top}$ is not tail-recursive.

This restriction is crucial to be able to connect PTRS to the also tail-recursive SCFGs. The final restriction is the elimination of non-determinism when choosing a rewrite rule. This is achieved by making sure that we cannot have different rewriting rules for terms and their subterms.

Definition 4.2.7: Non-overlapping PTRS

A PTRS $\mathcal{R}$ is *non-overlapping* if there are no two rules $l_1 \rightarrow r_1, l_2 \rightarrow r_2 \in \mathcal{R}$ with renamed variables such that $\mathcal{V}(l_1) \cap \mathcal{V}(l_2) = \emptyset$ where there exists a non-variable position π of l_1 such that $l_1|_{\pi}$ and l_2 are unifiable with a mgu σ . In other words, there can be no two rules applicable to the same term at the same position. If $l_1 \rightarrow r_1 = l_2 \rightarrow r_2$, then we require that $\pi \neq \varepsilon$.

Example 4.2.8: Non-overlapping PTRS

The PTRS $\mathcal{R}_\top$ is trivially non-overlapping, since we can always only choose one rule to rewrite in one way.

Finally we can summarize this section by formally defining the class of PTRSs we will be working with in the next section.

Definition 4.2.9: Ground, Non-overlapping and Tail-recursive PTRS (GNT)

We define **GNT** as the set of all PTRSs $\mathcal{R}$ that are ground, non-overlapping and tail-recursive.

4.3 Deciding Almost-Sure Termination in GNT

In this section we state and prove the central theorem of this thesis, which connects the restricted PTRSs we introduced before in section 4.2 to SCFGs via a reduction. We will see that answering the question of almost sure termination in the former will translate to the question of consistency in the latter. Together with the decision algorithm for consistency introduced in section 4.1 we will then be able to conclude the decidability of AST in GNT. We show this with a sound and complete decision procedure that combines the reduction from PTRSs to SCFGs with the algorithm for SCFG consistency.

Since this section is going to be very theoretical and formal, we will first give an example for the basic reduction we want to achieve.

Example 4.3.1: Reduction

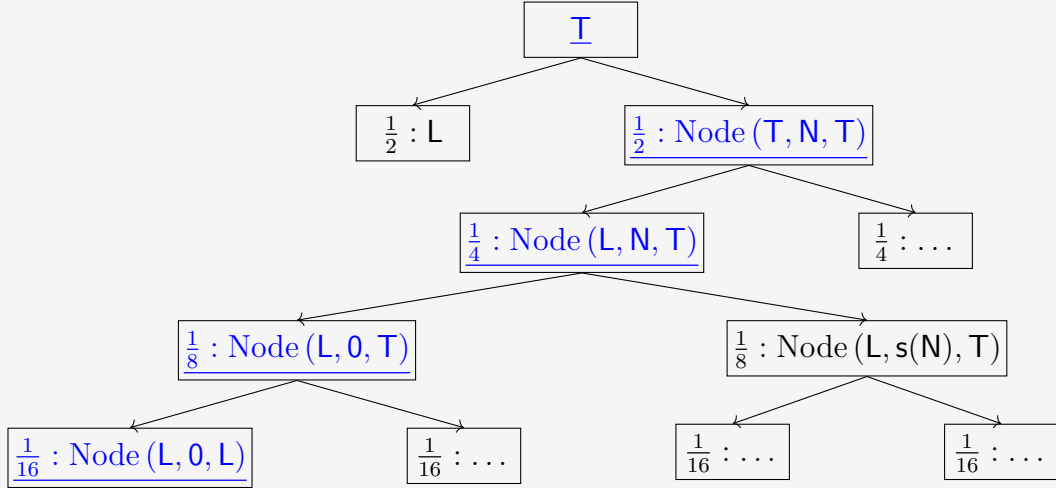
Consider the PTRS $\mathcal{R}$ with rules

$$\begin{aligned} T &\rightarrow \{\tfrac{1}{2} : L, \tfrac{1}{2} : \text{Node}(T, N, T)\} \\ N &\rightarrow \{\tfrac{1}{2} : 0, \tfrac{1}{2} : s(N)\} \end{aligned}$$

Our reduction will turn this into SCFGs with productions

$$\begin{array}{ll} T \xrightarrow{\frac{1}{2}} \varepsilon & T \xrightarrow{\frac{1}{2}} T N T \\ N \xrightarrow{\frac{1}{2}} \varepsilon & N \xrightarrow{\frac{1}{2}} N \end{array}$$

Intuitively our reduction makes use of the fact that for tail-recursive systems the nesting structure which differentiates rewriting from derivations in grammars can essentially be ignored and the terms flattened. The groundness means that we do not have to think about how to deal with variables, which we cannot translate to SCFGs. Consider now a visual representation of the leftmost innermost $\mathcal{R}$ -computation tree starting in term T :



Consider also the following derivation starting in nonterminal T :

$$T \xRightarrow{\frac{1}{2}} TNT \xRightarrow{\frac{1}{2}} NT \xRightarrow{\frac{1}{2}} T \xRightarrow{\frac{1}{2}} \varepsilon$$

Note how this derivation is exactly the same as the path to $\text{Node}(L, 0, L)$ highlighted in **blue** and underlined in the tree, except we removed all structure such as the constructor $\text{Node}(\dots)$ and we replaced normal forms with ε . Formally we will prove there is actually a bijection between paths in the tree and derivations, which will help us see that the convergence probability of the tree (sum of probabilities of nodes in normal form) is the same as the termination probability of the grammar (sum of probability of all derivations of ε).

Before giving the reduction, we want to simplify the rewriting process further, to be able to relate it to the derivation process more easily. More specifically, since we only allow leftmost derivations in SCFGs, we want a similar restriction for our rewriting process. It turns out, that this restriction is not a real restriction when it comes to almost-sure termination of PTRSs in GNT. In order for us to show this, we must first introduce another restriction for PTRS, which is already implicitly included in GNT.

Definition 4.3.2: Linear PTRS

A PTRS $\mathcal{R}$ is *left-linear* (LL) (*right-linear*, RL) if every variable occurs at most once in the left-hand side (right-hand side) of any rule. A PTRS is *linear* if it is both left- and right-linear.

Example 4.3.3: Linear PTRS

Once again we can consider $\mathcal{R}_\top$. Since it is ground, i.e., contains no variables, it is clearly also linear.

We can generalize the example and note that any ground PTRS will also be linear. This allows

us to utilize the following theorems from [KFG24].

Theorem 4.3.4: From iAST to AST [KFG24, Thm. 17]

If a PTRS $\mathcal{R}$ is non-overlapping and linear, then:

$$\mathcal{R} \text{ is AST} \Leftrightarrow \mathcal{R} \text{ is iAST}$$

By extension this also holds if $\mathcal{R}$ is non-overlapping and ground.

Theorem 4.3.5: From liAST to iAST [KFG24, Thm. 22]

If a PTRS $\mathcal{R}$ is non-overlapping, then:

$$\mathcal{R} \text{ is iAST} \Leftrightarrow \mathcal{R} \text{ is liAST}$$

We summarize and apply both theorems to GNT in the following lemma.

Lemma 4.3.6: liAST suffices for GNT

For any PTRS $\mathcal{R} \in \text{GNT}$ we have:

$$\mathcal{R} \text{ is AST} \Leftrightarrow \mathcal{R} \text{ is iAST} \Leftrightarrow \mathcal{R} \text{ is liAST}$$

Proof. Let $\mathcal{R}$ be a PTRS in GNT. If $\mathcal{R}$ is AST is is trivially liAST, since any leftmost innermost rewriting sequence is a rewriting sequence. If $\mathcal{R}$ is liAST, by using the fact that it is ground and non-overlapping and applying Theorem 4.3.5 we get that $\mathcal{R}$ is iAST. Then by applying Theorem 4.3.4 we get that $\mathcal{R}$ is AST. $\square$

This means that when looking at AST for PTRS in GNT in the following, we can restrict ourselves to considering only leftmost innermost derivations without losing generality. Now, for our proofs we will consider rewrite sequence trees instead of multi-distribution rewrite sequences, since their structure is more similar to that of derivations. Thus, we reintroduce the notion of convergence for RSTs.

Definition 4.3.7: RST Convergence

Let $\mathfrak{T}$ be a an RST and $H \subseteq \text{Leaf}$. Let $x_0, x_1, \dots$ be an arbitrary enumeration of H . Then we define $|\mathfrak{T}|_H \in [0, 1]$ by:

$$|\mathfrak{T}|_H := \begin{cases} \sum_{v \in H} p_v, & \text{if } |H| \text{ is finite,} \\ \lim_{i \rightarrow \infty} \sum_{v \in (x_k)_{k \leq i}} p_v, & \text{otherwise.} \end{cases}$$

If we have $|\mathfrak{T}|_H = c$ we say that $\mathfrak{T}$ *converges* w.r.t. H with probability c . If $H = \text{Leaf}$ we simply say that $\mathfrak{T}$ *converges* with probability c . For the sake of simplicity we will allow denoting sums over infinite sets H without the limit, implicitly referring to this

definition.

Now, as established before, for PTRSs in GNT, we will only consider (leftmost) innermost rewrite sequences. Therefore we introduce RSTs restricted to such sequences.

