Greibach normal form for weighted conext-free grammars

Turkenburg, R.T.C.

R. T. C. Turkenburg

Greibach Normal Form for Weighted Context-Free Grammars

Bachelor thesis

May 10, 2019

Thesis supervisors: dr. M. Bonsangue (LIACS)
dr. P. J.Bruin (MI)

Leiden University
Mathematical Institute
Leiden Institute of Advanced Computer Science

Contents

Chapter 1

Introduction

Formal language theory studies the way in which languages over a given alphabet can be represented. This leads to the definition of grammars and automata which can take many forms, differing in their usefulness and applications. This theory relates to both theoretical and practical fields such as automata theory as part of the study of computation, compiler construction and parsing.

In these studies we use so called grammars. These provide rules which can be used to derive strings in a given language. In this sense a grammar generates a language. A certain class of grammars, the context-free grammars, contain two main types of symbols: the terminals and non-terminals. Terminal symbols are the symbols of the alphabet over which the language is defined. This may be, for example, the set $\{a, b\}$ or the entire Latin alphabet. The non-terminal symbols can be thought of as placeholders which are replaced by terminal symbols during derivations of strings. A full definition of a grammar gives so called production rules, which tell us how non-terminal symbols can be replaced to derive strings.

Also of interest in this area are automata. The most basic form of these are finite automata which contain states and transitions. At any point, the automaton will be in one of its states and each transition gives the state which should be moved to, given a symbol in the input alphabet of the automaton. Such an automaton starts in a certain state and for a given input (a string over the input alphabet of the automaton) it will take the corresponding transitions. Once the input has been processed, the automaton will be in a certain state. Each state is either an accepting or non-accepting state. If the final state is accepting, the automaton is said to accept the input string. Otherwise, the string is not accepted. The set of all strings accepted by the automaton is called the language accepted by the automaton.

Beyond these finite automata, we have many more types of automata such as pushdown automata, which, as well as states and transitions, maintain a stack of symbols. This extra "memory" allows such automata to accept more languages than finite automata. More expressive still are Turing machines, however these are not discussed here. For a comprehensive introduction to automata and the languages they accept, see, for example [Mar11].

The motivation for presenting these automata is their equivalence with different classes of equations or grammars. For example, finite automata accept regular languages which can be represented by regular expressions. Pushdown automata accept the same languages which can be represented by context-free grammars.

The production rules of a grammar have their own syntactical properties and it is useful to study so called normal forms for these grammars. These are syntactical restrictions on the form of the productions. For example, the Chomsky normal form requires that each production replaces a non-terminal with two non-terminals, one terminal or the empty word 1. Another example is the Greibach normal form, which requires that each production replaces a non-terminal with a terminal followed by a string of non-terminals or replaces such a non-terminal with the empty word. These normal forms allow us to create, for example:

- A polynomial algorithm for deciding if a string is contained in the language represented by the given grammar (Chomsky normal form).
- A top-down parser which, for a string in the language generated by the given grammar of length n, will halt at depth n (Greibach normal form).

This study of formal languages has developed into the study of the more general concept of power series. These power series can be seen as possibly infinite sums of variables with coefficients in a given algebraic structure. Here, we will often take this structure to be a semiring as this gives a rich structure to work with. When formalising what we mean by solutions to the systems we will view (automata and corresponding systems of equations) these series occur as the space in which the solutions lie. In the process of developing this formalism, the theory of automata is viewed in the category theoretic framework of coalgebras. As formal languages in the usual sense can be represented as power series, this will give us the opportunity to gain an intuitive understanding of much of this theory in terms of more familiar concepts.

Much of our work in this thesis will be focused on the representation of power series as solutions of systems of equations [Win14]. The most important results will deal with the form of solutions to certain types of these systems, and will be combined into an algorithm for finding a normal form for these systems.

The Greibach Normal Form (GNF) is well known for context-free grammars [Gre65], but will be presented here for more general equations. These equations represent not just formal languages in the usual sense; the solutions are power series over an alphabet with coefficients in a semiring. We will see that when this semiring is the Boolean semiring, the results apply exactly to context-free grammars.

There are already a number of algorithms known for finding the Greibach normal form both specifically for context-free grammars [HU69] and the more general systems of equations [EL02]. The original paper establishing the Greibach normal form [Gre65] also guarantees that every context-free grammar has an equivalent context-free grammar in Greibach normal form. However, finding such a context-free grammar often results in a very large grammar and includes many steps. For equations over other semirings, the known algorithms have similar disadvantages. The algorithm which we present here is simpler and results in a relatively small final system.

In [Win14], there is mention of a number of results combining to give us a means of finding a GNF for a system of equations. However, the details of such a process are not given and this will influence our treatment of the same results. We will fill in these details and hope to give more insight into the development of the algorithm. This also gives us the opportunity to give the steps of the algorithm in explicit detail and discuss how it compares to some existing examples.

We proceed by introducing a number of concepts before presenting our novel algorithm for the GNF. Of interest is the combination of concepts from basic computer science and mathematics in this subject. We will therefore look at basic theory from computer science such as grammars and automata as well as definitions of important algebraic structures such as semirings and some

category theory. We endeavour to provide a brief but clear treatment of the relevant definitions and results while giving references to the literature where this is more appropriate.

Once these concepts have been presented, we will move to definitions and results related to the further systems which are the focus of the GNF. This will include the power series mentioned earlier as well as polynomial systems. These systems are similar to simple automata, however the transition functions take the form of polynomial expressions of the variables (containing no alphabet symbols) with coefficients in an arbitrary semiring.

Polynomial systems are at the core of the GNF, and we will show a correspondence between general systems of equations and polynomial systems, which in turn correspond to systems in GNF. These general systems are a generalisation of context-free grammars in which each production rule may have a coefficient in a given semiring. These systems are more expressive but we do need to take these coefficients into account when transforming systems into GNF.

In the final chapter of this thesis we will use this material to present our novel algorithm for finding the GNF. As discussed, this will be done in much the same way as in [Win14], however for our method of transforming equations we encounter equations which are not fully dealt with in [Win14]. This means that the proofs given in [Win14] do not fully guarantee that our algorithm gives the desired results. We hope to make this intuitively clear with a number of illustrative examples.

Chapter 2

Preliminary definitions

2.1 Algebraic Structures

Throughout this work, we will make use of a number of algebraic structures. The axioms which these must satisfy are well known, but they are presented here for completeness.

Definition 2.1.1. A *monoid* is a set M with a binary operation $\cdot : M \times M \to M$ and an element $1 \in M$ such that

1. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all $a, b, c \in M$
2. $1 \cdot a = a \cdot 1 = a$ for all $a \in M$

Such a structure is called commutative if, furthermore, for all $a, b \in M$ we have $a \cdot b = b \cdot a$.

Example 2.1.2. Two useful examples are $(\mathbb{N}, +, 0)$ the natural numbers with the operation of addition, and $(\mathbb{N}, \cdot, 1)$ the natural numbers with the operation of multiplication. In both these cases the monoids are commutative.

An example of a monoid which is not, in general, commutative is the free monoid over a set X. This is often written as X^* and is the set of all strings over the set X with the operation of string concatenation. For example, for $X = \{a, b\}$ we have $X^* = \{1, a, b, aa, ab, ba, bb, \ldots\}$. The concatenation of two strings $v, w \in X^*$ is then written as vw and is the symbols of v followed by the symbols of w. See also [Mar11].

The following structure will be used often as we will look at weighted grammars in which the weights are given by elements in a semiring.

Definition 2.1.3. A *semiring* is a set S with two binary operations $\cdot : S \times S \to S$ and elements $0, 1 \in S$ such that

1. $(S, +, 0)$ forms a commutative monoid i.e. it is a monoid and $a + b = b + a$ for all $a, b \in S$
2. $(S, \cdot, 1)$ is a monoid
3. The operation $\cdot$ distributes over $+$ from both sides i.e. $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$

4. The element $0 \in S$ annihilates all other elements under multiplication form both sides i.e. $0 \cdot a = a \cdot 0 = 0$

A semiring is commutative if all the elements also commute under the operation $\cdot$.

