Diplomarbeit

STALKER: Lerntheoretische Modellierung und Untersuchung

Matthias Degen

Darmstadt, den 29.12.2005

Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Knowledge Engineering
Prof. Dr. Johannes Fürnkranz

Betreuer:
Dr. Gunter Grieser

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, Dezember 2005

Matthias Degen

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Motivation und Einführung

Mit der Verbreitung des Internets hat die Menge an Informationen, zu der wir Zugang haben, stetig zugenommen. Es ist jedoch nicht immer einfach, unter der Flut von Informationen gehaltvolles Wissen herauszufiltern. Deshalb ist in den letzten Jahren das Gebiet der *Informationsextraktion (IE)* immer wichtiger geworden. Viele IE-Systeme basieren dabei auf Techniken des *maschinellen Lernens*. In unserer Diplomarbeit wollen wir uns unter lerntheoretischen Gesichtspunkten mit einem Teilgebiet der Informationsextraktion näher beschäftigen: der *Wrapper Induktion*. Bei der Wrapper Induktion werden anhand von Beispielen *Wrapper* gelernt. Wrapper sind Regeln, die von einer Gruppe von IE-Systemen zur Extraktion von Daten benötigt werden. Im ersten Teil unserer Arbeit werden wir ein Modell in Termini formaler Sprachen erstellen, das die *Wrapperklasse* des Wrapper-Induktionssystems *STALKER* sowie die durch die Klasse bestimmte Informationsextraktion beschreibt. Im zweiten Teil werden wir dann untersuchen, wann Wrapper dieses Modells im Sinne der *induktiven Inferenz im Limes* lernbar sind.

Im Folgenden wollen wir kurz in die Informationsextraktion und Lerntheorie einführen und unsere Motivation und Ziele für diese Arbeit erklären. Wir schließen das Kapitel mit dem restlichen Aufbau der Diplomarbeit ab.

1.1 Informationsextraktion

Das Ziel der Informationsextraktion ist es, die Qualität der Informationen zu erhöhen, in dem die Quantität reduziert wird. Dabei werden aus der Vielzahl der Daten relevante Informationen in eine strukturierte Form gebracht und der Rest weggelassen. Es existieren IE-Systeme, die Daten aus verschiedenen Dokumenten mit unterschiedlichen Repräsentationen in ein einheitliches Format transformieren können.

Die Informationsextraktion ist für jede Ansammlung von Dokumenten brauchbar aus der man spezielle Fakten extrahieren möchte. Insbesondere das Internet ist so eine Ansammlung von Dokumenten. Die Textstile und der Aufbau der Dokumente können ganz unterschiedlich ausfallen, angefangen von natürlichen Texten bis zu Dokumenten mit sehr strukturierten Daten wie z.B. Tabellen. Für jede dieser Dokumentarten gibt es IE-Systeme.

Die Aufgabengebiete der Informationsextraktion variieren. So ist die einheitliche Repräsentation für visuelle Darstellung und Gegenüberstellung von Fakten geeignet. Durch die datenbankartige Struktur der Daten können IE-Systeme aber auch als Vorverarbeitung für automatische Datenanalysen genutzt werden (z.B. für Data-Mining).

Das System STALKER, mit dem wir uns in unserer Arbeit beschäftigen, gehört zur Grupper der sogenannten Wrapper-Induktionssysteme. Diese Systeme sind speziell für strukturierte

Dokumente geschaffen worden. In Kapitel 2 werden wir uns genauer mit der Informationsextraktion und ihrer Systeme beschäftigen.

1.2 Algorithmische Lerntheorie

Was ist Lernen? Intuitiv haben die meisten Menschen eine Ahnung davon, was Lernen ist. Versucht man jedoch das Lernen und seine Bedeutung klar zu definieren, dann wird es schwieriger. Eine mögliche Definition von Lernen, die in der Kognitionspsychologie sehr verbreitet ist, ist Folgende (frei nach [And00]):

> *Lernen ist ein Prozess, bei dem aufgrund von Erfahrungen nachhaltige Veränderungen im Verhaltenspotential eintreten.*

Diese Definition hilft uns jedoch nicht bei der *algorithmischen Lerntheorie*. Sie ist nämlich zu unpräzise, wenn wir Einsichten darüber gelangen wollen, wie und was ein Computer lernen kann. Es existieren unterschiedliche Modelle darüber, wie das Computerlernen aussieht. Es gibt jedoch bestimmte Fragen, die man sich unabhängig vom Lernmodell stellen muss, wenn man sich genauer mit dem Lernen beschäftigen will:

- Wer lernt?
- Was wird gelernt?
- Welche Informationen habe ich über das, was ich lerne?
- Wie sehen Hypothesen aus?
- Wann habe ich etwas gelernt?

Je nachdem wie die einzelnen Fragen beantwortet werden, ergeben sich andere Modelle des Lernens. Im Kapitel 4 werden wir näher auf dieses Thema eingehen und verschiedene Lernmodelle vorstellen. Unsere späteren Lernbarkeitsuntersuchungen werden innerhalb des Modells des *induktiven Lernens im Limes* durchgeführt. Dieses Modell stellt dabei den Limescharakter des Lernens in den Vordergrund. Hier gilt nämlich dann etwas als gelernt, wenn eine Hypothese trotz neuerer Informationen beibehalten wird und diese Hypothese genau das zu Lernende beschreibt.

1.3 Zielsetzung

Systeme der Informationsextraktion benötigen Regeln mit deren Hilfe sie relevante Informationen in einem Dokument bestimmen können, um sie später zu extrahieren. Eine passende Regel für ein bestimmtes Extraktionsproblem zu finden, ist oft so schwierig und komplex, dass ein herkömmlicher Benutzer überfordert ist. Meistens übernehmen Experten diese Aufgabe. Die Fähigkeit eines Systems, Extraktionsregeln automatisch, z.B. anhand von Beispielen, zu

lernen, ist deshalb für den Benutzer eine erhebliche Erleichterung. In vielen Arbeiten, die sich mit IE-Systemen, die Regeln automatisch lernen, beschäftigen, liegt der Schwerpunkt der Arbeit auf der Untersuchung von zwei Gegenständen. Zum einen wird die *Ausdruckskraft* (engl. *expressiveness*) eines IE-Systems untersucht: Wie gut kann ein System reale IE-Probleme handhaben und bis zu welchem Grad kann ein System ein anderes System ersetzen?. Zum anderen wird die *Leistungsfähigkeit* (engl. *efficiency*) der IE-Systeme untersucht: Wieviele Beispiele sind nötig, um die Regeln für eine IE-Aufgabe zu lernen und wie lange braucht das System dafür? Hauptsächlich werden Ergebnisse dabei mit Hilfe empirischer Techniken gewonnen. In wenigen Arbeiten wurden zusätzlich noch Untersuchungen über die Lernbarkeit von Regeln hinsichtlich eines theoretischen Lernmodells geführt. Das Ziel dieser Arbeit ist es, ein Modell in Termini formaler Sprachen zu schaffen, das die Wrapperklasse von STALKER und die durch ihr bestimmte Extraktion von Daten beschreibt. Innerhalb des durch das Modell bestimmten formalen Rahmens, wollen wir dann untersuchen, welche Regeln für den Wrapper in Bezug auf das induktive Lernen im Limes gelernt werden können und welche nicht. Wir haben uns für STALKER entschieden, da es sehr ausdrucksstarke Regeln benutzt und deshalb bei vielen Dokumenten eine Extraktion durchführen kann. Unsere theoretische Untersuchung ist mit dem Wunsch verbunden, neue Einsichten über STALKER zu gewinnen, die man allein durch empirische Untersuchungen nicht erlangen kann. Als Vorbild für unsere Arbeit dienen dabei die Arbeiten von Grieser et al. zur Wrapperklasse *Island-Wrapper*([GJLT00, GL01, GJL02]). Diese Arbeiten kann man als Beispiele sehen, wie sich aus formalen lerntheoretischen Untersuchungen wieder neue Ideen für die Praxis eröffnen.

1.4 Die Struktur dieser Arbeit

Der Rest die Diplomarbeit ist wie folgt gegliedert:

Kapitel 2 gibt einen Überblick über das Gebiet der Informationesxtraktion. Dabei unterscheiden wir zwischen Informationsextraktion aus natürlichem Text, Informationesextraktion aus Webseiten und dem Bereich der Wrapper-Induktionssysteme.

In **Kapitel 3** befassen wir uns mit dem Wrapper-Induktionssystem STALKER. Dabei werden wir zuerst die Eigenschaften und Funktionen von STALKER klären und im zweiten Abschnitt des Kapitels ein neues theoretisches Modell einführen, das die Wrapperklasse von STALKER beschreibt.

Kapitel 4 widmet sich der algorithmischen Lerntheorie. Dazu werden wir im ersten Abschnitt einen Abriss über verschiedene theoretische Modelle des Lernens geben. Im zweiten Teil werden wir detailliert auf das Modell des induktiven Lernens im Limes eingehen und die dazu wichtigsten Begriffe definieren.

Kapitel 5 beschäftigt sich damit, wie ein Wrapper unseres Modells gelernt werden kann.

Dazu werden wir das Konzept des Lernens aus markierten Text erklären und die verschiedenenen Lernszenarios bestimmen, die es geben kann.

Kapitel 6 bildet den Hauptteil dieser Arbeit. Hier werden innerhalb des in Kapitel 5 eingeführten formalen Rahmens Untersuchungen hinsichtlich der induktiven Lernbarkeit im Limes durchgeführt.

In **Kapitel 7** werden wir eine Zusammenfassung unserer Arbeit geben, Schlussfolgerungen präsentieren, zu die uns die Untersuchungen geführt haben, und mögliche Ansätze für weitere Arbeiten in diesem Gebiet sowie offene Probleme aufzeigen.

2 Informationsextraktion

Bei der *Informationsextraktion(IE)* geht es um das Extrahieren von gewissen Daten aus einer Menge von Dokumenten. Dabei ist es das Ziel, relevante Informationen aus den Dokumenten in eine datenbankartige Struktur zu bringen und gleichzeitig die irrelevanten Informationen wegzulassen. Ein Beispiel wäre das Finden von allen Meldungen, die in den Tageszeitungen über einen bestimmten Konzern veröffentlicht werden. Ein zweites Beispiel wäre das Einlesen von Preisen eines Verkaufartikels aus verschiedenen Internetseiten in eine Datenbank zum Zwecke des Preisvergleichs. Einen Textbereich, der für die Extraktion relevant ist, nennen wir im Folgenden auch *Attribut*. Eine wichtige Komponente eines IE-Systems ist die Menge der *Extraktionsmuster* (engl. *extraction pattern*) oder auch *Extraktionsregeln* (engl. *extraction rules*), die zur Erkennung der Attribute einer bestimmten Extraktionsaufgabe nötig ist. Bei vielen Systemen werden diese Regeln von Experten per Hand erstellt. Dies kann sehr schwierig und aufwändig sein. Andere Systeme generieren die Extraktionsregeln automatisch anhand von Trainingsbeispielen. Für den letztgenannten Ansatz werden wir im Folgenden die wichtigsten IE-Systeme vorstellen. Wie wir an den Beispielen vom Anfang sehen konnten, gibt es für ganz unterschiedliche Arten von Dokumenten IE-Aufgaben. Wir unterscheiden zwischen Systemen, die aus natürlichen Texten Informationen extrahieren, Systemen, die aus unstrukturierten bzw. semistrukturierten Webseiten Informationen extrahieren und den Wrapper-Induktions-Systemen. Weiterführende Informationen über das Gebiet der Informationsextraktion finden sich in [Eik99], [Mus99] oder in [KT03]. Einen Überblick über die hier vorgestellten Systeme und ihre Eigenschaften gibt Tabelle 1.

2.1 Informationsextraktion aus natürlichen Texten

Systeme, die wir in diesem Abschnitt besprechen, sind dafür entwickelt worden aus rein natürlichsprachigen Texten, Informationen zu extrahieren. Ein Beispiel wäre das Herausfiltern von Neuigkeiten über Naturkatastrophen aus Zeitungen. Für die Extraktionsregeln dieser Systeme werden sowohl das Wissen über syntaktische Verbindungen zwischen Wörtern als auch das Wissen über die Semantik von Wörtern und Sätzen benötigt. Es sind deshalb zur Generierung von Extraktionsregeln verschiedene Vorverarbeitungsschritte nötig, wie *syntaktische Analysen* (engl. *syntactic analyzer*)und *semantische Klassifizierungen* (engl. *semantic tagger*).

AutoSlog [Ril93] war das erste System, das Extraktionsregeln aus Trainingsbeispielen lernte. Es benutzt dabei syntaktische Heuristiken, um aus markierten Beispielen Extraktionsregeln zu generieren. Bei AutoSlog werden diese Regeln *conceptional nodes* (*Konzept-Knoten*) genannt. Jeder AutoSlog-Knoten besteht aus einem *sprachlichem Muster* (engl. *linguistic pattern*), einem *auslösendem Schlüsselwort* (engl. *trigger*) und *semantischen Bedingungen* (engl. *constraints*) für die Komponenten des sprachlichen Musters. Zu extrahierende Informationen müssen entweder Subjekte, direkte Objekte oder Nominalphrasen sein. Eine Extraktion in AutoSlog funktioniert wie folgt: Enthält ein Satz ein Trigger-Wort, dann wird der zugehörige

IE-System	strukt.	semi	frei	Multi-Slot	benötigt Syntax	Nachverarbeitung
AutoSlog			x	nein	ja	ja
AutoSlog-TS			x	nein	ja	ja
CRYSTAL			x	ja	ja	ja
CRYSTAL + Webfoot		x	x	ja	ja	ja
PALKA			x	ja	ja	ja
LIEP			x	nur	nein	nein
HASTEN			x	ja	nein	nein
WHISK	x	x	x	ja	nein*	nein
SRV		x		nein	nein	nein
RAPIER		x		nein	nein	nein
WIEN	x			ja	nein	nein
Island-Wrapper	x			ja	nein	nein
SoftMealy	x	x		ja	nein	nein
STALKER	x	x		ja**	nein	nein

Tabelle 1: Vergleich der IE-Systeme

*Die Spalten **„struk.“**, **„semi“** und **„frei“** geben an, welche IE-Systeme Extraktionen aus strukturierten Dokumenten, aus semi-strukturierten Dokumenten bzw. aus Dokumenten mit natürlichen Texten durchführen können. **„Multi-Slot“** zeigt, ob ein System Multi-Slot-Extraktionen ausführen kann. IE-Systeme, die in der Spalte **„benötigt Syntax“**, ein „ja“ stehen haben, sind entworfen worden, immer den syntaktischen Kontext zu nutzen. Das bedeutet, falls ein zu extrahierendes Attribut in einem Fall ein Subjekt und im anderen Fall eine Nominalphrase ist, dann müssen diese Systeme zwei unterschiedliche Extraktionsregeln erzeugen. Die anderen Systeme können eine Regel generieren, die beide Fälle abdeckt. **„Nachverarbeitung“** gibt an, bei welchen Systemen nach der Extraktion noch eine Nachverarbeitung nötig ist, weil die relevanten Informationen nicht genau bestimmt werden. *****: Wird WHISK zur Extraktion von Daten aus natürlichen Texten angewendet, dann muss es den syntaktischen Zusammenhang benutzen. Bei Extraktionen aus struktierten oder semi-strukturierten Dokumenten ist dies nicht nötig. ******: STALKER generiert nur Single-Slot-Regeln. Durch die Embedded Catalog Beschreibung eines Dokumentes können die Attribute jedoch in Zusammenhang gebracht werden (Siehe Kapitel 3).*

Konzept-Knoten aktiviert. Das dazu passende sprachliche Muster wird mit dem Satz verglichen, um die relevante Information zu extrahieren. Dabei müssen die im Konzept-Knoten enthaltenen Bedingungen erfüllt sein. Das Trigger-Wort ist meistens ein Verb, kann aber auch ein Substantiv sein (wenn die zu extrahierende Information eine Nominalphrase ist). AutoSlog muss für jeden relevanten Textbereich eine neue Regel lernen (engl. *single-slot*). Die Regeln werden nach dem Prinzip *one-shot* gelernt, d.h. bei AutoSlog werden keine induktiven Schritte durchgeführt. Alle Ausgaben werden gleich an den Benutzer geleitet und nicht an den Lernalgorithmus zurückgegeben. AutoSlog bestimmt nur das syntaktische Feld, das die zu extrahierende Information enthält. Es ist also noch ein Nachverarbeitungsschritt nötig, um die genaue Information zu extrahieren. Eine Erweiterung von AutoSlog ist *AutoSlog-TS* [Ril96]. AutoSlog-TS benötigt keine markierten Beispiele, sondern generiert Extraktionsregeln für jede Nominalphrase in der Trainingsmenge, deren Syntax mit einer der syntaktischen Heuristiken übereinstimmt.

PALKA [KM95] lernt Regeln, die vergleichbar mit den Konzept-Knoten von AutoSlog sind (sogenannte *frame-phrasal-pattern structures* oder kurz *FP-structures*). FP-structures enthalten einen *Bedeutungsrahmen* (engl. *meaning frame*) und ein *phrasales Muster* (engl. *phrasal pattern*). Der Bedeutungsrahmen definiert ein zu extrahierendes Element mit den semantischen Bedingungen, die damit verknüpft sind. Das phrasale Muster ist eine geordnete Liste von lexikalischen Einträgen und semantischen Klassen, die aus einer vorher definierten Konzepthierarchie stammen. In der FP-structure wird ein Element des Bedeutungsrahmens mit den Elementen des phrasalen Musters verbunden. Passt bei der Extraktion ein phrasales Muster zu einem Satz, dann wird die FP-structure aktiviert und der korrespondierende Bedeutungsrahmen benutzt, um die Informationen zu extrahieren. Während bei AutoSlog ein Konzept-Knoten nur dann aktiviert wird, wenn ein Wort im Dokument mit einem Trigger-Wort übereinstimmt, kann bei PALKA eine FP-structure sowohl aktiviert werden wenn ein Wort im Dokument mit einem Wort aus der Konzepthierarchie übereinstimmt als auch dann, wenn ein Wort Element einer der semantischen Klassen aus der Konzepthierarchie ist. Hingegen kann bei PALKA die Bedingung der exakten Übereinstimmung eines Wortes nur an Verben gestellt werden.

Ein weiteres IE-System für freie Texte ist *CRYSTAL* [SFAL95]. Wie AutoSlog lernt CRYSTAL Konzept-Knoten anhand von Beispielen, die durch syntactic analyzer und semantic tagger aufbereitet wurden. Dabei generiert es nach dem Prinzip *bottom-up* zuerst eine Regel für jedes positive Trainingsbeispiel, um dann die Regeln weiter zu verallgemeinern, indem sie mit ähnlichen Regeln vereinigt werden. CRYSTAL muss nicht für jeden Textbereich eine neue Regel generieren, sondern erlernt auch Regeln, die es ermöglichen, mehrere in Zusammenhang stehende Textbereiche zu extrahieren (engl. *multi-slot*). Weiterhin erlaubt es, an alle Textteile sowohl semantische Bedingungen als auch die Bedingung zur exakten Übereinstimmung eines Wortes zu stellen. Wie bei AutoSlog werden die zu extrahierenden Informationen nicht genau identifiziert, sondern nur die syntaktischen Felder in denen diese Informationen enthalten sind. CRYSTAL kann man auch für Informationsextraktion aus Internetseiten benutzen. Dazu benötigt das System aber eine weitere Vorverarbeitung der Beispiele (*Webfoot*, siehe [Sod97]).

CRYSTALs Extraktionsregeln sind etwas ausdrucksvoller als die Regeln von PALKA, da bei PALKA die Bedingung der exakten Übereinstimmung eines Wortes nur an Verbstämme gestellt werden kann. Zusätzlich können bei CRYSTAL semantische Bedingungen an alle Phrasen gestellt werden und nicht nur an die zu extrahierenden Textteile wie bei PALKA.

LIEP [Huf95] ist ein System, das ähnliche Heuristiken wie AutoSlog benutzt, um Extraktionsregeln zu generieren. Es lernt im Gegensatz zu AutoSlog nur multi-slot-Regeln. LIEP ist nicht fähig, single-slot-Regeln zu lernen, da es den Kontext eines zu extrahierenden Textteils nur bezüglich seiner syntaktischen Beziehungen zu den anderen Textteilen findet. Die Regeln bestehen aus syntaktischen und semantischen Bedingungen. LIEP benötigt zum Lernen keine markierten Beispiele. Stattdessen werden aus jedem Satz im Beispieltext interessante Einheiten identifiziert und der Benutzer kann bestimmen welche Kombination an Einheiten relevant ist. Alle bisher genannten IE-Systeme identifizieren nur die syntaktischen Felder in denen relevante Information enthalten ist. LIEP dagegen identifiziert die relevanten Textbereiche exakt. Ein weiterer Unterschied zu den anderen Systemen liegt darin, dass LIEP nicht immer den syntaktischen Zusammenhang benutzen muss. Das bedeutet, falls ein zu extrahierendes Attribut in einem Fall ein Subjekt und im anderen Fall ein präpositionale Phrase ist, dann kann LIEP eine Regel generieren, die beide Fälle abdeckt. AutoSlog, PALKA und CRYSTAL müssen dagegen zwei unterschiedliche Extraktionsregeln erzeugen.

Das IE-System *HASTEN* [Kru95] generiert eine Menge von Paaren, die so genannten *Egraphs*. Ein Egraph enthält ein semantisches Label und eine Menge von Einschränkungen, wie semantische Klasse, Verbstamm, Verbform und exakte Wortübereinstimmung. In der Trainingsphase wird für jeden Egraph aus positiven Beispielen ein Gewicht gelernt. In der Extraktionsphase benutzt HASTEN eine Ähnlichkeitsmetrik, um die Egraphs mit den Dokumenten zu vergleichen. Danach werden die gelernten Gewichte benutzt, um einen Endwert für jeden passenden Egraph zu berechnen. Dieser Endwert wird am Ende mit einem vordefinierten Grenzwert verglichen. Wie LIEP kann HASTEN die relevanten Informationen exakt bestimmen und muss den syntaktischen Kontext nicht nutzen.

2.2 Informationsextraktion aus Internetseiten

Die im vorherigen Abschnitt vorgestellten IE-Systeme funktionieren hervorragend bei Extraktionsaufgaben aus natürlichen Texten. Internetseiten bieten jedoch zusätzlich zu natürlichem Text auch Informationen, die in anderer Form präsentiert werden, z.B. in Form von Tabellen oder ungrammatikalischem Text. Zum Extrahieren dieser Informationen eignen sich die IE-Systeme für natürlichen Text nicht. Im Folgenden stellen wir IE-Systeme vor, die speziell für das Extrahieren von Informationen aus Internetseiten entwickelt worden sind. Gemeinsam haben sie, dass ihre Extraktionsregeln sowohl syntaktische und semantische Einschränken nutzen, als auch sogenannte *Begrenzer* (Wörter oder syntaktische Ausdrücke, die vor und hinter den relevanten Texten stehen).

Die Extraktionsregeln des IE-Systems *WHISK* [Sod99] entsprechen einer beschränkten Klasse von regulären Ausdrücken. Diese Ausdrücke können zum einen Beschreibungen des Kontextes sein, um zu bestimmen, ob ein Textteil extrahiert werden soll. Zum anderen können diese Ausdrücke Begrenzer zur Bestimmung der Grenzen eines relevanten Textbereichs sein. Die Extraktionsregeln von WHISK sind damit sehr ausdrucksstark und eignen sich für eine Reihe von Extraktionsaufgaben, angefangen von der Extraktion aus natürlichem Text bis hin zur Extraktion aus sehr strukturiertem Text. Wird WHISK auf strukturierten oder semi-strukturierten Texten angewendet, dann ist keine Vorverarbeitung nötig. Bei natürlichen Texten ist WHISK jedoch auf eine Vorverarbeitung mittels syntactic analyzer und semantic tagger angewiesen. Je nach Struktur des Dokumentes generiert der Lernalgorithmus von WHISK passende Regeln. Z.B. wird der Algorithmus keine kontextabhängige Einschränkungen generieren, wenn eine Begrenzer-basierte Extraktion ausreichend genau ist. WHISK kann sowohl single-slot- als auch multi-slot-Regeln lernen. Wie LIEP und HASTEN kann es den syntaktischen Kontext eines Textbereichs ignorieren.

Beim IE-System *SRV* [Fre98a, Fre98b, Fre98c] wird das IE-Problem als Klassifizierungsproblem angesehen. Alle möglichen Phrasen gelten als Instanzen. Ein Klassifizierer bestimmt für jede Instanz eine Metrik, die die Zuversicht anzeigt, daß die Phrase in das Zielmuster passt. Ursprünglich benutzte SRV einen relationalen Regellerner mit einem ähnlichen Induktionsalgorithmus wie *FOIL* [Qui90]. Die Induktion erfolgt dabei nach dem Prinzip *top-down*: Es wird von der allgemeinsten Regel ausgegangen, die alle positiven und negativen Beispiele abdeckt. Dabei gilt die Annahme der *Weltabgeschlossenheit*, d.b. als negatives Beispiel wird jede Phrase angenommen, die nicht als zu extrahierende Information markiert ist. Mit jedem Induktionsschritt wird die Regel weiter eingeschränkt, bis keine negativen Beispiele durch die Regel abgedeckt werden. Ist eine Regel gut genug, werden alle positiven Beispiele, die mit dieser Regel abgedeckt werden aus der Trainingsmenge entfernt und der Prozess beginnt von vorne. Die Extraktionsmuster basieren dabei auf Bedingungen, wie Länge der Phrase, Zeichentyp, Orthographie, Wortart oder lexikale Bedeutung. Weitere Bedingungen können auf relationale Strukturen der Dokumente basieren. Später wurde SRV mit zwei zusätzlichen Klassifizierern, einem sogenannten *Rote-Learner* und einem *naiven Bayes-Klassifizierer*, getestet. Der Rote-Learner vergleicht dabei einfach die Phrase mit allen korrekten Beispielen, die das System während der Trainingsphase erhalten hat. Der naive Bayes-Klassifizierer berechnet eine geschätzte Wahrscheinlichkeit, dass Teile der Phrase im Zielmuster enthalten sind. SRV lernt im Unterschied zu WHISK nur single-slot Regeln. Die Bedingungen der SRV-Regeln sind jedoch vielfältiger als die Bedingungen der Regeln von WHISK.

RAPIER [CM99] lernt single-slot-Regeln, die sowohl syntaktische Informationen als auch semantische Klasseninformationen benutzen. Der Lernalgorithmus beginnt nach dem Prinzip bottom-up mit der spezifischsten Regel, die ein Lernziel der Beispielmenge abdeckt. Danach werden zufällig Regelpaare ausgewählt und mittels *Beam Search* die beste Generalisierung

zweier Regeln gefunden. Der allgemeinsten Regel werden dann noch Bedingungen hinzugefügt, um die Regel weiter zu verfeinern. Dieser Prozess wird solange wiederholt, bis kein Fortschritt mehr erzielt wird. Die Regeln bestehen dabei aus drei verschiedenen Attributen. Das *pre-filler-pattern* beschreibt den linken Begrenzer, das *post-filler-pattern* den rechten Begrenzer und das *filler-pattern* die Struktur der zu extrahierenden Information. Jeder *filler-pattern* besteht aus *pattern items*, die mit genau einem Wort übereinstimmen und *pattern lists*, die mit 0 bis n Wörtern übereinstimmen. Die Bedingungen der pattern items bzw. lists können Wortübereinstimmung, Wortart oder semantische Klasse sein. Damit gibt es bei den Regeln von RAPIER eine größere Vielfalt an Bedingungen als bei den Regeln von WHISK.

2.3 Wrapper und Wrapper-Induktion

Wrapper-Induktionssysteme ermöglichen das Extrahieren von strukturierten Informationen aus semistrukturierten Dokumenten. Anwendungsgebiete hierfür sind insbesondere das Extrahieren von Daten aus Webshop-Seiten oder Internetseiten, die Tabellen enthalten. Diese Art von Dokumente besitzen syntaktische Ausdrücke für die Interpretation in einem Browser. Die syntaktische Ausdrücke sind dem Benutzer verborgen. Normalerweise interessiert er sich auch nicht dafür, sondern nur für die von dem Dokument bereitgestellten Informationen über ein bestimmtes Thema. Diese Informationen sind aber in den syntakischen Ausdrücken eingebettet. Weiss man, wie diese Informationen eingepackt sind, dann kann man sie auch in einem Dokument lokalisieren und extrahieren. Für jede zu extrahierende Information benötigt ein Wrapper-Induktionssystem eine Regel, die den Anfang der Information und eine Regel, die das Ende der Information im Dokument bestimmt. Besitzt man diese Regeln (kennt man den *Wrapper*), dann kann man die gewünschten Daten extrahieren.

Viele Wrapper, die in der Praxis benutzt werden, sind per Hand geschrieben worden. Die manuelle Bestimmung eines Wrappers kann jedoch sehr aufwändig und fehlerbehaftet sein. Ändern sich die Strukturen innerhalb eines Dokumentes, dann wird der passende Wrapper oft unbrauchbar und es wird nötig, einen neuen zu programmmieren. Besser wäre es deshalb, die Bestimmung eines Wrappers zu automatisieren. Einen Weg, Wrapper zu lernen, ist mittels der so genannten *Wrapper Induktion*. Dabei markiert ein Benutzer in einem Dokument die zu extrahierenden Textteile. Dies ist für den Benutzer einfacher, als den Wrapper komplett selbst zu programmieren. Das Wrapper-Induktionssystem lernt dann mit Hilfe dieser Beispiele den Wrapper automatisch. Es existieren einige Wrapper-Induktionssysteme. Wir werden uns in dieser Arbeit mit dem System *STALKER* beschäftigen. Bevor wir im nächsten Kapitel das System beschreiben, werden wir nachfolgend einen Überblick über einige andere Arbeiten geben, die sich mit dem Thema Wrapperinduktion beschäftigen. Für eine Gegenüberstellung der Wrapper-Induktionssysteme und ihrer Eigenschaften siehe Tabelle 2.

WIEN [Kus97, Kus00] war das erste Wrapper-Induktionssystem. Es generiert Extraktionsregeln, die sich auf Begrenzer beziehen, die direkt vor und hinter der zu extrahierenden In-

formation stehen. Bei WIEN wird davon ausgegangen, dass es eine einzelne Regel gibt, die für alle Dokumente eines Typs gilt. Diese Regel soll das Extrahieren eine Tupels von mehreren Texteilen ermöglichen, die in Beziehung zueinander stehen. WIEN definiert mehrere Wrapper-Klassen. Die einfachste Klasse ist die *LR-Klasse*. In der LR-Klasse werden nur die Begrenzer für den Anfang und die Begrenzer für das Ende der zu extrahierenden Informationen genutzt. Andere Wrapper-Klassen, wie *HLRT*, *OCLR*, *HOCLR* sind Erweiterungen bzw. Verbesserungen der LR-Klasse, die zusätzlich Begrenzer für den Anfang und Ende eines Dokumentes, Begrenzer für den Anfang und das Ende eines Tupels bzw. sowohl die Begrenzer für das Dokument als auch für die Tupel nutzen. WIEN war imstande für 53% der Seiten einer empirischen Studie Regeln der LR-Klasse zu generieren. Für andere Klassen waren es sogar 70% der Seiten. Weiterhin wurde gezeigt, dass alle Klassen PAC-lernbar sind. Ein Nachteil dieses Wrapper-Induktionssystems ist, dass es keine Unterschiede in der Struktur der Dokumente handhaben kann. WIEN findet keine passende Regel, wenn sich die Reihenfolge eines zu extrahierenden Textteils in zwei Beispielen ändert oder wenn sich die syntaktischen Ausdrücke, in denen die Textteile eingepackt sind, in zwei Dokumenten unterscheiden. Weitere Schwierigkeiten hat WIEN bei der Extraktion von Informationen, die ineinander verschachtelt sind. Ansätze zur Lösung dieses Problems sind die Wrapper-Klassen *N-LR* und *N-HLRT*. Ihre Induktionen stellten sich jedoch als unpraktikabel heraus.

Grieser et al. [GJLT00, GL01] knüpften an den Arbeiten von Kushmerick an und führten ein theoretisches Modell der so genannten *Island-Wrapper-Klasse* ein, die eine Erweiterung der LR-Klasse ist. Im Unterschied zu den Wrapper-Klassen von WIEN beziehen sich hier die Extraktionsregeln auf Begrenzer, die nicht fest, sondern Elemente einer Begrenzersprache sind. Die Untersuchungen von Grieser et.al. richten sich auf die theoretischen Eigenschaften und Lernbarkeiten der formalisierten Island-Wrapper-Klasse. Der in diesen Arbeiten geschaffene formale Rahmen war auch das Vorbild für den formalen Rahmen unserer Untersuchungen.

Das Wrapper-Induktionssystem *SoftMealy* [HD98] generiert Regeln, die als nichtdeterministische endliche Automaten ausgedrückt werden. Dabei stellen Zustände die zu extrahierenden Informationen dar und Zustandsübergänge die Regeln, die die Begrenzer zwischen den Informationen bestimmen. Im Gegensatz zu WIEN kann SoftMealy auch fehlende Werte, Textbereiche mit mehreren Werten und verschiedene Permutationen handhaben. Um jedoch Informationen zu extrahieren, die je nach Dokument in unterschiedlicher Reihenfolge vorkommen, benötigt es eine Trainingsmenge, die jede mögliche Ordnung der Informationen enthält.

	WIEN	Island-Wrapper	SoftMealy	STALKER
fehlende Attribute	nein	nein	ja	ja
Permutationen	nein	nein	ja*	ja
Attribute mit mehreren Werten	nein	nein	ja	ja
Hierarchische Strukturen	ja**	nein	nein	ja
disjunktive Begrenzer	nein	ja	ja	ja
nichtexistierende Begrenzer	nein	nein	ja	ja
Begrenzer in Sequenz	nein	nein	nein	ja

Tabelle 2: Vergleich der Wrapper-Systeme

*Die Zeile **„Fehlende Attribute“** gibt an, welche Systeme auch dann einen Tupel von Informationen aus einem Dokument extrahieren können, wenn in diesem Dokument ein oder mehrere Attribute des Tupels fehlen. Alle Systeme, die in der Zeile **„Permutationen“** ein „ja“ stehen haben, können eine Menge von in Zusammenhang stehenden Textteilen unabhängig von der Reihenfolge der Textteile extrahieren. Normalerweise hat jedes Attribut nur einen Wert. Es kann jedoch auch sein, dass ein Attribut in einem Dokument mehrere Werte hat. Als Beispiel sei die Internetseite eines Restaurantführers genannt, die für jedes Restaurant auch die Menge aller Kreditkarten aufzählt, die in dem Restaurant zur Bezahlung benutzt werden können. **„Attribute mit mehreren Werten“** zeigt, welche Systeme die Atttribute mit mehreren Werten extrahieren können. Systeme, die in **„Hierarchische Strukturen“** ein „ja“ stehen haben, können Informationen extrahieren, die ineinander verschachtelt sind. Die Zeile **„Disjunktive Begrenzer“** gibt an, welche Systeme nicht darauf angewiesen sind, dass ein Begrenzer eines Attributes immer gleich ist. Diese Systeme können auch Attribute extrahieren, die in verschiedenen Dokumenten verschiedene Begrenzer haben. **„Nichtexistierende Begrenzer“** zeigt die Systeme, die auch Informationen extrahieren können, für die es keine richtigen Begrenzer gibt. Alle Systeme, die schließlich in der Zeile **„Begrenzer in Sequenz“** ein „ja“ stehen haben, können zur Extraktion auch Regeln nutzen, die pro Attribut mehrere Begrenzer in Sequenz verlangen. *****: Damit SoftMealy auch Attribute in unterschiedlicher Reihenfolge handhaben kann, benötigt es eine Trainingsmenge, die jede mögliche Ordnung der Attribute enthält. ******: Das „ja“ in dieser Zeile bezieht sich auf die Klassen N-LR und N-HLRT.*

3 Das Wrapper-Induktionssystem STALKER

Das folgende Kapitel befasst sich mit dem Wrapper-Induktionssystem *STALKER* ([MMK99]). Im ersten Abschnitt werden wir die Funktion und Eigenschaften von STALKER erklären. Im zweiten Abschnitt werden wir dann versuchen, die Wrapperklasse von STALKER in Termini formaler Sprachen zu modellieren.

3.1 Merkmale von STALKER

STALKER ist ein Wrapper-Induktionssystem, das auf Basis von markierten Beispielen Regeln lernt, die es ermöglichen, Informationen aus einem semistrukturiertem Dokument zu extrahieren. Stalker erlaubt im Gegensatz zu WIEN auch die Extraktion von Informationen aus hierarchisch strukturierten Dokumenten. Um dieses Problem zu bewältigen, wurde von Muslea et al. der *Embedded Catalog (EC)* Formalismus entworfen, mit dem der Aufbau von Dokumenten beschrieben werden kann. Eine EC Beschreibung eines Dokumentes entspricht dabei einem Baum, dessen Blätter die zu extrahierenden Informationen darstellen. Die internen Knoten eines EC-Baumes repräsentieren Listen mit n-stelligen Tupeln, wobei jedes Element der Liste entweder ein Blatt oder wieder eine Liste sein kann. Zur Veranschaulichung ist auf der nächsten Seite eine Internetseite eines Restaurantführers für Los Angeles und ein möglicher EC-Baum abgebildet.

Eine Regel, die für das gesamte Dokument und alle zu extrahierenden Informationen anwendbar ist, ist im Falle eines hierarchisch strukturierten Dokumentes nicht geeignet. Bei STALKER wird deshalb das Problem, eine Extraktionsregel für das gesamte Dokument zu finden, in Teilprobleme aufgeteilt. Für jeden zu extrahierenden Textbereich wird eine eigene Regel gesucht, die unabhängig von den anderen angewendet werden kann. Den Zusammenhang zwischen den einzelnen Regeln stellt der EC-Baum her. D.h. für jeden *Knoten* im EC-Baum wird eine *Extraktionsregel* gesucht, die den Knoten von dem Vater extrahiert und für jeden *Listenknoten* benötigt der Wrapper zusätzlich eine *Listeniterationsregel*, die die Liste in individuelle Tupel aufteilt. Sind der EC-Baum und die Regeln vorhanden, dann kann jeder Textteil extrahiert werden, in dem man den Baum von der Wurzel bis zu dem entsprechenden Knoten entlang geht und nacheinander die passende Extraktionsregel anwendet, um einen Knoten von seinem Vater zu extrahieren. Ist der Vater eines Knotens eine Liste, dann wird erst die Iterationsregel der Liste angewendet und danach die Extraktionsregel des Knotens auf jedes Tupel dieser Liste. Der Inhalt der Wurzel ist das gesamte Dokument, während der Inhalt der Söhne jeweils eine Teilsequenz des Dokumentes ist.

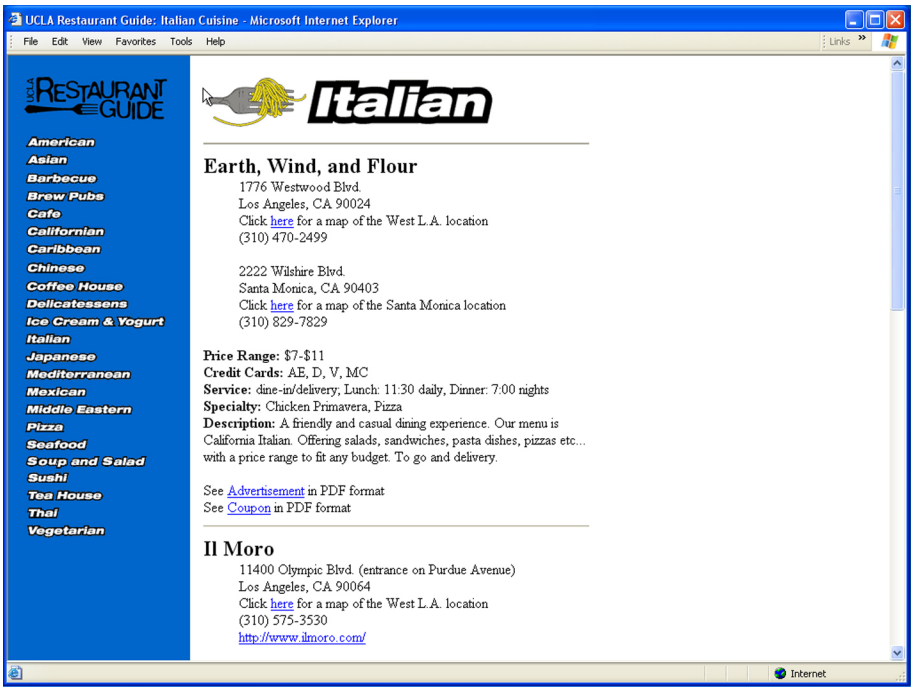


Abbildung 1: Internetseite UCLA Restaurantguide

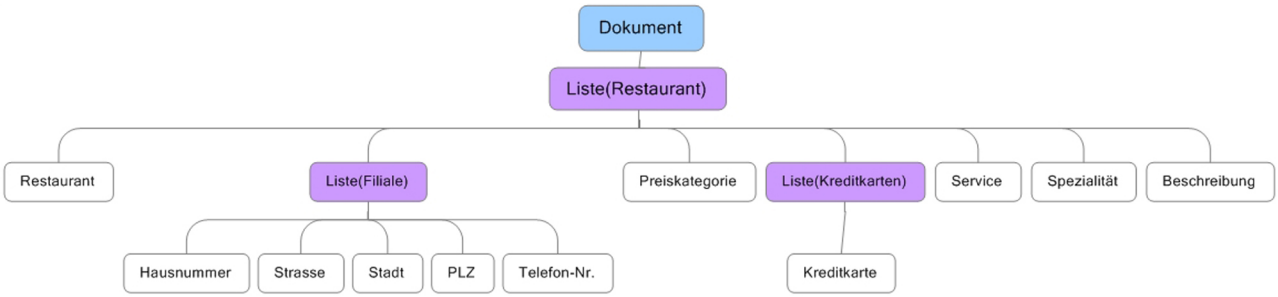
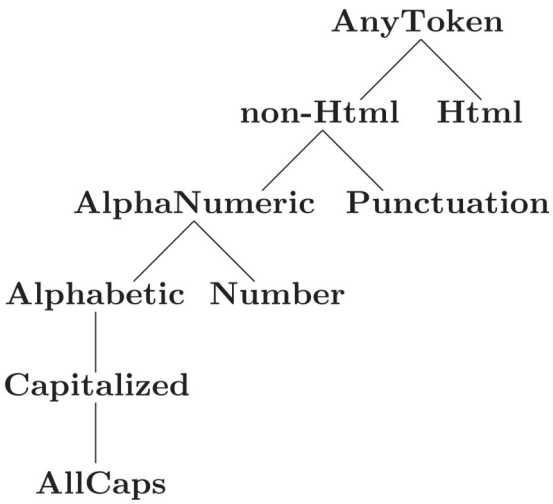


Abbildung 2: ECT für die Internetseite UCLA Restaurantguide



Aus [Mus02]. Die Wurzel AnyToken, die Menge aller Token, ist die allgemeinste Wildcard. Die Nachfahren sind Teilmengen von AnyToken. Html ist z.B. die Menge aller Token, die HTML-Befehle sind.