Definition 4.3.8: (Leftmost) Innermost $\mathcal{R}$ -Computation Tree

Let $\mathfrak{T} = (V, E, L)$ be a RST. We call $\mathfrak{T}$ an innermost $\mathcal{R}$ -Computation Tree if the following condition holds:

For every $x \in V$ with $xE = \{y_1, \dots, y_k\} \neq \emptyset$ we have $t_x \xrightarrow{i}_{\mathcal{R}} \left\{ \frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t_{y_k} \right\}$

To restrict the tree to a *Leftmost Innermost $\mathcal{R}$ -Computation Tree* we simply replace the innermost (i) rewriting step by a leftmost innermost (li) rewriting step.

Using such innermost $\mathcal{R}$ -computation trees we can now redefine (leftmost) innermost almost-sure termination for PTRS $\mathcal{R}$.

Definition 4.3.9: (Leftmost) Innermost AST

A PTRS $\mathcal{R}$ is *(leftmost) innermost almost-surely terminating ((l)i)AST* if every (leftmost) innermost $\mathcal{R}$ -computation tree converges with probability one, i.e., for every innermost $\mathcal{R}$ -computation tree $\mathfrak{T}$ we have $|\mathfrak{T}|_{\text{Leaf}} = 1$.

This definition is equivalent to the AST definition with (innermost) multi-distribution rewrite sequences from before, which is a known result that will therefore not be proven here.

The following lemma captures some important properties for ground, tail-recursive PTRS.

Lemma 4.3.10: Properties of ground, tail-recursive PTRS

Let $\mathcal{R}$ be a ground, tail-recursive PTRS and let t be a ground, tail-recursive term. The following properties then hold:

1. The positions of reducible expressions

$$\text{redex}(t) = \{\pi \in \text{Occ}(t) \mid l \rightarrow \mu \in \mathcal{R}, \sigma \in \text{Sub}(\mathcal{V}, \Sigma) : t|_{\pi} = l\sigma\}$$

are orthogonal and can be ordered in a chain

$$\pi_{\min} \prec \dots \prec \pi_{\max}$$

w.r.t. $\prec$. Then $t|_{\pi_{\min}}$ is the leftmost innermost redex of t and the lhs of a rule.

2. Any leftmost innermost rewrite step $t \xrightarrow{li}_{\mathcal{R}} \mu' = \{p_1 : t_1, \dots, p_n : t_n\}$ uses a rule of the form $t|_{\pi_{\min}} \rightarrow \mu = \{p_1 : r_1, \dots, p_n : r_n\} \in \mathcal{R}$ where $t_i = t[r_i]_{\pi_{\min}}$ such that

the following holds for $1 \leq i \leq n$:

Let

$$\pi_1 \prec \dots \prec \pi_m \text{ for } \pi_j \in \text{redex}(t) \text{ and } 1 \leq j \leq m,$$

and

$$\tau_1 \prec \dots \prec \tau_k \text{ for } \tau_l \in \text{redex}(r_i) \text{ and } 1 \leq l \leq k,$$

then

$$\pi_1.\tau_1 \prec \dots \prec \pi_1.\tau_k \prec \pi_2 \prec \dots \prec \pi_m$$

are the elements of $\text{redex}(t_i)$

Proof. We first prove part 1. independently of 2., so that we can then use it to prove 2..

1. Assume we had $\pi, \pi' \in \text{redex}(t)$ with $\pi \neq \pi'$ that are not orthogonal, i.e., $\pi \leq \pi'$ or $\pi' \leq \pi$. Let w.l.o.g. $\pi \leq \pi'$. By definition of $\text{redex}(t)$ and the groundness of $\mathcal{R}$ we have $t|_\pi \rightarrow \mu, t|_{\pi'} \rightarrow \mu' \in \mathcal{R}$. It follows that $\text{rootshmb}(t|_\pi), \text{rootshmb}(t|_{\pi'}) \in \mathcal{D}_{\mathcal{R}}$ by definition. With our assumption we thus have a contradiction to the tail-recursiveness of t , meaning that the positions in $\text{red}(t)$ must be orthogonal. Further we can order the elements of $\text{redex}(t)$ in an ascending chain w.r.t. $\prec$, the smallest element of which we label $\pi_{\min}$. To prove this note that any two distinct positions in $\text{redex}(t)$ are orthogonal and thus must differ at some characters $i, j \in \mathbb{N}$ after a common prefix. Then, by the well-known totality of $\leq_{\mathbb{N}}$, their comparability with $\prec$ results. Also by the orthogonality know that $t|_{\pi_{\min}}$ is the innermost redex since there is no reducible subterm, by the minimality w.r.t $\prec$ we know it is leftmost innermost. Finally, by groundness, we know that $t|_{\pi_{\min}}$ must be matched with a lhs with the empty substitution and is thus a lhs.
2. We know from 1. that there is a unique position $\pi_{\min} \in \text{redex}(t)$ such that $t|_{\pi_{\min}}$ is the leftmost innermost redex and the lhs of a rule $t|_{\pi_{\min}} \rightarrow \mu = \{p_1 : r_1, \dots, p_n : r_n\} \in \mathcal{R}$. Then, using this rule, we get the rewrite step $t \xrightarrow{l_i}_{\mathcal{R}} \mu' = \{p_1 : t_1, \dots, p_n : t_n\}$ with $t_i = t[r_i]_{\pi_{\min}}$ by definition of rewrite steps. Now, by 1., let $\pi_1 \prec \dots \prec \pi_m$ for $\pi_j \in \text{redex}(t)$, $1 \leq j \leq m$ and $\tau_1 \prec \dots \prec \tau_k$ for $\tau_l \in \text{redex}(r_i)$, $1 \leq l \leq k$. Since r_i occurs in t_i at position $\pi_1 = \pi_{\min}$, all terms that were in position τ_l in r_i are in position $\pi_1.\tau_l$ in t_1 . Now, since π_1 is the smallest position out of $\text{redex}(t)$, that means that within the characters of π_1 there is already a character that is smaller than some character at the same position for every other π_i with $1 < i \leq m$. This means, that $\pi_1.\tau_j$ for all $1 \leq j \leq k$ is still smaller than any of the π_i with $1 < i \leq m$. Also, we have the chain $\pi_1.\tau_1 \prec \dots \prec \pi_1.\tau_k$, since any positions that were ordered by $\prec$ before remain in the same order if we add the same prefix. We conclude that the elements of $\text{redex}(t_i)$ are ordered with the chain $\pi_1.\tau_1 \prec \dots \prec \pi_1.\tau_k \prec \pi_2 \prec \dots \prec \pi_m$.

□

Finally, we can give the reduction between rewriting systems and grammars that we were building toward.

Lemma 4.3.11: From Rewriting to Grammars

Let $\mathcal{R}$ be a PTRS in GNT and let $N = \{l \mid l \rightarrow \mu \in \mathcal{R}\}$ be the set of lhs of rules of $\mathcal{R}$. Also let $(G_{\mathcal{R}}^l)_{l \in N}$ be a family of $|\mathcal{R}|$ SCFGs with $G_{\mathcal{R}}^l = (N, \emptyset, P, l)$ where

$$P := \{l \xrightarrow{p} \rho(t) \mid l \rightarrow \mu \in \mathcal{R}, (p : t) \in \mu\}$$

and $\rho : N^* \rightarrow N^*$ is defined as:

$$\rho(t) = \begin{cases} t, & \text{if } t \in N \\ \rho(t_1) \dots \rho(t_n), & \text{if } t = f(t_1, \dots, t_n) \text{ and } t \notin N \\ \varepsilon, & \text{otherwise} \end{cases}$$

Then $\mathcal{R}$ is AST if and only if $G_{\mathcal{R}}^l$ is consistent for every $l \in N$.

The function ρ acts as a translator between terms and words. To achieve this, the inherent structure of terms is removed and only the reducible lhs of rules remain in a concatenated string, in the same left to right order as they occur in the structure. This is possible due to the tail-recursiveness of the underlying PTRS. We express this more formally in the following lemma.

Lemma 4.3.12: Property of ρ

Let $\mathcal{R}$ be a PTRS in GNT and t be a tail-recursive, ground term where we have $\pi_1 \prec \dots \prec \pi_n$ with $\pi_i \in \text{redex}(t)$ as defined in Lemma 4.3.10 for $1 \leq i \leq n$ and n is the number of reducible expressions. Then ρ , as defined in Lemma 4.3.11, satisfies

$$\rho(t) = \begin{cases} t|_{\pi_1} \dots t|_{\pi_n}, & n > 0 \\ \varepsilon, & \text{otherwise.} \end{cases}$$

Proof. We prove this by structural induction on t :

- $t \in N$:

We have $\text{redex}(t) = \{\pi_1 = \varepsilon\}$ by 1. of 4.3.10, since the root term itself is a redex we cannot have reducible subterms by tail-recusiveness. By the case definition $t \in N$ we then have $\rho(t) = t = t|_{\varepsilon}$.

- $t \neq f(t_1, \dots, t_n)$ and $t \notin N$:

Then t is in normal form w.r.t. $\mathcal{R}$, so t contains no redex. Thus we have $n = 0$ and $\rho(t) = \varepsilon$.