We will often write simply ab for $a \cdot b$. We see many similarities with a ring, however these have additive inverses and the last axiom is not, strictly speaking, necessary in the case of a ring as this follows from the other axioms.

Example 2.1.4. A trivial example is when $0 = 1$ as in this case we have $a = 1 \cdot a = 0 \cdot a = 0$. The underlying set is then $\{0\}$.

A more interesting example is the natural numbers $\mathbb{N}$ (including 0) with the usual operations of addition and multiplication.

Another example is the Boolean semiring $\mathbb{B}$ which has as its underlying set $\{0, 1\}$ with operations logical or ($\vee$) and and ($\wedge$). This will lead us to weighted grammars which represent the usual context-free languages.

An example of a non-commutative semiring is $(\mathcal{P}(X^*), \cdot, \cup, 1, \{1\})$. Here, $\mathcal{P}(X^*)$ is the powerset of the free monoid over X, also known as the set of all languages over the alphabet X, and we have the operations of string concatenation $\cdot$ and language union $\cup$. For two languages S and T the union ST is defined as the set

$$\{s \cdot t : s \in S, t \in T\}$$

The units for these operations are the empty string 1 and the language containing only the empty string $\{1\}$ respectively.

When we work over a ring we can define a module over an abelian group and a ring. However, here we have a commutative monoid and a semiring.

Definition 2.1.5. Given a commutative monoid M (written additively), a semiring S and an operation (scalar multiplication) $\cdot : S \times M \to M$, these form a *left S-module* if for $r, s \in S$ and $x, y \in M$ we have

1. $r \cdot (x + y) = r \cdot x + r \cdot y$
2. $(r + s) \cdot x = r \cdot x + s \cdot x$
3. $(rs) \cdot x = r \cdot (s \cdot x)$
4. $1_S \cdot x = x$
5. $0_S \cdot x = s \cdot 0_M = 0_M$

Remark. We can also define a right S-module. Here, the semiring works on the monoid from the right. In both cases, we will often leave the $\cdot$ out i.e. rx or xr instead of $r \cdot x$ or $x \cdot r$.

The following structure extends modules with another operation of multiplication, which in this case takes two elements from the algebra and gives another element of the algebra.

Definition 2.1.6. Given an S-module A, with S commutative, and a binary operation $\cdot : A \times A \to A$, this gives an *S-algebra* if for $x, y, z \in A$

1. $x \cdot (y + z) = x \cdot y + x \cdot z$ (Left Distributive)
2. $(x + y) \cdot z = x \cdot z + y \cdot z$ (Right Distributive)
3. $(ax) \cdot (by) = (ab)(x \cdot y)$ (Compatible with scalars)

Also, the S-algebra must contain an element $1 \in A$ such that for any $x \in A$ we have

4. $x \cdot 1 = 1 \cdot x = x$

In other words, an S-algebra must contain a unit element for the operation $\cdot$.

Remark. The first three axioms say that the binary operation is bilinear.

2.2 Brief overview of context-free grammars

The concept of context-free grammars is well known [Mar11] and a full treatment of the subject can be found in many textbooks. We will briefly introduce the subject here to fix the notation for some examples in later chapters.

Definition 2.2.1. Given an alphabet A, a *language* over this alphabet is a subset $L \subseteq A^*$.

Here A^* is the free monoid over A as discussed before.

Definition 2.2.2. A *context-free grammar* G consists of disjoint finite sets X and A where X is the set of *non-terminals* or *variables* and A is the set of *terminal symbols*, also known as the *alphabet*. There is also an $S \in X$, the *start variable*, and a set of *productions* P of the form $x \to \alpha$ with $x \in X$ and $\alpha \in (X \cup A)^*$.

Remark. This means that P is formally a subset of $X \times (X \cup A)^*$ without restriction in this definition.

The concepts we introduce later such as systems of equations will be seen to be extensions of these context-free grammars. In many instances we are also interested in the solution of such a grammar, also called the language generated by the grammar.

Each context-free grammar represents a context-free language in the following way.

Definition 2.2.3. Given a CFG G, *the language* $L(G)$ *generated by* G is

$$L(G) := \{w \in A^* : S \Rightarrow_G^* w\}$$

Here $\Rightarrow_G^*$ is the reflexive transitive closure of the relation $\Rightarrow_G$. This relation $\Rightarrow_G$ means that if we write $\alpha \Rightarrow_G \beta$ there is a production rule which can be used to replace a single occurrence of a variable in α resulting in β with $\alpha, \beta \in (X \cup A)^*$.

Example 2.2.4. We take the context-free grammar G to be $A = \{a, b\}$, $X = \{S\}$ and $P = \{S \to aSb, S \to 1\}$.

Then we can generate a string such as $aabb$ as follows

$$S \Rightarrow_G aSb \Rightarrow_G aaSbb \Rightarrow_G aabb$$

meaning that $aabb \in L(G)$.

Chapter 3

Coalgebras and Power Series

We first present a framework to define automata. These automata are a means to represent certain systems of equations and we will see the different representations of these systems. Our interest will mainly be in solutions to these systems and the correspondence with the Greibach Normal Form which will be presented in later chapters. This framework is in the language of category theory.

Further, we present some basic theory of power series. These are of interest here as they play an important role as the space in which solutions of equations are found. This is also viewed in the framework of category theory as the final object in the category of automata.

We then move to polynomials and systems in which the equations are of a polynomial form. These are of particular importance in the context of the Greibach Normal Form as these systems are in one-to-one correspondence to systems in this GNF.

3.1 Coalgebras

The following gives a category-theoretic means to view automata.

Definition 3.1.1. Given a category $\mathcal{C}$ and a functor $F : \mathcal{C} \to \mathcal{C}$, a *coalgebra* is a pair (A, α) with $A \in \mathcal{C}$ an object and $\alpha : A \to FA$ a morphism.

Definition 3.1.2. A *coalgebra homomorphism* between (A, α) and (B, β) is a morphism (of $\mathcal{C}$)

$$f : A \to B$$

such that the following diagram commutes

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\beta} \\ FA & \xrightarrow{Ff} & FB \end{array}$$

Remark. We see that for a functor F, coalgebras for F together with morphisms form a category.

3.2 Automata

Coalgebras can be used to define several types of automata. We first examine the most basic type.

Definition 3.2.1. Given a semiring S, a finite alphabet A and a set of states Q, an *S-automaton* is a triple (Q, o, δ) such that

1. $o : Q \to S$ is the output function
2. $\delta : Q \to Q^A$ is the transition function

We see that this defines a coalgebra for the **Set**-endofunctor $S \times (-)^A$ with the morphism given by (o, δ). This is shown in the following diagram:

$$\begin{array}{c} Q \\ {\scriptstyle (o,\delta)}\downarrow \\ S \times Q^A \end{array}$$

Remark. For $q \in Q$ and $a \in A$ we use the notation q_a for the image $\delta(q, a)$.

We may also extend the transition function to apply to $w \in A^*$. We call this extended function δ^* and it is defined inductively by

$$\delta^*(q, aw) = \delta^*(\delta(q, a), w)$$

Remark. For this new function we use q_w to refer to the image $\delta^*(q, w)$.

The image $\delta(q, a) = q_a$ is also called the derivative of q to a. The motivation for this name should become clear when we examine the same operations for power series later on.