Abbildung 3: Hierarchie der Wildcards

Die Extraktions- und Iterationsregeln bei STALKER können folgende Formen haben:

- $SkipTo(LM_1)...SkipTo(LM_n)$
- $SkipTo(LM_1)...SkipTo(LM_n)SkipUntil(LM_{n+1})$
- $SkipTo(LM_1)...SkipTo(LM_n)Next(LM_{n+1})$

SkipTo(*LM*) bedeutet dabei 'Ignoriere alle Zeichen, bis Du *LM* findest und stoppe dahinter. *SkipUntil*(*LM*) bedeutet 'Ignoriere alle Zeichen, bis Du *LM* findest und stoppe davor'. *Next*(*LM*) bedeutet 'Gehe nicht weiter und teste, ob das nächste Wort *LM* ist'. Die Argumente von *SkipTo*(), *SkipUntil*() und *Next*() sind so genannte *Landmarks*. Landmarks sind Aneinanderreihungen von *Tokens* und *Wildcards*. Tokens können Wörter, Zahlen, HTML-Befehle oder Satzzeichen sein. Wildcards repräsentieren Mengen von Tokens, wie z.B. '*Html*' für die Menge aller HTML-Befehle und '*Number*' für die Menge aller Zahlen (siehe Abbildung 3 für die komplette Hierarchie der Wildcards). Um einen Text zu extrahieren benötigt man eine Teilregel, die den Anfang des Textes bestimmt, und eine Teilregel für das Ende. Diese Regeln werden *Anfangsregel* und *Endregel* genannt. Die Teilregeln können entweder *Vorwärtsregeln* oder *Rückwärtsregeln* sein. Sind Anfangsregel und Endregel Vorwärtsregeln, dann wird das Dokument von vorne nach hinten durchsucht. Zuerst wird der Anfang des zu extrahierenden Textes gesucht und ab dieser Position das Ende. Sind beide Regeln Rückwärtsregeln, dann wird von hinten nach vorne zuerst das Ende des Textes und dann der Anfang gesucht. Ist die Anfangsregel eine Vorwärtsregel und die Endregel eine Rückwärtsregel, dann wird nach dem

```
[...] <hr noshade> </td> </tr> <tr>
<td> <b> Earth, Wind, and Flour </b>
<dd> 1776 Westwood Blvd.
<dd> Los Angeles, CA 90024 <br>
<dd> (310) 470-2499
<p> </p>
<dd> 2222 Wilshire Blvd.
<dd> Santa Monica, CA 90403 <br>
<dd> (310) 829-7829
<p> <b> Price Range: </b> $7-$11 <br>
<b> Credit Cards: </b> AE, D, V, MC <br>
<b> Service: </b> dine-in/delivery; Lunch: 11:30 daily, Dinner: 7:00 nights <br>
<b> Specialty: </b> Chicken Primavera, Pizza <br>
<b> Description: </b> A friendly and casual dining experience. Our menu is
California Italian. Offering salads, sandwiches, pasta dishes, pizzas etc... with a
price range to fit any budget. To go and delivery.
<p> </p> </dd> </td> </tr> <tr> <td> <hr> [...] <hr noshade> [...]
```

Abbildung 4: vereinfachter Auszug eines HTML-Codes der UCLA Restaurantguide Seite

Beginn des Textes von vorne nach hinten gesucht und nach dem Ende von hinten nach vorne. Als Beispiel für die Benutzung von Extraktionsregeln betrachten wir in Abbildung 4 einen vereinfachten Auszug eines HTML-Codes der UCLA Restaurantguide Homepage. Wir nehmen dabei an, dass sowohl alle Anfangsregeln als auch alle Endregeln Vorwärtsregeln sind. Um nun von allen Restaurants einer UCLA Restaurantguide Seite die Preiskategorie zu extrahieren, benötigen wir mehrere Regeln. Im Folgenden zeigen wir eine Möglichkeit, wie die Regeln aussehen können. Zuerst extrahiert STALKER die Liste aller Restaurants mit der Extraktionsregel für *Liste(Restaurant)*:

Anfangsregel: *SkipTo*(<hr noshade>)
('Ignoriere alle Zeichen bis Du das Token '<hr noshade>' findest und stoppe dahinter')

Endregel: *SkipTo*(<hr noshade>)
('Ignoriere alle Zeichen bis Du das Token '<hr noshade>' findest und stoppe dahinter')

Danach wird auf dieser Liste wiederholt die Iterationsregel für *Liste(Restaurant)* angewendet. Die syntaktischen Ausdrücke, die das Ende eines Restaurants begrenzen, sind nicht immer gleich. Normalerweise steht nach den Informationen eines Restaurants eine Linie ('<hr>'). Das letzte Restaurant der Seite wird dagegen durch eine Linie ohne Schattierung begrenzt ('<hr noshade>'). Die bisherigen Formen einer Regel sind dafür nicht ausdrucksstark genug. Regeln in STALKER können aber noch zusätzlich Disjunktionen besitzen, um mit Unterschieden im Format eines Dokumentes zurecht zu kommen.

- *either Regel 1 or Regel 2*

Das obere Beispiel bedeutet 'Entweder wende Regel 1 an oder Regel 2'. Zu beachten ist, dass disjunktive Regeln geordnete Listen sind. Lassen sich mehrere Regeln gleichzeitig anwenden, dann wird immer die in der Liste als erstes vorkommende Regel angewendet. Ein mögliche Iterationsregel für *Liste(Restaurant)* ist dann Folgende:

Anfangsregel: *SkipUntil*(<td> *Capitalized*)
('Ignoriere alle Zeichen, bis Du die Tokens '<td> ' gefolgt von einem Wort mit großem Anfangsbuchstaben findest und stoppe davor')

Endregel: *either SkipUntil*(<hr>) *or SkipUntil*(<hr noshape>)
('Entweder ignoriere alle Zeichen, bis Du das Token '<hr>' findest und stoppe davor oder ignoriere alle Zeichen, bis Du das Token '<hr noshape>' findest und stoppe davor')

Zum Schluss muss auf jedes extrahierte *Restaurant* die Extraktionsregel für *Preiskategorie* angewendet werden:

Anfangsregel: *SkipTo*(Price Range)*SkipTo*()
('Ignoriere alle Zeichen, bis Du das Token 'Price Range' findest und stoppe dahinter. Ignoriere danach alle Zeichen, bis du das Token '' findest und stoppe dahinter')

Endregel: *SkipUntil*(
)
('Ignoriere alle Zeichen bis Du das Token '
' findest und stoppe davor')

Da die Regeln unabhängig voneinander sind, können auch dann Attribute extrahiert werden, wenn andere Attribute in einem Dokument fehlen. Auch die Reihenfolge der Attribute ist nicht wichtig. Mit Hilfe des EC-Baumes können einzelne Attribute in Verbindung gebracht werden und auch verschachtelte Informationen extrahiert werden. Dank den Operationen *SkipUntil* und *Next* können außerdem Informationen extrahiert werden, die nicht durch syntaktische Ausdrücke begrenzt sind. Welche Vorteile STALKER gegenüber anderen Wrapper-Induktionssystemen hat, sieht man in Tabelle 2.

3.2 Modellierung von STALKER in Termini formaler Sprachen

In diesem Abschnitt versuchen wir, ein theoretisches Modell zu erstellen, das in Termini formaler Sprache beschreibt, wie man mit Hilfe des Wrapper STALKER Informationen extrahieren kann. Es ist anzumerken, dass dieses Modell nicht exakt das Wrapper-Induktionssystem STALKER ist, sondern eine Abstraktion davon. Mit Hilfe dieser Abstraktion versuchen wir neue Einsichten über STALKER zu gewinnen.

3.2.1 Benutzte Begriffe

Zuerst werden wir einige Begriffe definieren, die wir später in diesem Abschnitt nutzen werden, die aber auch für alle anderen Kapitel dieser Arbeit gelten. Wir nehmen eine gewisse Vertrautheit mit der Theorie der Formalen Sprachen an (siehe dazu z.B. [HMU02]).

Sei Σ ein endliches Alphabet. Mit Σ^* bezeichnen wir die Menge aller Wörter über Σ. $\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$, wobei ε das leere Wort ist. Jede Teilmenge $L \subseteq \Sigma^*$ ist eine Sprache.

Sei $x \in \Sigma^*$. x^0 gleicht dem leeren Wort und es gilt $x^1 = x$ sowie $x^{k+1} = xx^k$ für alle $k \geq 1$. Die Länge eines Wortes $|.|$ ist definiert als die Anzahl seiner Buchstaben, d.h. sei $x \in \Sigma^*$ mit $x = a_1a_2...a_n$ und $a_i \in \Sigma$ für alle $1 \leq i \leq n$, dann gilt $|x| = n$. Angenommen, x und y seien Wörter. Dann steht xy für die Konkatenation von x und y. Genauer gesagt, sei $x = a_1a_2...a_i$ und $y = b_1b_2...b_j$, dann ist xy das Wort der Länge $i + j$: $xy = a_1a_2...a_ib_1b_2...b_j$. Für alle $L, L' \subseteq \Sigma^*$ sei $L \circ L' = \{wv \mid w \in L, v \in L'\}$.

$w' \sqsubseteq_p w$ bedeutet, dass w' ein Präfix von w ist und $w' \sqsubset_p w$, dass w' ein echter Präfix von w ist. Dabei gilt nach Vereinbarung $\varepsilon \sqsubseteq_p w$ für alle $w \in \Sigma^*$ und $\varepsilon \sqsubset_p w'$ für alle $w' \in \Sigma^+$. $w' \sqsubseteq_s w$ schreiben wir, wenn w' ein Suffix von w ist und $w' \sqsubset_s w$, wenn w' ein echter Suffix von w ist.

$card(L)$ bezeichnet die Kardinalität der Sprache L. Für alle $k \in \mathbb{N}$ ist $\mathcal{L}_k$ die Familie aller nichleeren Sprachen $L \subseteq \Sigma^+$ mit einer begrenzten Kardinalität nicht größer als k (d.h. $card(L) \leq k$). $\mathcal{L}_f$ ist die Familie aller nichtleeren Sprachen $L \subseteq \Sigma^+$ mit einer endlichen Kardinalität (d.h. $card(L) < \infty$). Weiterhin seien $max\{A\}$ und $min\{A\}$ das Maximum und Minimum einer Menge $A \subseteq \mathbb{N}$.

3.2.2 Definition der Regeln

Nachdem wir die wichtigsten Begriffe geklärt haben, führen wir jetzt das Modell ein, das die Wrapperklasse von STALKER und in Termini formaler Sprachen beschreibt. Wie wir schon im ersten Teil dieses Kapitels erläutert haben, sind Wrapper durch die Regeln definiert, die man benötigt, um die relevanten Informationen zu extrahieren. Hat das Induktionssystem die Regeln gelernt, hat es auch den Wrapper gelernt.

Deshalb ist es für die Darstellung der Wrapperklasse von STALKER ausreichend, wenn wir in Termini formaler Sprachen die Extraktionsregel und Iterationsregel definieren sowie beschreiben, was passiert, wenn die Regeln bei einem Dokument oder Teil eines Dokumentes angewendet werden.

Extraktionsregeln

Eine Extraktionsregel besteht aus einer Teilregel (*Anfangsregel*), die den Beginn eines zu extrahierenden Textes in einem Dokument bestimmt und einer Teilregel (*Endregel*), die das Ende des Textes in dem Dokument bestimmt. Für unser Modell gehen wir davon aus, dass beide Teilregeln Vorwärtsregeln sind. Wie wir wissen, kann eine Teilregel auch eine disjunktive Regel sein. Dabei ist jedes Disjunkt selbst wieder eine Regel. Deshalb sehen wir in unserem Modell die Anfangsregeln und Endregeln als geordnete Listen von Regeln. Wenn wir im Weiteren von einer Regel sprechen, dann meinen wir immer ein Element dieser Listen. Andere Arten von Regeln werden mit dem vollständigen Namen geschrieben, wie z.B. Extraktionsregel, Anfangsregel oder Endregel. Eine Regel wird durch mehrere Operationen bestimmt, die bei der Anwendung hintereinander ausgeführt werden. Nach einer *SkipUntil*- oder *Next*-Operation folgt keine weitere Operation. Die *Landmarks*, die Argumente der Operationen, bestehen bei STALKER aus Aneinanderreihungen von Tokens oder Wildcards (siehe vorherigen Abschnitt). In unserem Modell sind sie Sprachen von Argumenten (sogenannte *Argumentensprachen*).

All diese Annahmen gehen den folgenden Definitionen voraus. Die Regeln unterscheiden sich somit nur durch die Anzahl der $SkipTo$-Operationen, durch die Art der letzten Operation und die Argumentensprachen. Seien $n \geq 1$ und $L_1, ..., L_n$ Argumentensprachen. Dann ist $R = (L_1, ..., L_n, op)$ mit $op \in \{ST, SU, NT\}$ eine Regel. op ist dabei die Art der letzten Operation. Bei $op = ST$ wird eine *SkipTo*-Operation ausgeführt, bei $op = SU$ eine *SkipUntil*-Operation und bei $op = NT$ eine *Next*-Operation.

Wir werden nun die Abbildung $Anfang_{AR}$ von einem Dokument d auf einen Teil des Dokumentes d' und die Abbildung $Ende_{ER}$ von einem Teil des Dokumentes d' auf einen Textabschnitt v einführen. $Anfang_{AR}(d)$ spezifiziert, wie mit Hilfe der Anfangsregel AR der Text vor dem Beginn einer zu extrahierenden Information abgeschnitten wird. d' entspricht dabei dem Teil des Dokumentes ab dem Beginn der zu extrahierenden Information. $Ende_{ER}(d)$ bestimmt, wie mit Hilfe der Endregel ER der Text nach dem Ende einer zu extrahierenden Information abgeschnitten wird. v entspricht dabei dem Teil des Dokumentes bis zum Ende der zu extrahierenden Information. Da Anfangs- und Endregel geordnete Listen von Regeln sind, werden wir zuerst die durch die Regel R bestimmte Abbildung $zerlege_R$ von einem Dokument d auf zwei Textabschnitte (v_1, v_2) definieren. R kann sowohl aus der Liste einer Anfangsregel sein als auch aus der Liste einer Endregel. $zerlege_R(d)$ legt fest, wie mit Hilfe von R ein Text in zwei Teile geteilt wird. Die Bedingungen der Definition bestimmen, wie die Argumente bei

einer Extraktion in Abhängigkeit von den Operationen der Regel benutzt werden. Wir werden zuerst die Definition angeben und dann die einzelnen Bedingungen erklären.

Definition 1 *Seien $n \geq 1$ und $L_1, ..., L_n$ Argumentensprachen. Sei $R = (L_1, ..., L_n, op)$ die korrespondierende Regel. Dann definiert diese Regel folgende Abbildung $zerlege_R$. Sei irgendein Dokument d gegeben, dann ist $zerlege_R(d) = (v_1, v_2)$, mit $v_1, v_2 \in \Sigma^*$, wenn $r_1 \in \Sigma^*, ..., r_n \in \Sigma^*$, $l_1 \in L_1, ..., l_n \in L_n$ existieren, so dass folgende Bedingungen erfüllt sind:*

1. $d = v_1 v_2$
2. $v_1 = r_1 l_1 ... r_n l_n$, *falls* $op = ST$
3. $v_1 = r_1 l_1 ... r_n$, *falls* $op = SU$
4. $v_1 = r_1 l_1 ... r_{n-1} l_{n-1}$, *falls* $op = NT$
5. $l_n \sqsubset_p v_2$, *falls* $op \neq ST$
6. *für alle* $i \in \{1, ..., n-1\} : \neg\exists l' \in L_i$ *mit* $l' \sqsubset_p l_i$
7. $\neg\exists l' \in L_n$ *mit* $l' \sqsubset_p l_n$, *falls* $op = ST$
8. *für alle* $i \in \{1, ..., n-1\} : \neg\exists r' \in \Sigma^*, l' \in L_i$ *mit* $r' \sqsubset_p r_i \wedge r_1 l_1 ... r_{i-1} l_{i-1} r' l' \sqsubseteq_p v_1 v_2$
9. $\neg\exists r' \in \Sigma^*, l' \in L_n$ *mit* $r' \sqsubset_p r_n \wedge r_1 l_1 ... r_{n-1} l_{n-1} r' l' \sqsubseteq_p v_1 v_2$, *falls* $op \neq NT$

△

Bedingung (1.) beschreibt, dass mit Hilfe der Regel das Dokument in zwei Teile aufgeteilt wird. Was passiert, wenn n *SkipTo*-Operationen hintereinander ausgeführt werden? Es wird ein Präfix des Dokumentes übersprungen bis ein Element aus der Argumentensprache der ersten *SkipTo*-Operation gefunden wird und nach diesem Element gehalten. Von dem Haltepunkt aus wird die nächste *SkipTo*-Operation durchgeführt, usw. Bedingung (2.) drückt dies in Termini formaler Sprachen aus. r_i ist dabei der Textteil, der übersprungen wird und l_i ein Element der Argumentensprache der *SkipTo*-Operation. Bei einer *SkipTo*-Operation wird das Dokument solange durchsucht bis das erste passende Element der Argumentensprache gefunden ist. Bedingung (8.) und Bedingung (9.) gewährleisten, dass wirklich nach dem ersten gefundenen Element gehalten wird. Bedingung (6.) und Bedingung (7.) stellen sicher, dass nach einem Element gehalten wird, das keinen Präfix besitzt, der auch Element der Argumentensprache ist. Ist die letzte Operation ein *SkipUntil*, dann ist der Haltepunkt vor dem gefundenen Element der Argumentensprache. Dies drückt Bedingung (3.) aus. Da vor dem Element gehalten wird, ist es nicht wichtig, ob das Element einen Präfix besitzt, der auch Element der Argumentensprache ist. Deshalb gilt hier Bedingung (7.) nicht. Ist die letzte Operation eine *Next*-Operation, wird nach dem letzten *SkipTo* gehalten und überprüft, ob das nachfolgende Wort ein Element der Argumentensprache ist. Ausgedrückt wird dies durch Bedingung (4.).

Da die *Next*-Operation keinen Text überspringt und vor dem Element der Argumentensprache hält, gelten für diese Operation Bedingung (7.) und Bedingung (9.) nicht. Ist die letzte Operation ein *SkipUntil* oder *Next*, dann wird vor dem passenden Argument der Operation gehalten und das Dokument am Haltepunkt in zwei Teile geteilt. Somit ist das Argument ein Präfix vom zweiten Teil. Bedingung (5.) drückt dies in Termini formaler Sprachen aus.

Nun können wir $Anfang_{AR}$ definieren:

Definition 2 *Seien $n \geq 1$ und $R_1, ..., R_n$ Regeln und sei $AR = (R_1, ..., R_n)$ die korrespondierende Anfangsregel. Dann definiert AR folgende Abbildung $Anfang_{AR}$. Sei irgendein Dokument d gegeben, dann ist $Anfang_{AR}(d) = v_2$, mit $v_2 \in \Sigma^+$, wenn ein $v_1 \in \Sigma^*$ und ein $i \in \{1, ..., n\}$ existiert, so dass folgende Bedingungen erfüllt sind:*

1. $(v_1, v_2) = zerlege_{R_i}(d)$
2. $\neg\exists i' \in \{1, ..., i-1\} \wedge v_1' \in \Sigma^* \wedge v_2' \in \Sigma^+$ *mit* $zerlege_{R_{i'}}(d) = (v_1', v_2')$

△

Bedingung (1.) besagt, dass mit Hilfe einer Regel aus der Liste von AR das Dokument in zwei Teile geteilt wird. Da durch die Anfangsregel der Beginn eines zu extrahierenden Textes bestimmt wird, ist $Anfang_{AR}$ eine Abbildung vom Dokument d auf v_2, den zweiten Teil des Dokumentes. Weil eine Anfangsregel eine geordnete Liste von Regeln ist, wird immer die erste Regel in der Liste, die auf ein Dokument anwendbar ist, angewendet. Bedingung (2.) drückt dies in Termini formaler Sprachen aus.

Als nächstes definieren die Abbildung $Ende_{ER}$.

Definition 3 *Seien $n \geq 1$ und $R_1, ..., R_n$ Regeln und sei $ER = (R_1, ..., R_n)$ die korrespondierende Endregel. Dann definiert ER folgende Abbildung $Ende_{ER}$. Sei irgendein Dokument d gegeben, dann ist $Ende_{ER}(d) = v_1$, mit $v_1 \in \Sigma^+$, wenn ein $v_2 \in \Sigma^*$ und ein $i \in \{1, ..., n\}$ existiert, so dass folgende Bedingungen erfüllt sind:*

1. $(v_1, v_2) = zerlege_{R_i}(d)$
2. $\neg\exists i' \in \{1, ..., i-1\} \wedge v_1' \in \Sigma^* \wedge v_2' \in \Sigma^+$ *mit* $zerlege_{R_{i'}}(d) = (v_1', v_2')$

△

$Ende_R$ unterscheidet sich von $Anfang_R$ nur in der Ausgabe. Die Bedingungen sind dieselben. Weil mit Hilfe der Endregel alle Daten nach dem Ende einer zu extrahierenden Information abgeschnitten werden, ist $Ende_{ER}$ eine Abbildung von Dokument d auf v_1, den ersten Teil des Dokumentes.

Mit Hilfe von $Anfang_{AR}$ und $Ende_{ER}$ bestimmen wir jetzt die Abbildung $extract_{EX}$. $extract_{EX}$ beschreibt die Extraktion von Informationen aus einem Dokument d mit Hilfe der Extraktionsregel EX. EX besteht dabei aus den Teilregeln AR und ER.

Definition 4 *Seien AR eine Anfangsregel, ER eine Endregel und $EX = (AR, ER)$ die korrespondierende Extraktionsregel. Dann definiert die EX folgende Abbildung $extract_{EX}$. Sei irgendein Dokument d gegeben, dann gilt für $extract_{EX}(d)$:*

$$extract_{EX}(d) = Ende_{ER}(Anfang_{AR}(d))$$

△

Aus der Definition von $extract_{EX}$ läßt sich folgern, dass die Teilregeln der Extraktionsregel Vorwärtsregeln sind. So wird bei der Extraktion erst die Anfangsregel auf das Dokument angewendet und dann auf dieses Ergebnis die Endregel.

Iterationsregeln

Der Definition der Iterationsregeln gehen dieselben Annahmen wie der Definition der Extraktionsregeln voraus. Mit Hilfe einer Iterationsregel wird eine Liste in ihre individuellen Elemente aufgeteilt. Dabei wird die Iterationsregel nach der Extraktion eines Elementes aus der Liste, und das ist der einzige Unterschied zur Extraktionsregel, nochmal auf den Rest der Liste angewendet. Auch die Iterationsregel besteht aus Anfangs- und Endregel. Zuerst werden wir die Abbildung $EndeIt_{ER}$ von einem Dokument d auf ein Tupel (v_1, v_2) definieren. $EndeIt_{ER}$ bestimmt, wie mit Hilfe der Endregel ER der Text nach dem Ende eines zu extrahierenden Elementes abgeschnitten wird. v_1 entspricht dabei dem Teil des Dokumentes bis zum Ende der zu extrahierenden Information und v_2 dem Rest.

Definition 5 *Seien $n \geq 1$ und $R_1, ..., R_n$ Regeln und sei $ER = (R_1, ..., R_n)$ die korrespondierende Endregel. Dann definiert ER folgende Abbildung $EndeIt_{ER}$. Sei irgendein Dokument d gegeben, dann ist $EndeIt_{ER}(d) = (v_1, v_2)$, mit $v_1 \in \Sigma^+$ und $v_2 \in \Sigma^*$, wenn ein $i \in \{1, ..., n\}$ existiert, so dass folgende Bedingungen erfüllt sind:*

1. $(v_1, v_2) = zerlege_{R_i}(d)$
2. $\neg\exists i' \in \{1, ..., i-1\} \wedge v_1' \in \Sigma^* \wedge v_2' \in \Sigma^+$ *mit* $zerlege_{R_{i'}}(d) = (v_1', v_2')$

△

Wir werden nun mit Hilfe von $Anfang_{AR}$ und $EndeIt_{ER}$ auf der nächsten Seite die Abbildung $rest_{IR}$ von einem Dokument d auf einen Textteil v_2 definieren. $rest_{IR}(d)$ bestimmt den Rest eines Dokumentes, der übrigbleibt, wenn mittels Iterationsregel ein Element aus einer Liste extrahiert wird. Der Rest ist dabei das Dokument ab dem Ende des extrahierten Elementes.

Definition 6 *Seien AR eine Anfangsregel, ER eine Endregel und $IR = (AR, ER)$ die korrespondierende Iterationsregel. Dann definiert IR folgende Abbildung $rest_{IR}$. Sei irgendein Dokument d gegeben, dann gilt für $rest_{IR}(d)$ Folgendes:*

$$rest_{IR}(d) = \{v_2 \mid \exists v_1 \in \Sigma^+, \textit{ so dass } (v_1, v_2) = EndeIt_{ER}(Anfang_{AR}(d))\}$$

△

Man kann die Iterationsregel als eine Extraktionsregel sehen, die nach einer Extraktion nochmal auf den übriggebliebenen Rest angewendet wird. Deshalb definieren wir jetzt mit Hilfe von $extract_{IR}$ und $rest_{IR}$ die Abbildung $iterate_{IR}$, die bestimmt, wie mittels der Iterationsregel IR die individuellen Elemente einer Liste von der Liste getrennt werden.

Definition 7 *Seien AR eine Anfangsregel, ER eine Endregel und $IR = (AR, ER)$ die korrespondierende Iterationsregel. Dann definiert IR folgende Abbildung $iterate_{IR}$. Sei irgendein Dokument d gegeben, dann gilt für $iterate_{IR}(d)$ Folgendes:*

$$iterate_{IR}(d) = \{v \mid v = extract_{IR}(d)\} \cup iterate_{IR}(rest_{IR}(d))$$

△

$iterate_{IR}$ ist eine Abbildung von einer Liste auf die Menge aller Elemente in der Liste, die durch die Iterationsregel extrahiert werden können. Dabei ist $iterate_{IR}$ rekursiv aufgebaut. In der Ergebnismenge ist das erste Element der Liste, das durch $extract_{IR}(d)$ bestimmt wird, und die Ergebnisse von $iterate_{IR}$ angewendet auf $rest_{IR}$ enthalten.

4 Algorithmische Lerntheorie

Im folgenden Kapitel beschäftigen wir uns mit der algorithmischen Lerntheorie. Dazu werden wir erst einen Überblick über wichtige theoretische Modelle des Lernens geben. Im zweiten Teil des Kapitels gehen wir dann genauer auf das *Modell des induktiven Lernens im Limes* von Gold ein.

4.1 Einführung und Überblick

Es gibt viele unterschiedliche Ansätze, Lernen genau zu definieren. Es lassen sich jedoch mehrere Parameter angeben, die in jedem Prozess des Lernens enthalten sein müssen. Jantke und Lange [JL89] haben die folgenden Parameter herausgearbeitet. Zuerst ist zu bestimmen, wie die Objekte aussehen, die gelernt werden sollen. Es existiert kein Verfahren, das beliebige Objekte lernen kann. Man benötigt also eine Menge von möglichen *Lernzielen*. Um aus dieser Menge das exakte Lernziel zu lernen, sind Informationen über dieses Objekt nötig. Da es viele Möglichkeiten gibt, wie man Informationen präsentieren kann, ist es wichtig die Menge der erhältlichen Informationen zu präzisieren. Zusätzlich zur Syntax des *Informationsangebotes* benötigt man die Semantik, um die präsentierte Information mit dem, was gelernt werden soll, in Beziehung zu setzen. Für den Prozess des Lernens ist neben der Frage „Was wird gelernt?“ auch die Frage „Wer lernt?“ entscheidend. Das lernende Subjekt, wir nennen es *Lernverfahren*, nutzt die präsentierten Informationen, um sich ein Bild von dem konkreten Lernziel zu machen. Lernziele sind jedoch nicht immer trivial. Es kommt oft vor, dass das Bild, das sich ein Lernverfahren von einem Lernziel gemacht hat, diesem nicht entspricht. Erhält ein Lernverfahren neue Informationen, dann kann sich das Bild wieder ändern. Die Ausgaben eines Lernverfahren sind also immer hypothetisch. Für das Lernen ist es deshalb wichtig, wie die Menge der *Hypothesen*, der sogenannte *Hypothesenraum*, genau aussieht. Neben der Syntax der Hypothesen ist auch hier die Semantik zu bestimmen, um den Zusammenhang zwischen Hypothesen und möglichen Lernzielen darzustellen. Der Prozess des Hypothesenbildens findet wie schon erwähnt nicht nur einmal statt, sondern wird iteriet, wenn neue Informationen hinzukommen. Deshalb muß zum Schluss definiert werden, wann der Lernprozess *erfolgreich* ist. Weiterführende Informationen zum Thema algorithmisches Lernen finden sich in [JORS99, ZL95, Lan00].

4.1.1 Induktive Inferenz

Basierend auf den Arbeiten von Solomonoff [Sol64] hat Gold [Gol67] das Modell des induktiven Lernens im Limes geschaffen. In diesem Modell wird insbesondere der *Limescharakter eines Lernprozesses* aus unvollständigen Informationen in den Vordergrund gestellt. Ein Lernverfahren bekommt eine wachsende Menge von Beispielen präsentiert und entwickelt daraus Hypothesen. Der Lernprozess ist erfolgreich, wenn die Folge der Hypothese konvergiert, d.h. wenn sich die Ausgabe des Lernverfahrens trotz neuen Daten nicht mehr ändert, und die Endausgabe mit dem Zielkonzept übereinstimmt. Damit muss während des Lernprozesses eine

Generalisierung stattgefunden haben. Wir werden im zweiten Teil des Kapitels genauer auf dieses Modell eingehen.
Ausgehend von diesem Modell gibt es einige Änderungen hinsichtlich der *Konvergenzkriterien* und Anforderungen an das Lernverfahren. Im Folgenden werden wir einige wichtige Änderungen aufzählen. Wie oben schon gesagt, gilt ein Lernverfahren als erfolgreich, wenn die Hypothesenfolge konvergiert und die Endhypothese mit dem Lernziel übereinstimmt. Es ist jedoch nicht klar, wann die Folge konvergiert. Deshalb wird beim *finiten Lernen* zusätzlich verlangt, dass der Zeitpunkt der Konvergenz bekannt ist. Das Goldsche Konvergenzkriterium besagt, dass die konvergierte Hypothese mit dem Lernziel genau übereinstimmen muss. Diese Anforderung wird beim *verhaltenskorrekten Lernen* gemildert. Hier muss die Endhypothese das Zielkonzept korrekt beschreiben, aber nicht mit ihm übereinstimmen. Beim *konsistenten Lernen* muss das Lernverfahren immer eine Hypothese ausgeben, die mit den bisher präsentierten Beispielen konsistent ist. Beim *konservativen Lernen* darf ein Lernverfahren nur dann die Hypothese wechseln, wenn sie einem neuen Beispiel widerspricht. Beim *inkrementellen Lernen* hat ein Lernverfahren nur noch Zugriff auf das letzte präsentierte Beispiel. Beim *monotonen* und *dual-monotonen Lernen* darf ein Lernverfahren seine Hypothese nur mittels Spezialisierung bzw. Generalisierung ändern.

4.1.2 PAC-Lernen

In der Praxis sind manchmal schon Lernverfahren sinnvoll, die nach dem Lernprozess ein Ergebnis ausgeben, das nicht exakt, sondern annähernd bei den relevanten und häufig auftretenden Fällen mit dem Lernziel übereinstimmt. Valiant [Val84] formalisierte diesen Aspekt in seinem Modell des *PAC-Lernens* (*Probably Approximately Correct-Learning*). Beim PAC-Lernen folgt auf eine *Trainingsphase*, in der ein Konzept gelernt wird, eine *Anwendungsphase*, in der das Konzept angewendet wird. Eine Frage, die sich dabei stellt ist die Verbindung zwischen den Beispielen in der Trainingsphase und den Beispielen in der Anwendungsphase. Ist die Trainingssituation eine andere, wie die Anwendungssituation, dann kann es passieren, dass die vom Lernverfahren ausgegebene Hypothese die Menge der Trainingsbeispiele so gut wie möglich approximiert, nicht aber die Menge der Anwendungsbeispiele. Um dieses Problem zu entschärfen, werden Beispiele zufällig bezüglich einer *Wahrscheinlichkeitsverteilung* D erzeugt. Beim PAC-Modell sind besonders zwei Parameter von Bedeutung: Die *Genauigkeit* ϵ legt den Fehler fest, der bei einer guten Annäherung erlaubt ist und die *Konfidenz* δ bestimmt die Wahrscheinlichkeit für diese Annäherung. Die Werte beider Parameter müssen zwischen 0 und 1 liegen.
Das PAC-Lernmodell sieht nun folgendermaßen aus. Angenommen, $c \in \mathcal{C}$ sei das zu lernende Konzept, ϵ die Genauigkeit, δ die Konfidenz und D die Wahrscheinlichkeitsverteilung bezüglich der Beispiele aus dem Lernbereich $\mathcal{X}$ gezogen werden. Ein Lernverfahren M bekommt nun bezüglich D zufällig m Beispiele aus $\mathcal{X}$. m ist dabei die zum Lernen benötigte Anzahl an Beispielen und wird nur anhand von ϵ und δ berechnet. Das Ziel des Lernverfahrens M ist, eine Hypothese h zu berechnen, so dass h mit einer Wahrscheinlichkeit von $1 - \delta$ höchstens einen

Fehler von ϵ besitzt. Seien z.B. $\epsilon = 0,10$ und $\delta = 0,04$, dann gibt das Lernverfahren mit einer Wahrscheinlichkeit von 96% eine Hypothese aus, die 90% der Beispiele korrekt klassifiziert. Das Lernverfahren ist effizient, wenn m (*sample complexity*) und die Zeit zur Verarbeitung dieser Beispiele (*time complexity*) polynomiell in $1\backslash\epsilon$, $1\backslash\delta$ und der Größe von c beschränkt ist.

4.1.3 Maschinelles Lernen

Ein anderes Gebiet des Lernens, das nicht zur algorithmischen Lerntheorie gezählt wird, ist das Gebiet des maschinellen Lernens (für weitere Informationen siehe z.B. [Mit97]). Algorithmen, die in diesem Gebiet untersucht werden, sollen vor allem bei praktischen Lernaufgaben anwendbar sein. Man kann die Algorithmen grob in drei Gruppen des Lernens einteilen. Beim *überwachten Lernen* (*supervised learning*) werden dem Algorithmus eine Eingabewert sowie ein Ausgabewert gegeben und der Algorithmus hat die Aufgabe, die Funktion zu lernen.
Beim *unüberwachten Lernen* (*unsupervised learning*) werden dem Algorithmus nur die Eingabewerte geliefert. Das *Verstärkungslernen* (*reinforcement learning*) steht zwischen dem überwachten und unüberwachten Lernen. Hier erhält der Algorithmus neben dem Eingangswert einer Funktion noch eine Bewertung.

4.2 Das Modell des induktiven Lernens im Limes

Im Folgenden erklären wir die Grundkonzepte von Golds Modell des *induktiven Lernens im Limes* (vgl. [Gol67]).

Wir beschränken uns dabei auf die indizierbare Familie rekursiver Sprachen (vgl. [Ang80]), die wir im Folgenden definieren. Eine Familie von nichtleeren Sprachen $\mathcal{L}$ heißt *indizierbare Familie rekursiver Sprachen* (kurz: *indizierbare Familie*) genau dann, wenn eine Aufzählung $(L_i)_{i\in\mathbb{N}}$ von $\mathcal{L}$ und ein totales und berechenbares Prädikat p existieren, so dass folgendes gilt:

- Für alle $i \in \mathbb{N}$ und alle $w \in \Sigma^+$ gilt: $p(i, w) \leftrightarrow w \in L_i$.

Viele praktisch relevante Sprachfamilien sind indizierbare Familien. Als Beispiele seien die Menge aller kontextsensitiven Sprachen, die Menge aller kontextfreien Sprachen und die Menge aller regulären Sprachen genannt.

Wie unterscheiden zwei Arten, wie wir die Informationen über eine Zielsprache spezifizieren können: das Lernen aus positiven Daten (Lernen aus einem Text) und das Lernen aus positiven und negativen Daten (Lernen aus einem Informanten).

Ein *Text* ist eine unendliche Folge von Wörtern einer Zielsprache, so dass jedes Wort dieser Sprache mindestens einmal in der Folge vorkommt. Sei L eine Sprache und $t = (s_k)_{k\in\mathbb{N}}$ eine unendliche Folge der Wörter aus Σ^*, so dass $content(t) = \{s_k \mid k \in \mathbb{N}\} = L$ gilt. Dann ist t ein Text der Sprache L. Für einen Text t bezeichnen wir mit t_x das Anfangsstück der Länge x, d.h. $t_x = s_0, ..., s_x$.

Ein *Informant* ist eine unendliche Folge aller Wörter des Alphabets. Die Wörter sind dabei nach ihrer Zugehörigkeit zu der Zielsprache klassifiziert. Sei L eine Sprache und $i = ((s_k, b_k))_{k\in\mathbb{N}}$ eine unendliche Folge der Wörter aus Σ^*, so dass $content(i) = \{s_k \mid k \in \mathbb{N}\} = \Sigma^*$, $content(i)^+ = \{s_k \mid k \in \mathbb{N}, b_k = +\} = L$ und $content(i)^- = \{s_k \mid k \in \mathbb{N}, b_k = -\} = \Sigma^* \backslash L$ gilt. Dann ist i ein Informant der Sprache L. Für einen Informanten i bezeichnen wir mit i_x das Anfangsstück der Länge x, d.h. $i_x = (s_0, b_0), ..., (s_x, b_x)$.

Ein Lernverfahren bekommt als Eingabe wachsende Abschnitte eines Textes t (eines Informanten i) der Zielsprache L und generiert seine Hypothese. Die Menge aller zulässigen Hypothesen wird *Hyopthesenraum* genannt. Das Lernverfahren lernt eine Zielsprache L aus einem Text t (Informanten i), wenn sich die Folge der Ausgaben auf eine Hypothese stabilisiert, die genau L beschreibt. Fortan nennen wir diese Lernverfahren IIM (Abkürzung von *inductive inference machine*). Man sagt, dass eine IIM L aus Text (Informant) lernt, wenn sie L aus jedem Text (jedem Informanten) der Sprache lernt. Weiterhin sagt man, dass eine Sprachfamilie $\mathcal{L}$ aus Text (Informant) lernbar ist, wenn es eine IIM gibt, die jede Sprache $L \in \mathcal{L}$ aus Text (Infor-

mant) lernt. Mit *LimTxt* (*LimInf*) bezeichnen wir die Menge aller indizierbaren Familien $\mathcal{L}$, für die es eine IIM gibt, die $\mathcal{L}$ aus Text (Informant) lernt.

