Eine Rekonstruktion der LR-Theorie zur Elimination von Redundanz

mit Anwendung auf den Bau von ELR-Parsern

vorgelegt von Diplom-Informatiker
Sönke Kannapinn

Am Fachbereich 13, Informatik,
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

vorgelegte Dissertation

Promotionsausschuß:

Vorsitzender:	Prof. Dr. Günter Hommel
Berichter:	Prof. Dr. Bleicke Eggers
Berichter:	Prof. Dr. Erhard Konrad
Berichter:	Prof. Dr. Dr. h.c. Hans Langmaack

Tag der wissenschaftlichen Aussprache: 6.7.2001

Berlin 2001

D 83

Eine Rekonstruktion der LR-Theorie zur Elimination von Redundanz

mit Anwendung auf den Bau von ELR-Parsern

Sönke Kannapinn

Gesetzt in der Stempel Garamond 12/17 Pkt. und der Gill Sans.
Satz mit LaTeX durch den Autor.
Grafiken erstellt mit Canvas auf Apple Macintosh.

Zusammenfassung

In dieser Arbeit befassen wir uns mit zwei Problemen aus dem Themenfeld der Konstruktion sogenannter LR-Parser und damit dem Gebiet der Syntaxanalyse für kontextfreie Sprachen.

Zum ersten zeigen wir auf, daß die herkömmliche LR-Parserkonstruktionstechnik Parser produziert, die erhebliche systematische Redundanzen aufweisen. Wir erarbeiten eine neue, theoretisch fundierte Methode, die sogenannte *allgemeine LR(k)-Parser* definiert. Zu einer gegebenen kontextfreien Grammatik charakterisiert die Definition eine ganze Familie von Kellertransduktoren, die alle die Form des entsprechenden kanonischen (und gleichzeitig „allgemeinen“) LR(k)-Parsers haben, diesem im Verhalten völlig gleichwertig sind und dessen Schlüsseleigenschaften der linearen Laufzeit und des Determinismus bewahren. Dennoch ist die formale Größe allgemeiner LR(k)-Parser potentiell deutlich kleiner als die des kanonischen Pendants. Wir zeigen, wie wir durch die Anwendung von aus der Automatentheorie bekannter Minimierungstechnik zu minimal großen allgemeinen LR(k)-Parsern gelangen können und zu „ziemlich kleinen“ auch durch eine direkte Konstruktion. In der Folge übertragen wir das Verfahren auch auf *allgemeine SLR(k)- und LALR(k)-Parser* und analysieren im Zuge dessen die neue Klasse der *ILALR(k)-Grammatiken,* für deren Mächtigkeit sich die Beziehung $\mathrm{LALR}(k) \subsetneq \mathrm{ILALR}(k) \subsetneq \mathrm{LR}(k)$ für $k > 0$ ergibt. Dabei ist ein kanonischer ILALR(k)-Parser sogar von kleinerer, höchstens gleicher Größe als der kanonische LALR(k)-Parser. Schließlich beschreiben wir auch einen Weg zur geschickten Implementierung der formal entworfenen Parser. Mit dem neuen Verfahren können wir klarstellen, daß die klassisch konstruierten LR-Parser Eigenschaften aufweisen, die für ihr korrektes Funktionieren im allgemeinen verzichtbar sind und mit Redundanz in der Konstruktion bezahlt werden. Wir arbeiten die in den traditionellen LR-Parsern unverzichtbaren Konstruktionsbestandteile und -eigenschaften heraus und tragen damit auch in didaktischer Hinsicht zum Verständnis der Konstruktion solcher Parser bei.

Zum zweiten beleuchten wir in dieser Arbeit die Konstruktion von LR-Parsern für die sogenannten erweiterten kontextfreien Grammatiken, von denen Syntaxdiagramme und die Erweiterte Backus-Naur-Form sowie die *regular right part grammars* verschiedene Erscheinungsformen sind. Auf diesem Gebiet ist bis heute keine theoretisch gänzlich befriedigende Arbeit veröffentlicht worden. Wir analysieren und kritisieren die bekannten Veröffentlichungen zum Thema; hierbei bildet ein eigenes, fundiertes Verfahren eine wesentliche Grundlage, und wir greifen auf Ergebnisse des ersten Teils der Arbeit zurück. Nebenbei präzisieren wir die Rolle der ε-Normalform von Brüggemann-Klein im Zusammenhang mit der Eindeutigkeit regulärer Ausdrücke.

Abstract

In this thesis, we present work on two problems from the field of LR parser construction, a family of syntax analysis techniques for context-free languages.

In the first part, we show that the traditional LR parser construction technique produces parsers which are burdened with a substantial amount of systematic redundance. We develop a new and well-founded method which defines what we call *general LR(k) parsers.* For a given grammar, the definition characterizes a whole family of pushdown transducers the members of which all have the shape of the respective canonical LR(k) parser (which is also "general"), share its essential behaviour, and preserve its important linear-run-time and determinism properties. Yet, the formal size of general LR(k) parsers is potentially much smaller than the formal size of their canonical version. We demonstrate how to construct general LR(k) parsers of minimal size by applying minimization techniques from automata theory, and how to construct "nearly-minimal" general LR(k) parsers directly. We then adopt the method to the construction of *general SLR(k) and LALR(k) parsers,* and introduce the class of *ILALR(k) grammars.* Here, we show that, concerning grammar classes, the relation $\mathrm{LALR}(k) \subsetneq \mathrm{ILALR}(k) \subsetneq \mathrm{LR}(k)$ holds for $k > 0$ while at the same time the formal size of a canonical ILALR(k) parser is smaller than or equal to the formal size of a canonical LALR(k) parser. As a round-up of the first part, we sketch an apt implementation of the formally presented parsers. The new method distills those properties of LR parsers that are indispensable for their correctness; and it shows which ingredients of the traditional construction technique are responsible for guaranteeing correctness and which are responsible for nice-to-have yet dispensable properties that cause redundance.

In the second part, we investigate the construction of LR parsers for the so-called extended context-free grammars of which syntax diagrams, Extended Backus–Naur Form, and regular right part grammars are different concrete versions. In that area of research, no work has been presented to the present day that is completely satisfying and theoretically deep. We analyze and criticize the relevant journal publications, and we develop, to that end, a theoretically well-founded method to construct such parsers profiting from results from the first part of this work. In the given context, we are also concerned with the concept of unambiguity of regular expressions, and we are able to state the role of Brüggemann-Klein's epsilon normal form of regular expressions more precisely.

Vorwort

Diese Arbeit hat viele Wurzeln in der Lehrveranstaltung *Syntaxanalyse* (früher *Compiler-Generierung 1),* die ich als Wissenschaftlicher Mitarbeiter mit Lehraufgaben an der Lehreinheit „Programmiersprachen und Compiler“ des Fachbereichs Informatik der Technischen Universität Berlin auch zweimal selbst lesen durfte.

Mein Vorgänger Stefan Krause war es, der die damals ganz neue Monographie von Sippu und Soisalon-Soininen (1988, 1990) über *Parsing Theory* „entdeckte“ und damit begann, Elemente daraus für die Präsentation von Lehrveranstaltungsstoff zu verwenden. Die Nähe der Vorlesung zu dieser Monographie haben Mario Kröplin und ich später weiter ausgebaut; in die Übung führte Mario als größere Programmieraufgabe die Erstellung eines ELL(1)-Parsergenerators für Grammatiken in EBNF nach dem Prinzip des rekursiven Abstiegs ein. Wie sich die vorteilhaft lesbaren erweiterten kontextfreien Grammatiken mit der Top-down-Parserstrategie behandeln ließen, schien uns genügend klar. Aber auch aus Sippus und Soisalon-Soininens Monographie war zu erkennen, daß die Ergebnisse der einschlägigen Literatur, dasselbe mit Bottom-up-Verfahren durchzuführen, nicht sofort überzeugten. Eine ausgegebene Diplomarbeit sollte die Arbeit von Chapman (1984) beleuchten; hierbei entdeckte Ungereimtheiten, denen ich nachging, und meine folgenden ersten Ansätze, dem Studenten ein fehlendes theoretisches Fundament nachzuliefern, brachten plötzlich so handfeste Ergebnisse zu Tage, daß ich beschloß, meinen zuerst gewählten Forschungsschwerpunkt (inkrementelle Attributevaluation) aufzugeben und mich auf LR-Theorie zu konzentrieren. Weitere Ergebnisse folgten nach. Wie so oft hat aber schließlich die topologisch sortierte Reihenfolge, in der sie in dieser Arbeit präsentiert werden, nur noch wenig mit der Chronologie gemein, in der sie gereift sind.

Ich möchte meinem Lehrer und ehemaligem Chef Prof. Bleicke Eggers danken für die vielen gemeinsam gegangenen Jahre und die universitäre Heimat, die er mir geboten hat; an seiner Lehreinheit zu lernen, zu forschen und zu lehren hat mich geprägt und mir viel bedeutet.

Prof. Hans Langmaack und Prof. Erhard Konrad danke ich sehr herzlich dafür, daß sie sich spontan als weitere Gutachter zur Verfügung gestellt haben.

Meinem ehemaligen Kollegen Mario Kröplin möchte ich danken für die präzise und konstruktive Kritik beim Korrekturlesen dieser Arbeit; von seiner unnachahmlichen Art,

Wissen zu strukturieren und mit anderen zu teilen, habe ich viel gelernt.

Für aufmerksame Lektüre und Anmerkungen danke ich auch Denis Kuniß, Rolf Müller und Stephan Weber.

Berlin im Januar 2001

Sönke Kannapinn

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

LR(k)-Parser

Die LR-Theorie stellt seit der Einführung von LR(k)-Grammatiken und -Parsern durch Knuth (1965) vor rund 35 Jahren im Bereich des Übersetzerbaus das Königsverfahren der Parserkonstruktion für Übersetzer und gehört seit langem zum Standardrepertoire von Vorlesungen und Büchern über generative Syntaxanalyse und Übersetzerbau: Deterministische LR(k)-Parser stellen die mächtigste effiziente Parsermethode für kontextfreie Grammatiken (kfGen) dar, die ihre Eingabe in der „natürlichen“ Richtung liest und dabei Lookahead auf k Zeichen beschränkt. Darüber hinaus erlauben sie laufzeiteffiziente Implementierungen. Der Vorteil der größtmöglichen Mächtigkeit wird jedoch mit praktischen und theoretischen Nachteilen erkauft.

Auf seiten der Theorie ist ein Preis für die Mächtigkeit von LR-Parsern, daß die LR-Theorie sicherlich nicht einfach zu nennen ist. Die vorliegende Arbeit zeigt, daß selbst mehr als 30 Jahre nach ihrer Entdeckung noch immer erstaunliche grundsätzliche Fortschritte zu erzielen sind.

Als ein wesentlicher praktischer Nachteil von LR-Parsern wird ihre beträchtliche Größe empfunden. Dies gilt bekanntlich für echte LR(k)-Parser in noch stärkerem Maße als für abgeleitete Varianten wie die von DeRemer (1969, 1971) definierten LALR(k)- und SLR(k)-Parser, die ihre gegenüber LR(k)-Parsern ($k \geq 1$) geringere Größe durch mindere Verfahrensmächtigkeit erkaufen. Etliche Veröffentlichungen widmen sich direkt oder indirekt diesem Größenproblem. Ihre prinzipielle Methode ist meist implementierungsorientiert, indem die Standard-Itemmengenkonstruktion, etwa in der „kanonischen“ Fassung von Aho und Ullman (1972), der Ausgangspunkt ist und verschiedene Optimierungen, wie z. B. die Elimination von Einheitsreduktionen oder die Analyse von unnützen Fehlereinträgen, nachträglich auf dem formalen Resultat oder sogar dessen Repräsentation als Parsertabellen angesetzt werden.