Example 3.2.2. We give an example of such a simple automaton in a number of forms where S is taken to be the Boolean semiring. The first way is in terms of the functions defined above. We have

$$\begin{aligned} &q_{0a} = q_1 \quad q_{0b} = q_0 \\ &q_{1a} = q_2 \quad q_{1b} = q_0 \\ &q_{2a} = q_2 \quad q_{2b} = q_0 \\ &o(q_0) = o(q_1) = 0 \quad o(q_2) = 1 \end{aligned}$$

We can also represent this system in the form of a diagram. In this case such a diagram is as follows

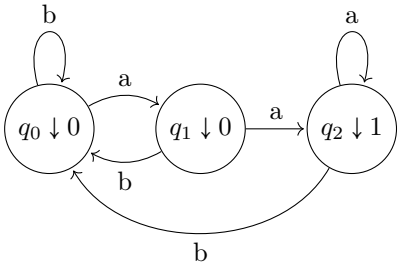


Figure 3.1: A simple automaton

Here the notation $q_i \downarrow s$ is used to mean $o(q_i) = s$ and we have an arrow $q_i \xrightarrow{a} q_j$ between states iff $\delta(q_i, a) = q_j$. In the context of formal languages, a state with output value 0 is said to be non-accepting where a state with output 1 is said to be accepting. Often one state is also specified as the initial state. If we do this for the automaton above, the solution for the state q_0 (also called the language accepted by the automaton when q_0 is the initial state) is that of the strings ending in `aa`. In this text we often do not indicate such an initial state as we are interested in the solutions to all states in a system. What these solutions are will be dealt with later and we will then revisit this example.

3.3 Power Series

When looking at automata we want to find solutions, a generalisation of the language generated by an automaton, which we will define precisely later. To this end, we first define the set of power series. The definition can be given in terms of functions or infinite sums. The former is often useful when viewing automata as coalgebras, as applying a function to a power series is then as simple as function composition. The latter is given here mainly for clarification that these can be written as infinite sums as may be usual in other texts.

Definition 3.3.1. Given a semiring S and an alphabet A, the *set of power series over* A is given by

$$S\langle\langle A\rangle\rangle := \{f : A^* \to S\}$$

or using formal sum notation

$$S\langle\langle A\rangle\rangle := \left\{ \sum_{w \in A^*} s_w w \right\}$$

Remark. Here, the set A^* is the set of words (monomials) over the alphabet A. It is important to note, the elements of A do not commute i.e. the monomials ab and ba over the alphabet $A = \{a, b\}$ are not equal.

Definition 3.3.2. The subset $S\langle\langle A\rangle\rangle_0$ of power series is given by

$$S\langle\langle A\rangle\rangle_0 := \{\sigma \in S\langle\langle A\rangle\rangle : \sigma(1) = 0\}$$

Remark. This is the subset of power series whose constant term is equal to zero.

It should be clear that we have embeddings of both S and A^* into the subset of power series. We have $\iota_S : S \to S\langle\langle A \rangle\rangle$ defined by

$$\iota_S(s)(w) = \begin{cases} s, & \text{if } w = 1, \\ 0, & \text{else.} \end{cases}$$

For A^* we have $\iota_{A^*} : A^* \to S\langle\langle A \rangle\rangle$ given by

$$\iota_{A^*}(v)(w) = \begin{cases} 1, & \text{if } w = v, \\ 0, & \text{else.} \end{cases}$$

These definitions allow us to view subsets of S, A^* and even A as subsets of $S\langle\langle A \rangle\rangle$ and we will usually do this without using the above embeddings explicitly.

We can also define a number of operations on the power series in $S\langle\langle A \rangle\rangle$. Given a semiring S we define the following for $w \in A^*$ and $\sigma, \tau \in S\langle\langle A \rangle\rangle$

$$\begin{aligned} 0(w) &= 0 \\ 1(w) &= \begin{cases} 1, & \text{if } w = 1, \\ 0, & \text{otherwise.} \end{cases} \\ (\sigma + \tau)(w) &= \sigma(w) + \tau(w) \\ (\sigma \cdot \tau)(w) &= \sum_{uv=w} \sigma(u)\tau(v) \end{aligned}$$

In the case that S is the Boolean semiring, the last two properties coincide with the usual operations on languages of concatenation and union.

It is easy to verify that this defines a semiring structure on $S\langle\langle A \rangle\rangle$. We can furthermore define left and right scalar products ($\cdot_l$ and $\cdot_r$ respectively) as follows for $s \in S$ and $\sigma \in S\langle\langle A \rangle\rangle$

$$\begin{aligned} (s \cdot_l \sigma)(w) &= s(\sigma(w)) \\ (\sigma \cdot_r s)(w) &= (\sigma(w))s \end{aligned}$$

Again, we can verify that these operations define a left and right S-module structure respectively on $S\langle\langle A \rangle\rangle$. Finally, if S is commutative, the combination of these operations can be shown to define an S-algebra structure on $S\langle\langle A \rangle\rangle$. This will become important in our treatment of polynomial systems and their link to the Greibach normal form.

We will now look towards solutions of such automata. The results in the next chapter will not use directly the first treatment of such solutions, rather, there are a number of extensions which will be applied in these cases.

Definition 3.3.3. The *automaton over S of power series* is given by $(S\langle\langle A \rangle\rangle, O, \Delta)$ where for $\sigma \in S\langle\langle A \rangle\rangle$ and $w \in A^*$ we have

$$O(\sigma) = \sigma(1) \qquad \text{and} \qquad \sigma_a(w) = \sigma(aw)$$

In terms of infinite sums, this expresses that the constant term of such a series is zero and that if we can write σ as

$$\sum_{w \in A^*} s_w w$$

then σ_a can be written as

$$\sum_{w=av} s_w v$$

In an earlier chapter, the term derivative was used for q_a and q_w without clear motivation. The next result should provide this motivation while also providing important rules for the manipulation of equations which include power series.

Proposition 3.3.4. *For all $a, b \in A$, $s \in S$ and $\sigma, \tau \in S\langle\langle A \rangle\rangle$ we have*

$$\begin{aligned} O(s) &= s \quad s_a = 0 \\ O(b) &= 0 \quad b_a = \begin{cases} 1, & \text{if } b = a, \\ 0, & \text{otherwise.} \end{cases} \\ O(\sigma + \tau) &= O(\sigma) + O(\tau) \quad (\sigma + \tau)_a = \sigma_a + \tau_a \\ O(\sigma \cdot \tau) &= O(\sigma) \cdot O(\tau) \quad (\sigma \cdot \tau)_a = \sigma_a \cdot \tau + O(\sigma) \cdot \tau_a \end{aligned}$$

Remark. This proposition makes the use of the term derivative clearer as we have the familiar sum rule and a slightly altered product rule.

Proof. We first give the proof of one of the very simple statements, namely the sum rule $(\sigma + \tau)_a = \sigma_a + \tau_a$. We have for $w \in A^*$

$$\begin{aligned} (\sigma + \tau)_a(w) &= (\sigma + \tau)(aw) \\ &= \sigma(aw) + \tau(aw) \\ &= \sigma_a(w) + \tau_a(w) \end{aligned}$$

Now, we also give the proof for the product rule as the rest are also very simple. Again with $w \in A^*$ we have

$$\begin{aligned} (\sigma \cdot \tau)_a(w) &= (\sigma \cdot \tau)(aw) \\ &= \sum_{uv=aw} (\sigma(u) \cdot \tau(v)) \\ &= \left(\sum_{st=w} \sigma(as) \cdot \tau(t) \right) + \sigma(1) \cdot \tau(aw) \\ &= \left(\sum_{st=w} \sigma_a(s) \cdot \tau(t) \right) + \sigma(1) \cdot \tau_a(w) \\ &= (\sigma_a \cdot \tau)(w) + O(\sigma) \cdot \tau_a(w) \\ &= (\sigma_a \cdot \tau + O(\sigma) \cdot \tau_a)(w) \end{aligned}$$

Here the third equality follows as, in each term of the sum, either a is part of the string u or in one term u is the empty string 1 and a must be part of v. □

The next result is very similar to a fundamental theorem of calculus, which can be used to write a power series in terms of its derivatives.

Proposition 3.3.5. *For a power series $\sigma \in S\langle\langle A \rangle\rangle$ we have*

$$\sigma = O(\sigma) + \sum_{a \in A} a\sigma_a$$

Proof. See [Win14, Proposition 2.12]. □

This result will, in fact, be used to show the correspondence between polynomial systems and systems in GNF.