Endliche Telltale-Mengen

Beschäftigt man sich mit dem Lernen aus positiven Daten, dann sieht man sich mit einer zusätzlichen Schwierigkeit konfrontiert, die beim Lernen aus positiven und negativen Daten nicht auftritt. Wenn ein Lernverfahren im Verlauf des Lernens eine Hypothese generiert, die eine Übermenge der Zielsprache ist, dann wird dem Lernverfahren beim Lernen aus positiven und negativen Daten irgendwann ein Gegenbeispiel präsentiert. Dieses Gegenbeispiel wird es jedoch niemals beim Lernen aus positiven Daten geben. Deshalb gibt es hier das Problem der *Übergeneralisierung*. Inferenz aus positiven Daten ist deshalb weniger mächtig als Inferenz aus positiven und negativen Daten. Gold (vgl. [Gol67]) hat z.B. gezeigt, dass eine Menge von formalen Sprachen über einem Alphabet Σ, die jede endliche Sprache zusammen mit mindestens einer unendlichen Sprache über Σ enthält, nicht korrekt aus positiven Daten gelernt werden kann.

Dank Angluin (vgl. [Ang80]) wissen wir jedoch, dass unter bestimmten Umständen das Problem der Übergeneralisierung lösbar ist und indizierbare Familien aus positiven Daten lernbar sind:

Theorem 1 ([Ang80]) *Sei $\mathcal{L}$ eine indizierbare Familie. Es gilt $\mathcal{L} \in LimTxt$, gdw. eine Aufzählung $(L_j)_{j\in\mathbb{N}}$ von $\mathcal{L}$ und eine Prozedur p existieren, so dass p bei der Eingabe von $j \in \mathbb{N}$ eine endliche Menge T_j mit folgenden Eigenschaften ausgibt:*

(i) für alle $j \in \mathbb{N}$ gilt $T_j \subseteq L_j$.

(ii) für alle $j, k \in \mathbb{N}$, falls $T_j \subseteq L_k$ gilt, dann ist $L_k \not\subset L_j$.

Eine Menge, die die Eigenschaften von T_j aus Theorem 1 besitzt, bezeichnen wir als *endliche Telltale-Menge*. Intuitiv gesprochen ist eine endliche Telltale-Menge T_L einer Sprache L eine endliche Teilmenge von L, so dass ein Lernverfahren Übergeneralisierung vermeidet, falls es L vermutet und T_L in einem Text auftritt.

5 Lernszenarios für das STALKER-Modell

Im Folgenden zeigen wir, wie ein Wrapper unseres Modells gelernt werden kann. Ein Wrapper ist vollständig durch seine Extraktions- und Iterationsregeln bestimmt. Die einzelnen Iterations- und Extraktionsregeln eines Wrappers stehen nur über den EC-Baum in Verbindung und sind sonst unabhängig voneinander. Wir können deshalb das Problem, einen Wrapper für das gesamte Dokument zu finden, in Teilprobleme aufteilen. Für jeden Knoten im EC-Baum wird eine Extraktionsregel und für jede Liste eine Iterationsregel gesucht. Sowohl die Extraktions- als auch die Iterationsregeln sind wiederum vollständig durch ihre Teilregeln bestimmt. Auch die Teilregeln sind unabhängig voneinander. Wenn wir diese Regeln lernen können, dann können wir auch die Extraktionsregeln und Iterationsregeln und damit den Wrapper lernen. In unserem Fall handelt es sich jedoch nicht um ein klassisches Lernproblem. Ein Benutzer gibt nämlich als Beispiele für den Lernprozess keine Regeln an, sondern er liefert nur implizit Informationen darüber, indem er in einem Beispiel den zu extrahierenden Textabschnitt markiert. Wir werden deshalb das aus [GL01] stammende Konzept des Lernens aus *markierten Text* übernehmen und auf unser Modell anpassen.

5.1 Lernen aus markiertem Text

Ein Benutzer markiert einen zu extrahierenden Text. Er bestimmt, wo der Text anfängt und wo er aufhört. Dadurch teilt er ein Dokument d in drei Teile x, v, y. Die Wortfolge xvy entspricht dabei dem Dokument d. Das Tupel $\langle x, v, y\rangle$ nennen wir ein *markiertes Dokument*.

Seien $n \geq 1$, $m \geq 1$ und $AR = (R_1^A, ..., R_n^A)$ eine Anfangsregel, $ER = (R_1^E, ..., R_m^E)$ eine Endregel sowie $EX = (AR, ER)$ eine Extraktionsregel. Weiterhin sei $M = \langle x, v, y\rangle$ ein markiertes Dokument. Dann ist M ein Beispiel für EX, wenn es $j, i \in \mathbb{N}$ und $v' \in \Sigma^+$ gibt und folgende Bedingungen erfüllt sind:

1. $zerlege_{R_j^A}(d) = (x, v') \wedge zerlege_{R_i^E}(v') = (v, y)$
2. $\neg\exists j' \in \{1, ..., j-1\} \wedge x_1' \in \Sigma^* \wedge x_2' \in \Sigma^+$ mit $zerlege_{R_{j'}^A}(d) = (x_1', x_2')$
3. $\neg\exists i' \in \{1, ..., i-1\} \wedge x_1' \in \Sigma^+ \wedge x_2' \in \Sigma^*$ mit $zerlege_{R_{i'}^E}(v') = (x_1', x_2')$

Da Iterationsregeln auch mehrmals hintereinander angewendet werden können, werden Beispiele für Iterationsregeln anders definiert als Beispiele für Extraktionsregeln.

Seien $n \geq 1$, $m \geq 1$ und $AR = (R_1^A, ..., R_n^A)$ eine Anfangsregel, $ER = (R_1^E, ..., R_m^E)$ eine Endregel sowie $IT = (AR, ER)$ eine Iterationssregel. Weiterhin sei $M = \langle x, v, y \rangle$ ein markiertes Dokument. Dann ist M zum einen ein Beispiel für IT, wenn es $j, i \in \mathbb{N}$ und $v' \in \Sigma^+$ gibt und folgende Bedingungen erfüllt sind:

1. $zerlege_{R_j^A}(d) = (x, v') \wedge zerlege_{R_i^E}(v') = (v, y)$
2. $\neg\exists j' \in \{1, ..., j-1\} \wedge x_1' \in \Sigma^* \wedge x_2' \in \Sigma^+$ mit $zerlege_{R_{j'}^A}(d) = (x_1', x_2')$
3. $\neg\exists i' \in \{1, ..., i-1\} \wedge x_1' \in \Sigma^+ \wedge x_2' \in \Sigma^*$ mit $zerlege_{R_{i'}^E}(v') = (x_1', x_2')$

Zum anderen ist M ein Beispiel für IT, wenn es $j, i, k \in \mathbb{N}$, $x', x_1, ..., x_{k+1}, y_1, ..., y_k \in \Sigma^*$ und $d_2, v', v_1, ..., v_k, v_1', ..., v_{k+1}' \in \Sigma^+$ gibt und folgende Bedingungen erfüllt sind:

1. $zerlege_{R_j^A}(d_2) = (x', v') \wedge zerlege_{R_i^E}(v') = (v, y)$
2. $\neg\exists j' \in \{1, ..., j-1\} \wedge x_1' \in \Sigma^* \wedge x_2' \in \Sigma^+$ mit $zerlege_{R_{j'}^A}(d_2) = (x_1', x_2')$
3. $\neg\exists i' \in \{1, ..., i-1\} \wedge x_1' \in \Sigma^+ \wedge x_2' \in \Sigma^*$ mit $zerlege_{R_{i'}^E}(v') = (x_1', x_2')$
4. $d_2 = rest_{IT}^k(d)$
5. $x = x_1 v_1 ... v_k x_{k+1}$
6. Für alle $0 \leq k' \leq k$ gilt: $\exists j' \in \{1, ..., n\}$ mit $zerlege_{R_{j'}^A}(rest_{IT}^{k'}(d)) = (x_{k'+1}, v_{k'+1}')$ und $\neg\exists j'' \in \{1, ..., j-1'\} \wedge x_1' \in \Sigma^* \wedge x_2' \in \Sigma^+$ mit $zerlege_{R_{j''}^A}(rest_{IT}^{k'}(d)) = (x_1', x_2')$
7. Für alle $1 \leq k' \leq k$ gilt: $\exists i' \in \{1, ..., m\}$ mit $zerlege_{R_{i'}^E}(v_{k'}') = (v_{k'}, y_{k'})$ und $\neg\exists i'' \in \{1, ..., i-1'\} \wedge x_1' \in \Sigma^+ \wedge x_2' \in \Sigma^*$ mit $zerlege_{R_{i''}^E}(v_{k'}') = (v_1', v_2')$

Dabei meinen wir mit $rest_{IT}^j(d)$ die j-fache iterative Anwendung von $rest_{IT}$ auf d und es sei nach Vereinbarung $rest_{IT}^0(d) = d$.

Ein markierter Text t für eine Extraktionsregel EX (Iterationsregel IT) ist eine unendliche Folge von markierten Dokumenten, die als Beispiele für EX (IT) dienen. Dabei muss jedes Beispiel von EX (IT) mindestens einmal in t vorkommen. Ein Lernverfahren bekommt als Eingabe wachsende Abschnitte eines markierten Textes t für eine Zielregel EX (IT) und generiert seine Hypothese. Es lernt die Extraktionsregel EX (Iterationsregel IT), wenn sich die Folge der Ausgaben auf eine Hypothese EX' (IT') stabilisiert, so dass $extract_{EX'}(d) = extract_{EX}(d)$ ($iterate_{IT'}(d) = iterate_{IT}(d)$) für alle Dokumente $d \in \Sigma^*$ gilt.

5.2 Regellernen

Im Weiteren wollen wir eine Auswahl an relevanten Lernszenarios aufstellen, die wir dann im nächsten Kapitel untersuchen werden. Da die Untersuchungen sonst den Umfang der Arbeit sprengen würden, werden wir zwei Einschränkungen an den Wrapper einführen.

Normalerweise können Extraktions- und Iterationsregeln auch disjunktive Regeln sein, wobei die Disjunkte selbst wieder Regeln sind. Deshalb haben wir bisher angenommen, dass Extraktions- und Iterationsregeln geordnete Listen von Regeln sind. Im Folgenden nehmen wir jedoch an, dass diese Listen nur ein Element beinhalten, dass es also keine disjunktive Regeln gibt. Wir nehmen weiter an, dass wir wissen, wieviele Operationen eine Regel enthält und was die letzte Operation ist.

Jede Regel definiert eine bestimmte Sprache. Die Elemente dieser Sprache sind alle Texte auf die sich die Regel anwenden lässt. Somit kann man das Lernen einer Regel als das Lernen einer Sprache sehen. Eine Regel ist vollständig durch ihre Operationen und die Argumentensprachen der Operationen bestimmt. Die zu lernenden Sprachen hängen also von den Operationen der Regeln und deren Argumentensprachen ab.

Wie wir im vorherigen Abschnitt gesehen haben, ist ein Beispiel für ein Lernverfahren keine ununterbrochene Wortfolge, sondern in drei aufeinanderfolgende Textteile aufgeteilt. Sei # ein Zeichen, das nicht in Σ vorkommt. Wir benutzen #, um diese Aufteilung in einer zu lernenden Sprache darzustellen. Dabei wird vor dem Anfang und nach dem Ende der zu extrahierenden Information ein # eingefügt.

Sei $R = (L_1, ..., L_n, op)$ eine Regel und sei $(\langle x_j, v_j, y_j \rangle)_{j \in \mathbb{N}}$ der präsentierte markierte Text. Das Lernen von Anfangsregeln unterscheidet sich nicht vom Lernen von Endregeln, da beide Vorwärtsregeln sind. Deshalb beschäftigen wir uns im Weiteren nur mit Anfangsregeln. Es ist somit ausreichend, $(x_j \# v_j)_{j \in \mathbb{N}}$ zu betrachten. $x_j \# v_j$ ist ein Beispiel für die Sprache, die durch die Regel R bestimmt ist. Wir wissen also, dass $x_j \# v_j$ Element dieser Sprache ist. In Abhängigkeit von der Anzahl der Operationen in der Regel und der Art der letzten Operation liefert die Sequenz dem Lernverfahren aber noch weitere Informationen.

Wir werden nun auf den nächsten Seiten für verschiedene Regeln die durch die Regel bestimmte Sprache definieren.

Lernen von Regeln mit einer *SkipTo*-Operation

Angenommen, die Regel besteht aus einer *SkipTo*-Operation. Dann liefert $(x_j \# v_j)_{j \in \mathbb{N}}$ Informationen über die Argumentensprache L dieser *SkipTo*-Operation. Wir wissen, dass alle Zeichen im Dokument übersprungen werden, bis das erste Element von L gefunden wird. Weiterhin wissen wir, dass nach dem gefundenen Element gehalten wird. Somit besteht x_j aus einer Menge von übersprungenen Zeichen $r \in \Sigma^*$ gefolgt vom einem Element $l \in L$ und der Rest des Dokumentes steht nach dem #. Da *SkipTo* immer nach dem ersten passenden Element hält, darf kein anderes Wort von L existieren, was vor l in x_j steht. Außerdem darf es kein kleineres Wort von L geben, das an derselben Position wie l startet. Um nun die durch die Regel bestimmte Sprache zu definieren, werden wir eine Hilfskonstruktion einführen, die die Bedingungen festhält, die die Operation *SkipTo* an einen Text stellt.

Definition 8 *Sei $\mathcal{L}$ eine indizierbare Familie rekursiver Sprachen und $L \in \mathcal{L}$. Dann definiert L die folgende Menge von Tupeln SkipTo(L). SkipTo(L) ist die Menge aller (r, l, v), so dass folgende Bedingungen erfüllt sind:*

1. $r \in \Sigma^*$, $v \in \Sigma^+$ *[Randbedingungen]*
2. $l \in L$ *[Sprachbedingung]*
3. $\neg \exists l' \in L$ *mit* $l' \sqsubset_p l$ *[Minimalitätsbedingung]*
4. $\neg \exists r' \in \Sigma^*, l' \in L$ *mit* $r' \sqsubset_p r \wedge r'l' \sqsubseteq_p rlv$ *[Startbedingung]*

△

Die Folge $(x \# v)_{j \in \mathbb{N}}$ stellt nun einen Text der folgenden Sprache dar:

- $ST(L) = \{rl \# v \mid (r, l, v) \in SkipTo(L)\}$

Lernen von Regeln mit einer *SkipUntil*-Operation

Angenommen, die Regel besteht aus einer *SkipUntil*-Operation. Dann liefert $(x_j \# v_j)_{j \in \mathbb{N}}$ Informationen über die Argumentensprache L dieser Operation. Wir wissen, dass alle Zeichen im Dokument übersprungen werden, bis das erste Element von L gefunden wird. Weiter wissen wir, dass vor dem gefundenen Element gehalten wird. Damit besteht x_j aus einer Menge von übersprungenen Zeichen $r \in \Sigma^*$ und nach dem # steht das Element l aus L gefolgt vom Rest des Dokumentes. Da *SkipUntil* immer vor dem ersten passenden Element hält, darf kein anderes Wort von L existieren, was vor l in x_j steht. Um nun die durch die Regel bestimmte Sprache zu definieren, werden wir erneut eine Hilfskonstruktion einführen, diesmal für die Operation *SkipUntil*.

Definition 9 *Sei $\mathcal{L}$ eine indizierbare Familie rekursiver Sprachen und $L \in \mathcal{L}$. Dann definiert L die folgende Menge von Tupeln SkipUntil(L). SkipUntil(L) ist die Menge aller (r, l, v), so dass folgende Bedingungen erfüllt sind:*

1. $r, v \in \Sigma^*$ *[Randbedingungen]*
2. $l \in L$ *[Sprachbedingung]*
3. $\neg \exists r' \in \Sigma^*, l' \in L$ *mit* $r' \sqsubset_p r \wedge r'l' \sqsubseteq_p rlv$ *[Startbedingung]*

△

Die Definition von *SkipUntil* unterscheidet sich von der Definition von *SkipTo* nur im Fehlen der Minimalitätsbedingung. Diese Bedingung ist nicht mehr nötig, da die *SkipUntil*-Operation auch bei Verletzung der Minimalitätsbedingung dasselbe Ergebnis liefert. Die Folge $(x \# v)_{j \in \mathbb{N}}$ stellt dann einen Text der folgenden Sprache dar:

- $SU(L) = \{r \# lv \mid (r, l, v) \in SkipUntil(L)\}$

Lernen von Regeln mit einer *Next*-Operation

Angenommen, die Regel besteht aus einer *Next*-Operation. Dann liefert $(x_j \# v_j)_{j \in \mathbb{N}}$ Informationen über die Argumentensprache L dieser Operation. Wir wissen, dass keine Zeichen übersprungen werden, sondern nur überprüft wird, ob das nachfolgende Wort eine Element von L ist. Damit wissen wir, dass x_j das leere Wort sein und nach dem # das Element l aus L gefolgt vom Rest des Dokumentes stehen muss. Da *Next* keine Zeichen überspringt, gibt es keine weiteren Informationen über die zu lernende Sprache. Um diese Sprache in Termini formaler Sprachen zu definieren, bilden wir eine Hilfskonstruktion, die die Bedingungen von *Next* festhält, die durch *Next*.

Definition 10 *Sei $\mathcal{L}$ eine indizierbare Familie rekursiver Sprachen und $L \in \mathcal{L}$. Dann definiert L die folgende Menge von Tupeln Next(L). Next(L) ist die Menge aller (l, v), so dass folgende Bedingungen erfüllt sind:*

1. *$l \in L$ [Sprachbedingung]*
2. *$v \in \Sigma^*$ [Randbedingung]*

△

Die Folge $(x \# v)_{j \in \mathbb{N}}$ stellt schließlich einen Text der folgenden Sprache dar:

- $SU(L) = \{\#lv \mid (l, v) \in Next(L)\}$

Weitere Lernbarkeitsaufgaben

Zusätzlich zu den oben definierten Regeln, existieren noch Regeln die mehrere Operationen enthalten. Auch die durch diese Regeln bestimmte Sprachen sind mittels der drei vorhin eingeführten Hilfskonstruktionen definierbar.

- $ST^{(n)}(L_1, ..., L_n) = \{r_1 l_1 ... r_n l_n \# v \mid (r_1, l_1, r_2 l_2 ... r_n l_n v) \in SkipTo(L_1) \wedge r_2 l_2 ... r_n l_n \# v \in ST^{(n-1)}(L_2, ..., L_n)\}$
- $ST^{(n)}SU(L_1, ..., L_{n+1}) = \{r_1 l_1 ... r_{n+1} \# l_{n+1} v \mid (r_1, l_1, r_2 l_2 ... r_{n+1} l_{n+1} v) \in SkipTo(L_1) \wedge r_2 l_2 ... r_{n+1} \# l_{n+1} v \in ST^{(n-1)}SU(L_2, ..., L_{n+1})\}$
- $ST^{(n)}NT(L_1, ..., L_{n+1}) = \{r_1 l_1 ... l_n \# l_{n+1} v \mid (r_1, l_1, r_2 l_2 ... l_n l_{n+1} v) \in SkipTo(L_1) \wedge r_2 l_2 ... l_n \# l_{n+1} v \in ST^{(n-1)}NT(L_2, ..., L_{n+1})\}$

$ST^{(n)}(L_1, ..., L_n)$ ist das n-malige Ausführen von *SkipTo*, $ST^{(n)}SU(L_1, ..., L_{n+1})$ das Ausführen von *SkipUntil* nach dem n-maligen Ausführen von *SkipTo* und $ST^{(n)}NT(L_1, ..., L_{n+1})$ das Ausführen von *Next* nach dem n-maligen Ausführen von *SkipTo*.

6 Lernbarkeitsuntersuchungen

Das folgende Kapitel stellt den Hauptteil unserer Arbeit dar. Wir werden nun die in Kapitel 5 vorgestellten Lernszenarios für das STALKER-Modell hinsichtlich der Lernbarkeit von endlichen Sprachen im Limes untersuchen.

6.1 Lernbarkeit von Regeln, die ausschließlich *SkipTo*-Operationen enthalten

Zunächst werden wir Regeln untersuchen, die nur *SkipTo*-Operationen besitzen. Gemeinsam haben sie, dass nach dem gefundenen Argument der letzten Operation gehalten wird.

Lernbarkeit von Regeln mit einer *SkipTo*-Operation

Sei $\mathcal{ST}(\mathcal{L})$ die Menge aller Regeln, in denen eine *SkipTo*-Operation durchgeführt wird, d.h. $\mathcal{ST}(\mathcal{L}) = \{ST(L) \mid L \in \mathcal{L}\}$. Wir werden nun zeigen, dass diese Menge für endliche Sprachen lernbar ist.

Theorem 2 $\mathcal{ST}(\mathcal{L}_f) \in LimTxt$.

Beweis. Wir wissen, dass eine Sprache aus Text lernbar ist, wenn wir eine Telltale-Menge für diese Sprache bilden können. Sei $L \in \mathcal{L}_f$. Wir definieren die Menge $T_{ST(L)} = \{l\#v \mid (\varepsilon, l, v) \in \mathit{SkipTo}(L) \wedge |v| = 1\}$.
Wir beweisen nun, dass $T_{ST(L)}$ eine Telltale-Menge von $ST(L)$ ist. Für den Beweis nehmen wir das Gegenteil an, d.h. seien $L, L' \in \mathcal{L}_f$ mit $T_{ST(L)} \subseteq ST(L') \subset ST(L)$.
Sei $w \in ST(L) \backslash ST(L')$ mit $w = rl\#v$ und $(r, l, v) \in \mathit{SkipTo}(L)$. Somit gilt $w \notin ST(L')$ und es existiert keine Zerlegung für w, die die Bedingungen der Definition von $\mathit{SkipTo}(L')$ erfüllt. Insbesondere gilt auch $(r, l, v) \notin \mathit{SkipTo}(L')$. Daraus folgt, dass eine der Bedingungen der Definition von $\mathit{SkipTo}(L')$ verletzt sein muss. Die Randbedingungen sind wegen $(r, l, v) \in \mathit{SkipTo}(L)$ immer erfüllt. So muss mindestens einer der folgenden Fälle gelten:

Fall (2.1) $l \notin L'$ [Die Sprachbedingung ist verletzt]

Fall (2.2) $\exists l' \in L'$ mit $l' \sqsubset_p l$ [Die Minimalitätsbedingung ist verletzt]

Fall (2.3) $\exists r' \in \Sigma^*, l' \in L'$ mit $r' \sqsubset_p r \wedge r'l' \sqsubseteq_p rlv$ [Die Startbedingung ist verletzt]

In den nachfolgenden Lemmata wird bewiesen, dass keiner der Fälle gelten kann. Siehe Lemma 3 für Fall (2.1), Lemma 1 für Fall (2.2) und Lemma 2 für Fall (2.3). Deshalb ist $(r, l, v) \in \mathit{SkipTo}(L')$ und damit auch $w \in ST(L')$. Dies ist ein Widerpruch zu unserer Annahme, dass $w \in ST(L) \backslash ST(L')$ gilt. Somit existiert kein $w \in ST(L) \backslash ST(L')$, was wiederum ein Widerspruch zu unserer Annahme ist, dass $T_{ST(L)} \subseteq ST(L') \subset ST(L)$ gilt. Aus diesem Grund ist $T_{ST(L)}$

eine Telltale-Menge von $ST(L)$. Es gilt also $ST(L) \in LimTxt$. Da $ST(L)$ beliebig ist, gilt $\mathcal{ST}(\mathcal{L}_f) \in LimTxt$.

■

Zur besseren Übersichtlichkeit der Beweise treffen wir für die restlichen Lemmata in diesem Kapitel folgende Vereinbarungen. Seien $L, L' \in \mathcal{L}_f$. Es gelte $T = T_{ST(L)} = \{l\#v \mid (\varepsilon, l, v) \in SkipTo(L)$ und $|v| = 1\}$, $ST = ST(L)$ und $ST' = ST(L')$.

Die folgenden drei Lemmata widerlegen, dass die Fälle (2.1) bis (2.3) gelten. Lemma 1 beweist, dass Fall (2.2) nicht gelten kann.

Lemma 1 *Seien $T \subseteq ST' \subset ST$ und $w \in ST \backslash ST'$ mit $w = rl\#v$ und $(r, l, v) \in SkipTo(L)$. Dann erfüllt (r, l, v) die Minimalitätsbedingung der Definition von $SkipTo(L')$, d.h. $\neg\exists l' \in L'$ mit $l' \sqsubset_p l$.*

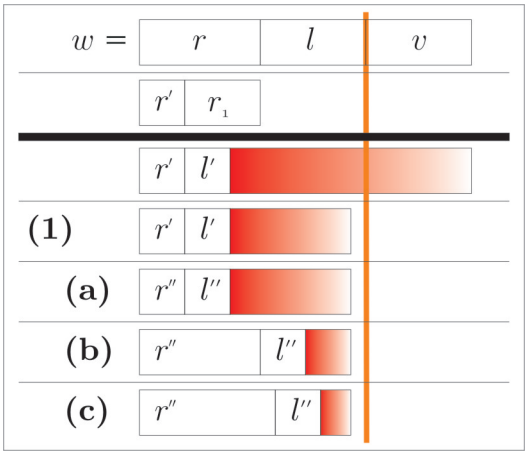
Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. es existiere ein $l' \in L'$ mit $l' \sqsubset_p l$. Wir betrachten das Wort $w' = r'l'\#v'$ mit dem kürzesten l', das die vorherige Bedingung erfüllt, $r' = \varepsilon$ und $l'v' = l$. Wir beweisen nun, dass $w' \in ST'$ gilt. Es sind $r' = \varepsilon$ und $v' \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l' \in L'$ [Die Sprachbedingung ist erfüllt]. Da l' das kürzeste Wort aus L' ist, das ein Präfix von l ist, existiert auch kein $l'' \in L'$ mit $l'' \sqsubset_p l'$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $r' = \varepsilon$ existieren keine $r'' \in \Sigma^*$ und $l'' \in L'$, so dass $r'' \sqsubset_p r' \wedge r''l'' \sqsubseteq_p r'l'v'$ gilt [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r', l', v') \in SkipTo(L')$ und $w' \in ST'$. Wegen $w \in ST$ gilt außerdem $l' \notin L$. Angenommen, $w' \in ST$ gilt. Dann muss eine andere Zerlegung von w' existieren, die die Bedingungen der Definition von $SkipTo(L)$ erfüllt. Sei $w' = r''l''\#v'$ diese Zerlegung mit $(r'', l'', v') \in SkipTo(L)$. Aus $r' = \varepsilon$ folgt $r''l'' = l'$ und $r'' \in \Sigma^+$. Dann gilt aber $r' = \varepsilon \sqsubset_p r''$ und $r'l \sqsubseteq_p r''l''v'$. Damit kann (r'', l'', v') die Startbedingung von $SkipTo(L)$ nicht erfüllen. Dies ist aber ein Widerspruch zu unserer Annahme, dass $(r'', l'', v') \in SkipTo(L)$ gilt. Also ist $w' \notin ST$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $ST' \subset ST$ gilt und somit erfüllt (r, l, v) die Minimalitätsbedingung der Definition von $SkipTo(L')$.

■

Lemma 2 widerlegt, dass Fall (2.3) gelten kann.

Lemma 2 *Seien $T \subseteq ST' \subset ST$ und $w \in ST \backslash ST'$ mit $w = rl\#v$ und $(r, l, v) \in SkipTo(L)$. Dann erfüllt (r, l, v) die Startbedingung der Definition von $SkipTo(L')$, d.h. $\neg\exists r' \in \Sigma^*, l' \in L'$ mit $r' \sqsubset_p r \wedge r'l' \sqsubseteq_p rlv$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. es existiere ein $r' \in \Sigma^*$ und ein



Die Abbildung zeigt die unterschiedlichen Fälle, die in einem Lemma auftreten können. Alle Angaben, die wir jetzt über das Bild machen, gelten auch für die anderen Bilder in diesem Kapitel. In den ersten Zeilen über der dicken schwarzen Linie stehen die Annahmen, die wir für ein Lemma vorraussetzen. In der ersten Zeile unter der der dicken schwarzen Linie steht die allgemeine Annahme, die im Lemma widerlegt werden soll. Darunter stehen die verschiedenen Fälle des Lemmas. Balken mit rotem Verlauf symbolisieren das variable Ende eines Wortes. Das Wort endet frühestens am Anfang des Balkens auf der dunklen Seite und spätestens am Ende des Balkens auf der hellen Seite. Der orangene Strich gibt die Position an, wo im Wort das # steht.

Abbildung 5: Lemma 2 Fall (1)

$l' \in L'$ mit $r' \sqsubset_p r \wedge r'l' \sqsubseteq_p rlv$. Aus der Menge der $r'l'$, die diese Bedingung erfüllen, betrachten wir dabei das Paar mit dem kürzesten r' und einem l', so dass kein $l'' \in L'$ existiert mit $l'' \sqsubset_p l'$. Dann gilt einer der drei folgenden Fälle:

Fall (1) $r'l' \sqsubset_p rl$

Fall (2) $rl \sqsubset_p r'l'$

Fall (3) $rl = r'l'$

Fall (1): Angenommen, $r'l' \sqsubset_p rl$ gilt. Dann existiert ein $v' \in \Sigma^+$ mit $r'l'v' = rlv$. Wir zeigen, dass $w' = r'l'\#v' \in ST'$ gilt. Es sind $r' \in \Sigma^*$ und $v' \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l' \in L'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r' und l' existieren kein $l'' \in L'$ mit $l'' \sqsubset_p l'$ [Die Minimalitätsbedingung ist erfüllt] und keine $r'' \in \Sigma^*$ und $l'' \in L'$, so dass $r'' \sqsubset_p r' \wedge r''l'' \sqsubseteq_p r'l'v'$ gilt [Die Startbedingung ist erfüllt]. Da (r', l', v') alle Bedingungen von $SkipTo(L')$ erfüllt, gilt $(r', l', v') \in SkipTo(L')$ und $w' \in ST'$. Wegen $w \in ST$ gilt außerdem $l' \notin L$. Angenommen, $w' \in ST$ gilt, dann muss eine andere Zerlegung für w' existieren, die die Bedingungen der Definition von $SkipTo(L)$ erfüllt. Sei $w' = r''l''\#v'$ diese Zerlegung mit $(r'', l'', v') \in SkipTo(L)$. Es gibt nun drei Möglichkeiten:

Fall (a): Angenommen, es gilt $r'' \sqsubset_p r$. Da außerdem $r''l'' = r'l' \sqsubseteq_p rlv$ gilt, erfüllt (r, l, v) nicht die Startbedingung von $SkipTo(L)$. Weil dies ein Widerspruch zu unserer Annahme ist, dass $(r, l, v) \in SkipTo(L)$ gilt, kann Fall (a) nicht auftreten.

Fall (b): Angenommen, es gilt $r'' = r$. Dann gilt aber auch $l'' \sqsubset_p l$. Damit kann (r, l, v) die Minimalitätsbedingung von $SkipTo(L)$ nicht erfüllen, was ein Widerspruch zu unserer Annahme ist, dass $(r, l, v) \in SkipTo(L)$ gilt. Somit haben wir Fall (b) widerlegt.

Fall (c): Angenommen, es gilt $r \sqsubset_p r''$. Da außerdem $rl \sqsubseteq_p r''l''v'$ gilt, kann (r'', l'', v') die Startbedingung von $SkipTo(L)$ nicht erfüllen. Dies ist ein Widerspruch zu unserer Annahme ist, dass $(r'', l'', v') \in SkipTo(L)$ gilt. Somit kann Fall (c) nicht auftreten.

Da wir alle drei Fälle widerlegt haben, kann $(r'', l'', v') \in SkipTo(L)$ nicht gelten. Also ist $w' \notin ST$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $ST' \subset ST$ gilt. Es existieren also keine $r' \in \Sigma^*, l' \in L'$ mit $r' \sqsubset_p r$ und $r'l' \sqsubset_p rl$.

Fall (2): Angenommen, $rl \sqsubset_p r'l'$ gilt. Wir betrachten das Wort $w' = r_\varepsilon l'\#v'$ mit einem $v' \in \Sigma$ und $r_\varepsilon = \varepsilon$. Wir zeigen zuerst, dass $(r_\varepsilon, l', v') \in SkipTo(L')$ gilt. Es sind $r_\varepsilon \in \Sigma^*$ und $v' \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l' \in L'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von l' existiert kein $l'' \in L'$ mit $l'' \sqsubset_p l'$ [Die Minimalitätsbedingung ist erfüllt]. Weil $r_\varepsilon = \varepsilon$ gilt, existieren keine $r'' \in \Sigma^*$ und $l'' \in L'$, so dass $r'' \sqsubset_p r_\varepsilon \wedge r''l'' \sqsubseteq_p r_\varepsilon l'v'$ gilt [Die Startbedingung ist erfüllt]. Da (r_ε, l', v') alle Bedingungen von $SkipTo(L')$ erfüllt, gilt $(r_\varepsilon, l', v') \in SkipTo(L')$ und $w' \in ST'$. Wegen $w \in ST$ gilt außerdem $l' \notin L$. Angenommen, $w' \in ST$ gilt, dann muss eine andere Zerlegung für w' existieren, die die Bedingungen der Definition von $SkipTo(L)$ erfüllt. Sei $w' = r''l''\#v'$ diese Zerlegung mit $(r'', l'', v') \in SkipTo(L)$. Wegen $r' \sqsubset_p r$ und $rl \sqsubset_p r'l'$ existiert ein $r_1 \in \Sigma^*$ mit $r'r_1 = r$, $r_1 l \sqsubset_p l'$ und $l' \sqsubseteq_p r_1 lv$. Es gibt drei Möglichkeiten:

Fall (a): Angenommen, es gilt $r'' \sqsubset_p r_1$. Dann gilt aber auch $r'r'' \sqsubset_p r'r_1 = r$. Da außerdem $r'r''l'' = r'l' \sqsubseteq_p rlv$ gilt, erfüllt (r, l, v) nicht die Startbedingung von $SkipTo(L)$. Weil dies ein Widerspruch zu unserer Annahme ist, dass $(r, l, v) \in SkipTo(L)$ gilt, kann Fall (a) nicht auftreten.

Fall (b): Angenommen, es gilt $r'' = r_1$. Dann gilt aber auch $l \sqsubset_p l''$. Damit kann (r'', l'', v') die Minimalitätsbedingung von $SkipTo(L)$ nicht erfüllen, was ein Widerspruch zu unserer Annahme ist, dass $(r'', l'', v') \in SkipTo(L)$ gilt. Somit haben wir Fall (b) widerlegt.

Fall (c): Angenommen, es gilt $r_1 \sqsubset_p r''$. Da außerdem $r_1 l \sqsubseteq_p r''l''v'$ gilt, kann (r'', l'', v') die Startbedingung von $SkipTo(L)$ nicht erfüllen. Dies ist ein Widerspruch zu unserer Annahme ist, dass $(r'', l'', v') \in SkipTo(L)$ gilt. Somit kann Fall (c) nicht auftreten.

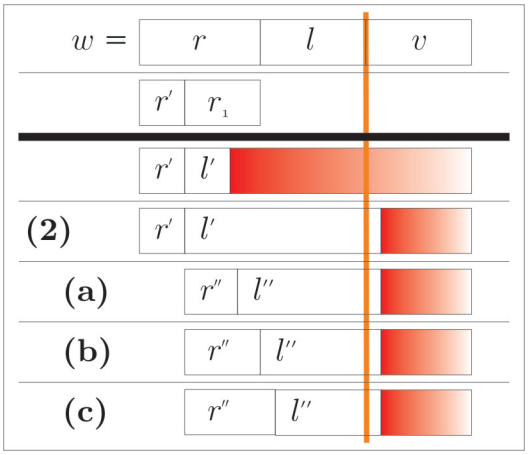


Abbildung 6: Lemma 2 Fall (2)

Da wir die Fälle (a) bis (c) widerlegt haben, kann $(r'', l'', v') \in \mathit{SkipTo}(L)$ nicht gelten. Also ist $w' \notin ST$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $ST' \subset ST$ gilt. Es existieren also keine $r' \in \Sigma^*, l' \in L'$ mit $r' \sqsubset_p r$ und $rl \sqsubset_p r'l'$.

Fall (3): Angenommen $rl = r'l'$ gilt. Dann betrachten wir die Zerlegung $w = r'l'\#v$. Aufgrund unserer Wahl von r' und l' gilt $(r', l', v) \in \mathit{SkipTo}(L')$ und damit auch $w \in ST(L')$. Dies ist aber ein Widerspruch zu unserer Annahme, dass $w \in ST \backslash ST'$ gilt. Es gibt also keine $r' \in \Sigma^*, l' \in L'$ mit $r' \sqsubset_p r$ und $rl = r'l'$.

Da alle Fälle widerlegt sind, erfüllt (r, l, v) die Startbedingung der Definition von $\mathit{SkipTo}(L')$. ■

Lemma 3 beweist, dass Fall (2.1) nicht gilt. In Lemma 1 und Lemma 2 haben wir unabhängig von Fall (2.1) gezeigt, dass die Fälle (2.2) und (2.3) nie gelten können. Deshalb können wir für den Beweis von Lemma 3 annehmen, dass Fall (2.2) und Fall (2.3) nicht gelten.

Lemma 3 *Seien $T \subseteq ST' \subset ST$ und $w \in ST \backslash ST'$ mit $w = rl\#v$. Weiterhin sei $(r, l, v) \in \mathit{SkipTo}(L)$ und (r, l, v) erfülle o.B.d.A. die Minimalitätsbedingung und Startbedingung der Definition von $\mathit{SkipTo}(L')$. Dann erfüllt (r, l, v) auch die Sprachbedingung der Definition von $\mathit{SkipTo}(L')$, d.h. es gilt $l \in L'$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. sei $l \notin L'$. Wir betrachten das Wort $w' = l\#v'$. $v' \in \Sigma$ ist dabei das erste Zeichen von v. Es gilt $w' \in T$. Aus $T \subseteq ST'$ folgt $w' \in ST'$. Da $l \notin L'$ gilt, muss eine andere Zerlegung für w' existieren, die die Bedingungen von ST' erfüllt. Angenommen, $r'l'\#v'$ sei diese Zerlegung. Dann gilt $(r', l', v') \in \mathit{SkipTo}(L')$ mit

$r'l' = l$. Wir betrachten die Zerlegung $w = rr'l'\#v$. Da $w \notin ST'$ gilt, ist $(rr', l', v) \notin \mathit{SkipTo}(L')$ und eine der Bedingungen der Definition von $\mathit{SkipTo}(L')$ darf nicht erfüllt sein. Die Randbedingung ist wegen $(r, l, v) \in \mathit{SkipTo}(L)$ erfüllt, die Sprach- und Minimalitätsbedingung sind wegen $(r', l', v') \in \mathit{SkipTo}(L')$ erfüllt. Da (r, l, v) die Startbedingung der Definition von SkipTo(L') erfüllt, wissen wir außerdem, dass keine $r'' \in \Sigma^*, l'' \in L'$ existieren mit $r'' \sqsubset_p r \wedge r''l'' \sqsubseteq_p rlv$. Somit kann es nur einen Grund für $(rr', l', v) \notin \mathit{SkipTo}(L')$ geben: $\exists r'' \in \Sigma^*, l'' \in L'$ mit $r'' \sqsubset_p r' \wedge r''l'' \sqsubseteq_p r'l'v$. Angenommen, dies gilt. Aus der Menge der $r''l''$, die diese Bedingung erfüllen, betrachten wir das Paar mit dem kürzesten r'' und einem l'', so dass kein $l_1 \in L'$ existiert mit $l_1 \sqsubset_p l''$. Wegen $(r', l', v') \in \mathit{SkipTo}(L')$ ist $r'l' \sqsubset_p r''l''$ und damit auch $l \sqsubset_p r''l''$. Es gibt nun zwei Möglichkeiten:

Fall (1): Es existieren $r_1 \in \Sigma^*$, $v_1 \in \Sigma^+$ und $l_1 \in L'$ mit $r_1 \sqsubset_p r'' \wedge r_1l_1 = r''l''v_1$.

Angenommen, dies gilt. Aus der Menge der r_1l_1, die diese Bedingung erfüllen, betrachten wir das Paar mit dem kürzesten r_1 und einem l_1, so dass kein $l_1' \in L'$ existiert mit $l_1' \sqsubset_p l_1$. Wir betrachten das Wort $w'' = r_1l_1\#v''$ mit einem $v'' \in \Sigma$. Wir beweisen nun, dass $(r_1, l_1, v'') \in \mathit{SkipTo}(L')$ gilt. Es sind $r_1 \in \Sigma^*$ und $v'' \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l_1 \in L'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_1 und l_1 existieren kein $l_1' \in L'$ mit $l_1' \sqsubset_p l_1$ [Die Minimalitätsbedingung ist erfüllt] und keine $r_1' \in \Sigma^*$ und $l_1' \in L'$, so dass $r_1' \sqsubset_p r_1 \wedge r_1'l_1' \sqsubseteq_p r_1l_1v''$ gilt [Die Startbedingung ist erfüllt]. Da (r_1, l_1, v'') alle Bedingungen von $\mathit{SkipTo}(L')$ erfüllt, gilt $(r_1, l_1, v'') \in \mathit{SkipTo}(L')$ und $w'' \in ST'$. Angenommen, $w'' \in ST$ gilt, dann muss eine Zerlegung für w'' existieren, die die Bedingungen der Definition von $\mathit{SkipTo}(L)$ erfüllt. Sei $w'' = r_1'l_1'\#v''$ diese Zerlegung mit $(r_1', l_1', v'') \in \mathit{SkipTo}(L)$. Es gilt $l \sqsubset_p r''l''$ und deshalb auch $l \sqsubset_p r_1'l_1'$. Dann gilt aber $r_\varepsilon \sqsubset_p r_1'$ und $r_\varepsilon l \sqsubseteq_p r_1'l_1'v''$ mit $r_\varepsilon = \varepsilon$. Somit kann (r_1', l_1', v'') die Startbedingung von $\mathit{SkipTo}(L)$ nicht erfüllen. Dies ist ein Widerspruch zu unserer Annahme ist, dass $(r_1', l_1', v'') \in \mathit{SkipTo}(L)$ gilt. Daraus folgt $w'' \notin ST$. Dies ist wiederum ein Widerspruch zur Annahme, dass $ST' \subset ST$ gilt. (r, l, v) erfüllt also die Sprachbedingung der Definition von $\mathit{SkipTo}(L')$.