Eine Ausnahme bildet die Parserkonstruktionstechnik von Pager (1977), die durch konfliktsichere Verschmelzungen von LR(1)-Itemmengen während des Konstruktionsprozesses Parser produziert, die man als LALR(1)-LR(1)-Zwitter bezeichnen könnte und die nicht mehr das reine Verhalten eines LR(k)-Parsers aufweisen. Für eine ausführliche Diskussion entsprechender Literatur siehe die Bibliographischen Anmerkungen am Ende von Kapitel 7

in (Sippu und Soisalon-Soininen 1990). Eine weitere Ausnahme bildet die experimentelle Arbeit von Fortes Gálvez (1992), die ein unorthodoxes Verfahren zur Konstruktion von LR(1)-Parsern kleiner Größe skizziert, aber Parser von nicht mehr linearer Laufzeit produziert.

Kanonische LR(k)-Parser

Formalisiert sind LR-Parser zu einer gegebenen kontextfreien Grammatik G sogenannte Kellertransduktoren, d. h. also Kellerautomaten mit Ausgabe, vom Typus „bottom up“ und damit, einfach erklärt, gewisse Verfeinerungen des einfachsten Kellerautomaten dieses Typs: des sogenannten Shift-reduce-Parsers zu G. Dieser Shift-reduce-Parser zu G besitzt Shift-Aktionen, die die Aufgabe haben, das jeweils nächste Eingabesymbol zu lesen und zu kellern, und Reduce-Aktionen, die jeweils die Anwendung einer Grammatikregel erkennen, indem sie auf dem Keller eine rechte Grammatikregelseite durch ihre linke Seite ersetzen. Wenn wir Kellerautomaten als strukturierte Ersetzungssysteme im Stil von (Sippu und Soisalon-Soininen 1988) formulieren, also die Konfigurationen vom Aufbau $\$\Phi\mid x\$$ mit dem Kellerinhalt Φ, der Resteingabe x und den in Φ und x nie vorkommenden Zeichen $\$$ und $\mid$ sind, dann sind

$$\mid a \to a \mid \qquad \text{ohne Ausgabe} \qquad (\text{„Shift von } a\text{“})$$

für jedes Terminalsymbol a von G die Shift-Aktionen des Shift-reduce-Parsers zu G. Und die Aktionen

$$\omega \mid \to A \mid \qquad \text{mit Ausgabe } (A \to \omega) \qquad (\text{„Reduktion von } \omega \text{ nach } A\text{“})$$

für $A \to \omega$ Produktion von G sind seine Reduce-Aktionen. Dieser Kellertransduktor ist, wenn G nicht trivial ist, nichtdeterministisch. Er akzeptiert aber dennoch genau die von G aufgespannte Sprache $L(G)$ und produziert dabei in der Ausgabe alle Beweise in Form der umgekehrten Rechtsableitung zur jeweiligen Eingabe. Anschaulich wird gedacht der jeweils gesuchte Ableitungsbaum, „links unten“ beginnend, „bottom up“ aufgebaut.[1]

Wir definieren die wichtige Klasse der LR(k)-Grammatiken in dieser Arbeit als diejenigen Grammatiken, deren Shift-Reduce-Parser durch verfeinernden Umbau der Aktionen „deterministisch gemacht“ werden können, ohne dabei das Akzeptanzverhalten zu beschneiden, wobei der Lookahead in die Resteingabe nicht länger als k Zeichen sein darf. Dies ist

[1] Wir setzen die übliche Konvention der graphischen Darstellung von Ableitungsbäumen voraus.

essentiell die Definition von Heilbrunner (1981), jedoch nicht mit Hilfe von Heilbrunners unendlichen „parsing relations“ formuliert, mit denen Heilbrunner etwas unglücklich von den für die Syntaxanalyse-Theorie und -Praxis so wichtigen Kellertransduktoren abstrahiert, sondern konsequent für Kellertransduktoren. Für die präzise Definition stellen wir formale Mittel bereit, die auch mathematisch beschreiben, was es in obiger informeller Definition heißt, einen nichtdeterministischen Kellertransduktor deterministisch zu machen, ohne sein Akzeptanzverhalten zu beschneiden. Über ihren didaktischen Nutzen hinaus zeigt die Definition in dieser Arbeit auch zentralen beweistechnischen Wert.

Ein allgemeines Verfahren, zu beliebigen Kellerautomaten gleichwertige deterministische zu finden, existiert nicht. Für LR(k)-Parser kann jedoch von der speziellen Eigenschaft eines Shift-reduce-Parsers profitiert werden, daß die Menge der in seinen terminierenden Berechnungen vorkommenden (sog. „lebensfähigen“) Kellerinhalte mit ihren k-Lookaheads eine reguläre Sprache bildet (Knuth 1965): Für die Überwachung der Terminierbarkeit des Kellerinhalts und des Vorliegens von dazu passendem k-Lookahead kann ein endlicher Hilfsautomat konstruiert werden. Und da die Aktionen des Parsers zudem seine Konfigurationen stets nur an Keller*ende* und Resteingabe*anfang* manipulieren, kann die Übergangsstruktur dieses (deterministisch gemachten) Hilfsautomaten in die Kellerinhalte bzw. die Aktionen des Shift-reduce-Parsers „hineingefaltet“ werden. Eine Shift-Aktion von a ist dann von der Form

$$q_0 \mid ax \to q_0\, a\, q_1 \mid x \qquad \text{mit } |x| \leq \max\{k \Leftrightarrow 1, 0\}\,,$$

eine Reduce-Aktion einer Regel $A \to X_1 \ldots X_n$ ist von der Form

$$q_0\, X 1\, q_1 \ldots X_n\, q_n \mid y \to q_0\, A\, q_1' \mid y \qquad \text{mit } |y| \leq k\,,$$

wobei die X_i einzelne Grammatiksymbole sind und die q_i passende Zustände, die der Hilfsautomat für den Kellerinhalt durchläuft. Die Laufzeit des so entstehenden LR(k)-Parsers verhält sich linear zur Länge der Eingabe.

Kern des Standardverfahrens zur Konstruktion eines geeigneten endlichen Hilfsautomaten und zum Ablesen der LR(k)-Parseraktionen davon ist das Konzept des zu einem Kellerinhalt γ „LR(k)-gültigen Items“ $[A \to \alpha \cdot \beta, y]$. $A \to \alpha\beta$ ist hier eine Grammatikregel, von deren rechter Seite bereits α im linken Kontext δ erkannt und gekellert ist ($\gamma = \delta\alpha$), und nach Erkennung von β und Reduktion von $\alpha\beta$ nach A ist y erlaubter k-Zeichen-Anfang der Resteingabe. Die Eigenschaft, dieselben gültigen LR(k)-Items zu besitzen, etabliert eine Äquivalenzrelation von endlichem Index auf den möglichen Kellerinhalten,

und die Zustände des deterministischen endlichen Hilfsautomaten des Standardverfahrens werden mit Mengen von LR(k)-Items identifiziert, die diese Äquivalenzrelation auf den Kellerinhalten induzieren. Dabei sind die verschiedenen Itemmengen an den Zuständen des Hilfsautomaten nur für die Parser*konstruktion* relevant, wo sie die zusätzlich zur Übergangsstruktur des Hilfsautomaten benötigte Information enthalten, die zum Ablesen der Parseraktionen benötigt wird, während der entstandene Parser selbst die Inhalte der Itemmengen nicht mehr „kennt“.

Nach Aho und Ullman (1972) und Sippu und Soisalon-Soininen (1990) nennen wir den vom Standardverfahren konstruierten LR(k)-Parser den *kanonischen LR(k)-Parser* und den dazu verwendeten Hilfsautomaten die *kanonische LR(k)-Maschine* zur Grammatik.

Redundanz in LR-Parsern

Der skizzierten Standardkonstruktion wohnen deutliche Redundanzen inne: Bekanntermaßen ziehen *implementierte* LR-Parser zum Standardverfahren bei ihren Aktionsanwendungen lediglich den aktuell zuoberst gekellerten Zustand der kanonischen LR(k)-Maschine und die aktuell nächsten k Eingabezeichen heran, um folgende Entscheidungen deterministisch zu treffen:

1. Ist das nächste Eingabezeichen zu shiften, oder muß eine Reduktion angewendet werden, oder liegt ein Syntaxfehler vor?
2. Und falls zu reduzieren ist, welches ist die rechte und welches die linke Seite der Grammatikregel gemäß der reduziert wird?

Der klassische kanonische LR(k)-Parser kann also eine Reduce-Aktion

$$q_0\, X_1\, q_1 \ldots X_n\, q_n \,|\, y \rightarrow q_0\, A\, q_1' \,|\, y \tag{1}$$

anwenden, wenn eine Konfiguration

$$\$ \ldots ? \ldots q_0\, X_1\, q_1 \ldots X_n\, q_n \,|\, y \ldots ? \ldots \$$$

vorliegt.

Um deterministisch zu entscheiden, *daß* hier die Reduktion von $X_1 \ldots X_n$ zu A die einzig korrekt anzuwendende Parseraktion ist, genügt es dem kanonischen LR(k)-Parser zur

Grammatik, gerade einmal die dunkelgrau unterlegten Teile auf der linken Seite der gezeigten Reduce-Aktion (1) bzgl. Keller und Resteingabe vorzufinden. Der hellgrau unterlegte Teil wird nur noch dafür benötigt, den neuen zuoberst zu kellernden Hilfsautomatenzustand q_1' zu bestimmen. Der gesamte nicht unterlegte Teil der linken Aktionenseite muß auf dem Keller nicht referenziert werden. Dies weist darauf hin, daß das auf Knuth (1965) zurückgehende Standardverfahren Redundanz beinhaltet.[2]

Offensichtlich trägt der bei einer Reduktion zuoberst auf dem Keller liegende Zustand der kanonischen LR(k)-Maschine mehr Information, als nötig wäre. Wir gelangen in dieser Arbeit zu der Einsicht, daß dafür ursächlich das Konzept der LR(k)-gültigen Items verantwortlich ist, das der gesamten Konstruktion zugrunde liegt. Dieses Konzept führt letztlich dazu, daß die Struktur der kanonischen LR(k)-Maschine die von ihm akzeptierte reguläre Sprache unnötig fein in Äquivalenzklassen strukturiert. Um die vorhandene Redundanz zu eliminieren, muß dieses Konzept durch ein geeigneteres, redundanzfreies ersetzt werden.

Die von der LR(k)-Maschine akzeptierte reguläre Sprache muß dabei genau dieselbe bleiben. Allein aufgrund der Übergangsstruktur der LR(k)-Maschine, also *ohne* auf zusätzliche Information aus dem „Inneren“ ihrer Zustände zurückzugreifen, können allerdings die Parseraktionen im allgemeinen nicht vollständig abgelesen werden: Es fehlt die Lookahead-Information, die bei $k > 0$ für die Reduce-Aktionen und bei $k > 1$ auch für die Shift-Aktionen nirgendwo abgegriffen werden kann. Wir definieren als redundanzfreies Substitut für LR(k)-gültige Items das Konzept des *LR(k)-Rechtskontextes*, dessen Berechnung an den Zuständen der LR(k)-Maschine genau die benötigte Lookahead-Zusatzinformation bereitstellt. Die so entstandenen Hilfsautomaten nennen wir *allgemeine LR(k)-Maschinen* und formulieren dazu, wie aus ihnen Shift- und Reduce-Aktionen abzulesen sind.

Die neue Vorgehensweise ist kompatibel zum klassischen Verfahren im folgenden Sinne: Faßt man eine kanonische LR(k)-Maschine als allgemeine LR(k)-Maschine auf (Beibehalten der Übergangsstruktur; an den Zuständen Extraktion der benötigten LR(k)-Rechtskontexte aus den Itemmengen-Informationen, was geradlinig möglich ist), so ist der aus dieser allgemeinen LR(k)-Maschine abgelesene Kellertransduktor weiterhin genau der kanonische LR(k)-Parser zur Grammatik.

[2] Schon Knuth selbst weist übrigens in seinem Pionierartikel auf diesen Umstand hin, indem er schreibt, daß die bekannte Itemmengenkonstruktion *„shows essentially all of the information available during parsing, and much of it can be recognized as repetitive or redundant“*.

Den neugierigen Leser verweisen wir an dieser Stelle auf Abbildung 3 auf Seite 47, wo für ein kleines Beispiel eine kanonische LR(1)-Maschine und ihre Umformulierung als allgemeine LR(1)-Maschine gezeigt ist. Schon der optische Eindruck vermittelt, daß der Übergang von LR(k)-gültigen Items zu LR(k)-Rechtskontexten in erheblichem Maße redundante Information verwirft.

