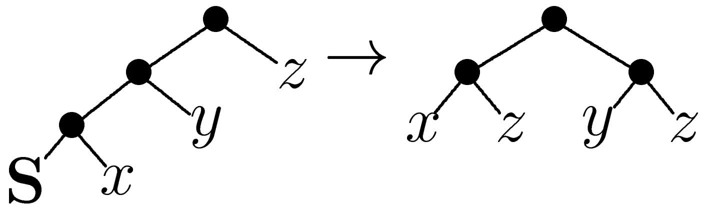
The Combinator S



vom Rat der Fakultät für Mathematik und Informatik

der Friedrich-Schiller-Universität Jena genehmigte

Dissertation

zur Erlangung des akademischen Grades

doctor rerum naturalium (Dr. rer. nat.)

von Diplom-Mathematiker Johannes Waldmann

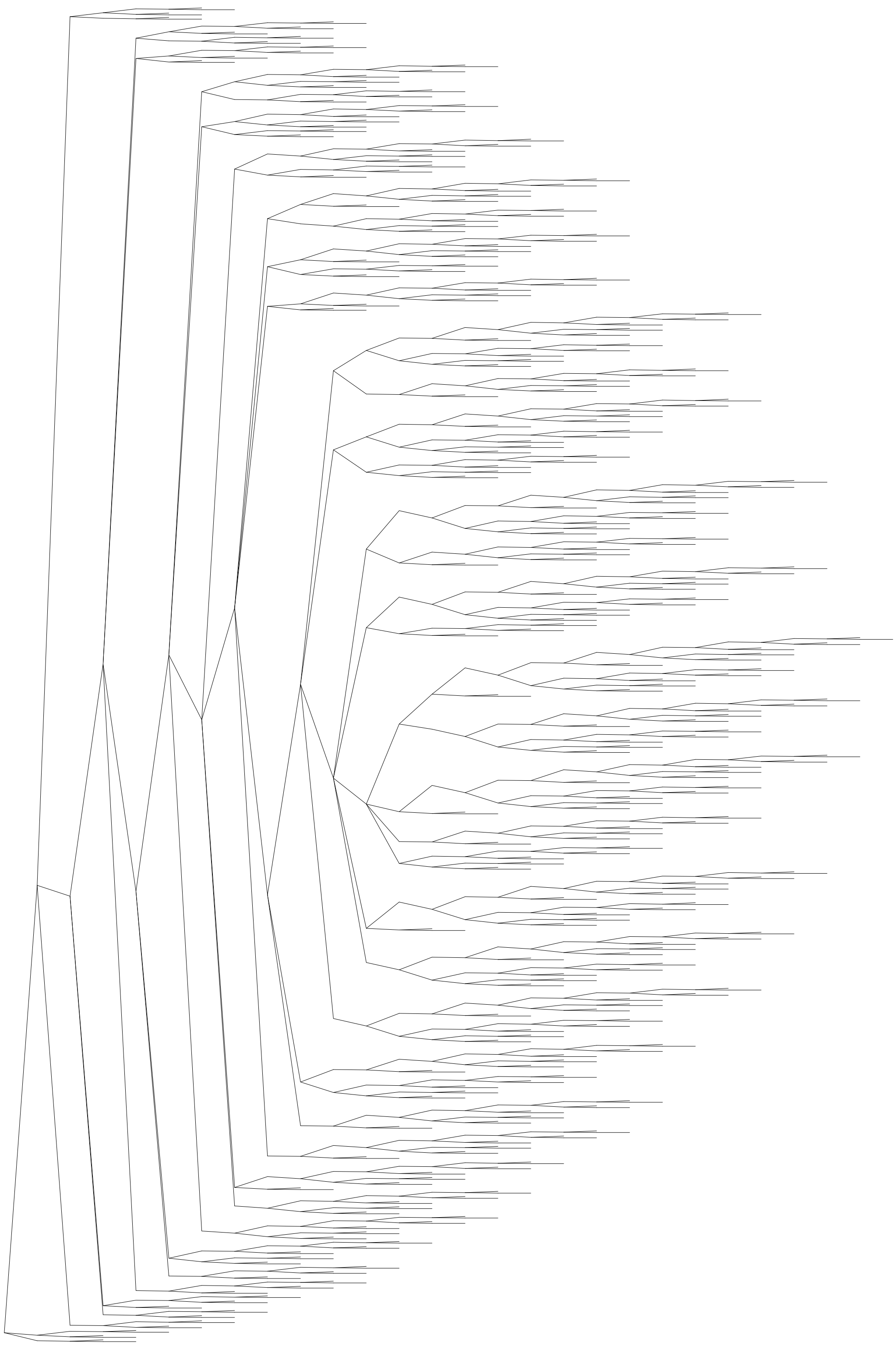
geboren am 3. Oktober 1967 in Gera

Gutachter:

1. Prof. Dr. Hans-Dietrich Hecker, Jena
2. Prof. Dr. Jan Willem Klop, Amsterdam
3. Prof. Dr. Horst Reichel, Dresden

Tag des Rigorosums: 5. Februar 1998

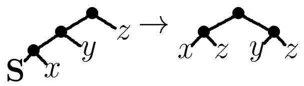
Tag der öffentlichen Verteidigung: 20. Februar 1998



S (S S S) S (S (S S) S)
after 50 leftmost innermost reductions

Introduction

This thesis analyzes the rewriting system $\mathbf{CL}(\mathbf{S})$: combinatory logic with the single combinator



We show that rational tree languages can be used in order to describe interesting properties of $\mathbf{S}$-terms.

Term rewrite systems are *complicated* in the following sense: they constitute a model of computation that is Turing-complete. Therefore, a lot of questions are undecidable in general.

On the other hand, rational languages are *handy*: they can be described by finite machines (tree automata), and here a lot of questions are decidable and a lot of operations on rational languages are effectively computable.

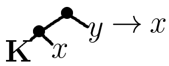
It is desirable to explore the borderline between these two realms: How far can we go in using rational languages when describing properties of term rewriting systems?

This thesis is intended to be a very detailed case study, yielding data that invites finding a more abstract description of the phenomena that are presented here.

We regard combinatory logic as a special term rewriting system. It acts on term algebras with some nullary function symbols (namely, the *combinators*) and exactly one binary function symbol (usually called *application*). Further, each combinator has an associated reduction rule whose left hand side must be of a simple shape.

Because of these restrictions, combinatory logic has the nice property that it is a *confluent* term rewrite system. Moreover, $\mathbf{CL}(\mathbf{S})$ is *non-erasing.*

Still, combinatory logic remains complicated in general because there are *complete sets* (called *bases*) of combinators that can simulate arbitrary rewrites, and therefore are computationally equivalent to Turing machines. A well known example is the two-element base $\{\mathbf{S}, \mathbf{K}\}$ with



If we now drop $\mathbf{K}$ from $\mathbf{CL}(\mathbf{S},\mathbf{K})$, we arrive at the system $\mathbf{CL}(\mathbf{S})$, which is no longer complete. Since $\mathbf{CL}(\mathbf{K})$ looks easy, we feel that $\mathbf{S}$ is responsible for the difficulty of $\mathbf{CL}(\mathbf{S},\mathbf{K})$. Therefore, $\mathbf{CL}(\mathbf{S})$ should be nearly as hard as $\mathbf{CL}(\mathbf{S},\mathbf{K})$. However some questions for $\mathbf{CL}(\mathbf{S})$ really *are* solvable, and we present these new results:

- $\mathbf{CL}(\mathbf{S})$ is top-terminating.
- $\mathbf{CL}(\mathbf{S})$ admits no ground loops.
- It is decidable whether an $\mathbf{S}$-term normalizes.
- The set of normalizing $\mathbf{S}$-terms is rational.

In this list, the last item has to be taken *cum grano salis* because it depends on a proof by computer that has only been partially completed.

This thesis also includes the description of the software system `RX`. It handles rational tree languages, and it has been used in finding and verifying results presented in this thesis.

Basically, `RX` acts as a desk top calculator that operates on rational languages. It is not tied to $\mathbf{CL}(\mathbf{S})$, but can do computations in arbitrary term algebras (with fixed arities).

On top of this, `RX` processes its input and output so that it co-operates nicely with text formatters as LaTeX[Lam94], in the spirit of a *literate programming* [Knu92] tool. For instance, `RX` had been used to format [Wal97].

Acknowledgments

The research presented in this thesis has been carried out at the Institut für Informatik at the Fakultät für Mathematik und Informatik at the Friedrich-Schiller-Universität Jena.

I am indebted to a lot of people for their assistance, encouragement and understanding during the time of my studies and PhD research.

Klaus Haberland's lectures greatly influenced my view of mathematics. I've always enjoyed discussing various topics in computer science with Traugott Schulmeiß.

Jan Willem Klop's article [Klo80] sparked my interest in combinators in general, and **S** in particular.

My supervisor, Hans-Dietrich Hecker, gave me the opportunity to follow my research interest, and the researchers from the Theoretical Computer Science group at Jena provided general support and encouragement.

Several term rewriters patiently answered and discussed questions on **CL**(**S**) with me. Especially I'd like to thank Henk Barendregt (Nijmwegen), Jan Willem Klop (CWI Amsterdam), Richard Statman (CMU Pittsburgh), for remarks on combinators; Dieter Hofbauer (TU Berlin), Alfons Geser (Tübingen), Vincent van Oostrom (VU Amsterdam), for general advice on term rewriting; Sophie Tison (LIFL, Lille), Thomas Genet (LORIA, Nancy), Florent Jacquemard (MPI Saarbrücken), Gregory Kucherov (LORIA, Nancy), for information on rational languages and rewriting.

Thomas Fröhlich (TU Ilmenau) provided me with some literature that wasn't available locally.

Alfons Geser spotted the reference to Zachos' paper [Zac78] "buried" in the bibliography of Wilfried Kurth's thesis [Kur90].

In September 1997, just before finishing writing this thesis, I visited the Research Group Theoretical Computer Science at Amsterdam University. I thank Jan Willem Klop and Vincent van Oostrom for hosting me, and providing a pleasant working environment.

I am very much indebted to Vincent van Oostrom for carefully reading the first version of this thesis.

Wolfgang Schwarz (Dresden) gave detailed comments on style and presentation.

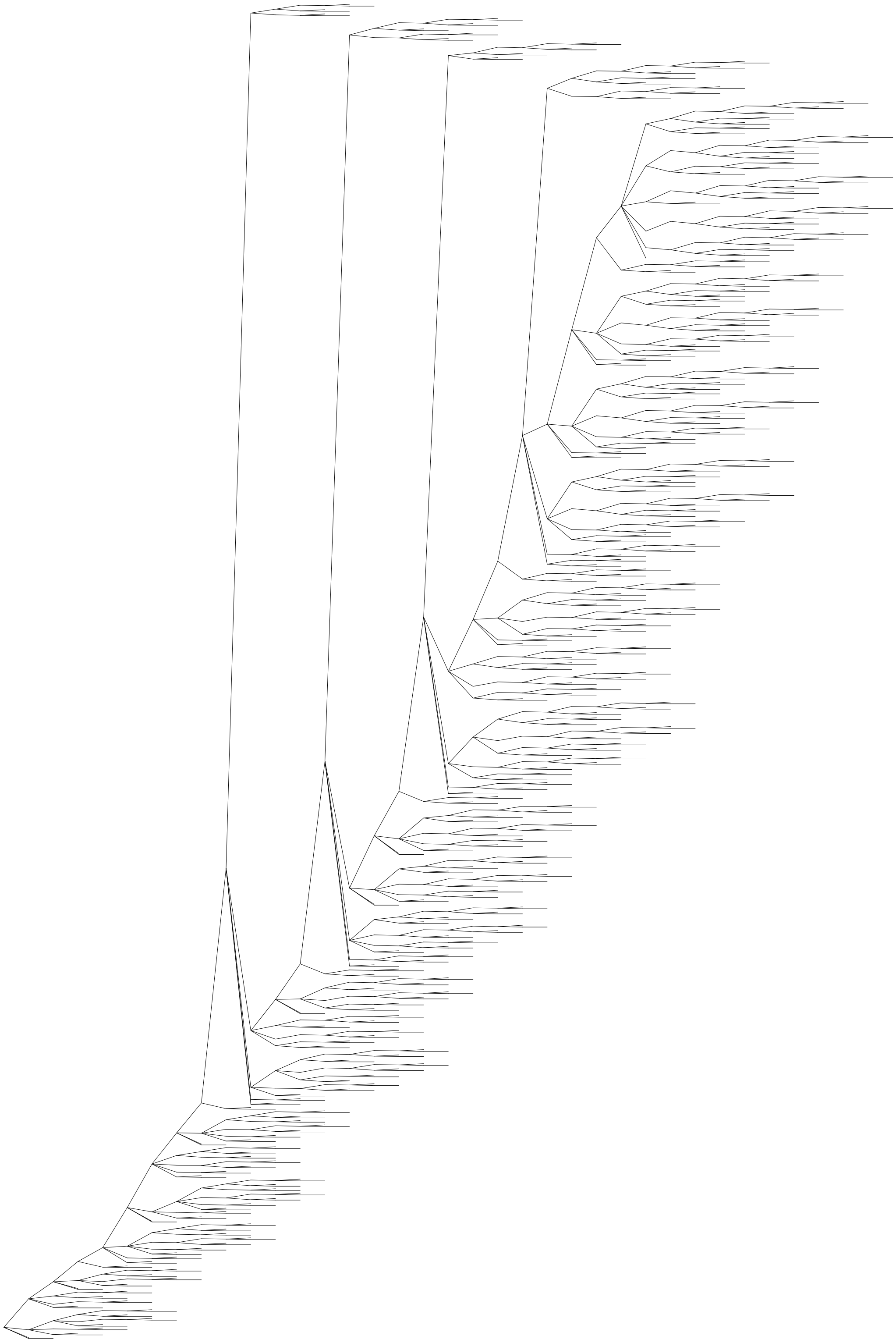
This thesis, and some of the research described therein, was prepared by using free software: the Glasgow Haskell Compiler [Tea96], the Haskell B Compiler [Aug96], the nhc Haskell compiler [Röj96] the Hugs Haskell interpreter [Jon96], the Emacs text editor, the TeX and LaTeX text formatter, tools like BibTeX and `makeindex`, the GNU/Linux operating system.

I am indebted to Frank Thierbach's pioneering LaTeX enthusiasm and wisdom. The people at our faculty's Computing Services Center, notably Erik Braun and Jutta Sieron, proved to be experts at installing and maintaining software.

Contents

Part I

Fundamentals



S (**S S S**) **S** (**S** (**S S**) **S**)
after 50 rightmost innermost reductions

Chapter 1

Abstract Reduction Systems

1.1 Introduction

An abstract reduction system (ARS) is just a set M, equipped with a relation R.

Abstract reduction does completely disregard the *internal* structure of the elements of M. Instead it focuses on the *external* structure that R introduces on M as a whole.
We imagine reductions as computations, so in particular we are interested in *well-founded* relations R because these correspond to *finite* computations that eventually halt, yielding some result.
Of course we would like those results to be *unique*. In general this is not the case, as R is only a relation, not a function. So we additionally require *confluence* of R.

Later on, we will apply the abstract theory to two concrete reduction systems: term rewriting systems and lambda calculus.

The presentation of ARS closely follows Nipkow's course notes [Nip95].

1.2 Relations

1.2.1. Definition. A *relation* R on a set M is a subset of $M \times M$.

Often the intended meaning of $(x, y) \in R$ is "x rewrites to y in one step", so we conveniently write this as $x \to_R y$, or even $x \to y$ when R is clear from the context.

Constructions

Relations can be composed.

1.2.2. Definition. $R_1 \circ R_2 \;=\; \{(x, z) \mid \exists y : (x, y) \in R_1, (y, z) \in R_2\}$

The following denotes the identity for the $\circ$ operator.

1.2.3. Definition. $\to^0 \;=\; 1_M \;=\; \{(x, x) \mid x \in M\}$

1.2.4. Definition. $\stackrel{=}{\to} \;=\; \to \cup \to^0$

Relations can be inverted.

1.2.5. Definition. $R^{-1} \;=\; \{(y, x) \mid (x, y) \in R\}$

We write $\leftarrow$ for $\to^{-1}$.

1.2.6. Definition. $\leftrightarrow \;=\; \leftarrow \cup \to$

Composition can be repeated for a fixed number of times.

1.2.7. Definition. $\to^{n+1} \;=\; \to \circ \to^n$

where $\to^0$ is defined above.
Composition can be repeated finitely often.

1.2.8. Definition. $\to^+ \;=\; \bigcup_{k>0} \to^k; \quad \to^* \;=\; \bigcup_{k\geq 0} \to^k$

We often write $\twoheadrightarrow$ instead of $\to^*$.

Properties

1.2.9. Definition. $\to$ is *transitive* iff $\to \circ \to \;\subseteq\; \to$.

1.2.10. Proposition. $\to^+$ is the transitive closure of $\to$.

This means: $\to^+$ is the smallest transitive relation that contains $\to$. (The following propositions about closures are to be understood similarly.)

1.2.11. Definition. $\to$ is *symmetric* iff $\to \;=\; \leftarrow$.

1.2.12. Proposition. $\leftrightarrow$ is the symmetric closure of $\to$.

1.2.13. Definition. $\to$ is *antisymmetric* iff $\to \cap \leftarrow \;\subseteq\; 1_M$.

1.2.14. Definition. $\to$ is *reflexive* iff $1_M \;\subseteq\; \to$.

1.2.15. Proposition. $\stackrel{=}{\to}$ is the reflexive closure of $\to$.

1.2.16. Proposition. $\to^*$ is the transitive reflexive closure of $\to$.

1.2.17. Definition. $\to$ is *irreflexive* iff $1_M \cap \to \;=\; \emptyset$.

Orders

A relation on a set creates order.

1.2.18. Definition. A relation is a *quasi order* iff it is reflexive and transitive.

Quasi orders are denoted by $\gtrsim$, and $\lesssim$ stands for $\gtrsim^{-1}$.

1.2.19. Definition. A relation is an *equivalence* iff it is a quasi order and symmetric.

Equivalences are usually denoted by $\approx$.

1.2.20. Proposition. A quasi order $\gtrsim$ induces an equivalence by $x \approx y$ iff $x \gtrsim y$ and $x \lesssim y$.

1.2.21. Definition. A relation is a *partial order* iff it is a quasi order and antisymmetric.

Partial orders are usually denoted by $\leq$. Then $\geq$ stands for $\leq^{-1}$.

1.2.22. Proposition. The quasi order $\gtrsim$ induces a partial order on the equivalence classes of $\approx$.

1.2.23. Definition. $\gtrsim$ is *total* (or: *linear*) iff $\lesssim \cup \gtrsim \; = \; M \times M$.

1.2.24. Definition. A relation is an *order* iff it is a partial order and total.

1.2.25. Definition. The *strict part* $>$ of a quasi order $\gtrsim$ is defined by $> \, = \; \gtrsim \setminus \lesssim$.

1.3 Collections

Sequences

1.3.1. Definition. M^n is the set of n-tuples with elements from M, and $M^* = \cup_{k \geq 0} M^k$.

1.3.2. Definition. Sequences are written down by enclosing their elements in parentheses. The empty sequence is denoted by $()$, and $x : (y_1, \ldots, y_n) = (x, y_1, \ldots, y_n)$. Sequences can be *concatenated*: $(x_1, \ldots, x_n) +\!\!+ (y_1, \ldots, y_m) = (x_1, \ldots, x_n, y_1, \ldots, y_m)$

Sequences are partially ordered by the prefix order.

1.3.3. Definition (prefix order). $(p_1, \ldots, p_m) \sqsubseteq (q_1, \ldots, q_n)$ iff $m \leq n$ and for $i \in \{1, \ldots, m\}$: $p_i = q_i$.

1.3.4. Definition. Sequences p and q are *independent*, written $p \parallel q$, iff neither $p \sqsubseteq q$ nor $p \sqsupseteq q$.

Multisets

Multisets can be thought of as sets with repeated elements.

1.3.5. Definition. A *finite multiset* on M is a function f from M into the naturals, with only finitely many $x \in M$ that have $f(x) > 0$.

As we only consider finite multisets throughout this thesis, we omit the attribute "finite" from now on.

For multiset operations, the usual notation for sets is employed. In the following, $A, B \dots$ denote multisets.

1.3.6. Definition. $x \in A$ iff $A(x) > 0$.

1.3.7. Definition. $A \cup B = f$ where $f(x) = A(x) + B(x)$.

1.3.8. Definition. $A \cap B = f$ where $f(x) = \max\{A(x), B(x)\}$.

1.3.9. Definition. $A \setminus B = f$ where $f(x) = \max\{0, A(x) - B(x)\}$.

1.4 Chains

Given a relation, we will be concerned with ordered chains of elements. As we have "reduction" in mind, we imagine that transition from one element of a chain to another describes a stepwise computation. Since a computation should halt, we're interested in avoiding infinite chains.

Normal Forms

1.4.1. Definition. If $x \to^+ y$, then y is a *successor* of x, and x is a *predecessor* of y. If $x \to y$, then y is an *immediate successor* of x, and x is an *immediate predecessor* of y.

1.4.2. Definition. An $x \in M$ is in *normal form* iff it has no successors.

1.4.3. Definition. An $x \in M$ *has a normal form* iff $\exists y : \; x \twoheadrightarrow y$ and y is in normal form.

1.4.4. Definition. A relation $\to$ on M is *normalizing* iff each $x \in M$ has a normal form.

1.4.5. Definition. A relation $\to$ is *terminating* iff there is no infinite chain $x_1 \to x_2 \to \dots$.

1.4.6. Remark. Often, normalizing and terminating are called *weakly* and *strongly* normalizing, respectively.

1.4.7. Definition. We write $\downarrow(x)$ if x terminates, and $x \twoheadrightarrow \infty$ if it does not.

Well-foundedness

1.4.8. Definition. An element x is *minimal* for $\to$ iff it has no successors.

1.4.9. Definition. A relation $\to$ is *well-founded* iff each nonempty subset $B \subseteq M$ contains a minimal element.

1.4.10. Proposition. A relation is terminating iff it is well-founded.

Proof. If $\to$ is not terminating, then there is some infinite reduction $x_1 \to x_2 \to \dots$. The set of all these x_i does not contain a minimal element.

On the other hand, if $\to$ is not well-founded, then there is some non-empty set B without a minimal element. Take an arbitrary $x_0 \in B$. Then x_0 must have at least one immediate successor $x_1 \in B$, otherwise x_0 would be minimal. Again x_1 has an immediate successor x_2, and so on. This constructs an infinite reduction $x_1 \to x_2 \to \dots$. □

If M is equipped with a well-founded relation, we may prove properties of M by *well-founded induction.*

1.4.11. Proposition (Well-founded induction). Let P be a predicate on M.

$$\text{If} \quad \forall x \in M : \left(\Big(\forall y \in M : (x \to y) \;\Rightarrow\; P(y) \Big) \;\Rightarrow\; P(x) \right), \quad \text{then} \quad \forall x \in M : P(x).$$

Proof. Assume there is an $x_1 \in M$ for which $P(x_1)$ does not hold. Then there must be some $x_2 \in M$ with $x_1 \to x_2$ such that $P(x_2)$ is false as well. This construction can be repeated, yielding an infinite chain $x_1 \to x_2 \to \ldots$. This contradicts well-foundedness of $\to$. □

1.4.12. Example. Classical induction on the naturals can be seen as an instance of well-founded induction. Define $x \to y$ as $x = 1 + y$.

Construction of well-founded relations

Two well-founded relations on the same set can be combined.

1.4.13. Proposition. If $\gtrsim_1$ and $\gtrsim_2$ are well-founded relations on M, then

$x \gtrsim y \quad$ iff $\quad x >_1 y$ or $(x \approx_1 y$ and $x >_2 y)$

defines a well-founded relation on M.

A relation on M can be extended to a relation on sequences with elements from M.

1.4.14. Proposition (lexicographic order). If $\gtrsim$ is a quasi order on M, then

$(x_1, \ldots, x_m) \gtrsim_{\text{lex}} (y_1, \ldots, y_n)$
iff there exists $k \leq \min\{m, n\}$ such that $\forall 1 \leq i < k : x_i \approx y_i$ and $x_k \gtrsim y_k$

defines a quasi order on M^*

1.4.15. Proposition. If $\gtrsim$ is well-founded on M, then for all n, $\gtrsim_{\text{lex}}$ is well-founded on M^n.

1.4.16. Remark. It is not well-founded on M^*.

A well-founded relation on a set M can be made into a well-founded relation on multisets of M. Intuitively we call a multiset B *smaller* than a multiset A if B can be obtained from A by replacing some elements (from the multiset X) with smaller ones (from Y).

1.4.17. Proposition (multiset order). If $\gtrsim$ is a well-founded relation on M, then

$A \gg B$ iff there exist multisets $X \neq \emptyset$ and Y such that $(A \setminus X) \cup Y = B$ and for each $y \in Y$, there is an $x \in X$ with $x > y$.

defines a well-founded relation on multisets of M.

Proof. Define $\gg_1$ just like $\gg$ but with the restriction that X be a singleton multiset. Then use $\gg \;=\; \gg_1^+$. □

1.4.18. Proposition. If $\gtrsim$ is total on M, then $A \gg B$ iff $A' >_{\text{lex}} B'$ where X' is X, sorted in descending order.

1.5 Well quasi orders

1.5.1. Definition. A quasi order $\gtrsim$ is a *well quasi order* (wqo) iff in each infinite sequence $x_1, x_2, \ldots$ there are $i < j$ with $x_i \lesssim x_j$.

One can use a wqo to show that a relation is terminating.

1.5.2. Proposition. Let $\gtrsim$ be a wqo on M, and $\to$ any relation on M. If $X \to^+ Y$ implies $X \not\gtrsim Y$, then $\to$ is terminating.

Usually, $\gtrsim$ encodes knowledge on the internal structure of the elements of M, while $\to$ describes external structure generated by some rewrite system.
This is the idea underlying reduction orders on terms, presented in section 2.8.

1.6 Confluence and Termination

1.6.1. Definition. A relation $\to$ has the *diamond property* iff $\leftarrow \circ \to \ \subseteq \ \to \circ \leftarrow$.

In other words, for all x, y, z with $x \to y$ and $x \to z$, there must exist a w such that $y \to w$ and $z \to w$.

1.6.2. Definition. A relation $\to$ is *locally confluent* iff $\leftarrow \circ \to \ \subseteq \ \to^* \circ \leftarrow^*$.

In other words, for all x, y, z with $x \to y$ and $x \to z$, there must exist a w such that $y \twoheadrightarrow w$ and $z \twoheadrightarrow w$.

1.6.3. Definition. A relation $\to$ is *confluent* iff $\leftarrow^* \circ \to^* \ \subseteq \ \to^* \circ \leftarrow^*$.

Equivalently, $\to^*$ has the diamond property.
Confluence does not follow from local confluence. See proposition 1.6.8.
However simple induction proves

1.6.4. Proposition. If $\to$ has the diamond property, then $\to^*$ also has.

1.6.5. Definition. A relation is *Church-Rosser* iff $\leftrightarrow^* \ \subseteq \ \to^* \circ \leftarrow^*$.

1.6.6. Proposition. A relation is Church-Rosser iff it is confluent.

Proof. By induction on the number of changes of direction in $\leftrightarrow^*$. □

1.6.7. Definition. A relation is *convergent* iff it is terminating and confluent.

1.6.8. Proposition (Newman-Lemma). If a relation $\to$ is terminating, then confluence and local confluence coincide.

1.6.9. Proposition. If $\to$ is confluent, then each x has at most one normal form.

So confluent relations describe computations that are well-behaved: if they halt with a result, then it might have been obtained through different paths, but it is unique.

Chapter 2

Term Rewriting Systems

2.1 Introduction

This chapter presents term rewriting systems (TRS) as a concrete examples of an abstract reduction systems, by specifying the set M of elements that are being rewritten.

Here M is the set of *terms*. These are elements of some free algebra. They have *internal* structure (via the subterm relation), and rewriting rules impose the *external* structure.

Historically, *string* rewriting had been considered first. We may regard strings as special terms, and therefore omit a separate treatment of string rewriting.

An extension of term rewriting is *higher order* rewriting, where terms may contain bound variables. We will not be interested in its general theory (for instance, see [KvOvR93]), but only in one particular example, lambda calculus, which we will deal with in the next chapter.

The presentation of TRS closely follows Avenhaus' textbook [Ave95]. Standard references are the surveys by Klop [Klo92] and that by Dershowitz and Jouannaud [DJ90]. We will follow their suggestions for notation [DJ91].

2.2 The Term Algebra

Terms

2.2.1. Definition. An *signature* is a pair (F, a) where F is a set and a is a mapping F to $\mathbb{N}$.
$a(f)$ is called the *arity* of $f \in F$.
Elements f with $a(f) = 0$ are called *constants.*

Usually, a is clear from the context so we just write down F.

2.2.2. Definition. A tuple (F, a, D) is an *algebra with carrier set* D if (F, a) is a signature and and for each $f \in F$, $f : D^{a(f)} \to D$.

The *term algebra* generated by F is the free algebra on some set of variables.

2.2.3. Definition. Given a signature F and a set V of variables, disjoint from F, the set of *terms* $T(F, V)$ is defined by

$$\begin{array}{rl} v \in V: & v \in T(F,V) \\ f \in F,\ n = a(f),\ t_1 \in T(F,V),\ \ldots,\ t_n \in T(F,V): & f(t_1, \ldots, t_n) \in T(F,V) \end{array}$$

2.2.4. Definition. The set of *ground* terms on F, denoted by $T(F)$, is $T(F, \emptyset)$.

2.2.5. Definition. The size $|t|$ of a term $t \in T(F, V)$ is defined by

$$\begin{array}{l} |v| = 1 \quad \text{for} \quad v \in V \\ |f(t_1, \ldots, t_n)| = 1 + |t_1| + \ldots + |t_n| \end{array}$$

2.2.6. Definition. The depth of a term $t \in T(F, V)$ is defined by

$$\begin{array}{l} \text{depth}(v) = 0 \quad \text{for} \quad v \in V \\ \text{depth}(f(t_1, \ldots, t_n)) = 1 + \max\{\text{depth}(t_1), \ldots, \text{depth}(t_n)\} \end{array}$$

Positions

A *position* is a sequence of natural numbers that indicates the location of some subterm.

2.2.7. Definition. The set of subterm positions, $\text{Pos}(t)$, is

$$\begin{array}{l} \text{Pos}(v) = \{()\} \quad \text{for} \quad v \in V \\ \text{Pos}(f(t_1, \ldots, t_n)) = \{()\} \cup \displaystyle\bigcup_{i \in \{1, \ldots, n\}, s \in \text{Pos}(t_i)} (i : s) \end{array}$$

Subterms

2.2.8. Definition. $t\,[p]$, the subterm of a term t at position p, is defined by

$$\begin{aligned} t\,[()] &= t \\ f(t_1,\ldots,t_n)\,[(p_1,p_2,\ldots)] &= t_{p_1}\,[(p_2,\ldots)] \end{aligned}$$

This is only defined in case $p \in \mathrm{Pos}(t)$.

2.2.9. Definition. $t\,[p \leftarrow t']$ is the result of replacing the subterm at position p in t with t'.

$$\begin{aligned} t\,[() \leftarrow t'] &= t' \\ f(t_1,\ldots,t_n)\,[(p_1,p_2,\ldots) \leftarrow t'] &= f(\ldots,\quad \underbrace{t_{p_1}\,[(p_2,\ldots) \leftarrow t']}_{\text{the } p_1\text{-th argument of } f} \quad,\ldots) \end{aligned}$$

This is only defined in case $p \in \mathrm{Pos}(t)$.

2.2.10. Definition. s *is a subterm of* t, written $s \trianglelefteq t$ iff there is $p \in \mathrm{Pos}(t)$ such that $t\,[p] = s$. s is a *strict* subterm, written $s \triangleleft t$, iff $p \neq ()$.

2.2.11. Proposition. $\triangleright$ is a well-founded partial order.

2.2.12. Proposition. $\triangleright$ is not a wqo.

2.2.13. Definition. The set of variables of a term $t \in T(F,V)$, denoted by $\mathrm{Var}(t)$, is $\{v \mid v \in V, v \trianglelefteq t\}$.

We define a notation for the set of all positions in t where a certain subterm x occurs:

2.2.14. Definition. $\mathrm{Pos}(x,t) = \{p \in \mathrm{Pos}(t) \mid t\,[p] = x\}$

Contexts

A context is a term with a hole.

2.2.15. Definition. A term $C \in T(F \cup [])$, where $[]$ is a new nullary constructor, with $\mathrm{Pos}([],t) = \{p\}$, is called *context*. It is written as $C[]$. The expression $C[X]$ stands for $C\,[p \leftarrow X]$.

Words as Terms

2.2.16. Definition. A term algebra where there is exactly one $z \in F$ with $a(z) = 0$, and all other $f \in F$ have $a(f) = 1$, is called *word algebra*.

In such cases, we write $fgffg$ instead of $f(g(f(f(g(z)))))$.
This gives a natural mapping between $T(F)$ and the sequences (words) $(F \setminus \{z\})^*$.
Words have additional structure because they can be concatenated, and with this operation they form a monoid. It is not directly possible to do this with terms. One may consider terms with designated "concatenation leaves". We will not use this in the sequel.

2.3 Substitutions

2.3.1. Definition. A *substitution* is a mapping from variables to terms $\sigma : V \to T(F,V)$, with only finitely many $v \in V$ for which $v\sigma \neq v$. Those v are called the *domain* of σ.