Fall (2): Es existieren keine $r_1 \in \Sigma^*$, $v_1 \in \Sigma^+$ und $l_1 \in L'$ mit $r_1 \sqsubset_p r'' \wedge r_1l_1 = r''l''v_1$.

Angenommen, dies gilt. Wir betrachten das Wort $w'' = r''l''\#v''$ mit einem $v'' \in \Sigma$. Wir beweisen nun, dass $(r'', l'', v'') \in \mathit{SkipTo}(L')$ gilt. Es sind $r'' \in \Sigma^*$ und $v'' \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l'' \in L'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r'' und l'' existieren kein $l_1 \in L'$ mit $l_1 \sqsubset_p l''$ [Die Minimalitätsbedingung ist erfüllt] und keine $r_1 \in \Sigma^*$ und $l_1 \in L'$, so dass $r_1 \sqsubset_p r'' \wedge r_1l_1 \sqsubseteq_p r''l''v''$ gilt [Die Startbedingung ist erfüllt]. Da (r'', l'', v'') alle Bedingungen von $\mathit{SkipTo}(L')$ erfüllt, gilt $(r'', l'', v'') \in \mathit{SkipTo}(L')$ und $w'' \in ST'$. Angenommen, $w'' \in ST$ gilt, dann muss eine Zerlegung für w'' existieren, die die Bedingungen der Definition von $\mathit{SkipTo}(L)$ erfüllt. Sei $w'' = r_1l_1\#v''$ diese Zerlegung mit $(r_1, l_1, v'') \in \mathit{SkipTo}(L)$. Es gilt $l \sqsubset_p r''l''$ und deshalb auch $l \sqsubset_p r_1l_1$. Dann gilt aber $r_\varepsilon \sqsubset_p r_1$ und $r_\varepsilon l \sqsubseteq_p r_1l_1v''$ mit $r_\varepsilon = \varepsilon$. Somit kann (r_1, l_1, v'') die Startbedingung von $\mathit{SkipTo}(L)$ nicht

erfüllen. Dies ist ein Widerspruch zu unserer Annahme ist, dass $(r_1, l_1, v'') \in \mathit{SkipTo}(L)$ gilt. Daraus folgt $w'' \notin ST$. Dies ist wiederum ein Widerspruch zur Annahme, dass $ST' \subset ST$ gilt. (r, l, v) erfüllt also die Sprachbedingung der Definition von $\mathit{SkipTo}(L')$.

■

Lernbarkeit von Regeln mit zwei *SkipTo*-Operationen

Sei $\mathcal{ST}^{(2)}(\mathcal{L})$ die Menge aller Regeln mit zwei aufeinanderfolgenden *SkipTo*-Operationen, d.h. $\mathcal{ST}^{(2)}(\mathcal{L}) = \{ST^{(2)}(L_1, L_2) \mid L_1, L_2 \in \mathcal{L}\}$. Wir zeigen nun, dass diese Menge für endliche Sprachen nicht lernbar ist.

Theorem 3 *Es gilt* $\mathcal{ST}^{(2)}(\mathcal{L}_f) \notin LimTxt$.

Beweis. Seien $\Sigma = \{a, b, c, d\}$, $L_1 = \{cd\}$ und $L_2 = \{ba, bb, bc, bd\}$. Wir werden beweisen, dass keine endliche Telltale-Menge von $ST^{(2)}(L_1, L_2)$ existiert, d.h. für jede endliche Menge $T \subseteq ST^{(2)}(L_1, L_2)$ existiert ein ST_2', so dass $T \subseteq ST_2'$ und $ST_2' \subset ST^{(2)}(L_1, L_2)$ gilt. Damit gilt aber nach Theorem 1 $\mathcal{ST}^{(2)}(\mathcal{L}_f) \notin LimTxt$. Zur besseren Übersichtlichkeit treffen wir für die restliche Beweisführung folgende Vereinbarung: $ST_2 = ST^{(2)}(L_1, L_2)$ und $ST = ST(L_2)$.
Für den Beweis nehmen wir das Gegenteil an: Sei $T \subseteq ST_2$ eine Telltale-Menge von ST_2, d.h. es existiert keine Menge ST_2', so dass $T \subseteq ST_2'$ und $ST_2' \subset ST_2$ gilt.
Angenommen, es gelte $m = max\{|w| \mid w \in T\}$ und es sei $ST_2' = ST^{(2)}(L_1', L_2')$ mit $L_1' = \{cdub \mid u \in \Sigma^* \wedge |u| \leq m \wedge b$ ist kein Teilwort von $u\}$ und $L_2' = \{a, b, c, d\}$. Sei außerdem $ST' = ST(L_2')$. Wir zeigen nun, dass Folgendes gilt:

Fall (1) $T \subseteq ST_2'$

Fall (2) $ST_2' \subset ST_2$

Fall (1): Sei $w \in T$. Aus $T \subseteq ST_2$ folgt $w \in ST_2$. Damit existieren $r_1, r_2 \in \Sigma^*$, $v_2 \in \Sigma^+$ und $l_2 \in L_1, l_2 \in L_2$ mit $w = r_1 l_1 r_2 l_2 \# v_2$, $r_2 l_2 \# v_2 \in ST$, $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1)$ und $(r_2, l_2, v_2) \in SkipTo(L_2)$. Dann gilt $l_1 r_2 l_2 \in \{cdu_1 bu_2 \mid u_1 \in \Sigma^* \wedge b$ ist keine Teilwort von $u_1 \wedge u_2 \in \Sigma\}$ und es existieren $l_1' \in \{cdub \mid u \in \Sigma^* \wedge b$ ist keine Teilwort von $u\}$, $r_2' = \varepsilon$ und $l_2' \in \Sigma$ mit $l_1' r_2' l_2' = r_1 l_1 r_2 l_2$.

Wir zeigen nun, dass $(r_1, l_1', r_2' l_2' v_2) \in SkipTo(L_1')$ gilt. Es sind $r_1 \in \Sigma^*$ und $r_2' l_2' v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Wegen $w \leq m$ gilt $l_1' \leq m$. Weil außerdem $l_1' \in \{cdub \mid u \in \Sigma^* \wedge b$ ist keine Teilwort von $u\}$ gilt, ist $l_1' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Da kein Element von L_1' ein Präfix besitzt, das auch Element von L_1' ist, kann auch kein $l' \in L_1'$ existieren mit $l' \sqsubset_p l_1'$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1)$ existieren keine $r' \in \Sigma^*$, $l' \in L_1$ mit $r' \sqsubset_p r_1 \wedge r' l' \sqsubseteq_p r_1 l_1 r_2 l_2 v_2$. Damit kann cd kein Teilwort von r_1 sein und es existieren auch keine $r' \in \Sigma^*$, $l' \in L_1'$ mit $r' \sqsubset_p r_1 \wedge r' l' \sqsubseteq_p r_1 l_1' r_2' l_2' v_2$ [Die Startbedingung ist erfüllt]. Da $(r_1, l_1', r_2' l_2' v_2)$ alle Bedingungen der Definition von $SkipTo(L_1')$ erfüllt, gilt $(r_1, l_1', r_2' l_2' v_2) \in SkipTo(L_1')$.

Wir zeigen weiter, dass $(r_2', l_2', v_2) \in SkipTo(L_2')$ gilt. Es sind $v_2 \in \Sigma^+$ und $r_2' = \varepsilon$ [Die Randbedingungen sind erfüllt]. Wegen $l_2' \in \Sigma$ ist $l_2' \in L_2'$ [Die Sprachbedingung ist erfüllt]. Da $l_2' \in \Sigma$ gilt, existiert kein Teilwort von l_2', das auch Element von L_2' ist. Somit gilt $\neg \exists l' \in L_2'$

mit $l' \sqsubset_p l'_2$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $r'_2 = \varepsilon$ existieren außerdem keine $r' \in \Sigma^*$, $l' \in L'_2$ mit $r' \sqsubset_p r'_2 \wedge r'l' \sqsubseteq_p r'_2 l'_2 v_2$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r'_2, l'_2, v_2) \in \mathit{SkipTo}(L'_2)$. Somit ist $r'_2 l'_2 \# v_2 \in ST'$. Da außerdem $(r_1, l'_1, r'_2 l'_2 v_2) \in \mathit{SkipTo}(L'_1)$ gilt, ist $w = r_1 l_1 r_2 l_2 \# v_2 = r_1 l'_1 r'_2 l'_2 \# v_2 \in ST'_2$ und wir haben bewiesen, dass $T \subseteq ST'_2$ gilt.

Fall (2): Sei $w' \in ST'_2$. Es existieren also $r'_1, r'_2 \in \Sigma^*$, $v'_2 \in \Sigma^+$ und $l'_1 \in L'_1, l'_2 \in L'_2$ mit $w' = r'_1 l'_1 r'_2 l'_2 \# v'_2$, $r'_2 l'_2 \# v'_2 \in ST'$, $(r'_1, l'_1, r'_2 l'_2 v'_2) \in \mathit{SkipTo}(L'_1)$ sowie $(r'_2, l'_2, v'_2) \in \mathit{SkipTo}(L'_2)$. Dann gilt $l'_1 r'_2 l'_2 \in \{cdu_1 bu_2 \mid u_1 \in \Sigma^* \wedge b \text{ ist keine Teilwort von } u_1 \wedge u_2 \in \Sigma\}$ und es existieren $l''_1 = cd$, $r''_2 \in \Sigma^*$ und $l''_2 \in \{bu \mid u \in \Sigma\}$ mit $l''_1 r''_2 l''_2 = l'_1 r'_2 l'_2$, so dass b kein Teilwort von r''_2 ist.

Wir beweisen, dass $(r'_1, l''_1, r''_2 l''_2 v'_2) \in \mathit{SkipTo}(L_1)$ gilt. Es sind $r'_1 \in \Sigma^*$ und $r''_2 l''_2 v'_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Wegen $l''_1 = cd$ gilt $l''_1 \in L_1$ [Die Sprachbedingung ist erfüllt]. Da L_1 nur ein Element besitzt, kann auch kein $l' \in L_1$ existieren mit $l' \sqsubset_p l''_1$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $(r'_1, l'_1, r'_2 l'_2 v'_2) \in \mathit{SkipTo}(L'_1)$ existieren keine $r' \in \Sigma^*$ und $l' \in L'_1$ mit $r' \sqsubset_p r'_1$ und $r'l' \sqsubseteq_p r'_1 l'_1 r'_2 l'_2 v'_2$. Daraus folgt, dass cd kein Teilwort von r'_1 ist. Damit existieren auch keine $r' \in \Sigma^*$, $l' \in L_1$ mit $r' \sqsubset_p r'_1 \wedge r'l' \sqsubseteq_p r'_1 l''_1 r''_2 l''_2 v'_2$ [Die Startbedingung ist erfüllt]. Da $(r'_1, l''_1, r''_2 l''_2 v'_2)$ alle Bedingungen der Definition von $\mathit{SkipTo}(L_1)$ erfüllt, gilt $(r_1, l''_1, r''_2 l''_2 v_2) \in \mathit{SkipTo}(L_1)$.

Wir zeigen weiter, dass $(r''_2, l''_2, v'_2) \in \mathit{SkipTo}(L'_2)$ gilt. Es sind $v'_2 \in \Sigma^+$ und $r''_2 \in \Sigma^*$ [Die Randbedingungen sind erfüllt]. Da $l''_2 \in \{bu \mid u \in \Sigma\}$ gilt, ist $l''_2 \in L_2$ [Die Sprachbedingung ist erfüllt]. Kein Element aus L_2 besitzt ein Teilwort, das auch Element von L_2 ist. Somit gilt $\neg\exists l' \in L_2$ mit $l' \sqsubset_p l''_2$ [Die Minimalitätsbedingung ist erfüllt]. b ist kein Teilwort von r''_2. Deshalb existieren keine $r' \in \Sigma^*$, $l' \in L_2$ mit $r' \sqsubset_p r''_2 \wedge r'l' \sqsubseteq_p r''_2 l''_2 v'_2$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r''_2, l''_2, v'_2) \in \mathit{SkipTo}(L_2)$. Somit ist $r''_2 l''_2 \# v'_2 \in ST$. Da außerdem $(r'_1, l''_1, r''_2 l''_2 v'_2) \in \mathit{SkipTo}(L_1)$ gilt, ist $w' = r'_1 l'_1 r'_2 l'_2 \# v'_2 = r'_1 l''_1 r''_2 l''_2 \# v'_2 \in ST_2$ und wir haben bewiesen, dass $ST'_2 \subseteq ST_2$ gilt.

Sei $w'' \in \{cdu_1 ba \# v_2 \mid v_2 \in \Sigma^+ \wedge u_1 \in \Sigma^* \wedge |u_1| > m \wedge cd \text{ und } b \text{ sind keine Teilworte von } u_1\}$. Offensichtlich ist $w'' \in ST_2$. Da $|u_1| > m$ gilt und cd kein Teilwort von u_1 ist, existiert kein Teilwort von w'', das Element von L'_1 ist. Damit gibt es auch keine Zerlegung, die Element von $\mathit{SkipTo}(L'_1)$ ist und es gilt $w'' \notin ST'_2$. Somit ist $ST_2 \backslash ST'_2 \neq \emptyset$. Aus $ST'_2 \subseteq ST_2$ und $ST_2 \backslash ST'_2 \neq \emptyset$ folgt $ST'_2 \subset ST_2$.

Wir haben gezeigt, dass für T ein ST'_2 existiert, so dass $T \subseteq ST'_2$ und $ST'_2 \subset ST_2$ gilt. Dies ist aber ein Widerspruch zu unserer Annahmen, dass T eine endliche Telltale-Menge ist. Da T beliebig ist, existiert keine endliche Telltale-Menge von $ST^{(2)}(L_1, L_2)$. Damit gilt $\mathcal{ST}^{(2)}(\mathcal{L}_f) \notin \mathit{LimTxt}$.

■

Lernbarkeit von Regeln mit mehr als zwei *SkipTo*-Operationen

Sei $\mathcal{ST}^{(n)}(\mathcal{L})$ die Menge aller Regeln mit n aufeinanderfolgenden *SkipTo*-Operationen, d.h. $\mathcal{ST}^{(n)}(\mathcal{L}) = \{ST^{(n)}(L_1, ..., L_n) \mid L_1, ..., L_n \in \mathcal{L}\}$. Wir zeigen nun, dass diese Menge für endliche Sprachen und $n \geq 3$ nicht lernbar ist.

Theorem 4 *Sei $n \geq 3$. Dann gilt $\mathcal{ST}^{(n)}(\mathcal{L}_f) \notin LimTxt$.*

Beweis. Seien $\Sigma = \{a, b, c, d, e, f\}$, $L_1 = \{ab\}$, $L_2 = \{cd\}$, $L_3 = \{af, bf, cf, df, ef, ff\}$ und $L_j = \{ef\}$ für alle $4 \leq j \leq n$. Wir werden beweisen, dass keine endliche Telltale-Menge von $ST^{(n)}(L_1, ..., L_n)$ existiert, d.h. für jede endliche Menge $T \subseteq ST^{(n)}(L_1, ..., L_n)$ existiert ein ST'_n, so dass $T \subseteq ST'_n$ und $ST'_n \subset ST^{(n)}(L_1, ..., L_n)$ gilt. Damit gilt aber nach Theorem 1 $\mathcal{ST}^{(n)}(\mathcal{L}_f) \notin LimTxt$. Zur besseren Übersichtlichkeit treffen wir für den restlichen Beweis folgende Vereinbarung: $ST_n = ST^{(n)}(L_1, ..., L_n)$ und $ST_{n-(i-1)} = ST^{(n-(i-1))}(L_i, ..., L_n)$ für ein $i \in \{1, ..., n\}$.
Für den Beweis nehmen wir das Gegenteil an: Sei $T \subseteq ST_n$ eine Telltale-Menge von ST_n, d.h. es existiert keine Menge ST'_n, so dass $T \subseteq ST'_n$ und $ST'_n \subset ST_n$ gilt.
Angenommen, es gelte $m = max\{|w| \mid w \in T\}$ und es sei $ST'_n = ST^{(n)}(L'_1, ..., L'_n)$ mit $L'_1 = \{abucd \mid u \in \Sigma^* \wedge |u| \leq m \wedge cd \text{ ist kein Teilwort von } u\}$, $L'_2 = \{a, b, c, d, e, f\}$, $L'_3 = \{f\}$ sowie $L'_j = \{ef\}$ für alle $4 \leq j \leq n$. Sei außerdem $ST'_{n-(i-1)} = ST^{(n-(i-1))}(L'_i, ..., L'_n)$ für ein $i \in \{1, ..., n\}$. Wir zeigen nun, dass Folgendes gilt:

Fall (1) $T \subseteq ST'_n$

Fall (2) $ST'_n \subset ST_n$

Fall (1): Sei $w \in T$. Aus $T \subseteq ST_n$ folgt $w \in ST_n$. Damit existieren $r_1, ..., r_n \in \Sigma^*, v_n \in \Sigma^+$ und $l_1 \in L_1, ..., l_n \in L_n$ mit $w = r_1 l_1 ... r_n l_n \# v_n$, $r_i l_i ... r_n l_n \# v_n \in ST_{n-(i-1)}$ und $(r_i, l_i, r_{i+1} ... l_n v_n) \in SkipTo(L_i)$ für alle $1 \leq i \leq n$. Zuerst zeigen wir, dass $(r_1, l_1 r_2 l_2, r_3 l_3 ... r_n l_n v_n) \in SkipTo(L'_1)$ gilt. Es gilt $r_1 \in \Sigma^*$ und $r_3 ... l_n v_n \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Aus der Definition von L_1 und L_2 folgt, dass $l_1 = ab$ und $l_2 = cd$ gilt. Weiterhin gilt $r_2 < m$ und wegen $(r_2, l_2, r_3 l_3 ... r_n l_n v_n) \in SkipTo(L_2)$ ist cd kein Teilwort von r_2. Damit gilt $l_1 r_2 l_2 = abr_2cd \in L'_1$ [Die Sprachbedingung ist erfüllt]. Kein Element von L'_1 besitzt ein Präfix, das auch Element von L'_1 ist. Dann gilt aber auch $\neg\exists l'_1 \in L'_1$ mit $l'_1 \sqsubset_p l_1 r_2 l_2$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $(r_1, l_1, r_2 l_2 ... r_n l_n v_n) \in SkipTo(L_1)$ ist ab kein Teilwort von r_1 und damit existieren keine $r'_1 \in \Sigma^*$, $l'_1 \in L'_1$ mit $r'_1 \sqsubset_p r_1 \wedge r'_1 l'_1 \sqsubseteq_p r_1 l_1 ... r_n l_n v_n$ [Die Startbedingung ist erfüllt]. Da $(r_1, l_1 r_2 l_2, r_3 ... l_n v_n)$ alle Bedingungen der Definition von $SkipTo(L'_1)$ erfüllt gilt $(r_1, l_1 r_2 l_2, r_3 ... l_n v_n) \in SkipTo(L'_1)$.

Wegen $(r_3, l_3, r_4 l_4 ... r_n l_n v_n) \in SkipTo(L_3)$ ist $r_3 l_3 \in \{u_1 u_2 f \mid u_1 \in \Sigma \wedge u_2 \in \Sigma^* \wedge f \text{ ist kein Teilwort von } u_2\}$. Damit existieren $l'_2 \in L'_2$, $r'_3 \in \Sigma^*$ und $l'_3 = f \in L'_3$, so dass f kein Teilwort von r'_3 ist und $l'_2 r'_3 l'_3 = r_3 l_3$ gilt. Wir zeigen nun, dass $(r'_2, l'_2, r'_3 l'_3 r_4 l_4 ... r_n l_n v_n) \in SkipTo(L'_2)$ mit $r'_2 = \varepsilon$

gilt. Es sind $r'_3l'_3r_4...l_nv_n \in \Sigma^+$ und $r'_2 = \varepsilon$ [Die Randbedingungen sind erfüllt] sowie $l'_2 \in L'_2$ [Die Sprachbedingung ist erfüllt]. Da $l'_2 \in \Sigma$ gilt, existiert kein Teilwort von l'_2, das Element von L'_2 ist. Somit gilt $\neg\exists l''_2 \in L'_2$ mit $l''_2 \sqsubset_p l'_2$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $r'_2 = \varepsilon$ existieren außerdem keine $r''_2 \in \Sigma^*$, $l''_2 \in L'_2$ mit $r''_2 \sqsubset_p r'_2 \wedge r''_2l''_2 \sqsubseteq_p r'_2l'_2r'_3l'_3r_4l_4...r_nl_nv_n$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(\varepsilon, l'_2, r'_3l'_3r_4l_4...r_nl_nv_n) \in SkipTo(L'_2)$. Wir zeigen weiter, dass $(r'_3, l'_3, r_4l_4...r_nl_nv_n) \in SkipTo(L'_3)$ gilt. Es sind $r'_3 \in \Sigma^*$ und $r_4...l_nv_n \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Außerdem gilt $l'_3 = f \in L'_3$ [Die Sprachbedingung ist erfüllt]. Damit existiert aber auch kein $l'''_3 \in L'_3$ mit $l'''_3 \sqsubset_p l'_3$ [Die Minimalitätsbedingung ist erfüllt]. Da f kein Teilwort von r'_3 ist, existieren auch keine $r''_3 \in \Sigma^*$, $l''_3 \in L'_3$ mit $r''_3 \sqsubset_p r'_3 \wedge r''_3l''_3 \sqsubseteq_p r'_3l'_3r_4l_4...r_nl_nv_n$ [Die Startbedingung ist erfüllt]. $(r'_3, l'_3, r_4l_4...r_nl_nv_n)$ erfüllt alle Bedingungen der Definition von $SkipTo(L'_3)$ und es gilt $(r'_3, l'_3, r_4l_4...r_nl_nv_n) \in SkipTo(L'_3)$.

Für alle $4 \leq j \leq n$ gilt $L_j = L'_j$. Daraus folgt $(r_j, l_j, r_{j+1}l_{j+1}...r_nl_nv_n) \in SkipTo(L'_j)$. Somit gilt $r'_2l'_2r'_3l'_3r_4...l_n\#v_n \in ST'_{n-1}$, $r'_3l'_3r_4...l_n\#v_n \in ST'_{n-2}$ und $r_jl_j...r_nl_n\#v_n \in ST'_{n-(j-1)}$ für alle $4 \leq j \leq n$. Da außerdem $(r_1, l_1r_2l_2, r_3...l_nv_n) \in SkipTo(L'_1)$ gilt, ist $w = r_1l_1...r_nl_n\#v_n = r_1l_1r_2l_2l'_2r'_3l'_3r_4l_4...r_nl_n\#v_n \in ST'_n$ und wir haben bewiesen, dass $T \subseteq ST'_n$ gilt.

Fall (2): Sei $w' \in ST'_n$. Es existieren also $r'_1, ..., r'_n \in \Sigma^*, v'_n \in \Sigma^+$ und $l'_1 \in L'_1, ..., l'_n \in L'_n$ mit $w' = r'_1l'_1...r'_nl'_n\#v'_n$, $r'_il'_i...r'_nl'_n\#v'_n \in ST'_{n-(i-1)}$ sowie $(r'_i, l'_i, r'_{i+1}l'_{i+1}...r'_nl'_nv'_n) \in SkipTo(L'_i)$ für alle $1 \leq i \leq n$. Wegen $(r'_1, l'_1, r'_2l'_2...r'_nl'_nv'_n) \in SkipTo(L'_1)$ gilt $r'_1l'_1 \in \{u_1abu_2cd \mid u_1 \in \Sigma^* \wedge ab$ ist keine Teilwort von $u_1 \wedge u_2 \in \Sigma^* \wedge cd$ ist kein Teilwort von $u_2\}$. Damit existieren $r''_1, r''_2 \in \Sigma^*, l''_1 = ab$ und $l''_2 = cd$, so dass $r''_1l''_1r''_2l''_2 = r'_1l'_1$ gilt, ab kein Teilwort von r''_1 und cd kein Teilwort von r''_2 ist. Zuerst zeigen wir, dass $(r''_1, l''_1, r''_2l''_2r'_2l'_2...r'_nl'_nv'_n) \in SkipTo(L_1)$ gilt. Es sind $r''_1 \in \Sigma^*$ und $r''_2l''_2r'_2...l'_nv'_n \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l''_1 = ab \in L_1$ [Die Sprachbedingung ist erfüllt]. Es existiert nur ein Element in L_1. Damit gilt $\neg\exists l' \in L_1$ mit $l' \sqsubset_p l''_1$ [Die Minimalitätsbedingung ist erfüllt]. Da ab kein Teilwort von r''_1 ist, existieren keine $r' \in \Sigma^*, l' \in L_1$ mit $r' \sqsubset_p r''_1 \wedge r'l' \sqsubseteq_p r''_1l''_1r''_2l''_2r'_2l'_2...r'_nl'_nv'_n$ [Die Startbedingung ist erfüllt]. Alle Bedingungen sind erfüllt und somit gilt $(r''_1, l''_1, r''_2l''_2r'_2l'_2...r'_nl'_nv'_n) \in SkipTo(L_1)$.

Wir beweisen weiter, dass $(r''_2, l''_2, r'_2l'_2...r'_nl'_nv'_n) \in SkipTo(L_2)$ gilt. Es sind $r''_2 \in \Sigma^*$ und $r'_2...l'_nv'_n \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l''_2 = cd \in L_2$ [Die Sprachbedingung ist erfüllt]. Es existiert nur ein Element in L_2. Damit gilt $\neg\exists l' \in L_2$ mit $l' \sqsubset_p l''_2$ [Die Minimalitätsbedingung ist erfüllt]. Da cd kein Teilwort von r''_2 ist, existieren keine $r' \in \Sigma^*, l' \in L_2$ mit $r' \sqsubset_p r''_2 \wedge r'l' \sqsubseteq_p r''_2l''_2r'_2l'_2...r'_nl'_nv'_n$ [Die Startbedingung ist erfüllt]. Alle Bedingungen sind erfüllt und somit gilt $(r''_2, l''_2, r'_2l'_2...r'_nl'_nv'_n) \in SkipTo(L_2)$.

Aus $(r'_2, l'_2, r'_3l'_3...r'_nl'_nv'_n) \in SkipTo(L'_2)$ und $(r'_3, l'_3, r'_4l'_4...r'_nl'_nv'_n) \in SkipTo(L'_3)$ folgt $r'_2 = \varepsilon$, $l'_2 \in \Sigma$, f ist kein Teilwort von r'_3 sowie $l'_3 = f$ und damit $r'_2l'_2r'_3l'_3 \in \{u_1u_2f \mid u_1 \in \Sigma \wedge u_2 \in \Sigma^* \wedge f$ ist kein Teilwort von $u_2\}$. Somit existieren $r'''_3 \in \Sigma^*$ und $l'''_3 \in L_3$, so dass $r'''_3l'''_3 = r'_2l'_2r'_3l'_3$ gilt. Wir zeigen nun, dass $(r''_3, l''_3, r'_4l'_4...r'_nl'_nv'_n) \in SkipTo(L_3)$ gilt. Es sind $r''_3 \in \Sigma^*$ und $r'_4...l'_nv'_n \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l''_3 \in L_3$ [Die Sprachbedingung ist erfüllt]. In L_3 existiert

kein Element, das ein Teilwort besitzt, das auch Element von L_3 ist. Damit gilt $\neg\exists l' \in L_3$ mit $l' \sqsubset_p l_3''$ [Die Minimalitätsbedingung ist erfüllt]. Weil $r_3''l_3'' \in \{u_1u_2f \mid u_1 \in \Sigma \wedge u_2 \in \Sigma^* \wedge f$ ist kein Teilwort von $u_2\}$ gilt, existieren keine $r' \in \Sigma^*$, $l' \in L_3$ mit $r' \sqsubset_p r_3'' \wedge r'l' \sqsubseteq_p r_3''l_3''r_4'l_4'...r_n'l_n'v_n'$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, haben wir bewiesen, dass $(r_3'', l_3'', r_4'l_4'...r_n'l_n'v_n') \in \mathit{SkipTo}(L_3)$ gilt.

Für alle $4 \leq j \leq n$ gilt $L_j = L_j'$. Daraus folgt $(r_j', l_j', r_{j+1}'l_{j+1}'...r_n'l_n'v_n') \in \mathit{SkipTo}(L_j)$. Somit gilt $r_2''l_2''r_3''l_3''r_4'...l_n'\#v_n' \in ST_{n-1}$, $r_3''l_3''r_4'...l_n'\#v_n' \in ST_{n-2}$ und $r_j'l_j'...r_n'l_n'\#v_n' \in ST_{n-(j-1)}$ für alle $4 \leq j \leq n$. Da außerdem $(r_1'', l_1'', r_2''l_2''r_3''l_3''r_4'...l_n'v_n') \in \mathit{SkipTo}(L_1)$ gilt, ist $w = r_1'l_1'...r_n'l_n'\#v_n' = r_1''l_1''r_2''l_2''r_3''l_3''r_4'l_4'...r_n'l_n'\#v_n' \in ST_n$ und wir haben bewiesen, dass $ST' \subseteq ST_n$ gilt.

Sei $w'' \in \{abu_1cdafu_2^{n-3}\#v_3 \mid v_3 \in \Sigma^+ \wedge u_1 \in \Sigma^* \wedge |u_1| > m \wedge cd$ ist kein Teilwort von $u_1 \wedge u_2 = ef\}$. Offensichtlich ist $w'' \in ST_n$. Da $|u_1| > m$ gilt, existiert kein Teilwort von w'', das Element von L_1' ist. Damit gibt es auch keine Zerlegung, die Element von $\mathit{SkipTo}(L_1')$ ist und es gilt $w'' \notin ST_n'$. Somit ist $ST_n \backslash ST_n' \neq \emptyset$. Aus $ST_n' \subseteq ST_n$ und $ST_n \backslash ST_n' \neq \emptyset$ folgt $ST_n' \subset ST_n$.

Wir haben gezeigt, dass für T ein ST_n' existiert, so dass $T \subseteq ST_n'$ und $ST_n' \subset ST_n$ gilt. Dies ist aber ein Widerspruch zu unserer Annahmen, dass T eine endliche Telltale-Menge ist. Da T beliebig ist, existiert keine endliche Telltale-Menge von $ST^{(n)}(L_1, ..., L_n)$. Damit gilt $\mathcal{ST}^{(n)}(\mathcal{L}_f) \notin \mathit{LimTxt}$.

■

6.2 Lernbarkeit von Regeln, die eine *SkipUntil*-Operation enthalten

Alle im folgenden Abschnitt untersuchten Regeln enthalten am Ende eine *SkipUntil*-Operation. Im Gegensatz zu den Regeln, die nur *SkipTo*-Operationen besitzen, wird vor dem gefundenen Argument von *SkipUntil* gehalten.

Lernbarkeit von Regeln mit einer *SkipUntil*-Operation

Sei $\mathcal{SU}(\mathcal{L})$ die Menge aller Regeln, in denen eine *SkipUntil*-Operation durchgeführt wird, d.h. $\mathcal{SU}(\mathcal{L}) = \{SU(L) \mid L \in \mathcal{L}\}$. Wir werden nun zeigen, dass diese Menge für endliche Sprachen lernbar ist.

Theorem 5 $\mathcal{SU}(\mathcal{L}_f) \in LimTxt$.

Beweis. Wir wissen, dass eine Sprache aus Text lernbar ist, wenn wir eine Telltale-Menge für diese Sprache bilden können. Sei $L \in \mathcal{L}_f$. Wir definieren die Menge $T_{SU(L)} = \{\#l \mid (\varepsilon, l, \varepsilon) \in \mathit{SkipUntil}(L)\}$.
Wir beweisen nun, dass $T_{SU(L)}$ eine Telltale-Menge von $SU(L)$ ist. Für den Beweis nehmen wir das Gegenteil an, d.h. seien $L, L' \in \mathcal{L}_f$ mit $T_{SU(L)} \subseteq SU(L') \subset SU(L)$.
Sei $w \in SU(L) \backslash SU(L')$ mit $w = r\#lv$ und $(r, l, v) \in \mathit{SkipUntil}(L)$. Somit gilt $w \notin SU(L')$ und es existiert keine Zerlegung für w, die die Bedingungen der Definition von $\mathit{SkipUntil}(L')$ erfüllt. Insbesondere gilt auch $(r, l, v) \notin \mathit{SkipUntil}(L')$.
Die Randbedingungen sind wegen $(r, l, v) \in \mathit{SkipUntil}(L)$ immer erfüllt. Deshalb folgt aus der Definition, dass mindestens einer der folgenden Fälle gelten muss:

Fall (5.1) $l \notin L'$ [Die Sprachbedingung ist verletzt]

Fall (5.2) $\exists r' \in \Sigma^*, l' \in L'$ mit $r' \sqsubset_p r \wedge r'l' \sqsubseteq_p rlv$ [Die Startbedingung ist verletzt]

Fall (5.1): Angenommen, $l \notin L'$ gilt. Wir nehmen außerdem o.B.d.A. an, dass (r, l, v) die Startbedingung der Definition von $\mathit{SkipUntil}(L')$ erfüllt. Wir betrachten das Wort $w' = \#l$. Es gilt $w' \in T_{SU(L)}$. Aus $T_{SU(L)} \subseteq SU(L')$ folgt $w' \in SU(L')$. Da $l \notin L'$ gilt, muss eine andere Zerlegung für w' existieren, die die Bedingungen der Definition von $SU(L)$ erfüllt. Angenommen $\#l'v'$ mit $l = l'v'$ sei diese Zerlegung. Wir betrachten das Wort $w = r\#l'v'v$ und zeigen, dass $(r, l', v'v) \in \mathit{SkipUntil}(L')$ gilt. Es sind sowohl $r \in \Sigma^*$ und $v'v \in \Sigma^*$ [Die Randbedingungen sind erfüllt] als auch $l' \in L'$ [Die Sprachbedingung ist erfüllt]. Da (r, l, v) die Sprachbedingung der Definition von $\mathit{SkipUntil}(L')$ erfüllt, existieren auch keine $r' \in \Sigma^*$ und $l'' \in L'$ mit $r' \sqsubset_p r \wedge r'l'' \sqsubseteq_p rl'v'v$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r, l', v'v) \in \mathit{SkipUntil}(L')$ und damit auch $w \in SU(L')$. Dies ist aber ein Widerspruch zu unserer Annahme, dass $w \in SU(L) \backslash SU(L')$ gilt. Somit kann Fall (5.1) nicht auftreten.

Fall (5.2): Angenommen, es existieren ein $r' \in \Sigma^*$ und ein $l' \in L'$ mit $r' \sqsubset_p r \wedge r'l' \sqsubseteq_p rlv$. Aus der Menge der $r'l'$, die diese Bedingung erfüllen, betrachten wir das Paar mit dem kürzesten r'. Wir beweisen nun, dass $(r', l', v') \in \mathit{SkipUntil}(L')$ gilt mit $v' = \varepsilon$. Es sind $r' \in \Sigma^*$ und $v' \in \Sigma^*$ [Die Randbedingungen sind erfüllt] sowie $l' \in L'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r' und l' existieren keine $r'' \in \Sigma^*$ und $l'' \in L'$, so dass $r'' \sqsubset_p r' \wedge r''l'' \sqsubseteq_p r'l'v'$ gilt [Die Startbedingung ist erfüllt]. Da (r', l', v') alle Bedingungen von $\mathit{SkipUntil}(L')$ erfüllt, gilt $(r', l', v') \in \mathit{SkipUntil}(L')$ und damit $w' = r'\#l' \in SU(L')$. Angenommen, $w' \in SU(L)$ gilt. Dann existiert eine Zerlegung $w' = r'\#l''v''$ mit $(r', l'', v'') \in \mathit{SkipUntil}(L)$. Es gilt $r' \sqsubset_p r$. Weiterhin ist $l'' \in L$ und wegen $l'' \sqsubseteq_p l'$ gilt auch $r'l'' \sqsubseteq_p rlv$. Damit kann (r, l, v) die Startbedingung der Definition von $\mathit{SkipUntil}(L)$ nicht erfüllen und es gilt $(r, l, v) \notin \mathit{SkipUntil}(L)$. Dies ist ein Widerspruch zu unserer Annahme, dass $(r, l, v) \in \mathit{SkipUntil}(L)$ gilt. Also gilt $w' \notin SU(L)$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $SU(L') \subset SU(L)$ gilt. Damit haben wir gezeigt, dass Fall (5.2) nicht auftreten kann.

Da wir sowohl Fall (5.1) als auch Fall (5.2) widerlegt haben, gilt $(r, l, v) \in \mathit{SkipUntil}(L')$ und damit auch $w \in SU(L')$. Dies ist ein Widerpruch zu unserer Annahme, dass $w \in SU(L) \backslash SU(L')$ gilt. Somit existiert kein $w \in SU(L) \backslash SU(L')$, was wiederum ein Widerspruch zu unserer Annahme ist, dass $T_{SU(L)} \subseteq SU(L') \subset SU(L)$ gilt. Aus diesem Grund ist $T_{SU(L)}$ eine Telltale-Menge von $SU(L)$. Es gilt also $SU(L) \in LimTxt$. Da $SU(L)$ beliebig ist, gilt $\mathcal{SU}(\mathcal{L}) \in LimTxt$. ∎

Lernbarkeit von Regeln mit einer *SkipTo*-Operation und einer *SkipUntil*-Operation

Sei $\mathcal{STSU}(\mathcal{L})$ die Menge aller Regeln mit einer *SkipTo*- und einer *SkipUntil*-Operation, d.h. $\mathcal{STSU}(\mathcal{L}) = \{STSU(L_1, L_2) \mid L_1, L_2 \in \mathcal{L}\}$. Wir zeigen nun, dass diese Menge für endliche Sprachen lernbar ist.

Theorem 6 $\mathcal{STSU}(\mathcal{L}_f) \in LimTxt$.

Beweis. Wir wissen, dass eine Sprache aus Text lernbar ist, wenn wir eine Telltale-Menge für diese Sprache bilden können. Seien $L_1, L_2 \in \mathcal{L}_f$. Wir definieren die Menge $T_{STSU(L_1,L_2)} = \{l_1\#l_2 \mid (\varepsilon, l_1, l_2) \in SkipTo(L_1), \#l_2 \in SU(L_2)$ und $(\varepsilon, l_2, \varepsilon) \in SkipUntil(L_2)\}$.
Seien $L_1, L_2 \in \mathcal{L}_f$. Wir beweisen nun, dass $T_{STSU(L_1,L_2)}$ eine Telltale-Menge von $STSU(L_1, L_2)$ ist. Für den Beweis nehmen wir das Gegenteil an, d.h. seien $L_1', L_2' \in \mathcal{L}_f$ und $T_{STSU(L_1,L_2)} \subseteq STSU(L_1', L_2') \subset STSU(L_1, L_2)$. Sei $w \in STSU(L_1, L_2) \backslash STSU(L_1', L_2')$. Daraus folgt, dass eine Zerlegung $w = r_1 l_1 r_2 \# l_2 v_2$ existiert, so dass $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1)$, $r_2\#l_2v_2 \in SU(L_2)$ und $(r_2, l_2, v_2) \in SkipTo(L_2)$ gilt. Es ist außerdem $w \notin STSU(L_1', L_2')$. Für jede Zerlegung von w sind somit entweder die Bedingungen der Definition von $SkipTo(L_1')$ oder die Bedingungen der Definition von $SU(L_2')$ nicht erfüllt. Insbesondere gilt einer der folgenden Fälle:

Fall (6.1) $(r_1, l_1, r_2 l_2 v_2) \notin SkipTo(L_1')$

Fall (6.2) $r_2\#l_2v_2 \notin SU(L_2')$

In Lemma 5 bzw. Lemma 4 beweisen wir, dass weder Fall (6.1) noch Fall (6.2) gelten kann. Damit gilt aber $w \in STSU(L_1', L_2')$. Dies ist ein Widerspruch zu unserer Annahme, dass $w \in STSU(L_1, L_2) \backslash STSU(L_1', L_2')$ gilt, was wiederum ein Widerspruch zu unserer Annahme ist, dass $T_{STSU(L_1,L_2)} \subseteq STSU(L_1', L_2') \subset STSU(L_1, L_2)$ gilt. Aus diesem Grund ist $T_{STSU(L_1,L_2)}$ eine Telltale-Menge von $STSU(L_1, L_2)$. Es gilt also $STSU(L_1, L_2) \in LimTxt$. Da $STSU(L_1, L_2)$ beliebig ist, gilt $\mathcal{STSU}(\mathcal{L}_f) \in LimTxt$.