Zu beachten ist, daß allgemeine LR(k)-Maschinen durch *Eigenschaften* definiert sind, die sie einhalten, und nicht, wie die kanonischen, durch ein präzises Konstruktionsverfahren. Zu einer gegebenen Grammatik ergibt sich so eine ganze *Familie* von endlichen Automaten mit zusätzlichen Berechnungen an den Zuständen, die dieselben Basiseigenschaften aufweisen; und wie in dieser Arbeit demonstriert wird, existieren nicht nur allgemeine LR(k)-Maschinen mit der Übergangsstruktur der kanonischen LR(k)-Maschinen. Wir zeigen in dieser Arbeit, daß diese Basiseigenschaften genügen, um sicherzustellen, daß jeder aus einer beliebigen allgemeinen LR(k)-Maschine abgelesene Kellertransduktor genau dann ein deterministischer LR(k)-Parser ist, wenn die zugrunde gelegte Grammatik eine LR(k)-Grammatik ist. Die entstehenden Kellertransduktoren bezeichnen wir daher als *allgemeine LR(k)-Parser.*

Im Zentrum unseres Beweises der Behauptung, daß das Verfahren zu einer LR(k)-Grammatik tatsächlich stets deterministische LR(k)-Parser liefert, steht der Nachweis, daß sich zwei beliebige allgemeine LR(k)-Parser – wie wir es in dieser Arbeit präzise definieren – *stark gleich verhalten.* Und unter anderem schließt dies das Verhalten des kanonischen LR(k)-Parsers ein. Der Beweis ist damit gewissermaßen *relativ* zu dem bekannten Beweis für den kanonischen LR(k)-Parser, wie er z. B. in (Sippu und Soisalon-Soininen 1990) nachzulesen ist, womit versucht werden soll, die Bezüge zur Klassik möglichst klar herauszuarbeiten. Daß allgemeine LR(k)-Parser ihre Eingaben in linearer Zeit abarbeiten, ergibt sich ebenfalls sofort relativ aus dieser Eigenschaft der kanonischen LR(k)-Parser.

Allgemeine LR(k)-Maschinen und -Parser sind minimierbar

Kanonische wie allgemeine LR(k)-Maschinen sind letztlich Moore-Automaten: endliche Automaten mit Berechnungen an den Zuständen. Damit sind Techniken aus der Automatentheorie zur Minimierung der Zustandsanzahl auf sie anwendbar. Während schon aufgrund der Konstruktion keine zwei Zustände einer kanonischen LR(k)-Maschine zusammenfaßbar sind, ist dies für die gleiche, aber als allgemeine aufgefaßte LR(k)-Maschine praktisch immer möglich, womit die erheblichen Redundanzen in der kanonischen LR(k)-Maschine eliminiert werden. Wir betrachten ebenfalls, wie sich die mit

den Zustandszusammenfassungen erklärte partielle Ordnung (modulo Isomorphie) auf die abgelesenen allgemeinen LR(k)-Parser überträgt. Es ergibt sich, daß die zu den minimalen allgemeinen LR(k)-Maschinen gehörenden allgemeinen LR(k)-Parser die geringste Anzahl von Aktionen, die geringste formale Größe und – im Nicht-LR(k)-Fall – die geringste Anzahl von Aktionenpaaren aufweist, in denen sich Nichtdeterminismus manifestiert. Wir sprechen daher von *minimalen allgemeinen LR(k)-Parsern.*

Der neugierige Leser mag sich an dieser Stelle für Abbildung 4 auf Seite 48 interessieren, wo für das oben erwähnte Beispiel die minimale allgemeine LR(1)-Maschine gezeigt ist.

Redundanzvermeidung schon bei der Konstruktion

Angenommen, wir wollten mit unserem bisherigen Kenntnisstand für eine gegebene Grammatik und ein gewähltes k einen minimalen allgemeinen LR(k)-Parser konstruieren. Dann kennen wir bis hierher nur die Technik, die kanonische LR(k)-Maschine zu konstruieren, diese dann als allgemeine LR(k)-Maschine „aufzufassen“ (LR(k)-Rechtskontexte statt LR(k)-gültiger Items) und als solche dann zu minimieren. Natürlich stört daran der hohe Ressourcenbedarf für das Zwischenprodukt, die kanonische LR(k)-Maschine. Attraktiver wäre ein Verfahren, mit dem die minimale allgemeine LR(k)-Maschine direkt konstruiert werden kann. Einige Überlegungen in dieser Arbeit legen nahe, daß ein solches Verfahren nicht existieren dürfte, wenn es mit der herkömmlichen Konstruktionstechnik über Items noch einigermaßen verwandt sein will. Wir entwickeln jedoch ein Verfahren, das besondere allgemeine LR(k)-Maschinen, die wir als „redundanzreduziert“ bezeichnen, direkt konstruiert:

Redundanzreduzierte LR(k)-Maschinen eliminieren gegenüber den kanonischen einen Großteil von deren Redundanz, wenn sie auch im allgemeinen geringfügig mehr Zustände aufweisen als die minimalen allgemeinen LR(k)-Maschinen. Zudem sind sie einfach zu konstruieren und weisen auch noch andere für die Praxis nützliche Eigenschaften auf. Sie sind daher nicht nur als Zwischenergebnis auf dem Weg zu minimalen allgemeinen LR(k)-Maschinen interessant, sondern machen wegen ihrer bereits guten Approximation der minimalen Zustandsanzahl den nachgeschalteten Minimierungsschritt in der Praxis eventuell ganz verzichtbar.

Die Schlüsselbeobachtung, die zu redundanzreduzierten LR(k)-Maschinen und ihren Parsern führt, ist, daß jedes einzelne Item $[A \to \alpha \cdot \beta, y]$, das LR($k$)-gültig zu einer Grammatiksymbolfolge γ ist, in Gestalt seiner Anteile A und α einen substantiellen Teil seiner – wie wir sagen – „individuellen Geschichte“ bezüglich γ codiert. Auf genau diese Anteile A

und α kann aber in der Konstruktion von LR(k)-Maschinen verzichtet werden; d. h. auch wenn nur noch die „zukunftsgerichteten“ Itemanteile β und y benutzt werden, können die Aktionen des Parsers durchaus noch abgelesen werden. Dies kann dazu genutzt werden, eine „Sparversion“ der kanonischen LR(k)-Maschine aufzubauen, die statt der vollen Items $[A \to \alpha \cdot \beta, y]$ nur noch Itemreste $[\beta, y]$ verwendet.

Technisch leisten wir die Konstruktion dadurch, daß wir, analog zur kanonischen Fassung, zunächst eine nichtdeterministische redundanzreduzierte LR(k)-Maschine mit Zuständen der Form $[\beta, y]$ definieren, deren Potenzkonstruktion dann die angestrebte *(deterministische) redundanzreduzierte LR(k)-Maschine* ist. Wir zeigen, daß die nichtdeterministischen und dann auch die deterministischen kanonischen vs. redundanzreduzierten LR(k)-Maschinen in einer Projektionsbeziehung stehen. Trotz nicht mehr vorhandener Itemteile A, α können der redundanzreduzierten LR(k)-Maschine immer noch leicht LR(k)-Rechtskontexte zugeordnet werden, die an ihren Zuständen berechnet werden; sie kann also als allgemeine LR(k)-Maschine angesehen werden. Daß der zugehörige *redundanzreduzierte LR(k)-Parser* (dessen Aktionen ebensogut auch direkt mit Hilfe der $[\beta, y]$-Itemreste abgelesen werden können) deterministisch genau dann ist, wenn die zugrunde liegende Grammatik eine LR(k)-Grammatik ist, erfordert damit keine Beweisarbeit mehr.

Der neugierige Leser findet die redundanzreduzierte LR(1)-Maschine für das oben erwähnte Beispiel in Abbildung 8 auf Seite 78.

Redundanzreduzierte LR(k)-Parser weisen eine für die Implementierung hilfreiche Eigenschaft auf: Sie sind *reduktionsdeterminiert.* So nennen wir LR(k)-Parser, die mit oberstem Kellerzustand und k-Lookahead stets bereits entscheiden können, *ob* eine Reduktion anzuwenden ist. Minimale allgemeine LR(k)-Parser weisen diese Eigenschaft i. a. nicht mehr auf.

Abgeleitete allgemeine LR-Parser

Die bekannten SLR(k)- und LALR(k)-Parser von DeRemer (1969) beruhen strukturell auf kanonischen LR(0)-Parsern, in deren Aktionen ein k-Lookahead ergänzt wird. Es ist naheliegend, die neuen Möglichkeiten zur Konstruktion erheblich kleinerer LR(0)-Maschinen als den kanonischen auch für diese abgeleiteten Parserverfahren nutzbar machen zu wollen. Dies ist durchaus mit Komplikationen verbunden.

Im Fall von LALR(k)-Parsern mag dies vielleicht nicht verwundern, denn in diesem Verfah-

ren, das sehr technisch durch Projektion jeder LR(k)-gültigen Itemmenge auf die entsprechende LR(0)-Itemmenge mit gleichem *core* erklärt ist, spielen ja die k-Items eine zentrale Rolle, von denen aber das neue Verfahren nun gerade abstrahiert. Um zu gleichwertigen *allgemeinen LALR(k)-Parsern* zu gelangen, müssen spezielle *LALR(k)-Rechtskontexte* definiert werden. Als Grundlage für allgemeine LALR(k)-Parser eignen sich die minimalen allgemeinen LR(0)-Maschinen i. a. nicht, da sie zu grobe Äquivalenzklassen induzieren. Wohl jedoch können die redundanzreduzierten LR(0)-Maschinen verwendet werden.

Für die in der Praxis populären LALR(1)-Parser bietet sich damit ein besonders attraktives Konstruktionsverfahren an: Redundanzreduzierte LR(0)-Maschinen sind implementierungstechnisch recht einfach zu berechnen. Und der effiziente Algorithmus von DeRemer und Pennello (1982) kann die LALR(1)-Lookaheads ohne Modifikation (ja, sogar effizienter) auch auf redundanzreduzierten statt kanonischen LR(0)-Maschinen bestimmen, da dieser Algorithmus nur die Übergangsstruktur, aber nicht die „Innereien" der LR(0)-Maschine benötigt.

Natürlich erlauben auch allgemeine LALR(k)-Parser – wie schon ihre LR(k)-Pendants – nicht mehr, allein mit k-Lookahead und oberstem Kellerzustand zu entscheiden, nach welcher Grammatikregel zu reduzieren ist. Auch für sie muß also in der Regel tieferer Kellerkontext referenziert werden. Dann aber bietet sich an, vom LALR(k)- zum *ILALR(k)-Verfahren („improved LALR")* überzugehen, das wir in dieser Arbeit vorstellen. Gegeben dieselbe Voraussetzung wie für einen LALR(k)-Parser, nämlich die kanonische LR(0)-Maschine, deren Itemmengen durch Projektion aus der kanonischen LR(k)-Maschine um k-Lookahead angereichert wurden, wird eine Reduce-Aktion nach $A \to \omega$ nicht aus dem Vorhandensein von $[A \to \omega \cdot, y]$ in der Itemmenge am Ende des erkannten Handles ω, sondern dem Vorhandensein von $[A \to \cdot\, \omega, y]$ in der Itemmenge am Beginn der Erkennung von ω abgelesen, wo schärfere Lookahead-Mengen bekannt sind.

Wir definieren ILALR(k)-Grammatiken und ihre Parser in dieser Arbeit präzise und zeigen bezüglich der Grammatik-Hierarchie: Jede LALR(k)-Grammatik ist auch eine ILALR(k)-Grammatik ($k \geq 0$.) Es existieren ILALR(1)-Grammatiken, die für kein $k \geq 0$ LALR(k)-Grammatiken sind. Und es existieren LR(1)-Grammatiken, die für kein $k \geq 0$ ILALR(k)-Grammatiken sind.