We now have enough theory to formalise what we mean by a solution for an automaton Q. This incorporates our treatment of automata as coalgebras with the definition of the power series automaton. In many cases, we will not be interested in explicitly finding solutions. However, it is very important that such solutions exist and that they are unique.

Proposition 3.3.6. *The automaton of power series is a final object in the category of automata. In other words, for any automaton (Q, o, δ) there is a unique morphism of automata $[\![-]\!] : Q \to S\langle\!\langle A \rangle\!\rangle$.*

Proof. We define the mapping $[\![-]\!] : Q \to S\langle\!\langle A \rangle\!\rangle$ as follows

$$[\![q]\!](w) := o(q_w)$$

We must now show that $O([\![q]\!]) = o(q)$ and $[\![q_a]\!] = [\![q]\!]_a$ for all $q \in Q$ and $a \in A$. For the first part we have

$$O([\![q]\!]) = [\![q]\!](1) = o(q_1) = o(q)$$

For the second part we have

$$[\![q_a]\!](w) = o(q_{aw}) = [\![q]\!](aw) = [\![q]\!]_a(w)$$

for $w \in A^*$.

To show that the mapping is unique we suppose there is a morphism $h : Q \to S\langle\!\langle A \rangle\!\rangle$. Then we see for $w \in A^*$

$$h(q)(w) = O(h(q)_w) = O(h(q_w)) = o(q_w) = [\![q]\!](w)$$

So the morphism $[\![-]\!]$ is indeed unique. □

This proposition thus gives meaning to solutions of such systems. Namely, for a state $q \in Q$ the solution is $[\![q]\!] \in S\langle\!\langle A \rangle\!\rangle$. We will now revisit the earlier example 3.2.2.

Example 3.2.2 (continuing from p. 10)**.** In the automaton representation, the strings which are accepted correspond with paths from initial state to accepting state. The string is built by appending the alphabet symbol on each edge taken. To see the correspondence with the definition of solutions given above we find the solution of the state q_0. In fact, what we do here is check if certain strings are in the language represented by the given system. This is the case if the coefficient of such a string in the power series is 1.

We have for the string `baa`:

$$[\![q_0]\!](baa) = o(q_{0baa}) = o(q_{0aa}) = o(q_{1a}) = o(q_2) = 1$$

So we see, as expected, that this string is in the language.

For the string `bab` we have:

$$[\![q_0]\!](bab) = o(q_{0bab}) = o(q_{0ab}) = o(q_{1b}) = o(q_0) = 0$$

So this string is not in the language.

Clearly, we can not find the entire power series in this manner as this is infinite. However, there are means (such as bisimulation) to prove the form of the solution which we do not present here.

The following definition presents the Kleene star operator used frequently in formal language theory. It is somewhat more general than this and we will discuss in what ways this is the case.

Definition 3.3.7. For $\sigma \in S\langle\langle A \rangle\rangle_0$ we define the power series σ^* as the solution to the equation

$$x = 1 + \sigma x$$

or equivalently the equation

$$x = 1 + x\sigma$$

Remark. To show that this series exists and is unique, it is necessary to define this in terms of limits. We do present this (see example 3.3.8), however this is also not completely formalised. This would require many definitions which are not necessary for the rest of the results we discuss here, and some of these are beyond the scope of this text.

It follows that $(\sigma^*)_a = \sigma_a\sigma^*$ and $o(\sigma^*) = 1$ as

$$(\sigma^*)_a = (1 + \sigma\sigma^*)_a = \sigma_a\sigma^* + o(\sigma)\sigma_a = \sigma_a\sigma^*$$

and

$$o(\sigma^*) = o(1) + o(\sigma) \cdot o(\sigma^*) = 1$$

Although the following example is not completely formal, it should provide some intuition for those familiar with the Kleene star operator.

Example 3.3.8. For a power series consisting of an alphabet symbol a or a string of such symbols w we have the expected result that

$$a^* = \sum_{i=0}^{\infty} a^i$$

and

$$w^* = \sum_{i=0}^{\infty} w^i$$

To formalise this fully we would need to define such infinite sums and use a technique such as bisimulation to prove that these equalities indeed hold.

Lemma 3.3.9 (Arden's Lemma)**.** *For $\sigma \in S\langle\langle A \rangle\rangle$ and $\tau \in S\langle\langle A \rangle\rangle_0$ the equations*

$$x = \sigma + \tau x \qquad \text{and} \qquad x = \sigma + x\tau$$

have the unique solutions

$$x = \tau^*\sigma \qquad \text{and} \qquad x = \sigma\tau^*$$

respectively.

Proof. (sketch) For the first equation we replace x on the right hand side with the given solution to get:

$$\begin{aligned} \sigma + \tau\tau^*\sigma &= (1 + \tau\tau^*)\sigma \\ &= \tau^*\sigma \end{aligned}$$

This equality holds due to the definition of τ^*.

The proof for the other equation is analogous. □

Remark. This is merely a sketch as this does not show the uniqueness of these solutions. To do this we would need a technique such as bisimulation to show that another candidate is equal to $\tau^*\sigma$ or $\sigma\tau^*$. This difficulty arises from the lack of additive and multiplicative inverses which restrict how we may manipulate the given equations.

Arden's lemma is an exceptionally important result which will be used in a number of situations to give solutions to systems we are working with. Later, when working with polynomials over the variables, we will be able to apply this lemma by viewing variables as power series. This can be done as the solution for a given variable is of course a power series.

Definition 3.3.10. The *set of S-rational power series over a set* $\Sigma \in S\langle\langle A\rangle\rangle$ is the smallest subset $S_{rat(\Sigma)}\langle\langle A\rangle\rangle \subseteq S\langle\langle A\rangle\rangle$ such that

- $\Sigma \subseteq S_{rat(\Sigma)}\langle\langle A\rangle\rangle$
- $A \subseteq S_{rat(\Sigma)}\langle\langle A\rangle\rangle$ and $S \subseteq S_{rat(\Sigma)}\langle\langle A\rangle\rangle$
- $S_{rat(\Sigma)}\langle\langle A\rangle\rangle$ is closed under $\cdot$ and $+$
- for $\sigma \in S_{rat(\Sigma)}\langle\langle A\rangle\rangle \cap S\langle\langle A\rangle\rangle_0$ also $\sigma^* \in S_{rat(\Sigma)}\langle\langle A\rangle\rangle$

This definition will be used in a number of the proofs which will lead us to the construction of the Greibach normal form. In many cases the exact structure of such S-rational power series will not be important; it is of most interest that these rational series give rise to the polynomial systems presented later.

For a more comprehensive treatment of coalgebras, automata, power series and further related concepts see, for example, [Rut00] and [Rut03].

Chapter 4

Streams

4.1 Introduction to Streams

Now that we have defined power series we will take a brief look at the concept of streams. Although these will not be used extensively in the rest of this text, they provide a number of very interesting examples which will hopefully help to give a better understanding of the broader concept of power series.

It should be noted that streams themselves are a large area of study and are of interest in a number of fields, they are a primitive data structure in programming languages such as Haskell.

Definition 4.1.1. A *stream* is an element of $S\langle\langle A\rangle\rangle$ where we take A to be a single element set. For convenience we will take the set $\{\Theta\}$.

What we have is therefore $S\langle\langle\{\Theta\}\rangle\rangle$ which, by definition is the set of functions

$$\{f : \{\Theta\}^* \to S\}$$

We can easily see that this is in bijection to the set

$$\{f : \mathbb{N} \to S\} =: S^{\mathbb{N}}$$

as we have the bijection $N : S\langle\langle\{\Theta\}\rangle\rangle \to S^{\mathbb{N}}$ given by

$$N(f)(n) = f(\Theta^n)$$

This means that streams can also be seen as infinite sequences of elements of S.