■

Zur besseren Übersichtlichkeit der Beweise treffen wir für die restlichen Lemmata in diesem Kapitel folgende Vereinbarungen. Seien $L_1, L_2, L_1', L_2' \in \mathcal{L}_f$. Sei $T = T_{STSU(L_1,L_2)} = \{l_1\#l_2 \mid (\varepsilon, l_1, l_2) \in SkipTo(L_1), \#l_2 \in SU(L_2)\}$. Weiterhin gelte $STSU = STSU(L_1, L_2)$, $STSU' = STSU(L_1', L_2')$, $SU = SU(L_2)$ und $SU' = SU(L_2')$.

Wir widerlegen zuerst den Fall (6.2).

Lemma 4 *Seien $T \subseteq STSU' \subset STSU$ und $w \in STSU \backslash STSU'$ mit $w = r_1 l_1 r_2 \# l_2 v_2$. Weiterhin seien $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1)$, $r_2\#l_2v_2 \in SU$ und $(r_2, l_2, v_2) \in SkipTo(L_2)$. Dann gilt $r_2\#l_2v_2 \in SU'$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. sei $r_2\#l_2v_2 \notin SU'$. Daraus folgt, dass keine Zerlegung von $r_2\#l_2v_2$ existiert, die die Bedingungen der Definition von $\mathit{SkipUntil}(L_2')$ erfüllt. Insbesondere gilt auch $(r_2, l_2, v_2) \notin \mathit{SkipUntil}(L_2')$. Daraus folgt, dass eine der Bedingungen der Definition von $\mathit{SkipUntil}(L_2')$ verletzt sein muss. Die Randbedingungen sind wegen $(r_2, l_2, v_2) \in \mathit{SkipUntil}(L_2)$ immer erfüllt. Deshalb folgt aus der Definition, dass mindestens eine der folgenden Fälle gelten muss:

Fall (1) $l_2 \notin L_2'$ [Die Sprachbedingung ist verletzt]

Fall (2) $\exists r_2' \in \Sigma^*, l_2' \in L_2'$ mit $r_2' \sqsubset_p r_2 \wedge r_2'l_2' \sqsubseteq_p r_2l_2v_2$ [Die Startbedingung ist verletzt]

Fall (1): Angenommen, $l_2 \notin L_2'$ gilt. Wir betrachten das Wort $w' = l_1\#l_2$. Es gilt $w' \in T$. Aus $T \subseteq STSU'$ folgt $w' \in STSU'$. Da $l_2 \notin L_2'$ gilt, muss eine andere Zerlegung für w' existieren, die die Bedingungen von $STSU'$ erfüllt. Angenommen $r_1'l_1'r_2'\#l_2'v_2'$ mit $r_1', r_2' \in \Sigma^*$, $l_1' \in L_1'$, $l_2' \in L_2'$ und $v_2' \in \Sigma^*$ sei diese Zerlegung. Da wir nachfolgend beweisen werden, dass (r_2, l_2, v_2) die Startbedingung unabhängig von Fall (1) erfüllt, nehmen wir o.B.d.A. an, dass $l_2 \notin L_2'$ der einzige Grund ist, warum $(r_2, l_2, v_2) \notin \mathit{SkipUntil}(L_2')$ gilt. Wir beweisen nun, dass $(r_2, l_2', v_2'v_2) \in \mathit{SkipUntil}(L_2')$ gilt. Es sind $r_2 \in \Sigma^*$ und $v_2'v_2 \in \Sigma^*$ [Die Randbedingungen sind erfüllt] sowie $l_2' \in L_2'$ [Die Sprachbedingung ist erfüllt]. Da (r_2, l_2, v_2) die Startbedingung von $\mathit{SkipUntil}(L_2')$ erfüllt, existieren auch keine $r_2'' \in \Sigma^*$ und $l_2'' \in L_2'$ mit $r_2'' \sqsubset_p r_2 \wedge r_2''l_2'' \sqsubseteq_p r_2l_2v_2 = r_2l_2'v_2'v_2$ [Die Startbedingung ist erfüllt]. Daraus folgt $(r_2, l_2', v_2'v_2) \in \mathit{SkipUntil}(L_2')$ und $r_2\#l_2'v_2'v_2 = r_2\#l_2v_2 \in SU'$. Dies ist aber ein Widerspruch zu unserer Annahme, dass $r_2\#l_2v_2 \notin SU'$ gilt. Somit muss die Sprachbedingung erfüllt sein.

Fall (2): Angenommen, es existieren ein $r_2' \in \Sigma^*$ und ein $l_2' \in L_2'$ mit $r_2' \sqsubset_p r_2 \wedge r_2'l_2' \sqsubseteq_p r_2l_2v_2$. Aus der Menge der $r_2'l_2'$, die diese Bedingung erfüllen, betrachten wir das Paar mit dem kürzesten r_2'. Sei $l' \in L_1'$, so dass kein $l'' \in L_1'$ existiert mit $l'' \sqsubset_p l'$. Wir beweisen nun, dass $w' \in STSU'$ gilt mit $w' = r'l'r_2'\#l_2'v_2'$, $r' = \varepsilon$, $v_2' \in \Sigma^*$ und $r_2'l_2'v_2' = r_2l_2v_2$. Zuerst zeigen wir, dass $(r', l', r_2'l_2'v_2') \in \mathit{SkipTo}(L_1')$ ist. Es sind $r' = \varepsilon$ und $r_2'l_2'v_2' \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l' \in L'$ [Die Sprachbedingung ist erfüllt]. Aufgrund unserer Wahl von l' existiert kein $l'' \in L_1'$ mit $l'' \sqsubset_p l'$ [Die Minimalitätsbedingung ist erfüllt]. Da $r' = \varepsilon$ gilt, existieren keine $r'' \in \Sigma^*$ und $l'' \in L_1'$ mit $r'' \sqsubset_p r' \wedge r''l'' \sqsubseteq_p r'l'r_2'l_2'v_2'$ [Die Startbedingung ist erfüllt]. Daraus folgt $(r', l', r_2'l_2'v_2') \in \mathit{SkipTo}(L_1')$.

Wir zeigen weiter, dass $(r_2', l_2', v_2') \in \mathit{SkipUntil}(L_2')$ gilt. Es sind $r_2' \in \Sigma^*$ und $v_2' \in \Sigma^*$ [Die Randbedingungen sind erfüllt] sowie $l_2' \in L_2'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_2' und l_2', existieren keine $r_2'' \in \Sigma^*$ und $l_2'' \in L_2'$ mit $r_2'' \sqsubset_p r_2' \wedge r_2''l_2'' \sqsubseteq_p r_2'l_2'v_2'$ [Die Startbedingung ist erfüllt]. Daraus folgt $(r_2', l_2', v_2') \in \mathit{SkipUntil}(L_2')$, $r_2'\#l_2'v_2' \in SU'$ und $w' \in STSU'$. Angenommen, es gilt $w' \in STSU$. Dann existiert eine Zerlegung $w' = r_1''l_1''r_2''\#l_2''v_2''$ mit $(r_1'', l_1'', r_2''l_2''v_2'') \in \mathit{SkipTo}(L_1)$, $r_2''\#l_2''v_2'' \in SU$ und $(r_2'', l_2'', v_2'') \in \mathit{SkipUntil}(L_2)$. Wegen $(r_2'', l_2'', v_2'') \in \mathit{SkipUntil}(L_2)$ ist $l_2'' \in L_2$. Weiterhin gilt $l_2''v_2'' = l_2'v_2'$. Daraus folgt, dass $l_2v_2 \sqsubset_s l_2''v_2''$ und $l_2''v_2'' \sqsubseteq_s r_2l_2v_2$ gilt. Dann existiert aber ein $r'' \in \Sigma^*$ mit $r'' \sqsubset_p r_2 \wedge r''l_2'' \sqsubseteq_p r_2l_2v_2$. Somit

kann (r_2, l_2, v_2) die Startbedingung von $SkipUntil(L_2)$ nicht erfüllen. Dies ist ein Widerspruch zu unserer Annahme, dass $(r_2, l_2, v_2) \in SkipUntil(L_2)$ gilt. Also ist $w' \notin STSU$. Das ist wiederum ein Widerspruch zu unserer Annahme, dass $STSU' \subset STSU$ gilt. Damit haben wir gezeigt, dass die Startbedingung erfüllt sein muss.

Wir haben Fall (1) und Fall (2) widerlegt. Daraus folgt $(r_2, l_2, v_2) \in SkipUntil(L_2')$ und somit gilt $r_2 \# l_2 v_2 \in SU'$.

■

Das nächste Lemma widerlegt Fall (6.1). In Lemma 4 haben wir gezeigt, dass Fall (6.2) unabhängig von Fall (6.1) nie gelten kann. Deshalb können wir für den Beweis von Lemma 5 annehmen, dass Fall (6.2) gilt.

Lemma 5 *Seien $T \subseteq STSU' \subset STSU$ und $w \in STSU \backslash STSU'$ mit $w = r_1 l_1 r_2 \# l_2 v_2$. Weiterhin seien $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1)$, $r_2 \# l_2 v_2 \in SU$ und $r_2 \# l_2 v_2 \in SU'$. Dann gilt $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1')$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. sei $(r_1, l_1, r_2 l_2 v_2) \notin SkipTo(L_1')$. Die Randbedingungen sind wegen $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1)$ immer erfüllt. Deshalb muss einer der folgenden Fälle gelten:

Fall (1) $l_1 \notin L_1'$ [Die Sprachbedingung ist verletzt]

Fall (2) $\exists l_1' \in L_1'$ mit $l_1' \sqsubset_p l_1$ [Die Minimalitätsbedingung ist verletzt]

Fall (3) $\exists r_1' \in \Sigma^*, l_1' \in L_1'$ mit $r_1' \sqsubset_p r_1 \wedge r_1' l_1' \sqsubseteq_p r_1 l_1 r_2 l_2 v_2$ [Die Startbedingung ist verletzt]

In den nachfolgenden Lemmata beweisen wir, dass alle drei Fälle nicht gelten können. Siehe Lemma 7 für Fall (1), Lemma 8 für Fall (2) und Lemma 9 für Fall (3). Damit gilt aber $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1')$.

■

Wir zeigen nun ein Lemma, das uns bei der Beweisführung von Lemma 7 hilft.

Lemma 6 *Seien $T \subseteq STSU' \subset STSU$ und $w \in STSU \backslash STSU'$ mit $w = r_1 l_1 r_2 \# l_2 v_2$. Es gelte $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1)$, $r_2 \# l_2 v_2 \in SU$, $r_2 \# l_2 v_2 \in SU'$ und $l_1 \notin L_1'$. Weiterhin nehmen wir an, dass $r', r'' \in \Sigma^*$ und $l' \in L_1'$ existieren mit $r' l' r'' = l_1$. Dann existieren keine $r_1' \in \Sigma^*$ und $l_1' \in L_1'$ mit $r_1' \sqsubset_p r' \wedge r_1' l_1' \sqsubseteq_p l_1 r_2 l_2 v_2 \wedge l_1 r_2 \sqsubset_p r_1' l_1'$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. seien $r_1' \in \Sigma^*$ und $l_1' \in L_1'$ mit $r_1' \sqsubset_p r' \wedge r_1' l_1' \sqsubseteq_p l_1 r_2 l_2 v_2 \wedge l_1 r_2 \sqsubset_p r_1' l_1'$. Aus der Menge der $r_1' l_1'$, für die dies gilt, betrachten

wir das Paar mit dem kleinsten r_1' und einem l_1', so dass kein $l_1'' \in L_1'$ existiert mit $l_1'' \sqsubset_p l_1'$. Dann gibt es folgende Möglichkeiten:

Fall (1): Es existieren r_2' und $l_2' \in L_2'$ mit $r_1'l_1'r_2'l_2' \sqsubseteq_p l_1r_2l_2v_2$.

Angenommen, Fall (1) gilt. Aus der Menge der $r_2'l_2'$ für die diese Bedingung gilt, betrachten wir das Paar mit dem kürzesten r_2'. Dann existiert ein $v_2' \in \Sigma^*$ mit $r_1'l_1'r_2'l_2'v_2' = l_1r_2l_2v_2$. Wir zeigen zuerst, dass $(r_1', l_1', r_2'l_2'v_2') \in \mathit{SkipTo}(L_1')$ gilt. Es sind $r_1' \in \Sigma^*$ und $r_2'l_2'v_2' \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l_1' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_1' und l_1', existieren kein $l_1'' \in L_1'$ mit $l_1'' \sqsubset_p l_1'$ [Die Minimalitätsbedingung ist erfüllt] und keine $r_1'' \in \Sigma^*$, $l_1'' \in L_1'$ mit $r_1'' \sqsubset_p r_1' \wedge r_1''l_1'' \sqsubseteq_p r_1'l_1'r_2'l_2'v_2'$ [Die Startbedingung ist erfüllt]. Da $(r_1', l_1', r_2'l_2'v_2')$ alle Bedingungen der Definition von $\mathit{SkipTo}(L_1')$ erfüllt, ist $(r_1', l_1', r_2'l_2'v_2') \in \mathit{SkipTo}(L_1')$. Wir beweisen nun, dass $(r_2', l_2', v_2') \in \mathit{SkipTo}(L_2')$ gilt. Es gilt $r_2' \in \Sigma^*$ und $v_2' \in \Sigma^*$ [Die Randbedingung ist erfüllt] sowie $l_2' \in L_2'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_2' und l_2' existieren keine $r_2'' \in \Sigma^*$, $l_2'' \in L_2'$ mit $r_2'' \sqsubset_p r_2' \wedge r_2''l_2'' \sqsubseteq_p r_2'l_2'v_2'$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r_2', l_2', v_2') \in \mathit{SkipUntil}(L_2')$ und $r_2'\#l_2'v_2' \in SU'$. Damit ist $w'' = r_1'l_1'r_2'\#l_2'v_2' \in STSU'$. Angenommen, $w'' \in STSU$ gilt. Dann existiert eine Zerlegung $w' = r_1''l_1''r_2''\#l_2''v_2''$ mit $(r_1'', l_1'', r_2''l_2''v_2'') \in \mathit{SkipTo}(L_1)$, $r_2''\#l_2''v_2'' \in SU$ und $(r_2'', l_2'', v_2'') \in \mathit{SkipUntil}(L_2)$. Aus $l_1r_2 \sqsubset_p r_1'l_1'r_2'$ folgt $l_1r_2 \sqsubset_p r_1''l_1''r_2''$. Somit muss $r_1'' = \varepsilon$ und $l_1'' = l_1$ gelten. Aus $l_1r_2 \sqsubset_p r_1''l_1''r_2'' = l_1r_2''$ folgt dann $r_2 \sqsubset_p r_2''$. Da $r_1''l_1''r_2''l_2''v_2'' = l_1r_2l_2v_2$ ist, gilt außerdem $r_2l_2 \sqsubseteq_p r_2''l_2''v_2''$. Damit kann (r_2'', l_2'', v_2'') die Startbedingung der Definition von $\mathit{SkipUntil}(L_2)$ nicht erfüllen, was eine Widerspruch zur Annahme ist, dass $(r_2'', l_2'', v_2'') \in \mathit{SkipUntil}(L_2)$ gilt. Also ist $w'' \notin STSU$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $STSU' \subset STSU$ gilt. Somit gilt Fall (1) nicht.

Fall (2): Es existieren keine r_2' und $l_2' \in L_2'$ mit $r_1'l_1'r_2'l_2' \sqsubseteq_p l_1r_2l_2v_2$. Dann gibt es zwei Möglichkeiten:

Fall (a): Es existieren $r_1'' \in \Sigma^*$, $v' \in \Sigma^+$ und $l_1'' \in L_1'$ mit $r_1'' \sqsubset_p r_1' \wedge r_1''l_1'' = r_1'l_1'v' \wedge r_1r_2l_2v_2 \sqsubset_p r_1''l_1''$.

> Angenommen, dies gilt. Aus der Menge der $r_1''l_1''$, die diese Bedingung erfüllen, betrachten wir das Paar mit dem kürzesten r_1'' und einem l_1'', so dass kein $l_x \in L_1'$ existiert mit $l_x \sqsubset_p l_1''$. Sei $w' = r_1''l_1''\#l_2v_2$. Wir zeigen zuerst, dass $(r_1'', l_1'', l_2v_2) \in \mathit{SkipTo}(L_1')$ gilt. Es sind $r_1'' \in \Sigma^*$ und $l_2v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l_1'' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_1'' und l_1'', existieren kein $l_x \in L_1'$ mit $l_x \sqsubset_p l_1''$ [Die Minimalitätsbedingung ist erfüllt] und keine $r_x \in \Sigma^*$, $l_x \in L_1'$ mit $r_x \sqsubset_p r_1'' \wedge r_xl_x \sqsubseteq_p r_1''l_1''l_2v_2$ [Die Startbedingung ist erfüllt]. Da (r_1'', l_1'', l_2v_2) alle Bedingungen der Definition von $\mathit{SkipTo}(L_1')$ erfüllt, ist $(r_1'', l_1'', l_2v_2) \in \mathit{SkipTo}(L_1')$. Da außerdem $\#l_2v_2 \in SU'$ gilt, ist $w' \in STSU'$. Angenommen, $w' \in STSU$ gilt. Dann existiert eine Zerlegung $w' = r_xl_xr_x'\#l_2v_2$ mit

$(r_x, l_x, r'_x l_2 v_2) \in \mathit{SkipTo}(L_1)$. Aus $l_1 r_2 l_2 v_2 \sqsubset_p r_x l_x$ folgt $l_1 r_2 v_2 \sqsubset_p r_x l_x r'_x$. Somit muss $r_x = \varepsilon$ und $l_x = l_1$ gelten. Aus $l_1 r_2 l_2 v_2 \sqsubset_p r_x l_x r'_x = l_1 r'_x$ folgt dann $r_2 \sqsubset_p r'_x$. Da $r_2 l_2 v_2 \sqsubset_p r'_x$ ist, gilt außerdem $r_2 l_2 \sqsubseteq_p r'_x l_2 v_2$. Damit kann (r'_x, l_2, v_2) die Startbedingung der Definition von $\mathit{SkipUntil}(L_2)$ nicht erfüllen, was eine Widerspruch zur Annahme ist, dass $(r'_x, l_2, v_2) \in \mathit{SkipUntil}(L_2)$ gilt. Also ist $w'' \notin STSU$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $STSU' \subset STSU$ gilt. Somit gilt Fall (1) nicht.

Fall (b): Es existieren keine $r''_1 \in \Sigma^*$, $v' \in \Sigma^+$ und $l''_1 \in L'_1$ mit $r''_1 \sqsubset_p r'_1 \wedge r''_1 l''_1 = r'_1 l'_1 v' \wedge r_1 r_2 l_2 v_2 \sqsubset_p r''_1 l''_1$.

Angenommen, dies gilt. Sei $w' = r'_1 l'_1 \# l_2 v_2$. Wir zeigen zuerst, dass $(r'_1, l'_1, l_2 v_2) \in \mathit{SkipTo}(L'_1)$ gilt. Es sind $r'_1 \in \Sigma^*$ und $l_2 v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l'_1 \in L'_1$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r'_1 und l'_1, existieren kein $l' \in L'_1$ mit $l' \sqsubset_p l'_1$ [Die Minimalitätsbedingung ist erfüllt] und keine $r' \in \Sigma^*$, $l' \in L'_1$ mit $r' \sqsubset_p r'_1 \wedge r' l' \sqsubseteq_p r'_1 l'_1 l_2 v_2$ [Die Startbedingung ist erfüllt]. Da $(r'_1, l'_1, l_2 v_2)$ alle Bedingungen der Definition von $\mathit{SkipTo}(L'_1)$ erfüllt, ist $(r'_1, l'_1, l_2 v_2) \in \mathit{SkipTo}(L'_1)$. Da außerdem $\# l_2 v_2 \in NT'$ gilt, ist $w' \in STNT'$. Angenommen, $w' \in STNT$ gilt. Dann existiert eine Zerlegung $w' = r' l' \# l_2 v_2$ mit $(r', l', l_2 v_2) \in \mathit{SkipTo}(L_1)$. Aus $l_1 r_2 \sqsubset_p r'_1 l'_1 = r' l'$ folgt $l_1 \sqsubset_p r' l'$. Ist $r' = \varepsilon$, dann gilt $l_1 \sqsubset_p l'$ und $(r', l', l_2 v_2)$ erfüllt nicht die Minimalitätsbedingung von $\mathit{SkipTo}(L_1)$. Ist $r' \neq \varepsilon$, dann gilt $r_\varepsilon \sqsubset_p r' \wedge r_\varepsilon l_1 \sqsubseteq_p r' l' l_2 v_2$ mit $r_\varepsilon = \varepsilon$ und $(r', l', l_2 v_2)$ erfüllt nicht die Startbedingung von $\mathit{SkipTo}(L_1)$. Somit gilt $(r', l', l_2 v_2) \notin \mathit{SkipTo}(L_1)$, was ein Widerspruch zu unserer Annahme ist, dass $(r', l', l_2 v_2) \in \mathit{SkipTo}(L_1)$ ist. Also ist $w' \notin STNT$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $STNT' \subset STNT$ gilt und deshalb kann Fall (b) nicht auftreten.

Da wir Fall (a) und Fall (b) widerlegt haben, kann auch nicht Fall (2) auftreten.

Wir haben gezeigt, dass weder Fall (1) noch Fall (2) auftreten kann, und damit auch keine $r'_1 \in \Sigma^*$ und $l'_1 \in L'_1$ existieren mit $r'_1 \sqsubset_p r' \wedge r'_1 l'_1 \sqsubseteq_p l_1 r_2 l_2 v_2 \wedge l_1 r_2 \sqsubset_p r'_1 l'_1$.

■

Lemma 7 zeigt, dass $(r_1, l_1, r_2 l_2 v_2)$ die Sprachbedingung der Definition von $\mathit{SkipTo}(L'_1)$ erfüllt, wenn $T \subseteq STSU' \subset STSU$, $w \in STSU \backslash STSU'$, $(r_1, l_1, r_2 l_2 v_2) \in \mathit{SkipTo}(L_1)$, $r_2 \# l_2 v_2 \in SU$ und $r_2 \# l_2 v_2 \in SU'$ mit $w = r_1 l_1 r_2 l_2 \# v_2$ gilt. In Lemma 8 und 8 werden wir beweisen, dass $(r_1, l_1, r_2 l_2 v_2)$ die Minimalitätsbedingung und die Startbedingung der Definition von $\mathit{SkipTo}(L'_1)$ unabhängig von der Sprachbedingung erfüllt. Deshalb können wir für den Beweis von Lemma 15 annehmen, dass $(r_1, l_1, r_2 l_2 v_2)$ die Minimalitätsbedingung und die Startbedingung der Definition von $\mathit{SkipTo}(L'_1)$ erfüllt.

Lemma 7 *Seien $T \subseteq STSU' \subset STSU$ und $w \in STSU \backslash STSU'$ mit $w = r_1 l_1 r_2 \# l_2 v_2$. Es gelte $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1)$, $r_2 \# l_2 v_2 \in SU$ und $(r_2, l_2, v_2) \in SkipUntil(L_2)$. Weiterhin nehmen wir o.B.d.A. an, dass $(r_1, l_1, r_2 l_2 v_2)$ die Minimalitätsbedingung und die Startbedingung der Definition von $SkipTo(L'_1)$ erfüllt und dass $r_2 \# l_2 v_2 \in SU'$ gilt. Dann erfüllt $(r_1, l_1, r_2 l_2 v_2)$ auch die Sprachbedingung der Definition von $SkipTo(L'_1)$, d.h. es gilt $l_1 \in L'_1$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. $l_1 \notin L'_1$. Wir betrachten das Wort $w' = l_1 \# l_2$. Es gilt $w' \in T$. Aus $T \subseteq STSU'$ folgt $w' \in STSU'$. Da $l_1 \notin L'_1$ gilt, muss eine andere Zerlegung existieren, die die Bedingungen der Sprache erfüllt. Angenommen, $w' = r' l' r'' \# l'' v''$ sei diese Zerlegung mit $(r', l', r'' l'' v'') \in SkipTo(L'_1)$ und $r'' \# l'' v'' \in SU'$. Dann ist $r' l' r'' = l_1$. Es gilt einer der folgenden Fälle (zur Veranschaulichung siehe Abbildung 5):

Fall (1) $r'' = \varepsilon$

Fall (2) $r'' \neq \varepsilon$

Fall (1): Angenommen, $r'' = \varepsilon$ gilt. Wir betrachten die Zerlegung $w = r_1 r' l' r_2 \# l_2 v_2$. Es gilt $r_2 \# l_2 v_2 \in SU'$. Würde $(r_1 r', l', r_2 l_2 v_2) \in SkipTo(L'_1)$ gelten, dann wäre $w \in STSU'$. Dies wäre ein Widerspruch zu unserer Annahme, dass $w \in STSU \backslash STSU'$ gilt. Deshalb nehmen wir das Gegenteil an: $(r_1 r', l', r_2 l_2 v_2) \notin SkipTo(L'_1)$. Da $(r_1, l_1, r_2 l_2 v_2)$ die Startbedingung der Definition von $SkipTo(L'_1)$ erfüllt und weil wegen $(r', l', r'' l'' v'') \in SkipTo(L'_1)$ sowohl $l' \in L'_1$ gilt als auch kein $l'' \in L'_1$ existiert mit $l'' \sqsubset_p l'$, kann es nur einen Grund geben, warum $(r_1 r', l', r_2 l_2 v_2) \notin SkipTo(L'_1)$ ist: $\exists r'_1 \in \Sigma^*, l'_1 \in L'_1$ mit $r'_1 \sqsubset_p r' \wedge r'_1 l'_1 \sqsubseteq_p r' l' r_2 l_2 v_2$.

Angenommen, dies gilt. Dann gibt es folgende Möglichkeiten:

Fall (a): $r'_1 l'_1 \sqsubseteq_p r' l' r_2$

> Angenommen, $r'_1 l'_1 \sqsubseteq_p r' l' r_2 = l_1 r_2$ gilt. Aus der Menge der $r'_1 l'_1$ für die dies gilt, betrachten wir das Paar mit dem kürzesten r'_1 und einem l'_1, so dass kein $l''_1 \in L'_1$ existiert mit $l''_1 \sqsubset_p l'_1$. Es existiert ein $r'_2 \in \Sigma^*$ mit $r'_1 l'_1 r'_2 = l_1 r_2$. Wir beweisen nun, dass $(r_1 r'_1, l'_1, r'_2 l_2 v_2) \in SkipTo(L'_1)$ gilt. Es gilt $r_1 r'_1 \in \Sigma^*$ und $r'_2 l_2 v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l'_1 \in L'_1$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von l'_1 existiert kein $l'' \in L'_1$ mit $l'' \sqsubset_p l'_1$ [Die Minimalitätsbedingung ist erfüllt]. Da $(r_1, l_1, r_2 l_2 v_2)$ die Startbedingung der Definition von $SkipTo(L'_1)$ erfüllt und weil aufgrund unserer Wahl von r'_1 und l'_1, keine $r'' \in \Sigma^*$ und $l'' \in L'_1$ existieren mit $r'' \sqsubset_p r'_1 \wedge r'' l'' \sqsubseteq_p r'_1 l'_1 r'_2 l_2 v_2$, gilt Folgendes: $\neg \exists r'' \in \Sigma^*, l'' \in L'_1$ mit $r'' \sqsubset_p r_1 r'_1 \wedge r'' l'' \sqsubseteq_p r_1 r'_1 l'_1 r'_2 l_2 v_2$ [Die Startbedingung ist erfüllt]. Da $(r_1 r'_1, l'_1, r'_2 l_2 v_2)$ alle Bedingungen der Definition von $SkipTo(L'_1)$ erfüllt, ist $(r_1 r'_1, l'_1, r'_2 l_2 v_2) \in SkipTo(L'_1)$. Weil außerdem $r_2 \# l_2 v_2 \in SU'$ und $r'_2 \sqsubset_s r_2$ gilt, ist $r'_2 \# l_2 v_2 \in SU'$. Damit gilt $w \in STSU'$, was ein Widerspruch zu unserer Annahme ist, dass $w \in STSU \backslash STSU'$ gilt. Deshalb kann der Fall (a) nicht auftreten.

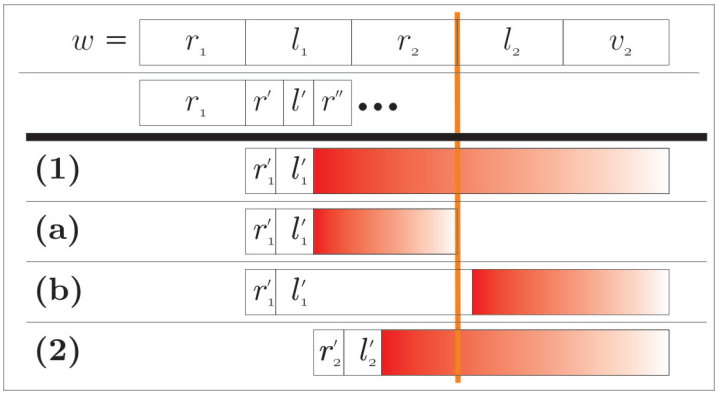


Abbildung 7: Lemma 7

Fall (b): $r'l'r_2 \sqsubset_p r'_1l'_1$

> Angenommen, $l_1r_2 = r'l'r_2 \sqsubset_p r'_1l'_1$ gilt. Dies ist aber ein Widerspruch zu Lemma 6. Deshalb kann Fall (b) nicht auftreten.

Da weder Fall (a) noch Fall (b) gelten kann, kann Fall (1) nicht auftreten.

Fall (2): Angenommen, $r'' \neq \varepsilon$ gilt. Wir betrachten uns die Zerlegung $w' = r_1r'l'r''r_2\#l_2v_2$. Es gilt $(r_1r', l', r''r_2l_2v_2) \in \mathit{SkipTo}(L'_1)$ (Dies kann man analog zur Widerlegung von Fall (1) zeigen). Würde $r''r_2\#l_2v_2 \in SU'$ gelten, dann wäre $w \in STSU'$, was ein Widerspruch zu unserer Annahme wäre, dass $w \in STSU \backslash STSU'$ gilt. Deshalb nehmen wir das Gegenteil an. Wir wissen, dass $r_2\#l_2v_2 \in SU'$ gilt, weshalb es nur einen Grund für $r''r_2\#l_2v_2 \notin SU'$ geben kann: $\exists r'_2 \in \Sigma^*, l'_2 \in L'_1$ mit $r'_2 \sqsubset_p r'' \wedge r'_2l'_2 \sqsubseteq_p r''r_2l_2v_2$.

Angenommen, dies gilt. Aus der Menge der $r'_2l'_2$ für die dies gilt, betrachten wir das Paar mit dem kürzesten r'_2. Es existiert ein $v'_2 \in \Sigma^*$ mit $r'_2l'_2v'_2 = r''r_2l_2v_2$. Wir beweisen nun, dass $(r'_2, l'_2, v'_2) \in \mathit{SkipUntil}(L'_2)$ gilt. Es gilt $r'_2 \in \Sigma^*$ und $v'_2 \in \Sigma^*$ [Die Randbedingungen sind erfüllt] sowie $l'_2 \in L'_2$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r'_2 und l'_2 existieren keine $r''_2 \in \Sigma^*$, $l''_2 \in L'_2$ mit $r''_2 \sqsubset_p r'_2 \wedge r''_2l''_2 \sqsubseteq_p r'_2l'_2v'_2$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, ist $(r'_2, l'_2, v'_2) \in \mathit{SkipUntil}(L'_2)$ und damit $r'_2l'_2\#v'_2 \in SU'$. Da außerdem $(r_1r', l', r''r_2l_2v_2) = (r_1r', l', r'_2l'_2v'_2) \in \mathit{SkipTo}(L'_1)$ gilt, ist $w' = r_1r'l'r'_2\#l'_2v'_2 \in STSU'$. Wegen $r_1r'l'r'_2 \sqsubset_p r_1l_1$ und $w \in STSU$ gilt $w' \notin STSU$. Dies ist eine Widerspruch zu unserer Annahme, dass $STSU' \subset STSU$ gilt. Deshalb ist $r''r_2\#l_2v_2 \in SU'$ und damit $w \in STSU'$, was wiederum ein Widerspruch zu unserer Annahme ist, dass $w \in STSU \backslash STSU'$ gilt. Deshalb kann Fall (2) nicht gelten.

Wir haben Fall (1) und Fall (2) widerlegt und somit erfüllt $(r_1, l_1, r_2 l_2 v_2)$ die Sprachbedingung der Definition von $SkipTo(L_1')$.

■

Das nachfolgende Lemma beweist, dass Fall (2) aus Lemma 5 nicht gelten kann. In Lemma 9 werden wir beweisen, dass $(r_1, l_1, r_2 l_2 v_2)$ die Startbedingung der Definition von $SkipTo(L_1')$ unabhängig von der Minimalitätsbedingung erfüllt. Deshalb können wir für den Beweis von Lemma 8 annehmen, dass $(r_1, l_1, r_2 l_2 v_2)$ die Startbedingung der Definition von $SkipTo(L_1')$ erfüllt.

Lemma 8 *Seien $T \subseteq STSU' \subset STSU$ und $w \in STSU \backslash STSU'$ mit $w = r_1 l_1 r_2 \# l_2 v_2$. Es gelte $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1)$ und $r_2 \# l_2 v_2 \in SU$. Weiterhin nehmen wir o.B.d.A. an, dass $(r_1, l_1, r_2 l_2 v_2)$ die Startbedingung der Definition von $SkipTo(L_1')$ erfüllt und dass $r_2 \# l_2 v_2 \in SU'$ gilt. Dann erfüllt $(r_1, l_1, r_2 l_2 v_2)$ die Minimalitätsbedingung der Definition von $SkipTo(L_1')$, d.h. $\neg \exists l_1' \in L_1'$ mit $l_1' \sqsubset_p l_1$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. sei $l_1' \in L_1'$ mit $l_1' \sqsubset_p l_1$. Wir betrachten die Zerlegung $w = r_1 l_1' r'' r_2 \# l_2 v_2$ mit dem kleinsten l_1', das diese Bedingung erfüllt, und $l_1' r'' = l_1$. Wir zeigen nun, dass $(r_1, l_1', r'' r_2 l_2 v_2) \in SkipTo(L_1')$ gilt. Es sind $r_1 \in \Sigma^*$ und $r'' r_2 l_2 v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l_1' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von l_1', existiert kein $l_1'' \in L_1'$ mit $l_1'' \sqsubset_p l_1'$ [Die Minimalitätsbedingung ist erfüllt]. Da $(r_1, l_1, r_2 l_2 v_2)$ die Startbedingung der Definition von $SkipTo(L_1')$ erfüllt, existieren auch keine $r_1'' \in \Sigma^*$, $l_1'' \in L_1'$ mit $r_1'' \sqsubset_p r_1 \wedge r_1'' l_1'' \sqsubseteq_p r_1 l_1' r'' r_2 l_2 v_2$ [Die Startbedingung ist erfüllt]. Da $(r_1, l_1', r'' r_2 l_2 v_2)$ alle Bedingungen der Definition von $SkipTo(L_1')$ erfüllt, ist $(r_1, l_1', r'' r_2 l_2 v_2) \in SkipTo(L_1')$. Würde $r'' r_2 \# l_2 v_2 \in SU'$ gelten, dann wäre $w \in STSU'$. Dies wäre ein Widerspruch zu unserer Annahme, dass $w \in STSU \backslash STSU'$ gilt. Deshalb nehmen wir das Gegenteil an: $r'' r_2 \# l_2 v_2 \notin SU'$. Wegen $r_2 \# l_2 v_2 \in SU'$ kann es nur einen Grund geben, warum $r'' r_2 \# l_2 v_2 \notin SU'$ gilt: $\exists r_2' \in \Sigma^*$, $l_2' \in L_2'$ mit $r_2' \sqsubset_p r''$ und $r_2' l_2' \sqsubset_p r'' r_2 l_2 v_2$. Aus der Men-

ge der $r_2' l_2'$ für die diese Bedingung gilt, betrachten das Paar mit dem kürzesten r_2'. Es existiert ein $v_2' \in \Sigma^*$, so dass $r_2' l_2' v_2' = r'' r_2 l_2 v_2$ gilt. Wir beweisen nun, dass $(r_2', l_2', v_2') \in SkipUntil(L_2')$ gilt. Es sind $r_2' \in \Sigma^*$ und $v_2' \in \Sigma^*$[Die Randbedingungen sind erfüllt] sowie $l_2' \in L_2'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_2' und l_2', existieren keine $r_2'' \in \Sigma^*$, $l_2'' \in L_2'$ mit $r_2'' \sqsubset_p r_2' \wedge r_2'' l_2'' \sqsubseteq_p r_2' l_2' v_2'$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, ist $(r_2', l_2', v_2') \in SkipUntil(L_2')$ und damit $r_2' l_2' \# v_2' \in SU'$. Da außerdem $(r_1, l_1', r'' r_2 l_2 v_2) = (r_1, l_1', r_2' l_2' v_2') \in SkipTo(L_1')$ gilt, ist $w' = r_1 l_1' r_2' \# l_2' v_2' \in STSU'$. Wegen $r_1 l_1' r_2' \sqsubset_p r_1 l_1$ und $w \in STSU$ gilt jedoch $w' \notin STSU$. Dies ist ein Widerspruch zu unserer Annahme, dass $STSU' \subset STSU$ gilt. Deshalb erfüllt $(r_1, l_1, r_2 l_2 v_2)$ die Minimalitätsbedingung der Definition von $SkipTo(L_1')$. ■

Lemma 9 zeigt, dass Fall (3) aus Lemma 5 nicht gelten kann.

Lemma 9 *Seien $T \subseteq STSU' \subset STSU$ und $w \in STSU \backslash STSU'$ mit $w = r_1 l_1 r_2 \# l_2 v_2$. Weiterhin gelte $(r_1, l_1, r_2 l_2 v_2) \in SkipTo(L_1)$, $r_2 \# l_2 v_2 \in SU$ und $r_2 \# l_2 v_2 \in SU'$. Dann erfüllt $(r_1, l_1, r_2 l_2 v_2)$ die Startbedingung der Definition von $SkipTo(L_1')$, d.h. $\neg \exists r_1' \in \Sigma^*, l_1' \in L_1'$ mit $r_1' \sqsubset_p r_1 \wedge r_1' l_1' \sqsubseteq_p r_1 l_1 r_2 l_2 v_2$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. seien $r_1' \in \Sigma^*$ und $l_1' \in L_1'$ mit $r_1' \sqsubset_p r_1 \wedge r_1' l_1' \sqsubseteq_p r_1 l_1 r_2 l_2 v_2$. Aus der Menge $r_1' l_1'$ für die diese Bedingung gilt, betrachten das Paar mit dem kürzesten r_1' und einem l_1', so dass kein $l_1'' \in L_1'$ existiert, mit $l_1'' \sqsubset_p l_1'$. Es gibt nun folgende Möglichkeiten (zur Veranschaulichung siehe Abbildung 6):

Fall (1) $r_1' l_1' \sqsubset_p r_1 l_1$

Fall (2) $r_1' l_1' \sqsubseteq_p r_1 l_1 r_2 \wedge r_1 l_1 \sqsubset_p r_1' l_1'$

Fall (3) $r_1 l_1 r_2 \sqsubset_p r_1' l_1'$

Fall (1): Angenommen, $r_1' l_1' \sqsubset_p r_1 l_1$ gilt. Dann existiert ein $r_2' \in \Sigma^+$ mit $r_1' l_1' r_2' = r_1 l_1$. Zuerst zeigen wir, dass $(r_1', l_1', r_2' r_2 l_2 v_2) \in SkipTo(L_1')$ gilt. Es sind $r_1' \in \Sigma^*$ und $r_2' r_2 l_2 v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l_1' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_1' und l_1', existieren kein $l_1'' \in L_1'$ mit $l_1'' \sqsubset_p l_1'$ [Die Minimalitätsbedingung ist erfüllt] und keine $r_1'' \in \Sigma^*$, $l_1'' \in L_1'$ mit $r_1'' \sqsubset_p r_1' \wedge r_1'' l_1'' \sqsubseteq_p r_1' l_1' r_2' r_2 l_2 v_2$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, ist $(r_1', l_1', r_2' r_2 l_2 v_2) \in SkipTo(L_1')$.