Usually $\sigma(X)$, the application of the substitution σ to a term X is written as $X\sigma$.

σ is extended to a mapping $\sigma' : T(F,V) \to T(F,V)$ by

$$\begin{aligned} v\sigma' &= v\sigma \quad \text{for all} \quad v \in V \\ f(t_1,\ldots,t_n)\sigma' &= f(t_1\sigma',\ldots,t_n\sigma') \end{aligned}$$

Usually, σ' is denoted by σ as well.

If σ is the substitution with $x\sigma = X, y\sigma = Y$ and $v\sigma = v$ for all other $v \in V$, then we write σ as $[x \leftarrow X, y \leftarrow Y]$.

Subsumption

2.3.2. Definition. A term t_1 is an *instance* of a term t_2 iff there is a substitution σ such that $t_1 = t_2\sigma$. This is also called t_2 *subsumes* t_1 and denoted by $t_1 \succeq t_2$. Equivalently, we say t_1 *matches the pattern* t_2, and the substitution σ is called a *match.*

2.3.3. Proposition. $t_1 \succeq t_2$ iff $\mathrm{Inst}(t_1) \subseteq \mathrm{Inst}(t_2)$,

where $\mathrm{Inst}(t)$ denotes the set of all instances of t.

2.3.4. Proposition. $\succeq$ is a well-founded quasi order on $T(F,V)$.

Subsumption, which was defined on terms, can also be extended to substitutions.

2.3.5. Definition. Let σ_1 and σ_2 be substitutions. We say that σ_1 *subsumes* σ_2, written $\sigma_1 \succeq \sigma_2$, iff for all v from their common domain we have $\sigma_1(t) \succeq \sigma_2(t)$

2.3.6. Proposition. $\succeq$ is a quasi order on substitutions.

Unification

2.3.7. Definition. A *unifier* of terms t_1 and t_2 is a substitution σ such that $t_1\sigma = t_2\sigma$. The terms t_1 and t_2 are called *unifyable.*

2.3.8. Definition. A *most general unifier* (mgu) of terms t_1 and t_2 is a unifier σ of t_1 and t_2 that subsumes all unifiers of t_1 and t_2.

mgus are computable, if the terms are unifyable. mgus are essentially unique.

2.4 Rewrites

Term Rewriting Systems

2.4.1. Definition. A *term rewriting system* (TRS for short) on a signature (F,a) is a set of pairs $(l,r) \in T(F,V) \times T(F,V)$ with $\mathrm{Var}(l) \supseteq \mathrm{Var}(r)$. These pairs are called *rules*, and a rule (l,r) is written as $l \to r$. Usually, we will consider TRS with finitely many rules.

2.4.2. Definition. A TRS R defines a relation $\to_R$ on terms $T(F, V)$ by $t_1 \to_R t_2$ iff

- there is $(l, r) \in R$ such that
- there is a position $p \in \text{Pos}(t_1)$ such that
- there is a substitution σ that matches $t_1\,[p]$ to l and
- $t_2 = t_1\,[p \leftarrow r\,\sigma]$.

$t_1\,[p]$ is called *redex* (from: *red*ucible *ex*pression), p is called *redex position*, $r\,\sigma$ is called *contractum* of the redex.

We simply write $\to$ instead of $\to_R$ whenever R is clear from the context.

2.4.3. Definition. $X \hookrightarrow Y$ means when reducing X, Y occurs as a subterm: $X \to Z$ and $Z \trianglerighteq Y$. We write $\hookrightarrow\!\!\!\!\rightarrow$ instead of $\hookrightarrow^*$.

Syntactic Properties of Rewrite Rules

2.4.4. Definition. A term $t \in T(F, V)$ is called *linear* iff no variable occurs more than once: for all $v \in V$, $|\text{Pos}(v, t)| \leq 1$.

2.4.5. Definition. A rule (l, r) is called *left* (resp. *right) linear*, iff l (resp. r) is a linear term. It is called *linear* iff both l and r are linear and $\text{Var}(l) = \text{Var}(r)$.

2.4.6. Definition. A TRS R is called *(left* resp. *right) linear* iff *all* its rules are.

2.4.7. Definition. A rule (l, r) is called *(left, right) ground* iff $(l,\ r)$ both of l and r are ground terms.

2.4.8. Definition. A TRS R is called *(left resp. right) ground* iff *all* its rules are.

2.4.9. Definition. A rule (l, r) is called *duplicating* iff there is $v \in V$ with $1 \leq |\text{Pos}(v, l)| < |\text{Pos}(v, r)|$.

2.4.10. Definition. A rule (l, r) is called *erasing* iff there is $v \in V$ with $|\text{Pos}(v, l)| > |\text{Pos}(v, r)| = 0$.

2.4.11. Definition. A rule (l, r) is called *collapsing* iff $r \in V$.

2.4.12. Definition. A TRS is called *duplicating* resp. *erasing* resp. *collapsing* iff *some* of its rules are.

2.5 Confluence of TRS

For rewrite systems in general, confluence is a desirable property, because it describes well-behaved computations.

In a TRS, non-confluence might arise because in a term, redexes might overlap, and reducing one of them destroys the other. This situation is captured in the following definition.

2.5.1. Definition. In a TRS R, let

- $(l_1 \to r_1)$ and $(l_2 \to r_2)$ be two rules, (They need not be distinct. It is assumed that $\mathrm{Var}(l_1)$ is disjoint from $\mathrm{Var}(l_2)$, which can be achieved by re-naming.)
- $p \in \mathrm{Pos}(l_1)$ with $l_1\,[p] \notin V$, where $p \neq ()$ if the rules are the same,
- $l_1\,[p]$ and l_2 are unifyable, where σ is their mgu.

Then $(l_1\,[p \leftarrow r_2]\ \sigma, r_1\,\sigma)$ is called a *critical pair* of R.

2.5.2. Remark. A rule might overlap with itself, but not at root position.

Some critical pairs that are joinable: there is some z with $x \twoheadrightarrow z$ and $y \twoheadrightarrow t$). Non-joinable critical pairs (x, y) are obstacles for confluence. For a system with such critical pairs, confluence can hopefully be obtained by adding one of the rules $x \to y$ and $y \to x$ to the TRS.

This process of adding rules that keep $\leftrightarrow^*$ invariant and terminating, but make $\to$ confluent, is called *completion*. Since combinatory logic is non-overlapping, we don't go into details of completion here.

2.5.3. Definition. A TRS is called *non-overlapping* iff it has no critical pairs.

2.5.4. Definition. A TRS is called *orthogonal* iff it is non-overlapping and left linear.

2.5.5. Proposition. Orthogonal TRS are confluent.

Proof. Use the relation of *parallel reduction*, defined below. $\overset{\shortparallel}{\to}$ that has the diamond property, and $\to\ \subseteq\ \overset{\shortparallel}{\to}\ \subseteq\ \to^*$. Then $\overset{\shortparallel}{\to}^*$ has the diamond property as well, and $\to^*\ \subseteq\ \overset{\shortparallel}{\to}^*\ \subseteq (\to^*)^* = \to^*$, which means $\to^* = \overset{\shortparallel}{\to}^*$, so $\to^*$ has the diamond property. $\square$

2.5.6. Definition (parallel reduction). $s_0 \overset{\shortparallel}{\to} s_n$ iff $s_0 \to s_1 \to \ldots \to s_n$ and the reductions happen at positions $p_0, \ldots, p_{n-1}$ that are pairwise independent: $i \neq j \ \Rightarrow\ p_i \parallel p_j$.

2.6 Reduction Strategies

A *strategy* maps a term to a reduction sequence starting with that term.

2.6.1. Definition. A strategy is *normalizing* iff it leads to a normal form, provided there is one.

2.6.2. Definition. A reduction step is *outermost* iff the contracted redex is not a proper subterm of a redex.

2.6.3. Proposition. For orthogonal systems, any outermost strategy is normalizing.

2.6.4. Definition. A reduction step is *innermost* iff the contracted redex has no proper subterm that is itself a redex.

An innermost strategy need not be normalizing. It is, however, in case the TRS is non-erasing.

2.6.5. Proposition. If a TRS is both non-overlapping and non-erasing, then normalization and termination coincide.

This says that *any* reduction strategy finds the normal form, provided it exists. In particular, we can use innermost reduction.

For a detailed analysis, see Gramlich's survey paper [Gra92].

Since $\mathbf{CL}(\mathbf{S})$ is orthogonal and non-erasing, we don't have to worry much about strategies.

2.7 Ground TRS

2.7.1. Definition. A TRS is *ground* iff in all of its rules $l \to r$, both l and r are ground terms.

Ground TRS are very restricted in their rewriting power. Some of the problems that are hard for general TRS are manageable for ground TRS.

2.7.2. Proposition. It is decidable whether a ground TRS is terminating.

2.7.3. Proposition. It is decidable whether a ground TRS is confluent.

A proof is in [Oya87].

2.8 Orders on Terms

Homeomorphic embedding

2.8.1. Definition. The relation $\geq_{\mathrm{HE}}$ of *homeomorphic embedding* on $T(F)$ is given by $\to_E^+$ where E is the TRS

$$\{f(x_1,\ldots,x_{a(f)}) \to x_i \mid f \in F, i \in \{1,\ldots,a(f)\}\}$$

2.8.2. Proposition. $\leq_{\mathrm{HE}}$ is a well-founded partial order.

This relation is "better" than the subterm relation $\trianglerighteq$ because we have

2.8.3. Theorem (Kruskal's Tree Theorem). $\geq_{\mathrm{HE}}$ is a wqo on $T(F)$.

This theorem will prove very useful in the analysis of infinite sequences of terms.

Reduction orders

2.8.4. Definition. A relation $>$ on $T(F,V)$ is *closed w.r.t. substitutions* iff for all substitutions $s : V \to T(F,V)$ and all terms $t_1, t_2 \in T(F,V)$:

$$t_1 > t_2 \quad \Rightarrow \quad s(t_1) > s(t_2)$$

2.8.5. Definition. A relation $>$ on $T(F)$ is *closed w.r.t. contexts* iff for all terms $t \in T(F,V)$, for all positions $p \in \mathrm{Pos}(t)$, for all terms $t_1, t_2 \in T(F,V)$:

$$t_1 > t_2 \quad \Rightarrow \quad t\,[p \leftarrow t_1] > t\,[p \leftarrow t_2]$$

2.8.6. Definition. A relation $>$ on $T(F,V)$ *contains the subterm relation* iff for all terms t, for all positions $p \neq ()$ in $\mathrm{Pos}(t)$:

$$t \quad > \quad t\,[p]$$

2.8.7. Definition. A *reduction order* is a quasi order $\gtrsim$ on terms $T(F,V)$ such that

- $>$ is well-founded
- $>$ is closed w.r.t. substitutions
- $>$ is closed w.r.t. contexts

2.8.8. Definition. A reduction order $\gtrsim$ is *compatible* with a TRS R iff $l > r$ for all rules $l \to r$ from R.

2.8.9. Proposition. A TRS R is terminating iff there is a reduction order compatible with R.

Simplification orders

2.8.10. Definition. A *simplification order* is a quasi order $\gtrsim$ on terms $T(F,V)$ such that

- $>$ is closed w.r.t. contexts
- $>$ contains the subterm relation

2.8.11. Proposition. Each simplification order $\gtrsim$ on $T(F,V)$ contains the relation of homeomorphic embedding: $\geq_{\mathrm{HE}} \;\subseteq\; \gtrsim$

That's good. Kruskal's theorem 2.8.3 can be applied.

2.8.12. Proposition. Each simplification order on $T(F)$ is well-founded.

2.8.13. Proposition. Each simplification order on $T(F,V)$ that is closed w.r.t. substitutions, is a reduction order.

Recursive Path Orders

2.8.14. Definition. If $\gtrsim$ is a quasi order on F, then the *recursive path order* $>_{\mathrm{rpo}}$ on $T(F,V)$ is given by $a = f(a_1,\ldots,a_n) >_{\mathrm{rpo}} g(b_1,\ldots,b_m) = b$ iff

- there exists i such that $a_i >_{\mathrm{rpo}} b$,
- or $f > g$ and for all j we have $s >_{\mathrm{rpo}} b_j$,
- or $f \approx g$ and $\{a_1,\ldots,a_n\} \gg_{\mathrm{rpo}} \{b_1,\ldots,b_m\}$

and $a = f(a_1,\ldots,a_n) >_{\mathrm{rpo}} x$ if $x \in \mathrm{Var}(a)$.

2.8.15. Proposition. $>_{\mathrm{rpo}}$ is a quasi order on $T(F,V)$.

2.8.16. Theorem. $>_{\mathrm{rpo}}$ is a reduction order.

This is again a deep theorem. It works because of Kruskal's theorem 2.8.3, and because the multiset order respects well-foundedness (proposition 1.4.17).

Chapter 3

Lambda Calculus and Combinatory Logic

3.1 Introduction

Lambda Calculus is an abstract reduction system that describes rewriting terms with *bound* variables. (So it's *not* a TRS.) It can be regarded as the syntactic theory of function abstraction and application. In this respect, it is a "quintessential" system because *all* its terms denote functions.

Instead of the extensional view as a set of (argument, value)-pairs, in lambda calculus, functions are described intensionally, as algorithms that compute values. So lambda calculus is a model of computation. Its complexity is equivalent to other models such as Turing machines or recursive functions. As is the case with these models, some properties of computations are necessarily undecidable.

Combinatory logic is a reduction system that is closely related to lambda calculus. The systems share the view that "everything is a function". However, terms in combinatory logic do *not* contain variables at all. So combinatory logic *is* a TRS. It is in fact an orthogonal TRS and therefore enjoys nice properties.

Yet there are translations between combinatory logic and lambda calculus, showing that both systems are computationally equivalent. Therefore undecidability results from lambda calculus carry over to combinatory logic (and to TRS in general).

In the main part of this thesis, we will consider the system **CL**(**S**) which is a subsystem of combinatory logic. We will show that certain undecidable properties of the full system become manageable in the subsystem.

The presentation of lambda calculus and combinatory logic follows Barendregt [Bar84], of course.

3.2 Lambda Terms

3.2.1. Definition. The set Λ of lambda terms is given by

1. (Variables) for $i > 0 : x_i \in \Lambda$
2. (Abstractions) for $i > 0, M \in \Lambda : \lambda x_i.M \in \Lambda$
3. (Applications) for $M, N \in \Lambda : (M \cdot N) \in \Lambda$.

The notions for terms introduced in section 2.2 will be carried over to Λ. There is one interesting point, however: in an abstraction $\lambda x.M$, we regard M as immediate subterm at position (1), and the variable x does not count as a subterm.
We will follow these standard conventions when writing down lambda terms:

- Application is left-associative, so $M \cdot N \cdot P$ stands for $(M \cdot N) \cdot P$.
- The application symbol is completely omitted, so (MN) stands for $(M \cdot N)$.
- Nested abstractions are combined into one λ symbol, so $\lambda xy.x$ stands for $\lambda x.(\lambda y.x)$.

3.2.2. Definition. The set $\mathrm{FVar}(t)$ of *free* variables of a lambda term t is

1. $\mathrm{FVar}(x_i) = \{x_i\}$
2. $\mathrm{FVar}(\lambda x_i.M) = \mathrm{FVar}(M) \setminus \{x_i\}$
3. $\mathrm{FVar}(MN) = \mathrm{FVar}(M) \cup \mathrm{FVar}(N)$

3.2.3. Definition. A lambda term M is *closed* iff $\mathrm{FVar}(M) = \emptyset$. The set of all closed terms is Λ^0.

3.2.4. Definition. A variable x_i is *bound* in a term t if there is some subterm $\lambda x_i.M$ of t.

3.3 Beta reduction

3.3.1. Definition. A lambda term of the form $(\lambda x.M)N$ is called *redex*. Its *contractum* is the term $M[x \leftarrow N]$. The relation β consists of all pairs (redex, contractum).

Again notations for substitutions for terms, as defined in section 2.3, are carried over to Λ.
Note however that in $M[x \leftarrow N]$ it is understood that only free occurrence of x in M are replaced, and that no free variable of N gets bound. This can be achieved by some renaming of variables. We omit details.

3.3.2. Definition. The closure of β w.r.t. contexts (see definition 2.8.5) is called $\rightarrow_\beta$.

We write $=_\beta$ for $\leftrightarrow_\beta^*$.
We often omit the subscript and just write $\rightarrow$ or $=$.

3.4 Confluence

As had been mentioned for ARS, confluence is a desirable property which ensures that computations are well-behaved.

3.4.1. Theorem (Church, Tait, Martin-Löf). $\rightarrow$ is confluent.

There are several ways to prove this theorem.

(Just proving local confluence is easy, but not enough, as the system is non-terminating.)

We mention one possibility that is similar to that of proposition 2.5.5, thereby showing that Λ in some respect behaves like an orthogonal TRS.

A *development* $X \rightarrow_{\text{dev}} Y$ of a term X containing *marked* redexes is a reduction chain that only reduces redexes that were marked in X. During such a reduction, existing markers may get copied but no new markers are introduced.

3.4.2. Proposition (Finite Developments). Developments $X \rightarrow_{\text{dev}} Y$ are always finite. The result Y of a *complete* development $X \rightarrow_{\text{cpl}} Y$ which does not contain markers any more, is unique.

Confluence then follows from

3.4.3. Proposition. $\rightarrow_{\text{cpl}}$ has the diamond property.

and the obvious

3.4.4. Proposition. $\rightarrow \subseteq \rightarrow_{\text{cpl}} \subseteq \rightarrow^*$

3.5 Non-erasing lambda terms

In analogy to non-erasing TRS, we define a subset of non-erasing lambda terms.

3.5.1. Definition. The set $\Lambda_{\mathbf{I}}$ is given by

1. (Variables) for $i > 0 : x_i \in \Lambda_{\mathbf{I}}$
2. (Abstractions) for $i > 0, M \in \Lambda_{\mathbf{I}}$: if $x_i \in \text{FVar}(M)$, then $\lambda x_i.M \in \Lambda_{\mathbf{I}}$
3. (Applications) for $M, N \in \Lambda_{\mathbf{I}} : (MN) \in \Lambda_{\mathbf{I}}$.

3.5.2. Proposition. $\Lambda_{\mathbf{I}}$ is forward closed under $\rightarrow$: If $M \in \Lambda_{\mathbf{I}}$ and $M \rightarrow N$, then $N \in \Lambda_{\mathbf{I}}$.

The subsystem $\Lambda_{\mathbf{I}}$ bears resemblance to an orthogonal and non-erasing TRS.

3.5.3. Theorem (Conservation Theorem). In $\Lambda_{\mathbf{I}}$, normalization and termination coincide.

3.6 Standardization

3.6.1. Definition. Each lambda term M is of the form $\lambda x_1.\cdots.\lambda x_n.HM_1\ldots M_m$ where either H is a variable and $n \geq 0$, $m \geq 0$, or $H = \lambda y.M_0$ and $n \geq 0$, $m \geq 1$.

In the first case, M is in *head normal form*. In the second case, HM_1 is called the *head redex* of M. Any redex in one of the M_i is called *internal*.

3.6.2. Definition. An expression is in *head normal form* iff it has no head redex.

3.6.3. Definition. A *head reduction* $\rightarrow_h$ contracts only head redexes; an *internal reduction* $\rightarrow_i$ contracts only internal redexes.

3.6.4. Definition. A *standard* reduction $M \longrightarrow\!\!\!\!\rightarrow_s P$ is a (possibly empty) sequence of head reductions $M \longrightarrow\!\!\!\!\rightarrow_h N = N_0 N_1 \ldots N_n$ followed by internal reductions that are standard w.r.t. the N_i.

3.6.5. Theorem (Curry and Feys). For all reductions $M \twoheadrightarrow N$, there exists a standard reduction $M \longrightarrow\!\!\!\!\rightarrow_s N$.

Proof. First one shows that each reduction $M \twoheadrightarrow N$ can be re-arranged in the form $M \longrightarrow\!\!\!\!\rightarrow_h P \longrightarrow\!\!\!\!\rightarrow_i N$.

Then $N = N_0 N_1 \ldots N_k$, and also $P = P_0 P_1 \ldots P_k$. Now the theorem is applied to $P_i \twoheadrightarrow N_i$ recursively.

This is a finite process by well-founded induction on the size of the N_i. $\square$

3.6.6. Corollary. M has a head normal form N iff the head reduction sequence $M \longrightarrow\!\!\!\!\rightarrow_h \ldots$ terminates.

Proof. Assume $M \twoheadrightarrow N$ where $N = N_0 N_1 \ldots N_k$ is in head normal form. We have $M \longrightarrow\!\!\!\!\rightarrow_h P \longrightarrow\!\!\!\!\rightarrow_i N$ by the standardization theorem, but then P must be in head normal for as well, and $M \longrightarrow\!\!\!\!\rightarrow_h P$ is a terminating head reduction chain. $\square$

Head reductions and the standardization theorem, albeit for combinatory logic, will be used when top-termination of $\mathbf{CL}(\mathbf{S})$ is considered in chapter 7.

3.7 Lambda-definable Functions

Lambda calculus can simulate computations on numbers. The presentation here closely follows [Bar84].

3.7.1. Definition (lambda-definable). Given a suitable encoding of numbers $\ulcorner\cdot\urcorner : \mathbb{N} \rightarrow \Lambda^0$, the function $f : N^k \rightarrow N$ is *lambda-defined* by $f_\lambda \in \Lambda^0$ if $f_\lambda \ulcorner x_1 \urcorner \ldots \ulcorner x_k \urcorner = \ulcorner f(x_1, \ldots, x_k) \urcorner$

We mention the difficulty of representing undefined values of computations.

With partial recursive functions, an undefined sub-computation implies the failure of the whole computation. This has to be simulated in lambda calculus.

One has the option of using terms from $\Lambda^0_{\mathbf{I}}$ for encoding. This had been done by Church originally. [Chu41] By the conservation theorem, each subterm of a term with normal form itself has a normal form. So we might represent "undefined" by any term without normal form. But the translation $\ulcorner 0 \urcorner$ does not seem to fit well into $\Lambda^0_{\mathbf{I}}$ and therefore creates technical problems.

The other option is to use terms from Λ^0. There, $\lceil 0 \rceil$ can easily be encoded. But in Λ^0, there are terms with normal form that have subterms without normal form. So "undefined" must be expressed by some other class of terms. Here, *unsolvable* terms (terms without head normal form) are appropriate. But then another technical problem arises. The composition of encodings of functions not necessarily corresponds to the encoding of the composition.

As we just intend to give the flavour of lambda-definability, we avoid all the technicalities by restricting ourselves to total functions from now on.

We first define a suitable encoding $\mathbb{N} \to \Lambda^0$. (There are many of them. We don't define the exact meaning of *suitable*.)

3.7.2. Definition (Truth values). $\mathbf{T} = \lambda xy.x$, $\mathbf{F} = \lambda xy.y$

3.7.3. Definition (Pairing, selectors). $\langle M, N \rangle = \lambda x.xMN$, $\langle P \rangle_1 = P\,\mathbf{T}$, $\langle P \rangle_2 = P\,\mathbf{F}$.

3.7.4. Proposition. $\langle\langle M, N \rangle\rangle_1 \twoheadrightarrow M$, $\langle\langle M, N \rangle\rangle_2 \twoheadrightarrow N$.

3.7.5. Definition (Numbers). $\lceil 0 \rceil = \lambda x.x$, $\lceil n+1 \rceil = \langle \mathbf{F}, \lceil n \rceil \rangle$

Note that numbers are distinct normal forms.

3.7.6. Proposition. $\lceil 0 \rceil\,\mathbf{T} = \mathbf{T}$, $\lceil n+1 \rceil\,\mathbf{T} = \mathbf{F}$

Now we do some basic computations with numbers.

3.7.7. Proposition. There are expressions $\mathbf{succ}, \mathbf{pred}, \mathbf{zero} \in \Lambda^0$ such that $\mathbf{succ}\,\lceil n \rceil = \lceil n+1 \rceil$, $\mathbf{pred}\,\lceil n+1 \rceil = \lceil n \rceil$, $\mathbf{zero}\,\lceil 0 \rceil = \mathbf{T}$, $\mathbf{zero}\,\lceil n+1 \rceil = \mathbf{F}$.

Proof. Define $\mathbf{succ} = \lambda x.\,\langle \mathbf{F}, x \rangle$, $\mathbf{pred} = \lambda x.x\,\mathbf{F}$, $\mathbf{zero} = \lambda x.x\,\mathbf{T}$. □

Obviously constant functions, selectors, the successor function and the substitution operator are lambda-definable.

To define iterative computations, we employ recursion.

3.7.8. Proposition. Each lambda-term M has a fixed point F such that $MF =_\beta F$.

It can be uniformly constructed:

3.7.9. Definition (Turing's Fixpoint combinator). $\Theta = HH$ where $H = \lambda hf.f(hhf)$.

3.7.10. Proposition. For any $f \in \Lambda$, $\Theta f \twoheadrightarrow f(\Theta f)$, that is, Θf is a fixed point of f.

Proof. $\Theta f = HHf \twoheadrightarrow f(HHf) = f(\Theta f)$ □

3.7.11. Proposition (Primitive recursion). If $g : \mathbb{N}^k \to \mathbb{N}$ and $h : \mathbb{N}^{k+2} \to \mathbb{N}$ are lambda-definable, then $f : \mathbb{N}^{k+1} \to \mathbb{N}$ with

$$\begin{array}{rcl} f(0, x_1, \ldots, x_k) & = & g(x_1, \ldots, x_k) \\ f(y+1, x_1, \ldots, x_k) & = & h(f(y, x_1, \ldots, x_k), y, x_1, \ldots, x_k) \end{array}$$

is lambda-definable.

Proof. $f_\lambda = \Theta(\lambda F.\lambda\, y\, x_1 \ldots x_k.\mathbf{zero}\, x\, (g_\lambda\, x_1 \ldots x_k)(h_\lambda\, (F\, (\mathbf{pred}\, y)\, x_1 \ldots x_k)\, x_1 \ldots x_k))$. □

3.7.12. Proposition (Minimalization). If $g : \mathbb{N}^{k+1} \to \mathbb{N}$ is lambda-definable, then $f : \mathbb{N}^k \to \mathbb{N}$ with

$$f(x_1, \ldots, x_k) = \min\{y \mid g(y, x_1, \ldots, x_k) = 0\}$$

is lambda-definable.

Proof. $f_\lambda = h\,\lceil 0 \rceil$ where $h = \Theta(\lambda H.\lambda\, y\, x_1 \ldots x_k.\mathbf{zero}(g_\lambda\, y\, x_1 \ldots x_k)\, y\, (H\, (\mathbf{succ}\, y)\, x_1 \ldots x_k))$ □

The previous constructions show:

3.7.13. Theorem (Church). The class of lambda-definable functions is exactly the class of partial recursive functions.

This indicates that questions on lambda terms are hard to answer in general. For instance it is undecidable whether two lambda term have equal normal forms. Otherwise we could decide whether two program texts encode the same function.

3.8 Combinatory Logic

3.8.1. Definition. Combinatory logic with the base combinators $\{C_1, \ldots, C_n\}$, denoted by

$$\mathbf{CL}\,(C_1, \ldots, C_n)$$

is a TRS on the signature $\{\cdot, C_1, \ldots, C_n\}$, where $a\,(\cdot) = 2$ and $a\,(C_i) = 0$ and for each i there is exactly one rule $C_i x_1 \ldots x_k \to M_i$, where M_i is a *pure* term.

3.8.2. Definition. A *pure* term consists entirely of applications and variables, and contains no constants.

We follow the lambda calculus conventions on left-associativity of application, and on omitting the application symbol.
The following combinators were introduced by Curry, Schönfinkel et al.

$$\begin{array}{l}
\mathbf{I}\ x \to x; \mathbf{K}\ x\ y \to x; \mathbf{S}\ x\ y\ z \to x\ z\ \ (y\ z) \\
\mathbf{B}\ x\ y\ z \to x\ \ (y\ z)\,; \mathbf{C}\ x\ y\ z \to x\ z\ y; \mathbf{J}\ x\ y\ z\ w \to x\ y\ \ (x\ w\ z) \\
\mathbf{W}\ x\ y \to x\ y\ y; \mathbf{L}\ x\ y \to x\ \ (y\ y)
\end{array}$$

From the syntactic restrictions of the term algebras and the reduction rules it follows that combinatory logic systems are orthogonal, so they have the Church-Rosser property. Yet they can be quite complicated.
And they are of great entertainment value, see for instance the ornithological book by Smullyan [Smu85]. The appendix of Hindley's and Seldin's text [HS86] tells you how to care for your pet combinators.

3.9 Completeness

3.9.1. Definition. A combinatory TRS is called *combinatorially complete* iff for all $n > 0$ and for all pure terms M with $\mathrm{FVar}(M) \subseteq \{x_1, \ldots, x_n\}$, there is a ground term N such that $N x_1 \ldots x_n \twoheadrightarrow M$.

3.9.2. Theorem. $\mathbf{CL}\,(\mathbf{S}, \mathbf{K}, \mathbf{I})$ is combinatorially complete.

Proof. We define a function $\lfloor\cdot\rfloor_x$ such that $\lfloor M \rfloor_x\ x \twoheadrightarrow M$. The argument of $\lfloor\cdot\rfloor_x$ is any term, and the result is a term where x does not occur.

1. $\lfloor x \rfloor_x = \mathbf{I}$.
2. If $x \ntrianglelefteq M$, then $\lfloor M \rfloor_x = \mathbf{K}\ M$.
3. $\lfloor MN \rfloor_x = \mathbf{S}\ \lfloor M \rfloor_x\ \lfloor N \rfloor_x$.

Now we set $N = \lfloor \ldots\ \lfloor\lfloor M \rfloor_{x_1}\rfloor_{x_2}\ \ldots \rfloor_{x_n}$. Beware: the size of N compared to M might grow exponentially with n. □

Since $\mathbf{S}\ \mathbf{K}\ \mathbf{K}\ x \to \mathbf{K}\ x\ \ (\mathbf{K}\ x) \to x \leftarrow \mathbf{I}\ x$, we can simulate the combinator $\mathbf{I}$ by $\mathbf{S}$ and $\mathbf{K}$:

3.9.3. Corollary. $\mathbf{CL}\,(\mathbf{S}, \mathbf{K})$ is combinatorially complete.

3.10 Non-erasing Combinators

Similar to $\Lambda_{\mathbf{I}}$, there is a non-erasing version of $\mathbf{CL}$, called $\mathbf{CL_I}$, where only combinators with non-erasing reduction rules are allowed. Still there are complete systems in $\mathbf{CL_I}$.

As was the case in $\Lambda_{\mathbf{I}}$, the conservation theorem also holds for $\mathbf{CL_I}$. This can also be derived from proposition 2.6.5.

Although $\mathbf{CL_I}$ is contained in $\mathbf{CL}$, is powerful enough:

3.10.1. Definition. A combinatory TRS is called $\mathbf{CL_I}$*-complete* iff for all $n > 0$ and for all pure terms M with $\mathrm{FVar}(M) = \{x_1, \ldots, x_n\}$, there is a ground term N such that $N x_1 \ldots x_n \twoheadrightarrow M$.

Notice the subtle difference to definition 3.9.1.

3.10.2. Theorem. $\mathbf{CL_I}\,(\mathbf{S}, \mathbf{B}, \mathbf{C}, \mathbf{I})$ is $\mathbf{CL_I}$-complete.

Proof. The definition of $\lfloor\cdot\rfloor_x$ is slightly modified. We define $\lfloor M \rfloor_x$ only for the case that $x \trianglelefteq M$.

1. M is a variable. Then $M = x$, and $\lfloor x \rfloor_x = \mathbf{I}$. The case $M \neq x$ is impossible because x must occur in M.
2. M is an application PQ. Because of $x \trianglelefteq M$, only the following three cases are possible:
   (a) $x \trianglelefteq P$ and $x \ntrianglelefteq Q$. Then $\lfloor PQ \rfloor_x = \mathbf{C}\ \lfloor P \rfloor_x\ Q$.
   (b) $x \ntrianglelefteq P$ and $x \trianglelefteq Q$. Then $\lfloor PQ \rfloor_x = \mathbf{B}\ P\ \lfloor Q \rfloor_x$.
   (c) $x \trianglelefteq P$ and $x \trianglelefteq Q$. Then $\lfloor PQ \rfloor_x = \mathbf{S}\ \lfloor P \rfloor_x\ \lfloor Q \rfloor_x$.

□

3.11 Combinatory Logic and Lambda calculus

There are translations between Λ^0 and $\mathbf{CL}\,(\mathbf{S},\mathbf{K})$, as well as between $\Lambda^0_{\mathbf{I}}$ and $\mathbf{CL}_{\mathbf{I}}\,(\mathbf{S},\mathbf{B},\mathbf{C},\mathbf{I})$.