As we have seen before, $S\langle\langle A\rangle\rangle$ can be given an automaton structure. It is then of interest what form such an automaton takes in the case of streams. For the output function O we have for $\sigma \in S\langle\langle\{\Theta\}\rangle\rangle$

$$O(\sigma) = \sigma(0) = \sigma(\Theta^0)$$

Therefore, in the infinite sequence interpretation, this corresponds with the first element of such a sequence. We now examine the derivative of streams. Again with $\sigma \in S\langle\langle\{\Theta\}\rangle\rangle$ we see that

$$\sigma_\Theta(\Theta^n) = \sigma(\Theta^{n+1})$$

Thus, taking the derivative of a stream gives us the original stream shifted by one element i.e. we start at the second element.

Remark. As derivatives are always taken to the only element in the alphabet we will often write σ' for the derivative of σ, instead of σ_Θ.

When looking at streams through the bijection to $S^{\mathbb{N}}$, the output and transition functions are often called **head** and **tail** respectively as the former gives the first element of a stream and the latter gives the rest. These are then written as

$$\mathbf{head}(\sigma) = \sigma(0) \quad \mathbf{tail}(\sigma)(n) = \sigma(n+1)$$

This gives an isomorphism of automata

$$(S\langle\!\langle\{\Theta\}\rangle\!\rangle, O, \Delta) \cong (S^{\mathbb{N}}, \mathbf{head}, \mathbf{tail})$$

in the expected way.

Example 4.1.2. We take $S = \mathbb{N}$ so that we have infinite sequences of natural numbers. An extremely basic example of a stream is given by the automaton

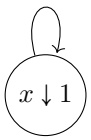


Figure 4.1: A simple automaton representing a stream

which represents the equations

$$o(x) = 1 \quad x' = x$$

The solution to this system can be found in the following way

$$[\![x]\!](n) = x^{(n)}(0) = x(0) = 1$$

where the notation $x^{(n)}$ is used for the n-th derivative of x. This means that the system's solution is the sequence $1, 1, 1, \ldots$ over $\mathbb{N}$.

It is not difficult to see that changing the value of $o(x)$ to some other value $n \in \mathbb{N}$ gives us the constant infinite sequence with value n.

This is perhaps not a very interesting example and does not illustrate fully the power of systems with a single alphabet symbol. The next example should make this clearer.

Example 4.1.3. We again take $S = \mathbb{N}$ and have the following automaton representation

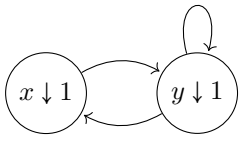


Figure 4.2: An automaton representing a more interesting stream

This represents the system

$$o(x) = o(y) = 1$$
$$x' = y$$
$$y' = x + y$$

where we define that the values for different paths of the same length in the automaton diagram should be summed.

In this case, it is not as easy to find the solution to this system as in the previous example. What we can do is examine a number of values for the sequence given by the variable x. Clearly we have $x(0) = 1$ and $x(1) = y(0) = 1$ so the sequence starts with $1, 1$. Then we have

$$x(2) = y(1) = x(0) + y(0) = 1 + 1 = 2$$
$$x(3) = y(2) = x(1) + y(1) = 1 + 2 = 3$$
$$x(4) = y(3) = x(2) + y(2) = 2 + 3 = 5$$
$$\vdots$$

One can show that the system arising as the solution of the state x is in fact the Fibonacci sequence.

For y we get the sequence starting $1, 2, 3, 5$ which is simply the same sequence as x but starting from the second element. This can also be seen from the correspondence between the derivative and the function **tail**.

Chapter 5

Algebraic Languages

5.1 Polynomial Systems

Polynomial systems are an extension of the automata presented in the previous section and will be shown to have a connection to the Greibach normal form in the next chapter. To define this system we will need a number of other definitions.

Definition 5.1.1. For a set of variables X, the *set of polynomials over* X is the set

$$S\langle X\rangle := \{f : X^* \to S \mid f(w) \neq 0 \text{ for finitely many } w \in X^*\}$$

or equivalently

$$S\langle X\rangle := \left\{ \sum_{w \in X^*} s_w w \mid s_w \neq 0 \text{ for finitely many } w \right\}$$

This is similar to the definition of power series but the number of terms is finite. We can also say that a polynomial f is a function from X^* to S with finite support where the support $\mathrm{supp}(f)$ is defined as the set $\{w \in X^* : f(w) \neq 0\}$.

Definition 5.1.2. For a finite alphabet A and a semiring S, a *polynomial system* is a triple (X, o, δ) such that

1. X is the set of variables
2. $o : X \to S$ is the output function
3. $\delta : X \to S\langle X\rangle^A$ is the transition function, giving for each $x \in X$ and $a \in A$ a polynomial over the set of variables X

We see that this definition expresses that a polynomial system forms a coalgebra for the functor $S \times S\langle -\rangle^A$ as shown in the following diagram.

$$\begin{array}{c} X \\ {\scriptstyle (o,\delta)}\downarrow \\ S \times S\langle X\rangle^A \end{array}$$

It is now natural to look at solutions to these systems. To do this we first look at a general result concerning the universality of the polynomials. In the language of category theory it can also be expressed as $S\langle X\rangle$ being the free S-algebra over the set X.

Lemma 5.1.3. *For an S-algebra M, let $\eta : X \to S\langle X\rangle$ be defined by*

$$\eta(x)(w) = \begin{cases} 1, & \text{if } x = w, \\ 0, & \text{else.} \end{cases}$$

and let $f : X \to M$ be a function. Then there exists a unique S-algebra morphism $f^{\#} : S\langle X\rangle \to M$ such that we have the following commutative diagram

$$\begin{array}{ccc} X & \xrightarrow{\eta} & S\langle X\rangle \\ {\scriptstyle f}\downarrow & \swarrow_{f^{\#}} & \\ M & & \end{array}$$

where the dashed arrow indicates that $f^{\#}$ is the unique S-algebra morphism extending f.

Proof. For $f^{\#}$ to have the desired properties (of an S-algebra morphism) we must have

$$\begin{aligned} f^{\#}\left(\sum_w s_w w\right) &= \sum_w s_w f^{\#}(w) \\ &= \sum_{w=x_{w1}\cdots x_{wn}} s_w f^{\#}(x_{w1}\cdots x_{wn}) \\ &= \sum_{w=x_{w1}\cdots x_{wn}} s_w f^{\#}(x_{w1})\cdots f^{\#}(x_{wn}) \\ &= \sum_{w=x_{w1}\cdots x_{wn}} s_w f(x_{w1})\cdots f(x_{wn}) \end{aligned}$$

where the first two equalities follow from $f^{\#}$ being an algebra morphism and the last equality follows from commutativity and $f^{\#}$ being the extension of f and thus equal on elements in X.

This shows that an S-algebra morphism extending f must have the form of $f^{\#}$ if such an extension exists. We must therefore show that $f^{\#}$ is indeed an S-algebra morphism.