- $t = f(t_1, \dots, t_n)$ and $t \notin N$:

Let $\pi_{i_1} \prec \dots \prec \pi_{i_m}$ with $i_m \in \mathbb{N}$ for $1 \leq i \leq n$ be the ordered elements of $\text{redex}(t_i)$, where i_m is the respective number of redexes of t_i . Then consider $\text{redex}(t)$, which can be ordered in the chain $1.\pi_{1_1} \prec \dots \prec 1.\pi_{1_m} \prec 2.\pi_{2_1} \prec \dots \prec n.\pi_{n_m}$. This follows

from the fact that the positions of the subterms already follow a strict left to right order with $\prec$, dictated by the first character, which is already different. Thus of course their respective redex chains can be ordered in the same way. We now want to prove $\rho(t) = t|_{1.\pi_1} \dots t|_{n.\pi_m}$. We have $\rho(t) = \rho(t_1) \dots \rho(t_n)$ by definition. By induction assumption, $\rho(t_i) = t_i|_{\pi_{i_1}} \dots t_i|_{\pi_{i_m}} = t|_{i.\pi_{i_1}} \dots t|_{i.\pi_{i_m}}$ for all $1 \leq i \leq n$. To avoid unnecessary case distinctions, let $t|_{\pi_{i_1}} \dots t|_{\pi_{i_m}} = \varepsilon$ if $i_m = 0$. The claim immediately follows. □

We introduce another intermediate lemma that relates computation trees and derivations.

Lemma 4.3.13: Correspondence between computation trees and derivations

Consider a PTRS $\mathcal{R}$ with $N = \{l \mid l \rightarrow \mu \in \mathcal{R}\}$ and a corresponding family of SCFGs $(G_{\mathcal{R}}^l)_{l \in N}$ as constructed in Lemma 4.3.11. Then, for any fixed $l \in N$ let $\mathfrak{T} = (V, E, L)$ be the unique leftmost innermost $\mathcal{R}$ -computation tree with root $\mathfrak{r}$ such that $L(\mathfrak{r}) = (1 : l)$ and let $D_{G_{\mathcal{R}}^l}(\varepsilon)$ be the set of leftmost derivations of ε in $G_{\mathcal{R}}^l$. The following now holds:

The function $\psi : P(\mathfrak{T}) \rightarrow D_{G_{\mathcal{R}}^l}$ which maps paths

$$(x_0, \dots, x_n) \in P(\mathfrak{T}) \text{ with } L(x_i) = (p_i : t_i) \text{ for } 0 \leq i \leq n$$

to derivations

$$d = \rho(t_0) \xrightarrow{p_0} \dots \xrightarrow{p_{n-1}} \rho(t_n) \in D_{G_{\mathcal{R}}^l}(\varepsilon) \text{ with } \mu(d) = p_n$$

is a bijection.

Here $P(\mathfrak{T}) = \{(x_0, \dots, x_n) \mid x_0 = \mathfrak{r}, x_n \in \text{Leaf}, (x_i, x_{i+1}) \in E \text{ for } 0 \leq i < n\}$ is the set of paths from the root node to a leaf node in $\mathfrak{T}$.

Proof. We prove that ψ is a bijection by proving well-definedness, injectivity and surjectivity. Before that, we note a few relevant things. First let $(x_0, \dots, x_n) \in P(\mathfrak{T})$ be a path from $x_0 = \mathfrak{r}$ to a node $x_n \in \text{Leaf}$ with $L(x_i) = (p_i : t_i)$ for $0 \leq i \leq n$. Also, since $\mathfrak{T}$ is a leftmost innermost $\mathcal{R}$ -computation tree we know that the path corresponds to a unique sequence of rules $(r_0 \rightarrow \mu_0, \dots, r_{n-1} \rightarrow \mu_{n-1})$ where by the groundness of $\mathcal{R}$ we have $t_i|_{\pi_{min}} = r_i$ and $t_{i+1} = t_i[r'_i]_{\pi}$ for $(\frac{p_{i+1}}{p_i} : r'_i) \in \mu_i$ and leftmost innermost position $\pi_{min} \in \text{Occ}(t_i)$ according to (1.) of Lemma 4.3.10. Note that by restricting ourselves to leftmost innermost rewriting sequences and by the fact that $\mathcal{R}$ is non-overlapping the choice of the redex $t_i|_{\pi}$ and the choice of the applied rule are both deterministic. Also note that the normalized probability $\frac{p_{i+1}}{p_i}$ comes from the fact that the probabilities p_i for $0 \leq i \leq n$ in the node labels of the tree are not the probabilities of the rules applied in the rewriting step from parent to child, but the product of rule probabilities along the path from the root node to the child, according to the condition denoted in Definition 4.3.8.

- ψ is well-defined:

To show that ψ is well-defined we show that the derivation d we map to is unique and also a proper derivation in $D_{G_{\mathcal{R}}^l}(\varepsilon)$. Formally we can split this up into the following statements which we prove individually:

1. $\rho(t_0) = l$
2. $\rho(t_n) = \varepsilon$
3. $\rho(t_i) \xrightarrow[p_i]{p_{i+1}}_{G_{\mathcal{R}}^l} \rho(t_{i+1})$ for each $0 \leq i < n$

(1.) Since $t_0 = l$ is given by the definition of the $\mathcal{R}$ -computation tree and $l \in N$, by definition of ρ we have $\rho(t_0) = l$.

(2.) We have $x_n \in \text{Leaf}$ and $L(x_n) = (p_n : t_n)$ for some $p_n \in [0, 1]$, thus t_n is in normal form w.r.t. $\mathcal{R}$, i.e., neither t_n nor any subterm can be a redex. By Lemma 4.3.12 we then have $\rho(t_n) = \varepsilon$.

(3.) Consider $\rho(t_i)$ for any $0 \leq i < n$. Then we can order the positions of reducible expressions of t_i (i.e., elements of $\text{redex}(t_i)$) in a chain $\pi_1 \prec \dots \prec \pi_m$ by Lemma 4.3.10. Then, by Lemma 4.3.12 we have $\rho(t_i) = t_i|_{\pi_1} \cdot \dots \cdot t_i|_{\pi_m}$, where by the groundness $t_i|_{\pi_j} \in N$ for all $1 \leq j \leq m$ and $t_i|_{\pi_1} = t_i|_{\pi_{\min}}$ is the leftmost innermost redex. Next, rule $l_i = t_i|_{\pi_1} \rightarrow \mu_i \in \mathcal{R}$ with $(\frac{p_{i+1}}{p_i} : r'_i) \in \mu_i$ translates to production rule

$t_i|_{\pi_1} \xrightarrow[p_i]{p_{i+1}} \rho(r'_i) \in P$ where $\tau_1 \prec \dots \prec \tau_k$ for $\tau_l \in \text{redex}(r'_i)$. Now by (2.) of Lemma 4.3.10 with $t_{i+1} = t_i[r'_i]_{\pi_1}$ we have $\pi_1.\tau_1 \prec \dots \prec \pi_1.\tau_k \prec \pi_2 \prec \dots \prec \pi_m$ as the ordered elements of $\text{redex}(t_{i+1})$. Then by 4.3.12 again we have $\rho(t_{i+1}) = t_i|_{\pi_1.\tau_1} \cdot \dots \cdot t_i|_{\pi_m}$.

Obviously then, with production rule $t_i|_{\pi_1} \xrightarrow[p_i]{p_{i+1}} \rho(r'_i)$, we have valid leftmost derivation step

$$\rho(t_i) = t_i|_{\pi_1} \cdot \dots \cdot t_i|_{\pi_m} \xrightarrow[p_i]{p_{i+1}}_{G_{\mathcal{R}}^l} t_i|_{\pi_1.\tau_1} \cdot \dots \cdot t_i|_{\pi_1.\tau_k} \cdot t_i|_{\pi_2} \cdot \dots \cdot t_i|_{\pi_m} = \rho(t_{i+1})$$

in $G_{\mathcal{R}}^l$.

- ψ is injective:

Consider any additional path, such that $(x_0, \dots, x_n) \neq (x'_0, \dots, x'_m) \in P(\mathfrak{T})$ where also $L(x'_i) = (p'_i : t'_i)$ for $0 \leq i \leq m$. Then at some position $1 \leq i \leq \min(n, m)$ we must have $x_i \neq x'_i$, where $x_j = x'_j$ for all $0 \leq j < i$. Since both paths are contained in $\mathfrak{T}$, we must have $x_i, x'_i \in x_{i-1}E$ with $L(x_{i-1}) = (p_{i-1} : t_{i-1})$ and by the condition

in 4.3.8 $t_{i-1} \xrightarrow[p_{i-1}]{l_i}_{\mathcal{R}} \mu_i$ with $(\frac{p_i}{p_{i-1}} : t_i), (\frac{p'_i}{p_{i-1}} : t'_i) \in \mu_i$. Now $\rho(t_{i-1}) \xrightarrow[p_{i-1}]{p_i}_{G_{\mathcal{R}}^l} \rho(t_i)$ and

$\rho(t_{i-1}) \xrightarrow[p_{i-1}]{p'_i}_{G_{\mathcal{R}}^l} \rho(t'_i)$ are different derivation steps.¹

¹With certain PTRSs, e.g., $A \rightarrow \{\frac{1}{3} : A, \frac{1}{3} : A, \frac{1}{3} : 0\}$, we could have $p_i = p'_i$ and $t_i = t'_i$ here, where the two derivation steps in the grammar would look the same. By construction, two different rules were still used, making the derivations formally different nonetheless, which suffices for this proof.