Angenommen, $r_2' r_2 \# l_2 v_2 \in SU'$ gelte, dann wäre $w \in STSU'$, was ein Widerspruch zu unserer Annahme wäre, dass $w \in STSU \backslash STSU'$ gilt. Deshalb nehmen wir das Gegenteil an. Wir wissen, dass $r_2 \# l_2 v_2 \in SU'$ gilt, weshalb es nur einen Grund für $r_2' r_2 \# l_2 v_2 \notin SU'$ geben kann: $\exists r_2'' \in \Sigma^*, l_2' \in L_2'$ mit $r_2'' \sqsubset_p r_2' \wedge r_2'' l_2' \sqsubseteq_p r_2' r_2 l_2 v_2$. Angenommen, dies gilt. Aus der Menge der $r_2'' l_2'$ für die diese Bedingung gilt, betrachten das Paar mit dem kürzesten r_2''. Es existiert ein $v_2' \in \Sigma^*$, so dass $r_2'' l_2' v_2' = r_2' r_2 l_2 v_2$ gilt. Wir beweisen jetzt, dass $(r_2'', l_2', v_2') \in SkipUntil(L_2')$ gilt. Es ist $r_2'' \in \Sigma^*$ und $v_2' \in \Sigma^*$ [Die Randbedingungen ist erfüllt] sowie $l_2' \in L_2'$ [Die Sprachbedingung ist erfüllt]. Aufgrund unserer Wahl von r_2'' und l_2', existieren keine $r'' \in \Sigma^*$, $l'' \in L_2'$ mit $r'' \sqsubset_p r_2'' \wedge r'' l'' \sqsubseteq_p r_2'' l_2' v_2'$ [Die Startbedingung ist erfüllt]. Weil (r_2'', l_2', v_2') alle Bedingungen von $SkipUntil(L_2')$ erfüllt, ist $(r_2'', l_2', v_2') \in SkipUntil(L_2')$ und $r_2'' \# l_2' v_2' \in SU'$. Da außerdem $(r_1', l_1', r_2'' l_2' v_2') \in SkipTo(L_1')$ gilt, ist $w' = r_1' l_1' r_2'' \# l_2' v_2' \in STSU'$. Wegen $r_1' l_1' r_2'' \sqsubset_p r_1 l_1$ und $w \in STSU$ gilt jedoch $w' \notin STSU$. Dies ist eine Widerspruch zu unserer Annahme, dass $STSU' \subset STSU$ gilt. Deshalb ist $r_2' r_2 \# l_2 v_2 \in SU'$ und damit $w \in STSU'$. Da das aber ein Widerspruch zu unserer Annahme ist, dass $w \in STSU \backslash STSU'$ ist, gilt Fall (1) nicht.

Fall (2): Angenommen, $r_1' l_1' \sqsubseteq_p r_1 l_1 r_2 \wedge r_1 l_1 \sqsubset_p r_1' l_1'$ gilt. Aus der Menge der $r_1' l_1'$, für die

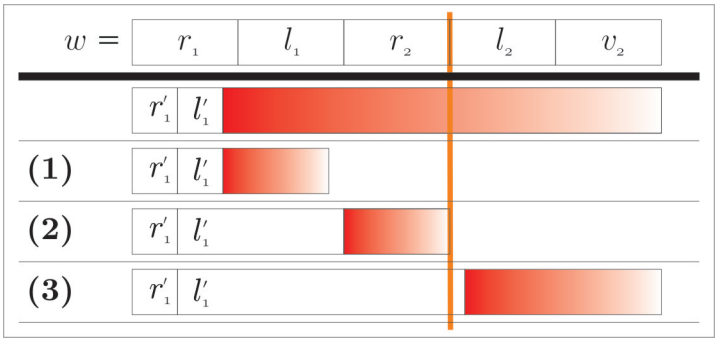


Abbildung 8: Lemma 9

diese Annahme gilt, betrachten wir das Paar mit dem kürzesten r'_1 und einem l'_1, so dass kein $l''_1 \in L'_1$ existiert, mit $l''_1 \sqsubset_p l'_1$. Es gibt ein $r'_2 \in \Sigma^+$ mit $r'_1 l'_1 r'_2 = r_1 l_1 r_2$. Wir zeigen zuerst, dass $(r'_1, l'_1, r'_2 l_2 v_2) \in \mathit{SkipTo}(L'_1)$ gilt. Es sind $r'_1 \in \Sigma^*$ und $r'_2 l_2 v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l'_1 \in L'_1$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r'_1 und l'_1, existieren kein $l''_1 \in L'_1$ mit $l''_1 \sqsubset_p l'_1$ [Die Minimalitätsbedingung ist erfüllt] und keine $r''_1 \in \Sigma^*$, $l''_1 \in L'_1$ mit $r''_1 \sqsubset_p r'_1 \wedge r''_1 l''_1 \sqsubseteq_p r'_1 l'_1 r'_2 r_2 l_2 v_2$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, ist $(r'_1, l'_1, r'_2 l_2 v_2) \in \mathit{SkipTo}(L'_1)$. Wegen $r_2 \# l_2 v_2 \in SU'$ und $r'_2 \sqsubseteq_s r_2$ gilt außerdem $r'_2 \# l_2 v_2 \in SU'$. Damit ist aber $w \in STSU'$, was ein Widerspruch zu unserer Annahme ist, dass $w \in STSU \backslash STSU'$ gilt. Somit gilt Fall (2) nicht.

Fall (3):Angenommen, $r_1 l_1 r_2 \sqsubset_p r'_1 l'_1$ gilt. Wir unterscheiden zwei Fälle:

Fall (a): Es existieren r'_2 und $l'_2 \in L'_2$ mit $r'_1 l'_1 r'_2 l'_2 \sqsubseteq_p r_1 l_1 r_2 l_2 v_2$.

> Angenommen, Fall (a) gilt. Aus der Menge $r'_2 l'_2$ für die diese Bedingung gilt, betrachten wir das Paar mit dem kürzesten r'_2. Dann existiert ein $v'_2 \in \Sigma^*$ mit $r'_1 l'_1 r'_2 l'_2 v'_2 = r_1 l_1 r_2 l_2 v_2$. Wir zeigen zuerst, dass $(r'_1, l'_1, r'_2 l'_2 v'_2) \in \mathit{SkipTo}(L'_1)$ gilt. Es sind $r'_1 \in \Sigma^*$ und $r'_2 l'_2 v'_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l'_1 \in L'_1$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r'_1 und l'_1, existieren kein $l''_1 \in L'_1$ mit $l''_1 \sqsubset_p l'_1$ [Die Minimalitätsbedingung ist erfüllt] und keine $r''_1 \in \Sigma^*$, $l''_1 \in L'_1$ mit $r''_1 \sqsubset_p r'_1 \wedge r''_1 l''_1 \sqsubseteq_p r'_1 l'_1 r'_2 l'_2 v'_2$ [Die Startbedingung ist erfüllt]. Da $(r'_1, l'_1, r'_2 l'_2 v'_2)$ alle Bedingungen der Definition von $\mathit{SkipTo}(L'_1)$ erfüllt, ist $(r'_1, l'_1, r'_2 l'_2 v'_2) \in \mathit{SkipTo}(L'_1)$. Wir beweisen nun, dass $(r'_2, l'_2, v'_2) \in \mathit{SkipUntil}(L'_2)$ gilt. Es gilt $r'_2 \in \Sigma^*$ und $v'_2 \in \Sigma^*$ [Die Randbedingungen sind erfüllt] sowie $l'_2 \in L'_2$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r'_2 existieren keine $r''_2 \in \Sigma^*$, $l''_2 \in L'_2$ mit $r''_2 \sqsubset_p r'_2 \wedge r''_2 l''_2 \sqsubseteq_p r'_2 l'_2 v'_2$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, ist $(r'_2, l'_2, v'_2) \in \mathit{SkipUntil}(L'_2)$ und $r'_2 \# l'_2 v'_2 \in SU'$. Damit gilt $w' = r'_1 l'_1 r'_2 \# l'_2 v'_2 \in STSU'$. Angenommen, $w' \in STSU$ gilt. Dann existiert eine

Zerlegung $w' = r_1''l_1''r_2''\#l_2''v_2''$ mit $(r_1'', l_1'', r_2''l_2''v_2'') \in \mathit{SkipTo}(L_1)$, $r_2''\#l_2''v_2'' \in SU$ und $(r_2'', l_2'', v_2'') \in \mathit{SkipUntil}(L_2)$. Aus $r_1l_1r_2 \sqsubset_p r_1'l_1'r_2'$ folgt $r_1l_1r_2 \sqsubset_p r_1''l_1''r_2''$. Ist $r_1 \sqsubset_p r_1''$, dann kann $(r_1'', l_1'', r_2''l_2''v_2'')$ die Sprachbedingung von $\mathit{SkipTo}(L_1)$ nicht erfüllen. Das ist ein Widerspruch zu der Annahme, dass $(r_1'', l_1'', r_2''l_2''v_2'') \in \mathit{SkipTo}(L_1)$ gilt. Ist $r_1'' \sqsubset_p r_1$, dann kann $(r_1, l_1, r_2l_2v_2)$ die Sprachbedingung von $\mathit{SkipTo}(L_1)$ nicht erfüllen, da zusätzlich $r_1''l_1'' \sqsubseteq_p r_1l_1r_2l_2v_2$ gilt. Das ist ein Widerspruch zu der Annahme, dass $(r_1, l_1, r_2l_2v_2) \in \mathit{SkipTo}(L_1)$ gilt. Ist $r_1'' = r_1$, dann kann $(r_1'', l_1'', r_2''l_2''v_2'')$ die Minimalitätsbedingung von $\mathit{SkipTo}(L_1)$ nicht erfüllen, da $l_1 \sqsubset_p l_1''$ gilt. Das ist ein Widerspruch zu der Annahme, dass $(r_1'', l_1'', r_2''l_2''v_2'') \in \mathit{SkipTo}(L_1)$ gilt. Also ist $w' \notin STSU$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $STSU' \subset STSU$ gilt. Somit gilt Fall (a) nicht.

Fall (b): Es existieren keine r_2' und $l_2' \in L_2'$ mit $r_1'l_1'r_2'l_2' \sqsubseteq_p r_1l_1r_2l_2v_2$. Es existieren aber $r_2' \in \Sigma^*$, $r_3' \in \Sigma^*$ und $l_2' \in L_2'$ mit $r_1'l_1'r_2'r_3' = r_1l_1r_2l_2v_2$ und $r_3' \sqsubset_p l_2'$.

Angenommen, Fall (b) gilt. Aus der Menge der $r_2'l_2'$ für die diese Bedingung gilt, betrachten wir das Paar mit dem kürzesten r_2'. Wir beweisen nun, dass $w'' = r_\varepsilon l_1'r_2'\#l_2'v_2' \in STSU'$ gilt mit $r_\varepsilon = \varepsilon$ und $v_2' = \varepsilon$. Dafür zeigen wir zuerst, dass $(r_\varepsilon, l_1', r_2'l_2'v_2') \in \mathit{SkipTo}(L_1')$ gilt. Es sind $r_\varepsilon \in \Sigma^*$ und $r_2'l_2'v_2' \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l_1' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von l_1', existieren kein $l_1'' \in L_1'$ mit $l_1'' \sqsubset_p l_1'$ [Die Minimalitätsbedingung ist erfüllt]. Aufgrund von $r_\varepsilon = \varepsilon$ existieren keine $r_1'' \in \Sigma^*$, $l_1'' \in L_1'$ mit $r_1'' \sqsubset_p r_\varepsilon \wedge r_1''l_1'' \sqsubseteq_p r_\varepsilon l_1'r_2'l_2'v_2'$ [Die Startbedingung ist erfüllt]. Da $(r_\varepsilon, l_1', r_2'l_2'v_2')$ alle Bedingungen der Definition von $\mathit{SkipTo}(L_1')$ erfüllt, ist $(r_\varepsilon, l_1', l_2'v_2') \in \mathit{SkipTo}(L_1')$. Wir beweisen nun, dass $(r_2', l_2', v_2') \in \mathit{SkipUntil}(L_2')$ gilt. Es gilt $r_2' \in \Sigma^*$ und $v_2' \in \Sigma^*$ [Die Randbedingung ist erfüllt] sowie $l_2' \in L_2'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_2' und l_2' existieren keine $r_2'' \in \Sigma^*$, $l_2'' \in L_2'$ mit $r_2'' \sqsubset_p r_2' \wedge r_2''l_2'' \sqsubseteq_p r_2'l_2'v_2'$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r_2', l_2', v_2') \in \mathit{SkipUntil}(L_2)$ und $r_2'\#l_2'v_2' \in SU'$. Damit ist $w'' = r_\varepsilon l_1'r_2'\#l_2'v_2' = l_1'r_2'\#l_2' \in STSU'$. Angenommen, es gilt $w'' \in STSU$. Dann existiert eine Zerlegung $w'' = r_1''l_1''r_2''\#l_2''v_2''$ mit $(r_1'', l_1'', r_2''l_2''v_2'') \in \mathit{SkipTo}(L_1)$, $r_2''\#l_2''v_2'' \in SU$ und $(r_2'', l_2'', v_2'') \in \mathit{SkipUntil}(L_2')$. Wegen $r_1' \sqsubset_p r_1$ und $r_1l_1r_2 \sqsubset_p r_1'l_1'$ existiert ein $r' \in \Sigma^*$ mit $r_1'r' = r_1$ und $r'l_1r_2 \sqsubset_p l_1'$. Aus $r'l_1r_2 \sqsubset_p l_1'r_2'r_3'$ folgt $r'l_1r_2 \sqsubset_p l_1'r_2'l_2'$ und damit $r'l_1r_2 \sqsubset_p r_2''l_1''r_2''l_2''v_2''$. Ist $r' \sqsubset_p r_1''$, dann kann $(r_1'', l_1'', r_2''l_2''v_2'')$ die Sprachbedingung von $\mathit{SkipTo}(L_1)$ nicht erfüllen. Das ist ein Widerspruch zu der Annahme, dass $(r_1'', l_1'', r_2''l_2''v_2'') \in \mathit{SkipTo}(L_1)$ gilt. Ist $r_1'' \sqsubset_p r'$, dann gilt $r_1'r_1'' \sqsubset_p r_1'r' = r_1$ und $(r_1, l_1, r_2l_2v_2)$ kann nicht die Sprachbedingung von $\mathit{SkipTo}(L_1)$ erfüllen, da zusätzlich $r_1'r_1''l_1'' \sqsubseteq_p r_1l_1r_2l_2v_2$ gilt. Das ist ein Widerspruch zu der Annahme, dass $(r_1, l_1, r_2l_2v_2) \in \mathit{SkipTo}(L_1)$ gilt. Also ist $w'' \notin STSU$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $STSU' \subset STSU$ gilt. Deshalb kann Fall (b) nicht auftreten.

Da weder Fall (a) noch (b) gilt, kann auch Fall (3) nicht auftreten.

Wir haben alle drei Fälle widerlegt und damit gezeigt, dass $(r_1, l_1, r_2 l_2 v_2)$ die Startbedingung der Definition von $SkipTo(L'_1)$ erfüllt.

■

Lernbarkeit von Regeln mit zwei *SkipTo*-Operationen und einer *SkipUntil*-Operation

Sei $\mathcal{ST}^{(2)}\mathcal{SU}(\mathcal{L}_f)$ die Menge aller Regeln mit zwei aufeinanderfolgenden *SkipTo*-Operationen und einer *SkipUntil*-Operationen, d.h. $\mathcal{ST}^{(2)}\mathcal{SU}(\mathcal{L}) = \{ST^{(2)}SU(L_1, L_2, L_3) \mid L_1, L_2, L_3 \in \mathcal{L}\}$. Wir zeigen nun, dass diese Menge für endliche Sprachen nicht lernbar ist.

Theorem 7 *Es gilt* $\mathcal{ST}^{(2)}\mathcal{SU}(\mathcal{L}_f) \notin LimTxt$.

Beweis. Seien $\Sigma = \{a, b, c, d, e, f\}$, $L_1 = \{cd\}$, $L_2 = \{ba, bb, bc, bd, be, bf\}$ und $L_3 = \{ef\}$. Wir werden beweisen, dass keine endliche Telltale-Menge von $ST^{(2)}SU(L_1, L_2, L_3)$ existiert, d.h. für jede endliche Menge $T \subseteq ST^{(2)}SU(L_1, L_2, L_3)$ gibt es ein ST_2SU', so dass $T \subseteq ST_2SU'$ und $ST_2SU' \subset ST^{(2)}SU(L_1, L_2, L_3)$ gilt. Damit gilt aber nach Theorem 1 $\mathcal{ST}^{(2)}\mathcal{SU}(\mathcal{L}_f) \notin LimTxt$.
Zur besseren Übersichtlichkeit treffen wir für die restliche Beweisführung folgende Vereinbarung: $ST_2SU = ST^{(2)}SU(L_1, L_2, L_3)$, $STSU = STSU(L_2, L_3)$ und $SU = SU(L_3)$.
Für den Beweis nehmen wir das Gegenteil an: Sei $T \subseteq ST_2SU$ eine Telltale-Menge von ST_2SU, d.h. es existiert keine Menge ST_2SU', so dass $T \subseteq ST_2SU'$ und $ST_2SU' \subset ST_2SU$ gilt.
Angenommen, es gelte $m = max\{|w| \mid w \in T\}$ und es sei $ST_2SU' = ST^{(2)}SU(L_1', L_2', L_3')$ mit $L_1' = \{cdub \mid u \in \Sigma^* \wedge |u| \leq m \wedge b \text{ ist kein Teilwort von } u\}$, $L_2' = \{a, b, c, d, e, f\}$ und $L_3' = \{ef\}$. Seien außerdem $STSU' = STSU(L_2', L_3')$ und $SU' = SU(L_3')$. Wir zeigen nun, dass Folgendes gilt:

Fall (1) $T \subseteq ST_2SU'$

Fall (2) $ST_2SU' \subset ST_2SU$

Fall (1): Sei $w \in T$. Aus $T \subseteq ST_2SU$ folgt $w \in ST_2SU$. Damit existieren $r_1, r_2, r_3 \in \Sigma^*$, $v_3 \in \Sigma^*$ und $l_1 \in L_1, l_2 \in L_2, l_3 \in L_3$ mit $w = r_1l_1r_2l_2r_3\#l_3v_3$, $r_2l_2r_3\#l_3v_3 \in STSU$, $r_3\#l_3v_3 \in SU$, $(r_1, l_1, r_2l_2r_3l_3v_3) \in SkipTo(L_1)$, $(r_2, l_2, r_3l_3v_3) \in SkipTo(L_2)$ und $(r_3, l_3, v_3) \in SkipUntil(L_3)$. Dann gilt $l_1r_2l_2 \in \{cdu_1bu_2 \mid u_1 \in \Sigma^* \wedge b \text{ ist keine Teilwort von } u_1 \wedge u_2 \in \Sigma\}$ und es existieren $l_1' \in \{cdub \mid u \in \Sigma^* \wedge b \text{ ist keine Teilwort von } u\}$, $r_2' = \varepsilon$ sowie $l_2' \in \Sigma$ mit $l_1'r_2'l_2' = r_1l_1r_2l_2$.

Wir zeigen nun, dass $(r_1, l_1', r_2'l_2'r_3l_3v_3) \in SkipTo(L_1')$ gilt. Es sind $r_1 \in \Sigma^*$ und $r_2'l_2'r_3l_3v_3 \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Wegen $w \leq m$ gilt $l_1' \leq m$. Weil außerdem $l_1' \in \{cdub \mid u \in \Sigma^* \wedge b \text{ ist keine Teilwort von } u\}$ gilt, ist $l_1' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Da kein Element von L_1' ein Präfix besitzt, das auch Element von L_1' ist, kann auch kein $l' \in L_1'$ existieren mit $l' \sqsubset_p l_1'$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $(r_1, l_1, r_2l_2v_2) \in SkipTo(L_1)$ existieren keine $r' \in \Sigma^*, l' \in L_1$ mit $r' \sqsubset_p r_1 \wedge r'l' \sqsubseteq_p r_1l_1r_2l_2r_3l_3v_3$. Damit ist cd kein Teilwort von r_1 und es existieren auch keine $r' \in \Sigma^*, l' \in L_1'$ mit $r' \sqsubset_p r_1 \wedge r'l' \sqsubseteq_p r_1l_1'r_2'l_2'r_3l_3v_3$ [Die Startbedingung ist erfüllt]. Da $(r_1, l_1', r_2'l_2'r_3l_3v_3)$ alle Bedingungen der Definition von $SkipTo(L_1')$ erfüllt, gilt $(r_1, l_1', r_2'l_2'r_3l_3v_3) \in SkipTo(L_1')$.

Wir zeigen weiter, dass $(r'_2, l'_2, r_3 l_3 v_3) \in \mathit{SkipTo}(L'_2)$ gilt. Es sind $r_3 l_3 v_3 \in \Sigma^+$ und $r'_2 = \varepsilon$ [Die Randbedingungen sind erfüllt]. Wegen $l'_2 \in \Sigma$ ist $l'_2 \in L'_2$ [Die Sprachbedingung ist erfüllt] und es existiert kein Teilwort von l'_2, das auch Element von L'_2 ist. Somit gilt $\neg\exists l' \in L'_2$ mit $l' \sqsubset_p l'_2$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $r'_2 = \varepsilon$ existieren außerdem keine $r' \in \Sigma^*$, $l' \in L'_2$ mit $r' \sqsubset_p r'_2 \wedge r'l' \sqsubseteq_p r'_2 l'_2 r_3 l_3 v_3$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r'_2, l'_2, r_3 l_3 v_3) \in \mathit{SkipTo}(L'_2)$.

Wegen $L_3 = L'_3$ gilt $(r_3, l_3, v_3) \in \mathit{SkipUntil}(L'_3)$. Somit sind $r_3 \# l_3 v_3 \in SU'$ und $r'_2 l'_2 r_3 \# l_3 v_3 \in STSU'$. Zusätzlich gilt $(r_1, l'_1, r'_2 l'_2 r_3 l_3 v_3) \in \mathit{SkipTo}(L'_1)$. Daraus folgt $w = r_1 l_1 r_2 l_2 r_3 \# l_3 v_3 = r_1 l'_1 r'_2 l'_2 r_3 \# l_3 v_3 \in ST_2SU'$ und wir haben bewiesen, dass $T \subseteq ST_2SU'$ gilt.

Fall (2): Sei $w' \in ST_2SU'$. Es existieren also $r'_1, r'_2, r'_3 \in \Sigma^*$, $v'_3 \in \Sigma^*$ und $l'_1 \in L'_1, l'_2 \in L'_2, l'_3 \in L'_3$ mit $w' = r'_1 l'_1 r'_2 l'_2 r'_3 \# l'_3 v'_3$, $r'_2 l'_2 r'_3 \# l'_3 v'_3 \in STSU'$, $r'_3 \# l'_3 v'_3 \in SU'$, $(r'_1, l'_1, r'_2 l'_2 r'_3 l'_3 v'_3) \in \mathit{SkipTo}(L'_1)$, $(r'_2, l'_2, r'_3 l'_3 v'_3) \in \mathit{SkipTo}(L'_2)$ und $(r'_3, l'_3, v'_3) \in \mathit{SkipUntil}(L'_3)$. Dann gilt $l'_1 r'_2 l'_2 \in \{cdu_1 bu_2 \mid u_1 \in \Sigma^* \wedge b \text{ ist keine Teilwort von } u_1 \wedge u_2 \in \Sigma\}$ und es existieren $l''_1 = cd$, $r''_2 \in \Sigma^*$ und $l''_2 \in \{bu \mid u \in \Sigma\}$ mit $l''_1 r''_2 l''_2 = l'_1 r'_2 l'_2$, so dass b kein Teilwort von r''_2 ist.

Wir beweisen, dass $(r'_1, l''_1, r''_2 l''_2 r'_3 l'_3 v'_3) \in \mathit{SkipTo}(L_1)$ gilt. Es sind $r'_1 \in \Sigma^*$ und $r''_2 l''_2 r'_3 l'_3 v'_3 \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Wegen $l''_1 = cd$ gilt $l''_1 \in L_1$ [Die Sprachbedingung ist erfüllt]. Da L_1 nur ein Element besitzt, kann auch kein $l' \in L_1$ existieren mit $l' \sqsubset_p l''_1$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $(r'_1, l'_1, r'_2 l'_2 r'_3 l'_3 v'_3) \in \mathit{SkipTo}(L'_1)$ existieren keine $r' \in \Sigma^*$ und $l' \in L'_1$ mit $r' \sqsubset_p r'_1$ und $r'l' \sqsubseteq_p r'_1 l'_1 r'_2 l'_2 r'_3 l'_3 v'_3$. Daraus folgt, dass cd kein Teilwort von r'_1 ist. Damit existieren auch keine $r' \in \Sigma^*$, $l' \in L_1$ mit $r' \sqsubset_p r'_1 \wedge r'l' \sqsubseteq_p r'_1 l''_1 r''_2 l''_2 r'_3 l'_3 v'_3$ [Die Startbedingung ist erfüllt]. Da $(r'_1, l''_1, r''_2 l''_2 r'_3 v'_3)$ alle Bedingungen der Definition von $\mathit{SkipTo}(L_1)$ erfüllt, gilt $(r'_1, l''_1, r''_2 l''_2 r_3 l_3 v_3) \in \mathit{SkipTo}(L'_1)$.

Wir zeigen weiter, dass $(r''_2, l''_2, r'_3 l'_3 v'_2) \in \mathit{SkipTo}(L'_2)$ gilt. Es sind $r'_3 l'_3 v'_3 \in \Sigma^+$ und $r''_2 \in \Sigma^*$ [Die Randbedingungen sind erfüllt]. Da $l''_2 \in \{bu \mid u \in \Sigma\}$ gilt, ist $l''_2 \in L_2$ [Die Sprachbedingung ist erfüllt]. Kein Element aus L_2 besitzt ein Teilwort, das auch Element von L_2 ist. Somit gilt $\neg\exists l' \in L_2$ mit $l' \sqsubset_p l''_2$ [Die Minimalitätsbedingung ist erfüllt]. b ist kein Teilwort von r''_2. Deshalb existieren keine $r' \in \Sigma^*$, $l' \in L_2$ mit $r' \sqsubset_p r''_2 \wedge r'l' \sqsubseteq_p r''_2 l''_2 r'_3 l'_3 v'_3$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r''_2, l''_2, r'_3 l'_3 v'_3) \in \mathit{SkipTo}(L_2)$.

Wegen $L_3 = L'_3$ gilt $(r'_3, l'_3, v'_3) \in \mathit{SkipUntil}(L_3)$. Somit sind $r'_3 \# l'_3 v'_3 \in SU$ und $r''_2 l''_2 r'_3 \# l'_3 v'_3 \in STSU$. Zusätzlich gilt $(r'_1, l''_1, r''_2 l''_2 r'_3 l'_3 v'_3) \in \mathit{SkipTo}(L_1)$. Daraus folgt $w = r'_1 l'_1 r'_2 l'_2 r'_3 \# l'_3 v'_3 = r'_1 l''_1 r''_2 l''_2 r'_3 \# l'_3 v'_3 \in ST_2SU$ und wir haben bewiesen, dass $ST_2SU' \subseteq ST_2SU$ gilt.

Sei $w'' \in \{cdu_1 ba \# ef \mid u_1 \in \Sigma^* \wedge |u_1| > m \wedge cd \text{ und } b \text{ sind keine Teilworte von } u_1\}$. Offensichtlich ist $w'' \in ST_2SU$. Da $|u_1| > m$ gilt und cd kein Teilwort von u_1 ist, existiert kein Teilwort von w'', das Element von L'_1 ist. Damit gibt es auch keine Zerlegung, die Element von $\mathit{SkipTo}(L'_1)$ ist und es gilt $w'' \notin ST_2SU'$. Somit ist $ST_2SU \backslash ST_2SU' \neq \emptyset$. Aus $ST_2SU' \subseteq ST_2SU$

und $ST_2SU \backslash ST_2SU' \neq \emptyset$ folgt $ST_2SU' \subset ST_2SU$.

Wir haben gezeigt, dass für T ein ST_2SU' existiert, so dass $T \subseteq ST_2SU'$ und $ST_2SU' \subset ST_2SU$ gilt. Dies ist aber ein Widerspruch zu unserer Annahmen, dass T eine endliche Telltale-Menge ist. Da T beliebig ist, existiert keine endliche Telltale-Menge von $ST^{(2)}SU(L_1, L_2, L_3)$. Damit gilt $\mathcal{ST}^{(2)}\mathcal{SU}(\mathcal{L}_f) \notin LimTxt$.

■

Lernbarkeit von Regeln mit mehr als zwei *SkipTo*-Operationen und einer *SkipUntil*-Operationen

Sei $\mathcal{ST}^{(n)}\mathcal{SU}(\mathcal{L}_f)$ die Menge aller Regeln mit n aufeinanderfolgenden *SkipTo*-Operationen und einer *SkipUntil*-Operationen, d.h. $\mathcal{ST}^{(n)}\mathcal{SU}(\mathcal{L}) = \{ST^{(n)}SU(L_1, ..., L_{n+1}) \mid L_1, ..., L_{n+1} \in \mathcal{L}\}$. Wir zeigen nun, dass diese Menge für endliche Sprachen und $n \geq 3$ nicht lernbar ist.

Theorem 8 *Sei $n \geq 3$. Dann gilt $\mathcal{ST}^{(n)}\mathcal{SU}(\mathcal{L}_f) \notin LimTxt$.*

Beweis. Seien $\Sigma = \{a, b, c, d, e, f\}$, $L_1 = \{ab\}$, $L_2 = \{cd\}$, $L_3 = \{af, bf, cf, df, ef, ff\}$ und $L_j = \{ef\}$ für alle $4 \leq j \leq n + 1$. Wir werden beweisen, dass keine endliche Telltale-Menge von $ST^{(n)}SU(L_1, ..., L_{n+1})$ existiert, d.h. für jede endliche Menge $T \subseteq ST^{(n)}SU(L_1, ..., L_{n+1})$ existiert ein ST_nSU', so dass $T \subseteq ST_nSU'$ und $ST_nSU' \subset ST^{(n)}SU(L_1, ..., L_{n+1})$ gilt. Damit gilt aber nach Theorem 1 $\mathcal{ST}^{(n)}\mathcal{SU}(\mathcal{L}_f) \notin LimTxt$. Zur besseren Übersichtlichkeit treffen wir für den restlichen Beweis folgende Vereinbarungen: $ST_nSU = ST^{(n)}SU(L_1, ..., L_{n+1})$, $ST_{n-(i-1)}SU = ST^{(n-(i-1))}SU(L_i, ..., L_{n+1})$ für ein $i \in \{1, ..., n\}$ und $SU = SU(L_{n+1})$.
Für den Beweis nehmen wir das Gegenteil an: Sei $T \subseteq ST_nSU$ eine Telltale-Menge von ST_nSU, d.h. es existiert keine Menge ST_nSU', so dass $T \subseteq ST_nSU'$ und $ST_nSU' \subset ST_nSU$ gilt.
Angenommen, es gelte $m = max\{|w| \mid w \in T\}$ und es sei $ST_nSU' = ST^{(n)}SU(L'_1, ..., L'_{n+1})$ mit $L'_1 = \{abucd \mid u \in \Sigma^* \wedge |u| \leq m \wedge cd \text{ ist kein Teilwort von } u\}$, $L'_2 = \{a, b, c, d, e, f\}$, $L'_3 = \{f\}$ sowie $L'_j = \{ef\}$ für alle $4 \leq j \leq n+1$. Sei außerdem $ST_{n-(i-1)}SU' = ST^{(n-(i-1))}SU(L'_i, ..., L'_{n+1})$ für ein $i \in \{1, ..., n\}$ und $SU' = SU(L'_{n+1})$. Wir zeigen nun, dass Folgendes gilt:

Fall (1) $T \subseteq ST_nSU'$

Fall (2) $ST_nSU' \subset ST_nSU$

Fall (1): Sei $w \in T$. Aus $T \subseteq ST_nSU$ folgt $w \in ST_nSU$. Damit existieren $r_1, ..., r_{n+1}, v_{n+1} \in \Sigma^*$ und $l_1 \in L_1, ..., l_{n+1} \in L_{n+1}$ mit $w = r_1l_1...r_nl_nr_{n+1}\#l_{n+1}v_{n+1}$, so dass Folgendes gilt: $r_il_i...r_{n+1}\#l_{n+1}v_{n+1} \in ST_{n-(i-1)}SU$ und $(r_i, l_i, r_{i+1}...l_{n+1}v_{n+1}) \in SkipTo(L_i)$ für alle $1 \leq i \leq n$ sowie $r_{n+1}\#l_{n+1}v_{n+1} \in SU$ und $(r_{n+1}, l_{n+1}, v_{n+1}) \in SkipUntil(L_{n+1})$. Zuerst zeigen wir, dass $(r_1, l_1r_2l_2, r_3...l_{n+1}v_{n+1}) \in SkipTo(L'_1)$ gilt. Es gilt $r_1 \in \Sigma^*$ und $r_3...l_{n+1}v_{n+1} \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Aus der Definition von L_1 und L_2 folgt, dass $l_1 = ab$ und $l_2 = cd$ gilt. Weiterhin gilt $r_2 < m$ und wegen $(r_2, l_2, r_3l_3...r_{n+1}l_{n+1}v_{n+1}) \in SkipTo(L_2)$ ist cd kein Teilwort von r_2. Damit gilt $l_1r_2l_2 = abr_2cd \in L'_1$ [Die Sprachbedingung ist erfüllt]. Kein Element von L'_1 besitzt ein Präfix, das auch Element von L'_1 ist. Dann gilt aber auch $\neg\exists l'_1 \in L'_1$ mit $l'_1 \sqsubset_p l_1r_2l_2$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $(r_1, l_1, r_2l_2...r_{n+1}l_{n+1}v_{n+1}) \in SkipTo(L_1)$ ist ab kein Teilwort von r_1 und damit existieren keine $r'_1 \in \Sigma^*$, $l'_1 \in L'_1$ mit $r'_1 \sqsubset_p r_1 \wedge r'_1l'_1 \sqsubseteq_p r_1l_1...r_{n+1}l_{n+1}v_{n+1}$ [Die Startbedingung ist erfüllt]. Da $(r_1, l_1r_2l_2, r_3...l_{n+1}v_{n+1})$ alle Bedingungen der Definition von $SkipTo(L'_1)$ erfüllt gilt $(r_1, l_1r_2l_2, r_3...l_{n+1}v_{n+1}) \in SkipTo(L'_1)$.

Wegen $(r_3, l_3, r_4l_4...r_{n+1}l_{n+1}v_{n+1}) \in SkipTo(L_3)$ ist $r_3l_3 \in \{u_1u_2f \mid u_1 \in \Sigma \wedge u_2 \in \Sigma^* \wedge f$

ist kein Teilwort von $u_2\}$. Dadurch existieren $l'_2 \in L'_2$, $r'_3 \in \Sigma^*$ und $l'_3 = f \in L'_3$, so dass f kein Teilwort von r'_3 ist und $l'_2 r'_3 l'_3 = r_3 l_3$ gilt. Wir zeigen nun, dass $(r'_2, l'_2, r'_3 l'_3 r_4 l_4 ... r_{n+1} l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_2) mit $r'_2 = \varepsilon$ gilt. Es sind $r'_3 l'_3 r_4 ... l_{n+1} v_{n+1} \in \Sigma^+$ und $r'_2 = \varepsilon$ [Die Randbedingungen sind erfüllt] sowie $l'_2 \in L'_2$ [Die Sprachbedingung ist erfüllt]. Da $l'_2 \in \Sigma$ gilt, existiert kein Teilwort von l'_2, das Element von L'_2 ist. Somit gilt $\neg\exists l''_2 \in L'_2$ mit $l''_2 \sqsubset_p l'_2$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $r'_2 = \varepsilon$ existieren außerdem keine $r''_2 \in \Sigma^*$, $l''_2 \in L'_2$ mit $r''_2 \sqsubset_p r'_2 \wedge r''_2 l''_2 \sqsubseteq_p r'_2 l'_2 r'_3 l'_3 r_4 l_4 ... r_{n+1} l_{n+1} v_{n+1}$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(\varepsilon, l'_2, r'_3 l'_3 r_4 l_4 ... r_{n+1} l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_2). Wir zeigen weiter, dass $(r'_3, l'_3, r_4 l_4 ... r_{n+1} l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_3) gilt. Es sind $r'_3 \in \Sigma^*$ und $r_4 ... l_{n+1} v_{n+1} \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Außerdem gilt $l'_3 = f \in L'_3$ [Die Sprachbedingung ist erfüllt]. Damit existiert aber auch kein $l''_3 \in L'_3$ mit $l''_3 \sqsubset_p l'_3$ [Die Minimalitätsbedingung ist erfüllt]. Da f kein Teilwort von r'_3 ist, existieren auch keine $r''_3 \in \Sigma^*$, $l''_3 \in L'_3$ mit $r''_3 \sqsubset_p r'_3 \wedge r''_3 l''_3 \sqsubseteq_p r'_3 l'_3 r_4 l_4 ... r_{n+1} l_{n+1} v_{n+1}$ [Die Startbedingung ist erfüllt]. $(r'_3, l'_3, r_4 l_4 ... r_{n+1} l_{n+1} v_{n+1})$ erfüllt alle Bedingungen der Definition von *SkipTo*(L'_3) und es gilt $(r'_3, l'_3, r_4 l_4 ... r_{n+1} l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_3).

Wegen $L_j = L'_j$ gilt $(r_j, l_j, r_{j+1} ... l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_j) für alle $4 \leq j \leq n$. Da auch $L_{n+1} = L'_{n+1}$ ist, gilt $(r_{n+1}, l_{n+1}, v_{n+1}) \in$ *SkipUntil*(L'_{n+1}). Somit sind $r'_2 l'_2 r'_3 l'_3 r_4 ... r_{n+1} \# l_{n+1} v_{n+1} \in ST_{n-1}SU'$, $r'_3 l'_3 r_4 ... r_{n+1} \# l_{n+1} v_{n+1} \in ST_{n-2}SU'$, $r_j l_j ... r_{n+1} \# l_{n+1} v_{n+1} \in ST_{n-(j-1)}SU'$ für alle $4 \leq j \leq n$ und $r_n l_n \# v_n \in SU'$. Da außerdem $(r_1, l_1 r_2 l_2, r_3 ... l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_1) gilt, ist $w = r_1 l_1 ... r_{n+1} \# l_{n+1} v_{n+1} = r_1 l_1 r_2 l_2 l'_2 r'_3 l'_3 r_4 l_4 ... r_{n+1} \# l_{n+1} v_{n+1} \in ST_n SU'$ und wir haben bewiesen, dass $T \subseteq ST_n SU'$ gilt.

Fall (2): Sei $w' \in ST_n SU'$. Es existieren also $r'_1, ..., r'_{n+1}, v'_{n+1} \in \Sigma^*$ und $l'_1 \in L'_1, ..., l'_{n+1} \in L'_{n+1}$ mit $w' = r'_1 l'_1 ... r'_{n+1} \# l'_{n+1} v'_{n+1}$, $r'_i l'_i ... r'_{n+1} \# l'_{n+1} v'_{n+1} \in ST_{n-(i-1)}SU'$ und $(r'_i, l'_i, r'_{i+1} ... l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L'_i) für alle $1 \leq i \leq n$, $r'_{n+1} \# l'_{n+1} v'_{n+1} \in SU'$ und $(r'_{n+1}, l'_{n+1}, v'_{n+1}) \in$ *SkipUntil*(L'_{n+1}). Wegen $(r'_1, l'_1, r'_2 ... l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L'_1) gilt $r'_1 l'_1 \in \{u_1 a b u_2 c d \mid u_1 \in \Sigma^* \wedge ab$ ist keine Teilwort von $u_1 \wedge u_2 \in \Sigma^* \wedge cd$ ist kein Teilwort von $u_2\}$. Damit existieren $r''_1, r''_2 \in \Sigma^*, l''_1 = ab$ und $l''_2 = cd$, so dass $r''_1 l''_1 r''_2 l''_2 = r'_1 l'_1$ gilt, ab kein Teilwort von r''_1 und cd kein Teilwort von r''_2 ist. Zuerst zeigen wir, dass $(r''_1, l''_1, r''_2 l''_2 r'_2 l'_2 ... r'_{n+1} l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L_1) gilt. Es sind $r''_1 \in \Sigma^*$ und $r''_2 l''_2 r'_2 ... l'_{n+1} v'_{n+1} \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l''_1 = ab \in L_1$ [Die Sprachbedingung ist erfüllt]. Es existiert nur ein Element in L_1. Damit gilt $\neg\exists l' \in L_1$ mit $l' \sqsubset_p l''_1$ [Die Minimalitätsbedingung ist erfüllt]. Da ab kein Teilwort von r''_1 ist, existieren keine $r' \in \Sigma^*$, $l' \in L_1$ mit $r' \sqsubset_p r''_1 \wedge r' l' \sqsubseteq_p r''_1 l''_1 r''_2 l''_2 r'_2 l'_2 ... r'_{n+1} l'_{n+1} v'_{n+1}$ [Die Startbedingung ist erfüllt]. Alle Bedingungen sind erfüllt und somit gilt $(r''_1, l''_1, r''_2 l''_2 r'_2 ... l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L_1).

Wir beweisen weiter, dass $(r''_2, l''_2, r'_2 l'_2 ... r'_{n+1} l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L_2) gilt. Es sind $r''_2 \in \Sigma^*$ und $r'_2 ... l'_{n+1} v'_{n+1} \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l''_2 = cd \in L_2$ [Die Sprachbedingung ist erfüllt]. Es existiert nur ein Element in L_2. Damit gilt $\neg\exists l' \in L_2$ mit $l' \sqsubset_p l''_2$ [Die Minimalitätsbedingung ist erfüllt]. Da cd kein Teilwort von r''_2 ist, existieren keine $r' \in \Sigma^*$, $l' \in L_2$ mit $r' \sqsubset_p r''_2 \wedge r' l' \sqsubseteq_p r''_2 l''_2 r'_2 l'_2 ... r'_{n+1} l'_{n+1} v'_{n+1}$ [Die Startbedingung ist erfüllt]. Alle Bedingungen sind erfüllt und somit gilt $(r''_2, l''_2, r'_2 l'_2 ... r'_{n+1} l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L_2).