3.11.1. Definition. The mapping $\langle\cdot\rangle_\Lambda : \mathbf{CL} \to \Lambda^0$ is given by

$$\begin{aligned}
&\langle\mathbf{S}\rangle_\Lambda = \lambda\, x\; y\; z\; .\; x\; z\;\; (y\; z)\,; \langle\mathbf{K}\rangle_\Lambda = \lambda\, x\; y\; .\; x; \langle\mathbf{I}\rangle_\Lambda = \lambda\, x\; .\; x\\
&\langle\mathbf{B}\rangle_\Lambda = \lambda\, x\; y\; z\; .\; x\;\; (y\; z)\,; \langle\mathbf{C}\rangle_\Lambda = \lambda\, x\; y\; z\; .\; x\; z\; y\\
&\langle M \mathbin{\hat{}} N\rangle_\Lambda = \langle M\rangle_\Lambda\; \langle N\rangle_\Lambda
\end{aligned}$$

3.11.2. Proposition. If $M \to_{\mathbf{CL}} N$, then $\langle M\rangle_\Lambda \to^+_\beta \langle N\rangle_\Lambda$.

This translation is not very tight, however, since one reduction step in **CL** corresponds to one *or more* steps in Λ, and the intermediate steps in Λ don't correspond to terms in **CL**. Moreover, normal forms, and normalization, are not preserved.

3.11.3. Definition. The mapping $\langle\cdot\rangle_{\mathbf{CL}} : \Lambda^0 \to \mathbf{CL}$ resp. $\Lambda^0_{\mathbf{I}} \to \mathbf{CL}_{\mathbf{I}}$ is given by

$$\langle M\; N\rangle_{\mathbf{CL}} = \langle M\rangle_{\mathbf{CL}}\; \langle N\rangle_{\mathbf{CL}}; \langle\lambda\, x\; .\; M\rangle_{\mathbf{CL}} = \lfloor\langle M\rangle_{\mathbf{CL}}\rfloor_x$$

where $\lfloor\cdot\rfloor_x$ is defined in the proofs of theorems 3.9.2 or 3.10.2, respectively. $\langle\cdot\rangle_{\mathbf{CL}}$ is in fact a mapping from a lambda term M to **CL**-terms with variables $\mathrm{FVar}(M)$.

3.11.4. Proposition. If $M \to_\beta N$, then $\langle M\rangle_{\mathbf{CL}} \to^+_{\mathbf{CL}} \langle N\rangle_{\mathbf{CL}}$.

This is even less tight, and not at all an "inverse" of $\langle\cdot\rangle_\Lambda$.
For a more detailed analysis of the relation between these two notions of reduction, see [Hin77].

Even if the translations do not faithfully represent single reduction steps, they show that Λ and **CL** have equivalent computational power. So from theorem 3.7.13 we derive

3.11.5. Theorem. The $\mathbf{CL}\,(\mathbf{S},\mathbf{K})$-definable functions are exactly the partial recursive functions.

This shows that $\mathbf{CL}\,(\mathbf{S},\mathbf{K})$ is hard as well. For instance, it is not decidable whether two terms have the same normal form.
As $\mathbf{CL}\,(\mathbf{S},\mathbf{K})$ is an instance of an orthogonal TRS, this question is not decidable for this class of TRS. There are even stronger results.

Returning, for a moment, to the topic of this thesis, it seems not clear what functions are $\mathbf{CL}(\mathbf{S})$-definable. There isn't a self-evident encoding of the naturals, even. Since $\mathbf{CL}(\mathbf{S})$ isn't combinatorially complete, severe restrictions are to be expected, and one could indeed hope that termination could be decidable.

Chapter 4

Finite Automata and Rational Languages

4.1 Introduction

The concept of formal languages was originally invented to eventually capture natural languages as well. Or, even more, to explore machine understanding of language. So the idea of automatons operating in formal languages was at hand.

A finite automaton reads words and performs state transitions. It can only read a word once from left to right, and it only has a finite number of states available.

An automata accepts a word if it starts reading from a prescribed state and finishes reading in a set of prescribed final states. The languages thus acceptable are called *regular*, or *rational*.

In Chomsky's hierarchy of word languages, rational languages are the most simple class. They have many nice properties. Especially, they are closed under Boolean operations.

Finite automata can be generalized so that they operate on arbitrary term algebras. The notions of acceptability and rationality carry over, as well as the fundamental results, including closure under Boolean operations.

The material presented in this chapter is taken from Gécseg's and Steinby's standard book [GS84], except for the section on relations between rational languages and term rewriting systems which is collected from recent publications.

Finite Automata in `RX`

This chapter also serves as a guide to the *implementation* of the `RX` system (whereas chapter A is a guide to the *usage* of that program). Most of the definitions and results given here are directly implemented in `RX`, and we provide pointers to the source code wherever possible.

4.2 Automata and Languages

Automata

The state transitions of an automaton can be seen as relations between terms whose nodes are labeled with the states. So it's no surprise that we use TRS to define finite automata.

We extend a given signature by new nullary symbols which correspond to the states of the automaton.

4.2.1. Definition. An non-deterministic automaton (R, S, S') on the signature (F, a) is a TRS R on the signature $(F \cup S, a)$ where $S = \{s_1, s_2, \ldots\}$ are new constants (with $a(s_i) = 0$), called *states* and a subset S' of S, called *final* or *accepting* states and all rules have a specific form:

The automaton is called *bottom-up* if all rules have the form

$$f(x_1, \ldots, x_n) \to y \quad \text{where} \quad x_i \in S, y \in S$$

The automaton is called *top-down* if all rules have the form

$$y \to f(x_1, \ldots, x_n) \quad \text{where} \quad x_i \in S, y \in S$$

An *epsilon move* is a rule

$$x \to y \quad \text{where} \quad x \in S, y \in S$$

Note that all rules are ground rules. So a finite automaton is a ground TRS.

4.2.2. Definition. The automaton is called *finite* iff S is finite. (The set of rules is then finite as well.)

4.2.3. Definition. A bottom-up automaton is called *deterministic* iff no two left hand sides of the rules coincide.

4.2.4. Definition. A bottom-up automaton is called *complete* iff all possible left hand sides really occur in the rule set.

Implementation

The file `FAtypes.hs` contains data declarations for automata:

1. `BDFA` : bottom-up deterministic
2. `BNFA` : bottom-up non-deterministic
3. `TNFA` : top-down non-deterministic
4. `ETNFA` : top-down non-deterministic with epsilon moves

4.2.5. Example. The following piece of code (taken from that file) declares a `BDFA` with state type `a` as a parameter.

```
data BNFA a = BNFA
        TCons           -- what algebra we're in
        (Set a)         -- all states
        (Set a)         -- accepting states
        (FiniteMap (STerm a) (Set a))   -- transition table
```

Rational Languages

4.2.6. Definition. The language accepted by a bottom-up automaton (R, S, S') on the signature F is

$$\{t \mid t \in T(F), \exists s \in S' : t \twoheadrightarrow_R s\}$$

4.2.7. Definition. The language accepted by a top-down automaton (R, S, S') on the signature F is

$$\{t \mid t \in T(F), \exists s \in S' : s \twoheadrightarrow_R t\}$$

4.2.8. Definition. A language is called *rational* iff it is accepted by some finite automaton.

4.2.9. Definition. Two automata are *equivalent* iff they accept the same language.

4.2.10. Definition. A language L in (F, a) defines an equivalence $\cong_L$ on terms from $T(F)$ by

$$t_1 \cong_L t_2 \quad \text{iff} \quad \forall t \in T(F), p \in \mathrm{Pos}(t) : t\,[p \leftarrow t_1] \in L \Leftrightarrow t\,[p \leftarrow t_2] \in L$$

4.2.11. Theorem. L is rational if $\cong_L$ has finite index.

Proof. Take the equivalence classes of $\cong_L$ as the states of an automaton that accepts L. □

4.3 Variants of Finite Automata

Extended Automata

Syntactic restrictions can be lifted somewhat. This makes writing down automata more convenient.

4.3.1. Definition. An extended automaton on the signature (F, a) is a TRS on the signature $(F \cup S \cup \{u\}, a)$ where S is the set of states as above, and u is a new binary symbol (to be read as "union").
All rules have the form

$$y \to z \quad \text{where} \quad y \in S, z \in T(F \cup S \cup \{u\})$$

4.3.2. Proposition. Each extended automaton has an equivalent (non-extended) automaton.

Proof. Nested constructor applications are be removed by introducing new states. Example:

$$y \to f(\ldots, g(\ldots), \ldots) \quad \text{is replaced by} \quad y \to f(\ldots, z, \ldots); \quad z \to g(\ldots)$$

The u symbol is resolved by introducing two epsilon moves.

$$y \to u(a, b) \quad \text{is replaced by} \quad y \to a \quad \text{and} \quad y \to b$$

□

The file `Gram2FA.hs` contains the function `gram` that converts an expression that describes an extended rational automaton into an `ETNFA`, employing the algorithm just given.

Relations between different automata

4.3.3. Proposition. For each non-deterministic bottom-up (resp. top-down) automaton with epsilon moves there is an equivalent non-deterministic bottom-up (resp. top-down) automaton without epsilon moves.

Proof. Replace the left (right) hand side of each epsilon rule by its epsilon closure, i. e. all states that are reachable by epsilon moves alone. □

4.3.4. Proposition. For each non-deterministic bottom-up automaton there is an equivalent non-deterministic top-down automaton, and vice versa.

Proof. Just reverse the direction of the rules of the automaton's TRS. Note that determinism is lost in such a transformation. □

These conversions are implemented in the file `FAconv.hs`.

4.4 Canonical forms of automata

Trim automata

For this section we adopt the view that automata *generate* (rather that *accept*) a language, i. e. they produce a set of ground terms.

An automaton may contain useless states. A state is called *useless* if it does not take part in the creation of terms.

4.4.1. Definition. In a bottom-up automaton (R, S, S'), the set $P \subseteq S$ of *reachable* states is the smallest set P with the property

$$\text{if} \quad f(q_1, \ldots, q_n) \to p \in R \quad \text{and} \quad \forall i : q_i \in P \quad \text{then} \quad p \in P$$

So P contains all states that produce a non-empty language. Especially, all states that produce constants (with $n = 0$) are elements of P.

Note that the set S' of accepting states doesn't play any role in the definition of P.

4.4.2. Definition. In a bottom-up automaton (R, S, S'), the set $Q \subseteq S$ of *connected* states is the smallest set Q with the properties

$$S' \subset Q$$
$$\text{if} \quad f(q_1, \ldots, q_n) \to p \in R \quad \text{and} \quad p \in Q \quad \text{then} \quad \forall i : q_i \in Q$$

So Q contains the accepting states, and all states that are connected to them.

4.4.3. Definition. An automaton is called *trim* if all states are reachable and connected.

4.4.4. Proposition. For each bottom-up automaton, there is an equivalent trim automaton.

Proof. First delete all states that are not reachable. Then, in remaining automaton, delete all states that are not connected. (The order of these steps is important here.) □

These functions are implemented in the file `FAuseful.hs`.

Determinism

4.4.5. Theorem. For each non-deterministic bottom-up automaton without epsilon moves, there is an equivalent deterministic and complete bottom-up automaton without epsilon moves.

Proof. The set of states of the deterministic automaton is the power set of the states of the non-deterministic automaton. The set of accepting states consists of those subsets of S that contain an element from S'. □

Note that the number of states is likely to explode.

In general, top-down automata cannot be made deterministic.

The implementation is contained in the file `FAdet.hs`. To save time and space, we only generate those elements of the power set that occur as states in a trim deterministic automaton.

Minimal deterministic automata

4.4.6. Definition. A *homomorphism* from an automaton A_1 to A_2 is a mapping from the states of A_1 to the states of A_2 that respects the state transitions and the set of accepting states.

4.4.7. Theorem. For each language L, there is a unique minimal (w.r.t. homomorphism) deterministic bottom-up automaton that recognizes it.

Proof. It is the automaton whose states are the classes of $\cong_L$. □

Note that the deterministic automaton obtained from the power set construction need not be minimal. However:

4.4.8. Proposition. The minimal automaton is homoeomorphically embedded in the power set automaton. It is found by identifying states that are congruent.

Minimalization is implemented in the file `FAmin.hs`. We follow the procedure given in [GS84]. It approximates $\cong_L$, starting with just two classes, containing accepting and non-accepting states. This relation is successively refined by looking for incongruent states, and singling them out into new equivalence classes.

4.4.9. Remark. It would be convenient to have minimal non-deterministic automata, too. But they are not unique, and moreover hard to find. See the papers [ADN92] and [MP95] that describe the situation, and some constructions (for word automata).

They use the idea to construct the minimal deterministic automaton for the mirror image of the given word language. For term languages, the mirror image would correspond to a top-down automaton. However, such an automaton cannot be made deterministic. This shows the difficulty of the problem.

4.5 Operations on rational languages

4.5.1. Proposition. Rational languages are closed under term formation.

This means that for rational languages L_i, the language $L = \{f(x_1, \ldots, x_n) \mid x_i \in L_i\}$ is rational.

Proof. The automaton that accepts L has rules $l \to f(l_1, \ldots, l_n)$ for all tuples of accepting states l_i of the automata for L_i. □

The automaton for L need not be deterministic even if all of the L_i were.

4.5.2. Proposition. Rational languages are closed under union.

Proof. Represent the languages L_1 and L_2 by automata with start sets S_1' and S_2'. The automaton that accepts $L_1 \cup L_2$ has state set $\{s\} \cup S_1 \cup S_2$ and one accepting state s with epsilon moves to (or from) all elements of S_1' and S_2'. □

This goes through for either bottom-up or top-down automata.

Determinism is lost.

The implementation is in the file `FAunion.hs`.

4.5.3. Proposition. Rational languages are closed under intersection.

Proof. Represent the languages L_1 and L_2 by automata without epsilon moves. Name their state sets S_1 and S_2, and their accepting states S_1' and S_2'. The automaton that accepts $L_1 \cap L_2$ has state set $S_1 \times S_2$ and accepting states $S_1' \times S_2'$ □

This goes through for either bottom-up or top-down automata.

If the input automata were deterministic, the result will be, too.

The implementation is in the file `FAintersect.hs`.

4.5.4. Proposition. Rational languages are closed under complement.

Proof. Do the power set construction, making the automaton deterministic. But then call those subsets accepting that don't contain elements from S'. □

The implementation is in the file `FAneg.hs`.

4.5.5. Proposition. Rational languages are closed under set difference.

Proof. Compute $L_1 \setminus L_2$ as the intersection of L_1 with the complement of L_2. □

The implementation is in the file `FAminus.hs`.

4.6 Term Rewriting Systems and Rational Languages

The relationship between TRS and rational languages is an area of ongoing research. This section gives a very brief account of basic results.

Linearization

One point of interest is the set of normal forms in a given TRS. With this respect, only the left hand sides of the rules of the TRS are looked at. The following is a simple observation.

4.6.1. Proposition. If a TRS R is left linear, then the set of R-normal forms is rational.

Proof. This is due to closure properties of rational languages. For any term algebra, the set of all terms is rational. For left linear TRS, the set of all redexes is rational. The set of all terms containing redexes is rational. The set of all terms not containing redexes is rational. This is exactly the set of normal forms. $\square$

See also the example in section A.7 that shows how such a computation is done in the system `RX`.
If the TRS is not linear, then in some cases it is still possible to describe the normal forms, as was shown by Kucherov and Tajine [KT92], Hofbauer and Huber [HH92], Vágvölgyi and Gilleron [VG92].

4.6.2. Theorem. For a TRS R it is decidable whether the set of all R-normal forms is rational. If it is, then a left linear TRS R' can be constructed such that R'-normal forms are R-normal forms.

As we will be looking at $\mathbf{CL}(\mathbf{S})$, which is left linear, we don't need linearization.

Preservation of Regularity

Ground normal forms only depend on left hand sides of rewrite rules, while their right hand sides are ignored. As soon as rewriting really comes into play, things get much more complicated. The set of successors of a rational language isn't necessarily rational, and it is undecidable whether a TRS preserves rationality.

4.6.3. Definition. Let $\to$ be a relation, and L a set of terms. Then

- successors: $\Delta^{\to}(L) = \{t' \mid t \in L, t \to t'\}$,
- predecessors: $\Delta^{\leftarrow}(L) = \{t' \mid t \in L, t \to t'\}$,
- forward closure: $\Delta^{\twoheadrightarrow}(L) = \{t' \mid t \in L, t \twoheadrightarrow t'\}$.
- backward closure: $\Delta^{\twoheadleftarrow}(L) = \{t' \mid t \in L, t \twoheadrightarrow t'\}$.
- congruence closure: $\Delta^{=}(L) = \{t' \mid t \in L, t \leftrightarrow^* t'\}$.

Gilleron and Tison prove [GT95]

4.6.4. Theorem. The following problem is undecidable:
input: TRS R, **question**: is $\Delta^{=}_{R}(L)$ rational for all rational languages L?

Rationality *is* preserved for some restricted classes of TRS.
Dauchet and Tison show (see [DT90])

4.6.5. Theorem. If R is a ground TRS, and L is rational, then $\Delta^{\twoheadrightarrow}_{R}(L)$ and $\Delta^{=}_{R}(L)$ are rational.

This is not a surprise since finite automata and ground rewrite systems are closely related. Salomaa considers monadic TRS (see [Sal88])

4.6.6. Definition. A TRS is *monadic* iff for every rule $l \to r$, $\mathrm{depth}(l) \geq 1$ and $\mathrm{depth}(r) \leq 1$.

4.6.7. Theorem. If R is a right linear monadic TRS, and L rational, then $\Delta_R^{\twoheadrightarrow}(L)$ is rational.

Coquide et. al. generalize this to semi-monadic TRS (see [CDGV91])

4.6.8. Definition. A TRS is *semi-monadic* iff for every rule $l \to r$, either r is a variable or a term $f(t_1, \ldots, t_n)$ where each t_i is a variable or a ground term.

4.6.9. Theorem. If R is a linear semi-monadic TRS, and L rational, then $\Delta_R^{\twoheadrightarrow}(L)$ is rational.

We will soon be looking at **CL(S)**, which does not belong to any of these classes. (Neither does **CL(S)** with the arrow pointing backwards.) Therefore it is somehow surprising that rational languages are usable in describing certain properties of **CL(S)**.

Approximation of forward closure

Even if $\Delta^{\twoheadrightarrow}(L)$ for rational L is not rational, we might try to approximate it by a rational superset. We follow the presentation by Genet [Gen97].

Let R be the term rewriting system under consideration, and let L be produced by the automaton (A, S, S'). So A is a ground term rewriting system.

The idea is to look for possible R-redexes inside A. For a rule $l \to_R r \in R$, we look for substitutions σ with $l\sigma \twoheadrightarrow_A q$. That means that the state q of the automaton produces some redexes. We now apply the rewrite rule to the redex, and add to A a state transition from its contractum to q: $r\sigma \to_A q$. So both the redex and the contractum can now be produced from the state q.

In fact $r\sigma$ need not be monadic, so the rule cannot be directly added because it might be of an extended form. This has to be resolved as described in proposition 4.3.2. In doing so, some new states have to be introduced. Therefore, the process isn't guaranteed to terminate, as new states might incur new redex positions. So one needs a clever algorithm that tries to re-use already existing states of the automaton for the resolution of the $r\sigma \to q$ rules.

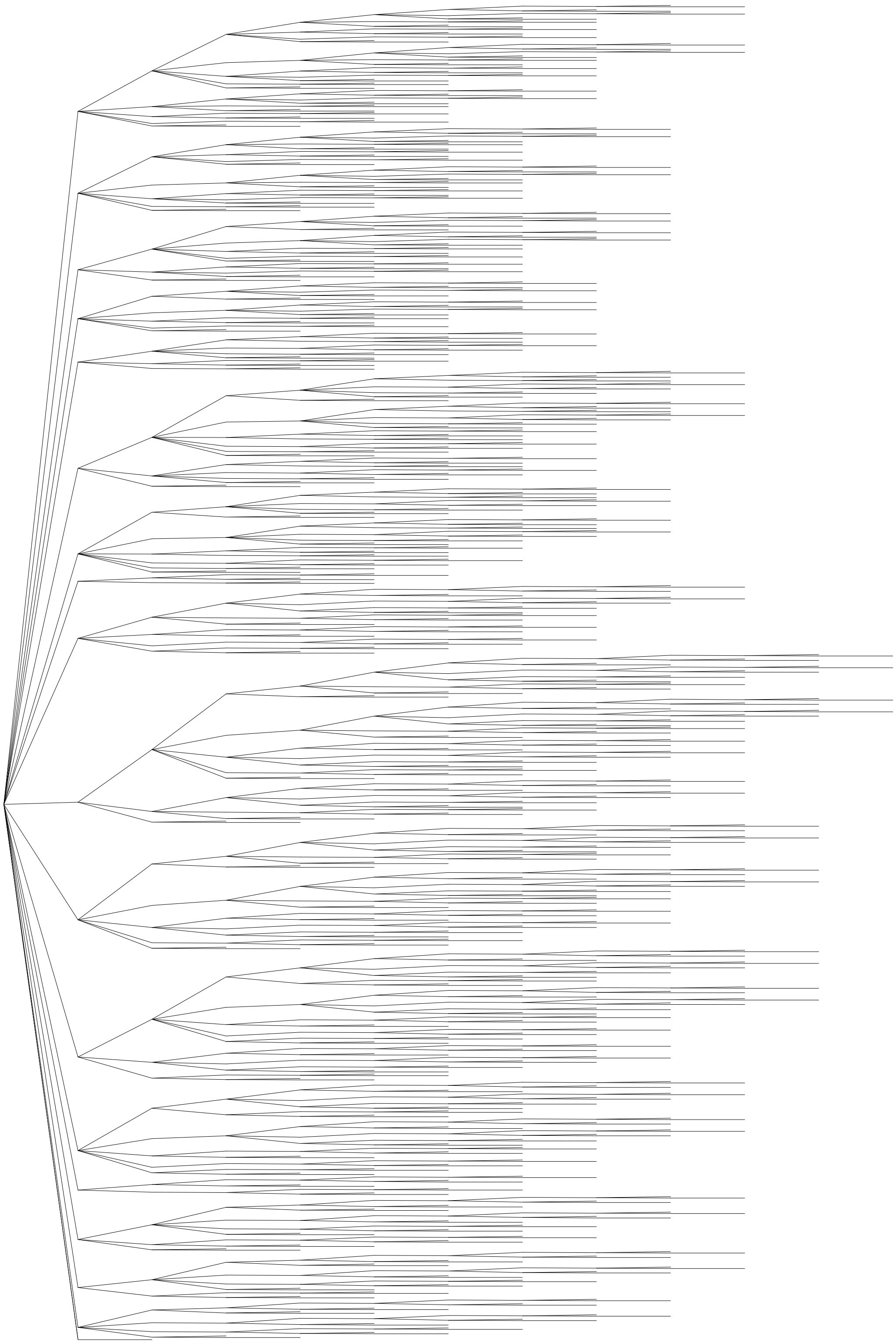
We call A *saturated* w.r.t. R if for all $l \to_R r \in R$, for all $q \in S$, for all $l\sigma \twoheadrightarrow_A q$ we also have $r\sigma \twoheadrightarrow_A q$. It is clear that a saturated automaton that originally contained L gives a rational approximation of $\Delta^{\twoheadrightarrow}(L)$ from above.

For the **CL(S)** case, fortunately, for some L saturated automata can be obtained. In fact we guess languages L that are already saturated, proving $L = \Delta^{\twoheadrightarrow}(L)$ or $L = \Delta^{\twoheadleftarrow}(L)$ (by reversing arrows in R).

Obviously, saturation of an automaton is a decidable property, and `RX` implements a test for saturation.

Part II

The Combinator S



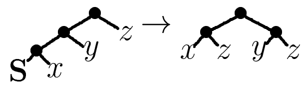
S (**S S S**) **S** (**S** (**S S**) **S**)
after 50 head reductions

Chapter 5

Introduction to CL(S)

5.1 Introduction

CL(**S**) is a combinatory term rewriting system with just one combinator



All combinatory TRS are left linear and non-overlapping. By proposition 2.5.5, they are confluent. Therefore normal forms are unique if they exist.

Moreover, the set $\mathcal{N}$ of normal forms in **CL**(**S**) is rational by proposition 4.6.1.

Furthermore, **CL**(**S**) is non-erasing, so normalization and termination coincide by proposition 2.6.5.

These properties are shared by all non-erasing **CL** systems.

But from now on, we will investigate properties that are special to **CL**(**S**). In the present chapter, we will first fix a convenient notation, and then present basic observations.

They will prepare for the subject treatment in the subsequent chapters.

5.2 Notations

Symbols

Symbols appear in different fonts. We use

- bold face, upper case, for combinators $(\mathbf{S}, \mathbf{K}, \ldots)$,
- italic face, lower case, for variables in the system $(x, y, \ldots)$,
- italic face, upper case, for meta-variables (that denote terms or sets of terms) $(X, Y, \ldots)$,
- calligraphic face, upper case, for fixed sets of terms $(\mathcal{P}, \mathcal{Q}, \ldots)$,
- sans serif face, upper case, for certain fixed terms $(\mathsf{T} = \mathsf{S}\,\mathsf{S},\ \mathsf{A} = \mathsf{T}\,\mathsf{S} = (\mathsf{S}\,\mathsf{S})\,\mathsf{S})$.

We sometimes switch between one- and two-dimensional representation of terms: $\mathbf{S}\ \mathbf{S}\ \mathbf{S} = (\mathsf{S}\,\mathsf{S})\,\mathsf{S}$.
Moreover, since the rest of this thesis exclusively deals with the $\mathbf{S}$ combinator, we often write it as $\circ$ from now on.

Sets of Terms

5.2.1. Definition. $\mathcal{N}$ denotes the set of all terms in *normal* form.

5.2.2. Proposition. $\mathcal{N} = \mathbf{S} \ \cup \ \mathsf{S}\,\mathcal{N} \ \cup \ (\mathsf{S}\,\mathcal{N})\,\mathcal{N}$

5.2.3. Definition. $\mathcal{M}$ denotes the set of *all* terms.

5.2.4. Proposition. $\mathcal{M} = \mathbf{S} \ \cup \ \mathcal{M}\,\mathcal{M}$

Inside an expression, we often just write $*$ for $\mathcal{M}$.
We often write a term when meaning a set containing exactly this term.
Operations are understood to be extended from terms to sets of terms. For instance, $(\mathsf{S}\,(\mathsf{S} \cup \mathsf{T}))\,\mathsf{T}$ denotes the set $\left\{(\mathsf{S}\,\mathsf{S})\,\mathsf{T}, (\mathsf{S}\,\mathsf{T})\,\mathsf{T}\right\}$.
If E and F are sets, $E \to F$ denotes $\forall X \in E : \exists Y \in F : X \to Y$, while $E \not\to F$ stands for $\forall X \in E : \neg\exists Y \in F : X \to Y$.
The set $\mathcal{M}_k$ consists of all terms whose left spine has length $\geq k$.

5.2.5. Definition. $\mathcal{M}_0 = \mathcal{M}; \mathcal{M}_{k+1} = \mathcal{M}_k\,\mathcal{M}$.

For example, $\mathcal{M}_0$ are all terms, and $\mathcal{M}_1$ are all terms except $\mathbf{S}$.
The term S_k has a right spine of length $k-1$ and all its left children are $\mathbf{S}$:

5.2.6. Definition. $\mathsf{S}_1 = \mathbf{S}; \mathsf{S}_{k+1} = \mathsf{S}\,\mathsf{S}_k$; $\mathcal{P}_k = \mathsf{S}_1 \ \cup \ \ldots \ \cup \ \mathsf{S}_k$; $\mathcal{Q}_k = \mathcal{M} \setminus \mathcal{P}_k$.

The set $\mathcal{P}_k$ is the collection of all S_i up to k. The complement of $\mathcal{P}_k$ is denoted by $\mathcal{Q}_k$. These are exactly the terms whose right spine has a length $\geq k$ or who have some subterm with left spine longer than 1. For example, $\mathcal{Q}_0$ are all terms, $\mathcal{Q}_2$ are all terms except $\mathbf{S}$ and T, and $\mathsf{A} = (\mathsf{S}\,\mathsf{S})\,\mathsf{S} \in \mathcal{Q}_k$ for any k.
From the above description we immediately derive

5.2.7. Corollary. For all k we have $\mathcal{P}_k \subseteq \mathcal{P}_{k+1}$ and $\mathcal{Q}_k \supseteq \mathcal{Q}_{k+1}$.

5.2.8. Corollary. If $X \in \mathcal{Q}_k$, and $X \lhd Y$, then $Y \in \mathcal{Q}_{k+1}$

We collect all of the $\mathcal{P}_*$ into the set $\mathcal{P}$. These are all terms whose left children of the right spine are leaves. The complement of $\mathcal{P}$ is denoted by $\mathcal{Q}$.

5.2.9. Definition. $\mathcal{P} = \bigcup_{k\geq 1} \mathsf{S}_k; \quad \mathcal{Q} = \mathcal{M} \setminus \mathcal{P}$

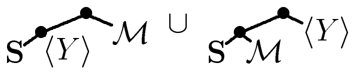
5.2.10. Corollary. For all k we have $\mathcal{P}_k \subseteq \mathcal{P}$ and $\mathcal{Q} \subseteq \mathcal{Q}_k$.

5.2.11. Corollary. $\mathcal{P} = \mathbf{S} \cup \mathbf{S}^\wedge\mathcal{P}; \mathcal{Q} = \mathcal{M}_2 \cup \mathbf{S}^\wedge\mathcal{Q}$.

5.2.12. Corollary. If $X \in \mathcal{Q}$ and $X \lhd Y$, then $Y \in \mathcal{Q}$.

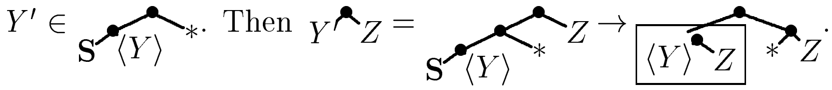
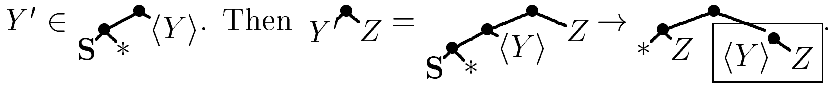
Patterns

Certain subterm positions are related to certain reduction properties. If Y is some expression, we'll define the set $\langle Y\rangle$ of *triads* over Y. It has the property $\langle Y\rangle^\wedge Z \hookrightarrow\!\!\!\!\rightarrow Y^\wedge Z$.

5.2.13. Definition. $\langle Y\rangle = Y \cup$ 

5.2.14. Proposition. $\langle Y\rangle^\wedge Z \hookrightarrow\!\!\!\!\rightarrow Y^\wedge Z$

Proof. The claim is proved by structural induction. Let $Y' \in \langle Y\rangle$.

1. $Y' = Y$. Then the claim is vacuously true.
2. 
3. 

$\square$

Another construction is the stairway pattern $\langle {}^X\!/\,{}_Y\rangle$.

5.2.15. Definition. $\langle {}^X\!/\,{}_Y\rangle = Y \cup X^\wedge\langle {}^X\!/\,{}_Y\rangle$

In the literature this is also denoted by X^*Y but we chose this notation because of the similarity to $\langle Y\rangle$, of which the stairways are special case in the following situation:

5.2.16. Corollary. If $X \in \mathbf{S}^\wedge *$, then $\langle {}^X\!/\,{}_Y\rangle \subseteq \langle Y\rangle$.

5.3 Predecessors and Successors

5.3.1. Proposition. For any **S**-term Y, the set of predecessors $\Delta^{\twoheadleftarrow}(Y)$ is finite and computable.

Proof. If $X \to Y$, then $|X| \leq |Y|$. There are only finitely many terms X that fulfill this condition. They can all be enumerated, and $X \to Y$ can be checked. This generates all *immediate* predecessors of Y.

Their immediate predecessors can be computed as well, and this process can be continued, but it must eventually stop (when all terms no larger than $|Y|$ have been checked). □

5.3.2. Remark. So far, this says nothing about the presence or absence of *cycles* $Y \to^{+} Y$. This will be dealt with in proposition 6.2.2.

5.3.3. Proposition. Given **S**-terms X and Y, the question "does $X \twoheadrightarrow Y$ hold?" is decidable.

Proof. Just enumerate all predecessors of Y and check whether X occurs. □

5.3.4. Remark. The question "does $X \leftrightarrow^{*} Y$ hold?" is the *word problem.* We conjecture that it is decidable as well. However, with current knowledge, a proof seems out of reach.

5.3.5. Remark. What happens with $\Delta^{\twoheadleftarrow}(L)$ for rational L? We do not know. It it were rational (and effectively computable from an automaton for L), then $\Delta^{\twoheadleftarrow}(\mathcal{N})$ would immediately solve the question of normalization.

Because **CL**(**S**) is duplicating, $\Delta^{\twoheadrightarrow}(L)$ and even $\Delta^{\rightarrow}(L)$ need not be rational for rational L.

5.3.6. Example. $\mathsf{A}\,\mathcal{P}$ is rational, but $\Delta^{\rightarrow}\left(\mathsf{A}\,\mathcal{P}\right) = \Delta^{\twoheadrightarrow}\left(\mathsf{A}\,\mathcal{P}\right) = \left\{\mathsf{S}_k\,\mathsf{S}_k \mid k > 1\right\}$ is not.

While for singleton L, the set $\Delta^{\twoheadleftarrow}(L)$ is finite (therefore rational), even in this case $\Delta^{\twoheadrightarrow}(L)$ need not be rational, as shown by terms with infinite reductions.

However there *are* rational sets of **S**-terms that behave well under reduction.

5.3.7. Definition. $\mathcal{N}_{00} = \mathbf{S} \cup \mathsf{T}\,\mathcal{N}_{00} \cup \mathcal{N}_{00}\,\mathsf{S}$

5.3.8. Proposition. $\mathcal{N}_{00}$ is forward and backward closed under reduction: $\mathcal{N}_{00} = \Delta^{=}(\mathcal{N}_{00})$.