- ψ is surjective:

Consider any derivation $d = \gamma_0 \xRightarrow{p'_0} \dots \xRightarrow{p'_{n-1}} \gamma_n \in D_{G_{\mathcal{R}}^l}(\varepsilon)$. By definition at each step $\gamma_i \xRightarrow{p'_i} \gamma_{i+1}$ for $0 \leq i < n-1$ we applied some rule $r_i \xrightarrow{p'_i} \rho(r'_i) \in P$ which exists due to a rule $r_i \rightarrow \mu_i \in \mathcal{R}$ with $(p_i : r'_i) \in \mu_i$. We prove that we have a unique path $(x_0, \dots, x_n) \in P(\mathfrak{T})$ with $L(x_i) = (p_i : t_i)$ that is mapped to d by induction on the length $n \in \mathbb{N}$:

- $n = 0$: We have $\gamma_0 = l = \rho(t_0) = t_0$ for unique root node $x_0 = \mathfrak{r}$ of $\mathfrak{T}$ where $L(x_0) = (1 : t_0)$ by definition.
- $n > 0$: Let $(x_0, \dots, x_i)$ with $L(x_i) = (p_i : t_i)$ for $0 \leq i < n$ be the unique path that is mapped to $\gamma_0 \xRightarrow{p'_0} \dots \xRightarrow{p'_i} \gamma_i$ by induction assumption, i.e. $\gamma_j = \rho(t_j)$ for $0 \leq j \leq i$. Also consider production $r_i \xrightarrow{p'_i} \alpha \in P$, which is used for leftmost derivation step $\gamma_i \xRightarrow{p'_i} \gamma_{i+1}$. We thus know that r_i is the leftmost nonterminal of $\gamma_i = \rho(t_i)$. By Lemma 4.3.12 we then have $r_i = t_i|_{\pi_{min}}$. Now reconsider that for the production rule there exists a unique corresponding rule $r_i \rightarrow \mu_i \in \mathcal{R}$ with $(p'_i : r'_i) \in \mu_i$ and we must have $\alpha = \rho(r'_i)$. It follows that using this rule we get $t_{i+1} = t_i[r'_i]_{\pi_{min}}$ after a leftmost innermost rewriting step. With Lemma 4.3.10 (2.) and considering the position chains $\pi_1 \prec \dots \prec \pi_m$ of $\text{redex}(t_i)$ and $\tau_1 \prec \dots \prec \tau_k$ of $\text{redex}(r'_i)$, let $\pi_1.\tau_1 \prec \dots \prec \pi_1.\tau_k \prec \pi_2 \prec \dots \prec \pi_m$ be the resulting chain of $\text{redex}(t_{i+1})$. Then $\gamma_{i+1} = \rho(t_{i+1})$. This follows immediately from the definition of ρ and the corresponding Lemma 4.3.12 about ρ , which give us that with $\gamma_i = \rho(t_i)$ and the used production rule on the leftmost nonterminal the derivation step to γ_{i+1} must have looked exactly as the one at the end of the well-definedness proof.

We have proven that there is a unique path $(x_0, \dots, x_n)$ that maps to derivation d . The only thing we have left to show is that the probabilities match, i.e., $\mu(d) = p_n$. We currently know $\mu(d) = \prod_{0 \leq i \leq n-1} p'_i$. Since at any rewriting step from t_i to t_{i+1} we use a rule with probability p'_i and we know by the condition in Definition 4.3.8 $p'_i = \frac{p_{i+1}}{p_i}$, we get $\mu(d) = \prod_{0 \leq i \leq n-1} p'_i = \prod_{0 \leq i \leq n-1} \frac{p_{i+1}}{p_i} = p_n$ for any length that allows a rewriting step, i.e., $n > 0$ (otherwise we simply have $\mu(d) = 1 = p_0$). The last equation may not seem obvious, so we give a quick formal proof with an induction on $n \in \mathbb{N} \setminus \{0\}$:

- $n = 1$:

$$\prod_{0 \leq i \leq 1-1} \frac{p_{i+1}}{p_i} = \frac{p_1}{p_0} = p_1 \text{ since } p_0 = 1.$$
- $n > 1$:

$$\begin{aligned} \prod_{0 \leq i \leq n-1} \frac{p_{i+1}}{p_i} &= \frac{p_n}{p_{n-1}} \cdot \prod_{0 \leq i \leq n-2} \frac{p_{i+1}}{p_i} \\ &= \frac{p_n}{p_{n-1}} \cdot p_{n-1} && \text{(Induction Assumption)} \\ &= p_n \end{aligned}$$

□

The last intermediate theorem we need to be able to proof the correctness of our reduction is about innermost AST.

Theorem 4.3.14: Witness Theorem for PTRS [KG23]

If a PTRS $\mathcal{R}$ is not innermost AST then there exists an innermost $\mathcal{R}$ -computation tree $\mathfrak{T}$ that converges with probability < 1 and starts with $(1 : t)$ such that $t = l\sigma$ for some rule $l \rightarrow r \in \mathcal{R}$ and some substitution $\sigma \in \text{Sub}(\Sigma, \mathcal{V})$, and every proper subterm of $l\sigma$ is in normal form w.r.t. $\mathcal{R}$.

Finally we can prove Lemma 4.3.11.

Proof of Lemma 4.3.11. We prove $\mathcal{R}$ is AST $\Leftrightarrow G_{\mathcal{R}}^l$ is consistent for every $l \in N$.

$\Rightarrow$:

$G_{\mathcal{R}}^l$ is inconsistent for some l i.e., $\mu(L(G_{\mathcal{R}}^l)) < 1$. Then let $\mathfrak{T} = (V, E, L)$ be the unique leftmost innermost $\mathcal{R}$ -computation tree with root $\mathfrak{r}$ such that $L(\mathfrak{r}) = (1 : l)$. We differentiate $L(G_{\mathcal{R}}^r) = \emptyset$ and $L(G_{\mathcal{R}}^l) = \{\varepsilon\}$. In the first case, every derivation is infinite. By the bijection of Lemma 4.3.13 every path in $\mathfrak{T}$ is infinite, which means that the leftmost innermost rewriting sequence it corresponds to converges to 0. Thus $\mathcal{R}$ is not AST by Lemma 4.3.6. On the other hand, if we have at least one finite derivation, we have $L(G_{\mathcal{R}}^l) = \{\varepsilon\}$. $L(G_{\mathcal{R}}^l) \subseteq \{\varepsilon\}$ since there exist no terminal symbols and thus every finite derivation derives ε . For $L(G_{\mathcal{R}}^l) \supseteq \{\varepsilon\}$ note that there exists a finite derivation by assumption. We get:

$$\begin{aligned}
\mu(\{\varepsilon\}) &= \sum_{w \in \{\varepsilon\}} \mu(w) = \mu(\varepsilon) < 1 \\
&\Rightarrow \sum_{d \in D_{G_{\mathcal{R}}^l}(\varepsilon)} \mu(d) < 1 \\
&\Rightarrow \sum_{(x_0, \dots, x_n) \in P(\mathfrak{T})} p_{x_n}^{\mathfrak{T}} < 1 && \text{(Lemma 4.3.13)} \\
&\Rightarrow \sum_{x \in \text{Leaf}^{\mathfrak{T}}} p_x^{\mathfrak{T}} = |\mathfrak{T}|_{\text{Leaf}} < 1 \\
&\Rightarrow \mathcal{R} \text{ is not leftmost innermost AST} \\
&\Rightarrow \mathcal{R} \text{ is not AST} && \text{(Lemma 4.3.6)}
\end{aligned}$$

$\Leftarrow$:

$\mathcal{R}$ is not AST. Then, by $\mathcal{R} \in \text{GNT}$ and Lemma 4.3.6 $\mathcal{R}$ is not innermost AST. By the witness theorem, i.e., Theorem 4.3.14, and by the groundness of $\mathcal{R}$, there exists an innermost $\mathcal{R}$ -computation tree $\mathfrak{T}$ that converges with probability < 1 and starts with $(1 : l)$ such that $l \rightarrow r \in \mathcal{R}$ and every proper subterm of l is in normal form w.r.t. $\mathcal{R}$. From this tree we can now construct a leftmost innermost tree $\mathfrak{T}'$ starting with the same root $(1 : l)$ and with $|\mathfrak{T}'|_{\text{Leaf}} \leq |\mathfrak{T}|_{\text{Leaf}}$, as in the construction for the proof of Theorem 22 in [KFG24]. Now addi-

tionally consider the grammar $G_{\mathcal{R}}^l$. We have:

$\mathcal{R}$ is not AST

$\Rightarrow \mathcal{R}$ is not innermost AST (Lemma 4.3.6)

$\Rightarrow \exists l \in N : |\mathfrak{T}|_{\text{Leaf}} = \sum_{x \in \text{Leaf}^{\mathfrak{T}}} p_x^{\mathfrak{T}} < 1$ for $\mathfrak{T}$ with $L(\mathfrak{r}) = (1 : l)$ (Theorem 4.3.14)

$\Rightarrow \exists l \in N : |\mathfrak{T}'|_{\text{Leaf}} = \sum_{x \in \text{Leaf}^{\mathfrak{T}'}} p_x^{\mathfrak{T}'} < 1$ for $\mathfrak{T}'$ with $L(\mathfrak{r}) = (1 : l)$ ([KFG24, Thm. 22])

$\Rightarrow \exists l \in N : \sum_{(x_0, \dots, x_n) \in P(\mathfrak{T}')} p_{x_n}^{\mathfrak{T}'} < 1$ for $\mathfrak{T}'$ with $L(\mathfrak{r}) = (1 : l)$

$\Rightarrow \exists l \in N : \sum_{d \in D_{G_{\mathcal{R}}^l}(\varepsilon)} \mu(d) < 1$ (Lemma 4.3.13)

$\Rightarrow \exists l \in N : \mu(\varepsilon) = \sum_{w \in \{\varepsilon\}} \mu(w) = \mu(L(G_{\mathcal{R}}^l)) < 1$

$\Rightarrow \exists l \in N : G_{\mathcal{R}}^l$ is inconsistent

$\Rightarrow G_{\mathcal{R}}^l$ is not consistent for every $l \in N$.