Und schließlich zeigen wir auf, daß für die effiziente Berechnung von ILALR(1)-Lookhead-Mengen aus der Übergangsstruktur von LR(0)-Maschinen eine leichte (vereinfachende) Modifikation des bekannten Algorithmus von DeRemer und Pennello (1982) eingesetzt werden kann. Wie für allgemeine LALR(k)-Parser sind auch für allgemeine ILALR(k)-

Parser minimale allgemeine LR(0)-Maschinen i. a nicht als Grundlage geeignet, wohl aber (neben den kanonischen) die redundanzreduzierten LR(0)-Maschinen.

Unerwartet kompliziert gestaltet sich die Formulierung *allgemeiner SLR(k)-Parser.* Der Grund hierfür liegt nicht in den Lookaheads für die Reduktionen, die sich natürlich als FOLLOW-Mengen alleine aus der Grammatik ergeben, sondern in den Lookaheads für die Shift-Aktionen (für $k \geq 2$). Hierfür greift DeRemer (1969) nämlich doch wieder auf Detailinformationen aus den kanonischen LR(0)-Itemmengen zurück, die in allgemeinen LR(0)-Maschinen nicht vorhanden sind. Folglich gelangen wir, wie analog schon im LALR(k)- und ILALR(k)-Fall, auch hier nur dann zu korrekten allgemeinen SLR(k)-Parsern, wenn wir spezielle *SLR(k)-Rechtskontexte* definieren, womit die Konstruktion entsprechender Parser recht kompliziert gerät und daher der Beschreibung *simple* aus dem Akronym SLR nicht mehr gerecht wird.

Implementierung von allgemeinen LR-Parsern

LR-Parser formal in ähnlicher Weise zu präsentieren, wie wir es in dieser Arbeit tun, hat bereits eine lange Tradition: Die Formulierung über Kellertransduktoren als spezielle Ersetzungssysteme, die wir aus (Sippu und Soisalon-Soininen 1988) übernommen haben, ist zwar eleganter als die *extended pushdown transducers* aus (Aho und Ullman 1972), aber von der Ausdruckskraft doch mehr oder weniger identisch. Dennoch scheint in der Standardliteratur über LR-Parsing nirgendwo beschrieben zu sein, wie ein *extended pushdown transducer* oder ein Kellertransduktor à la (Sippu und Soisalon-Soininen 1988) *allgemein* geschickt implementiert werden kann, sondern es wird immer nur die spezielle Implementierung der *kanonischen* LR-Parser mittels der bekannten Parseraktions- und GOTO-Tabellen wiedergegeben. Diese spezielle Implementierung profitiert aber davon, daß der implementierte Parser nur den obersten Kellereintrag und das k-Lookahead zu analysieren hat, um seine Entscheidungen zu fällen, während beliebige *extended pushdown transducers* oder Kellertransduktoren à la (Sippu und Soisalon-Soininen 1988) so nicht implementiert werden können. Für die Implementierung allgemeiner LR-Parser stellt daher ein eigener Anhangsabschnitt dieser Arbeit einen Vorschlag vor.

Da besonders die Tiefe des von den verschiedenen Parseraktionen einbezogenen Kellerkontextes unterschiedlich ist, bietet es sich prinzipiell an, zur Entscheidungsfindung Kellerinhalt sukzessive, beginnend beim zuoberst gekellerten Symbol, zu lesen, bis zusammen mit dem Lookahead klar ist, welche Aktion vom Parser anzuwenden ist. Alle anzutreffenden Situationen sind vorweg planbar: Mit den statisch bekannten Parseraktionen können

die gültigen Symbolfolgen, die rückwärts in den Keller hinein angetroffen werden dürfen, in Form von Entscheidungsbäumen geplant werden, deren Knoten jeweils beschreiben, welche Aktionen zum bereits angetroffenen oberen Kellerkontext noch passen und damit gleichzeitig die gültigen Fortsetzungen in den Keller hinein vorgeben. Abbildung 30 auf Seite 175 zeigt einen beispielhaften solchen Baum.

Bei der Repräsentation der an den Knoten zu fällenden Entscheidungen kann ausgenutzt werden, daß die angetroffenen Kellerinhalte nicht völlig beliebig sind, sondern stets lebensfähige LR-Kellerinhalte sind. Implementierungstechnisch ergeben sich dann dünn besetzte Vektoren, auf deren fehlende Einträge garantiert nicht zugegriffen wird. Ihre Inhalte können also mit heuristischen „Optimierungs"-Methoden platzsparend „ineinander" geschoben werden. Die Vorgehensweise weist insgesamt starke Ähnlichkeiten mit Implementierungstechniken für sogenannte Bottom-up-Baumautomaten auf, wie sie im generativen Compilerbau beispielsweise für die Berechnung von globaler Attributabhängigkeitsinformation oder für die Generierung effizienter Code-Selektoren eingesetzt werden (Wilhelm und Maurer 1992). Wir übertragen Ideen aus einem Artikel von Börstel, Möncke und Wilhelm (1991) auf die Gegebenheiten bei unseren implementierten Kellerautomaten und demonstrieren das vorgeschlagene Verfahren an einem Beispiel.

LR-Parser für erweiterte kontextfreie Grammatiken

Für Programmiersprachen mit komplexer Syntax weisen rein kontextfreie Grammatiken eine große, schwer zu überblickende Zahl von Regeln und Nichtterminalen auf. Deren große Zahl wird natürlich zum einen durch die inhärente Komplexität der Syntax bewirkt; starken Anteil daran hat aber auch die einfache Struktur der reinen kontextfreien Grammatiken, die es erforderlich macht, für viele bedeutungsmäßig untergeordnete Teilstrukturen bereits Nichtterminale mit „Hilfs"-Charakter einzuführen und thematisch zusammengehörige Teilstrukturen über mehrere Grammatikregeln zu verstreuen, was wiederum die Lesbarkeit der gesamten Grammatik stark beeinträchtigt. Ein kurzes Beispiel[3] illustriert, worin der Vorteil bestimmter Erweiterungsformen von kfGen gegenüber reinen kfGen besteht: Indem auf rechten Seiten (hier) reguläre Ausdrücke mit Iterations-, Alternativen- und Optionalkonstruktoren zugelassen werden (die Grammatik ist hier notiert in der Erweiterten Backus-Naur-Form, kurz EBNF), gerät die Grammatikpräsentation weitaus prägnanter: Beispielsweise muß die eine EBNF-Regel zum Nichtterminal **set**

[3] aus (Reiser und Wirth 1992)

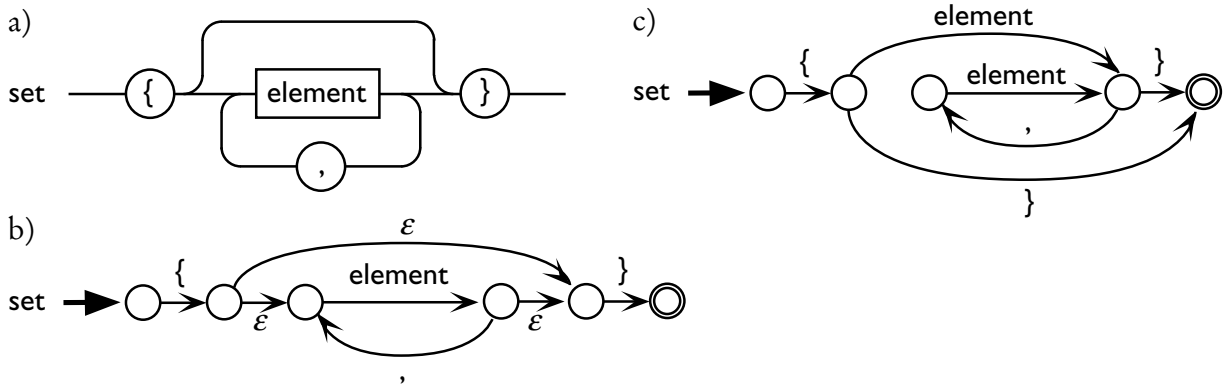
```
set = "{" [ element { "," element } ] "}" .
```

rein kontextfrei umständlicher formuliert werden; beispielsweise mit fünf (BNF-)Regeln

```
set = "{" optElemList "}" .
optElemList = elemList | .
elemList = element | elemList "," element .
```

wobei zwei Hilfsnichtterminale eingeführt werden müssen und die Lesbarkeit sich verschlechtert.

Ebenfalls ein populäres Beschreibungshilfsmittel für Programmiersprachengrammatiken sind Syntaxdiagramme (Jensen und Wirth 1974), mit denen auf der formalen Seite die sogenannten *regular right part grammars (RRPGen)* korrespondieren. Diese sind ebenfalls Erweiterungen kontextfreier Grammatiken dahingehend, daß bei ihnen die bei einem Ableitungsschritt für eine linke Seite einzusetzenden Symbolfolgen nicht durch einen regulären Ausdruck, sondern durch einen endlichen Automaten beschrieben werden. Wir zeigen obige EBNF-Regel a) als Syntaxdiagramm sowie in b) und c) umgeschrieben zu einem endlichen Automaten mit ε-Übergängen bzw. ohne solche.



Was die Produktion von Wörtern der insgesamt beschriebenen Sprache anbelangt, gelangen wir damit zu einem *zwei*stufigen Prozeß, da nun jeder einzelne Ableitungsschritt selbst wieder einen eingelagerten Prozeß zur Erzeugung der einzusetzenden Symbolfolge enthält. Grundsätzlich erlauben die so skizzierten Grammatiktypen, daß mit ihnen formale Sprachen i. a. strukturierter und somit übersichtlicher beschrieben werden. Dennoch beschreiben sie „nur" kontextfreie Sprachen, da die regulären Beschreibungshilfsmittel stets auch rein kontextfrei „simuliert" werden können. Wir wollen als gemeinsame Bezeichnung von *erweiterten kontextfreien Grammatiken (ekfGen)* sprechen.

Komplexe Programmiersprachen weisen oft eine weitere Eigenschaft auf: Intuitive kontextfreie Grammatiken für sie sind häufig nur mit LR-Verfahren zu bewältigen. Mit dem oben Gesagten läßt dies den Wunsch aufkommen, die LR-Analyseverfahren von den reinen kfGen auf die ekfGen zu übertragen, um die Vorteile der Prägnanz des Syntaxbeschreibungshilfsmittels und der Mächtigkeit des Analyseverfahrens zu vereinen. Auch besteht eine gewisse Hoffnung, daß sich die zwar aufwendigeren, aber dafür selteneren Reduktionen, die ein solcher Parser durchführen müßte, insgesamt effizienter bewerkstelligen lassen könnten als die vielen einfachen eines herkömmlichen LR-Parsers.

Versuche, LR-Parser für ekfGen (kurz: ELR-Parser) zu konstruieren, hat es in der Literatur seit Mitte der 70er Jahre immer wieder gegeben. Fast alle vorgeschlagenen Verfahren weisen jedoch letztlich dieselben Schwächen auf: Ihre theoretische Fundierung rangiert im Bereich zwischen „mangelhaft“ und „nicht vorhanden“. Zu rezeptartig und zu hastig werden Konstruktionsverfahren in den Vordergrund gestellt, anstatt zunächst die definitorischen Grundlagen (und ihre Konsequenzen) in genügender Tiefe zu bedenken.

Das wichtigste Problem, das für ELR-Parser in adäquater und theoretisch sauberer Weise gelöst werden muß, ist die Frage, wie nun, da nicht mehr nur endlich viele mögliche rechte Regelseiten zu ihren linken Seiten zu reduzieren sind, sondern es i. a. unendlich viele Kandidaten für „Handles“ gibt, der Parser für die Reduktion das vordere Ende eines aktuell vorliegenden Handles auf dem Keller lokalisieren kann. Natürlich muß diese Handle-Bestimmung aus variabel vielen Einzelschritten zusammengesetzt sein (worin dann die oben erklärte zweite Stufe des Ableitungsprozesses ihre Analogie findet). Fast alle Verfahren aus der Literatur konstruieren hierzu sogenannte *Readback-Automaten* – eine Art endlicher Automaten, die in den Keller hinein lesen, bis der Beginn des Handles erkannt wird, und dabei, analog dem Eingabe-Look*ahead* von k Symbolen, mit einem sogenannten Look*back* von m Kellersymbolen ausgestattet werden.