Proof. As set forth in section 4.6, saturation of $\mathcal{N}_{00}$ w.r.t. **CL**(**S**) (with the arrow pointing forwards and backwards) can be checked. We indicate the check for the forward closure: From the grammar for $\mathcal{N}_{00}$ we derive that a redex in $\mathcal{N}_{00}$ must have the form $((\mathsf{S}\,\mathsf{S})\,\mathcal{N}_{00})\,\mathsf{S}$. Its contractum is $(\mathsf{S}\,\mathsf{S})\,(\mathcal{N}_{00}\,\mathsf{S}) = \mathsf{T}\,(\mathcal{N}_{00}\,\mathsf{S}) \subseteq \mathsf{T}\,\mathcal{N}_{00} \subseteq \mathcal{N}_{00}$. □

5.4 Monsters

5.4.1. Definition. A *monster* is a term with normal form that is very large and many reduction steps away.

To get an impression of what *very* and *many* could mean, we look at an example.

5.4.2. Definition. $\mathsf{A}_0 = \mathsf{S}; \mathsf{A}_{n+1} = \mathsf{T}\,\mathsf{A}_n \quad \mathsf{T}_1 = \mathsf{T}; \mathsf{T}_{m+1} = \mathsf{T}\,\mathsf{T}_m$

5.4.3. Example. Names are chosen such that $\mathsf{A}_1 = \mathsf{A}; \mathsf{T}_1 = \mathsf{T}$

5.4.4. Proposition. $(\mathsf{A}_n\,\mathsf{A}_m)\,\mathsf{S}$ are monsters. Their normal form has size $n^{\Theta(m)}$ and is reachable in $\Theta(n \cdot m)$ steps.

For the proof, we additionally define

$$\mathsf{B}_0(X) = \mathsf{S}\,X; \mathsf{B}_{m+1}(X) = (\mathsf{S}\,X)\,\mathsf{B}_m(X); \mathsf{C}_0(X) = \mathsf{T}\,X; \mathsf{C}_{m+1}(X) = (\mathsf{S}\,X)\,\mathsf{C}_m(X)$$

$$\mathsf{D}_{0,n} = (\mathsf{S}\,\mathsf{A}_n)\,\mathsf{S}; \mathsf{D}_{m+1,n} = \mathsf{C}_{n+1}(\mathsf{D}_{m,n})$$

Note that all these are normal forms (if X is a normal form).

5.4.5. Lemma. $\mathrm{nf}\left(\mathsf{A}_m\,X\right) = \mathsf{B}_m(X)$ as well as $\mathrm{nf}\left(\mathsf{T}_{m+1}\,X\right) = \mathsf{C}_m(X)$, and both reductions take m steps.

Proof. $\mathsf{A}_0\,X = \mathsf{S}\,X = \mathsf{B}_0(X)$.

$\mathsf{A}_{m+1}\,X = (\mathsf{T}\,\mathsf{A}_m)\,X \to (\mathsf{S}\,X)\,\boxed{\mathsf{A}_m\,X} \twoheadrightarrow (\mathsf{S}\,X)\,\mathsf{B}_m(X) = \mathsf{B}_{m+1}(X)$.

$\mathsf{T}_1\,X = \mathsf{T}\,X = \mathsf{C}_0(X)$.

$\mathsf{T}_{m+1}\,X = (\mathsf{T}\,\mathsf{T}_m)\,X \to (\mathsf{S}\,X)\,\boxed{\mathsf{T}_m\,X} \twoheadrightarrow (\mathsf{S}\,X)\,\mathsf{C}_m(X) = \mathsf{C}_{m+1}(X)$. □

By replacing X with S in this proof, and using the identities $\mathsf{A}_0\,\mathsf{S} = \mathsf{S}\,\mathsf{S} = \mathsf{T}_1$ and $\mathsf{T}_1\,\mathsf{S} = (\mathsf{S}\,\mathsf{S})\,\mathsf{S} = \mathsf{A}_1$, we obtain

5.4.6. Lemma. $\mathrm{nf}\left(\mathsf{A}_m\,\mathsf{S}\right) = \mathsf{T}_{m+1}$ as well as $\mathrm{nf}\left(\mathsf{T}_m\,\mathsf{S}\right) = \mathsf{A}_m$, and reductions take m and $m-1$ steps, respectively.

5.4.7. Lemma. $\mathrm{nf}\left(\mathsf{B}_m(\mathsf{A}_n)\,\mathsf{S}\right) = \mathsf{D}_{m,n}$. It is reached in $2 \cdot m \cdot n$ reduction steps.

Proof. $\mathsf{B}_0(\mathsf{A}_n)\,\mathsf{S} = (\mathsf{S}\,\mathsf{A}_n)\,\mathsf{S}$.

$\mathsf{B}_{m+1}(\mathsf{A}_n)\,\mathsf{S} = ((\mathsf{S}\,\mathsf{A}_n)\,\mathsf{B}_m(\mathsf{A}_n))\,\mathsf{S} \to \boxed{\mathsf{A}_n\,\mathsf{S}}\,\boxed{\mathsf{B}_m(\mathsf{A}_n)\,\mathsf{S}}$, using one reduction step.

By lemma 5.4.6, $\mathsf{A}_n\,\mathsf{S} \twoheadrightarrow \mathsf{T}_{n+1}$ in n steps.

By induction, $\mathsf{B}_m(\mathsf{A}_n)\,\mathsf{S} \twoheadrightarrow \mathsf{D}_{m,n}$ in $2 \cdot m \cdot n$ steps.

Therefore the whole expression reduces to $\mathsf{T}_{n+1}\,\mathsf{D}_{m,n}$ in $(2 \cdot m + 1) \cdot n$ steps.

This again has normal form $\mathsf{C}_{n+1}(\mathsf{D}_{m,n}) = \mathsf{D}_{m+1,n}$ which is reached in n more steps by lemma 5.4.5.

In all, this were $(2 \cdot m + 2) \cdot n$ steps. □

So we proved that the normal forms are $\mathsf{D}_{m,n}$ indeed. Now we compute their sizes.

5.4.8. Proposition. $|\mathsf{C}_m(X)| = \Theta(m \cdot |X|)$

Proof. $|\mathsf{C}_0(X)| = |\mathbf{S}\ \mathbf{S}\ X| = 4 + |X|$.

$|\mathsf{C}_{m+1}(X)| = |\mathbf{S}\ X\ (\mathsf{C}_m(X))| = 3 + |X| + |\mathsf{C}_m(X)|$. ◻

5.4.9. Proposition. $|\mathsf{D}_{m,n}| = n^{\Theta(m)}$

Proof. $|\mathsf{D}_{0,n}| = |\mathbf{S}\ \mathsf{A}_n\ \mathbf{S}| = \Theta(n)$.

$|\mathsf{D}_{m+1,n}| = |\mathsf{C}_{n+1}(\mathsf{D}_{m,n})| = \Theta((n+1) \cdot |\mathsf{D}_{m,n}|)$. ◻

Proof of proposition 5.4.4. $\mathsf{A}_m\ \mathsf{A}_n\ \mathbf{S} \twoheadrightarrow \mathsf{B}_m(\mathsf{A}_n)\ \mathbf{S}$, using m steps, by lemma 5.4.5. $\mathsf{B}_m(\mathsf{A}_n)\ \mathbf{S} \twoheadrightarrow \mathsf{D}_{m,\mathsf{A}_n}$, using $2 \cdot m \cdot n$ steps, by lemma 5.4.7. ◻

5.4.10. Proposition. $\mathbf{S}\ \mathsf{A}\ \mathsf{T}_n\ \mathbf{S}\ \mathbf{S}\ \mathbf{S}$ are monsters. Their normal form has size $n^{\Theta(n)}$, and it is reached in at least $\Theta(n^2)$ steps.

Proof. $\mathsf{S}\ \mathsf{A}\ \mathsf{T}_n\ \mathsf{S}\ \mathsf{S}\ \mathsf{S} \to \mathsf{A}\ \mathsf{S}\ \mathsf{T}_n\ \mathsf{S}\ \mathsf{S} \to \mathsf{T}\ \mathsf{T}\ \mathsf{T}_n\ \mathsf{S}\ \mathsf{S} \to \mathsf{S}\ \mathsf{T}_n\ (\mathsf{T}\ \mathsf{T}_n)\ \mathsf{S}\ \mathsf{S} = \mathsf{S}\ \mathsf{T}_n\ \mathsf{T}_{n+1}\ \mathsf{S}\ \mathsf{S} \to \underline{\mathsf{T}_n\ \mathsf{S}}\ \underline{(\mathsf{T}_{n+1}\ \mathsf{S})}\ \mathsf{S} \twoheadrightarrow \mathsf{A}_n\ \mathsf{A}_{n+1}\ \mathsf{S}$. ◻

We note that T_n and A_n are normal forms of *flat* **S**-terms, for instance $\mathbf{S}\,\mathbf{S}\,\mathbf{S}\,\mathbf{S}\,\mathbf{S}\,\mathbf{S} \twoheadrightarrow \mathsf{T}\ (\mathsf{T}\ \mathsf{T}) = \mathsf{T}_3$.

5.4.11. Example. $\mathsf{S}\ \mathsf{A}\ (\mathbf{S}\,\mathbf{S}\,\mathbf{S}\,\mathbf{S}\,\mathbf{S}\,\mathbf{S})\ \mathbf{S}\ \mathbf{S}\ \mathbf{S}$ has a normal form of size 6131, which is reached after 55 steps.

We conjecture that there are no monsters that grow substantially larger. However, a proof for a statement like "all terms X reach their normal form in $O\left(|X|^2\right)$ innermost reduction steps" doesn't seem immediately at hand. It would entail an algorithm that decides termination.

A History lesson on monsters

The following information was kindly provided by Henk Barendregt.

van der Poel and him were looking for monsters in **CL**(**S**) w.r.t. *strong* reduction (see section 5.6) and found **S** T **S** T **S** and **S** T T T **S** **S**. See also [vdPSvdM80], Appendix E.

The first expression has a strong normal form roughly of size 10^6. The second one reduces in one step to $\mathsf{T}\ \mathsf{T}\ (\mathsf{T}\ \mathsf{T})\ \mathbf{S} = \mathsf{T}_2\ \mathsf{T}_2\ \mathbf{S}$, where $\mathsf{T}_m\ \mathsf{T}_n\ \mathbf{S}$ can be shown to be monsters by similar arguments as presented above.

5.5 Terms without normal form

We give two examples of infinite reductions in **CL**(**S**).

An infinite, unique reduction

5.5.1. Example. The expression $X_0 X_0$ where $\mathsf{T} = \mathsf{S}\mathsf{S}$; $X_0 = (\mathsf{S}\mathsf{T})\mathsf{T}$; $X_{n+1} = \mathsf{T} X_n$ has an infinite reduction. It has already been described by E. Zachos in [Zac78]. All redexes that occur during the reduction have the form $X_n X_m$ for some n and m. There are two types of reduction steps: one goes *downward*: $X_{n+1} X_m = ((\mathsf{S}\mathsf{S})X_n)X_m \to (\mathsf{S}X_m)\boxed{X_n X_m}$ and the other goes *upward*: $X_0 X_m = ((\mathsf{S}\mathsf{T})\mathsf{T})X_m \to (\mathsf{T}X_m)(\mathsf{T}X_m) = X_{m+1} X_{m+1}$

It is interesting to note that the reduction graph of $X_0 X_0$ is a *pure line* (each expression contains exactly one redex).

An infinite head reduction

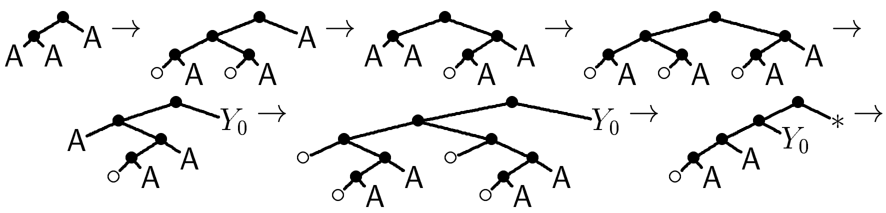
5.5.2. Example. The expression

$$(\mathsf{A}\mathsf{A})\mathsf{A} \quad \text{where} \quad \mathsf{A} = (\mathsf{S}\mathsf{S})\mathsf{S}$$

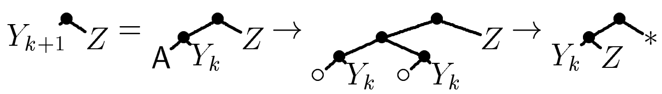
has an infinite reduction. It had been found by H. Barendregt, see [Bar84]. Again the reduction can be described by a pattern, but it is not so obvious:

$$Y_0 = (\circ\mathsf{A})((\circ\mathsf{A})\mathsf{A});\ Y_{n+1} = \mathsf{A}Y_n$$

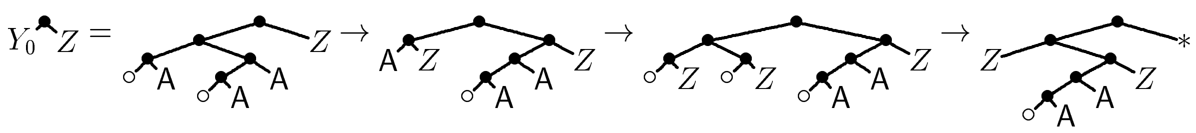
First we show that the pattern is reached.

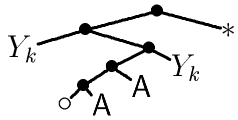


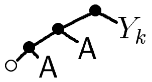
(Here, $*$ denotes a subexpression whose value is irrelevant because no later reduction touches it.) Then we show how the pattern reduces. For any Z, there is a *downwards* move that in general goes like

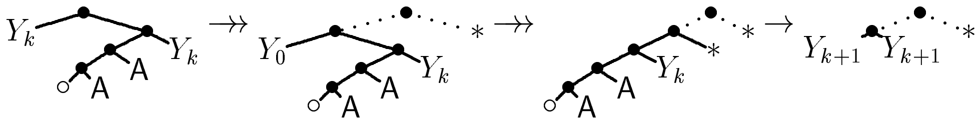


and an *upwards* move



The placeholder Z is instantiated with Y_k to give $Y_0\,Y_k \twoheadrightarrow$ 

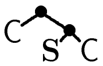
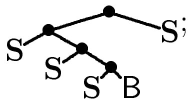
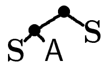
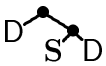
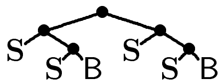
and now $Z =$  gives



This is an infinite head reduction.

More infinite unique reductions

J. W. Klop raised the question whether there are any pure lines in **CL**(**S**) (question 3.3 in [Klo80]). We just saw one example $X_0\,X_0$. Indeed there are more, and their typical start terms are:

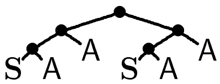
1.  where C = ; B = 
2.  where D = 

They have infinite, unique head reductions.

For a complete formulation of this statement, and a proof, see the *very* technical report [Wal96].

A history lesson on terms without normal form

(The following information was kindly provided by Henk Barendregt and Jan Willem Klop)

In 1975, Klop, Bergstra, Volken and Barendregt placed a 'bet' on finding an expression in **CL**(**S**) that has no normal form. Barendregt won with , and a similar term  with a simpler proof.

Later Dobouè found the shortest term without normal form: **S T S S S S**, and its reduct **A T S S S**, but without proof, which was later supplied by Monique Baron, and simplified by Klop, Bergstra, Volken, and Barendregt.

The paper [vdPSvdM80], Appendix E, mentions the expression **S T A T S** of which no normal form could be found.

5.6 Weak and Strong reduction

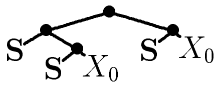
This thesis considers reductions in the **CL**(**S**) term model only. One could also consider the translation of **CL**(**S**) into Λ^0 via the function $\langle\cdot\rangle_\Lambda$, and then do rewriting there. In this respect, $\rightarrow_{\mathbf{CL}}$ is called *weak* reduction and $\rightarrow_\beta$ is called *strong* reduction. These two concepts differ substantially.

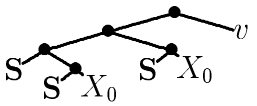
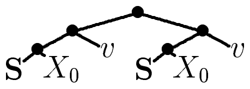
5.6.1. Proposition. A weak normal form need not be in strong normal form.

5.6.2. Example. $\mathsf{S}{}^\wedge\mathsf{S}$ is in weak normal form. Its strong normal form is

$$\lambda\, x\, y\, z\, .\ (x\ z\ (y\ z))\ \mathbf{S} \rightarrow \lambda\, y\, z\, .\ (\mathbf{S}\ z\ (y\ z)) = \lambda\, y\, z\, .\ (\lambda\, x'\, y'\, z'\, .\ (x'\ z'\ (y'\ z'))\ z\ (y\ z)) \twoheadrightarrow \lambda\, y\, z\, .\ (\lambda\, z'\ (z\, z'\ (y\ z\, z')))$$

5.6.3. Proposition. There is a term that is in weak normal form, but has no strong normal form at all.

5.6.4. Example. We use notation of example 5.5.1. The term is $Y =$ . It is enough to show that $Y{}^\wedge v$ for a variable v, has no weak normal form.

 $\rightarrow$  $\subseteq \langle X_0\rangle{}^\wedge\langle X_0\rangle$

by slightly abusing the definition of $\langle\cdot\rangle$. Roughly speaking, this behaves like $X_0{}^\wedge X_0$ which has no normal form. (A full proof would apply proposition 8.2.3.)

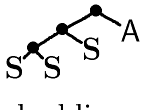
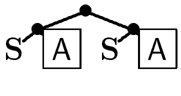
5.7 Homeomorphic Embeddings

By Kruskal's theorem 2.8.3, each infinite reduction $X_1 \rightarrow X_2 \rightarrow \ldots$ contains X_i and X_j with $i < j$ and $X_i \leq_{\mathrm{HE}} X_j$.

5.7.1. Example. See example 5.5.1. For all $m < n$, we have $X_m{}^\wedge X_m \hookrightarrow\!\!\!\!\rightarrow X_n{}^\wedge X_n$. From $X_m \lhd X_n$ we derive $X_m{}^\wedge X_m \leq_{\mathrm{HE}} X_n{}^\wedge X_n$ which uniformly gives the desired embeddings.

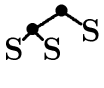
However the terminating reductions don't necessarily *omit* embeddings:

5.7.2. Proposition. There are $X, Y \in$ **CL**(**S**) such that $X \rightarrow^+ Y$ and $X \leq_{\mathrm{HE}} Y$, but $\downarrow(X)$.

5.7.3. Example. $\mathsf{A}{}^\wedge\mathsf{A} =$  $\rightarrow$  $\geq_{\mathrm{HE}} \mathsf{A}{}^\wedge\mathsf{A}$

So the homeomorphic embedding relation isn't really informative in **CL**(**S**).
This is also shown by

5.7.4. Proposition. There are $X, Y \in$ **CL**(**S**) where $X \leq_{\mathrm{HE}} Y$, and $X \twoheadrightarrow \infty$, but $\downarrow(Y)$.

5.7.5. Example. $X =$ **S** T **S S S S** has no normal form, but $Y =$ **S** A **S S S S** has one. Still $X \leq_{\mathrm{HE}} Y$ because of $\mathsf{T} = \mathsf{S}{}^\wedge\mathsf{S} \leq_{\mathrm{HE}}$  $= \mathsf{A}$

Chapter 6

Loops

6.1 Introduction

Often infinite reductions in term rewriting systems can be described by patterns that re-appear, possibly in varied instantiations.

6.1.1. Definition. A *loop* is a reduction $X \to^+ C[X\sigma]$, where X is a term possibly containing variables, $C[\cdot]$ is a context, and σ is a substitution.

A loop can be composed with itself, giving an infinite reduction. For ground TRS, infinite reductions necessarily contain ground loops, and it is decidable whether such a TRS admits them.

In general, however, presence of loops is not decidable, and non-termination not necessarily means that loops exist.
For instance,

6.1.2. Conjecture. $\mathbf{CL}(\mathbf{S})$ admits no loops.

Empirical observations support that. We apply two different methods that prove weakened versions of that claim.

For completeness, we start by listing yet another method. It is already known in the literature. This is a potential method due to J. Bergstra that shows the absence of *cycles* from $\mathbf{CL}(\mathbf{S})$.
Then we define a notion of *monotone* term which helps to show the absence of certain (not all) *ground* loops.
This method can easily be applied to other rewrite systems. We do so for head reductions in $\mathbf{CL}(\mathbf{S}, \mathbf{K})$.
After that we'll turn to excluding *all* ground loops from $\mathbf{CL}(\mathbf{S})$, using quite a different approach.
This proof uses recursive path orderings. It should be applicable to other rewrite systems as well.
Yet all attempts fail to show the absence of *arbitrary* loops from $\mathbf{CL}(\mathbf{S})$.

Loops have been investigated by Zantema and Geser [ZG96]. They present a one-rule TRS that is non-terminating, but non-looping; and also a two-rule string rewriting system with that property.

6.2 Cycles

6.2.1. Definition. A *cycle* is a reduction $X \to^+ X$.

A cycle can be composed with itself infinitely often, so the presence of cycles shows non-termination of the rewriting system.

6.2.2. Proposition. $\mathbf{CL}(\mathbf{S})$ admits no cycles.

(The following proof is due to J. Bergstra and J. W. Klop, [BK79].)
In $L = \mathbf{S}\ X\ Y\ Z \to X\ Z\ (Y\ Z) = R$, we have $|L|$ with $|R|$ iff $Z = \mathbf{S}$, else $|L| < |R|$.
Now define a potential function $\|\mathbf{S}\| = 1; \|X\ Y\| = 2 \cdot \|X\| + \|Y\|$. Then

$$\begin{aligned} \|L\| &= \|\mathbf{S}\ X\ Y\ Z\| = 8 + 4 \cdot \|X\| + 2 \cdot \|Y\| + \|Z\| \\ \|R\| &= \|X\ Z\ (Y\ Z)\| = 4 \cdot \|X\| + 2 \cdot \|Y\| + 3 \cdot \|Z\| \\ \|R\| - \|L\| &= 2 \cdot \|Z\| - 8 \end{aligned}$$

Observe that if $Z = \mathbf{S}$, then $\|L\| > \|R\|$.
Both $|\cdot|$ and $\|\cdot\|$ are strictly monotone w. r. t. term construction:
6.2.3. Lemma. If $\|Y\| < \|Z\|$, then for all X and for all $p \in \operatorname{Pos}(X)$: $\|X[p \leftarrow Y]\| < \|X[p \leftarrow Z]\|$.

Then we conclude:

6.2.4. Lemma. If $Y \to Z$, then

1. $|Y| < |Z|$
2. or $|Y| = |Z|$ and $\|Y\| > \|Z\|$.

This leads to

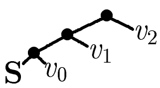
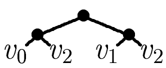
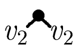
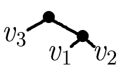
6.2.5. Lemma. If $Y \to^+ Z$, then $(|Y|, \|Y\|) \neq (|Z|, \|Z\|)$

Therefore $X \neq Y$, and there is no $X \to^+ X$.
It would be nice to have more of such potential functions to solve other problems in $\mathbf{CL}(\mathbf{S})$.

6.3 Monotone Terms

We consider $\mathbf{CL}(\mathbf{S}, V)$, terms with variables from a set V that is totally ordered: $v_0 < v_1 < v_2 < \ldots$.

6.3.1. Definition. A term $X \in \mathbf{CL}(\mathbf{S}, V)$ is called *monotone* iff for all  $\trianglelefteq X$ with $w \in V$, and for all $v \trianglelefteq Y$ with $v \in V$, we have $v < w$.

6.3.2. Example.  is monotone.  is monotone.  is *not* monotone.  is monotone.

6.3.3. Proposition. If $X \in \mathbf{CL}(\mathbf{S}, V)$ where X is monotone, and $X \to Y$, then Y is monotone.

Proof. Let $p +\!\!+ (1)$ be the position of a variable $w \trianglelefteq Y$, and $p +\!\!+ (0) +\!\!+ q$ be the position of a variable $v \trianglelefteq Y$.

Now let r be the position of the redex $\mathbf{S}\,A\,B\,C \trianglelefteq X$ (and the contractum $A\,C\,(B\,C)$ in Y).

If $r \| p$, then reduction did not change anything in $X\,[p]$, so $v < w$ remained intact because X is monotone.

If $p +\!\!+ (1)$ denotes a strict subterm of A, B, or one of the C in the contractum, then the subterm at position p was already present in X, so monotonicity holds.

The position $p +\!\!+ (1)$ cannot denote A or B in the contractum, because these are left subterms of their parent node.

If $p +\!\!+ (1)$ denotes one of the C in the contractum, then we have to verify that $v < w = C$. In X, the redex is $\mathbf{S}\,A\,B\,w$. By monotonicity, for all $v \trianglelefteq A \trianglelefteq \mathbf{S}\,A\,B$ and all $v \trianglelefteq B \trianglelefteq \mathbf{S}\,A\,B$ we know $v < w$. Therefore $A\,w$ and $B\,w$ in the contractum are monotone as well.

Finally, $p \sqsubseteq r$ is impossible because $Y\,[r]$ is a contractum, but $Y\,[p]$ is a variable. $\square$

6.3.4. Proposition. There is no term X in $\mathbf{CL}(\mathbf{S})$ such that $X\,v \hookrightarrow\!\!\!\rightarrow v\,v$.

Proof. The term $X\,v$ is monotone, because X does not contain variables at all. So all its successors must be monotone as well, by proposition 6.3.3. None of them can contain $v\,v$, which is not monotone. $\square$

6.3.5. Remark. If there were such an X, then $X\,X \hookrightarrow\!\!\!\rightarrow X\,X \hookrightarrow\!\!\!\rightarrow \ldots$ would be a ground loop. Now we know that ground loops don't have this simple form. We will soon prove that $\mathbf{CL}(\mathbf{S})$ admits no ground loops at all, using quite a different technique however, because it is not clear how monotonicity could be defined for *ground* terms.

The notion of monotonicity can be applied to $\mathbf{CL}(\mathbf{S}, \mathbf{K})$, because it is respected at least by head reductions.

6.3.6. Proposition. If $X \in \mathbf{CL}(\mathbf{S}, \mathbf{K}, V)$ where X is monotone, and $X \rightarrow_h Y$, then Y is monotone.

Proof. If the redex is $\mathbf{S}\ A\ B\ C$, or the redex position is independent from the variable position, then the reasoning is exactly as in proposition 6.3.3.

If the redex is $\mathbf{K}\ A\ B$, then the contractum is A, which is harmless because it is at head position, and therefore cannot be a variable whose left neighbours had to be checked.

Of course this would be violated for non-head redexes. $\square$

This also proves a conjecture stated by Jan Willem Klop in [Klo80]:

6.3.7. Proposition. There is no $X \in \mathbf{CL}(\mathbf{S}, \mathbf{K})$ with $X\ a \hookrightarrow\!\!\!\rightarrow_h a\ a$.

6.4 Ground Loops

6.4.1. Definition. A *ground loop* is a reduction $X \to^+ C[X]$, where $C[]$ is a context and X is a ground term.

A ground loop, composed with itself, would lead to an infinite reduction containing only finitely many different redexes, so the redex size would be bounded by some constant.

Yet we are going to show that for any fixed n, the restriction of $\mathbf{CL}(\mathbf{S})$ that only allows rewrites where the redex size does not exceed n, is terminating.

Restriction of $\mathbf{CL}(\mathbf{S})$ by labeling

In fact we are not bounding the *size* of the redexes but rather their *right depth* (length of their right spine). Moreover, we don't measure this depth exactly but use a safe approximation instead. This approximation is attached as a label to the application nodes and conservatively updated where reductions take place.

We fix some notation for this.

The labeled system $\mathbf{CL}_n(\mathbf{S})$ has the signature $\{\mathbf{S}, \circ_1, \circ_2, \ldots, \circ_n, \circ_\infty\}$ where $\mathbf{S}$ is as before, while the $\circ_i$ are labeled application nodes. There are $n+1$ of them. The rewrite rules of $\mathbf{CL}_n(\mathbf{S})$ are

$$\left\{ ((\mathbf{S} \circ_* x) \circ_* y) \circ_k z \to (x \circ_k z) \circ_{k+1} (y \circ_k z) \mid k \leq n \right\}$$

where $n+1 = \infty$ and the $*$ match arbitrary numbers.

Next, we prove that $\mathbf{CL}_n(\mathbf{S})$ is terminating.

After that we derive how this carries over to the original $\mathbf{CL}(\mathbf{S})$.

$\mathbf{CL}_n(\mathbf{S})$ is terminating

6.4.2. Theorem. For all n, $\mathbf{CL}_n(\mathbf{S})$ is terminating.

Proof. Order the signature by $\mathbf{S} > \circ_1 > \circ_2 > \ldots > \circ_n > \circ_\infty$.

This gives a recursive path order $>_{\mathrm{rpo}}$ which is a reduction order for $\mathbf{CL}_n(\mathbf{S})$, as can be easily verified:

We need to show that

$$L = ((\mathbf{S} \circ_* x) \circ_* y) \circ_k z >_{\mathrm{rpo}} (x \circ_k z) \circ_{k+1} (y \circ_k z) = R$$

The top symbols of L and R are $\circ_k > \circ_{k+1}$.

We verify $L >_{\mathrm{rpo}} x \circ_k z$ and $L >_{\mathrm{rpo}} y \circ_k z$. Here the top symbols coincide. We have to prove the multiset relation $\left\{ (\mathbf{S} \circ_* x) \circ_* y, z \right\} \gg \{x, z\}$ (and similar for $\{y, z\}$). This is true because $\mathbf{S} \circ_* x \rhd x$. $\square$

Consistency of labeling in $\mathbf{CL}_n(\mathbf{S})$

6.4.3. Definition. The *right depth* of a terms in $\mathbf{CL}(\mathbf{S})$ or $\mathbf{CL}_n(\mathbf{S})$ is

$$d_{\mathrm{r}}(\mathbf{S}) = 0; d_{\mathrm{r}}\left(x \overset{*}{\wedge} y\right) = 1 + d_{\mathrm{r}}(y)$$

We are going to show that the labeling of the application nodes is preserved as a correct approximation (from below) of the right depth of the terms.

6.4.4. Definition. A term X is called *consistent* iff for each subterm $X' \trianglelefteq X$ that has k as its top label, $d_{\mathrm{r}}(X') \geq k$.

Obviously, subterms of consistent terms are consistent.
Consistency is maintained by the reduction rules in $\mathbf{CL}_n(\mathbf{S})$.

6.4.5. Proposition. If X is consistent and $X \to_{\mathbf{CL}_n(\mathbf{S})} Y$, then Y is consistent.

Proof. Consider on reduction step $X \to_{\mathbf{CL}_n(\mathbf{S})} Y$. Assume the redex $((\mathbf{S} \overset{*}{\wedge} B) \overset{*}{\wedge} C) \overset{k}{\wedge} D \trianglelefteq X$ had been contracted.
Subterms of B, C, D just get copied and remain consistent.
Subterms whose position is independent from the redex position are not affected at all.
Subterms lying on the path from the redex position to the root might increase their right depth due to the reduction, but they don't change their top label. So the approximation remains correct.
We consider terms $B \overset{k}{\wedge} D$ and $C \overset{k}{\wedge} D$ in the contractum. We know that the redex $R = ((\mathbf{S} \overset{*}{\wedge} B) \overset{*}{\wedge} C) \overset{k}{\wedge} D$ was consistent, so $k \leq d_{\mathrm{r}}(R) = 1 + d_{\mathrm{r}}(D)$. It is allowed to attach the label k in $B \overset{k}{\wedge} D$ because $d_{\mathrm{r}}\left(B \overset{k}{\wedge} D\right) = 1 + d_{\mathrm{r}}(D) \geq k$. (The same reasoning is valid for $C \overset{k}{\wedge} D$.) So $B \overset{k}{\wedge} D$ and $C \overset{k}{\wedge} D$ are consistent.
Finally $Z = (B \overset{k}{\wedge} D) \overset{k+1}{\wedge} (C \overset{k}{\wedge} D)$ is consistent because $d_{\mathrm{r}}(Z) = 1 + d_{\mathrm{r}}\left(C \overset{k}{\wedge} D\right) \geq 1 + k$ and $C \overset{k}{\wedge} D$ has just been proven consistent. □

Translation between $\mathbf{CL}(\mathbf{S})$ and $\mathbf{CL}_n(\mathbf{S})$

From an unlabeled term in $\mathbf{CL}(\mathbf{S})$, we can create a consistently labeled term in $\mathbf{CL}_n(\mathbf{S})$ (for large enough n) by just tagging each node:

6.4.6. Definition. The mapping $\mathrm{tag}(\cdot) : \mathbf{CL}(\mathbf{S}) \to \mathbf{CL}_n(\mathbf{S})$ is performed by labeling each non-leaf subterm X with $d_{\mathrm{r}}(X)$.

6.4.7. Definition. The mapping $\mathrm{forget}(\cdot) : \mathbf{CL}_n(\mathbf{S}) \to \mathbf{CL}(\mathbf{S})$ just deletes all labels.

Obviously, reductions in $\mathbf{CL}_n(\mathbf{S})$ can be unlabeled.

6.4.8. Proposition. If $X \to_{\mathbf{CL}_n(\mathbf{S})} Y$ for some n, then $\mathrm{forget}(X) \to \mathrm{forget}(Y)$.