Aus $(r'_2, l'_2, r'_3 l'_3 ... r'_{n+1} l'_{n+1} v'_{n+1}) \in \mathit{SkipTo}(L'_2)$ und $(r'_3, l'_3, r'_4 l'_4 ... r'_{n+1} l'_{n+1} v'_{n+1}) \in \mathit{SkipTo}(L'_3)$ folgt $r'_2 = \varepsilon$, $l'_2 \in \Sigma$, f ist kein Teilwort von r'_3 sowie $l'_3 = f$ und damit $r'_2 l'_2 r'_3 l'_3 \in \{u_1 u_2 f \mid u_1 \in \Sigma \wedge u_2 \in \Sigma^* \wedge f$ ist kein Teilwort von $u_2\}$. Somit existieren $r''_3 \in \Sigma^*$ und $l''_3 \in L_3$, so dass $r''_3 l''_3 = r'_2 l'_2 r'_3 l'_3$ gilt. Wir zeigen nun, dass $(r''_3, l''_3, r'_4 l'_4 ... r'_{n+1} l'_{n+1} v'_{n+1}) \in \mathit{SkipTo}(L_3)$ gilt. Es sind $r''_3 \in \Sigma^*$ und $r'_4 ... l'_{n+1} v'_{n+1} \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l''_3 \in L_3$ [Die Sprachbedingung ist erfüllt]. In L_3 existiert kein Element, das ein Teilwort besitzt, was auch Element von L_3 ist. Damit gilt $\neg \exists l' \in L_3$ mit $l' \sqsubset_p l''_3$ [Die Minimalitätsbedingung ist erfüllt]. Weil $r''_3 l''_3 \in \{u_1 u_2 f \mid u_1 \in \Sigma \wedge u_2 \in \Sigma^* \wedge f$ ist kein Teilwort von $u_2\}$ gilt, existieren keine $r' \in \Sigma^*$, $l' \in L_3$ mit $r' \sqsubset_p r''_3 \wedge r' l' \sqsubseteq_p r''_3 l''_3 r'_3 l'_3 ... r'_{n+1} l'_{n+1} v'_{n+1}$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, haben wir bewiesen, dass $(r''_3, l''_3, r'_4 l'_4 ... r'_{n+1} l'_{n+1} v'_{n+1}) \in \mathit{SkipTo}(L_3)$ gilt.

Wegen $L_j = L'_j$ gilt $(r'_j, l'_j, r'_{j+1} ... l'_{n+1} v'_{n+1}) \in \mathit{SkipTo}(L_j)$ für alle $4 \leq j \leq n$. Da auch $L_{n+1} = L'_{n+1}$ ist, gilt $(r'_{n+1}, l'_{n+1}, v'_{n+1}) \in \mathit{SkipUntil}(L_{n+1})$. Somit sind $r''_2 l''_2 r''_3 l''_3 r'_4 ... r'_{n+1} \# l'_{n+1} v'_{n+1} \in ST_{n-1}SU$, $r''_3 l''_3 r'_4 ... r'_{n+1} \# l'_{n+1} v'_{n+1} \in ST_{n-2}SU$, $r'_j l'_j ... r'_{n+1} \# l'_{n+1} v'_{n+1} \in ST_{n-(j-1)}SU$ für alle $4 \leq j \leq n$ und $r'_{n+1} \# l'_{n+1} v'_{n+1} \in SU$. Da außerdem $(r''_1, l''_1, r''_2 l''_2 r''_3 l''_3 r'_4 ... l'_{n+1} v'_{n+1}) \in \mathit{SkipTo}(L_1)$ gilt, ist $w = r'_1 l'_1 ... r'_{n+1} \# l'_{n+1} v'_{n+1} = r''_1 l''_1 r''_2 l''_2 r''_3 l''_3 r'_4 l'_4 ... r'_{n+1} \# l'_{n+1} v'_{n+1} \in ST_nSU$ und wir haben bewiesen, dass $ST_nSU' \subseteq ST_nSU$ gilt.

Sei $w'' \in \{abu_1 cdafu_2^{n-3} \# ef \mid u_1 \in \Sigma^* \wedge |u_1| > m \wedge cd$ ist kein Teilwort von $u_1 \wedge u_2 = ef\}$. Offensichtlich ist $w'' \in ST_nSU$. Da $|u_1| > m$ gilt, existiert kein Teilwort von w'', das Element von L'_1 ist. Damit gibt es auch keine Zerlegung, die Element von $\mathit{SkipTo}(L'_1)$ ist und es gilt $w'' \notin ST_nSU'$. Somit ist $ST_nSU \backslash ST_nSU' \neq \emptyset$. Aus $ST_nSU' \subseteq ST_nSU$ und $ST_nSU \backslash ST_nSU' \neq \emptyset$ folgt $ST_nSU' \subset ST_nSU$.

Wir haben gezeigt, dass für T ein ST_nSU' existiert, so dass $T \subseteq ST_nSU'$ und $ST_nSU' \subset ST_nSU$ gilt. Dies ist aber ein Widerspruch zu unserer Annahmen, dass T eine endliche Telltale-Menge ist. Da T beliebig ist, existiert keine endliche Telltale-Menge von $ST^{(n)}SU(L_1, ..., L_{n+1})$. Damit gilt $\mathcal{ST}^{(n)}\mathcal{SU}(\mathcal{L}_f) \notin \mathit{LimTxt}$.

■

6.3 Lernbarkeit von Regeln, die eine *Next*-Operation enthalten

Zuletzt untersuchen wir Regeln, deren letzte Operation eine *Next*-Operation ist. Im Gegensatz zu anderen Regeln, wird hier schon nach der letzten *SkipTo*-Operation gehalten (falls die Regel *SkipTo*-Operationen enthält). Diese Regeln besitzen die zusätzliche Bedingung, dass ein Präfix des nach dem Haltepunkt folgenden Textes ein Element der Argumentensprache der *Next*-Operation sein muss.

Lernbarkeit von Regeln mit einer *Next*-Operation

Sei $\mathcal{NT}(\mathcal{L})$ die Menge aller Regeln, in denen eine *Next*-Operation durchgeführt wird, d.h. $\mathcal{NT}(\mathcal{L}) = \{NT(L) \mid L \in \mathcal{L}\}$. Wir werden nun zeigen, dass diese Menge für endliche Sprachen lernbar ist.

Theorem 9 $\mathcal{NT}(\mathcal{L}_f) \in LimTxt$.

Beweis. Wir wissen, dass eine Sprache aus Text lernbar ist, wenn wir eine Telltale-Menge für diese Sprache bilden können. Sei $L \in \mathcal{L}_f$. Wir definieren die Menge $T_{NT(L)} = \{\#l \mid (l, \varepsilon) \in \mathit{Next}(L)\}$.
Wir beweisen nun, dass $T_{NT(L)}$ eine Telltale-Menge von $NT(L)$ ist. Für den Beweis nehmen wir das Gegenteil an, d.h. seien $L, L' \in \mathcal{L}_f$ mit $T_{NT(L)} \subseteq NT(L') \subset NT(L)$.
Sei $w \in NT(L) \backslash NT(L')$ mit $w = \#lv$ und $(l, v) \in \mathit{Next}(L)$. Somit gilt $w \notin NT(L')$ und es existiert keine Zerlegung von w, die die Bedingungen der Definition von $\mathit{Next}(L')$ erfüllt. Insbesondere gilt auch $(l, v) \notin \mathit{Next}(L')$.
Die Randbedingungen sind wegen $(l, v) \in \mathit{Next}(L)$ immer erfüllt. Deshalb folgt aus der Definition, dass die Sprachbedingung verletzt sein muss, d.h. $l \notin L'$.

Angenommen, $l \notin L'$ gilt. Wir betrachten das Wort $w' = \#l$. Es gilt $w' \in T_{NT(L)}$. Aus $T_{NT(L)} \subseteq NT(L')$ folgt $w' \in NT(L')$. Da $l \notin L'$ gilt, muss eine andere Zerlegung für w' existieren, die die Bedingungen der Definition von $NT(L)$ erfüllt. Angenommen $\#l'v'$ mit $l = l'v'$ sei diese Zerlegung. Wir betrachten die Zerlegung $w = \#l'v'v$. Es gilt $(l', v'v) \in \mathit{Next}(L')$ und damit auch $w \in NT(L')$. Dies ist aber ein Widerspruch zu unserer Annahme, dass $w \in NT(L) \backslash NT(L')$ gilt. Somit gilt $(l, v) \in \mathit{Next}(L')$. Damit gilt aber auch wieder $w \in NT(L')$, was ein Widerpruch zu unserer Annahme ist, dass $w \in NT(L) \backslash NT(L')$ gilt. Es kann also nicht $T_{NT(L)} \subseteq NT(L') \subset NT(L)$ gelten. Wir haben somit bewiesen, dass $T_{NT(L)}$ eine Telltale-Menge von $NT(L)$ ist. Es gilt also $NT(L) \in LimTxt$. Da $NT(L)$ beliebig ist, gilt $\mathcal{NT}(\mathcal{L}) \in LimTxt$.
■

Lernbarkeit von Regeln mit einer *SkipTo*- und einer *Next*-Operation

Sei $\mathcal{STNT}(\mathcal{L})$ die Menge aller Regeln mit einer *SkipTo*- und einer *Next*-Operation, d.h. $\mathcal{STNT}(\mathcal{L}) = \{STNT(L_1, L_2) \mid L_1, L_2 \in \mathcal{L}\}$. Wir zeigen nun, dass diese Menge für endliche Sprachen lernbar ist.

Theorem 10 $\mathcal{STNT}(\mathcal{L}_f) \in LimTxt$.

Beweis. Wir wissen, dass eine Sprache aus Text lernbar ist, wenn wir eine Telltale-Menge für diese Sprache bilden können. Seien $L_1, L_2 \in \mathcal{L}_f$. Wir definieren die Menge $T_{STNT(L_1,L_2)} = \{l_1\#l_2 \mid (l_1, l_2) \in SkipTo(L_1)$ und $\#l_2 \in NT(L_2)\}$.
Seien $L_1, L_2 \in \mathcal{L}$. Wir beweisen nun, dass $T_{STNT(L_1,L_2)}$ eine Telltale-Menge von $STNT(L_1, L_2)$ ist. Für den Beweis nehmen wir das Gegenteil an, d.h. seien $L'_1, L'_2 \in \mathcal{L}_f$ und $T_{STNT(L_1,L_2)} \subseteq STNT(L'_1, L'_2) \subset STNT(L_1, L_2)$. Sei $w \in STNT(L_1, L_2) \backslash STNT(L'_1, L'_2)$. Daraus folgt, dass eine Zerlegung $w = r_1 l_1 \# l_2 v_2$ existiert mit $(r_1, l_1, l_2 v_2) \in SkipTo(L_1)$, $\#l_2 v_2 \in NT(L_2)$ und $(l_2, v_2) \in Next(L_2)$. Es gilt außerdem $w \notin STNT(L'_1, L'_2)$. Für jede Zerlegung von w sind somit entweder die Bedingungen der Definition von $SkipTo(L'_1)$ oder die Bedingungen der Definition von $NT(L'_2)$ nicht erfüllt. Insbesondere gilt einer der folgenden Fälle:

Fall (10.1) $(r_1, l_1, l_2 v_2) \notin SkipTo(L'_1)$

Fall (10.2) $\#l_2 v_2 \notin NT(L'_2)$

In Lemma 11 bzw. Lemma 10 beweisen wir, dass weder Fall (10.1) noch Fall (10.2) gelten kann. Damit gilt aber $w \in STNT(L'_1, L'_2)$. Dies ist ein Widerspruch zu unserer Annahme, dass $w \in STNT(L_1, L_2) \backslash STNT(L'_1, L'_2)$ gilt. Es kann also nicht $T_{STNT(L_1,L_2)} \subseteq STNT(L'_1, L'_2) \subset STNT(L_1, L_2)$ gelten. Wir haben somit bewiesen, dass $T_{STNT(L_1,L_2)}$ eine Telltale-Menge von $STNT(L_1, L_2)$ ist und es gilt $STNT(L_1, L_2) \in LimTxt$. Da $STNT(L_1, L_2)$ beliebig ist, gilt $\mathcal{STNT}(\mathcal{L}_f) \in LimTxt$.

■

Zur besseren Übersichtlichkeit der Beweise treffen wir für die restlichen Lemmata in diesem Kapitel folgende Vereinbarungen. Seien $L_1, L_2, L'_1, L'_2 \in \mathcal{L}_f$. Sei $T = T_{STNT(L_1,L_2)} = \{l_1\#l_2 \mid (\varepsilon, l_1, l_2) \in SkipTo(L_1), \#l_2 \in NT(L_2)\}$. Weiterhin gelte $STNT = STNT(L_1, L_2)$, $STNT' = STNT(L'_1, L'_2)$, $NT = NT(L_2)$ und $NT' = NT(L'_2)$.

Wir widerlegen zuerst Fall (10.2)

Lemma 10 *Seien $T \subseteq STNT' \subset STNT$ und $w \in STNT \backslash STNT'$ mit $w = r_1 l_1 \# l_2 v_2$. Weiterhin seien $(r_1, l_1, l_2 v_2) \in SkipTo(L_1)$, $\#l_2 v_2 \in NT$ und $(l_2, v_2) \in Next(L_2)$. Dann gilt $\#l_2 v_2 \in NT'$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. sei $\#l_2v_2 \notin NT'$. Dann existiert keine Zerlegung für $\#l_2v_2$, die die Bedingungen von $Next(L_2')$ erfüllt. Insbesondere gilt auch $(l_2, v_2) \notin Next(L_2')$.
Die Randbedingung ist wegen $(l_2, v_2) \in Next(L_2)$ immer erfüllt. Deshalb folgt aus der Definition, dass die Sprachbedingung verletzt sein muss, d.h. $l_2 \notin L_2'$.

Angenommen, $l_2 \notin L_2'$ gilt. Wir betrachten das Wort $w' = l_1\#l_2$. Dann gilt $w' \in T$. Aus $T \subseteq STNT'$ folgt $w' \in STNT'$. Da $l_2 \notin L_2'$ gilt, muss eine andere Zerlegung für w' existieren, die die Bedingungen der Definition von $STNT'$ erfüllt. Angenommen $r_1'l_1'\#l_2'v_2'$ sei diese Zerlegung mit $(r_1', l_1', l_2'v_2') \in SkipTo(L_1')$, $\#l_2'v_2' \in NT$ und $(l_2', v_2') \in Next(L_2')$. Wegen (l_2', v_2') $\in Next(L_2')$ gilt auch $(l_2', v_2'v_2) \in Next(L_2')$. Damit ist $\#l_2'v_2'v_2 \in NT'$. Dies ist ein Widerspruch zu unserer Annahme, dass $\#l_2'v_2'v_2 = \#l_2v_2 \notin NT'$ gilt. Somit haben wir bewiesen, dass $\#l_2v_2 \in NT'$ gilt.

■

Das nächste Lemma widerlegt Fall (10.1). Da wir in Lemma 10 bewiesen haben, dass Fall (10.2) nie gelten kann, können wir für den Beweis dieses Lemmas annehmen, dass Fall (10.2) nicht gilt.

Lemma 11 *Seien $T \subseteq STNT' \subset STNT$ und $w \in STNT \backslash STNT'$ mit $w = r_1l_1\#l_2v_2$. Außerdem gelte $(r_1, l_1, l_2v_2) \in SkipTo(L_1)$ und $\#l_2v_2 \in NT$. Wir nehmen außerdem o.B.d.A. an, dass $\#l_2v_2 \in NT'$ ist. Dann gilt $(r_1, l_1, l_2v_2) \in SkipTo(L_1')$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. sei $(r_1, l_1, l_2v_2) \notin SkipTo(L_1')$. Die Randbedingungen sind wegen $(r_1, l_1, l_2v_2) \in SkipTo(L_1)$ immer erfüllt. Dann muss einer der folgenden Fälle gelten:

Fall (1) $l_1 \notin L_1'$ [Die Sprachbedingung ist verletzt]

Fall (2) $\exists l_1' \in L_1'$ mit $l_1' \sqsubset_p l_1$ [Die Minimalitätsbedingung ist verletzt]

Fall (3) $\exists r_1' \in \Sigma^*, l_1' \in L_1'$ mit $r_1' \sqsubset_p r_1 \wedge r_1'l_1' \sqsubseteq_p r_1l_1l_2v_2$ [Die Startbedingung ist verletzt]

In den nachfolgenden Lemmata beweisen wir, dass alle drei Fälle nicht gelten können. Siehe Lemma 14 für Fall (1), Lemma 12 für Fall (2) und Lemma 13 für Fall (3). Damit gilt aber $(r_1, l_1, l_2v_2) \in SkipTo(L_1')$.

■

Lemma 12 beweist, dass Fall (2) aus Lemma 10 nicht gelten kann.

Lemma 12 *Seien $T \subseteq STNT' \subset STNT$ und $w \in STNT \backslash STNT'$ mit $w = r_1 l_1 \# l_2 v_2$. Weiterhin gelte $(r_1, l_1, l_2 v_2) \in SkipTo(L_1)$, $\# l_2 v_2 \in NT$ und $\# l_2 v_2 \in NT'$. Dann erfüllt $(r_1, l_1, l_2 v_2)$ die Minimalitätsbedingung der Definition von $SkipTo(L_1')$, d.h. $\neg \exists l_1' \in L_1'$ mit $l_1' \sqsubset_p l_1$.*

Beweis. Für den Beweis betrachten wir das Wort $w' = l_1 \# l_2$. Es gilt $w' \in T$. Aus $T \subseteq STNT'$ folgt $w' \in STNT'$. Dann muss ein $r' \in \Sigma^*$ und ein $l' \in L_1'$ existieren mit $r'l' = l_1$ und $(r', l', l_2) \in SkipTo(L_1')$. Angenommen, es existiert ein $l_1' \in L_1'$ mit $l_1' \sqsubset_p l_1$. Ist $r' \neq \varepsilon$, dann gilt aber $r_1' \sqsubset_p r'$ und $r_1' l_1' \sqsubseteq_p r' l' l_2$ mit $r_1' = \varepsilon$. Ist $r' = \varepsilon$, dann gilt $l_1' \sqsubset_p l'$. Somit erfüllt (r', l', l_2) entweder die Start- oder die Minimalitätsbedingung von $SkipTo(L_1')$ nicht und es gilt $(r', l', l_2) \notin SkipTo(L_1')$. Dies ist aber ein Widerspruch zu unserer Annahme, dass $(r', l', l_2) \in SkipTo(L_1')$ gilt. Damit haben wir bewiesen, dass kein $l_1' \in L_1'$ mit $l_1' \sqsubset_p l_1$ existieren kann.

■

Lemma 13 widerlegt Fall (3) aus Lemma 10.

Lemma 13 *Seien $T \subseteq STNT' \subset STNT$ und $w \in STNT \backslash STNT'$ mit $w = r_1 l_1 \# l_2 v_2$. Weiterhin gelte $(r_1, l_1, l_2 v_2) \in SkipTo(L_1)$, $\# l_2 v_2 \in NT$ und $\# l_2 v_2 \in NT'$. Dann erfüllt $(r_1, l_1, l_2 v_2)$ die Startbedingung der Definition von $SkipTo(L_1')$, d.h. $\neg \exists r_1' \in \Sigma^*, l_1' \in L_1'$ mit $r_1' \sqsubset_p r_1 \wedge r_1' l_1' \sqsubseteq_p r_1 l_1 l_2 v_2$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. es existiere ein $r_1' \in \Sigma^*$ und ein $l_1' \in L_1'$ mit $r_1' \sqsubset_p r_1 \wedge r_1' l_1' \sqsubseteq_p r_1 l_1 l_2 v_2$. Aus der Menge der $r_1' l_1'$, die diese Bedingung erfüllen, betrachten wir dabei das Paar mit dem kürzesten r_1' und einem l_1', so dass kein $l_1'' \in L_1'$ existiert mit $l_1'' \sqsubset_p l_1'$. Dann gilt einer der drei folgenden Fälle:

Fall (1) $r_1' l_1' \sqsubset_p r_1 l_1$

Fall (2) $r_1 l_1 \sqsubset_p r_1' l_1'$

Fall (3) $r_1 l_1 = r_1' l_1'$

Fall (1): Angenommen, $r_1' l_1' \sqsubset_p r_1 l_1$ gilt. Wir betrachten das Wort $w' = r_1' l_1' \# l_2 v_2$. Wir zeigen zuerst, dass $(r_1', l_1', l_2 v_2) \in SkipTo(L_1')$ gilt. Es sind $r_1' \in \Sigma^*$ und $l_2 v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l_1' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_1' und l_1', existieren kein $l_1'' \in L_1'$ mit $l_1'' \sqsubset_p l_1'$ [Die Minimalitätsbedingung ist erfüllt] und keine $r_1'' \in \Sigma^*$, $l_1'' \in L_1'$ mit $r_1'' \sqsubset_p r_1' \wedge r_1'' l_1'' \sqsubseteq_p r_1' l_1' l_2 v_2$ [Die Startbedingung ist erfüllt]. Da $(r_1', l_1', l_2 v_2)$ alle Bedingungen der Definition von $SkipTo(L_1')$ erfüllt, ist $(r_1', l_1', l_2 v_2) \in SkipTo(L_1')$. Da

außerdem $\#l_2v_2 \in NT'$ gilt, ist $w' \in STNT'$. Angenommen, $w' \in STNT$ gilt. Dann existiert eine Zerlegung $w' = r_1''l_1''\#l_2v_2$ mit $(r_1'', l_1'', l_2v_2) \in \mathit{SkipTo}(L_1)$. Aus $r_1'l_1' \sqsubset_p r_1l_1$ folgt $r_1''l_1'' \sqsubset_p r_1l_1$. Ist $r_1 \sqsubset_p r_1''$, dann kann (r_1'', l_1'', l_2v_2) die Sprachbedingung von $\mathit{SkipTo}(L_1)$ nicht erfüllen. Das ist ein Widerspruch zu der Annahme, dass $(r_1'', l_1'', l_2v_2) \in \mathit{SkipTo}(L_1)$ gilt. Ist $r_1'' \sqsubset_p r_1$, dann kann (r_1, l_1, l_2v_2) die Sprachbedingung von $\mathit{SkipTo}(L_1)$ nicht erfüllen, da zusätzlich $r_1''l_1'' = r_1'l_1' \sqsubseteq_p r_1l_1l_2v_2$ gilt. Das ist ein Widerspruch zu der Annahme, dass $(r_1, l_1, l_2v_2) \in \mathit{SkipTo}(L_1)$ gilt. Also ist $w' \notin STNT$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $STNT' \subset STNT$ gilt. Damit existieren keine $r_1' \in \Sigma^*, l_1' \in L_1'$ mit $r_1' \sqsubset_p r_1$ und $r_1'l_1' \sqsubset_p r_1l_1$.

Fall (2): Angenommen, $r_1l_1 \sqsubset_p r_1'l_1'$ gilt. Wir betrachten das Wort $w' = r_1'l_1'\#l_2v_2$. Wir zeigen zuerst, dass $(r_1', l_1', l_2v_2) \in \mathit{SkipTo}(L_1')$ gilt. Es sind $r_1' \in \Sigma^*$ und $l_2v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l_1' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_1' und l_1', existieren kein $l_1'' \in L_1'$ mit $l_1'' \sqsubset_p l_1'$ [Die Minimalitätsbedingung ist erfüllt] und keine $r_1'' \in \Sigma^*$, $l_1'' \in L_1'$ mit $r_1'' \sqsubset_p r_1' \wedge r_1''l_1'' \sqsubseteq_p r_1'l_1'l_2v_2$ [Die Startbedingung ist erfüllt]. Da (r_1', l_1', l_2v_2) alle Bedingungen der Definition von $\mathit{SkipTo}(L_1')$ erfüllt, ist $(r_1', l_1', l_2v_2) \in \mathit{SkipTo}(L_1')$. Da außerdem $\#l_2v_2 \in NT'$ gilt, ist $w' \in STNT'$. Angenommen, $w' \in STNT$ gilt. Dann existiert eine Zerlegung $w' = r_1''l_1''\#l_2v_2$ mit $(r_1'', l_1'', l_2v_2) \in \mathit{SkipTo}(L_1)$. Aus $r_1l_1 \sqsubset_p r_1'l_1'$ folgt $r_1l_1 \sqsubset_p r_1''l_1''$. Ist $r_1 \sqsubset_p r_1''$, dann kann (r_1'', l_1'', l_2v_2) die Sprachbedingung von $\mathit{SkipTo}(L_1)$ nicht erfüllen. Das ist ein Widerspruch zu der Annahme, dass $(r_1'', l_1'', l_2v_2) \in \mathit{SkipTo}(L_1)$ gilt. Ist $r_1'' \sqsubset_p r_1$, dann kann (r_1, l_1, l_2v_2) die Sprachbedingung von $\mathit{SkipTo}(L_1)$ nicht erfüllen, da zusätzlich $r_1''l_1'' = r_1'l_1' \sqsubseteq_p r_1l_1l_2v_2$ gilt. Das ist ein Widerspruch zu der Annahme, dass $(r_1, l_1, l_2v_2) \in \mathit{SkipTo}(L_1)$ gilt. Also ist $w' \notin STNT$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $STNT' \subset STNT$ gilt. Somit existieren keine $r_1' \in \Sigma^*, l_1' \in L_1'$ mit $r_1' \sqsubset_p r_1$ und $r_1l_1 \sqsubset_p r_1'l_1'$.

Fall (3): Angenommen $r_1l_1 = r_1'l_1'$ gilt. Dann betrachten wir die Zerlegung $w = r_1'l_1'\#l_2v_2$. Wir zeigen nun, dass $(r_1', l_1', l_2v_2) \in \mathit{SkipTo}(L_1')$ gilt. Es sind $r_1' \in \Sigma^*$ und $l_2v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l_1' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_1' und l_1', existieren kein $l_1'' \in L_1'$ mit $l_1'' \sqsubset_p l_1'$ [Die Minimalitätsbedingung ist erfüllt] und keine $r_1'' \in \Sigma^*$, $l_1'' \in L_1'$ mit $r_1'' \sqsubset_p r_1' \wedge r_1''l_1'' \sqsubseteq_p r_1'l_1'l_2v_2$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r_1', l_1', l_2v_2) \in \mathit{SkipTo}(L_1')$ und damit $w \in STNT'$. Dies ist aber ein Widerspruch zu unserer Annahme, dass $w \in STNT \backslash STNT'$ gilt. Es gibt also keine $r_1' \in \Sigma^*, l_1' \in L_1'$ mit $r_1' \sqsubset_p r_1$ und $r_1l_1 = r_1'l_1'$.

Da wir alle Fälle widerlegt haben, erfüllt (r_1, l_1, l_2v_2) die Startbedingung der Definition von $\mathit{SkipTo}(L_1')$.

■

Im nächsten Lemma beweisen, wir, dass (r_1, l_1, l_2v_2) die Sprachbedingung der Definition von $\mathit{SkipTo}(L_1')$ erfüllt, wenn $T \subseteq STNT' \subset STNT$ und $w \in STNT \backslash STNT'$ gilt mit $w =$

$r_1 l_1 \# l_2 v_2$, $(r_1, l_1, l_2 v_2) \in SkipTo(L_1)$, $\# l_2 v_2 \in NT$ und $\# l_2 v_2 \in NT'$. Um dies zu beweisen, gehen wir von der Annahme aus, dass die die Minimalitätsbedingung und Startbedingung auch erfüllt sind. Dies können wir o.B.d.A. annehmen, da wir in den vorherigen Lemmata gezeigt haben, dass diese unabhängig von der Sprachbedingung gelten.

Lemma 14 *Seien $T \subseteq STNT' \subset STNT$ und $w \in STNT \backslash STNT'$ mit $w = r_1 l_1 \# l_2 v_2$. Weiterhin sei $(r_1, l_1, l_2 v_2) \in SkipTo(L_1)$, $\# l_2 v_2 \in NT$, $\# l_2 v_2 \in NT'$ und $(r_1, l_1, l_2 v_2)$ erfülle o.B.d.A. die Minimalitätsbedingung und Startbedingung der Definition von $SkipTo(L_1')$. Dann erfüllt $(r_1, l_1, l_2 v_2)$ auch die Sprachbedingung der Definition von $SkipTo(L_1')$, d.h. es gilt $l_1 \in L_1'$.*

Beweis. Für den Beweis nehmen wir das Gegenteil an, d.h. sei $l_1 \notin L_1'$. Wir betrachten das Wort $w' = l_1 \# l_2$. Es gilt $w' \in T$. Aus $T \subseteq STNT'$ folgt $w' \in STNT'$. Da $l_1 \notin L_1'$ gilt, muss eine andere Zerlegung für w' existieren, die die Bedingungen von $STNT'$ erfüllt. Angenommen, $r_1' l_1' \# l_2$ sei diese Zerlegung. Dann gilt $(r_1', l_1', l_2) \in SkipTo(L_1')$ mit $r_1' l_1' = l_1$. Wir betrachten die Zerlegung $w = r_1 r_1' l_1' \# l_2 v_2$. Da $w \notin STNT'$ gilt, ist $(r_1 r_1', l_1', l_2 v_2) \notin SkipTo(L_1')$ und eine der Bedingungen der Definition von $SkipTo(L_1')$ darf nicht erfüllt sein. Die Randbedingung und die Minimalitätsbedingung sind erfüllt. Da $(r_1, l_1, l_2 v_2)$ die Startbedingung der Definition von $SkipTo(L_1')$ erfüllt, wissen wir außerdem, dass keine $r_1'' \in \Sigma^*, l_1'' \in L_1'$ existieren mit $r_1'' \sqsubset_p r_1 \wedge r_1'' l_1'' \sqsubseteq_p r_1 l_1 l_2 v_2$. Somit kann es nur einen Grund für $(r_1 r_1', l_1', l_2 v_2) \notin SkipTo(L_1')$ geben: $\exists r_1'' \in \Sigma^*, l_1'' \in L_1'$ mit $r_1'' \sqsubset_p r_1' \wedge r_1'' l_1'' \sqsubseteq_p r_1' l_1' l_2 v_2 \wedge r_1' l_1' l_2 \sqsubset_p r_1'' l_1''$. Angenommen, dies gilt. Aus der Menge der $r_1'' l_1''$, die diese Bedingung erfüllen, betrachten wir dabei das Paar mit dem kürzesten r_1'' und einem l_1'', so dass kein $l \in L_1'$ existiert mit $l \sqsubset_p l_1''$. Es gibt nun zwei Möglichkeiten:

Fall (1): Es existieren $r' \in \Sigma^*$, $v' \in \Sigma^+$ und $l' \in L_1'$ mit $r' \sqsubset_p r_1'' \wedge r' l' = r_1'' l_1'' v'$.

Angenommen, dies gilt. Aus der Menge der $r' l'$, die diese Bedingung erfüllen, betrachten wir das Paar mit dem kürzesten r' und einem l', so dass kein $l'' \in L_1'$ existiert mit $l'' \sqsubset_p l'$. Sei $w' = r' l' \# l_2 v_2$. Wir zeigen zuerst, dass $(r', l', l_2 v_2) \in SkipTo(L_1')$ gilt. Es sind $r' \in \Sigma^*$ und $l_2 v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r' und l', existieren kein $l'' \in L_1'$ mit $l'' \sqsubset_p l'$ [Die Minimalitätsbedingung ist erfüllt] und keine $r'' \in \Sigma^*$, $l'' \in L_1'$ mit $r'' \sqsubset_p r' \wedge r'' l'' \sqsubseteq_p r' l' l_2 v_2$ [Die Startbedingung ist erfüllt]. Da $(r', l', l_2 v_2)$ alle Bedingungen der Definition von $SkipTo(L_1')$ erfüllt, ist $(r', l', l_2 v_2) \in SkipTo(L_1')$. Da außerdem $\# l_2 v_2 \in NT'$ gilt, ist $w' \in STNT'$. Angenommen, $w' \in STNT$ gilt. Dann existiert eine Zerlegung $w' = r'' l'' \# l_2 v_2$ mit $(r'', l'', l_2 v_2) \in SkipTo(L_1)$. Aus $r_1' l_1' l_2 = l_1 l_2 \sqsubset_p r_1'' l_1''$ folgt $l_1 \sqsubset_p r_1'' l_1''$. Da außerdem $r_1'' l_1'' \sqsubseteq_p r' l'$ ist, gilt $l_1 \sqsubset_p r' l' = r'' l''$. Ist $r'' = \varepsilon$, dann gilt $l_1 \sqsubset_p l''$ und $(r'', l'', l_2 v_2)$ erfüllt nicht die Minimalitätsbedingung von $SkipTo(L_1)$. Ist $r'' \neq \varepsilon$, dann gilt $r_\varepsilon \sqsubset_p r'' \wedge r_\varepsilon l_1 \sqsubseteq_p r'' l'' l_2 v_2$ mit $r_\varepsilon = \varepsilon$ und $(r'', l'', l_2 v_2)$ erfüllt nicht die Startbedingung von $SkipTo(L_1)$. Somit gilt $(r'', l'', l_2 v_2) \notin SkipTo(L_1)$, was ein Wider-

spruch zu unserer Annahme ist, dass $(r'', l'', l_2v_2) \in \mathit{SkipTo}(L_1)$ ist. Also ist $w' \notin STNT$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $STNT' \subset STNT$ gilt und deshalb kann Fall (1) nicht gelten.

Fall (2): Es existieren keine $r' \in \Sigma^*$, $v' \in \Sigma^+$ und $l' \in L_1'$ mit $r' \sqsubset_p r_1'' \wedge r'l' = r_1''l_1''v'$.

Angenommen, dies gilt. Sei $w' = r_1''l_1''\#l_2v_2$. Wir zeigen zuerst, dass $(r_1'', l_1'', l_2v_2) \in \mathit{SkipTo}(L_1')$ gilt. Es sind $r_1'' \in \Sigma^*$ und $l_2v_2 \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l_1'' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Wegen unserer Wahl von r_1'' und l_1'', existieren kein $l' \in L_1'$ mit $l' \sqsubset_p l_1''$ [Die Minimalitätsbedingung ist erfüllt] und keine $r' \in \Sigma^*$, $l' \in L_1'$ mit $r' \sqsubset_p r_1'' \wedge r'l' \sqsubseteq_p r_1''l_1''l_2v_2$ [Die Startbedingung ist erfüllt]. Da (r_1'', l_1'', l_2v_2) alle Bedingungen der Definition von $\mathit{SkipTo}(L_1')$ erfüllt, ist $(r_1'', l_1'', l_2v_2) \in \mathit{SkipTo}(L_1')$. Da außerdem $\#l_2v_2 \in NT'$ gilt, ist $w' \in STNT'$. Angenommen, $w' \in STNT$ gilt. Dann existiert eine Zerlegung $w' = r'l'\#l_2v_2$ mit $(r', l', l_2v_2) \in \mathit{SkipTo}(L_1)$. Aus $r_1'l_1'l_2 = l_1l_2 \sqsubset_p r_1''l_1''$ folgt $l_1 \sqsubset_p r_1''l_1'' = r'l'$. Ist $r' = \varepsilon$, dann gilt $l_1 \sqsubset_p l'$ und (r', l', l_2v_2) erfüllt nicht die Minimalitätsbedingung von $\mathit{SkipTo}(L_1)$. Ist $r' \neq \varepsilon$, dann gilt $r_\varepsilon \sqsubset_p r' \wedge r_\varepsilon l_1 \sqsubseteq_p r'l'l_2v_2$ mit $r_\varepsilon = \varepsilon$ und (r', l', l_2v_2) erfüllt nicht die Startbedingung von $\mathit{SkipTo}(L_1)$. Somit gilt $(r', l', l_2v_2) \notin \mathit{SkipTo}(L_1)$, was ein Widerspruch zu unserer Annahme ist, dass $(r', l', l_2v_2) \in \mathit{SkipTo}(L_1)$ ist. Also ist $w' \notin STNT$. Dies ist wiederum ein Widerspruch zu unserer Annahme, dass $STNT' \subset STNT$ gilt und deshalb kann Fall (2) nicht auftreten.

Da wir sowohl Fall (1) als auch Fall (2) widerlegt haben, existieren keine $r_1'' \in \Sigma^*$ und $l_1'' \in L_1'$ mit $r_1'' \sqsubset_p r_1' \wedge r_1''l_1'' \sqsubseteq_p r_1'l_1'l_2v_2 \wedge r_1'l_1'l_2 \sqsubset_p r_1''l_1''$. Dies ist ein Widerspruch zu unserer Annahme, dass $(r_1r_1', l_1', l_2v_2) \notin \mathit{SkipTo}(L_1')$ gilt. Also gilt $(r_1r_1', l_1', l_2v_2) \in \mathit{SkipTo}(L_1')$ und $w \in STNT'$. Das ist aber wiederum ein Widerspruch zur Annahme, dass $w \in STNT \backslash STNT'$ gilt. Wir haben somit gezeigt, dass (r_1, l_1, l_2v_2) die Sprachbedingung der Definition von $\mathit{SkipTo}(L_1')$ erfüllt. ∎

Lernbarkeit von Regeln mit zwei *SkipTo*-Operationen und einer *Next*-Operation

Sei $\mathcal{ST}^{(2)}\mathcal{NT}(\mathcal{L}_f)$ die Menge aller Regeln mit zwei aufeinanderfolgenden *SkipTo*-Operationen und einer *SkipUntil*-Operationen, d.h. $\mathcal{ST}^{(2)}\mathcal{NT}(\mathcal{L}) = \{ST^{(2)}NT(L_1, L_2, L_3) \mid L_1, L_2, L_3 \in \mathcal{L}\}$. Wir zeigen nun, dass diese Menge für endliche Sprachen nicht lernbar ist.

Theorem 11 *Es gilt* $\mathcal{ST}^{(2)}\mathcal{NT}(\mathcal{L}_f) \notin LimTxt$.

Beweis. Seien $\Sigma = \{a, b, c, d, e, f\}$, $L_1 = \{cd\}$, $L_2 = \{ba, bb, bc, bd, be, bf\}$ und $L_3 = \{ef\}$. Wir werden beweisen, dass keine endliche Telltale-Menge von $ST^{(2)}NT(L_1, L_2, L_3)$ existiert, d.h. für jede endliche Menge $T \subseteq ST^{(2)}NT(L_1, L_2, L_3)$ gibt es eine ST_2NT', so dass $T \subseteq ST_2NT'$ und $ST_2NT' \subset ST^{(2)}NT(L_1, L_2, L_3)$ gilt. Damit gilt aber nach Theorem 1 (?) $\mathcal{ST}^{(2)}\mathcal{NT}(\mathcal{L}_f) \notin LimTxt$. Zur besseren Übersichtlichkeit treffen wir für die restliche Beweisführung folgende Vereinbarung: $ST_2NT = ST^{(2)}NT(L_1, L_2, L_3)$, $STNT = STNT(L_2, L_3)$ und $NT = NT(L_3)$. Für den Beweis nehmen wir das Gegenteil an: Sei $T \subseteq ST_2NT$ eine Telltale-Menge von ST_2NT, d.h. es existiert keine Menge ST_2NT', so dass $T \subseteq ST_2NT'$ und $ST_2NT' \subset ST_2NT$ gilt. Angenommen, es gelte $m = max\{|w| \mid w \in T\}$ und es sei $ST_2NT' = ST^{(2)}NT(L_1', L_2', L_3')$ mit $L_1' = \{cdub \mid u \in \Sigma^* \wedge |u| \leq m \wedge b \text{ ist kein Teilwort von } u\}$, $L_2' = \{a, b, c, d, e, f\}$ und $L_3' = \{ef\}$. Seien außerdem $STNT' = STNT(L_2', L_3')$ und $NT' = NT(L_3')$. Wir zeigen nun, dass Folgendes gilt:

Fall (1) $T \subseteq ST_2NT'$

Fall (2) $ST_2NT' \subset ST_2NT$

Fall (1): Sei $w \in T$. Aus $T \subseteq ST_2NT$ folgt $w \in ST_2NT$. Damit existieren $r_1, r_2 \in \Sigma^*$, $v_3 \in \Sigma^*$ und $l_1 \in L_1, l_2 \in L_2, l_3 \in L_3$ mit $w = r_1l_1r_2l_2\#l_3v_3$, $r_2l_2\#l_3v_3 \in STNT$, $\#l_3v_3 \in NT$, $(r_1, l_1, r_2l_2l_3v_3) \in SkipTo(L_1)$, $(r_2, l_2, l_3v_3) \in SkipTo(L_2)$ und $(l_3, v_3) \in Next(L_3)$. Dann gilt $l_1r_2l_2 \in \{cdu_1bu_2 \mid u_1 \in \Sigma^* \wedge b \text{ ist keine Teilwort von } u_1 \wedge u_2 \in \Sigma\}$ und es existieren $l_1' \in \{cdub \mid u \in \Sigma^* \wedge b \text{ ist keine Teilwort von } u\}$, $r_2' = \varepsilon$ sowie $l_2' \in \Sigma$ mit $l_1'r_2'l_2' = r_1l_1r_2l_2$.