Hier nun, wo erwartungsgemäß die ohnehin recht schwierige klassische LR-Theorie für kfGen nochmals an Komplexität zunimmt, scheitert bei allen Literaturarbeiten, die Readback-Automaten konstruieren, die allzu zielorientierte Strategie, auf einen exakten theoretischen Unterbau zu verzichten und sogleich den Parser*konstruktionsalgorithmus* mit Verlaß auf die Intuition vermeintlich analog von kfGen auf ekfGen zu übertragen. So können wir dann auch in unserer Literaturanalyse eine ganze Reihe von z. T. krassen Fehlern und Fehlentscheidungen aufdecken. Die Grundlage für unsere Kritik legt eine sorgfältige theoretische Vorbereitung.

Wir nähern uns in dieser Arbeit der Frage, wann überhaupt wir eine erweiterte kontextfreie

Grammatik (ekfG) eine ELR(k)-Grammatik nennen wollen, sehr behutsam an. Unsere Begriffsbildung und Formalisierung orientiert sich am Rahmenwerk von Heilbrunner (1979), indem zunächst einmal für die Formulierung der regulären Sprachen auf den rechten Grammatikregelseiten nicht reguläre Ausdrücke oder endliche Automaten, sondern rechtslineare Subgrammatiken verwendet werden. Dies hat den Vorteil, daß so der Bereich der kontextfreien Grammatiken gar nicht erst verlassen wird und sich auf den klassischen LR(k)-Begriff berufen werden kann. Insbesondere ergibt sich als notwendige Voraussetzung für die ELR(k)-Eigenschaft, daß alle rechtslinearen Subgrammatiken eindeutig sind.

Für ekfGen in den üblichen Erscheinungsformen müssen die rechten Regelseiten zu äquivalenten rechtslinearen Subgrammatiken umformuliert werden. Bei EBNFen ist hierbei zu beachten, daß im Falle von eingebetteten Iterationen intuitive Transformationsmuster die benötigte Rechtslinearität nicht zusichern, sondern im Effekt den Reduktionszeitpunkt vom EBNF-Regelende zur eingebetteten Iteration vorverlegen. Analoges gilt auch für eingebettete Optional- und Alternativ-Konstrukte.

Schon weil im Kontext des Übersetzerbaus der *Struktur* der Grammatikregeln für spätere Übersetzungsphasen eine tragende Rolle zukommt, fordern wir in dieser Arbeit, daß die Umformulierung von endlichen Automaten oder regulären Ausdrücken zu äquivalenten Subgrammatiken *strukturerhaltend* erfolgen muß.

Für RRPGen mit ihren endlichen Automaten auf rechten Regelseiten kann jeder Automat in geradliniger Weise durch eine rechtslineare Subgrammatik ersetzt werden, die genau die Automatenstruktur widerspiegelt. Die Ein- bzw. Mehrdeutigkeit der Automaten überträgt sich auf die Ein- bzw. Mehrdeutigkeit der Subgrammatik.

Für ekfGen mit regulären Ausdrücken auf rechten Regelseiten existieren im wesentlichen zwei verschiedene Techniken, äquivalente nichtdeterministische endliche Automaten (NEAs) zu den regulären Ausdrücken zu konstruieren: Noch genau die induktive Struktur der jeweiligen regulären Ausdrücke geben die NEAs mit ε-Übergängen wieder, die in bekannter Weise induktiv über dem Aufbau der regulären Ausdrücke konstruiert werden. Eine Alternative ist die Konstruktion der sogenannten Glushkov-NEAs, die ε-frei sind und in deren Struktur sich der induktive Aufbau der zugehörigen regulären Ausdrücke nicht mehr immer eindeutig wiederspiegelt. Gleichzeitig sind die Glushkov-NEAs gerade diejenigen NEAs, die sich aus den oben erwähnten ε-NEAs durch die Standardtechnik der Elimination von ε-Übergängen ergeben. Die Eindeutigkeit eines regulären Ausdrucks wird mit der Eindeutigkeit entweder des zugehörigen ε-NEAs oder des zugehörigen Glushkov-

NEAs erklärt. Im ersten Fall spricht Brüggemann-Klein (1993) von starker, im zweiten von schwacher Eindeutigkeit und analysiert auch den formalen Zusammenhang zwischen beiden Eindeutigkeitskonzepten.

Wir können in der vorliegenden Arbeit nachweisen, daß dieser Zusammenhang *allein* durch die sogenannte ε-Normalform und nicht erst zusätzlich durch die Stern-Normalform von Brüggemann-Klein (1993) erklärt wird. Anschaulich ist ein regulärer Ausdruck in ε-Normalform, wenn durch keinen seiner Teilausdrücke das leere Wort mehrdeutig beschrieben wird. In ε-NEAs existieren dann keine mehrdeutigen ε-Wege, so daß Ein- bzw. Mehrdeutigkeit von ε- und Glushkov-NEAs zusammenfallen. Eine äquivalente rechtslineare Subgrammatik, die die induktive Struktur des regulären Ausdrucks genau wiedergibt, erhalten wir direkt aus den Übergängen des ε-NEAs; und gerade mit Hilfe der (in Grammatikregeln abgebildeten) durchlaufenen ε-Übergänge kann dann die induktive Entstehung eines Worts aus den Bestandteilen des regulären Ausdruck rekonstruiert werden. Konstruieren wir dagegen die Subgrammatik auf Grundlage der Übergänge des Glushkov-NEAs, so ist die Rekonstruktion induktiver Wortstruktur schwieriger und nur dann möglich, wenn der reguläre Ausdruck ε-Normalform aufweist und somit die von ε- zu Glushkov-NEAs „verlorengegangenen“ ε-Übergänge (zumindest gedacht) rekonstruiert werden können.

Im Zusammenhang mit unserer Definition des Begriffs der ELR(k)-Grammatik legen wir, wie oben angekündigt, Wert darauf, daß Transformationen rechter Regelseiten in äquivalente rechtslineare Subgrammatiken strukturerhaltend zu erfolgen haben. Und für ekfGen mit regulären Ausdrücken auf rechten Regelseiten ist es gerade die induktive Struktur der regulären Ausdrücke, die relevant ist. Daher ergibt sich für uns als notwendige Voraussetzung für die ELR(k)-Eigenschaft einer ekfG mit regulären Ausdrücken, daß alle regulären Ausdrücke stark eindeutig sein müssen.

Die Konstruktion von ELR(k)-Parsern für ekfGen ist für unseren Ansatz damit grundsätzlich geklärt, Modifikationen zu ELALR(k)- oder ESLR(k)-Verfahren sind offensichtlich, und aufgrund der systematischen Zurückführung auf reine kfGen liegt eine saubere theoretische Fundierung vor. Bemerkenswerterweise mußten wir uns dabei um spezielle Readback-Automaten nicht bemühen; sie ergeben sich sozusagen implizit:

So sind wir in der Lage, exemplarisch die von Chapman (1984) konstruierten Readback-Automaten seiner LALR(1,1)-Parser (d. h. $m = k = 1$) für RRPGen mit unseren rein kontextfreien Mitteln strukturell zu rekonstruieren. Zu beachten ist, daß wir zur Rekonstruktion *redundanzreduzierte* LALR(1)-Parser heranziehen müssen, die den ad hoc vorgehenden Autoren in der Literatur unbekannt sind. Leser, die hier an Visualisierungen

interessiert sind, verweisen wir auf die Abbildungen 21 bis 28 zwischen den Seiten 132 und 146. Wir analysieren die Arbeit von Chapman (1984) sehr ausführlich, die Arbeiten von LaLonde (1977, 1979), Purdom und Brown (1981), Nakata und Sassa (1986), Shin und Choe (1993), Fortes Gálvez (1994) und Lee und Kim (1997) – alle ebenfalls für RRPGen – dann zügiger, gehen in dieser Einleitung jedoch von diesen nur noch auf die systematisch interessanteren von LaLonde und Chapman ein, auch zumal die anderen genannten Arbeiten größtenteils starke formale Schwächen aufweisen. Schließlich erweist sich noch die gute frühere Arbeit von Madsen und Kristensen (1976), die ekfGen mit regulären Ausdrücken verwendet, als lohnenswert für eine detailliertere Diskussion.

Unsere Analyse erarbeitet die folgenden Hauptergebnisse:

- Bis heute ist in der Literatur die Behauptung unwidersprochen, daß die den Readback-Automaten zugestandene Tiefe m des Lookbacks in den Keller hinein für festgehaltenes k eine Hierarchie von LR(m,k)-Grammatiken (LaLonde 1977, 1979) bzw. LALR(m,k)-Grammatiken (Chapman 1984) induzieren – und natürlich vermuten die Autoren eine *echte* Hierarchie. Wir arbeiten heraus, daß eine echte solche Hierarchie für $m \geq 1$ nicht existiert:[4] Ebensowenig nützt es einem klassischen LR-Parser für eine kfG, noch tieferen als den Kellerkontext um das Handle zu berücksichtigen! Statt dessen macht es nur Sinn, von den Klassen ELR(k), ELALR(k) etc. zu sprechen.

- Auf rechten Regelseiten in RRPGen mehrdeutige NEAs (bzw. mehrdeutige reguläre Ausdrücke) zuzulassen, ist nicht nur problematisch, wenn es darum geht, solche ekfGen mit semantischem Code anzureichern; es sorgt auch für Schwierigkeiten in der Reduktions-Maschinerie, wenn diese ohne irgendeine Potenzkonstruktion *direkt* auf der Grundlage der originalen, möglicherweise mehrdeutigen NEAs konstruiert wird. So ist beispielsweise die Determinismuseigenschaft der von Chapman angegebenen LALR(1,1)-Parser abhängig von der *Struktur* der NEAs auf den rechten Regelseiten, während sein Rechtsableitungs- und ELR(k)-Begriff aber von deren Struktur abstrahieren. Und wir zeigen, daß RRPGen existieren, die nach Chapman ELR(0)-Grammatiken sind, für die sich aber sein Parserkonstruktionsverfahren offensichtlich als unzureichend herausstellt, da sie aufgrund mehrdeutiger Readback-Automaten keine LALR(1,1)-Grammatiken sind. Auch keiner der anderen Autoren hat bemerkt, daß diese Nichtdeterminismen in Readback-Automaten letztlich nur Folge *mehrdeutiger* NEAs auf den rechten Regelseiten ist, obwohl schon Heilbrun-

[4] Der Fall $m = 0$ ist uninteressant.

ner (1979) die für ein korrektes Parserkonstruktionsverfahren grundlegende Theorie zur Verfügung stellt.

- Die von LaLonde und Chapman verwendeten Readback-Automaten *sind* keine deterministischen endlichen Automaten (DEAs), sondern ähneln diesen nur stark. Sie lesen – anders als DEAs – nur ein stückweit in ihre Eingabe hinein, statt sie ganz zu konsumieren. Insbesondere stellt es einen Nichtdeterminismus des Parsers dar, wenn einer seiner Readback-Automaten zugleich in einen Endzustand geraten *und* weiter in den Keller hinein lesen kann.

- Kein einziger aller Autoren, die sich um LR-Parserkonstruktion für ekfGen bemühen, arbeitet überzeugend heraus, daß der konstruierte Parser formal als deterministischer Kellertransduktor angegeben werden kann. Die Verbindung mit der Automatentheorie wird damit erst von der vorliegenden Arbeit vollzogen.

- Wir stellen fest, daß *korrekte* ELR(k)- bzw. ELALR(k)-Parser im Sinne von LaLonde bzw. Chapman konstruiert werden können, indem die (möglicherweise mehrdeutigen) NEAs auf rechten Regelseiten durch deterministische (mithin eindeutige) Pendants ersetzt, diese zu (eindeutigen) rechtslinearen kontextfreien Subgrammatiken transliteriert werden und anschließend ein klassischer kanonischer (oder auch allgemeiner) LR(k)- bzw. LALR(k)-Parser für die entstandene kfG konstruiert wird. Dieses Verfahren ist theoretisch fundiert, während in der Literatur unreflektiert schon mit der verbreiteten Itemmengen-Konstruktion für RRPGen das Terrain des theoretisch Fundierten immer wieder unbemerkt verlassen wurde.