Can reductions in $\mathbf{CL}(\mathbf{S})$ be labeled? Yes, at least as long as the right depth of their redexes is bounded.

6.4.9. Proposition. Assume there is (finite or infinite) reduction

$$X_1 \stackrel{R_1}{\to} X_2 \stackrel{R_2}{\to} X_3 \stackrel{R_3}{\to} \dots$$

in $\mathbf{CL}(\mathbf{S})$ where $R_1, R_2, R_3, \dots$ are the redexes being reduced.
Assume that for all k, $d_{\mathrm{r}}(R_k) \leq n$. Then there is a reduction

$$X'_1 \stackrel{R'_1}{\to}_{\mathbf{CL}_n(\mathbf{S})} X'_2 \stackrel{R'_2}{\to}_{\mathbf{CL}_n(\mathbf{S})} X'_3 \stackrel{R'_3}{\to}_{\mathbf{CL}_n(\mathbf{S})} \cdots$$

in $\mathbf{CL}_n(\mathbf{S})$ with $X'_1 = \mathrm{tag}(X_1)$ and for all k, $\mathrm{forget}(X'_k) = X_k$.

Proof. Each redex R'_k is a consistent subterm, by proposition 6.4.5. That's why the top label of R'_k is less or equal to $d_{\mathrm{r}}(R'_k)$ but that is equal to $d_{\mathrm{r}}(R_k) \leq n$. Therefore R'_k really was an $\mathbf{CL}_n(\mathbf{S})$-redex. □

6.4.10. Theorem. Assume there is an infinite reduction

$$X_1 \stackrel{R_1}{\to} X_2 \stackrel{R_2}{\to} X_3 \stackrel{R_3}{\to} \dots$$

in $\mathbf{CL}(\mathbf{S})$ where $R_1, R_2, R_3, \dots$ are the redexes being reduced. Then the sequence $d_{\mathrm{r}}(R_k)$ is not bounded.

Proof. Otherwise, proposition 6.4.9 would apply. Then all reductions could be placed in some $\mathbf{CL}_n(\mathbf{S})$. But this is a terminating system by theorem 6.4.2, so the reduction chain could not be infinite. □

6.4.11. Corollary. $\mathbf{CL}(\mathbf{S})$ admits no ground loops.

Proof. A ground loop would give an infinite reduction chain with only finitely many different redexes, so their right depth would be bounded. This is impossible by theorem 6.4.10 □

Sadly, this method doesn't directly allow to show the absence of non-ground loops because they could produce a sequence of redexes with unbounded right depth.

Chapter 7

Top-termination

7.1 Introduction

If a TRS is non-terminating, then there are infinite reductions. These reductions can be classified.

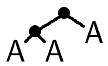
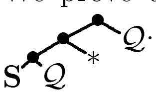
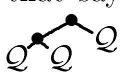
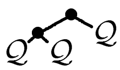
7.1.1. Definition. An reduction sequence is *perpetual* iff it rewrites the root infinitely often.

A desirable property of non-terminating TRS is the absence of perpetual reductions.

7.1.2. Definition. A TRS R is top-terminating iff there is no perpetual reduction.

Trivially, every terminating TRS is top-terminating.
While termination guarantees that redexes eventually vanish, top termination does allow infinite reductions but redex positions move deeper and deeper. This means that for each level of the term, eventually reductions will only change nodes below that level. That's why we can speak of the limit of an infinite reduction.
In this chapter, we will show that $\mathbf{CL}(\mathbf{S})$ indeed is top-terminating. We conjecture that limits of infinite reductions can be finitely described, and these descriptions should allow solving the word problem in $\mathbf{CL}(\mathbf{S})$. But at the moment no proof seems within reach.

Since $\mathbf{CL}(\mathbf{S})$ is a combinatory TRS, it enjoys the standardization property, which loosely says "you can do all head reductions first". We will show that in $\mathbf{CL}(\mathbf{S})$, there are no perpetual *head* reductions. From that it follows that there are no perpetual reductions at all.

For a start, we'll give a very short proof for  not having a head normal form. Indeed no $X \in$  has. Yet this proof does *not* say anything on top termination of the head reduction sequence, so we need a more involved argument.
Our plan is to assume that there were a reduction chain that touched the root infinitely often. We prove theorem 7.3.6 that says that this reduction eventually contains a redex of the form . This goes to  in the next step. Then we prove theorem 7.5.10 telling us that in a head reduction from  on, the root can only be touched finitely often.

7.2 A First Attempt At $\mathcal{Q}\mathcal{Q}\mathcal{Q}$

We begin by giving a different proof for $\mathsf{A}\mathsf{A}\mathsf{A}$ having an infinite head reduction. Note that $\mathsf{A} \in \mathcal{Q}$. We have in fact

7.2.1. Proposition. $\mathcal{Q}\mathcal{Q}\mathcal{Q}$ and $\mathsf{S}\mathcal{Q}*\mathcal{Q}$ have infinite head reduction.

Proof. Consider $X\mathcal{Q}\mathcal{Q}$ with $X \in \mathcal{Q}$.

1. $X \in \mathsf{S}\mathcal{Q}$. Then $X\mathcal{Q}\mathcal{Q} = \mathsf{S}\mathcal{Q}\mathcal{Q}\mathcal{Q} \to \mathcal{Q}\mathcal{Q}(\mathcal{Q}\mathcal{Q}) \subseteq \mathcal{Q}\mathcal{Q}\mathcal{Q}$.
2. X has a head normal form $\mathsf{S}**$. Then $X\mathcal{Q}\mathcal{Q} \twoheadrightarrow_h \mathsf{S}**\mathcal{Q}\mathcal{Q} \to_h *\mathcal{Q}(*\mathcal{Q})\mathcal{Q} \subseteq \mathcal{Q}\mathcal{Q}\mathcal{Q}$.
3. X has no head normal form. Then $X\mathcal{Q}\mathcal{Q}$ has an infinite head reduction as well, and the arguments of X are never touched.

In each case except the last one, we do one head reduction step that leads to an expression that is in $\mathcal{Q}\mathcal{Q}\mathcal{Q}$ again. This proves that $\mathcal{Q}\mathcal{Q}\mathcal{Q}$ has an infinite head reduction.
On the other hand, $\mathsf{S}\mathcal{Q}*\mathcal{Q} \to_h \mathcal{Q}\mathcal{Q}(*\mathcal{Q}) \subseteq \mathcal{Q}\mathcal{Q}\mathcal{Q}$. $\square$

Note that this proof does not say much about where the head redexes really are—compared to the explicit derivation of $\mathsf{A}\mathsf{A}\mathsf{A}$ given in example 5.5.2 which explicitly shows that the left spine grows longer and longer. Thus we really need some different ideas to show top-termination of $\mathbf{CL}(\mathbf{S})$.

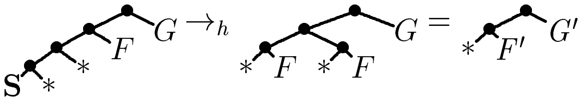
7.3 Reaching $\mathcal{Q}\mathcal{Q}\mathcal{Q}$

7.3.1. Proposition. If $\mathsf{S}BCD \to_h^+ *FG$, then $D \trianglelefteq F$ and $CD \trianglelefteq G$.

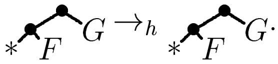
Proof. The reduction starts $\mathsf{S}BCD \to_h BD(CD)$. We are going to show by induction on the length of the reduction chain that the required property remains invariant from now on. In each reduction step, the head redex could be at these positions:

1. at the top: $\mathsf{S}*FG \to_h *G(FG) = *F'G'$

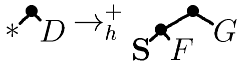
   We have $D \triangleleft CD \trianglelefteq G = F'$ and $CD \trianglelefteq G \triangleleft G'$.

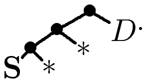
2. one step down the left spine: 

   We have $D \trianglelefteq F \triangleleft F'$ and  $\trianglelefteq G = G'$.

3. deeper on the left spine: . This does not change F and G.

$\square$

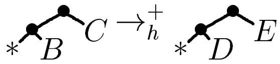
7.3.2. Proposition. If , then $D \trianglelefteq F$ and $D \triangleleft G$.

Proof. Consider the first of these reductions that touches D. This reduction must exist. Its redex has the form . To this, apply proposition 7.3.1 $\square$

Assume now that there were a head reduction including infinitely many top redexes. This could be written as

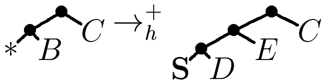
$$V_{(1,1)} \twoheadrightarrow_h V_{(1,n_1)} \rightarrow_h U_{(1,1)} \twoheadrightarrow_h U_{(1,m_1)} \rightarrow_h$$
$$V_{(2,1)} \twoheadrightarrow_h V_{(2,n_1)} \rightarrow_h U_{(2,1)} \twoheadrightarrow_h \dots$$

where each of the $V_{(i,k)}$ is a top redex, and none of the $U_{(i,k)}$ is a redex.
In such a chain, each block of consecutive top redexes $V_{(i,*)}$ is finite. By assumption, there should still be infinitely many top redexes, so each of the blocks of consecutive non-top redexes $U_{(i,*)}$ must be finite as well.
We now want to compare $V_{(i,1)}$ to $V_{(i+1,1)}$, using $U_{(i,1)}$ as intermediate step.

7.3.3. Proposition. For all k, we have $V_{(k,1)} =$  $= U_{(k+1,1)}$

with $C \trianglelefteq D$ and  $\trianglelefteq E$.

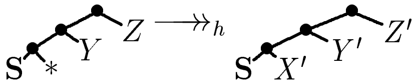
Proof. By proposition 7.3.1. $\square$

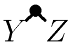
7.3.4. Proposition. For all k, we have $U_{(k,1)} =$  $= V_{(k+1,1)}$

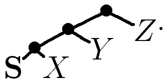
with $B \trianglelefteq D$ and $B \trianglelefteq E$.

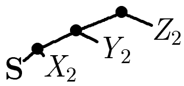
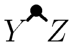
Proof. None of the reductions take place at the top, so C is not touched and we can apply proposition 7.3.2 to the left subterm of the root. $\square$

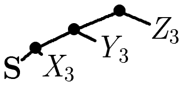
The last two propositions can be merged into

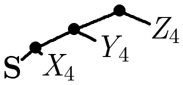
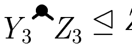
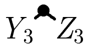
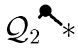
7.3.5. Proposition. For all k, we have $V_{(k,1)} =$  $= V_{(k+1,1)}$

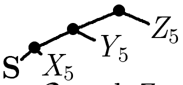
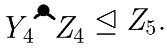
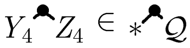
with $Z \trianglelefteq X'$, $Z \triangleleft Y'$, and  $\trianglelefteq Z'$.

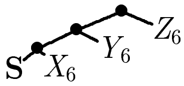
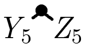
Now we apply this proposition a few times, starting with some arbitrary $V_{(1,1)} =$ .

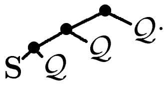
We obtain $V_{(2,1)} =$  with  $\trianglelefteq Z_2$. Therefore Z_2 cannot be $\mathbf{S}$, so it must be in $\mathcal{Q}_1$.

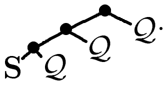
For $V_{(3,1)} =$  we have $Z_2 \lhd Y_3$. Since $Z_2 \in \mathcal{Q}_1$ this means $Y_3 \in \mathcal{Q}_2$.

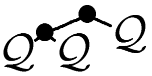
For $V_{(4,1)} =$  we know  $\unlhd Z_4$. But  $\in$  $\subseteq \mathcal{Q}$, leading to $Z_4 \in \mathcal{Q}$.

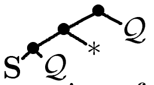
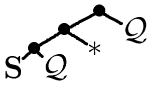
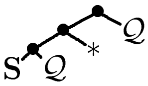
For $V_{(5,1)} =$  again $Z_4 \lhd Y_5$ and  $\unlhd Z_5$. Since $Z_4 \in \mathcal{Q}$ and  $\subseteq \mathcal{Q}$ we know $Y_5 \in \mathcal{Q}$ and $Z_5 \in \mathcal{Q}$.

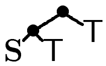
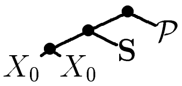
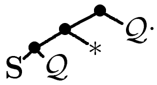
Finally, $V_{(6,1)} =$  has $Y_5 \unlhd X_6$ as well as $Z_5 \lhd Y_6$ and  $\unlhd Z_6$.

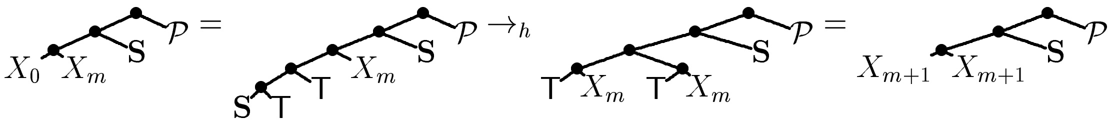
This says that $V_{(6,1)} \in$ . So we proved

7.3.6. Theorem. In a head reduction that has infinitely many steps at the top, eventually there is a top redex .

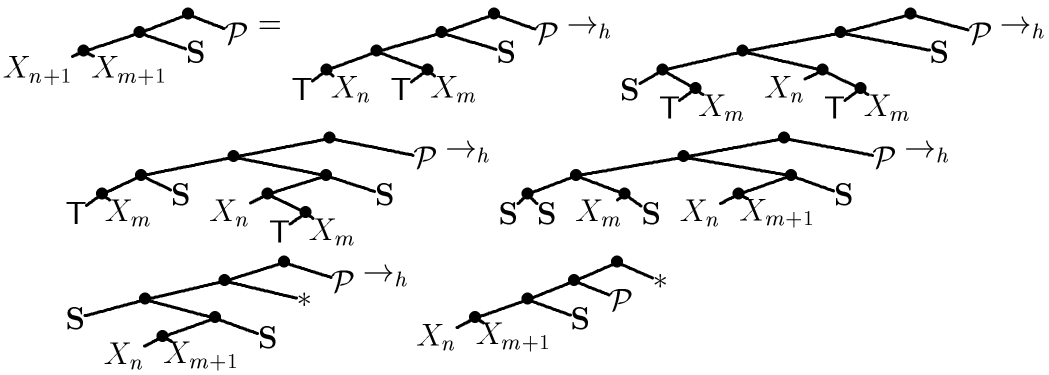
7.4 Avoiding 

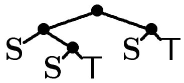
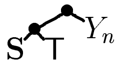
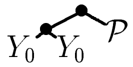
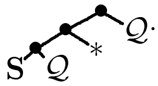
The previous arguments might have created the impression that all infinite head reductions eventually contain some  redex. But the existence of the  redex had only been shown under the assumption of a head reduction that has enough top steps. There are reductions that avoid the  pattern completely by just descending to the left soon enough.

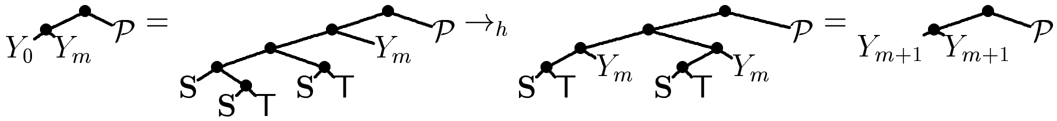
7.4.1. Example. $X_0 =$ ; $X_{n+1} =$ . Then  has an infinite head reduction where no redex has form . The reduction consists of the upward step



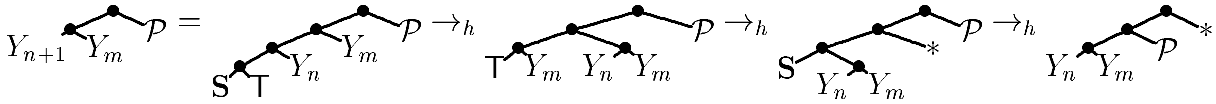
and the downward step



7.4.2. Example. $Y_0 =$ ; $Y_{n+1} =$ . Then  has an infinite head reduction where no redex has form . Here the upward step is



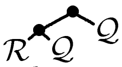
while the downward step goes



Yet these exceptions don't hurt because nevertheless the reductions are top-terminating.

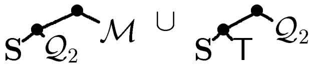
7.5 Inside 

We are going to show that each head reduction starting from  touches the root only finitely often.

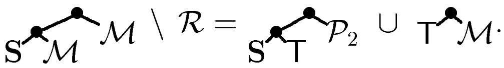
In order to do so, we present a set $\mathcal{R} \subseteq \mathcal{Q}$ such that  has the desired property immediately, i. e. no further knowledge (than being in $\mathcal{Q}$) about the two arguments of $\mathcal{R}$ is required.

Finally we deal with the complementary cases . Here we need to iterate the reasoning as some of the $\mathcal{Q}$ arguments might now get in front position. But while doing so, they also grow and therefore the process cannot be repeated infinitely often.

We give the definition now

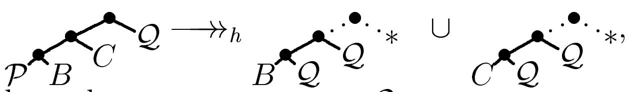
7.5.1. Definition. $\mathcal{R} =$ 

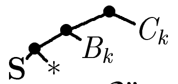
and justify it after preparing some tools. As an immediate consequence we obtain

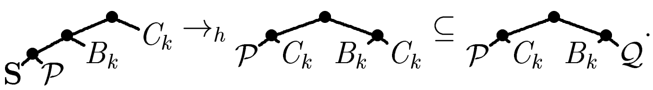
7.5.2. Definition. $\overline{\mathcal{R}} =$ 

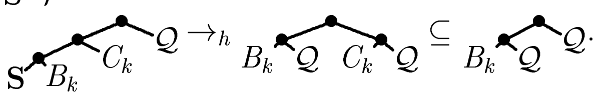
This can be checked with the program `RX` (see Appendix).

7.5.3. Proposition. If $B \in \mathcal{Q}$ and $C \in \mathcal{Q}$, then there is a reduction

, i. e. the result starts with B or C and is followed by at least two arguments $\mathcal{Q}$.

Proof. We show that in all redexes  that occur during this reduction, both B_k and C_k have the form "B or C, followed by some $\mathcal{Q}$".

This is invariant because .

We stop the reduction at . □

7.5.4. Proposition. $((\mathcal{Q}_2\,\mathcal{Q})\,\mathcal{Q})\,\mathcal{Q}$ has infinite head reduction where the root is rewritten only finitely often.

Proof. We have $\mathcal{Q}_2 = \mathcal{Q} \cup \mathsf{S}(\mathsf{S}\,\mathcal{P})$.

In the first case, in the reduction of $((\mathcal{Q}\,\mathcal{Q})\,\mathcal{Q})\,\mathcal{Q}$ the root is never rewritten because $(\mathcal{Q}\,\mathcal{Q})\,\mathcal{Q}$ alone has no head normal form.

Otherwise, $(((\mathsf{S}(\mathsf{S}\,\mathcal{P}))\,\mathcal{Q})\,\mathcal{Q})\,\mathcal{Q} \to_h (((\mathsf{S}\,\mathcal{P})\,\mathcal{Q})(\mathcal{Q}\,\mathcal{Q}))\,\mathcal{Q} \to_h ((\mathcal{P}(\mathcal{Q}\,\mathcal{Q}))(\mathcal{Q}(\mathcal{Q}\,\mathcal{Q})))\,\mathcal{Q} \subseteq ((\mathcal{P}(\mathcal{Q}\,\mathcal{Q}))(\mathcal{Q}\,\mathcal{Q}))\,\mathcal{Q}$.

By proposition 7.5.3, this goes to $((\mathcal{Q}\,\mathcal{Q})\,\mathcal{Q})\,\mathcal{Q}$ in each case, and so we are done. □

Now we look at $(X\,\mathcal{Q})\,\mathcal{Q}$ with $X \in \mathcal{Q}$.

If $X = \mathsf{S}\,X'$, then $(X\,\mathcal{Q})\,\mathcal{Q} = ((\mathsf{S}\,X')\,\mathcal{Q})\,\mathcal{Q} \to_h (X'\,\mathcal{Q})(\mathcal{Q}\,\mathcal{Q}) \subseteq (X'\,\mathcal{Q})\,\mathcal{Q}$ and we consider X' instead (this transition can happen only finitely often).

So assume $X \in (\mathcal{M}\,\mathcal{M})\,\mathcal{M}$. If it has no head normal form, then head reductions in $(X\,\mathcal{Q})\,\mathcal{Q}$ never touch the root of the term.

Therefore let X have head normal form $H \in (\mathsf{S}\,*)\,*$.

If $H \in \mathcal{R}$, then we will apply proposition 7.5.5 which comes next. Otherwise $H \in \overline{\mathcal{R}}$ and we will use proposition 7.5.9 which is stated and proved later.

7.5.5. Proposition. $(\mathcal{R}\,\mathcal{Q})\,\mathcal{Q}$ has infinite head reduction where the root is rewritten only finitely often.

Proof. Recall $\mathcal{R} = (\mathsf{S}\,\mathsf{T})\,\mathcal{Q}_2 \cup (\mathsf{S}\,\mathcal{Q}_2)\,\mathcal{M}$.

1. $(((\mathsf{S}\,\mathsf{T})\,\mathcal{Q}_2)\,\mathcal{Q})\,\mathcal{Q} \to_h ((\mathsf{T}\,\mathcal{Q})(\mathcal{Q}_2\,\mathcal{Q}))\,\mathcal{Q} \to_h ((\mathsf{S}(\mathcal{Q}_2\,\mathcal{Q}))\,*)\,\mathcal{Q} \to_h ((\mathcal{Q}_2\,\mathcal{Q})\,\mathcal{Q})(*\,\mathcal{Q}) \subseteq ((\mathcal{Q}_2\,\mathcal{Q})\,\mathcal{Q})\,\mathcal{Q}$

2. $(((\mathsf{S}\,\mathcal{Q}_2)\,*)\,\mathcal{Q})\,\mathcal{Q} \to_h ((\mathcal{Q}_2\,\mathcal{Q})(*\,\mathcal{Q}))\,\mathcal{Q} \subseteq ((\mathcal{Q}_2\,\mathcal{Q})\,\mathcal{Q})\,\mathcal{Q}$

and we apply proposition 7.5.4 in both cases. □

To deal with the case $(\overline{\mathcal{R}}\,\mathcal{Q})\,\mathcal{Q}$, we need more tools.

7.5.6. Proposition. If $Y \in \mathcal{Q}$ and $Z \in \mathcal{Q}$, then $((\overline{\mathcal{R}}\,Y)\,Z) \twoheadrightarrow_h ((Y\,\mathcal{Q})\,\mathcal{Q}) \cup ((Z\,\mathcal{Q})\,\mathcal{Q})$.

Proof. Recall $\overline{\mathcal{R}} = ((\mathsf{S}\,\mathsf{T})\,\mathcal{P}_2) \cup (\mathsf{T}\,\mathcal{M})$.

1. $X = ((\mathsf{S}\,\mathsf{T})\,\mathsf{S})$. Then $((X\,Y)\,Z) = ((((\mathsf{S}\,\mathsf{T})\,\mathsf{S})\,Y)\,Z) \to_h (((\mathsf{T}\,Y)\,(\mathsf{S}\,Y))\,Z) \to_h (((\mathsf{S}\,(\mathsf{S}\,Y))\,(Y\,(\mathsf{S}\,Y)))\,Z)$ $\to_h (((\mathsf{S}\,Y)\,Z)\,((Y\,(\mathsf{S}\,Y))\,Z)) \to_h ((Y\,((Y\,(\mathsf{S}\,Y))\,Z))\,(Z\,((Y\,(\mathsf{S}\,Y))\,Z)))$.
2. $X = ((\mathsf{S}\,\mathsf{T})\,\mathsf{T})$. Then $((X\,Y)\,Z) = ((((\mathsf{S}\,\mathsf{T})\,\mathsf{T})\,Y)\,Z) \to_h (((\mathsf{T}\,Y)\,(\mathsf{T}\,Y))\,Z) \to_h (((\mathsf{S}\,(\mathsf{T}\,Y))\,(Y\,(\mathsf{T}\,Y)))\,Z)$ $\to_h (((\mathsf{T}\,Y)\,Z)\,((Y\,(\mathsf{T}\,Y))\,Z)) \to_h (((\mathsf{S}\,Z)\,(Y\,Z))\,((Y\,(\mathsf{T}\,Y))\,Z)) \to_h ((Z\,((Y\,(\mathsf{T}\,Y))\,Z))\,((Y\,Z)\,((Y\,(\mathsf{T}\,Y))\,Z)))$.
3. $X \in (\mathsf{T}\,*)$. Then $((X\,Y)\,Z) = (((\mathsf{T}\,*)\,Y)\,Z) \to_h (((\mathsf{S}\,Y)\,(*\,Y))\,Z) \to_h ((Y\,Z)\,((*\,Y)\,Z))$.

$\square$

7.5.7. Proposition. $((\mathcal{Q}\,(\mathcal{Q}\,\mathcal{Q}))\,(\mathcal{Q}\,\mathcal{Q}))$ has infinite head reduction where the root is rewritten only finitely often.

Proof. Let the expression be $((X\,(\mathcal{Q}\,\mathcal{Q}))\,(\mathcal{Q}\,\mathcal{Q}))$.

If $X = (\mathsf{S}\,X')$, then $((X\,(\mathcal{Q}\,\mathcal{Q}))\,(\mathcal{Q}\,\mathcal{Q})) = (((\mathsf{S}\,X')\,(\mathcal{Q}\,\mathcal{Q}))\,(\mathcal{Q}\,\mathcal{Q}))$
$\to_h ((X'\,(\mathcal{Q}\,\mathcal{Q}))\,((\mathcal{Q}\,\mathcal{Q})\,(\mathcal{Q}\,\mathcal{Q}))) \subseteq ((X'\,(\mathcal{Q}\,\mathcal{Q}))\,(\mathcal{Q}\,\mathcal{Q}))$ and repeat the argument for X'. This transition can only happen finitely often.

Now $X \in ((\mathcal{M}\,\mathcal{M})\,\mathcal{M})$. If this has no head normal form, then we're done. Assume it has head normal form $X' \in ((\mathsf{S}\,*)\,*)$.

If $X' \in \mathcal{R}$ then the claim is established by proposition 7.5.5.

If $X' \in \overline{\mathcal{R}}$ then by proposition 7.5.6 $(X'\,((\mathcal{Q}\,\mathcal{Q})\,(\mathcal{Q}\,\mathcal{Q}))) \twoheadrightarrow_h (((\mathcal{Q}\,\mathcal{Q})\,\mathcal{Q})\,\mathcal{Q})$ and again the root cannot be rewritten any more. $\square$

7.5.8. Proposition. If $Y \in \mathcal{Q}$ and $Z \in \mathcal{Q}$,
then $((((\mathsf{S}\,\mathsf{T})\,\mathcal{P}_2)\,Y)\,Z) \twoheadrightarrow_h ((Y\,(\mathcal{Q}\,\mathcal{Q}))\,(\mathcal{Q}\,\mathcal{Q})) \cup ((Z\,(\mathcal{Q}\,\mathcal{Q}))\,(\mathcal{Q}\,\mathcal{Q}))$.

Proof. Follows from the proof of proposition 7.5.6. $\square$

7.5.9. Proposition. $\overline{\mathcal{R}}\mathcal{Q}\mathcal{Q}$ has infinite head reduction where the root is rewritten only finitely often.

Proof. Let $XYZ \in \overline{\mathcal{R}}\mathcal{Q}\mathcal{Q}$. If $X \in \mathbf{S}\mathsf{T}\mathcal{P}_2$, then $\mathbf{S}\mathsf{T}\mathcal{P}_2\mathcal{Q}\mathcal{Q} \twoheadrightarrow_h \mathcal{Q}(\mathcal{Q}\mathcal{Q})(\mathcal{Q}\mathcal{Q})$ by proposition 7.5.8 and this has the required property by proposition 7.5.7.
Therefore assume $X \in \mathsf{T}*$.
Then $XY\mathcal{Q} = \mathsf{T}*Y\mathcal{Q} \to_h \mathbf{S}Y(*Y)\mathcal{Q} \to_h Y\mathcal{Q}(*Y\mathcal{Q}) \subseteq Y\mathcal{Q}(\mathcal{Q}\mathcal{Q})$.
If $Y = \mathbf{S}Y'$, then $Y\mathcal{Q}(\mathcal{Q}\mathcal{Q}) = \mathbf{S}Y'\mathcal{Q}(\mathcal{Q}\mathcal{Q}) \to_h Y'(\mathcal{Q}\mathcal{Q})(\mathcal{Q}(\mathcal{Q}\mathcal{Q})) \subseteq \mathcal{Q}(\mathcal{Q}\mathcal{Q})(\mathcal{Q}\mathcal{Q})$ and we apply proposition 7.5.7.
So let $Y \in \mathcal{M}\mathcal{M}\mathcal{M}$. If this has no head normal form, then we are done. Assume it has a head normal form $Y' \in \mathbf{S}*$.
If $Y' \in \mathcal{R}$ then we use proposition 7.5.5. Therefore let $Y' \in \overline{\mathcal{R}}$.
If $Y' \in \mathbf{S}\mathsf{T}\mathcal{P}_2$, then by proposition 7.5.8, $Y'\mathcal{Q}(\mathcal{Q}\mathcal{Q}) \subseteq Y'\mathcal{Q}\mathcal{Q} \twoheadrightarrow_h \mathcal{Q}(\mathcal{Q}\mathcal{Q})(\mathcal{Q}\mathcal{Q})$ and we now use proposition 7.5.7.
Thus, assume $Y' \in \mathsf{T}*$. We have $Y'\mathcal{Q}(\mathcal{Q}\mathcal{Q}) = \mathsf{T}*\mathcal{Q}(\mathcal{Q}\mathcal{Q}) \to_h \mathbf{S}\mathcal{Q}(*\mathcal{Q})(\mathcal{Q}\mathcal{Q})$
$\to_h \mathcal{Q}(\mathcal{Q}\mathcal{Q})(*\mathcal{Q}(\mathcal{Q}\mathcal{Q})) \subseteq \mathcal{Q}(\mathcal{Q}\mathcal{Q})(\mathcal{Q}\mathcal{Q})$. Then we are done, using proposition 7.5.7. $\square$

So we proved

7.5.10. Theorem. $\mathcal{Q}\mathcal{Q}\mathcal{Q}$ has infinite head reduction where the root is rewritten only finitely often.

In light of theorem 7.3.6, this completes the proof of top-termination for $\mathbf{CL}(\mathbf{S})$.

7.6 Infinite normal forms

(This section is just an aside; it will not be used in the sequel. We use concepts and results from infinitary rewriting, as described in [KKSdV95], without repeating their definitions here.)

Top-termination guarantees that infinite reductions are *strongly convergent.*

(That's good. We don't have to reason about Cauchy convergent reductions as defined by Dershowitz in [DKP91].)

Since $\mathbf{CL}(\mathbf{S})$ is orthogonal and non-collapsing, the infinitary Church-Rosser property holds, and we can take the limit of a fair reduction (that does not disregard some redex completely) as the unique (possibly infinite) normal form of an $\mathbf{S}$-term.

We conjecture that such limits of $\mathbf{S}$-terms can always be described by patterns similar to that given in the examples 5.5.1 and 5.5.2 above.

Moreover, they seem to have enough leaves at finite depths so that the word problem in $\mathbf{CL}(\mathbf{S})$ might be decidable by the comparison of these patterns.

7.6.1. Example. We return to $\mathsf{S}\ \mathsf{T}\ \mathsf{T}\ (\mathsf{S}\ \mathsf{T}\ \mathsf{T})$ (example 5.5.1) with its X_k patterns.

$$f\,(m,n) = \lim\, (X_m\, X_n)$$
$$f\,(m+1,n) = \lim\, ((\mathsf{T}\, X_m)\, X_n) = \lim\, ((\mathsf{S}\, X_n)\,(X_m\, X_n)) = (\mathsf{S}\, X_n)\,\boxed{\lim\, (X_m\, X_n)} = (\mathsf{S}\, X_n)\, f\,(m,n)$$
$$f\,(0,n) = \lim\, (((\mathsf{S}\,\mathsf{T})\,\mathsf{T})\, X_n) = \lim\, ((\mathsf{T}\, X_n)\,(\mathsf{T}\, X_n)) = f\,(n+1,n+1)$$

If we drop all subscripts and arguments, then we obtain a grammar for a set of infinite trees.

$$X \to ((\mathsf{S}\,\mathsf{T})\,\mathsf{T}) \cup (\mathsf{T}\, X);\ f \to ((\mathsf{S}\, X)\, f)$$

This grammar, possibly with some decoration, should be sufficient to characterize $\mathsf{S}\ \mathsf{T}\ \mathsf{T}\ (\mathsf{S}\ \mathsf{T}\ \mathsf{T})$.

The limits so obtained don't contain redexes. Still they might contain subterms with infinite left spines. Such subterms *are* in head normal form, where *head normal form* is defined as a term that cannot be reduced to a redex.

On the other hand, terms M where $\lim M$ has an infinite left spine behave like *unsolvable* terms because $\lim\, (M\, N) = \lim\, ((\lim M)\,(\lim N))$. This property will be used in the next example.