First we have for $p \in S\langle X\rangle$ and $k \in S$

$$\begin{aligned} f^{\#}(kp) &= f^{\#}\left(\sum_w k s_w w\right) \\ &= \sum_{w=x_{w1}\cdots x_{wn}} k s_w f(x_{w1}\cdots x_{wn}) \\ &= k \sum_{w=x_{w1}\cdots x_{wn}} s_w f(x_{w1}\cdots x_{wn}) \\ &= k f^{\#}(p) \end{aligned}$$

Next, we see for $p_1, p_2 \in S\langle X\rangle$ that

$$\begin{aligned} f^{\#}(p_1 + p_2) &= f^{\#}\left(\sum_w (s_w + t_w)\, w\right) \\ &= \sum_{w=x_{w1}\cdots x_{wn}} (s_w + t_w)\, f(x_{w1})\cdots f(x_{wn}) \\ &= \sum_{w=x_{w1}\cdots x_{wn}} s_w f(x_{w1})\cdots f(x_{wn}) + \sum_{w=x_{w1}\cdots x_{wn}} t_w f(x_{w1})\cdots f(x_{wn}) \\ &= f^{\#}(p_1) + f^{\#}(p_2) \end{aligned}$$

Finally, we have for $p_1, p_2 \in S\langle X\rangle$

$$\begin{aligned} f^{\#}(p_1 \cdot p_2) &= f^{\#}\left(\sum_w \left(\sum_{uv=w} s_u \cdot t_v\right) w\right) \\ &= \sum_w \left(\sum_{uv=w} s_u \cdot t_v\right) f^{\#}(w) \\ &= \sum_w \left(\sum_{uv=w} s_u \cdot t_v\right) f^{\#}(u) \cdot f^{\#}(v) \\ &= \left(\sum_u s_u f^{\#}(u)\right) \cdot \left(\sum_v t_v f^{\#}(v)\right) \\ &= f^{\#}(p_1) \cdot f^{\#}(p_2) \end{aligned}$$

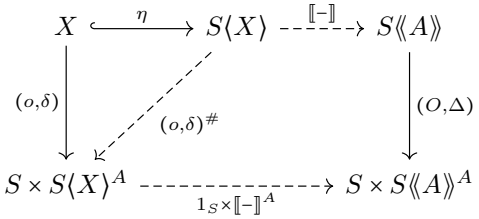
Thus, $f^{\#}$ is an S-algebra morphism.

□

5.2 Solutions

We now show that polynomial systems, just as the simple automata considered earlier, have unique solutions in $S\langle\langle A\rangle\rangle$. Due to the product we use for the polynomials and power series, it is not possible to use lemma 5.1.3 directly, as we can not easily define the necessary S-algebra structure with this convolution. We do however use a slightly weaker form of lemma 5.1.3 which applies to S-modules, where we do require that S is commutative as this makes the resulting final mapping an S-algebra morphism. We also use the final mapping for automata from section 3.2 to give us the following result.

Proposition 5.2.1. *For a polynomial system (X, o, δ) the following diagram commutes*



Proof. (outline) The function $(o, \delta)^{\#}$ can be defined in two steps. First, we extend (o, δ) inductively to a mapping on X^* via the sum and product rules, similar to how we proved the existence of

the earlier morphisms. Then, we extend this using the weaker form of lemma 5.1.3 applied to S-modules to the mapping $(o,\delta)^{\#}$. As mentioned earlier, we can not use the universal mapping together with an algebra structure on $S \times S\langle X\rangle^A$ due mainly to the product we use here. Now $(S\langle X\rangle, (o,\delta)^{\#})$ (where we can view $(o,\delta)^{\#}$ as two separate functions $o^{\#}$ and $\delta^{\#}$) forms an automaton which has unique solutions in $S\langle\langle A\rangle\rangle$. □

Remark. This gives unique solutions for polynomial systems, namely $[\![\eta(x)]\!]$ for $x \in X$.

One can show that the mapping $[\![-]\!]$ above is an S-algebra morphism. This may aid in finding solutions for variables in terms of their derivatives.

The next lemma gives us the guarantee of solutions to equations of a certain form. It may not be obvious why we are interested in this form, however we will see that when manipulating systems into GNF, we find these types of equations. Fortunately, the proof of the lemma also gives us a construction of such solutions and illustrates the importance of Arden's lemma (lemma 3.3.9).

Lemma 5.2.2. *For $k \in \mathbb{N}$, if for all $i, j \leq k$ we have $r_{ij} \in S_{rat(X)}\langle\langle A\rangle\rangle \cap S\langle\langle A\rangle\rangle_0$ and $p_i \in S_{rat(X)}\langle\langle A\rangle\rangle$ for some set $X \subseteq S\langle\langle A\rangle\rangle$, then the system of equations given by*

$$x_i = p_i + \sum_{j=0}^{k} x_j r_{ij}$$

for $i \in \{0,\dots,k\}$ has a unique solution and the solutions x_i are in $S_{rat(X)}\langle\langle A\rangle\rangle$ for all i.

Proof. (sketch) The proof is by induction on k.

For the base case $k = 0$ we have the single equation

$$x_0 = p_0 + x_0 r_{00}$$

and Arden's lemma gives the solution $x_0 = (r_{00})^* p_0$ which is in $S_{rat(X)}\langle\langle A\rangle\rangle$ by the definition of rational series.

For systems with $k > 0$ equations we again use Arden's lemma to solve one of the equations, thereby reducing the number of equations by one and allowing the use of the induction hypothesis. The details can be seen in [Win14, Lemma 2.22] except that the x_j are on the left in all terms of the sum. This means the proof is analogous to that given in [Win14] where we use the relevant version of Arden's lemma. □

Remark. This proof shows how we can use Arden's lemma to find solutions to such a system. This should be kept in mind as this technique will be used in our algorithm for finding the GNF.

Chapter 6

Greibach Normal Form

We now come to the main results of this work. These results concern more general systems compared to those presented so far. These systems also have their solutions defined differently, but what we will see is that the systems and therefore solutions are in fact in correspondence with the polynomial systems defined in the previous chapter.

For the final result of the chapter, which provides the construction of the Greibach normal form, we will need lemma 5.2.2 along with a correspondence between polynomial systems and sets of S-rational series. The latter is also presented in [Win14], however we try to present this result in the context of the GNF to show more clearly how this is used to construct such a normal form.

The final construction of the GNF is thus based on results from [Win14], in which this construction is briefly mentioned. The main aim is to fully develop this treatment into an algorithm for finding a GNF as this is sadly not provided by Winter.

It is important to note that in this section when we refer to a semiring S, this semiring is assumed to be commutative. This facilitates a number of our definitions and results.

Definition 6.0.1. A *system of equations* over an alphabet A is a pair (X, p) with X a finite set of variables and p a function $p : X \to S\langle X + A\rangle$.

Remark. It may not be immediately clear why such a pair would give rise to equations, however we often view such a system as the equations $x = p(x)$ for $x \in X$. The motivation for a description using functions will become clearer when defining solutions of such systems.

Definition 6.0.2. A system of equations is in *Greibach Normal Form* if for all $x \in X$ we have

$$p(x)(w) \neq 0 \iff w \in AX^* \cup \{1\}$$

This definition says that each non-constant term of a polynomial in the system must start with an alphabet symbol followed by a product of variables (with, of course, an element in S preceding each term). This means that a system such as

$$x_1 = 2ax_1x_2 + bx_2 + a$$
$$x_2 = 3bx_1 + b$$

where $A = \{a, b\}$, $X = \{x_1, x_2\}$ and $S = \mathbb{N}$ is in GNF, but a system such as

$$x_1 = 2x_1x_2 + bx_2 + 1$$
$$x_2 = 3x_1 + b$$

is not because, for example, the first term in the equation for x_1 does not start with an alphabet symbol. Here the sets A, X and S are the same as the first example.

Systems in GNF are of interest in a number of fields. One of these is in the theory of compilers where the fact that GNF gives a grammar with no left recursion means that it is easy to find a top-down parser for the corresponding grammar. In this application we are using the case where $S = \mathbb{B}$ i.e. when the systems are context-free grammars.

We will now examine how such systems in GNF correspond to polynomial systems as defined earlier.

Let p be a system of equations in Greibach normal form. Then for each $x \in X$ we have that $p(x)$ is of the form $p(x) = \sum_{\substack{a \in A \\ w \in X^*}} s_{aw} aw + s_1 \cdot 1$. Then for each $b \in A$ the b-derivative of x is

$$x_b = \sum_{\substack{a=b \\ w \in X^*}} s_{aw} w$$

and we take $o(x) = s_1$ which is a polynomial system.

For the other way, we let (X, o, δ) be a polynomial system. Then we have

$$p(x) = \sum_{a \in A} a x_a + o(x)$$

which is a system of equations as each polynomial in $S\langle X \rangle$ is also an element of $S\langle X + A \rangle$.