□

We can now use this lemma to define the central theorem of this thesis.

Problem 4.3.15: AST Problem

Input: A PTRS $\mathcal{R}$.

Question: Is $\mathcal{R}$ AST?

Theorem 4.3.16: Decidability of AST in GNT

AST is decidable for every PTRS in GNT.

Proof. Let $\mathcal{R}$ be a PTRS in GNT. Construct $G_{\mathcal{R}}^l$ for every $l \in \{l \mid l \rightarrow \mu \in \mathcal{R}\}$ as in Lemma 4.3.11. Decide, whether every grammar in the family is consistent. If yes, then $\mathcal{R}$ is AST, if no, then its not. By Lemma 4.3.11, this procedure is both sound ($\Leftarrow$) and complete ($\Rightarrow$). Also, with the decidability of SCFG consistency (see Theorem 4.1.19), it is clearly computable. Thus we have a decision procedure for AST of PTRS in GNT. □

Example 4.3.17: Decidability of AST in GNT

Consider $\mathcal{R}$ from Example 4.3.1 from the beginning of this section again. It is clearly ground, non-overlapping and tail-recursive. Thus we can apply our reduction, resulting

in the grammar productions

$$\begin{array}{ll} T \xrightarrow{\frac{1}{2}} \varepsilon & T \xrightarrow{\frac{1}{2}} TNT \\ N \xrightarrow{\frac{1}{2}} \varepsilon & N \xrightarrow{\frac{1}{2}} N \end{array}$$

which were already mentioned before. By our results, to decide AST, we must now decide consistency for each of the grammars with these productions and a left-hand side of one of the rules as start symbol. With the method we introduced in the chapter before, we do this by first constructing a corresponding moment matrix and seeing whether its spectral radius is > 1 . We have

$$B(1) = \begin{pmatrix} \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 2 & \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 \\ \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 0 & \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 \end{pmatrix} = \begin{pmatrix} 1 & \frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix}$$

and characteristic polynomial

$$\det \begin{pmatrix} 1-x & \frac{1}{2} \\ 0 & \frac{1}{2}-x \end{pmatrix} = (1-x) \cdot (\frac{1}{2}-x) - \frac{1}{2} \cdot 0$$

where we can easily see that the roots are $\frac{1}{2}$ and 1, which means that the spectral radius of $B(1)$ is $1 \not> 1$, so the original PTRS is AST. Note how when constructing the moment matrix and in any further step the start symbol of the matrix is irrelevant, thus every grammar in the family has the same matrix and gets the same result, $1 \not> 1$, i.e., consistency. This is especially relevant later during our implementation, because it means that for each connected component we only have to construct a single matrix and check its spectral radius only once.

4.4 Removing Restrictions

In the previous section we have seen that ground, non-overlapping and tail-recursive PTRS can be transformed into a family of SCFGs to answer the question of almost-sure termination for the PTRS by instead looking at SCFG consistency, which is known to be decidable. Now, we build on this foundation by lifting some of the rather harsh restrictions and seeing that AST for a broader class of PTRS is still decidable.

The first restriction we tackle is groundness. Generally, stochastic grammars are like markov chains in the sense that each derivation step is only determined by the current state, i.e., the current nonterminal. Variables in term rewriting systems introduce some sort of memory that cannot be translated to SCFGs. We circumvent this issue by erasing all variables withing the first step of any rewrite sequence. A system that does this is called *right-ground*.

Definition 4.4.1: Right-Ground PTRS

We call a PTRS $\mathcal{R}$ *right-ground*, if for every rule $l \rightarrow \mu \in \mathcal{R}$ and $r \in \text{Supp}(\mu)$ we have $\mathcal{V}(r) = \emptyset$, i.e., right-hand sides of rules contain no variables.

In right-ground PTRS we can now have arbitrary variables on lhs and no variables on rhs of rules. We replace the property of groundness by the looser right-groundness and get a broader fragment of PTRS.

Definition 4.4.2: Right-ground, Non-overlapping and Tail-recursive PTRS (RNT)

We define **RNT** as the set of all PTRS $\mathcal{R}$ that are right-ground, non-overlapping and tail-recursive.

Our main goal is still achieving decidability of AST. Thus we want to use the decidability of GNT by mapping PTRSs in RNT to PTRSs in GNT which we can decide AST for. To this end, we require an AST preserving transformation. The idea of this transformation is to simply replace all lhs of rules containing variables by their instantiated versions occurring on rhs of rules.

Example 4.4.3: Resolving Right-Groundness to Groundness

Consider the following PTRS in **RNT**:

$$f(x) \rightarrow \{\frac{1}{2} : g, \frac{1}{2} : c(f(a), f(b))\}$$

The transformed, iAST equivalent PTRS in **GNT** looks like the following:

$$f(a) \rightarrow \{\frac{1}{2} : g, \frac{1}{2} : c(f(a), f(b))\}$$

$$f(b) \rightarrow \{\frac{1}{2} : g, \frac{1}{2} : c(f(a), f(b))\}$$

This works since after just one rewriting step, any arbitrary term that matched with a variable on a lhs, is removed and we can only ever produce the instantiated versions of the lhs, meaning that our transformed, ground system can also match them and we can recreate the same rewriting sequences. Unfortunately, we can only show that it is innermost AST preserving, since for the proof we need to use the witness theorem from before which only works for iAST. For GNT we know by 4.3.4 that iAST and AST are equivalent, for RNT this is not clear. If we wanted to apply the same theorem, we would need to add a restriction to left-linear PTRS. or alternatively prove that RNT suffices for the equivalence of iAST and AST in a completely new theorem. This could probably be done by an induction similar to the one contained in the proof of the witness theorem but is beyond the scope of this thesis, which is why we will just show that iAST for RNT is decidable and AST for RLNT (RNT with the addition of left-linearity).

Lemma 4.4.4: From iAST in RNT to iAST in GNT

Let $\mathcal{R}$ be a PTRS in RNT and $N = \{t|_{\pi} \mid l \rightarrow \mu \in \mathcal{R}, t \in \mu, \pi \in \text{Occ}(t), l' \rightarrow \mu' \in \mathcal{R}, \sigma \in \text{Sub}(\Sigma, \mathcal{V}), t|_{\pi} = l'\sigma\}$ be the set of reducible terms on rhs of rules. We define

$$\mathcal{R}' = \{t \rightarrow \mu \mid t \in N, l \rightarrow \mu \in \mathcal{R}, \sigma \in \text{Sub}(\Sigma, \mathcal{V}), t = l\sigma\}$$

such that $\mathcal{R}' \in \text{GNT}$ and

$$\mathcal{R} \text{ is iAST} \Leftrightarrow \mathcal{R}' \text{ is iAST}$$

Proof. We prove $\mathcal{R}' \in \text{GNT}$ and then the second statement. Since $\mathcal{R}$ was finite and we can thus only have finitely many redexes on rhs of rules, $\mathcal{R}'$ is also finite and thus fits our definition of PTRS. Next we check whether the new system is actually ground, non-overlapping and tail-recursive. Note that no right-hand side was changed from $\mathcal{R}$, thus the tail-recursiveness remains. By definition $t \in \mu$ and the right-groundness of $\mathcal{R}$, each lhs t of $\mathcal{R}'$ is a ground term, giving us full groundness. We will prove that $\mathcal{R}'$ is non-overlapping via contradiction. Assume we created two rules $t_1 \rightarrow \mu_1, t_2 \rightarrow \mu_2$ that are overlapping, so at some non-variable position $\pi \in \text{Occ}(t_1)$ we have $t_1|_{\pi}\sigma = t_2\sigma$ for some $\sigma \in \text{Sub}(\mathcal{V}, \Sigma)$. By definition, we must have rules $l_1 \rightarrow \mu_1, l_2 \rightarrow \mu_2 \in \mathcal{R}$ such that $t_1 \in \mu_1$ and $t_2 \in \mu_2$. By that, t_1 and t_2 are ground and tail-recursive. The groundness tells us, that $\sigma = \emptyset$, the tail-recursiveness tells us that $\pi = \varepsilon$, since both terms have defined symbols at the root position and cannot have defined symbols below to match the other one at the root position with. So we must have $t_1 = t_2$. Now by construction, we must also have a rule $l' \rightarrow \mu' \in \mathcal{R}$ such that $t_1 = t_2 = l'\sigma'$ for some $\sigma' \in \text{Sub}(\mathcal{V}, \Sigma)$. This rule must be unique, since $\mathcal{R}$ is non-overlapping and two identical terms with only one defined symbol could not have been matched by different rules. Thus $\mu_1 = \mu' = \mu_2$, so $t_1 \rightarrow \mu_1 = t_2 \rightarrow \mu_2$, which means the rules could only be overlapping at a position $\pi \neq \varepsilon$, which we established cannot be the case. Thus our assumption was wrong and we cannot have created overlapping rules. Now for the second statement, we prove both directions individually.