7.6.2. Example. Now we want to describe $\lim\,(\mathsf{A}\ \mathsf{A}\ \mathsf{A})$ (example 5.5.2).

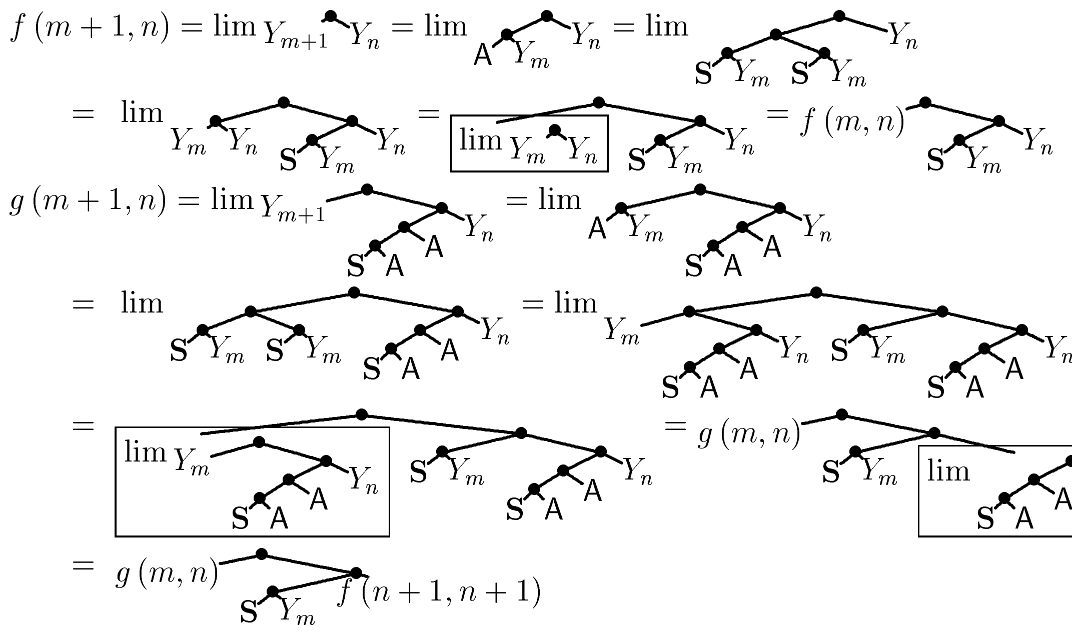
Because of $((\mathsf{A}\,\mathsf{A})\,\mathsf{A}) \twoheadrightarrow_h (Y_1\, Y_1)\,((\mathsf{S}\,((\mathsf{S}\,\mathsf{A})\,\mathsf{A}))\, Y_0)$, this amounts to determining $f\,(1,1)$ where

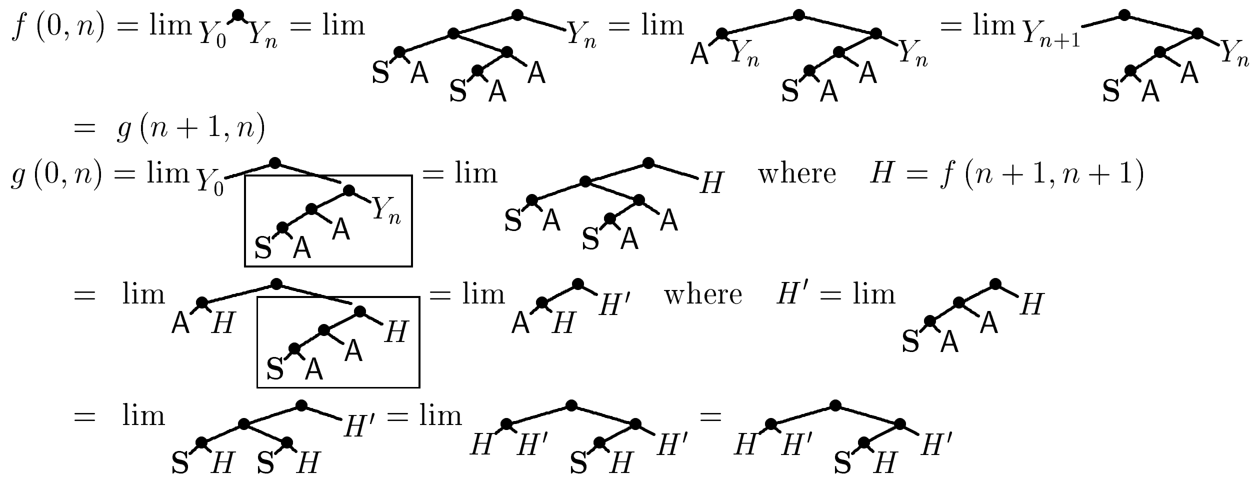
7.6.3. Definition. $f\,(m,n) = \lim\, (Y_m\, Y_n)$ and the auxiliary $g\,(m,n) = \lim\, (Y_m\,(((\mathsf{S}\,\mathsf{A})\,\mathsf{A})\, Y_n))$

As $(Y_m\, Y_n) \in ((Q\, Q)\, Q)$ and $(Y_m\,(((\mathsf{S}\,\mathsf{A})\,\mathsf{A})\, Y_n)) \in ((Q\, Q)\, Q)$, both have an infinite head reduction, as has just been shown. Therefore both $f\,(m,n)$ and $g\,(m,n)$ have an infinite left spine. The following is immediate

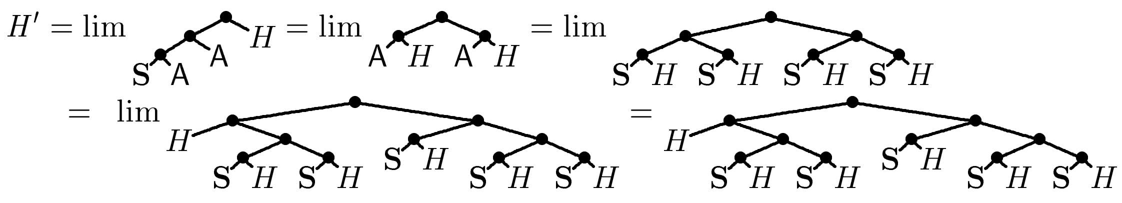
7.6.4. Lemma. $\lim\, (((\mathsf{S}\,\mathsf{A})\,\mathsf{A})\, Y_n) = \lim\, ((\mathsf{A}\, Y_n)\,(\mathsf{A}\, Y_n)) = \lim\, (Y_{n+1}\, Y_{n+1}) = f\,(n+1,n+1)$

The reductions already given can be translated to





The names H and H' are used only locally, for convenience of notation. We still need to determine H', using the fact that H has an infinite left spine.



Again, dropping all subscripts and arguments in Y_*, $f\,(*,*)$, $g\,(*,*)$, H, H' results in a grammar for infinite trees.

Chapter 8

Decidability of Normalization

8.1 Introduction

To decide whether an arbitrary **S**-term from $\mathcal{M}$ normalizes, it is enough to have a procedure that decides this for the case $\mathcal{N}\mathcal{N}$, i.e. it may assume that both children of the root already are in normal form. If they were not, we could first apply the procedure to these subterms recursively and compute their normal forms if the procedure says they exist.

Since $\mathbf{CL}(\mathbf{S})$ is orthogonal and non-erasing, it enjoys the conservation theorem which says that if there is some infinite reduction from X, then all reductions from X are infinite. So it is sufficient to exhibit one infinite reduction in order to show that X has no normal form.

We start by giving three classes of expressions with infinite reductions.

Continuing, we will construct subsets of $\mathcal{N}\mathcal{N}$ that contain terms with similar reduction properties. By some sort of magic, these subsets cover all of $\mathcal{N}\mathcal{N}$.

Actually a small range of terms in $\mathcal{N}\mathcal{N}$ remains uncovered but we will show that after some reductions these terms move into the colonized areas.

The construction begins by partitioning the set $\mathcal{N}$ itself into classes $\mathcal{N}_0$, $\mathcal{N}_1$ and $\mathcal{N}_2$, and then combines pairs of these. It turns out that $\mathcal{N}_0$ is responsible for finite reductions, and $\mathcal{N}_2$ for infinite ones.

During the process, the case $\mathcal{N}_1\mathcal{N}_1$ needs special attention. Here, a further partition of $\mathcal{N}_1$ into the sets $\mathcal{L}_0$, $\mathcal{L}_1$ and $\mathcal{L}_2$ is called for. By considering all possible combinations of these three sets, the investigation is completed.

Most of these constructions have been found empirically, by staring at thousands of examples. So the presentation here necessarily generates some ad-hoc feeling. A straight top-down approach might be more desirable, but that seems to transcend today's knowledge of $\mathbf{CL}(\mathbf{S})$.

8.2 Infinite Reductions

We now give three classes of expressions not having a normal form.

They are all of the shape $\langle * \rangle *$.

We start by generalizing the reduction shown in example 5.5.1. We note that each redex has the form $\langle \mathsf{S}\mathsf{T}\mathsf{T}\rangle\langle \mathsf{S}\mathsf{T}\mathsf{T}\rangle$ and that $\mathsf{S}\mathsf{T}\mathsf{T}$ is the smallest term in the set $\mathcal{Q}_2\mathcal{Q}_1$.

8.2.1. Definition. $F = \left\langle \mathcal{Q}_2 \mathcal{Q}_1 \right\rangle$

We are going to show that $F F \twoheadrightarrow \infty$. We need

8.2.2. Proposition. $\mathcal{Q}_1 F \subseteq F$

Proof. We use the fact that $\mathcal{Q}_1 = \mathsf{T} \cup \mathcal{Q}_2$. Let $X \in \mathcal{Q}_1; Y \in F$.

1. $X = \mathsf{T}$. Then $X Y \in \mathbf{S} * F \subseteq \langle F \rangle \subseteq F$.
2. $X \in \mathcal{Q}_2$. As $Y \in F \subseteq \mathcal{Q}_1$, we have $X Y \in \mathcal{Q}_2 \mathcal{Q}_1 \subseteq F$.

□

This will now be used in the proof of

8.2.3. Proposition. $\left\langle \mathcal{Q}_2 \mathcal{Q}_1 \right\rangle \left\langle \mathcal{Q}_2 \mathcal{Q}_1 \right\rangle \twoheadrightarrow \infty$

Proof. We have $\left\langle \mathcal{Q}_2 \mathcal{Q}_1 \right\rangle \left\langle \mathcal{Q}_2 \mathcal{Q}_1 \right\rangle \hookrightarrow\!\!\!\twoheadrightarrow \mathcal{Q}_2 \mathcal{Q}_1 F \ni X Y Z$.
If $X \twoheadrightarrow \infty$, then the $X Y Z \twoheadrightarrow \infty$. So assume X has a normal form.
It could have shape $\mathbf{S} *$ or $\mathbf{S} * *$.

1. $X \twoheadrightarrow \mathbf{S} \mathcal{Q}_1$. $\mathbf{S} \mathcal{Q}_1 Y F \to \mathcal{Q}_1 F (Y F) \subseteq \mathcal{Q}_1 F (\mathcal{Q}_1 F) \subseteq F F \twoheadrightarrow \infty$
2. $X \twoheadrightarrow \mathbf{S} * *$. $\mathbf{S} * * \mathcal{Q}_1 F \to * \mathcal{Q}_1 (* \mathcal{Q}_1) F$.

   By corollaries 5.2.8 and 5.2.7 we have $* \mathcal{Q}_1 \subseteq \mathcal{Q}_2 \subseteq \mathcal{Q}_1$,
   so $* \mathcal{Q}_1 (* \mathcal{Q}_1) F \subseteq \mathcal{Q}_2 \mathcal{Q}_1 F \subseteq F F \twoheadrightarrow \infty$.

In either case we find an infinite reduction because we've done one reduction step whose result again has the form $F F$.

□

The following pattern turns out to be a generalization of $\mathsf{A} \mathsf{A} \mathsf{A} \twoheadrightarrow \infty$. We recall that $\mathsf{A} \in \mathcal{Q}_k$ for any k.

8.2.4. Proposition. $\left\langle \mathcal{Q}_3 \mathcal{Q}_2 \right\rangle \mathcal{Q}_1 \twoheadrightarrow \infty$

Proof. Let $\left\langle \mathcal{Q}_3 \mathcal{Q}_2 \right\rangle \mathcal{Q}_1 \hookrightarrow\!\!\!\twoheadrightarrow \mathcal{Q}_3 \mathcal{Q}_2 \mathcal{Q}_1 \ni X Y Z$.
If $X \twoheadrightarrow \infty$, then the $X Y Z \twoheadrightarrow \infty$. So assume X has a normal form. It could be $\mathbf{S} \mathcal{Q}_2$ or $\mathbf{S} * *$.

1. $X \twoheadrightarrow \mathsf{S}\,\mathcal{Q}_2$. $((\mathsf{S}\,\mathcal{Q}_2)\,\mathcal{Q}_2)\,\mathcal{Q}_1 \to (\mathcal{Q}_2\,\mathcal{Q}_1)\,(\mathcal{Q}_2\,\mathcal{Q}_1) \subseteq F\,F \twoheadrightarrow \infty$ by proposition 8.2.3

2. $X \twoheadrightarrow (\mathsf{S}\,*)\,*$. $(((\mathsf{S}\,*)\,*)\,\mathcal{Q}_2)\,\mathcal{Q}_1 \to ((*\,\mathcal{Q}_2)\,(*\,\mathcal{Q}_2))\,\mathcal{Q}_1$ By corollaries 5.2.8 and 5.2.7 we have $*\,\mathcal{Q}_2 \subseteq \mathcal{Q}_3 \subseteq \mathcal{Q}_2$, so $((*\,\mathcal{Q}_2)\,(*\,\mathcal{Q}_2))\,\mathcal{Q}_1 \subseteq (\mathcal{Q}_3\,\mathcal{Q}_2)\,\mathcal{Q}_1 \twoheadrightarrow \infty$ by recursive application of proposition 8.2.4.

$\square$

Finally we mention a variant of the $(\mathcal{Q}_3\,\mathcal{Q}_2)\,\mathcal{Q}_1$ pattern:

8.2.5. Proposition. $\left\langle (\mathsf{S}\,\mathcal{Q}_3)\,\mathcal{Q}_0 \right\rangle \mathcal{Q}_2 \twoheadrightarrow \infty$

Proof. $\left\langle (\mathsf{S}\,\mathcal{Q}_3)\,\mathcal{Q}_0 \right\rangle \mathcal{Q}_2 \lhook\joinrel\twoheadrightarrow ((\mathsf{S}\,\mathcal{Q}_3)\,\mathcal{Q}_0)\,\mathcal{Q}_2 \to (\mathcal{Q}_3\,\mathcal{Q}_2)\,(\mathcal{Q}_0\,\mathcal{Q}_2) \in (\mathcal{Q}_3\,\mathcal{Q}_2)\,\mathcal{Q}_1$. Then apply proposition 8.2.4.

$\square$

8.2.6. Remark. We have shown

$$\left\langle \mathcal{Q}_2\,\mathcal{Q}_1 \right\rangle \left\langle \mathcal{Q}_2\,\mathcal{Q}_1 \right\rangle \twoheadrightarrow \infty;\quad \left\langle \mathcal{Q}_3\,\mathcal{Q}_2 \right\rangle \mathcal{Q}_1 \twoheadrightarrow \infty;\quad \left\langle (\mathsf{S}\,\mathcal{Q}_3)\,\mathcal{Q}_0 \right\rangle \mathcal{Q}_2 \twoheadrightarrow \infty$$

Is it a coincidence that the sum of the indices of these expression is "nearly" always 6? There should be a better way to describe these patterns.

8.3 The case $\mathcal{N}\,\mathcal{N}$

We will classify the set $\mathcal{N}$ of normal forms according the existence of certain subterms that might or might not occur during the reduction of $X\,a$ where $X \in \mathcal{N}$ and a is a variable.
The most restricted class is

8.3.1. Definition. $\mathcal{N}_0 = \left\langle \mathsf{T} \middle/ \mathsf{S} \cup \mathsf{S}\,\mathcal{N} \right\rangle$

8.3.2. Example. $\mathsf{A} = (\mathsf{S}\,\mathsf{S})\,\mathsf{S} = \mathsf{T}\,\mathsf{S} \in \langle \mathsf{T} / \mathsf{S} \rangle \subseteq \mathcal{N}_0$.

A term from $\mathcal{N}_0$ never moves its argument into functional position:

8.3.3. Example. $\mathsf{A}\,a = ((\mathsf{S}\,\mathsf{S})\,\mathsf{S})\,a \to (\mathsf{S}\,a)\,(\mathsf{S}\,a)$

8.3.4. Proposition. $\mathcal{N}_0\,a$ has a normal form, and $\mathcal{N}_0\,a \not\twoheadrightarrow a\,*$

Proof. $\mathcal{N}_0\,a = \left\langle \mathsf{T} \middle/ \mathsf{S} \cup \mathsf{S}\,\mathcal{N} \right\rangle a \twoheadrightarrow \left\langle \mathsf{S}\,a \middle/ \mathsf{S}\,a \cup \mathsf{S}\,\mathcal{N}\,a \right\rangle$ and no further reductions can happen. We see that a always is the right child of its parent.

□

Since $\mathcal{N}_0$ effectively ignores its argument, we have

8.3.5. Proposition. $\downarrow\left(\mathcal{N}_0\,\mathcal{N}\right)$

Proof. The normal form of $\mathcal{N}_0\,\mathcal{N}$ is that of $\mathcal{N}_0\,a$ with a replaced by $\mathcal{N}$. No reductions can happen inside $\mathcal{N}$, because it already is a normal form. No new redexes can be created.

□

A slightly more extended class is

8.3.6. Definition. $\mathcal{N}_{0,1} = \left\langle \mathsf{S}\,\mathcal{P}_2 \middle/ \mathsf{S} \cup \mathsf{S}\,\mathcal{N} \cup \mathsf{S}\,(\mathsf{S}\,\mathsf{T})\,\mathcal{P}_2 \right\rangle$

8.3.7. Example. $\mathsf{S}\,\mathsf{T}\,\mathsf{T} \subseteq \mathsf{S}\,\mathcal{P}_2\,(\mathsf{S}\,\mathcal{N}) \subseteq \left\langle \mathsf{S}\,\mathcal{P}_2 \middle/ \mathsf{S}\,\mathcal{N} \right\rangle \subseteq \mathcal{N}_{0,1}$

A term from $\mathcal{N}_{0,1}$ might move its argument into functional position, but there at most one argument is supplied.

8.3.8. Example. $\mathsf{S}\,\mathsf{T}\,\mathsf{T}\,a \to \mathsf{T}\,a\,(\mathsf{T}\,a) \to \mathsf{S}\,(\mathsf{T}\,a)\,\boxed{a\,(\mathsf{T}\,a)}$

8.3.9. Proposition. $\mathcal{N}_{0,1}\,a$ has a normal form, and $\mathcal{N}_{0,1}\,a \not\twoheadrightarrow a\,*\,*$.

Proof. The base cases are

1. $\mathsf{S}\,a$ is in normal form.

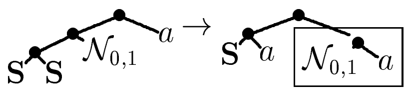
2. $\mathsf{S}\,\mathcal{N}\,a$ is in normal form.

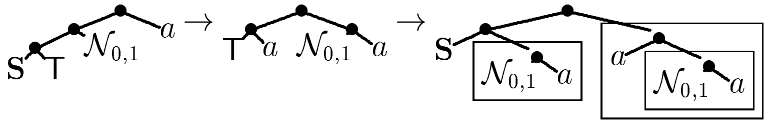
3. $\mathsf{S}\,(\mathsf{S}\,\mathsf{T})\,\mathsf{S}\,a \to \mathsf{S}\,\mathsf{T}\,a\,(\mathsf{S}\,a) \to \mathsf{T}\,(\mathsf{S}\,a)\,(a\,(\mathsf{S}\,a)) \to \mathsf{S}\,\boxed{a\,(\mathsf{S}\,a)}\,\left(\mathsf{S}\,a\,\boxed{a\,(\mathsf{S}\,a)}\right)$

4. $\mathsf{S}\,(\mathsf{S}\,\mathsf{T})\,\mathsf{T}\,a \to \mathsf{S}\,\mathsf{T}\,a\,(\mathsf{T}\,a) \to \mathsf{T}\,(\mathsf{T}\,a)\,(a\,(\mathsf{T}\,a))$
$\to \mathsf{S}\,(a\,(\mathsf{T}\,a))\,(\mathsf{T}\,a\,(a\,(\mathsf{T}\,a))) \to \mathsf{S}\,\boxed{a\,(\mathsf{T}\,a)}\,\left(\mathsf{S}\,\boxed{a\,(\mathsf{T}\,a)}\,\boxed{a\,\boxed{a\,(\mathsf{T}\,a)}}\right)$

where the claim can be verified at the boxed subterms.

The inductive steps are

1. 

2. 

□

If we look at the proof of proposition 8.3.9, we notice

8.3.10. Corollary. If $\mathcal{N}_{0,1}{}^{\wedge}a \hookrightarrow\!\!\!\rightarrow a{}^{\wedge}X$, then $a \lhd X$.

which will be used later on.

Proposition 8.3.9 will now be used to exhibit another set of normalizing terms:

8.3.11. Proposition. $\downarrow \left(\mathcal{N}_{0,1}{}^{\wedge}\mathcal{N}_0\right)$

Proof. First reduce $\mathcal{N}_{0,1}{}^{\wedge}\mathcal{N}_0$ as if $\mathcal{N}_0$ were a variable. By proposition 8.3.9, the only redexes that might arise are of the form $\mathcal{N}_0{}^{\wedge}*$. They might be nested, but all of them can be reduced to normal form by proposition 8.3.5.

□

By definition we have $\mathcal{N}_0 \subseteq \mathcal{N}_{0,1}$, and for later convenience we define

8.3.12. Definition. $\mathcal{N}_1 = \mathcal{N}_{0,1} \setminus \mathcal{N}_0; \mathcal{N}_2 = \mathcal{N} \setminus \mathcal{N}_{0,1}$.

So $\mathcal{N}$ is the disjoint sum of $\mathcal{N}_0$, $\mathcal{N}_1$, and $\mathcal{N}_2$.

Essentially, $\mathcal{N}_0$ is responsible for normalization, and $\mathcal{N}_2$ for non-normalization, as will be highlighted in the next two statements.

8.3.13. Proposition. $\mathcal{N}_2{}^{\wedge}\mathcal{Q}_2 \twoheadrightarrow \infty$.

Proof. By a manipulation of rational tree languages it can be shown that

$$\mathcal{N}_2 \subseteq \left\langle \mathcal{Q}_3{}^{\wedge}\mathcal{Q}_2 \right\rangle \ \cup\ \left\langle (\mathsf{S}{}^{\wedge}\mathcal{Q}_3){}^{\wedge}\mathcal{Q}_0 \right\rangle$$

This is exactly the right form to apply propositions 8.2.4 and 8.2.5.

(Moreover, this is exactly the kind of statement that can be verified by the program `RX`.) □

8.3.14. Proposition. $\mathcal{N}_1{}^{\wedge}\mathcal{N}_2 \twoheadrightarrow \infty$.

Proof. By corollary 8.3.10, somewhere in a reduction from $\mathcal{N}_1{}^{\wedge}\mathcal{N}_2$ there is a subexpression $\mathcal{N}_2{}^{\wedge}X$ with some $\mathcal{N}_2$ contained in X. Clearly then $X \in \mathcal{Q}_2$, so we have $\mathcal{N}_2{}^{\wedge}\mathcal{Q}_2$ and can apply proposition 8.3.13.

□

Combining propositions 8.3.5, 8.3.11 on one hand and 8.3.13, 8.3.14 on the other, the cases still missing in the analysis of $\mathcal{N}{}^{\wedge}\mathcal{N}$ are $\mathcal{N}_2{}^{\wedge}\mathcal{P}_2$ and $\mathcal{N}_1{}^{\wedge}\mathcal{N}_1$.

8.4 The sub-case $\mathcal{N}_1\ \mathcal{N}_1$

We look deeper into the set $\mathcal{N}_1$. We single out the subset of terms that move their argument a into functional position but which only supply them with the argument $\mathsf{S}\,a$.

8.4.1. Definition. $\mathcal{L}_0 = \left\langle \mathsf{T} \middle/ \mathsf{S}(\mathsf{S}\,\mathcal{P}_2)\,\mathsf{S} \right\rangle$

8.4.2. Proposition. $\mathcal{L}_0\,a$ has a normal form, and $\left\{Z \mid \mathcal{L}_0\,a \twoheadrightarrow a\,Z\right\} = \{\mathsf{S}\,a\}$.

Proof. The base cases are

1. $\mathsf{S}\,\mathsf{T}\,\mathsf{S}\,a \to \mathsf{T}\,a\,(\mathsf{S}\,a) \to \mathsf{S}\,(\mathsf{S}\,a)\,\boxed{a\,(\mathsf{S}\,a)}$

2. $\mathsf{S}\,(\mathsf{S}\,\mathsf{T})\,\mathsf{S}\,a \to \mathsf{S}\,\mathsf{T}\,a\,(\mathsf{S}\,a) \to \mathsf{T}\,(\mathsf{S}\,a)\,(a\,(\mathsf{S}\,a)) \to \mathsf{S}\,\boxed{a\,(\mathsf{S}\,a)}\,(\mathsf{S}\,a\,\boxed{a\,(\mathsf{S}\,a)})$

Then inductive step reads $\mathsf{T}\,\mathcal{L}_0\,a \to \mathsf{S}\,a\,\boxed{\mathcal{L}_0\,a}$.

□

Using $\mathcal{L}_0$, we find another class of normalizing terms.

8.4.3. Proposition. $\downarrow\left(\mathcal{L}_0\,\mathcal{N}_1\right)$.

Proof. Reduce $\mathcal{L}_0\,\mathcal{N}_1$ as if $\mathcal{N}_1$ were a variable. By proposition 8.4.2, the only redexes that may arise have the form $\mathcal{N}_1\,(\mathsf{S}\,\mathcal{N}_1)$. But $\mathsf{S}\,\mathcal{N}_1 \subseteq \mathsf{S}\,\mathcal{N} \subseteq \mathcal{N}_0$, so $\mathcal{N}_1\,(\mathsf{S}\,\mathcal{N}_1) \subseteq \mathcal{N}_1\,\mathcal{N}_0$ and therefore they normalize by proposition 8.3.11.

□

Another useful subset of $\mathcal{N}_1$ is

8.4.4. Definition. $\mathcal{L}_1 = \left\langle \mathsf{T} \middle/ \mathsf{S}\,\mathsf{T}\,(\mathsf{S}\,\mathcal{P}_2) \right\rangle$.

It has the following property:

8.4.5. Proposition. $\mathcal{L}_1\,a$ has a normal form, and $\left\{Z \mid \mathcal{L}_1\,a \twoheadrightarrow a\,Z\right\} = \left\{\mathsf{S}\,\mathcal{P}_2\,a\right\}$.

Proof. The base case is $\mathsf{S}\,\mathsf{T}\,(\mathsf{S}\,\mathcal{P}_2)\,a \to \mathsf{T}\,a\,(\mathsf{S}\,\mathcal{P}_2\,a) \to \mathsf{S}\,(\mathsf{S}\,\mathcal{P}_2\,a)\,\boxed{a\,(\mathsf{S}\,\mathcal{P}_2\,a)}$.

The inductive step is $\mathsf{T}\,\mathcal{L}_1\,a \to \mathsf{S}\,a\,\boxed{\mathcal{L}_1\,a}$.

□

This leads to yet another set of normalizing **S**-terms.

8.4.6. Proposition. $\downarrow\left(\mathcal{L}_1\cdot\mathcal{L}_0\right)$

Proof. Reduce $\mathcal{L}_1\cdot\mathcal{L}_0$, treating $\mathcal{L}_0$ as a variable. By proposition 8.4.5, the only new redexes have the form $\mathcal{L}_0\cdot((\mathsf{S}\cdot\mathcal{P}_2)\cdot\mathcal{L}_0)$. We have $\mathcal{L}_0 \subseteq \mathcal{N}_1$, therefore $(\mathsf{S}\cdot\mathcal{P}_2)\cdot\mathcal{L}_0 \subseteq (\mathsf{S}\cdot\mathcal{P}_2)\cdot\mathcal{N}_1 \subseteq \mathcal{N}_1$ by definition. But them $\mathcal{L}_0\cdot((\mathsf{S}\cdot\mathcal{P}_2)\cdot\mathcal{L}_0) \subseteq \mathcal{L}_0\cdot\mathcal{N}_1$ which normalizes by proposition 8.4.3.

□

Up to here we found some normalizing subsets of $\mathcal{N}_1\cdot\mathcal{N}_1$. Now we will show that all other elements of $\mathcal{N}_1\cdot\mathcal{N}_1$ have an infinite reduction.
For convenience, we take

8.4.7. Definition. $\mathcal{L}_{1,2} = \mathcal{N}_1 \setminus \mathcal{L}_0; \mathcal{L}_2 = \mathcal{L}_{1,2} \setminus \mathcal{L}_1$.

So $\mathcal{N}_1$ is the disjoint sum of $\mathcal{L}_0$, $\mathcal{L}_1$, and $\mathcal{L}_2$.
From $\mathcal{L}_{1,2}$ we build more non-normalizing terms.

8.4.8. Proposition. $\mathcal{L}_{1,2}\cdot\mathcal{L}_{1,2} \twoheadrightarrow \infty$

Proof. By manipulating rational languages we derive $\mathcal{L}_{1,2} \subseteq \left\langle \mathcal{Q}_2\cdot\mathcal{Q}_1 \right\rangle$. So proposition 8.2.3 is applicable.

□

Combining propositions 8.4.3, 8.4.6, and 8.4.8, there's exactly one case remaining in the analysis of $\mathcal{N}_1\cdot\mathcal{N}_1$, and that's $\mathcal{L}_2\cdot\mathcal{L}_0$. It is covered by the following

8.4.9. Proposition. $\mathcal{L}_2\cdot\mathcal{N}_1 \twoheadrightarrow \infty$

Proof. First we observe that

$$\mathcal{L}_2 \subseteq \left\langle \mathsf{T} \middle/ ((\mathsf{S}\cdot(\mathsf{S}\cdot\mathsf{T}))\cdot\mathsf{T}) \cup ((\mathsf{S}\cdot\mathsf{T})\cdot\mathcal{Q}_3) \right\rangle$$

(This can be verified, using `RX`.) Then we have two base cases:

1. $((\mathsf{S}\cdot\mathsf{T})\cdot\mathcal{Q}_3)\cdot\mathcal{N}_1 \to (\mathsf{T}\cdot\mathcal{N}_1)\cdot(\mathcal{Q}_3\cdot\mathcal{N}_1) \to (\mathsf{S}\cdot(\mathcal{Q}_3\cdot\mathcal{N}_1))\cdot\boxed{\mathcal{N}_1\cdot(\mathcal{Q}_3\cdot\mathcal{N}_1)}$. By corollary 8.3.10, applied to the boxed expression, there is a reduction $\mathcal{N}_1\cdot(\mathcal{Q}_3\cdot\mathcal{N}_1) \hookrightarrow\!\!\!\!\twoheadrightarrow (\mathcal{Q}_3\cdot\mathcal{N}_1)\cdot X$ with $\mathcal{Q}_3\cdot\mathcal{N}_1 \lhd X$, so that $X \subseteq \mathcal{Q}_2$. Moreover we sure have $\mathcal{N}_1 \subseteq \mathcal{Q}_2$, therefore the $(\mathcal{Q}_3\cdot\mathcal{N}_1)\cdot X \subseteq (\mathcal{Q}_3\cdot\mathcal{Q}_2)\cdot\mathcal{Q}_1$ and proposition 8.2.4 can be applied.
2. $((\mathsf{S}\cdot(\mathsf{S}\cdot\mathsf{T}))\cdot\mathsf{T})\cdot\mathcal{N}_1 \to ((\mathsf{S}\cdot\mathsf{T})\cdot\mathcal{N}_1)\cdot(\mathsf{T}\cdot\mathcal{N}_1) \subseteq ((\mathsf{S}\cdot\mathsf{T})\cdot\mathcal{Q}_3)\cdot\mathcal{N}_1$ which gets us back to the previous case.

The inductive step is just $(\mathsf{T}\cdot\mathcal{L}_2)\cdot\mathcal{N}_1 \to (\mathsf{S}\cdot\mathcal{N}_1)\cdot\boxed{\mathcal{L}_2\cdot\mathcal{N}_1}$.

□

8.5 The sub-case $\mathcal{N}_2\,\mathcal{P}_2$

Combining all of the previous, the analysis for $\mathcal{N}_2{}^\wedge\mathcal{Q}_2$ is complete, but $\mathcal{N}_2{}^\wedge\mathcal{P}_2 = \mathcal{N}_2{}^\wedge\mathsf{S} \cup \mathsf{T}$ is missing. We will show that after doing some reductions, the decision procedure for $\mathcal{N}_2{}^\wedge\mathcal{Q}_2$ can be used.

We define a notion of *complete reduction.* After one complete reduction step of an $X{}^\wedge\mathsf{S}$ redex, with X in normal form, no further redexes $X{}^\wedge\mathsf{S}$ can occur. And after a complete reduction step of an $X{}^\wedge\mathsf{T}$ redex, no further redexes $X{}^\wedge\mathcal{P}_2$ are possible. That's why we then may use the procedure that decides normalization for $\mathcal{N}{}^\wedge\mathcal{Q}_2$.