- Die frühe Arbeit von Madsen und Kristensen (1976), die ELR(k)-Parser für ekfGen mit regulären Ausdrücken konstruiert, ist lobenswert, weil sie theoretisch über weite Strecken akkurat ist und sich auf stark eindeutige reguläre Ausdrücke beschränkt. (Bemerkenswert ist dabei, daß ihre Arbeit zeitlich weit vor der von Brüggemann-Klein (1993) angegebenen Quelle (Sippu und Soisalon-Soininen 1988) für das Konzept der starken Eindeutigkeit regulärer Ausdrücke datiert.) Im Vergleich zu unserer eigenen Vorgehensweise gerät jedoch ihr Weg, die theoretische Fundierung ihrer ELR(k)-Parser zu leisten, merklich umständlicher: Ihre Itemmengen-Konstruktion, die sie direkt auf der Basis von regulären Ausdrücken und mit *zwei* Sorten von Arbeitspunkten formulieren, ist im formalen Umgang sperrig und, was die zweite Sorte Arbeitspunkt betrifft, fragwürdig. Und schließlich scheinen, so wie sie beschrieben sind, die rekursiven Prozeduren, die in ihren Parsern die Reduktionen abwickeln und

die induktive Struktur der Handles rekonstruieren, im Ablauf ineffizienter als unsere Readback-Automaten; und ob sich damit ihre Parser formal als Kellertransduktoren formulieren lassen, bleibt unbeantwortet. Anders als bei unserem eigenen Verfahren ist hier der Bezug zur Automatentheorie unklar. Angesichts der insgesamt aber soliden Qualität ihrer Arbeit stimmt es nachdenklich, wie wenig Aufmerksamkeit ihr in der Literatur zukommt; die wenigsten Autoren scheinen ihre Kernaussagen verstanden zu haben.

Die vorliegende Arbeit schließt mit ihrem zweiten Teil eine seit Jahrzehnten bestehende Lücke zwischen Theorie und Praxis. Es ist zu hoffen, daß die hier gelegten Grundlagen und erfolgten Analysen endlich dem theoretisch fundierten Einsatz von LR-Parsern für ekfGen in der Compilerbau-Praxis neuen Vorschub leisten können.

Auch für den Bereich der kfGen bietet die vorliegende Arbeit in ihrem ersten Teil mit den dortigen Ausführungen zur Redundanz von LR-Parsern eine Reihe von neuen Anregungen und Erkenntnissen theoretischer wie praktischer Natur für die Konstruktion von LR-Parsern. Inwieweit diese Ergebnisse aufgegriffen werden, wird sich zeigen müssen.

2 Grundlagen

Der erste Band (1988) der zweibändigen Monographie von Sippu und Soisalon-Soininen über Parsertheorie formuliert elegant Grammatiken, endliche Automaten und Kellerautomaten als Spezialfälle von Ersetzungssystemen und stellt ein ausführliches und abgeschlossenes Rahmenwerk an nützlicher Notation, Terminologie und theoretischen Grundlagen zur Verfügung, das wir hier als Standard ansehen wollen. Wir präsentieren unten essentielle Formalisierungen und Begriffsdefinitionen, die weitgehend konform mit (Sippu und Soisalon-Soininen 1988) sind. Im Zusammenhang mit dem LR(k)-Begriff, dem sich der zweite Band (1990) der Monographie widmet, werden wir speziellere Terminologie ausführlicher wiederholen und wesentliche Verfahrensschritte rekonstruieren. Allgemein geben wir Quantorisierungen meist explizit an, sie sind ansonsten offensichtlich. Maschinen und ihre Komponenten kennzeichnen wir einheitlich gemäß ihrem Typ; so wird ein nichtdeterministischer endlicher Automat etwa mit M bezeichnet, ein deterministisches Pendant mit $\hat{M}$ und ein zugehöriger Kellerautomat mit $\tilde{M}$.

Zu mathematischen Präliminarien und formalen Sprachen wird auf die Abschnitte 1.1–1.2 von (Sippu und Soisalon-Soininen 1988) verwiesen. Hier sei nur erwähnt, daß wir unter einem *Alphabet* eine endliche, nichtleere Menge verstehen und daß ε in der Menge V^* der Wörter über einem Alphabet V stets das leere Wort bezeichnet. Sei ferner $x \in V^*$ (der Länge $|x|$). Dann ist x^{R} die Umkehrung von x, und für ein festes $k \geq 0$ bezeichnet $k:x$ das Präfix und $x:k$ das Suffix der Länge $\min\{k,|x|\}$ von x. Verallgemeinert ist mit $L \subseteq V^*$ dann $k:L = \{(k:x) : x \in L\}$, $L:k = \{(x:k) : x \in L\}$ und $L^{\mathrm{R}} = \{x^{\mathrm{R}} : x \in L\}$.

Ersetzungssysteme

Ein *Ersetzungssystem* ist ein Paar $G = (V,P)$, in dem V ein Alphabet und P eine endliche Relation auf V^* ist. Ein Paar $(\omega_1,\omega_2) \in P$ wird *Regel* (oder *Produktion* oder, im Zusammenhang mit Maschinen, auch *Aktion* oder *Übergang)* genannt und als $\omega_1 \to \omega_2$ notiert. Es wird ω_1 *linke Seite* und ω_2 *rechte Seite* der Regel genannt.

Auf V^* werden *Ableitungsrelationen* $\underset{G}{\overset{r}{\Rightarrow}}$ (oder $\overset{r}{\Rightarrow}$ „in G“ oder kurz $\overset{r}{\Rightarrow}$) für jede Regel $r \in P$, $\underset{G}{\overset{\pi}{\Rightarrow}}$ (oder $\overset{\pi}{\Rightarrow}$ „in G“ oder kurz $\overset{\pi}{\Rightarrow}$) für jede Regelfolge $\pi \in P^*$ und $\underset{G}{\Rightarrow}$ (oder $\Rightarrow$

„in G“ oder kurz $\Rightarrow$) eingeführt:

$$\begin{aligned}
&\overset{r}{\underset{G}{\Longrightarrow}} = \{(\alpha\omega_1\beta, \alpha\omega_2\beta) : \alpha, \beta \in V^*\}\,,\\
&\overset{\varepsilon}{\underset{G}{\Longrightarrow}} = \{(\gamma, \gamma) : \gamma \in V^*\}\,,\\
&\overset{\pi}{\underset{G}{\Longrightarrow}} = \overset{r}{\underset{G}{\Longrightarrow}}\overset{\pi'}{\underset{G}{\Longrightarrow}} \quad \text{für } \pi = r\pi' \text{ mit } r \in P\,,\\
&\overset{}{\underset{G}{\Longrightarrow}} = \bigcup_{r \in P} \overset{r}{\underset{G}{\Longrightarrow}}\,.
\end{aligned}$$

Die *Größe* von G ist

$$|G| = \max\{\sum_{(\omega_1,\omega_2)\in P} |\omega_1\omega_2|, |V|\}\,.$$

Endliche Automaten

Sei $M = (V, P)$ ein Ersetzungssystem, und seien Q und T disjunkte, nichtleere Mengen, die $V = Q \cup T$ erfüllen. Seien ferner $q_s \in Q$ und $F \subseteq Q$. Dann ist M ein *endlicher Automat* mit *Zustandsalphabet* Q, *Eingabealphabet* T, *Anfangszustand* q_s und *Endzustandsmenge* F, notiert als $M = (Q, T, P, q_s, F)$, wenn jede Regel in P von der Form

$$q_1 x \to q_2$$

ist mit $q_1, q_2 \in Q$ und $x \in T^*$. M ist *in Normalform*[5] (resp. *ε-frei),* wenn die linken Seiten aller Regeln von M aus $Q(T \cup \{\varepsilon\})$ (resp. QT) sind; ein ε-freier, normierter endlicher Automat ist *buchstabierend.* Eine *Konfiguration* von M ist ein Element aus QT^*; für $q \in Q$ und $w \in T^*$ ist qw eine *Startkonfiguration* von M, falls $q = q_s$, und eine *Endkonfiguration* von M, falls $q \in F$ und $w = \varepsilon$ ist. $(\gamma_0, \gamma_1, \ldots, \gamma_n)$, $n \geq 0$, ist eine *Berechnung von M auf w,* falls $\gamma_0 = q_s w$ ist und

$$\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n \quad \text{in } M$$

gilt, und eine *akzeptierende* Berechnung von M auf w, falls zusätzlich γ_n eine Endkonfiguration ist – wir sagen dann, *M akzeptiert w*. Die *von M akzeptierte Sprache* ist

$$L(M) = \{w \in T^* : q_s w \Rightarrow^* q \text{ in } M \text{ für ein } q \in F\}\,.$$

[5] Alle in dieser Arbeit vorkommenden endlichen Automaten sind in Normalform, worauf in der Folge nicht mehr hingewiesen wird.

Konfigurationen und die darin vorkommenden Zustände nennen wir auch *erreichbar,* wenn sie in irgendwelchen Berechnungen von M auftreten, und ansonsten *unerreichbar.* Einen Zustand q, für den $\Rightarrow^*(qT^*) \cap F = \emptyset$ ist, mit dem also keine Konfiguration über $\Rightarrow^*$ eine Endkonfiguration erreichen kann, nennen wir *tot.*

M ist *mehrdeutig,* falls es eine Eingabe mit zwei verschiedenen sie akzeptierenden Berechnungen gibt; anderenfalls ist M *eindeutig.* M ist *nichtdeterministisch,* falls es eine Konfiguration von M gibt, auf die zwei verschiedene Regeln von M anwendbar sind; anderenfalls ist M *deterministisch.* Äquivalent ist M genau dann nichtdeterministisch, wenn es in M zwei verschiedene Regeln $qx \to q_1$ und $qy \to q_2$ gibt, so daß y Präfix von x ist. Jeder deterministische endliche Automat (kurz DEA) ist eindeutig, vorausgesetzt, er besitzt keine ε-Übergänge aus Endzuständen. Ist M deterministisch, so existiert zu allen $q \in Q$ und $w \in T^*$ höchstens ein $q' \in Q$, so daß

$$qw \Rightarrow^* q' \quad \text{in } M$$

gilt. Und in diesem Falle definieren wir $\mathrm{GOTO}_M(q,w) = q'$.

Einen buchstabierenden DEA M, der für jedes $q \in Q$ und $a \in T$ eine Aktion $qa \to q_a$ aufweist, nennen wir *vollständig spezifiziert.*

Die *Größe* des endlichen Automaten M ist seine Größe als Ersetzungssystem (V,P) oder (in pathologischen Fällen) $|F|$, falls dies größer ist.

Kontextfreie Grammatiken

Seien $G = (V,P)$ ein Ersetzungssystem, $T \subset V$ und $S \in V \setminus T = N$. G ist eine *kontextfreie Grammatik* (kurz kfG oder einfach *Grammatik*) mit *Nichtterminalen* N, *Terminalen* T und *Startsymbol* S, notiert als $G = (V,T,P,S)$, wenn $P \subseteq N \times V^*$ ist. Im Zusammenhang mit Grammatiken benutzen wir Symbole $A,B,C,\ldots \in N$ sowie $a,b,c,\ldots \in T$, ferner $\alpha,\beta,\gamma,\ldots \in V^*$ und weiter $u,v,\ldots,z \in T^*$ sowie $X,Y,Z \in V \cup \{\varepsilon\}$. Die *von G erzeugte Sprache* ist $L(G) = \Rightarrow^*(S) \cap T^* = \{w \in T^* : S \Rightarrow^* w\}$. $\gamma \in \Rightarrow^*(S)$ wird *Satzform,* $w \in L(G)$ wird *Satz* von G genannt. Eine Sprache $L_1 \subseteq T^*$ ist *kontextfrei,* wenn sie von einer kontextfreien Grammatik erzeugt wird.