$\Rightarrow$:

Assume $\mathcal{R}'$ is not iAST. Then we have an infinite innermost rewrite sequence $(\mu_n)_{n \in \mathbb{N}}$ such that $\lim_{n \rightarrow \infty} |\mu_n| < 1$. Since for any rewrite step and any rule $t \rightarrow \mu \in \mathcal{R}'$ that was matched with substitution σ' we have a rule $l \rightarrow \mu \in \mathcal{R}$ where $l\sigma = t$ for some σ , we can also match $l \rightarrow \mu$ in $\mathcal{R}$ with $\sigma' \circ \sigma$ and get a rewrite step in $\mathcal{R}$. Thus $(\mu_n)_{n \in \mathbb{N}}$ is also an infinite rewrite sequence in $\mathcal{R}$ and $\mathcal{R}$ is not iAST.

$\Leftarrow$:

Assume $\mathcal{R}$ is not iAST. Then by Theorem 4.3.14 there exists an innermost $\mathcal{R}$ -computation tree $\mathfrak{T}$ that converges with probability < 1 and starts with $(1 : t)$ such that $t = l\sigma$ for some rule $l \rightarrow \mu \in \mathcal{R}$ and some substitution $\sigma \in \text{Sub}(\Sigma, \mathcal{V})$, and every proper subterm of $l\sigma$ is in normal form w.r.t. $\mathcal{R}$. There must now be a $t' \in \mu$ such that t' induces an infinite rewrite sequence with a corresponding innermost $\mathcal{R}$ -computation tree $\mathfrak{T}'$ with root $(1 : t')$ that also converges with probability < 1 . If not all terms we rewrite to in the first step would be converging and

the original tree thus would also converge. Thus consider now $\mathfrak{T}'$ with start $(1 : t')$ and $t' \in \mu$. Note that t' must be ground and tail-recursive. Thus the set $\mathfrak{R}(t') = \{r \trianglelefteq t' \mid r \in N\}$ contains the orthogonal reducible subterms of t' , which are by definition all in N since t' is on a rhs of a rule. Note that by the orthogonality of $\mathfrak{R}(t')$ and tail-recursiveness of $\mathcal{R}$ we can regard the rewriting sequences induced by the reducible subterms of t' separately in the context of termination, as they do not influence each other². Once again there must be a subterm $t'' \trianglelefteq t'$ with $t'' \in \mathfrak{R}(t')$ such that t'' is not iAST. This means there is an innermost $\mathcal{R}$ -computation tree $\mathfrak{T}''$ that starts with $(1 : t'')$ and converges with probability < 1 . This tree is now also an innermost $\mathcal{R}'$ -computation tree. We show by an induction on the depth $n \in \mathbb{N}$ of $\mathfrak{T}''$ that every time a rule in $\mathcal{R}$ was applied we could also apply a corresponding rule in $\mathcal{R}'$:

- $n = 0$:

The only $\mathcal{R}$ reducible subterm at depth 0 is t'' itself, since it is the root node and cannot contain reducible subterms, being contained in a rhs of a tail-recursive system. Let $l \rightarrow \mu \in \mathcal{R}$ be the only³ rule such that $l\sigma = t''$ for some $\sigma \in \text{Sub}(\Sigma, \mathcal{V})$. Since $t'' \in N$ we then must have a rule $t'' \rightarrow \mu \in \mathcal{R}'$ that results in the same rhs μ when applied.

- $n > 0$:

Any node at this height consists of a tail-recursive, ground term, since we start with a tail-recursive, ground term and we can only rewrite to tail-recursive, ground terms. In particular, rewriting a subterm of a tail-recursive, ground term to another tail-recursive, ground term does not change that property for the whole term. Then by Lemma 4.3.10 all reducible subterms can be ordered orthogonally. Now in $\mathfrak{T}'$ one of these subterms, let us name it r , was chosen to rewrite with using rule $l \rightarrow \mu \in \mathcal{R}$ where $l\sigma = r$. By definition then $r \in N$, since r must have been on a rhs of a rule, giving us $r \rightarrow \mu \in \mathcal{R}'$.

Since $\mathfrak{T}'$ is a $\mathcal{R}'$ -computation tree that converges with probability < 1 we know that $\mathcal{R}'$ is not iAST. □

With this we can formulate the following theorem about the decidability of iAST for RNT.

Theorem 4.4.5: Decidability of iAST for RNT

iAST is decidable for every PTRS in RNT.

Proof. Let $\mathcal{R}$ be a PTRS in RNT. Let $\mathcal{R}'$ be the corresponding PTRS in GNT according to Lemma 4.4.4. Decide, whether $\mathcal{R}'$ is AST. If it is, $\mathcal{R}$ is iAST, if not, $\mathcal{R}$ is not iAST.

The construction from Lemma 4.4.4 is trivially computable, and the proof of 4.3.16 gives us a computable algorithm for deciding AST in GNT. Also by Lemma 4.4.4 we know that $\mathcal{R}$ is iAST iff $\mathcal{R}'$ is iAST. Finally $\mathcal{R}'$ is iAST iff $\mathcal{R}'$ is AST by Theorem 4.3.4, giving us the soundness and completeness of our procedure. □

²One could prove this in more detail in future work.

³Since $\mathcal{R}$ is non-overlapping

As mentioned before, resolving groundness by allowing arbitrary variables on the lhs forced us to focus on iAST since we could not apply the Witness Theorem in its current form. We might be able to prove the decision procedure also works for full AST with an adapted Witness Theorem, but this exceeded this thesis' scope. An alternative mentioned was to simply restrict the occurring variables on lhs to linearity, allowing us to easily achieve decidability of full AST.

Definition 4.4.6: Right-ground, Linear, Non-overlapping and Tail-recursive PTRS (RLNT)

We define RLNT as the minimal set such that for each PTRS $\mathcal{R}$ that is right-ground, (left-)linear, non-overlapping and tail-recursive we have $\mathcal{R} \in \text{RLNT}$.

For these slightly more restricted systems we can now decide full AST.

Theorem 4.4.7: Decidability of AST for RLNT

AST is decidable for every PTRS in RLNT.

Proof. Every PTRS in RLNT is also in RNT. Thus iAST is already known to be decidable for RLNT by Theorem 4.4.5. Now since every PTRS in RLNT is both non-overlapping and linear by Theorem 4.3.4 we know that iAST and AST are equivalent. The decidability of AST then immediately follows from the decidability of iAST. $\square$

5 Implementation

In this chapter we will take a closer look at how the algorithm that was introduced in chapter 4 can be implemented. As an example, we implemented the results in the verification tool AProVE, which we will quickly introduce before going over the main aspects of the implementation.

5.1 AProVE

The novel decision algorithm established in this paper has been incorporated into the termination analysis framework AProVE. This tool accepts programs as input and investigates their termination behavior as well as their computational complexity. Of particular interest to us are instances that can be modeled as probabilistic term rewriting systems (PTRSs) for which termination properties such as AST or iAST must be established. These are modeled as objects of type `PTRSPROBLEM` and our goal is to resolve them, returning an answer to the termination or complexity question in doing so.

Within AProVE, the analysis of a `PTRSPROBLEM` is delegated to a collection of processors. Each processor attempts to transform the given problem into a simpler one that is equivalent with respect to termination. These processors can be executed sequentially, and the overall computation terminates as soon as one processor yields a conclusive result. Our results were implemented in the form of two processors, the first one of which is applied when the PTRS is in GNT form and which can immediately decide both AST and iAST, the second of which can transform RNT to GNT PTRS according to our reduction in the case of iAST.

In other words, whenever a program is assigned to the PTRS processor and the objective is to prove or refute (innermost) almost-sure termination ((i)AST), we check whether it is Ground/Right-ground, Non-overlapping and Tail-recursive (GNT/RNT). In that case our newly developed processors are invoked and resolve the problem.

Our result is mostly based upon the decidability of the existential theory of the reals. In particular we only need to decide the validity of a univariate sentence consisting of two polynomials with degree $\leq n$ given the number of rules n , i.e., the dimension of the moment matrix n . It turns out Microsofts SMT solver Z3 (see [Hua+25]) can handle such formulae well. Though nonlinear real arithmetic in general is very expensive and Z3 is not complete for this kind of formula, when presented only non-linear polynomial constraints over reals, Z3 uses a specialized complete solver. In the following one can find an example use of Z3 to solve the characteristic polynomials we have constructed in Example 4.1.20 in section 4.1.