8.5.1. Definition. The complete reduction relation $X \xrightarrow{!} W$ is defined for all $X \in \mathcal{N}{}^\wedge\mathcal{M}$ and $W \in \mathcal{M}$ by

$$\mathsf{S}{}^\wedge Z \xrightarrow{!} \mathsf{S}{}^\wedge Z; \quad (\mathsf{S}{}^\wedge X){}^\wedge Z \xrightarrow{!} (\mathsf{S}{}^\wedge X){}^\wedge Z$$

$$((\mathsf{S}{}^\wedge X){}^\wedge Y){}^\wedge Z \xrightarrow{!} X'{}^\wedge Y' \quad \text{where} \quad X{}^\wedge Z \xrightarrow{!} X' \text{ and } Y{}^\wedge Z \xrightarrow{!} Y'$$

8.5.2. Corollary. Obviously $X \xrightarrow{!} W$ implies $X \twoheadrightarrow W$. For each $X \in \mathcal{N}{}^\wedge\mathcal{M}$, there is exactly one W with $X \xrightarrow{!} W$, so we can view this relation as a function, defined by structural recursion on the normal forms.

In a redex $((\mathsf{S}{}^\wedge X){}^\wedge Y){}^\wedge Z$, call Z the *argument* of the redex. We are also interested in subterms that might become arguments later. In a term, call a subterm (position) an *argument candidate* if it occurs as the right child of a subterm that has a left spine longer than two (i. e. whose left child is in $\mathcal{M}_2$). The purpose of $\xrightarrow{!}$ is that in $X{}^\wedge Y \xrightarrow{!} Z$, all argument candidates in terms reachable from Z are larger than Y. This is made precise by

8.5.3. Proposition. If $X \in \mathcal{N}$, and $X{}^\wedge\mathsf{S}_k \xrightarrow{!} Z_0 \to Z_1 \to \ldots$, then none of the Z_i contains a subterm $\mathcal{M}_2{}^\wedge\mathcal{P}_k$.

Proof. We prove that the property holds for Z_0. Then we proceed by induction.

We are going to show that $X{}^\wedge\mathsf{S}_k \xrightarrow{!} Z_0$ implies that no $\mathcal{M}_2{}^\wedge\mathcal{P}_k \trianglelefteq Z_0$. We use structural induction on X.

1. $X = \mathsf{S} \cup \mathsf{S}{}^\wedge X'$. Then $X{}^\wedge\mathsf{S}_k \xrightarrow{!} X{}^\wedge\mathsf{S}_k$ and does not contain $\mathcal{M}_2{}^\wedge\mathcal{P}_k$ because it's a normal form and $\mathcal{M}_2{}^\wedge\mathcal{P}_k$ is not.

2. $X = (\mathsf{S}{}^\wedge X_1){}^\wedge X_2$. Then $X{}^\wedge\mathsf{S}_k \xrightarrow{!} X_1'{}^\wedge X_2'$ where $X_1{}^\wedge\mathsf{S}_k \xrightarrow{!} X_1'$ and $X_2{}^\wedge\mathsf{S}_k \xrightarrow{!} X_2'$. By induction, the claim is true for X_1' and X_2'. We have $\mathsf{S}_k \triangleleft X_2'$, therefore $X_2' \in \mathcal{Q}_k$ and $X_1'{}^\wedge X_2' \notin \mathcal{M}_2{}^\wedge\mathcal{P}_k$.

Now chose the smallest i such that Z_i contains $\mathcal{M}_2{}^\wedge\mathcal{P}_k$. (We have $i > 0$.) This subexpression must have been created in the reduction $Z_{i-1} \to Z_i$. We list the possible relative positions of the redex $((\mathsf{S}{}^\wedge C){}^\wedge D){}^\wedge E$ (resp. its contractum $(C{}^\wedge E){}^\wedge(D{}^\wedge E)$) and the pattern $\mathcal{M}_2{}^\wedge\mathcal{P}_k$.

1. The pattern occurs inside C, D, or E. Then it would already have occured in Z_{i-1}, so i is not minimal.

2. $C{}^\wedge E \in \mathcal{M}_2{}^\wedge\mathcal{P}_k$ or $D{}^\wedge E \in \mathcal{M}_2{}^\wedge\mathcal{P}_k$ Then $E \in \mathcal{P}_k$ and in Z_{i-1} the redex $((\mathbf{S}{}^\wedge C){}^\wedge D){}^\wedge E \in ((\mathbf{S}{}^\wedge *){}^\wedge *){}^\wedge\mathcal{P}_k \subseteq \mathcal{M}_2{}^\wedge\mathcal{P}_k$ and i is not minimal.

3. $(C{}^\wedge E){}^\wedge(D{}^\wedge E) \in \mathcal{M}_2{}^\wedge\mathcal{P}_k$. In Z_{i-1}, we had $((\mathbf{S}{}^\wedge C){}^\wedge D){}^\wedge E \in ((\mathbf{S}{}^\wedge *){}^\wedge *){}^\wedge E \subseteq \mathcal{M}_2{}^\wedge E$. Because i is minimal, we have $E \in \mathcal{Q}_k$ leading to $D{}^\wedge E \in \mathcal{Q}_k$ and the contradiction $(C{}^\wedge E){}^\wedge(D{}^\wedge E) \in \mathcal{M}_2{}^\wedge\mathcal{Q}_k$.

4. $(\cdots((C{}^\wedge E){}^\wedge(D{}^\wedge E)){}^\wedge * \cdots){}^\wedge G \in \mathcal{M}_2{}^\wedge\mathcal{P}_k$. That is, $G \in \mathcal{P}_k$. But then $(\cdots(((\mathbf{S}{}^\wedge C){}^\wedge D){}^\wedge E){}^\wedge * \cdots){}^\wedge G \in \mathcal{M}_2{}^\wedge\mathcal{P}_k$ in Z_{i-1} as well, contradicting the minimality of i.

□

8.6 Deciding Normalization

8.6.1. Theorem. There is a procedure that decides whether an expression in $\mathbf{CL}(\mathbf{S})$ has a normal form.

Proof. $\mathbf{S}$ has a normal form. Therefore assume the expression is $X{}^\wedge Y$.
Recursively apply the procedure to X and Y and compute their normal forms if they exist. This recursion is terminating because it's structural. If one (or both) of the normal forms don't exist, halt and answer *not normalizing*. Otherwise call them X' and Y', respectively.
If $Y' \in \mathcal{Q}_2$, answer *normalizing* if $X'{}^\wedge Y' \in \mathcal{N}_0{}^\wedge\mathcal{N} \cup \mathcal{L}_0{}^\wedge\mathcal{N}_1 \cup \mathcal{L}_1{}^\wedge\mathcal{L}_0 \cup \mathcal{N}_1{}^\wedge\mathcal{N}_0$, and *not normalizing* else.
If $Y' \in \mathcal{P}_2$, then case T or $\mathbf{S}$ applies:
Case T: If $Y' = \mathsf{T}$, determine Z from $X'{}^\wedge Y' \xrightarrow{!} Z$. Then apply the procedure to Z. By proposition 8.5.3, no redex $\mathcal{M}_2{}^\wedge\mathsf{T}$ will occur in reductions from Z. Therefore case T will not happen again, so recursion terminates.
Case $\mathbf{S}$: If $Y' = \mathbf{S}$, determine Z from $X'{}^\wedge Y' \xrightarrow{!} Z$. Then apply the procedure to Z. By proposition 8.5.3, no redex $\mathcal{M}_2{}^\wedge\mathbf{S}$ will occur in reductions from Z. Therefore case $\mathbf{S}$ will not happen again, so recursion terminates. □

8.6.2. Remark. It is possible to omit cases $\mathbf{S}$ and T from the decision procedure, by explicitly giving rational sets X_S and X_T such that $X{}^\wedge\mathbf{S}$ (for $X \in \mathcal{N}$) normalizes iff $X \in X_S$, and similar for T. The exact construction has been omitted here because it is quite technical.

In the procedure given above, of course one would like to remove the restriction that X and Y be already in normal form, to save all those recursive calls. Indeed this will be done in chapter 9 where we give an explicit construction for the set of all normalizing $\mathbf{S}$-terms.

8.7 Related work

The Starling

The combinator **S** is called the *starling* in Smullyan's book [Smu85].

The decidability of normalization of **S**-terms had already been conjectured by E. Zachos in 1978.

Zachos completely analyzes the normalization of **S**-terms of size up to 9, by either finding the normal form or exhibiting patterns (similar to those given in the examples 5.5.1 or 5.5.2) that lead to an infinite reduction.

Still a proof that each non-normalizing term really admits such a pattern would be desirable.

The Lark

The combinator **L** with reduction rule $\mathbf{L}xy \to x(yy)$ is called the *lark* by Smullyan. It has been investigated in depth by R. Statman [Sta89]. He showed that the word problem is decidable for $\mathbf{CL}(\mathbf{L})$, using a very technical argument.

Later, Sprenger and Wymann-Böni [SWB93] gave a different proof. Essentially, they considered the system $x(yy) \to \mathbf{L}xy$ which is terminating, but not confluent. Then they try to make the system confluent. But completion does not stop. So they make an ingenious guess on the resulting infinite TRS, which can later be verified.

We conjecture that the word problem should also be decidable for the starling but it might be substantially more difficult than for the lark: If deciding normalization is a step towards deciding convertibility, than this first step is trivial for the lark, but, as we've seen, quite hard for the starling.

Chapter 9

Normalizing terms in CL(S)

9.1 Introduction

This chapter presents the rational language of normalizing terms in $\mathbf{CL}(\mathbf{S})$.

This amounts to finding pred $(\mathcal{N})$, the set of (many-step) predecessors of $\mathcal{N}$. In general, this need not be rational set, (see section 4.6,) and even if it were, there is no method known for finding it.

So what we're really doing here is to present a rational set of normalizing terms which is saturated (forward and backward closed under reduction) and which contains $\mathcal{N}$.

The construction is based on the subsets of $\mathcal{N}$ that have already been investigated in chapter 8. It has been found by doing some "educated guesses" and later verifying them by the program `RX`.

While this approach solves the problem for this case, a general method for constructing (backward) closures of rational languages (here: normal forms) would be desirable, so that we could avoid the guesswork.

It has to be admitted that the verification of the given grammar was only partial, as will be explained. Therefore we adopt the view that the main result of the thesis is *decidability* (chapter 8, which does not depend on machine proofs) rather than *rationality* (this chapter) of pred $(\mathcal{N})$.

9.2 A first look at $\mathrm{pred}(\mathcal{N})$

For this section, we agree on the following notation.

9.2.1. Definition. $\mathrm{pred}_Y(Z) = \{X \mid (X\,Y) \twoheadrightarrow Z\}$; $\mathrm{pred}(Z) = \{X \mid X \twoheadrightarrow Z\}$

That is, $\mathrm{pred}_Y(Z)$ are all terms X that reduce to Z when supplied with an argument from Y.
Earlier, $\mathrm{pred}(Z)$ had been called $\Delta^{\twoheadleftarrow}(Z)$.
By theorem 8.6.1, we have

9.2.2. Proposition.

$$\begin{aligned}\mathrm{pred}(\mathcal{N}) = \mathbf{S} &\cup (\mathrm{pred}(\mathcal{N}_0)\,\mathrm{pred}(\mathcal{N})) \cup (\mathrm{pred}(\mathcal{L}_0)\,\mathrm{pred}(\mathcal{N}_{0,1}))\\ &\cup (\mathrm{pred}(\mathcal{L}_1)\,\mathrm{pred}(\mathcal{L}_0)) \cup (\mathrm{pred}(\mathcal{N}_{0,1})\,\mathrm{pred}(\mathcal{N}_0))\\ &\cup (\mathrm{pred}_{\mathsf{S}}(\mathcal{N})\,\mathsf{S}) \cup (\mathrm{pred}_{\mathsf{T}}(\mathcal{N})\,\mathsf{T})\end{aligned}$$

Indeed this can be completed to a regular grammar for $\mathrm{pred}(\mathcal{N})$.

9.2.3. Proposition.

$$\begin{aligned}\mathrm{pred}(\mathcal{N}_0) = \mathbf{S} &\cup (\mathsf{S}\,\mathrm{pred}(\mathcal{N})) \cup (\mathsf{T}\,\mathrm{pred}(\mathcal{N}_0))\\ &\cup (\mathrm{pred}_{\mathsf{S}}(\mathcal{N}_0)\,\mathsf{S}) \cup (\mathrm{pred}_{\mathsf{T}}(\mathcal{N}_0)\,\mathsf{T})\end{aligned}$$

Proof. The expression matches the structure of $\mathcal{N}_0 = \mathbf{S} \cup (\mathsf{S}\,\mathcal{N}) \cup (\mathsf{T}\,\mathcal{N}_0)$. ☐

Similarly,

9.2.4. Proposition.

$$\begin{aligned}\mathrm{pred}(\mathcal{N}_{0,1}) = \mathbf{S} &\cup (\mathsf{S}\,\mathrm{pred}(\mathcal{N})) \cup ((\mathsf{S}\,(\mathsf{S}\,\mathsf{T}))\,\mathcal{P}_2) \cup ((\mathsf{S}\,\mathcal{P}_2)\,\mathrm{pred}(\mathcal{N}_{0,1}))\\ &\cup (\mathrm{pred}_{\mathsf{S}}(\mathcal{N}_{0,1})\,\mathsf{S}) \cup (\mathrm{pred}_{\mathsf{T}}(\mathcal{N}_{0,1})\,\mathsf{T})\end{aligned}$$

Proof. The expression matches the structure of $\mathcal{N}_{0,1} = \mathbf{S} \cup (\mathsf{S}\,\mathcal{N}) \cup ((\mathsf{S}\,(\mathsf{S}\,\mathsf{T}))\,\mathcal{P}_2) \cup ((\mathsf{S}\,\mathcal{P}_2)\,\mathcal{N}_{0,1})$. ☐

9.2.5. Proposition.

$$\begin{aligned}\mathrm{pred}_{\mathsf{S}}(\mathcal{N}) = \mathbf{S} &\cup (\mathsf{S}\,\mathrm{pred}(\mathcal{N}))\\ &\cup \mathrm{pred}\left((\mathsf{S}\,\mathrm{pred}_{\mathsf{S}}(\mathcal{N}_0))\,\mathrm{pred}_{\mathsf{S}}(\mathcal{N})\right) \cup \mathrm{pred}\left((\mathsf{S}\,\mathrm{pred}_{\mathsf{S}}(\mathcal{L}_0))\,\mathrm{pred}_{\mathsf{S}}(\mathcal{N}_{0,1})\right)\\ &\cup \mathrm{pred}\left((\mathsf{S}\,\mathrm{pred}_{\mathsf{S}}(\mathcal{L}_1))\,\mathrm{pred}_{\mathsf{S}}(\mathcal{L}_0)\right) \cup \mathrm{pred}\left((\mathsf{S}\,\mathrm{pred}_{\mathsf{S}}(\mathcal{N}_{0,1}))\,\mathrm{pred}_{\mathsf{S}}(\mathcal{N}_0)\right)\\ &\cup \mathrm{pred}_{\mathsf{S}}(\mathrm{pred}_{\mathsf{S}}(\mathcal{N})\,\mathsf{S}) \cup \mathrm{pred}_{\mathsf{T}}(\mathrm{pred}_{\mathsf{S}}(\mathcal{N})\,\mathsf{T})\end{aligned}$$

Proof. Take one $X \in \mathrm{pred}_{\mathsf{S}}(\mathcal{N})$. In the reduction $(X\,\mathsf{S}) \twoheadrightarrow \mathcal{N}$, consider the first step where the redex is at the root of the term. It has the form $((\mathsf{S}\,Y)\,Z)\,\mathsf{S}$, and its contractum is $(Y\,\mathsf{S})\,(Z\,\mathsf{S})$.
This must be an element of $\mathrm{pred}(\mathcal{N})$, so we can apply the classification of proposition 9.2.2. ☐

Similarly,

9.2.6. Proposition.

$$\begin{aligned}\mathrm{pred}_{\mathsf{T}}(\mathcal{N}) = \mathsf{S} \;\cup\; & \mathsf{S}\;\mathrm{pred}(\mathcal{N})\\ \cup\; \mathrm{pred}\left(\mathsf{S}\;\mathrm{pred}_{\mathsf{T}}(\mathcal{N}_0)\;\mathrm{pred}_{\mathsf{T}}(\mathcal{N})\right) \;&\cup\; \mathrm{pred}\left(\mathsf{S}\;\mathrm{pred}_{\mathsf{T}}(\mathcal{L}_0)\;\mathrm{pred}_{\mathsf{T}}(\mathcal{N}_{0,1})\right)\\ \cup\; \mathrm{pred}\left(\mathsf{S}\;\mathrm{pred}_{\mathsf{T}}(\mathcal{L}_1)\;\mathrm{pred}_{\mathsf{T}}(\mathcal{L}_0)\right) \;&\cup\; \mathrm{pred}\left(\mathsf{S}\;\mathrm{pred}_{\mathsf{T}}(\mathcal{N}_{0,1})\;\mathrm{pred}_{\mathsf{T}}(\mathcal{N}_0)\right)\\ \cup\; \mathrm{pred}_{\mathsf{S}}(\mathrm{pred}_{\mathsf{T}}(\mathcal{N}))\;\mathsf{S} \;&\cup\; \mathrm{pred}_{\mathsf{T}}(\mathrm{pred}_{\mathsf{T}}(\mathcal{N}))\;\mathsf{T}\end{aligned}$$

9.3 The case $\mathrm{pred}_{\mathsf{T}}(\mathcal{N})$

We immediately see

9.3.1. Proposition. $\mathrm{pred}_{\mathsf{T}}(\mathcal{L}_0) = \emptyset$

Therefore the expression reduces to

$$\begin{aligned}\mathrm{pred}_{\mathsf{T}}(\mathcal{N}) = \mathsf{S} \;\cup\; & \mathsf{S}\;\mathrm{pred}(\mathcal{N})\\ \cup\; \mathrm{pred}\left(\mathsf{S}\;\mathrm{pred}_{\mathsf{T}}(\mathcal{N}_0)\;\mathrm{pred}_{\mathsf{T}}(\mathcal{N})\right) \;&\cup\; \mathrm{pred}\left(\mathsf{S}\;\mathrm{pred}_{\mathsf{T}}(\mathcal{N}_{0,1})\;\mathrm{pred}_{\mathsf{T}}(\mathcal{N}_0)\right)\\ \cup\; \mathrm{pred}_{\mathsf{S}}(\mathrm{pred}_{\mathsf{T}}(\mathcal{N}))\;\mathsf{S} \;&\cup\; \mathrm{pred}_{\mathsf{T}}(\mathrm{pred}_{\mathsf{T}}(\mathcal{N}))\;\mathsf{T}\end{aligned}$$

We now give the following explicit constructions. Their correctness can be shown by using the program `RX`, which follows the algorithm for checking saturation indicated in section 4.6.

9.3.2. Proposition.

$$\begin{aligned}\mathrm{pred}_{\mathsf{S}}(\mathcal{L}_0) &= \left\langle \mathsf{T} \middle/ \mathsf{S}\;\mathsf{S}\;\mathcal{P}_2 \right\rangle\\ \mathrm{pred}(\mathcal{L}_0) &= \left\langle \mathsf{T} \middle/ \mathrm{pred}_{\mathsf{S}}(\mathcal{L}_0)\;\mathsf{S} \right\rangle\end{aligned}$$

9.3.3. Proposition.

$$\begin{aligned}\mathrm{pred}_{\mathsf{T}}(\mathcal{L}_1) &= \mathsf{S}\;\mathsf{T} \;\cup\; \mathsf{A}\\ \mathrm{pred}_{\mathsf{S}}(\mathcal{L}_1) &= \left\langle \mathsf{T} \middle/ \left\langle \mathsf{T} \middle/ \mathsf{S}\;\mathsf{T} \right\rangle \mathsf{S} \right\rangle\\ \mathrm{pred}(\mathcal{L}_1) &= \left\langle \mathsf{T} \middle/ \mathsf{S}\;\mathsf{T}\;\mathsf{S}\;\mathsf{T} \;\cup\; \mathrm{pred}_{\mathsf{T}}(\mathcal{L}_1)\;\mathsf{T} \;\cup\; \mathrm{pred}_{\mathsf{S}}(\mathcal{L}_1)\;\mathsf{S} \right\rangle\end{aligned}$$

9.3.4. Proposition.

$$\mathrm{pred}_{\mathsf{T}}(\mathcal{N}_0) = \mathcal{P}_2;\, \mathrm{pred}_{\mathsf{S}}(\mathcal{N}_0) = \mathcal{N}_{00}$$

9.3.5. Proposition.

$$\operatorname{pred}_{\mathsf{T}}(\mathcal{N}_{0,1}) = \mathcal{N}_{00} \cup \operatorname{pred}_{\mathsf{S}}(\mathcal{L}_0)$$
$$\operatorname{pred}_{\mathsf{S}}(\mathcal{N}_{0,1}) = \operatorname{pred}_{\mathsf{T}}(\mathcal{N}_{0,1}) \cup \left\langle \mathsf{T} \middle/ \left\langle \mathsf{T} \middle/ \mathsf{S}\,\mathcal{N}_{00} \right\rangle \mathsf{S} \right\rangle$$

From proposition 9.2.6 we obtain

9.3.6. Proposition. $\operatorname{pred}_{\mathsf{T}}(\mathcal{N}) \supseteq \operatorname{pred}\left(\mathsf{S}\,\operatorname{pred}_{\mathsf{T}}(\mathcal{N}_0)\,\operatorname{pred}_{\mathsf{T}}(\mathcal{N})\right) \supseteq \mathsf{S}\,\mathcal{P}_2\,\operatorname{pred}_{\mathsf{T}}(\mathcal{N})$

In fact we have

9.3.7. Proposition.

$$\operatorname{pred}_{\mathsf{T}}(\mathcal{N}) = \mathsf{S} \cup \mathsf{S}\,\operatorname{pred}(\mathcal{N}) \cup \operatorname{pred}_{\mathsf{T}}(\mathcal{N}_{0,1})\,\mathcal{P}_2 \cup \mathsf{S}\,\mathcal{P}_2\,\operatorname{pred}_{\mathsf{T}}(\mathcal{N})$$
$$\cup \operatorname{pred}_{\mathsf{S}}(\operatorname{pred}_{\mathsf{T}}(\mathcal{N}))\,\mathsf{S} \cup \operatorname{pred}_{\mathsf{T}}(\operatorname{pred}_{\mathsf{T}}(\mathcal{N}))\,\mathsf{T}$$

As had been observed in proving theorem 8.6.1, the difficult cases are $X\,\mathsf{S}$ and $X\,\mathsf{T}$, or rather $\operatorname{pred}_{\mathsf{S}}(\mathcal{N})$ and $\operatorname{pred}_{\mathsf{T}}(\mathcal{N})$. That's why we need to evaluate stacks of $\operatorname{pred}_{\mathcal{P}_2}(\cdot)$.

9.3.8. Proposition.

$$\operatorname{pred}_{\mathsf{S}}(\operatorname{pred}_{\mathsf{T}}(\mathcal{N})) = \operatorname{pred}_{\mathsf{S}}(\mathcal{N}_{0,1}) \cup \left\langle \mathsf{T} \middle/ \mathsf{S}\,\operatorname{pred}_{\mathsf{T}}(\mathcal{N}_{0,1}) \right\rangle$$
$$\operatorname{pred}_{\mathsf{T}}(\operatorname{pred}_{\mathsf{T}}(\mathcal{N})) = \left\langle \mathsf{T} \middle/ \mathsf{S}\,\operatorname{pred}_{\mathsf{T}}(\mathcal{N}_{0,1}) \right\rangle$$
$$\operatorname{pred}_{\mathsf{T}}(\operatorname{pred}_{\mathsf{S}}(\mathcal{N})) = \left\langle \mathsf{T} \middle/ \mathsf{S}\,(\mathsf{S}\,\mathsf{T})\,\mathsf{T} \right\rangle \cup \operatorname{pred}_{\mathsf{S}}(\mathcal{L}_1)$$

That completes the expression for $\operatorname{pred}_{\mathsf{T}}(\mathcal{N})$, but we're still missing $\operatorname{pred}_{\mathsf{S}}(\mathcal{N})$.

9.4 The case $\operatorname{pred}_{\mathsf{S}}(\mathcal{N})$

Intricate expressions are necessary for stacks of $\operatorname{pred}_{\mathsf{S}}(\cdot)$, so they'll deserve names in their own right.

9.4.1. Definition. *one* $= \operatorname{pred}_{\mathsf{S}}(\mathcal{N})$, *two* $= \operatorname{pred}_{\mathsf{S}}(\operatorname{pred}_{\mathsf{S}}(\mathcal{N}))$, *three* $= \operatorname{pred}_{\mathsf{S}}(\operatorname{pred}_{\mathsf{S}}(\operatorname{pred}_{\mathsf{S}}(\mathcal{N})))$, *four* $= \operatorname{pred}_{\mathsf{S}}(\operatorname{pred}_{\mathsf{S}}(\operatorname{pred}_{\mathsf{S}}(\operatorname{pred}_{\mathsf{S}}(\mathcal{N}))))$

Fortunately, we don't need larger numbers, as *five* $=$ *six* $= \ldots = \mathcal{N}_{00}$.
From proposition 9.2.5 we derive

9.4.2. Proposition. *one* $= \operatorname{pred}_{\mathsf{S}}(\mathcal{N}) \supseteq \operatorname{pred}\left(\mathsf{S}\,\operatorname{pred}_{\mathsf{S}}(\mathcal{N}_0)\,\operatorname{pred}_{\mathsf{S}}(\mathcal{N})\right) \supseteq \mathsf{S}\,\mathcal{N}_{00}\,$*one*

and we easily verify

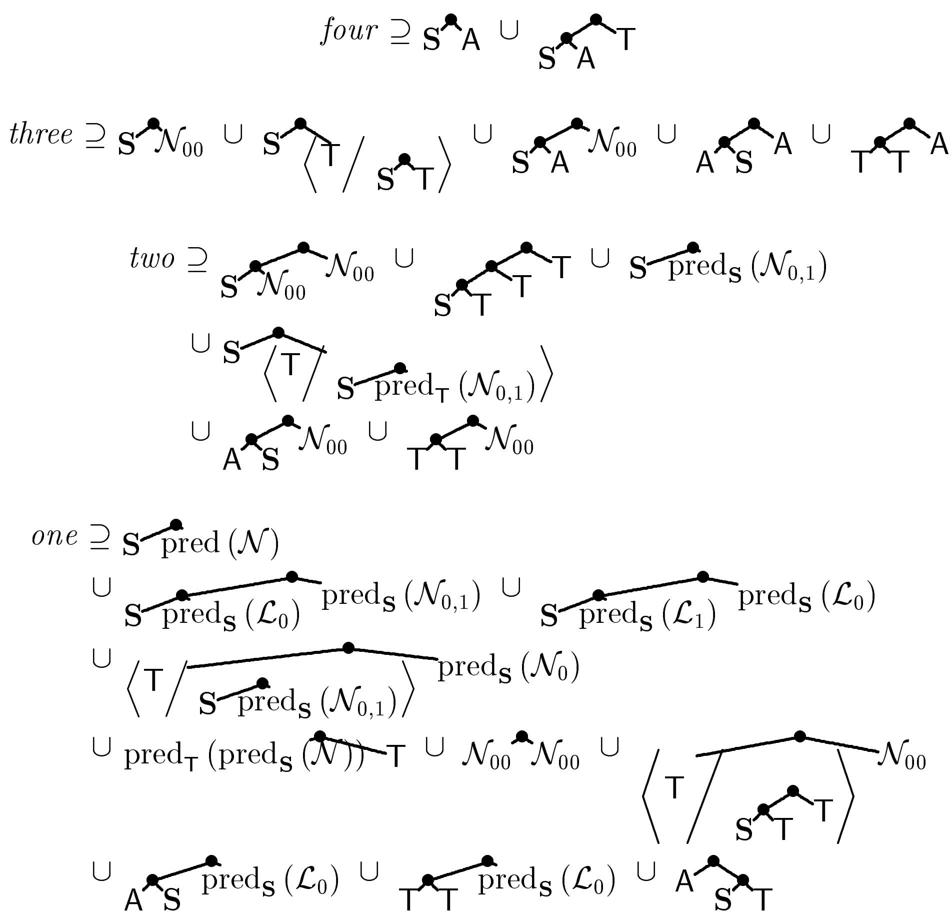
9.4.3. Proposition. $two \supseteq \mathsf{T}\,two$, $three \supseteq \mathsf{T}\,three$, $four \supseteq \mathsf{T}\,four$

9.4.4. Proposition. $one \supseteq two\,\mathsf{S}$, $two \supseteq three\,\mathsf{S}$, $three \supseteq four\,\mathsf{S}$

9.4.5. Proposition. $four \supseteq \mathcal{N}_{00}$, $three \supseteq \mathcal{N}_{00}$, $two \supseteq \mathcal{N}_{00}$, $one \supseteq \mathcal{N}_{00}$

The propositions 9.4.2, 9.4.3, 9.4.4, 9.4.5 are interpreted as rules in the grammars for *one*, *two*, *three*, *four*. They have to be completed by the following

9.4.6. Proposition.



That completes the regular grammar that generates $\mathrm{pred}\,(\mathcal{N})$.

All of the constructions can be checked with `RX`.

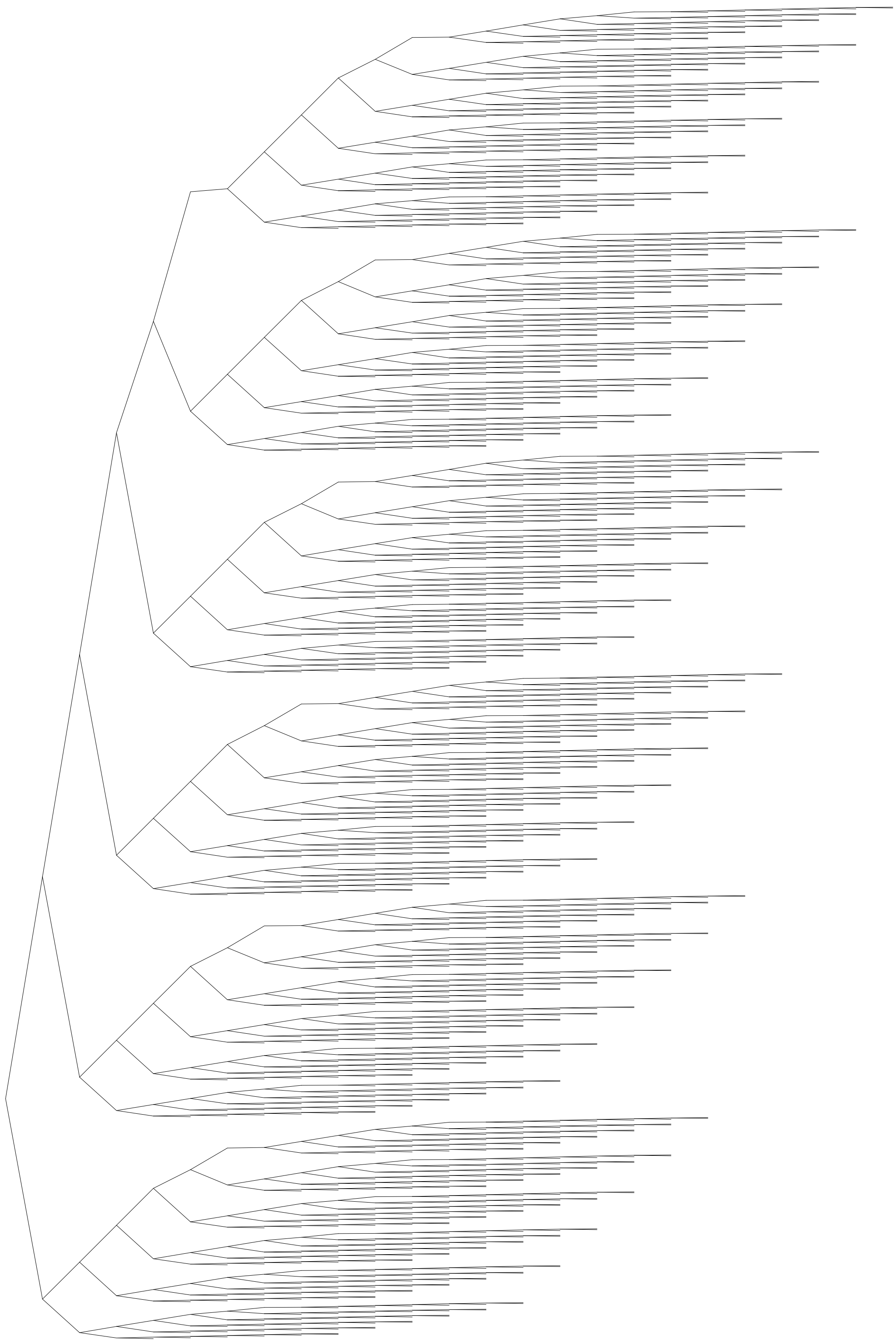
9.5 Conclusion

To be fair, it has to be admitted that the a complete check has not been achieved so far. The grammar is rather large, and pushes `RX`'s limits.

However, extensive saturation checking has been done for reasonable sub-grammars. Further, it can be verified by pen and paper that all **S**-terms generated by the full grammar really are normalizing. The question is of course whether we forgot to produce some normalizing terms. Again extensive testing of example terms shows that this doesn't seem to be the case. As a conclusion, what really had been shown here is that $\mathrm{pred}\,(\mathcal{N})$ *strongly* seems to be rational, and its grammar looks more or less like the one presented above, and it could be obtained and verified by using some more accuracy and/or time in machine or human computation.