Für jede Regel $r = A \to \omega$ von G ist die spezielle Ableitungsrelation $\underset{G,\mathrm{rm}}{\overset{r}{\Longrightarrow}}$ (oder $\underset{\mathrm{rm}}{\overset{r}{\Longrightarrow}}$ „in G“ oder kurz $\underset{\mathrm{rm}}{\overset{r}{\Longrightarrow}}$) auf V^* gegeben durch

$$\underset{G,\mathrm{rm}}{\overset{r}{\Longrightarrow}} = \{(\alpha Ax, \alpha\omega x) : \alpha \in V^*, x \in T^*,\}\ .$$

Weiter definieren wir auf V^* die speziellen Ableitungsrelationen $\underset{G,\mathrm{rm}}{\Longrightarrow}$ (oder $\underset{\mathrm{rm}}{\Rightarrow}$ „in G“ oder kurz $\underset{\mathrm{rm}}{\Rightarrow}$) und $\underset{G,\mathrm{rm}}{\overset{\pi}{\Longrightarrow}}$ (oder $\underset{\mathrm{rm}}{\overset{\pi}{\Rightarrow}}$ „in G“ oder kurz $\underset{\mathrm{rm}}{\overset{\pi}{\Rightarrow}}$) für jede Regelfolge $\pi \in P^*$ durch

$$\begin{aligned}
&\underset{G,\mathrm{rm}}{\Longrightarrow} = \bigcup_{r \in P} \underset{G,\mathrm{rm}}{\overset{r}{\Longrightarrow}} , \\
&\underset{G,\mathrm{rm}}{\overset{\varepsilon}{\Longrightarrow}} = \{(\gamma,\gamma) : \gamma \in V^*\} , \\
&\underset{G,\mathrm{rm}}{\overset{\pi}{\Longrightarrow}} = \underset{G,\mathrm{rm}}{\overset{r}{\Longrightarrow}} \underset{G,\mathrm{rm}}{\overset{\pi'}{\Longrightarrow}} \quad \text{für } \pi = r\pi' \text{ mit } r \in P .
\end{aligned}$$

$\underset{G,\mathrm{rm}}{\overset{r}{\Longrightarrow}}$, $\underset{G,\mathrm{rm}}{\Longrightarrow}$ und $\underset{G,\mathrm{rm}}{\overset{\pi}{\Longrightarrow}}$ sind *Rechtsableitungsrelationen.* Elemente aus $\underset{G,\mathrm{rm}}{\Longrightarrow}^*(S)$ heißen *Rechtssatzformen.* $(\gamma_0,\gamma_1,\ldots,\gamma_n)$, $n \geq 0$, ist eine *Ableitung von γ_n aus γ_0 (in G) (der Länge n),* falls

$$\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n \quad \text{in } G$$

gilt, und eine *Rechtsableitung von γ_n aus γ_0 (in G) (der Länge n),* falls

$$\gamma_0 \underset{\mathrm{rm}}{\Rightarrow} \gamma_1 \underset{\mathrm{rm}}{\Rightarrow} \cdots \underset{\mathrm{rm}}{\Rightarrow} \gamma_n \quad \text{in } G$$

gilt, und im Falle $\gamma_0 = S$ nennen wir $(\gamma_0,\gamma_1,\ldots,\gamma_n)$ eine *Ableitung von γ_n (in G)* resp. eine *Rechtsableitung von γ_n (in G).*

G ist *mehrdeutig,* falls für ein $w \in L(G)$ mehr als eine Rechtsableitung von w in G existiert; anderenfalls ist G *eindeutig.*

Ein Symbol $X \in V$ ist *nützlich,* falls $X = S$ ist oder $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$ in G für irgendwelche $w \in T^*$ und $\alpha,\beta \in V^*$ gilt; anderenfalls ist X *nutzlos.* G wird *reduziert* genannt, falls V keine nutzlosen Symbole enthält. $A \in N$ (resp. $\alpha \in V^*$) ist *leerableitbar,* falls $A \Rightarrow^+ \varepsilon$ (resp. $\alpha \Rightarrow^* \varepsilon$) in G gilt.

Für ein $k \geq 0$ und ein $\gamma \in V^*$ ist

$$\begin{aligned}
\mathrm{FIRST}_{G,k}(\gamma) &= k : (\Rightarrow^*(\gamma) \cap T^*) \quad \text{und} \\
\mathrm{FOLLOW}_{G,k}(\gamma) &= \{y \in T^* : S \Rightarrow^* \alpha\gamma\beta \text{ in } G \text{ und } y \in \mathrm{FIRST}_{G,k}(\beta) \text{ für} \\
&\qquad\qquad \text{irgendwelche } \alpha,\beta \in V^* \} ,
\end{aligned}$$

wobei wir, wo möglich, $\mathrm{FIRST}_{G,k}$ zu FIRST_k und $\mathrm{FOLLOW}_{G,k}$ zu FOLLOW_k abkürzen werden.

Die *Größe* der Grammatik G ist ihre Größe als Ersetzungssystem (V,P).

Kellerautomaten

Sei $M = (V, P)$ ein Ersetzungssystem, so daß $V = Q \cup T \cup \{\$, \mid\}$ ist, $Q, T \subseteq V$ sind und $\$$ und $\mid$ zwei verschiedene, nicht zu $Q \cup T$ gehörende Symbole sind. Seien weiter $\gamma_s \in Q^*$ und $F \subset Q^*$ endlich. Dann ist M ein *Kellerautomat*[6] *mit Kelleralphabet Q, Eingabealphabet T, initialem Kellerinhalt γ_s, finalen Kellerinhalten F, Endesymbol $\$$ und Trennsymbol $\mid$*, notiert als $M = (Q, T, P, \gamma_s, F, \$, \mid)$, wenn jede Regel aus P von der Form

$$\delta\alpha \mid xy \to \delta\beta \mid y$$

ist für irgendwelche $x \in T^*$, $y \in T^*\{\varepsilon, \$\}$, $\alpha, \beta \in Q^*$ und $\delta \in \{\varepsilon, \$\}$.[7]

Eine *Konfiguration* von M ist eine Zeichenkette von der Form

$$\$\gamma \mid w\$$$

mit dem *Kellerinhalt* $\gamma \in Q^*$ und der *Resteingabe* $w \in T^*$. $\$\gamma : 1$ wird *oberstes Kellersymbol,* $1 : w\$$ wird *aktuelles Eingabesymbol* genannt. $\$\gamma \mid w$ ist eine (bzw. die) *Startkonfiguration (für w),* falls $\gamma = \gamma_s$, und eine *Endkonfiguration,* falls $w = \varepsilon$ und $\gamma \in F$ ist. $(\gamma_0, \gamma_1, \ldots, \gamma_n)$, $n \geq 0$, mit γ_0 Startkonfiguration für w, so daß

$$\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n \quad \text{in } M$$

gilt, ist eine *Berechnung von M auf w,* die zudem *akzeptierend* heißt, wenn γ_n Endkonfiguration ist; in diesem Falle sagen wir auch, *M akzeptiert w*. Die *von M akzeptierte Sprache* $L(M)$ ist die Menge der akzeptierten Eingaben. Konfigurationen, die in irgendwelchen Berechnungen von M auftreten, nennen wir auch *erreichbar;* sonstige sind *unerreichbar.* Eine Aktion, die auf irgendeine erreichbare Konfiguration anwendbar ist, bezeichnen wir auch als *nützlich,* sonstige als *nutzlos.*

M ist *mehrdeutig,* wenn es auf irgendeiner Eingabe mehrere akzeptierende Berechnungen von M gibt; anderenfalls ist M *eindeutig. M* ist *nichtdeterministisch,* falls es eine Konfiguration von M gibt, auf die zwei verschiedene seiner Aktionen anwendbar sind; anderenfalls

[6] Lesern, denen der hier vorgestellte Typus von Kellerautomaten „zu großzügig“ vorkommt und die Bedenken hinsichtlich der effizienten Implementierbarkeit solcher Automaten hegen, sei ein Blick in den Anhangsabschnitt A empfohlen, in dem eine Implementierung ausführlich thematisiert wird und an dessen Beginn unter anderem die enge Verwandschaft des vorgestellten Kellerautomatenmodells zu einem schon von Aho und Ullman (1972) verwendeten erläutert wird.

[7] In der von Sippu und Soisalon-Soininen (1988) gewählten Formulierung kann der Kellerbegrenzer $\$$ verloren gehen.

ist M *deterministisch.*[8] Äquivalent ist M genau dann nichtdeterministisch, wenn es zwei verschiedene Aktionen $\alpha \mid x \to \alpha' \mid x'$ und $\beta \mid y \to \beta' \mid y'$ von M gibt, so daß einer von α, α' Suffix des anderen und einer von x, y Präfix des anderen ist. Jeder deterministische Kellerautomat ist eindeutig, vorausgesetzt, auf keine der Endkonfigurationen ist noch eine Aktion anwendbar. M ist *in Normalform,* wenn jede seiner Aktionen von der Form

$$\alpha \mid x \to \beta \mid \quad \text{mit } |\alpha| \leq 2 \text{ und } |x| \leq 1$$

ist. Jeder Kellerautomat M kann in einen Kellerautomaten M' in Normalform transformiert werden, der dieselbe Sprache akzeptiert und eindeutig (bzw. deterministisch) genau dann ist, wenn auch M eindeutig (bzw. deterministisch) ist. Eine (kontextfreie) Sprache ist *deterministisch,* wenn sie von einem deterministischen Kellerautomaten akzeptiert wird.

Die *Größe* des Kellerautomaten M ist seine Größe als Ersetzungssystem (V, P) oder (in pathologischen Fällen) $\sum_{\gamma \in F} |\gamma|$, falls dies größer ist.

Kellertransduktoren

M ist ein *Kellertransduktor mit Ausgabealphabet* Δ *und Ausgabekomponente* τ, geschrieben (M, τ), wenn M ein Kellerautomat und τ ein Homomorphismus (bzgl. Konkatenation) von P^* nach Δ^* ist, wobei P die Aktionen von M sind. Akzeptiert M eine Eingabe w mit einer Berechnung, in der π' die Folge der dabei von M angewendeten Aktionen ist, so *produziert* (M, τ) *die Ausgabe* π *auf* w, wenn $\pi = \tau(\pi')$ ist.

Als die Größe eines Kellertransduktors wird die Größe seines Kellerautomaten angesehen. Dessen Größe wiederum, wie auch die eines endlichen Automaten und die einer Grammatik, ist die Größe des unterlegten Ersetzungssystems.

2.1 Heilbrunners Ansatz zur LR-Theorie

> *„It is not a simple task to give a convincing explanation of the commonly used LR(k) definition."*

[8] Es sei angemerkt, daß gemäß dieser Definition nur solche Situationen, in denen der Kellerautomat mehrere Aktionen gleichzeitig anwenden kann, als Nichtdeterminismus angesehen werden, nicht jedoch solche, in denen er terminieren *kann,* aber nicht *muß,* d. h. Endkonfigurationen, auf die noch Aktionen anwendbar sind. (Eine entsprechende Bemerkung gilt auch für endliche Automaten.)

So die Einschätzung von Heilbrunner (1981), der eine andere Sicht auf den LR(k)-Begriff erarbeitet als die traditionelle, von Aho und Ullman (1972) geprägte: In Heilbrunners eigenen Worten

> *„a grammar is an LR(k) grammar if and only if the straightforward nondeterministic bottom-up parsing algorithm for this grammar can be made deterministic by eliminating superflous moves.“*

Diese Idee ist sehr intuitiv und gefällig, da damit auch einem Noch-nicht-Kenner von LR-Theorie bereits ein Gefühl dafür vermittelt werden kann, worin im Kern das Problem besteht, einen deterministischen LR(k)-Parser zu konstruieren. Heilbrunner eliminiert jedoch leider in seiner Arbeit von vornherein konkrete Kellertransduktoren, für die ja gerade die *Endlichkeit* ihrer Aktionenmengen ein wesentliches Merkmal ist, indem er an ihrer statt eine *unendliche* und abstrakte Übergangsrelation einführt, die die Anwendungsmöglichkeiten der Aktionen charakterisiert:

> *„Considering a pushdown-automaton whose move relation essentially agrees with the relation $\Vdash$ defined by*
>
> $$(\alpha\beta, z, \pi) \Vdash (\alpha A, z, A \to \beta\pi) \quad \textit{whenever } A \to \beta \in \Pi,$$
> $$(\psi, az, \pi) \Vdash (\psi a, z, \pi) \quad \textit{for all } a \in \Sigma,$$
>
> *is one of the two usual ways of proving that context-free languages can be accepted by pushdown-automata. [...] The relation $\Vdash$ describes the behaviour of a nondeterministic machine. This machine has to be modified if it is to be used for a practical purpose. First we try to make the machine deterministic by allowing fewer moves, which means that we have to look for subsets $\vdash$ of $\Vdash$ which are partial functions. Next, the machine is simplified by permitting only k symbols of lookahead for some fixed $k \geq 0$ [...].“*

Hauptsächlich zu kritisieren an Heilbrunners Präsentation der LR-Theorie ist dann auch, daß sie sehr mathematiklastig und die Verbindung zur Implementierung zu schwach ist:

(i) Die von Heilbrunner eliminierten Kellertransduktoren sind *das* formale Maschinenmodell für kontextfreie Syntaxanalyse mit der LR-Methode; sie durch abstrakte Übergangsrelationen zu ersetzen schafft eine Lücke zur Implementierung.