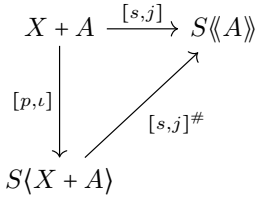
Definition 6.0.3. A system of equations is called *proper* if the following equalities hold

$$p(x)(1) = 0 \qquad p(x)(y) = 0 \text{ for } y \in X$$

Remark. For a standard grammar this expresses a lack of empty word and unit productions. Empty word productions are those of the form $X \to 1$ and unit productions have just a single variable on the right hand side e.g. $X \to Y$.

For polynomial systems, we already have the definition of a solution. Now we also require such a definition for the systems of equations just defined. These solutions will again be found in $S\langle\langle A \rangle\rangle$, but we do not have the same mapping as for polynomial systems as there is no automaton structure in this case.

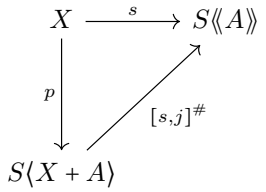
Definition 6.0.4. A *solution* to a system of equations is a function $s : X \to S\langle\langle A \rangle\rangle$ such that the following diagram commutes:



Here, the function ι is the embedding of A into $S\langle X + A \rangle$ and j the embedding of A into $S\langle\langle A \rangle\rangle$. Furthermore, the function $[s, j]^{\#}$ arises as the unique extension of $[s, j]$ as given in lemma 5.1.3. The use of this result is one of the cases in which the commutativity of S is important.

Remark. The idea here is that if we "fill in", for each variable $x \in X$, the value $s(x)$ for all instances of x in the equations $x = p(x)$, all equalities in terms of power series must hold.

We may also provide this definition in terms of simply the functions s and p without explicit reference to ι and j as these are fixed. We then have the diagram



Example 6.0.5. For the system with $X = \{x, y\}$ and $A = \{a\}$ given by

$$x = ay$$
$$y = a$$

the solution $s : X \to S\langle\langle A \rangle\rangle$ is given by

$$s : x \mapsto aa$$
$$y \mapsto a$$

and we see that when we "fill in" these values we get the equations

$$aa = aa$$
$$a = a$$

which clearly hold.

Definition 6.0.6. A solution of a system of equations is called *strong* when $O(s(x)) = 0$ for all $x \in X$. Using the earlier notation we may also write $s(x) \in S\langle\langle A \rangle\rangle_0$ for all $x \in X$.

Example 6.0.7. For an example of a strong solution we can simply take the solution to the previous solution in example 6.0.5 as $aa \in S\langle\langle A \rangle\rangle_0$ and $a \in S\langle\langle A \rangle\rangle_0$.

Proposition 6.0.8. *Let S be a commutative semriring. If there is a finite set $\Sigma \in S\langle\langle A \rangle\rangle$ with $\sigma \in \Sigma$, such that for all $\tau \in \Sigma$ and $a \in A$, the derivative τ_a is S-rational over the set Σ then there is a polynomial system whose solution is the power series σ.*

Proof. The proof is given as part of the proof of [Win14, Proposition 3.16].

The idea is that such a set can be turned into an S-automaton where the set of states are the rational series over the set Σ. To make this polynomial, we add states which represent the starred parts in the original states. □

The proof in [Win14] also proves the implication in the other direction, however we do not need this here.

We add here that for the new variables for starred subexpressions we must also add equations which leave solutions unchanged. We know that for a starred subexpression t^*

$$(t^*)_a = t_a t^* + o(t)(t^*)_a = t_a t^*$$

Here $o(t) = 0$ due to the definition of t^*. This means we have an equation for t^* based only on the derivatives of t for which we already have equations from the original set Σ. This facilitates our development of the GNF algorithm as it gives us a polynomial system which we have seen corresponds to a system in GNF.

6.1 Proposition 3.17 (Winter)

In this section we will use the well known result that all proper systems of equations have a unique strong solution. We will give this result here using the definitions and notation of the rest of this text. This result is based on the proof from [PS09, Section 3, Theorem 1].

Theorem 6.1.1. *Every proper system of equations has a unique strong solution. It may have other solutions which are not strong.*

Proof. We create an approximating sequence using the function p from the definition of the given system of equations. We define s_0 as being the function $s_0 : X \to S\langle\langle A \rangle\rangle$ given by $s_0(x) = 0$ for all $x \in X$. This is thus the "solution" made up of only zeroes. We then define s_{i+1} as being the function given by $s_{i+1}(x) = [s_i, j]^{\#} \circ p(x)$ for $i \geq 0$ where $[s_i, j]^{\#}$ is as defined in definition 6.0.4.

We now define a trunctation operator $T_i : S\langle\langle A \rangle\rangle \to S\langle\langle A \rangle\rangle$ as

$$T_i(\sigma) = \sum_{|w| \leq i} \sigma(w)w$$

We can think of this operator as "forgetting" all terms of the power series beyond a certain length. We apply this operator to intermediate solutions s_i by applying it to the images of all variables to give $T_i(s_i(x))$ for all $x \in X$. We can prove by induction that $T_i(s_i(x)) = T_i(s_{i+t}(x))$ for all $i, t \in \mathbb{N}$ and $x \in X$. This shows that the sequence converges as the coefficients of all monomials of length at most i in the solution remain unchanged after the i-th iteration. Define s to take the values to which this process converges for each $x \in X$. We see also that $s_0(x)(1) = 0$ for all $x \in X$ and so all s_i must map all variables to power series in $S\langle\langle A \rangle\rangle_0$. This means that if s is a solution, this solution must be strong.

Again using induction we see that $T_i(s(x)) = T_i([s, j]^{\#} \circ p(x))$ for all $i \geq 0$ and $x \in X$, so the function s indeed defines a solution. This shows that a proper system has at least one strong solution. It remains to show that this solution is unique.

Let s' be another strong solution for the system. Then we must have $T_0(s'(x)) = T_0(s(x))$ for all $x \in X$ as $T_0(s'(x)) = s'(1) \cdot 1 = 0$ and $T_0(s(x)) = s(1) \cdot 1 = 0$. We now assume that $T_n(s'(x)) = T_n(s(x))$ for some $n \geq 0$. Then for $x \in X$ we have

$$\begin{aligned} T_{n+1}(s'(x)) &= T_{n+1}(s'_{n+1}(x)) \\ &= T_{n+1}([s'_n, j]^{\#} \circ p(x)) \\ &= T_{n+1}([s_n, j]^{\#} \circ p(x)) \\ &= T_{n+1}(s_{n+1}(x)) \\ &= T_{n+1}(s(x)) \end{aligned}$$

so by induction the solutions are equal.

Therefore, s is the unique strong solution to the system of equations. □

The following result can be found in [Win14]. It is, however, presented here in a somewhat extended form. This is seen mainly in the description of the steps taken to transform a system of equations into the first alternative form. We also refer more explicitly to the definitions and results used and try to distinguish clearly between systems of equations and power series, for example when referring to these as proper or rational.

This explanation should hopefully become clearer still with the example given in the next section.

Theorem 6.1.2. *Given a proper system of equations over a set of variables $X = \{x_0, \ldots, x_n\}$ we have a system in Greibach normal form over the set of variables $X' = \{x_0, \ldots, x_k\}$, with $k \geq n$, canonically obtainable from the original system and whose image under the final coalgebra mapping is the unique strong solution of the original system.*

Proof. We can transform the original system $p : X \to S\langle X + A\rangle$ into the following form

$$x_i = \sum_{j=0}^{k} x_j q_{ij} + \sum_{a \in A} a r_{ia}$$

We do this as follows

- Move all coefficients to the second position in each term
- Group terms beginning with a variable and those beginning with an alphabet symbol
- Replace any alphabet symbol $a \in A$ which does not occur at the beginning of a production with a variable $\bar{a}$ and add the equation $\bar{a} = a$ to the system

Now the q_{ij} are rational over X and proper (as power series) and the r_{ia} are also rational over X (again as power series). This is because the original system of equations is polynomial and proper i.e. each term with one variable must also have alphabet symbols or more variables.