Example 5.1.1: Z3 Playground

Consider the polynomials $h_1(x) = -x^3 + x^2 + \frac{1}{8}$ and $h_2(x) = -x^3 + x^2$ from Example 4.1.20. We now construct the corresponding sentences, which state that they have roots greater than 1, such that we can check their validity with Z3, in the Z3 format.

```
(echo "Does -x^3 + x^2 + 1/8 have a root > 1?")
(declare-const x Real)
(assert (> x 1))
(assert (= (+ (+ (- (^ x 3)) (^ x 2)) 0.125) 0))
(check-sat)

(echo "Does -x^3 + x^2 have a root > 1?")
(declare-const y Real)
(assert (> y 1))
(assert (= (+ (- (^ y 3)) (^ y 2)) 0))
(check-sat)
```

The corresponding output looks like this:

```
Does -x^3 + x^2 + 1/8 have a root > 1?
sat

Does -x^3 + x^2 have a root > 1?
unsat
```

Fortunately, Z3 is already embedded in AProVE, since other processors and functionalities require SMT solving. So we simply had to transform inputs adequately to fit the supported format. To this end, we had to build polynomials $f_i(x)$ of type `SimplePolynomial`, transform them into constraints $f_i(x) \Delta 0$, realized with type `CoefficientConstraint`, translate those to atomic predicates of type `SMTLIBTheoryAtom` and finally combine them to a set of `Formula<SMTLIBTheoryAtom>` representing the boolean combination $R(x)$ of those predicates in the sentence $\exists x R(x)$.

5.2 Adaptation of the Framework

The first step when it comes to the application of the new processor is to verify whether they are even applicable. Fortunately, the class `PTRSProblem` provides us with functionalities to check for groundness and non-overlappingness, so those did not have to be implemented. Since tail-recursiveness has not been studied much in the context of PTRS, AProVE did not have a means to recognize it. But the implementation is quite simple, the algorithm that was used keeps track of the defined symbols in a set and then traverses the tree representation of terms

in a DFS keeping track of whether a path contains more than 1 defined symbol. If such a path is found, the system is not tail-recursive, otherwise it is.

The much more interesting addition to `PTRSPROBLEM` is a function to determine whether a PTRS is weakly normalizing. Though it only emerged as a derivative result of this work as part of the implemented SCFG consistency algorithm the functionality might generally be interesting for other scenarios too, which is why it is not hidden away as a private function within a processor. But before getting into the main idea we define weak normalization for PTRS.

Definition 5.2.1: PTRS Weak Normalization

A PTRS is *weakly normalizing* if for each start term there is a rewriting sequence that reaches a normal form with nonzero probability, i.e., we can rewrite to a multi-distribution that contains a normal form within a finite number of steps.

It is well known that weak normalization is generally undecidable in the non-probabilistic setting already, which of course translates to the probabilistic setting. With the restriction to tail-recursiveness we achieve full decidability, as we can use a variant of the marking algorithm employed to check for useless nonterminals in SCFGs.

Theorem 5.2.2: Decidability of PTRS Weak Normalization

Weak normalization is decidable in GNT.

Before we can prove this we show the correlation to SCFGs.

Lemma 5.2.3: Equivalence of Weak Normalization and No Unproductive Cycles

A PTRS in GNT is weakly normalizing if and only if the family of grammars $(G_{\mathcal{R}}^l)_{l \rightarrow \mu \in \mathcal{R}}$ as defined in 4.3.11 contains no unproductive nonterminals.

Proof. Let $G_{\mathcal{R}}^l$ contain an unproductive nonterminal N . Say, w.l.o.g. that $l = N$ (otherwise just choose $G_{\mathcal{R}}^N$). Then consider any derivation starting from N . Also consider the leftmost innermost $\mathcal{R}$ -computation tree starting from N . We know that for each derivation we have a corresponding path in this tree. Also note, that any normal term is mapped to ε by ρ . Since N is unproductive, we never derive ε with any derivation, which by the bijection means we never rewrite to a normal form in any path. Thus we have a sequence that never rewrites to a normal form starting with N and $\mathcal{R}$ is thus not weakly normalizing.

Say no $G_{\mathcal{R}}^l$ contains an unproductive nonterminal. Thus from every nonterminal we can derive a terminal string (ε). Now consider any start term. We prove with a structural induction on that it is weakly normalizing, i.e., we can reach a normal form with nonzero probability:

- We start with a lhs l of a rule. Consider the unique leftmost innermost $\mathcal{R}$ -computation tree $\mathfrak{T}$. Since each path corresponds to a derivation in $G_{\mathcal{R}}^l$ and we know that we can derive ε in finitely many steps, we know via the bijection that the rewrite sequence represented by $\mathfrak{T}$ also reaches a normal form.

- We have an arbitrary term t . Consider the innermost redex. By the groundness of $\mathcal{R}$ is must be a lhs. With the induction assumption we know that it is weakly normalizing, i.e., reaches at least one normal form. Replace the redex with that normal form. Now we have two cases:
 - There is no more redex and we can rewrite t to a normal form trivially by just rewriting the innermost redex.
 - We have another innermost redex. Again use the induction assumption and replace it with the normal form it can be rewritten to. Inductively we can reason that this terminates with us reaching a normal form by the finiteness of the term.

□

We can now proof our theorem.

Proof of Theorem 5.2.2. Let $\mathcal{R}$ be a PTRS in GNT. The following procedure decide weak normalization for $\mathcal{R}$:

1. Set $S = \mathcal{R}$.
2. Repeat the following until an iteration makes no changes to S :
 - Remove from S all rules that contain terms in normal form w.r.t. S on their rhs
3. If S is empty return `true`, else return `false`

The procedure is trivially computable, since it is easy to check whether a term is in normal form given a set of rules. It also terminates since we either decrease the finite set S until it is empty in each step or terminate immediately if we did not decrease its size.

The procedure is correct since it is analogous to the marking algorithm to find useless nonterminals in (S)CFGs which is well-known to be correct and via Lemma 5.2.3 we know that weak normalization is equivalent to there not being useless nonterminals. One can easily see that the removal of rules in each iteration achieves the same as marking the occurrences of its lhs in the remaining rhs in terms.

□

We implemented this procedure just as described within the `PtrsProblem` class, which means that it can also be used by other processors. It is important to note that this method only works because we could use our reduction from PTRS in GNT to SCFGs, and does not work on arbitrary PTRS (weak normalization/reachability are both generally undecidable).

5.3 The New Processors

We implemented two new processors within the AProVE framework, the first is applied when the PTRS is already in GNT and the second transforms RNT PTRS to GNT according to the reduction we introduced. There was no need to implement the equivalence between AST and

iAST for these PTRS, since there is already a processor that checks for linearity and non-overlappingness and equivalently transforms AST to iAST problems. So it suffices to apply our processors for iAST problems, which conveniently makes the transformation from RNT to GNT and equivalence transformation.

ASTDecidableGNTProcessor

The first processor is the `ASTDecidableGNTProcessor`. It receives a PTRS in GNT and first checks whether it is weakly normalizing. As stated before, this is the case if and only if the family of grammars we reduce it to contains useless nonterminals, making them inconsistent and thus making the PTRS not AST. So if the PTRS is not weakly normalizing we can stop and return NO. If the PTRS is in fact weakly normalizing, in the consistency algorithm we can now skip the step of checking for useless nonterminals and construct the moment matrix of the PTRS. Note that we skipped the first step in the consistency algorithm, which would correspond to removing unreachable nonterminals and constructing clean grammars. This is not necessary, since in our PTRS, we can start with any term and thus all lhs are reachable. But the moment matrix method is only defined on grammars where all nonterminals are reachable, so we split the matrix up into its connected components (CCs). Thus we get almost the exact same set of matrices that we would get if we first transformed the PTRS to SCFGs, removed unreachable nonterminals, and then constructed their corresponding matrices. Almost, since whenever a CC contains more than one nonterminal, we would have constructed a matrix for each sub-CC (i.e., CCs induced by reachable nonterminals within the CC), so this is also a big speed-up for large CCs. The correctness is not affected by this, since the whole CC is consistent if and only if all its sub-CCs are consistent¹. Finally, given the set of submatrices, we compute for each its characteristic polynomial via laplace expansion and feed it to Z3 as mentioned in the previous section, which determines whether there is a root > 1 . We return YES, i.e., the PTRS is AST, if Z3 returns NO for all queries, and NO otherwise.

RNTtoGNTProcessor

This processor is fairly straightforward. If the conditions apply, i.e., the PTRS is right-ground, non-overlapping and tail-recursive, we construct the set S of all redexes on rhs and then build a new PTRS in GNT where for each rule $l \rightarrow \mu$ and all elements t of S that could be unified with l we include $t \rightarrow \mu$.