Part III

Aftermath



Normal form of the monster **S** A (**S S S S S S**) **S S S**
reached after 55 innermost reductions

Chapter 10

Conclusion

This thesis presented a detailed analysis of some properties of the system $\mathbf{CL}(\mathbf{S})$: absence of ground loops, top-termination, decidability of normalization, rationality of normalizing terms.

Absence of ground loops is shown by employing standard techniques: labeling and recursive path orders. So the proof looks quite nice and abstract.

The other results are proved by rather technical manipulations. In the case of top termination and decidability of normalization, these can still all be done and checked by hand, although this is a time-consuming process.

To verify the grammar that gives the rational language of normalizing expressions, the software `RX` was employed. The theory underlying that system has been described, and its usage is detailed in an appendix to this thesis.

Of course decidability of normalization follows from rationality of normalizing terms. Yet we chose to present these results separately because one depends on machine verification, and the other does not.

All of the proofs use the fact that $\mathbf{CL}(\mathbf{S})$ is an orthogonal, non-erasing term rewriting system.
On the other hand they depend on the structure of the $\mathbf{S}$ rule. However, these dependencies are implicit, and scattered through all of the text. Therefore, for instance, the regular expressions that describe normalization look rather intricate, and they seem to appear *out of the blue*. It remains to find a general argument that tells *why* rational languages can describe $\mathbf{CL}(\mathbf{S})$, and a construction that *synthesizes* these languages, just given the $\mathbf{S}$ rule as input.

On one hand, the situation is aesthetically intriguing: a very rich structure arises from quite simple boundary conditions — just one, small rewrite rule. On the other hand, "surprising" answers indicate that the question wasn't asked in the right language. This thesis sheds some new light on $\mathbf{CL}(\mathbf{S})$ but this system remains something far from completely understood. The most intriguing open question is that of decidability of the word problem. We indicated how this might be attacked, using descriptions of infinite normal forms by finite automata.

We hope that further research into $\mathbf{CL}(\mathbf{S})$ will solve the word problem, and will also better explain the structure of the results that were obtained up to now, and lead to generalizations that are applicable to a larger class of rewrite systems.

Chapter 11

Bibliography

[ADN92] A. Arnold, A. Dicky, and M. Nivat. A note about minimal non-deterministic automata. *Bulletin of the EATCS*, 47:166–169, 1992.

[AGM92] S. Abramsky, D. Gabbay, and T. Maibaum, editors. *Handbook of Logic in Computer Science*, volume 2. Clarendon Press, Oxford, 1992.

[Aug96] Lennart Augustson. *The HBC compiler, Documentation for Release 0.9999.3*. Department of Computer Science, Chalmers University of Technology, Göteborg, ftp://ftp.cs.chalmers.se/pub/haskell/chalmers/, 1996.

[Ave95] Jürgen Avenhaus. *Reduktionssysteme*. Springer, 1995.

[Bar84] H. P. Barendregt. *The Lambda Calculus, Its Syntax and Sematics*. Elsevier Science Publishers, 1984.

[BK79] Jan Bergstra and Jan Willem Klop. Church–Rosser strategies in the Lambda calculus. *Theoretical Computer Science*, 9:27–38, 1979.

[CDGV91] J. L. Coquideé, M. Dauchet, R. Gilleron, and S. Vágvölgyi. Bottom-up tree pushdown automata and rewrite systems. In Dershowitz [Der91], pages 287–298.

[Chu41] Alonzo Church. *The Calculi of Lambda Conversion*, volume 6 of *Annals of Mathematics Studies*. Princeton University Press, 1941.

[Der91] N. Dershowitz, editor. *Rewriting Techniques and Applications*, number 488 in Lecture Notes in Computer Science. Springer Verlag, 1991.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. *Rewrite Systems*, pages 243–319. Volume B of van Leeuwen [vL90], 1990.

[DJ91] Nachum Dershowitz and Jean-Pierre Jouannaud. Notations for rewriting. *Bull. EATCS*, 43:162–172, February 1991.

[DKP91] Nachum Dershowitz, Stéfane Kaplan, and David A. Plaisted. Rewrite, rewrite, rewrite, rewrite, rewrite, ... *Thoeretical Computer Science*, 83:71–96, 1991.

[DT90] M. Dauchet and S. Tison. The first order theory of ground rewriting systems is decidable. In LICS [LIC90], pages 242–248.

[ELS95] Uffe H. Engberg, Kim G. Larsen, and Arne Skou, editors. *Workshop on Tools and Algorithms for the Construction and Analysis of Systems (Aarhus, Denmark)*, number NS-95-2 in Notes Series, Department of Computer Science University of Aarhus, May 1995. BRICS.

[Gen97] Thomas Genet. Proving termination of sequential reduction relation using tree automata. talk given at 3rd International Workshop on Termination, Ede, The Netherlands, May 1997.

[Gra92] Bernhard Gramlich. Relating innermost, weak, uniform and modular termination of term rewriting systems. In Voronkov [Vor92], pages 285–296.

[GS84] F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[GT95] Rémi Gilleron and Sophie Tison. Regular tree langugages and rewrite systems. *Fund. Inf.*, 24:157–175, 1995.

[HH92] Dieter Hofbauer and Maria Huber. Computing linearizations using test sets. In Rusinowitch and Rémy [RR92], pages 287–301.

[Hin77] Roger Hindley. Combinatory reductions and lambda reductions compared. *Zeitschr. f. math. Logik und Grundlagen d. Math.*, 23:169–180, 1977.

[HS86] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-calculus.* Cambridge University Press, 1986.

[Jon96] Mark P. Jones. *Hugs 1.3, The Haskell User's Gofer System.* Department of Computer Science, University of Nottingham, ftp://ftp.cs.nott.ac.uk/pub/haskell/hugs/, 1996.

[KKSdV95] Richard Kennaway, Jan Willem Klop, Ronan Sleep, and Fer-Jan de Vries. Transfinite reductions in orthogonal term rewriting systems. *Information and Computation*, 119(1):18–38, May 1995.

[Klo80] Jan Willem Klop. *Reduction Cycles in Combinatory Logic*, pages 193–214. In Seldin and Hindley [SH80], 1980.

[Klo92] Jan Willem Klop. *Term Rewriting Systems*, pages 2–117. Volume 2 of Abramsky et al. [AGM92], 1992.

[Knu92] Donald Erwin Knuth. *Literate Programming*, volume 27 of *CSLI Lecture Notes.* Center for the Study of Language and Information, Standford, 1992.

[KT92] Gregory Kucherov and Mohamed Tajine. Decidability of regularity and related properties of ground normal form languages. In Rusinowitch and Rémy [RR92], pages 272–286.

[Kur90] Winfried Kurth. *Termination und Konfluenz von Semi-Thue-Systemen mit nur einer Regel.* PhD thesis, Technische Universität Clausthal, 1990.

[KvOvR93] J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. CWI report CS-R9362, Centrum foor Wiskunde en Informatica, 1993.

[Lam94] Leslie Lamport. *LaTeX, A Document Preparation System.* Addison Wesley, 1994. second edition.

[LIC90] IEEE Computer Society Press. *Logic in Computer Science.* IEEE Computer Society Press, 1990.

[MP95] Oliver Matz and Andreas Potthoff. Computing small nondeterministic finite automata. In Engberg et al. [ELS95], pages 74–88.

[Nip95] Tobias Nipkow. Equational reasoning: Lecture notes for a course on term rewriting etc. http://hpbroy3.informatik.tu-muenchen.de/˜nipkow/misc/er.dvi.gz, 1995.

[Oya87] M. Oyamaguchi. The church-rosser property for ground term rewriting systems is decidable. *Theoretical Computer Science*, 49:43–79, 1987.

[Röj96] Niklas Röjemo. *nhc: a space-efficient Haskell compiler.* Department of Computer Science, Chalmers University of Technology, Göteborg, ftp://ftp.cs.chalmers.se/pub/haskell/nhc13/, 1996.

[RR92] M. Rusinowitch and J. L. Rémy, editors. *Conditional Term Rewriting Systems, Third International Workshop*, number 656 in Lecture Notes in Computer Science. Springer Verlag, 1992.

[Sal88] K. Salomaa. Deterministic tree pushdown automata and monadic tree rewriting systems. *J. Comp. System Sci.*, 37:367–394, 1988.

[SH80] J. P. Seldin and J. R. Hindley, editors. *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980.

[Smu85] Raymond Smullyan. *To Mock a Mockingbird: and other logic puzzles including an amazing adventure in combinatory logic.* Knopf, New York, 1985.

[Sta89] Richard Statmann. The word problem for Smullyan's lark combinator is decidable. *J. Symbolic Comput.*, 7:103–112, 1989.

[SWB93] M. Sprenger and M. Wymann-Böni. How to decide the lark. *Theoretical Computer Science*, 110:419–432, 1993.

[Tea96] The GHC Team. *The Glorious Glasgow Haskell Compilation System, Version 2.01, User's Guide.* Department Of Computer Science, University of Glasgow, ftp://ftp.dcs.gla.ac.uk/pub/haskell/glasgow/, 1996.

[vdPSvdM80] W. L. van der Poel, C. E. Schaap, and G. van der Mey. New arithemtical operators in the theory of combinators. *Indagationes Mathematicae*, 42 (3):271–325, September 1980.

[VG92] S. Vágvölgyi and R. Gilleron. For a rewrite system it is decidable whether the set of irreducible ground terms is recognizable. *Bulletin of the EATCS*, 48:197–209, 1992.

[vL90] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science*, volume B. Elsevier Science Publishers B.V., 1990.

[Vor92] A. Voronkov, editor. *International Conference on Logic Programming and Automated Reasoning, St. Petersburg*, number 624 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1992.

[Wal95] J. Waldmann. Typesetting haskell with LaTeX. Forschungsergebnisse Math/95/9, Friedrich-Schiller-Universität Jena, Fakultät für Mathematik und Informatik, 1995.

[Wal96] J. Waldmann. Infinitite unique head reductions for $Sxyz \to xz(yz)$. Forschungsergebnisse 96/27, Friedrich-Schiller-Universität Jena, Fakultät für Mathematik und Informatik, 1996.

[Wal97] Johannes Waldmann. *The Combinator S*. PhD thesis, Friedrich-Schiller-Universität Jena, 1997.

[Zac78] Efstathios Zachos. Kombinatorische Logik und S-Terme. Berichte des Instituts für Informatik 26, Eidgenössische Technische Hochschule Zürich, 1978.

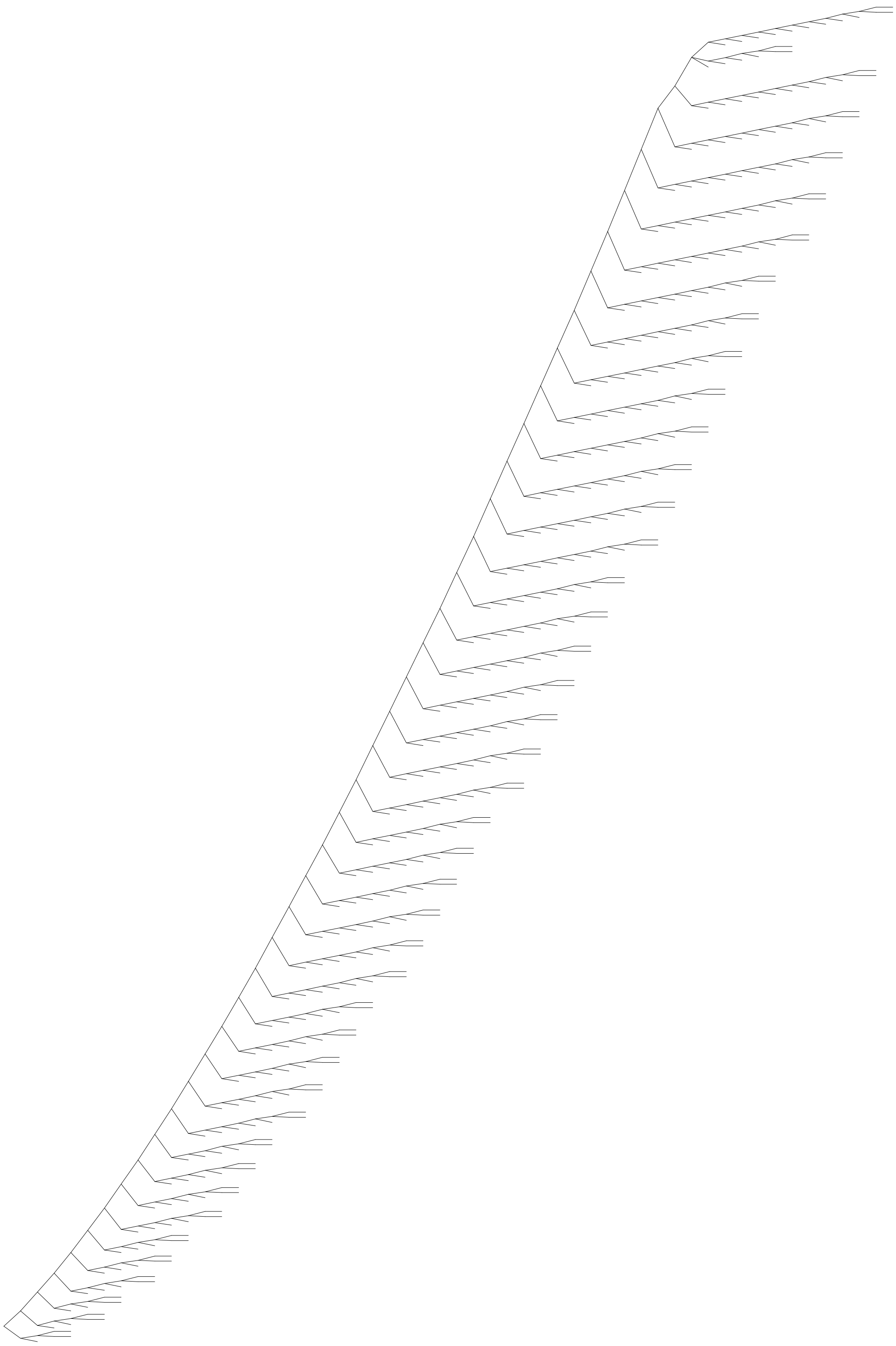
[ZG96] H. Zantema and A. Geser. Non-looping rewriting. Technical Report UU-CS-1996-03, Utrecht University, January 1996.

Chapter 12

Index

Part IV

Appendix



S T T (**S T T**)
after 50 reductions

Appendix A

RX User Manual

A.1 Introduction

This chapter describes the program RX, written within the framework of this thesis. RX handles rational tree languages in term algebras with arbitrary signatures. RX performs the usual operations on rational languages.

RX has been used to guess, and later verify, a lot of the relations between languages that are used in the decision procedure for $\mathbf{CL}(\mathbf{S})$, and the grammar for normalizing terms, given in chapters 8 and 9 of this thesis.

Furthermore, RX's input and output behaviour can be customized so that it integrates smoothly with a typesetting system, much in the spirit of a *literate programming* tool. RX input may be contained in a text document, and RX output may be embedded in such a document. Additionally, RX output can contain formatting instructions to the typesetter. In fact RX is used to format the very thesis you are now reading.

A.2 RX internals

RX internally represents rational tree languages by finite tree automata. They can be non-deterministic (bottom-up or top-down) or deterministic (bottom-up).

For details of the underlying theory, see chapter 4.

The external form, used for input and output, is a non-deterministic top-down automaton that could also be called a rational grammar.

RX is written entirely in the pure, lazy functional programming language Haskell. It uses Haskell-1.3 features. RX is known to work with these compilers/interpreters: hugs, ghc-2.02, hbc-0.9999.4. It does not depend on compiler-specific extensions or libraries.

RX uses publically available libraries for

1. Finite Maps and Sets
2. Parsing
3. Pretty Printing

(Copies of these are included with the RX source code distribution.) These libraries helped me a lot when programming RX, as they provide tried and tested and optimized solutions for standard tasks, and allowed me to focus on finite automata programming.

The source code of RX, and some additional information, is available under terms of the GNU public license from `http://www5.informatik.uni-jena.de/~joe/rx/`. [1]

A.3 RX syntax

RX lexically and syntactically borrows from the Haskell programming language.

RX lexic

RX code consists of lines of characters.

The input may contain comments. A comment extends from `--` to the end of the line, or it is enclosed in between `{-` and `-}`.

RX has these lexical tokens:

1. number (a sequence of digits)
2. name (an alphabetic character, followed by a sequence of alphanumeric characters)
3. operator (a sequence of non-alphanumeric non-blank characters)
4. string (enclosed in double quotes)
5. group symbols `(`, `)`, `[`, `]`, `{`, `}`
6. group separator `,` (comma). Note that `;` (semi-colon) is not a separator but an operator.

[1] now `http://www.informatik.uni-leipzig.de/~joe/rx/`

The names `arity` and `form` are reserved.

A name may contain characters `"_"`, `"^"`, and `"'"`, and they'll have their expected interpretation when the output is embedded in a LaTeX document.

Names and operators together are called *identifiers*.

A name enclosed in back-quotes behaves as if it were an operator. From now on, *operator* encompasses that meaning.

An operator enclosed in parentheses behaves as if it were a name. From now on, *name* encompasses that meaning.

`RX` syntax

`RX` code is a sequence of *declarations* and *expressions*.

A *declaration* declares one of these things:

1. the form of an identifier (see below),

2. the arity (number of required arguments) of names.

   Example: `arity 2 power, times`

3. the precedence and associativity of operators.

   Example:

   ```
   infixl 7 *
   infixl 5 +
   ```

An *expression* is a sequence of *clauses* separated by operators.

A *clause* is a name followed by a sequence of *atoms*.

An *atom* is a name or a *group*.

A *group* is a sequence of expressions, separated by group separators, enclosed in group symbols. The group `(x)`, where `x` is an expression, is semantically equivalent to `x` alone.

`RX` special syntax

In `RX`, there is no partial application and there are no higher order functions. In a function application, the function must get exactly the number of arguments that its arity requires.

Arities that have not been explicitly declared (see above) are implicitly set to 0 for names and 2 for operators.

This is more restrictive that Haskell but it allows for a nice shorthand notation: the operator `@` (called *apply*) is implicitly inserted whenever a function has *more* arguments than necessary. Let `f` have arity 2, then `f x y b c` is parsed as `((f x y) @ b) @ c`. This corresponds to the convention used in combinatory logic and lambda calculus.

This behavior can be controlled by the switch `implicit` and by the option `apply`. If `implicit` is `on`, then the binary operator that is the value of the option `apply` is inserted. Normally, this is `@`.

A.4 RX semantics

In RX, the semantics of each identifier is a function from state × sequences of values to state × values. The length of the sequence is the arity of the identifier.

Currently, the values are finite tree automata, but this could be extended to a wider range of types.

The state is just the mapping from identifiers to their meaning (semantics). The pair (identifier, meaning) is called binding.

The state map can be extended by new bindings, but existing meanings cannot be overridden.

The operator `=` adds a new binding. Examples:

```
q = 0
f a b = a + b * b
```

If the arity of the bound identifier is 0 (a constant declaration), then the value of the whole expression is that of the right hand side. If the arity is more than 0 (a function definition), then the value of the whole expression is undefined.

The operator ; combines the evaluation of two expressions sequentially. In `a; b`, first `a` is evaluated, then `b` is evaluated in the state that resulted from the evaluation of `a`. The value of `b` is the final result.

RX special semantics

So far, the description does not contain any hint on the range of values that RX operates on. This modularity is intentional, and could be explored when extending RX or re-using parts of it for other purposes.

But we are going to use RX for tree automata. The will be input and output as rational tree grammars.

First we need to specify the term algebra we're working in. This happens implicitly.

Each expression has a term algebra associated with it that consists of the *constructors* that happen to be contained in the expression's value.

A *constructor* is an identifier that is not bound.

Example:

```
arity 0 S
infixl 60 @
```

This declares two constructors, one nullary, one binary.

A grammar is built by the name `grammar` that gets two arguments: a start expression and a set of rules. Example:

```
grammar x { x -> S, x -> S x + S x x }
```

Remember that this is in fact `x -> S @ x + (S @ x) @ x` by the rule of implicit application.

Each rule has a variable on the left and an *rule expression* on the right. The start expression of the grammar also is an *rule expression*.

A *rule expression* looks like an expression (see above) but there are restrictions on where grammar variables might occur: Starting from the top of the syntax tree of a rule expression, on the path to any grammar variable contained therein, only constructors and `++` operators are allowed. In branches of rule expressions where no grammar variables occur, arbitrary expressions are permitted.

Note that `grammar` does not evaluate its arguments. Rather, it behaves like a LISP special form that gets to see the syntax tree of its argument.

RX predefined identifiers

There are predefined operators for rational languages:

1. `++` union
2. `\\` difference
3. `&` intersection

and predefined functions that only transform a language's representation, but keep its meaning (the set of its words).

1. `unify`: unify seemingly identical states in the automaton
2. `useful` only keep reachable and productive states
3. `det` make automaton deterministic
4. `min` make automaton deterministic and then minimal

Evaluation of expressions

If the switch `eval` is `on`, each expression is evaluated and the evaluation may result in state changes (because new bindings were added).

However, if evaluating an inline expression changes the current RX state, this change is not promoted to other code blocks or snippets.

That is, assignments cannot be hidden inside the text. Rather, they must be made visible in code blocks. (You can still cheat there, by turning the output off, see below.)

If the switch `exp` is `on`, the input expression is printed. For normal typesetting, you would

```
.set (eval=off,exp=on)
```

If the switch `res` is `on`, the result of evaluating the input is printed. (If evaluation took place at all.)

Preprocessing

The switches `unify,useful,det,min` tell whether the so named functions should be automatically inserted on top of all expressions that are being evaluated. In bindings, this refers to expressions right of the `=`.

The switch `expand` tells whether the input expansion is shown or hidden.

Postprocessing

The result of an evaluation is a finite tree automaton. It is shown as a rational grammar. There are switches that control the appearance of that grammar.

1. `foldconst` rules `x -> e` where `e` does not contain grammar variables, are applied to all other rules, and the deleted.

2. `foldnonrec` rules `x -> e` where `e` does not contain `x`, are applied to all other rules, and the deleted.

3. `hidegrammar` if the rule set is empty, don't say `grammar x {}`. Rather, use `x` instead.

A.5 Controlling RX's behaviour

In addition to the core language, RX has means for preprocessing its input, and specifying aspects of its behaviour.

Options

While processing input, RX keeps a set of *options*.
An *option set* is a mapping from strings to strings. It maps *options* to *values*.
A *switch* is an option that only takes on values of `on` or `off`.
Options can be set by *directives*, or on the command line when invoking RX.
A *directive* starts with a dot as the very first character in a line.
Options can be set globally by

```
.set (opt1=val1, opt2=val2, ...)
```

A `.set` is effective from where it stands to the end of the file it is located in.
Option names and values can be given as RX names or strings.
A value can also refer to another option's value by prepending a dollar sign to it, as in `.set (foo=$bar)`.
Options can be set locally by

```
.begin (opt1=val1,...)
    ...
.end
```

The new option set is effective only in between `.begin` and `.end`.

Default values for options

The file `Defaults.hs` from the RX source distribution contains the option set that is compiled in.
All of those options can be changed, however.

Importing files

RX input can be split over several files. The `.import` directive includes the contents of a file into the input stream.

```
.import file
```

`file` must be the file's complete name as an RX string.

The string `"-"`, when used as a file name, means the standard input stream. This is only really useful on the command line, see below.

It is possible to set some options that are only in effect while the imported file is processed

```
.import (opt1=val1,...) file
```

In any case, none of the possible `.set` commands in the *imported* file has an effect on the options in the *importing* file.

RX command line invocation

When the RX executable is invoked, it analyzes its command line, takes appropriate actions, and writes its output to the standard output stream, from where it may be redirected by other means.

On the command line, you may specify option values and files to read.

Example:

```
RX opt1=val1 opt2=val2 filea opt3=val3 fileb filec -
```

is handled as

```
.set (opt1=val1)
.set (opt2=val2)
.import "filea"
.set (opt3=val3)
.import "fileb"
.import "filec"
.import "-"
```

Note that the last line means "read standard input".

Additionally, RX might understand options for its underlying runtime system. They depend on the compiler that had been used when building RX. A common idiom is

```
RX +RTS -H50M -RTS opt1=val1 ...
```

telling the system to use lots of heap space.

A.6 Literate Programming

A literate document contains text and code. From this, RX generates a text document that has code expressions and possibly their values converted to formatted text.

The switch `output` says whether to generate any output at all. Beware that even if this is off, imports, sets, and evaluations may take place.

RX knows whether it is currently looking at text or code by looking up the value of the option `current`. So a typical code block is

```
.begin(current=code)
   ...
.end
```

The physical lines inside such a block are glued together to form logical lines that are fed into the RX evaluator one by one.

A *logical line* consists of one or more *physical lines*. Each physical line that starts with a non-blank character begins a new logical line. Physical lines that start with a blank character are called *continuation lines*. They are appended to the most recent logical line.

Each logical line must be a syntactically valid RX expression.

Apart from code blocks, there may be code snippets contained inside lines of text.

Example:

```
 ... the expression $S Q1 Q0$ denotes ...
```

Such inline code must be a valid expression. It is also sent to the RX evaluator.

There are two variations of inline code. They differ in whether the separator is omitted or kept. Per default,

```
.set (keepsep="$",omitsep="|")
```

and `$x$` will result in `$y$`, while `|x|` will result in `y`, where `y` denotes the result of typesetting `x`.

The default setting should prove reasonable, and in fact RX inline code mode can be seen as a replacement for LaTeX's math mode.

Typesetting customization

RX has two styles for typesetting its output: `plain` and `latex`.

Both can be assigned independently to the options `code` and `text`.

Example:

```
.set (text=latex,code=plain)
```

tells RX to use `\verb` for inline code and the `{verbatim}` environment for code blocks.

You can assign a special LaTeX expansion to RX identifiers. This will be used when `code=latex`. It is achieved by form declarations, of which there are two kinds:

A *passive form* is just a string that gets printed instead of the identifier's name. Example:

```
.begin(current=code,output=off)
form S = "\textbf{S}"
form (+) = "\cup"
.end
```

Note that form declarations happen inside code, not text.

An *active* form is a formatting macro that takes arguments. Example:

```
.begin(current=code,output=off)
form (/) 2 = "\frac#1#2"
.end
```

In the arguments of the operators, parentheses are inserted so that the output is correctly parseable with the given operator precedences and associativities. This remains unaffected when a passive form is used for an operator.

When an active form is in effect, its arguments are typeset for a very low precedence. So, there will be no outer parentheses, because it is assumed that the form takes care of visual grouping. For instance, `4 / (1 + 3)` will come out as `\frac{4}{1 + 3}`, given the above declaration.

(As an aside, this idea of omitting some parentheses was already present in the `smugweb` system [Wal95].)

Typesetting trees

As seen in this thesis, `RX` can be conveniently used to read and typeset trees. Reading is eased by the *implicit apply* convention which inserts `@` nodes. These nodes can be visually typeset by giving `@` an active form. In this paper we use `tree.sty` written by Edward M. Reingold and Nachum Dershowitz which in turn uses PICTEX.

A.7 RX example

Here is some input that is used in the thesis itself.
First, describe the set of all S-terms.

```
all = grammar t {t -> S ++ t  t}
== grammar x1 {x1 -> x1  x1 ++ S}
```

So `all` is a constant whose value is an automaton that accepts (produces) the set of all terms.
The set of all redexes obviously is

```
redex = S  all  all  all
==
grammar
x0
{x0 -> x1  x0 ++ x1  x1 ++ x1  x2 ++ x1  S ++ x1  x4,
 x1 -> x2  x0 ++ x2  x1 ++ x2  x2 ++ x2  S ++ x2  x4,
 x2 -> S  x0 ++ S  x1 ++ S  x2 ++ S  S ++ S  x4,
 x4
 ->
 x0  x0 ++ x0  x1 ++ x0  x2 ++ x0  S ++ x0  x4 ++ x4  x0 ++ x4  x1
 ++ x4  x2 ++ x4  S ++ x4  x4}
```

Now we define a function with one argument.

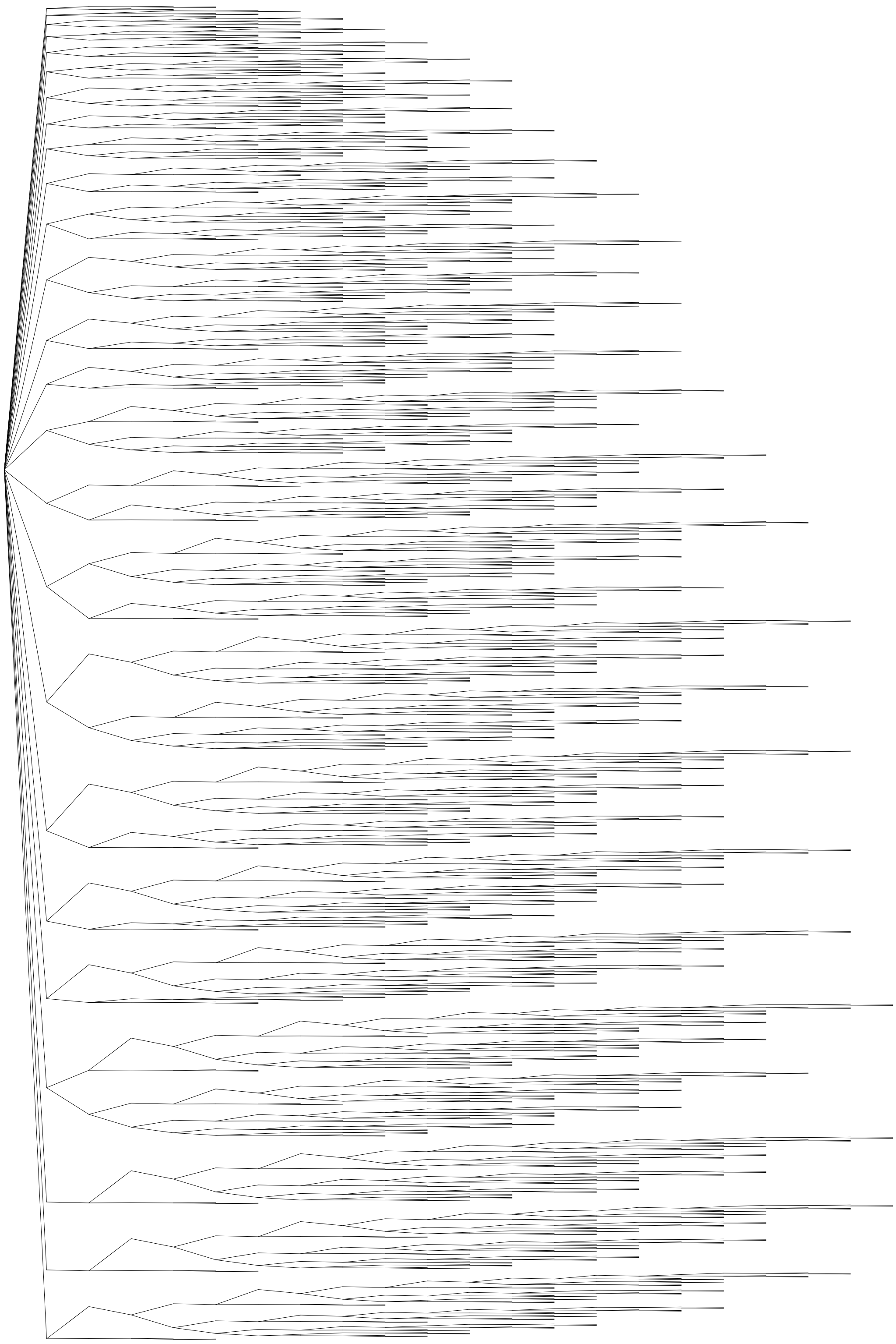
```
contains x = grammar y {y -> x, y -> y  all ++ all  y}
== defined
```

The argument is any language `x`, and the result is the language of all terms `y` that contain `x` as a subterm.
This allows to compute the set of normal forms.

```
normal = all \\ contains redex
==
grammar
(x0 ++ x1 ++ S)
{x0 -> x1  x0 ++ x1  x1 ++ x1  S, x1 -> S  x0 ++ S  x1 ++ S  S}
```

Finally, this could be compared to the representation

```
normal' = grammar t {t -> S, t -> S  t, t -> S  t  t}
==
grammar
(x0 ++ S ++ x2)
{x0 -> S  x0 ++ S  S ++ S  x2, x2 -> x0  x0 ++ x0  S ++ x0  x2}
```



S A S (S A S)
after 50 reductions

Curriculum vitae

	Johannes Waldmann
3. 10. 1967	born in Gera, Germany
1974 - 1984	comprehensive school in Hermsdorf
1984 - 1986	extended secondary school in Stadtroda
1986	Abitur
1986 - 1988	compulsory military service
1988 - 1992	study of mathematics and computer science at Friedrich-Schiller-Universität, Jena
1992	Diplom
1993 - 1997	employment as research assistant at the Institut für Informatik, Friedrich-Schiller-Universität, Jena

Jena, den 21. 10. 1997 Johannes Waldmann

Selbständigkeitserklärung

Ich habe die vorliegende Arbeit selbständig
und nur unter Verwendung der angegebenen
Hilfsmittel und Literatur angefertigt.

Jena, den 21. 10. 1997 Johannes Waldmann