We now take derivatives (using the sum and product rules) to obtain the equations

$$(x_i)_a = r_{ia} + \sum_{j=0}^{k} (x_j)_a q_{ij}$$

Here the terms with $o(x_j)$ disappear as we have strong solutions. It now follows from lemma 5.2.2 that the $(x_i)_a$ are rational in X (more precisely, the image of X under the unique strong solution) and so by proposition 6.0.8 we have a polynomial system with the same solution as the original system of equations. This corresponds with a system in Greibach normal form.

□

6.2 Example of finding GNF

Example 6.2.1. We now examine the example [EL02, Example 6.5] in the notation of the rest of this text as

$$x = yz + a$$
$$y = xb + zx$$
$$z = xy + zz$$

where the alphabet is $A = \{a, b\}$ and $X = \{x, y, z\}$.

Applying the first three steps of the algorithm from theorem 6.1.2 we must add a variable $\bar{b}$ to the system which is given by

$$\bar{b} = b$$

Then the second equation changes to

$$y = x\bar{b} + zx$$

We now perform operations on the set of equations which maintain the same solution. The first of these is to add variables and equations which represent the derivatives of the original variables.

$$\begin{aligned}
x_a &= y_a z + 1 \\
x_b &= y_b z \\
y_a &= x_a \bar{b} + z_a x \\
y_b &= x_b \bar{b} + z_b x \\
z_a &= x_a y + z_a z \\
z_b &= x_b y + z_b z \\
\bar{b}_a &= 0 \\
\bar{b}_b &= 1
\end{aligned}$$

We also know that for a power series σ we have

$$\sigma = O(\sigma) + \sum_{a \in A} a\sigma_a$$

so we use this to rewrite the equations for the variables x, y, z and $\bar{b}$ so that the solutions are maintained. This means we also have the equations

$$\begin{aligned}
x &= ax_a + bx_b \\
y &= ay_a + by_b \\
z &= az_a + bz_b \\
\bar{b} &= a\bar{b}_a + b\bar{b}_b
\end{aligned}$$

The $O(x)$ are all zero as we have the guarantee that the solutions are strong. We now use [Win14, Lemma 2.22] to find 'solutions' for this system. This mainly involves substitution and eliminating equations one-by-one with Arden's lemma, which will again leave the solutions unchanged. In this case we can do this in pairs as the two derivatives of the same equation do not include any reference to each other. We start at the bottom of the list and see that the solutions for $\bar{b}_a$ and $\bar{b}_b$ are simply

$$\bar{b}_a = 0 \quad \text{and} \quad \bar{b}_b = 1$$

We then move on and find

$$z_a = x_a y z^* \quad \text{and} \quad z_b = x_b y z^*$$

We now replace instances of z_a and z_b in the other equations. We, in fact, should do this for every step. However, for the previous solutions, these do not occur in the other equations. We see that y_a and y_b become

$$y_a = x_a \bar{b} + x_a y z^* x \quad \text{and} \quad y_b = x_b \bar{b} + x_b y z^* x$$

We now see that the next step is already done, as these equations are exactly what will come from applying Arden's lemma. We therefore move on to the equations for x. We first substitute in y_a and y_b to get

$$x_a = x_a \bar{b} z + x_a y z^* x z + 1 \quad \text{and} \quad x_b = x_b \bar{b} z + x_b y z^* x z$$

Applying Arden's lemma to these gives us

$$x_a = (\bar{b}z + yz^*xz)^* \quad \text{and} \quad x_b = 0$$

We can now resubstitute these equations into the others to get the system

$$\begin{aligned}
x_a &= (\bar{b}z + yz^*xz)^* \\
x_b &= 0 \\
y_a &= (\bar{b}z + yz^*xz)^*\bar{b} + (\bar{b}z + yz^*xz)^*yz^*x \\
y_b &= 0 \\
z_a &= (\bar{b}z + yz^*xz)^*yz^* \\
z_b &= 0 \\
\bar{b}_a &= 0 \\
\bar{b}_b &= 1
\end{aligned}$$

We are now close to a GNF for this system, however we still have some starred subexpressions. The next step is to replace these subexpressions with new variables t^i_*. We replace z^* with t^1_* and replace $(\bar{b}z + yz^*xz)^*$ with t^2_*. We now need equations for these variables which maintain the correct solutions to the system of equations. The obvious candidates come from the definition of the star operator:

$$\begin{aligned}
t^1_* &= zt^1_* + 1 \\
t^2_* &= (\bar{b}z + yz^*xz)t^2_* + 1
\end{aligned}$$

The problem with these equations is they are not in GNF as they begin with variables, however the variables they begin with do have equations in GNF. Substitution then gives us

$$\begin{aligned}
t^1_* &= a(\bar{b}z + yz^*xz)^*yz^*t^1_* + 1 \\
t^2_* &= (bz + a((\bar{b}z + yz^*xz)^*\bar{b} + (\bar{b}z + yz^*xz)^*yz^*x)z^*xz)t^2_* + 1
\end{aligned}$$

These equations now again contain starred subexpressions, but these can be replaced by the new variables we are defining to obtain

$$\begin{aligned}
t^1_* &= at^2_*yt^1_*t^1_* + 1 \\
t^2_* &= bzt^2_*t^2_* + at^2_*\bar{b}t^2_* + at^2_*yt^1_*xt^1_*xzt^2_* + 1
\end{aligned}$$

Now the equations are in GNF for these new variables. Looking back to the other equations, these can be brought into GNF by substituting the equations representing the derivatives back into the equations for the original variables x, y, z and $\bar{b}$ to provide the GNF. This gives

$$\begin{aligned}
x &= at^2_* \\
y &= at^2_*\bar{b} + at^2_*yt^1_*x \\
z &= at^2_*yt^1_* \\
\bar{b} &= b
\end{aligned}$$

Now we can remove the equations representing the derivatives from the system as this does not change the solution set for the variables x, y, z and $\bar{b}$. Together, the equations for $x, y, z, \bar{b}, t^1_*$ and t^2_* give a complete system in GNF equivalent to the original system.

6.3 Remarks on Complexity

The equations we find in the previous example are fairly complex, at least for the starred subexpressions. This arises mainly from the repeated substitutions we perform on the equations.

On the other hand, we do not gain too many new variables with this procedure. In contrast to the standard methods as referenced in [EL02] which gives 119 rules and the new method given in [EL02] which gives 28 rules, our method gives just 11 rules with 6 variables.

It should be noted that the definition of Greibach normal form differs somewhat in different parts of the literature. In our case it seems that we need to allow the empty word to occur in the GNF to be able to obtain equations for the starred subexpressions, as these allow for "empty word productions" (in the language of context-free grammars).

Bibliography

[EL02] Zoltán Esik and Hans Leiß. Greibach normal form in algebraically complete semirings. In *International Workshop on Computer Science Logic*, pages 135–150. Springer, 2002.

[Gre65] Sheila A Greibach. A new normal-form theorem for context-free phrase structure grammars. *Journal of the ACM (JACM)*, 12(1):42–52, 1965.

[HU69] John E Hopcroft and Jeffrey D Ullman. *Formal languages and their relation to automata*. Addison-Wesley Longman Publishing Co., Inc., 1969.

[Mar11] John C Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill NY, 4th edition, 2011.

[PS09] Ion Petre and Arto Salomaa. Algebraic systems and pushdown automata. In *Handbook of Weighted Automata*, pages 257–289. Springer, 2009.

[Rut00] Jan JMM Rutten. Universal coalgebra: a theory of systems. *Theoretical computer science*, 249(1):3–80, 2000.

[Rut03] Jan JMM Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theoretical Computer Science*, 308(1-3):1–53, 2003.

[Win14] Joost Winter. *Coalgebraic characterizations of automata-theoretic classes*. PhD thesis, CWI, 2014.