5.4 Evaluation

As a last step we want to evaluate our new method. To do this, we looked at the set of PTRSs that was used in the Termination Competition 2025 (see [Ter25]). AProVE, where we implemented our results, also participated, which allows for a comparison of its performance on the test set with and without the new processors. The results are summarized in the following table:

¹it is well-known for SCFGs that global consistency and local consistency are equivalent and we will thus not provide a proof

	Without	With
AST	60	60
iAST	112	112
Not AST	0	2

We see that we did not get any new positive results (i.e., iAST / AST) on the test set, even though our theorem was applicable to 10 PTRSs out of the test set. This means that other analysis technique that were already implemented in AProVE could already prove iAST/AST for the ground, non-overlapping and tail-recursive systems out of the test set. On the other hand, we see that we can now output negative results, i.e., not AST, which was not possible before. When interpreting these results one has to consider that the test set is specialized in real programs, where we seldomly have groundness, non-overlappingness and tail-recursiveness occurring all at once, which is why our results were only applicable in less than 10% of test PTRSs. It further emphasizes that the results of this thesis, at least for now, are more of a theoretical nature. The fact that there is a full decision procedure for a fragment of PTRSs means that we now just have to find ways to map to this fragment with iAST/AST equivalent transformation to make this fragment bigger. A few ideas of how to do so are mentioned in the next chapter when talking about what future work might look like.

6 Conclusion and Future Work

At its core, the main contribution of this thesis is drawing a connection between rewriting and grammars. More specifically, we have shown how the much more powerful general probabilistic rewriting can be restricted in ways that allow for a translation to a stochastic context-free grammar, such that (innermost) almost sure termination in the former becomes equivalent to the well-known and long studied consistency in the latter. This translation from one formalism to the other, which is what we call a reduction, could then be leveraged as a central component in an algorithm to decide almost sure termination for the restricted PTRSs, where we use it as a preprocessing step before answering consistency of the resulting SCFG. To be able to compute this transformation and get valid results we required the PTRS to be ground, non-overlapping and tail-recursive for AST and later weakened groundness to right-groundness via an additional theorem that gave an iAST-preserving transformation from right-ground to ground systems.

Finally we implemented all results in the verification tool AProVE via two new processors, which add the first full decision procedure for (i)AST to the framework.

In future work it might be interesting to prove that iAST and AST are equivalent for RNT PTRS, so that this procedure becomes an AST decision procedure for RNT PTRS too. Also more restrictions might be removed. The first two options we want to explore are removing non-overlappingness via 1-exit recursive markov decision processes (RMDPs) which allow for nondeterminism and removing tail-recursivity via probabilistic pushdown automata (PPDAs), where we could use the stack for remembering nested function calls.

Finally we summarize the main decidability results of this thesis in the following graphic.

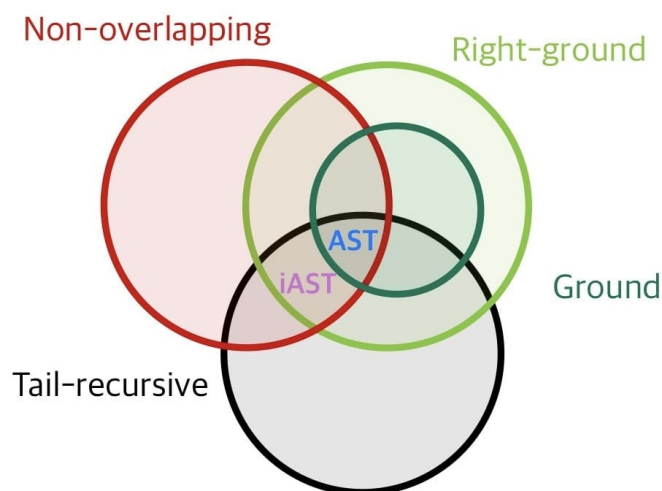


Figure 6.1: (i)AST Decidability Results for PTRS

Bibliography

- [Cho56] N. Chomsky. “Three models for the description of language”. In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 113–124.
- [LT85] P. Lancaster and M. Tismenetsky. *The Theory of Matrices: With Applications*. Computer Science and Scientific Computing. Elsevier Science, 1985.
- [Der87] Nachum Dershowitz. “Termination of rewriting”. In: *Journal of Symbolic Computation* 3.1 (1987), pp. 69–115.
- [BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
- [Col98] George E. Collins. “Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition”. In: *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Ed. by Bob F. Caviness and Jeremy R. Johnson. Vienna: Springer Vienna, 1998, pp. 85–121.
- [KH99] B Knudsen and J Hein. “RNA secondary structure prediction using stochastic context-free grammars and evolutionary history.” In: *Bioinformatics* 15.6 (June 1999), pp. 446–454.
- [AG00] Thomas Arts and Jürgen Giesl. “Termination of term rewriting using dependency pairs”. In: *Theoretical Computer Science* 236.1 (2000), pp. 133–178.
- [HMu01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. “Introduction to automata theory, languages, and computation, 2nd edition”. In: *SIGACT News* 32.1 (Mar. 2001), pp. 60–65.
- [BK02] Olivier Bournez and Claude Kirchner. “Probabilistic Rewrite Strategies. Applications to ELAN”. In: *Rewriting Techniques and Applications*. Ed. by Sophie Tison. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 252–266.
- [BG05] Olivier Bournez and Florent Garnier. “Proving Positive Almost-Sure Termination”. In: *Term Rewriting and Applications*. Ed. by Jürgen Giesl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 323–337.
- [Gie+06] Jürgen Giesl et al. “Mechanizing and Improving Dependency Pairs”. In: *Journal of Automated Reasoning* 37.3 (Oct. 2006), pp. 155–203.

- [GJ08] Dick Grune and Cerial J. H. Jacobs. *Parsing Techniques*. Monographs in Computer Science. New York, NY: Springer, 2008.
- [EY09] Kousha Etessami and Mihalis Yannakakis. “Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations”. en. In: *Journal of the ACM* 56.1 (Jan. 2009), pp. 1–66.
- [Kor+09] Martin Korp et al. “Tyrolean Termination Tool 2”. In: *Rewriting Techniques and Applications*. Ed. by Ralf Treinen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 295–304.
- [CBM13] Stefano Crespi Reghizzi, Luca Breveglieri, and Angelo Morzenti. *Formal Languages and Compilation*. en. Texts in Computer Science. London: Springer London, 2013.
- [Gor+14] Andrew D. Gordon et al. “Probabilistic programming”. In: *Future of Software Engineering Proceedings*. FOSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 167–181.
- [GR14] Jan Gorodkin and Walter L. Ruzzo, eds. *RNA Sequence, Structure, and Function: Computational and Bioinformatic Methods*. en. Vol. 1097. Methods in Molecular Biology. Totowa, NJ: Humana Press, 2014. Chap. 8, pp. 143–162.
- [YKS14] Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. “Nagoya Termination Tool”. In: *Rewriting and Typed Lambda Calculi*. Ed. by Gilles Dowek. Cham: Springer International Publishing, 2014, pp. 466–475.
- [Gie+17] Jürgen Giesl et al. “Analyzing Program Termination and Complexity Automatically with AProVE”. In: *Journal of Automated Reasoning* 58.1 (Jan. 2017), pp. 3–31.
- [ALY20] Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. “On probabilistic term rewriting”. In: *Science of Computer Programming* 185 (2020), p. 102338.
- [GL20] Raúl Gutiérrez and Salvador Lucas. “mu-term: Verify Termination Properties Automatically (System Description)”. In: *Automated Reasoning*. Ed. by Nicolas Peltier and Viorica Sofronie-Stokkermans. Cham: Springer International Publishing, 2020, pp. 436–447.

- [GG23] Krishnendu Ghosh and Julia Goldman. “Formal Analysis of Rewriting System Representing RNA Folding:” en. In: *Proceedings of the 16th International Joint Conference on Biomedical Engineering Systems and Technologies*. Lisbon, Portugal: SCITEPRESS - Science and Technology Publications, 2023, pp. 235–242.
- [KG23] Jan-Christoph Kassing and Jürgen Giesl. “Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs”. In: *Automated Deduction – CADE 29*. Ed. by Brigitte Pientka and Cesare Tinelli. Cham: Springer Nature Switzerland, 2023, pp. 344–364.
- [KDG24] Jan-Christoph Kassing, Stefan Dollase, and Jürgen Giesl. “A Complete Dependency Pair Framework for Almost-Sure Innermost Termination of Probabilistic Term Rewriting”. In: *Functional and Logic Programming*. Ed. by Jeremy Gibbons and Dale Miller. Singapore: Springer Nature Singapore, 2024, pp. 62–80.
- [KFG24] Jan-Christoph Kassing, Florian Frohn, and Jürgen Giesl. “From Innermost to Full Almost-Sure Termination of Probabilistic Term Rewriting”. In: *Foundations of Software Science and Computation Structures: 27th International Conference, FoS-SaCS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6–11, 2024, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, 2024, pp. 206–228.
- [Hua+25] Ruanqianqian Huang et al. *Online Z3 Guide*. Microsoft Research, RiSE. 2025. URL: <https://microsoft.github.io/z3guide/> (visited on 09/23/2025).
- [Ter25] Termination and Complexity Competition Steering Committee. *Termination Competition 2025*. Termination-Portal.org. Sept. 8, 2025. URL: https://termination-portal.org/wiki/Termination_Competition_2025 (visited on 09/23/2025).