Wir zeigen nun, dass $(r_1, l_1', r_2'l_2'l_3v_3) \in SkipTo(L_1')$ gilt. Es sind $r_1 \in \Sigma^*$ und $r_2'l_2'l_3v_3 \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Wegen $w \leq m$ gilt $l_1' \leq m$. Weil außerdem $l_1' \in \{cdub \mid u \in \Sigma^* \wedge b \text{ ist keine Teilwort von } u\}$ gilt, ist $l_1' \in L_1'$ [Die Sprachbedingung ist erfüllt]. Da kein Element von L_1' ein Präfix besitzt, das auch Element von L_1' ist, kann auch kein $l' \in L_1'$ existieren mit $l' \sqsubset_p l_1'$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $(r_1, l_1, r_2l_2l_3v_3) \in SkipTo(L_1)$ existieren keine $r' \in \Sigma^*$, $l' \in L_1$ mit $r' \sqsubset_p r_1 \wedge r'l' \sqsubseteq_p r_1l_1r_2l_2l_3v_3$. Damit ist cd kein Teilwort von r_1 und es existieren auch keine $r' \in \Sigma^*$, $l' \in L_1'$ mit $r' \sqsubset_p r_1 \wedge r'l' \sqsubseteq_p r_1l_1'r_2'l_2'l_3v_3$ [Die Startbedingung ist erfüllt]. Da $(r_1, l_1', r_2'l_2'l_3v_3)$ alle Bedingungen der Definition von $SkipTo(L_1')$ erfüllt, gilt $(r_1, l_1', r_2'l_2'l_3v_3) \in SkipTo(L_1')$.

Wir zeigen weiter, dass $(r_2', l_2', l_3v_3) \in SkipTo(L_2')$ gilt. Es sind $l_3v_3 \in \Sigma^+$ und $r_2' = \varepsilon$ [Die

Randbedingungen sind erfüllt]. Wegen $l'_2 \in \Sigma$ ist $l'_2 \in L'_2$ [Die Sprachbedingung ist erfüllt]. Da $l'_2 \in \Sigma$ gilt, existiert kein Teilwort von l'_2, das auch Element von L'_2 ist. Somit gilt $\neg\exists l' \in L'_2$ mit $l' \sqsubset_p l'_2$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $r'_2 = \varepsilon$ existieren außerdem keine $r' \in \Sigma^*$, $l' \in L'_2$ mit $r' \sqsubset_p r'_2 \wedge r'l' \sqsubseteq_p r'_2 l'_2 l_3 v_3$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r'_2, l'_2, l_3 v_3) \in \mathit{SkipTo}(L'_2)$.

Wegen $L_3 = L'_3$ gilt $(l_3, v_3) \in \mathit{Next}(L'_3)$. Somit sind $\#l_3 v_3 \in NT'$ und $r'_2 l'_2 \# l_3 v_3 \in STNT'$. Zusätzlich gilt $(r_1, l'_1, r'_2 l'_2 l_3 v_3) \in \mathit{SkipTo}(L'_1)$. Daraus folgt $w = r_1 l_1 r_2 l_2 \# l_3 v_3 = r_1 l'_1 r'_2 l'_2 \# l_3 v_3 \in ST_2NT'$ und wir haben bewiesen, dass $T \subseteq ST_2NT'$ gilt.

Fall (2): Sei $w' \in ST_2NT'$. Es existieren also $r'_1, r'_2 \in \Sigma^*$, $v'_3 \in \Sigma^*$ und $l'_1 \in L'_1, l'_2 \in L'_2, l'_3 \in L'_3$ mit $w' = r'_1 l'_1 r'_2 l'_2 \# l'_3 v'_3$, $r'_2 l'_2 \# l'_3 v'_3 \in STNT'$, $\# l'_3 v'_3 \in NT'$, $(r'_1, l'_1, r'_2 l'_2 l'_3 v'_3) \in \mathit{SkipTo}(L'_1)$, $(r'_2, l'_2, l'_3 v'_3) \in \mathit{SkipTo}(L'_2)$ und $(l'_3, v'_3) \in \mathit{Next}(L'_3)$. Dann gilt $l'_1 r'_2 l'_2 \in \{cdu_1 bu_2 \mid u_1 \in \Sigma^* \wedge b$ ist keine Teilwort von $u_1 \wedge u_2 \in \Sigma\}$ und es existieren $l''_1 = cd$, $r''_2 \in \Sigma^*$ und $l''_2 \in \{bu \mid u \in \Sigma\}$ mit $l''_1 r''_2 l''_2 = l'_1 r'_2 l'_2$, so dass b kein Teilwort von r''_2 ist.

Wir beweisen, dass $(r'_1, l''_1, r''_2 l''_2 l'_3 v'_3) \in \mathit{SkipTo}(L_1)$ gilt. Es sind $r'_1 \in \Sigma^*$ und $r''_2 l''_2 l'_3 v'_3 \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Wegen $l''_1 = cd$ gilt $l''_1 \in L_1$ [Die Sprachbedingung ist erfüllt]. Da L_1 nur ein Element besitzt, kann auch kein $l' \in L_1$ existieren mit $l' \sqsubset_p l''_1$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $(r'_1, l'_1, r'_2 l'_2 l'_3 v'_3) \in \mathit{SkipTo}(L'_1)$ existieren keine $r' \in \Sigma^*$ und $l' \in L'_1$ mit $r' \sqsubset_p r'_1$ und $r'l' \sqsubseteq_p r'_1 l'_1 r'_2 l'_2 l'_3 v'_3$. Daraus folgt, dass cd kein Teilwort von r'_1 ist. Damit existieren auch keine $r' \in \Sigma^*$, $l' \in L_1$ mit $r' \sqsubset_p r'_1 \wedge r'l' \sqsubseteq_p r'_1 l''_1 r''_2 l''_2 l'_3 v'_3$ [Die Startbedingung ist erfüllt]. Da $(r'_1, l''_1, r''_2 l''_2 l'_3 v'_3)$ alle Bedingungen der Definition von $\mathit{SkipTo}(L_1)$ erfüllt, gilt $(r'_1, l''_1, r''_2 l''_2 l_3 v_3) \in \mathit{SkipTo}(L'_1)$.

Wir zeigen weiter, dass $(r''_2, l''_2, l'_3 v'_3) \in \mathit{SkipTo}(L'_2)$ gilt. Es sind $l'_3 v'_3 \in \Sigma^+$ und $r''_2 \in \Sigma^*$ [Die Randbedingungen sind erfüllt]. Da $l''_2 \in \{bu \mid u \in \Sigma\}$ gilt, ist $l''_2 \in L_2$ [Die Sprachbedingung ist erfüllt]. Kein Element aus L_2 besitzt ein Teilwort, das auch Element von L_2 ist. Somit gilt $\neg\exists l' \in L_2$ mit $l' \sqsubset_p l''_2$ [Die Minimalitätsbedingung ist erfüllt]. b ist kein Teilwort von r''_2. Deshalb existieren keine $r' \in \Sigma^*$, $l' \in L_2$ mit $r' \sqsubset_p r''_2 \wedge r'l' \sqsubseteq_p r''_2 l''_2 l'_3 v'_3$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(r''_2, l''_2, l'_3 v'_3) \in \mathit{SkipTo}(L_2)$.

Wegen $L_3 = L'_3$ gilt $(l'_3, v'_3) \in \mathit{Next}(L_3)$. Somit sind $\#l'_3 v'_3 \in NT$ und $r''_2 l''_2 \# l'_3 v'_3 \in STNT$. Zusätzlich gilt $(r'_1, l''_1, r''_2 l''_2 l'_3 v'_3) \in \mathit{SkipTo}(L_1)$. Daraus folgt $w = r'_1 l'_1 r'_2 l'_2 \# l'_3 v'_3 = r'_1 l''_1 r''_2 l''_2 \# l'_3 v'_3 \in ST_2NT$ und wir haben bewiesen, dass $ST_2NT' \subseteq ST_2NT$ gilt.

Sei $w'' \in \{cdu_1 ba \# ef \mid u_1 \in \Sigma^* \wedge |u_1| > m \wedge cd$ und b sind keine Teilworte von $u_1\}$. Offensichtlich ist $w'' \in ST_2NT$. Da $|u_1| > m$ gilt und cd kein Teilwort von u_1 ist, existiert kein Teilwort von w'', das Element von L'_1 ist. Damit gibt es auch keine Zerlegung, die Element von $\mathit{SkipTo}(L'_1)$ ist und es gilt $w'' \notin ST_2NT'$. Somit ist $ST_2NT \backslash ST_2NT' \neq \emptyset$. Aus $ST_2NT' \subseteq ST_2NT$ und $ST_2NT \backslash ST_2NT' \neq \emptyset$ folgt $ST_2NT' \subset ST_2NT$.

Wir haben gezeigt, dass für T ein ST_2NT' existiert, so dass $T \subseteq ST_2NT'$ und $ST_2NT' \subset ST_2NT$ gilt. Dies ist aber ein Widerspruch zu unserer Annahmen, dass T eine endliche Telltale-Menge ist. Da T beliebig ist, existiert keine endliche Telltale-Menge von $ST^{(2)}NT(L_1, L_2, L_3)$. Damit gilt $\mathcal{ST}^{(2)}\mathcal{NT}(\mathcal{L}_f) \notin LimTxt$.

■

Lernbarkeit von Regeln mit mehr als zwei *SkipTo*-Operationen und einer *Next*-Operationen

Sei $\mathcal{ST}^{(n)}\mathcal{NT}(\mathcal{L}_f)$ die Menge aller Regeln mit n aufeinanderfolgenden *SkipTo*-Operationen und einer *Next*-Operationen, d.h. $\mathcal{ST}^{(n)}\mathcal{NT}(\mathcal{L}) = \{ST^{(n)}NT(L_1, ..., L_{n+1}) \mid L_1, ..., L_{n+1} \in \mathcal{L}\}$. Wir zeigen nun, dass diese Menge für endliche Sprachen und $n \geq 3$ nicht lernbar ist.

Theorem 12 *Sei $n \geq 3$. Dann gilt $\mathcal{ST}^{(n)}\mathcal{NT}(\mathcal{L}_f) \notin LimTxt$.*

Beweis. Seien $\Sigma = \{a, b, c, d, e, f\}$, $L_1 = \{ab\}$, $L_2 = \{cd\}$, $L_3 = \{af, bf, cf, df, ef, ff\}$ und $L_j = \{ef\}$ für alle $4 \leq j \leq n+1$. Wir werden beweisen, dass keine endliche Telltale-Menge von $ST^{(n)}NT(L_1, ..., L_{n+1})$ existiert, d.h. für jede endliche Menge $T \subseteq ST^{(n)}NT(L_1, ..., L_{n+1})$ existiert ein ST_nNT', so dass $T \subseteq ST_nNT'$ und $ST_nNT' \subset ST^{(n)}NT(L_1, ..., L_{n+1})$ gilt. Damit gilt aber nach Theorem 1 $\mathcal{ST}^{(n)}\mathcal{NT}(\mathcal{L}_f) \notin LimTxt$. Zur besseren Übersichtlichkeit treffen wir für den restlichen Beweis folgende Vereinbarungen: $ST_nNT = ST^{(n)}NT(L_1, ..., L_{n+1})$, $ST_{n-(i-1)}NT = ST^{(n-(i-1))}NT(L_i, ..., L_{n+1})$ für ein $i \in \{1, ..., n\}$ und $NT = NT(L_{n+1})$.
Für den Beweis nehmen wir das Gegenteil an: Sei $T \subseteq ST_nNT$ eine Telltale-Menge von ST_nNT, d.h. es existiert keine Menge ST_nNT', so dass $T \subseteq ST_nNT'$ und $ST_nNT' \subset ST_nNT$ gilt.
Angenommen, es gelte $m = max\{|w| \mid w \in T\}$ und es sei $ST_nNT' = ST^{(n)}NT(L'_1, ..., L'_{n+1})$ mit $L'_1 = \{abucd \mid u \in \Sigma^* \wedge |u| \leq m \wedge cd \text{ ist kein Teilwort von } u\}$, $L'_2 = \{a, b, c, d, e, f\}$, $L'_3 = \{f\}$ sowie $L'_j = \{ef\}$ für alle $4 \leq j \leq n+1$. Sei außerdem $ST_{n-(i-1)}NT' = ST^{(n-(i-1))}NT(L'_i, ..., L'_{n+1})$ für ein $i \in \{1, ..., n\}$ und $NT' = NT(L'_{n+1})$. Wir zeigen nun, dass Folgendes gilt:

Fall (1) $T \subseteq ST_nNT'$

Fall (2) $ST_nNT' \subset ST_nNT$

Fall (1): Sei $w \in T$. Aus $T \subseteq ST_nNT$ folgt $w \in ST_nNT$. Damit existieren $r_1, ..., r_n, v_{n+1} \in \Sigma^*$ und $l_1 \in L_1, ..., l_{n+1} \in L_{n+1}$ mit $w = r_1l_1...r_nl_n\#l_{n+1}v_{n+1}$, $\#l_{n+1}v_{n+1} \in NT$ und $(l_{n+1}, v_{n+1}) \in Next(L_{n+1})$ sowie $r_il_i...r_nl_n\#l_{n+1}v_{n+1} \in ST_{n-(i-1)}NT$ und $(r_i, l_i, r_{i+1}...l_{n+1}v_{n+1}) \in SkipTo(L_i)$ für alle $1 \leq i \leq n$. Zuerst zeigen wir, dass $(r_1, l_1r_2l_2, r_3l_3...r_nl_nl_{n+1}v_{n+1}) \in SkipTo(L'_1)$ gilt. Es gilt $r_1 \in \Sigma^*$ und $r_3...l_{n+1}v_{n+1} \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Aus der Definition von L_1 und L_2 folgt, dass $l_1 = ab$ und $l_2 = cd$ gilt. Weiterhin gilt $r_2 < m$ und wegen $(r_2, l_2, r_3...l_nl_{n+1}v_{n+1}) \in SkipTo(L_2)$ ist cd kein Teilwort von r_2. Damit gilt $l_1r_2l_2 = abr_2cd \in L'_1$ [Die Sprachbedingung ist erfüllt]. Kein Element von L'_1 besitzt ein Präfix, das auch Element von L'_1 ist. Dann gilt aber auch $\neg\exists l'_1 \in L'_1$ mit $l'_1 \sqsubset_p l_1r_2l_2$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $(r_1, l_1, r_2l_2...r_nl_nl_{n+1}v_{n+1}) \in SkipTo(L_1)$ ist ab kein Teilwort von r_1 und damit existieren keine $r'_1 \in \Sigma^*$, $l'_1 \in L'_1$ mit $r'_1 \sqsubset_p r_1 \wedge r'_1l'_1 \sqsubseteq_p r_1l_1...r_nl_nl_{n+1}v_{n+1}$ [Die Startbedingung ist erfüllt]. Da $(r_1, l_1r_2l_2, r_3...l_{n+1}v_{n+1})$ alle Bedingungen der Definition von $SkipTo(L'_1)$ erfüllt gilt $(r_1, l_1r_2l_2, r_3...l_{n+1}v_{n+1}) \in SkipTo(L'_1)$.

Wegen $(r_3, l_3, r_4l_4...r_nl_nl_{n+1}v_{n+1}) \in SkipTo(L_3)$ ist $r_3l_3 \in \{u_1u_2f \mid u_1 \in \Sigma \wedge u_2 \in \Sigma^* \wedge f$

ist kein Teilwort von $u_2\}$. Dadurch existieren $l'_2 \in L'_2$, $r'_3 \in \Sigma^*$ und $l'_3 = f \in L'_3$, so dass f kein Teilwort von r'_3 ist und $l'_2 r'_3 l'_3 = r_3 l_3$ gilt. Wir zeigen nun, dass $(r'_2, l'_2, r'_3 l'_3 r_4 l_4 ... r_n l_n l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_2) mit $r'_2 = \varepsilon$ gilt. Es sind $r'_3 l'_3 r_4 ... l_{n+1} v_{n+1} \in \Sigma^+$ und $r'_2 = \varepsilon$ [Die Randbedingungen sind erfüllt] sowie $l'_2 \in L'_2$ [Die Sprachbedingung ist erfüllt]. Da $l'_2 \in \Sigma$ gilt, existiert kein Teilwort von l'_2, das Element von L'_2 ist. Somit gilt $\neg\exists l''_2 \in L'_2$ mit $l''_2 \sqsubset_p l'_2$ [Die Minimalitätsbedingung ist erfüllt]. Wegen $r'_2 = \varepsilon$ existieren außerdem keine $r''_2 \in \Sigma^*$, $l''_2 \in L'_2$ mit $r''_2 \sqsubset_p r'_2 \wedge r''_2 l''_2 \sqsubseteq_p r'_2 l'_2 r'_3 l'_3 r_4 l_4 ... r_n l_n l_{n+1} v_{n+1}$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, gilt $(\varepsilon, l'_2, r'_3 l'_3 r_4 l_4 ... r_n l_n l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_2).

Wir zeigen weiter, dass $(r'_3, l'_3, r_4 l_4 ... r_n l_n l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_3) gilt. Es sind $r'_3 \in \Sigma^*$ und $r_4 ... l_{n+1} v_{n+1} \in \Sigma^+$ [Die Randbedingungen sind erfüllt]. Außerdem gilt $l'_3 = f \in L'_3$ [Die Sprachbedingung ist erfüllt]. Damit existiert aber auch kein $l''_3 \in L'_3$ mit $l''_3 \sqsubset_p l'_3$ [Die Minimalitätsbedingung ist erfüllt]. Da f kein Teilwort von r'_3 ist, existieren auch keine $r''_3 \in \Sigma^*$, $l''_3 \in L'_3$ mit $r''_3 \sqsubset_p r'_3 \wedge r''_3 l''_3 \sqsubseteq_p r'_3 l'_3 r_4 l_4 ... r_n l_n l_{n+1} v_{n+1}$ [Die Startbedingung ist erfüllt]. $(r'_3, l'_3, r_4 l_4 ... r_n l_n l_{n+1} v_{n+1})$ erfüllt alle Bedingungen der Definition von *SkipTo*(L'_3) und es gilt $(r'_3, l'_3, r_4 l_4 ... r_n l_n l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_3).

Für alle $4 \leq j \leq n$ gilt $L_j = L'_j$. Daraus folgt $(r_j, l_j, r_{j+1} l_{j+1} ... r_n l_n l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_j). Da auch $L_{n+1} = L'_{n+1}$ gilt, ist $(l_{n+1}, v_{n+1}) \in$ *Next*(L'_{n+1}). Somit gilt $r'_2 l'_2 r'_3 l'_3 r_4 ... r_n l_n \# l_{n+1} v_{n+1} \in ST_{n-1}NT'$, $r'_3 l'_3 r_4 ... r_n l_n \# l_{n+1} v_{n+1} \in ST_{n-2}NT'$, $r_j l_j ... r_n l_n \# l_{n+1} v_{n+1} \in ST_{n-(j-1)}NT'$ für alle $4 \leq j \leq n$ und $\# l_{n+1} v_{n+1} \in NT'$. Da außerdem $(r_1, l_1 r_2 l_2, r_3 ... l_{n+1} v_{n+1}) \in$ *SkipTo*(L'_1) gilt, ist $w = r_1 l_1 ... r_n l_n \# l_{n+1} v_{n+1} = r_1 l_1 r_2 l_2 l'_2 r'_3 l'_3 r_4 l_4 ... r_n l_n \# l_{n+1} v_{n+1} \in ST_n NT'$ und wir haben bewiesen, dass $T \subseteq ST_n NT'$ gilt.

Fall (2): Sei $w' \in ST_n NT'$. Es existieren also $r'_1, ..., r'_n, v'_{n+1} \in \Sigma^*$ und $l'_1 \in L'_1, ..., l'_{n+1} \in L'_{n+1}$ mit $w' = r'_1 l'_1 ... r'_n l'_n \# l'_{n+1} v'_{n+1}$, $r'_i ... l'_n \# l'_{n+1} v'_{n+1} \in ST_{n-(i-1)}NT'$ und $(r'_i, l'_i, r'_{i+1} ... l'_n l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L'_i) für alle $1 \leq i \leq n$, $\# l_{n+1} v_{n+1} \in NT'$ und $(l_{n+1}, v_{n+1}) \in$ *Next*(L'_{n+1}). Wegen $(r'_1, l'_1, r'_2 l'_2 ... r'_n l'_n l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L'_1) gilt $r'_1 l'_1 \in \{u_1 a b u_2 c d \mid u_1 \in \Sigma^* \wedge ab$ ist keine Teilwort von $u_1 \wedge u_2 \in \Sigma^* \wedge cd$ ist kein Teilwort von $u_2\}$. Damit existieren $r''_1, r''_2 \in \Sigma^*, l''_1 = ab$ und $l''_2 = cd$, so dass $r''_1 l''_1 r''_2 l''_2 = r'_1 l'_1$ gilt, ab kein Teilwort von r''_1 und cd kein Teilwort von r''_2 ist. Zuerst zeigen wir, dass $(r''_1, l''_1, r''_2 l''_2 r'_2 l'_2 ... r'_n l'_n l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L_1) gilt. Es sind $r''_1 \in \Sigma^*$ und $r''_2 l''_2 r'_2 ... l'_{n+1} v'_{n+1} \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l''_1 = ab \in L_1$ [Die Sprachbedingung ist erfüllt]. Es existiert nur ein Element in L_1. Damit gilt $\neg\exists l' \in L_1$ mit $l' \sqsubset_p l''_1$ [Die Minimalitätsbedingung ist erfüllt]. Da ab kein Teilwort von r''_1 ist, existieren keine $r' \in \Sigma^*$, $l' \in L_1$ mit $r' \sqsubset_p r''_1 \wedge r' l' \sqsubseteq_p r''_1 l''_1 r''_2 l''_2 r'_2 l'_2 ... r'_n l'_n l'_{n+1} v'_{n+1}$ [Die Startbedingung ist erfüllt]. Alle Bedingungen sind erfüllt und somit gilt $(r''_1, l''_1, r''_2 l''_2 r'_2 l'_2 ... r'_n l'_n l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L_1).

Wir beweisen weiter, dass $(r''_2, l''_2, r'_2 l'_2 ... r'_n l'_n l'_{n+1} v'_{n+1}) \in$ *SkipTo*(L_2) gilt. Es sind $r''_2 \in \Sigma^*$ und $r'_2 ... l'_{n+1} v'_{n+1} \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l''_2 = cd \in L_2$ [Die Sprachbedingung ist erfüllt]. Es existiert nur ein Element in L_2. Damit gilt $\neg\exists l' \in L_2$ mit $l' \sqsubset_p l''_2$ [Die Minimalitätsbedingung ist erfüllt]. Da cd kein Teilwort von r''_2 ist, existieren keine $r' \in \Sigma^*$,

$l' \in L_2$ mit $r' \sqsubset_p r''_2 \wedge r'l' \sqsubseteq_p r''_2 l''_2 r'_2 l'_2 ... r'_n l'_n l'_{n+1} v'_{n+1}$ [Die Startbedingung ist erfüllt]. Alle Bedingungen sind erfüllt und somit gilt $(r''_2, l''_2, r'_2 l'_2 ... r'_n l'_n l'_{n+1} v'_{n+1}) \in SkipTo(L_2)$.

Aus $(r'_2, l'_2, r'_3 l'_3 ... r'_n l'_n l'_{n+1} v'_{n+1}) \in SkipTo(L'_2)$ und $(r'_3, l'_3, r'_4 l'_4 ... r'_n l'_n l'_{n+1} v'_{n+1}) \in SkipTo(L'_3)$ folgt $r'_2 = \varepsilon$, $l'_2 \in \Sigma$, f ist kein Teilwort von r'_3 sowie $l'_3 = f$ und damit $r'_2 l'_2 r'_3 l'_3 \in \{u_1 u_2 f \mid u_1 \in \Sigma \wedge u_2 \in \Sigma^* \wedge f$ ist kein Teilwort von $u_2\}$. Somit existieren $r''_3 \in \Sigma^*$ und $l''_3 \in L_3$, so dass $r''_3 l''_3 = r'_2 l'_2 r'_3 l'_3$ gilt. Wir zeigen nun, dass $(r''_3, l''_3, r'_4 l'_4 ... r'_n l'_n l'_{n+1} v'_{n+1}) \in SkipTo(L_3)$ gilt. Es sind $r''_3 \in \Sigma^*$ und $r'_4 ... l'_{n+1} v'_{n+1} \in \Sigma^+$ [Die Randbedingungen sind erfüllt] sowie $l''_3 \in L_3$ [Die Sprachbedingung ist erfüllt]. In L_3 existiert kein Element, das ein Teilwort besitzt, das auch Element von L_3 ist. Damit gilt $\neg\exists l' \in L_3$ mit $l' \sqsubset_p l''_3$ [Die Minimalitätsbedingung ist erfüllt]. Weil $r''_3 l''_3 \in \{u_1 u_2 f \mid u_1 \in \Sigma \wedge u_2 \in \Sigma^* \wedge f$ ist kein Teilwort von $u_2\}$ gilt, existieren keine $r' \in \Sigma^*$, $l' \in L_3$ mit $r' \sqsubset_p r''_3 \wedge r'l' \sqsubseteq_p r''_3 l''_3 r'_3 l'_3 ... r'_n l'_n l'_{n+1} v'_{n+1}$ [Die Startbedingung ist erfüllt]. Da alle Bedingungen erfüllt sind, haben wir bewiesen, dass $(r''_3, l''_3, r'_4 l'_4 ... r'_n l'_n l'_{n+1} v'_{n+1}) \in SkipTo(L_3)$ gilt.

Für alle $4 \leq j \leq n$ gilt $L_j = L'_j$. Daraus folgt $(r'_j, l'_j, r'_{j+1} l'_{j+1} ... r'_n l'_n l'_{n+1} v'_{n+1}) \in SkipTo(L_j)$. Da außerdem $L_{n+1} = L'_{n+1}$ gilt, ist $(l'_{n+1}, v'_{n+1}) \in Next(L_{n+1})$. Somit gilt $r''_2 l''_2 r''_3 l''_3 r'_4 ... l'_n \# l'_{n+1} v'_{n+1} \in ST_{n-1}NT$, $r''_3 l''_3 r'_4 ... r'_n l'_n \# l'_{n+1} v'_{n+1} \in ST_{n-2}NT$, $r'_j l'_j ... r'_n l'_n \# l'_{n+1} v'_{n+1} \in ST_{n-(j-1)}NT$ für alle $4 \leq j \leq n$ und $\# l'_{n+1} v'_{n+1} \in NT$. Da außerdem $(r''_1, l''_1, r''_2 l''_2 r''_3 l''_3 r'_4 ... l'_{n+1} v'_{n+1}) \in SkipTo(L_1)$ gilt, ist $w = r'_1 l'_1 ... r'_n l'_n \# l'_{n+1} v'_{n+1} = r''_1 l''_1 r''_2 l''_2 r''_3 l''_3 r'_4 l'_4 ... r'_n l'_n \# l'_{n+1} v'_{n+1} \in ST_nNT$ und wir haben bewiesen, dass $ST_nNT' \subseteq ST_nNT$ gilt.

Sei $w'' \in \{abu_1 cdaf u_2^{n-3} \# ef \mid u_1 \in \Sigma^* \wedge |u_1| > m \wedge cd$ ist kein Teilwort von $u_1 \wedge u_2 = ef\}$. Offensichtlich ist $w'' \in ST_nNT$. Da $|u_1| > m$ gilt, existiert kein Teilwort von w'', das Element von L'_1 ist. Damit gibt es auch keine Zerlegung, die Element von $SkipTo(L'_1)$ ist und es gilt $w'' \notin ST_nNT'$. Somit ist $ST_nNT \backslash ST_nNT' \neq \emptyset$. Aus $ST_nNT' \subseteq ST_nNT$ und $ST_nNT \backslash ST_nNT' \neq \emptyset$ folgt $ST_nNT' \subset ST_nNT$.

Wir haben gezeigt, dass für T ein ST_nNT' existiert, so dass $T \subseteq ST_nNT'$ und $ST_nNT' \subset ST_nNT$ gilt. Dies ist aber ein Widerspruch zu unserer Annahmen, dass T eine endliche Telltale-Menge ist. Da T beliebig ist, existiert keine endliche Telltale-Menge von $ST^{(n)}NT(L_1, ..., L_{n+1})$. Damit gilt $\mathcal{ST}^{(n)}\mathcal{NT}(\mathcal{L}_f) \notin LimTxt$.

■

7 Fazit

Diese Arbeit war zweigeteilt. Im ersten Teil haben wir ein theoretisches Modell in Termini formaler Sprachen eingeführt, das die Wrapperklasse von STALKER beschreibt. Ein Wrapper von STALKER basiert auf einer Menge von gelernten Regeln. Keine Regel ist dabei für das gesamte Dokument und alle zu extrahierenden Informationen anwendbar. STALKER lernt nur Single-Slot-Regeln. Dadurch sind die Regeln eines Wrappers unabhängig voneinander. Mit Hilfe eines EC-Baumes lassen sich jedoch Zusammenhänge zwischen den Attributen herstellen. Es existieren Extraktionsregeln für die Knoten eines EC-Baumes und Iterationsregeln für die Listenknoten. Der Unterschied zwischen Extraktions- und Iterationsregel liegt darin, dass Iterationsregeln auch mehrfach hintereinander angewendet werden können. Sowohl die Extraktions- als auch die Iterationsregel besteht aus zwei Teilregeln: eine Anfangsregel und eine Endregel. Jede Teilregel wird durch eine Anzahl von Operationen bestimmt und jede Operation besitzt ein Argument. In unserem Modell ist das Argument ein Element einer Argumentensprache. Jede Regel ist durch die Anzahl der Operationen, die Argumentensprachen und die letzte Operation bestimmt. Zusätzlich können Teilregeln Disjunktionen enthalten. Disjunktive Regeln werden dabei als geordnete Listen von Regeln angesehen. Durch die Art der Regeln hat das Wrapper-Induktionssystem STALKER im Vergleich zu den anderen Systemen einige Vorteile. Da STALKER nur Single-Slot-Regeln lernt und alle Regeln unabhängig voneinander sind, können auch dann Attribute extrahiert werden, wenn andere Attribute in einem Dokument fehlen. Genausowenig wird der Extraktionsprozess durch die Reihenfolge der Attribute beeinflusst. Mit Hilfe des EC-Baumes können trotzdem die einzelnen Attribute in Verbindung gebracht werden. STALKER ist fähig, verschachtelte Informationen zu extrahieren und dank den Operationen *SkipUntil* und *Next* können außerdem Informationen extrahiert werden, die nicht durch syntaktische Ausdrücke begrenzt sind.

Im zweiten Teil haben wir versucht zu untersuchen, ob ein Wrapper dieses Modells aus Text lernbar ist. Da wir es nicht mit einem klassischen Lernbarkeitsproblem zu tun hatten, haben wir das Konzept des Lernens aus markiertem Text von [GL01] übernommen und an unser Modell angepasst. Um den Wrapper zu lernen, ist es nötig, die Extraktions- und Iterationsregeln zu lernen. Jede dieser Regeln kann unabhängig von den anderen gelernt werden. Auch die Teilregeln der Extraktions- und Iterationsregeln sind unabhängig voneinander. Normalerweise können Teilregeln disjunktive Regeln sein. Für unsere Lernbarkeitsuntersuchungen haben wir die Regeln etwas vereinfacht. Wir sind davon ausgegangen, dass es keine Disjunktionen gibt und schon bekannt ist, wieviele Operationen eine Regel enthält und wie die letzte Operation aussieht. Wie wir festgestellt haben, bestimmt jede Regel eine Sprache. Hat man die Sprache gelernt, dann hat man auch die Regel gelernt. Da die Regeln durch ihre Operationen bestimmt sind, hängt auch die zu lernende Sprache von diesen Operationen ab. Wir haben dann in Abbhängikeit der Operationen, die zu lernenden Sprachen definiert und schließlich auf Lernbarkeit untersucht. Dabei haben wir herausgefunden, dass Regeln mit mehr als einer *SkipTo*-Operation und endlichen Argumentsprachen nicht aus Text lernbar sind. Regeln mit

endlichen Argumentensprachen, die aus Text lernbar sind, besitzen entweder nur eine Operation oder bestehen aus einer *SkipTo*- mit einer *SkipUntil*- bzw. *Next*-Operation. Zu vermuten ist, dass dann auch die ursprünglichen Regeln mit Disjunktionen nicht aus Text lernbar sind.

Trotz unserer Vermutung hinsichtlich der Lernbarkeit von disjunktiven Regeln, könnten aufbauend auf dieser Arbeit in zukünftigen Arbeiten untersucht werden, wie sich Disjunktionen auf die Lernbarkeit von Regeln auswirken. Da wir die verschiedenen Regeln bezüglich des induktiven Lernens aus Text untersucht und bei vielen Regeln ein negatives Ergebnis gefunden haben, wäre es interressant zu wissen, wie die Regeln bezüglich anderen Lernmodellen lernbar sind. Es wäre z.B. die Frage, ob ein Wrapper dieses Modells mit Hilfe von *Consistency Queries* (vgl. [GJL02] lernbar ist.

Literatur

[And00] Anderson, John R.: *Learning and Memory: An Integrated Approach*. John Wiley & Sons, Inc., New York, 2000.

[Ang80] Angluin, Dana: *Inductive inference of formal languages from positive data*. Information and Control, 45:117–135, 1980.

[CM99] Califf, M. E. und R. J. Mooney: *Relational Learning of Pattern-Match Rules for Information Extraction*. In: *Proc. 16th National Conference on Artificial Intelligence (AAAI-99)*, Seiten 328–334, Orlando, FL, 1999. AAAI Press.

[Eik99] Eikvil, Line: *Information Extraction from World Wide Web - A Survey*. Technischer Bericht 945, Norweigan Computing Center, 1999.

[Fre98a] Freitag, Dayne: *Information Extraction from HTML: Application of a General Machine Learning Approach*. In: *Proc. 15th National Conference on Artificial Intelligence (AAAI-98)*, Seiten 517–523, Madison, WI, 1998. AAAI Press.

[Fre98b] Freitag, Dayne: *Machine Learning for Information Extraction in Informal Domains*. Doktorarbeit, Carnegie Mellon University, 1998.

[Fre98c] Freitag, Dayne: *Multistrategy Learning for Information Extraction*. In: *Proc. 15th International Conference on Machine Learning (ICML-98)*, Seiten 161–169, San Francisco, 1998. Morgan Kaufmann.

[GJL02] Grieser, Gunter, Klaus P. Jantke und Steffen Lange: *Consistency Queries in Information Extraction*. In: Cesa-Bianchi, Nicolò, Masayuki Numao und Rüdiger Reischuk (Herausgeber): *Proc. 13th International Conference on Algorithmic Learning Theory*, Band 2533 der Reihe *Lecture Notes in Artificial Intelligence*, Seiten 173–187, Berlin, 2002. Springer-Verlag.

[GJLT00] Grieser, Gunter, Klaus P. Jantke, Steffen Lange und Bernd Thomas: *A Unifying Approach to HTML Wrapper Representation and Learning*. In: Arikawa, S. und S. Morishita (Herausgeber): *Proc. 3rd International Conference on Discovery Science*, Band 1967 der Reihe *Lecture Notes in Artificial Intelligence*, Seiten 50–64, Berlin, 2000. Springer-Verlag.

[GL01] Grieser, Gunter und Steffen Lange: *Learning Approaches to Wrapper Induction*. In: Russel, I. und J. Kolen (Herausgeber): *Proc. 14th International FLAIRS Conference*, Seiten 249–253. AAAI-Press, 2001.

[Gol67] Gold, Mark E.: *Language identification in the limit*. Information and Control, 14:447–474, 1967.

[HD98] Hsu, Chun-Nan und Ming-Tzung Dung: *Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web.* Information Systems, 23(8):521–538, 1998.

[HMU02] Hopcroft, John E., Rajeev Motwani und Jeffrey D. Ullman: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie.* Pearson Studium, München, 2002.

[Huf95] Huffmanm, Scott B.: *Connectionist, Statistical, and Symbolic Approaches to Learning information extraction patterns from examples.* In: *Learning for Natural Language Processing*, Band 1040, Seiten 246–260, Berlin, 1995. Springer-Verlag.

[JL89] Jantke, Klaus P. und Steffen Lange: *Algorithmisches Lernen.* In: Grabowski, Jan, Klaus P. Jantke und Helmut Thiele (Herausgeber): *Grundlagen der künstlichen Intelligenz*, Seiten 246–277. Akademie-Verlag, Berlin, 1989.

[JORS99] Jain, Sanjay, Daniel Osherson, James S. Royer und Arun Sharma: *Systems That Learn: An Introduction to Learning Theory.* MIT Press, Cambridge, Massachusetts, London, England, 1999.

[KM95] Kim, Jun-Tae und Dan I. Moldovan: *Acquisition of linguistic patterns for knowledge-based information extraction.* IEEE Transactions on Knowledge and Data Engineering, 7(5):713–724, 1995.

[Kru95] Krupka, George R.: *Description of the sra system as used for muc-6.* In: *Proc. 6th Message Understanding Conference (MUC-6)*, Seiten 221–235, San Fransisco, 1995. Morgan Kaufmann.

[KT03] Kushmerick, Nicholas und Bernd Thomas: *Adaptive information extraction: Core technologies for information agents.* In: Klusch, Matthias, Sonia Bergamaschi, Peter Edwards und Paolo Peta (Herausgeber): *Intelligent Information Agents R&D in Europe: An AgentLink perspective*, Band 2586 der Reihe *Lecture Notes in Computer Science*, Seiten 79–103, Berlin, 2003. Springer-Verlag.

[Kus97] Kushmerick, Nicholas: *Wrapper Induction for Information extraction.* Doktorarbeit, University of Washington, 1997.

[Kus00] Kushmerick, Nicholas: *Wrapper Induction: Efficiency and expressiveness.* Artificial Intelligence, 118(1-2):15–68, 2000.

[Lan00] Lange, Steffen: *Algorithmic Learning of Recursive Languages.* Mensch & Buch, Berlin, 2000.

[Mit97] Mitchell, Tom M.: *Machine Learning.* McGraw-Hill, 1997.

[MMK99] Muslea, Ion, Steve Minton und Craig Knoblock: *A hierarchical approach to wrapper induction.* In: Etzioni, Oren, Jörg P. Müller und Jeffrey M. Bradshaw (Herausgeber): *Proc. 3rd International Conference on Autonomous Agents (Agents'99)*, Seiten 190–197, Seattle, WA, USA, 1999. ACM Press.

[Mus99] Muslea, Ion: *Extraction Patterns for Information Extraction Tasks: A Survey.* In: *Proc. AAAI-99 Workshop on machine learning for information extraction*, 1999.

[Mus02] Muslea, Ion: *Active Learning with Multiple Views.* Doktorarbeit, University of Southern California, 2002.

[Qui90] Quinlan, J. R.: *Learning Logical Definitions from Relations.* Machine Learning, 5:239–266, 1990.

[Ril93] Riloff, Ellen: *Automatically Constructing a Dictionary for Information Extraction Tasks.* In: *Proc. 11th National Conference on Artificial Intelligence (AAAI-93)*, Seiten 811–816, 1993.

[Ril96] Riloff, Ellen: *Automatically generating extraction patterns from untagged text.* In: *Proc. 13th National Conference on Artificial Intelligence (AAAI-96)*, Seiten 1044–1049, 1996.

[SFAL95] Soderland, Stephen, David Fisher, Jonathan Aseltine und Wendy Lehnert: *CRYSTAL: Inducing a Conceptual Dictionary.* In: *Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Seiten 1314–1319, San Francisco, 1995. Morgan Kaufmann.

[Sod97] Soderland, Stephen: *Learning to Extract Text-Based Information from the World Wide Web.* In: *Proc. 3rd International Conference on Knowledge Discovery and Data Mining (KDD-97)*, Seiten 251–254, 1997.

[Sod99] Soderland, Stephen: *Learning Information Extraction Rules for Semi-Structured and Free Text.* Journal of Machine Learning, 34(1-3):233–272, 1999.

[Sol64] Solomonoff, R. J.: *A Formal Theory of Inductive Inference.* Information and Control, 7:1–22,224–254, 1964.

[Val84] Valiant, L.: *A Theory of the Learnable.* Communications of the Association for Computing Machinery, 27(11):1134–1142, 1984.

[ZL95] Zeugmann, Thomas und Steffen Lange: *A Guided Tour Across the Boundaries of Learning Recursive Languages.* In: *Algorithmic Learning for Knowledge-Based Systems*, Band 961, Seiten 190–258. Springer-Verlag, Berlin, 1995.

Danksagung

Danken möchte ich Dr. Gunter Grieser für seine gute Betreuung und seinen vielen Tips während meiner Arbeit.

Weiterhin möche ich Prof. Dr. Johannes Fürnkranz für die lockere Atmosphäre während der Diplomandenseminare des Fachgebietes Knowledge Engineering danken.

Meiner Freundin Alena danke ich dafür, dass sie mich während der letzten Phase der Diplomarbeit immer wieder aufgemuntert hat.

Zu guter Letzt möche ich meinen Eltern für ihre kontinuierliche Unterstützung während meines Studiums danken.