(ii) Der entscheidende Abstraktionsschritt weg von Kellertransduktoren ist nur informell skizziert *(„essentially agrees with … “).*

(iii) Heilbrunner führt dann auch keine einzige konkrete endliche Aktionenmenge einer Beispielmaschine vor, die einen LR(k)-Parser darstellte, sondern bleibt bei einer abstrakten Maschinenbeschreibung mittels Parsing-Relationen stehen. Die Lücke zur Implementierung von LR-Parsern wird nicht geschlossen.

Wir wollen *nicht* von konkreten Kellertransduktoren in der LR-Theorie abstrahieren, uns aber dennoch auf die Essenz der Arbeit von Heilbrunner (1981) berufen können. Anstatt wie Heilbrunner direkt spezielle Übergangsrelationen $\vdash \subseteq \Vdash$ zu studieren, werden wir daher im folgenden Abschnitt 2.2 zunächst allgemein definieren, wann ein beliebiger Kellertransduktor „sich verhält wie“ ein anderer gleichen Ein- und Ausgabealphabets. Und konkret hat dann der in Abschnitt 2.3 explizit eingeführte bekannte (i. a. nichtdeterministische) „Shift-reduce-Parser“ zur betrachteten Grammatik – soweit wir Heilbrunners Worte interpretieren – die abstrakte Übergangsrelation $\Vdash$, und jedem Kellertransduktor, der sich wie dieser Shift-reduce-Parser verhält, könnten wir eine abstrakte Übergangsrelation $\vdash \subseteq \Vdash$ im Sinne von Heilbrunner zuordnen. Entsprechende Definitionen von „LR(k)-Parsern“ und „LR(k)-Grammatiken“ sind dann ein geradliniger weiterer Schritt.

2.2 Verhaltensgleiche Kellertransduktoren

Ziel dieses Abschnitts ist es, einen Begriff zu prägen, der zu zwei (auch nichtdeterministischen) Kellertransduktoren $(\tilde{M},\tau)$ und $(\tilde{M}',\tau')$ analoges beobachtbares Verhalten beschreibt.

Seien $M = (Q,T,P,\gamma_s,F,\$,\mid)$, $\tau : P^* \to \Delta^*$, $M' = (Q',T',P',\gamma'_s,F',\$,\mid)$ und schließlich $\tau' : P'^* \to \Delta'^*$. Wir wollen allgemein sagen, $(\tilde{M},\tau)$ *verhält sich wie* $(\tilde{M}',\tau')$, wenn jeweils Ein- und Ausgabealphabete identisch sind (d. h. $T = T'$, $\Delta = \Delta'$) und ferner zu jeder Berechnung

$$\$\gamma_s \mid z\$ \overset{\pi}{\Longrightarrow} \$\psi \mid x\$ \quad \text{in } \tilde{M}$$

eine Berechnung

$$\$\gamma'_s \mid z\$ \overset{\pi'}{\Longrightarrow} \$\psi' \mid x\$ \quad \text{in } \tilde{M}' \quad \text{mit } |\pi'| = |\pi| \quad \text{und} \quad \tau'(\pi') = \tau(\pi)$$

existiert und dabei $\$\psi' \mid x\$$ Endkonfiguration von $\tilde{M}'$ ist, wenn $\$\psi \mid x\$$ Endkonfiguration von $\tilde{M}$ ist. Man beachte insbesondere, daß die konkreten Kellerinhalte in dieser Definition irrelevant sind.

Sofort einzusehen ist das

Lemma 2.1
Verhält sich $(\tilde{M},\tau)$ wie $(\tilde{M}',\tau')$, so gilt $L(\tilde{M}) \subseteq L(\tilde{M}')$, und jede mögliche Ausgabe, die $\tilde{M}$ auf einer Eingabe z erzeugt, wird auch von $\tilde{M}'$ auf z erzeugt. □

Verhält sich $(\tilde{M},\tau)$ wie $(\tilde{M}',\tau')$ und gilt sogar $L(\tilde{M}) = L(\tilde{M}')$ und $\tilde{M}$ und $\tilde{M}'$ produzieren auf gleichen Eingaben gleiche Ausgaben, so sagen wir, $(\tilde{M},\tau)$ *verhält sich stark wie* $(\tilde{M}',\tau')$.[9]

Die Relationen „verhält sich wie“ und „verhält sich stark wie“ sind reflexiv und transitiv, jedoch i. a. weder symmetrisch noch antisymmetrisch. (Auch nichtterminierende Berechnungen sind zu beachten!) Aber natürlich gilt das

Lemma 2.2
Verhält sich $(\tilde{M},\tau)$ wie $(\tilde{M}',\tau')$ und umgekehrt, so verhält sich $(\tilde{M},\tau)$ stark wie $(\tilde{M}',\tau')$ und umgekehrt. □

Als ein hinreichendes Kriterium ist leicht induktiv einzusehen, daß sich $(\tilde{M},\tau)$ wie $(\tilde{M}',\tau')$ verhält, wenn mit Hilfe einer konkreten totalen („Kellerumrechnungs“-)Funktion *explain* die erreichbaren Konfigurationen $\$\varphi \mid z\$$ von $\tilde{M}$ in Konfigurationen von $\tilde{M}'$ so abbildet werden können, daß gilt:

1. (Startkonfigurationen)
   Seien γ_s, γ_s' die initialen Kellerinhalte von $\tilde{M}$ resp. $\tilde{M}'$. Ist $\varphi = \gamma_s$, so gilt

   $$explain(\$\varphi \mid z\$) = explain(\$\gamma_s \mid z\$) = \$\gamma_s' \mid z\$ \,.$$

2. (Übergänge)
   Ist r eine Aktion von $\tilde{M}$, so daß

   $$\$\varphi \mid z\$ \overset{r}{\Longrightarrow} \$\psi \mid x\$ \quad \text{in } \tilde{M}$$

   erfüllt ist, so existiert in $\tilde{M}'$ eine Aktion r', so daß gilt:

   $$explain(\$\varphi \mid z\$) = \$\varphi' \mid z\$ \overset{r'}{\Longrightarrow} \$\psi' \mid x\$ = explain(\$\psi \mid x\$) \quad \text{in } \tilde{M}'$$
   $$\text{und} \quad \tau(r) = \tau'(r') \,.$$

[9] Es sei darauf hingewiesen, daß nach Definition nur bei *terminierenden* Berechnungen Ausgaben produziert werden.

3. (Endkonfigurationen)
   Ist $\$\varphi\,|\,z\,\$$ Endkonfiguration von $\tilde{M}$, so ist $explain(\$\varphi\,|\,z\,\$)$ Endkonfiguration von $\tilde{M}'$.

Die Funktion *explain* impliziert dabei eine enge Verwandschaft von nützlichen Aktionen von $\tilde{M}$ zu Aktionen von $\tilde{M}'$, während über nutzlose Aktionen von $\tilde{M}$ nichts ausgesagt wird: Punkt 2 garantiert für eine nützliche Aktion r von $\tilde{M}$ die Existenz einer (dann auch nützlichen) Aktion r' auf seiten von $\tilde{M}'$, von der zwar gesagt werden kann, daß sie dieselbe Wirkung auf die Resteingabe und dieselbe Ausgabe wie r zeigt, jedoch induziert deswegen *explain* nicht unbedingt eine Abbildung, sondern i. a. nur eine linkstotale Relation von den nützlichen Aktionen von $\tilde{M}$ in die (nützlichen) Aktionen von $\tilde{M}'$.

2.3 LR(k)-Grammatiken

Sei von nun an eine kontextfreie Grammatik (kfG) $G = (V, T, P, S)$ gegeben, wobei V keines der Symbole $\$, |, S'$ enthalte. Wir setzen voraus, daß G reduziert ist und $S \Rightarrow^+ S$ nicht erfüllt. Der *Shift-reduce-Parser zu G* ist der Kellertransduktor $(\tilde{M}_{\mathrm{sr}}^G, \tau_{\mathrm{sr}}^G)$, kurz $(\tilde{M}_{\mathrm{sr}}, \tau_{\mathrm{sr}})$, mit dem Kellerautomaten $\tilde{M}_{\mathrm{sr}} = (V, T, shift \cup reduce, \varepsilon, \{S\}, \$, |)$ und der Ausgabekomponente τ_{sr}, die ein Homomorphismus (bzgl. Konkatenation) von $(shift \cup reduce)^*$ in P^* ist, wobei

$$\begin{aligned} shift &= \{\, |a \to a| \,:\, a \in T\,\}\,, & \tau_{\mathrm{sr}}(|a \to a|) &= \varepsilon\,, \\ reduce &= \{\, \omega| \to A| \,:\, A \to \omega \in P\,\}\,, & \tau_{\mathrm{sr}}(\omega| \to A|) &= A \to \omega\,. \end{aligned}$$

Sei $z \in T^*$. Gilt $S \underset{\mathrm{rm}}{\overset{\pi}{\Longrightarrow}} z$ in G, so ist π^{R} ein *Rechtsparse von z in G,* und einen Kellertransduktor mit Eingabealphabet T und Ausgabealphabet P, der genau die Rechtsparses seiner Eingaben produziert, nennen wir einen *Rechtsparser zu G.* Der Shift-reduce-Parser $(\tilde{M}_{\mathrm{sr}}, \tau_{\mathrm{sr}})$ zu G ist ein Rechtsparser zu G,[10] formal:

$$S \underset{\mathrm{rm}}{\overset{\pi}{\Longrightarrow}} z \quad \text{in } G \quad \Leftrightarrow \quad \$|z\$ \overset{\pi'}{\Longrightarrow} \$S|\$ \;\wedge\; \tau_{\mathrm{sr}}(\pi') = \pi^{\mathrm{R}} \quad \text{in } \tilde{M}_{\mathrm{sr}} \text{ für ein } \pi'\,. \tag{2}$$

Allerdings ist $\tilde{M}_{\mathrm{sr}}$ i. a. „sehr“ nichtdeterministisch.

Wir beobachten, daß, gegeben ein Kellertransduktor $(\tilde{M}, \tau)$, der sich wie $(\tilde{M}_{\mathrm{sr}}, \tau_{\mathrm{sr}})$ verhält, jede Berechnung von $(\tilde{M}, \tau)$ auf einer Eingabe z stets „gedeutet“ werden kann als eine ausgabeidentische Berechnung von $(\tilde{M}_{\mathrm{sr}}, \tau_{\mathrm{sr}})$ auf z. Insbesondere läßt sich jede nützliche Aktion r von $\tilde{M}$ über ihre Wirkung auf die Resteingabe und über die Ausgabe $\tau(r)$ als eine

[10] Z. B. zeigt dies Theorem 5.21 in (Sippu und Soisalon-Soininen 1988); man beachte jedoch, daß der Terminus „Rechtsparser“ dort etwas anders definiert ist.

„Reduce-Aktion nach einer Regel von G“ oder eine „Shift-Aktion eines Terminalsymbols“ klassifizieren.

Ein Kellertransduktor $(\tilde{M}, \tau)$, der sich stark wie $(\tilde{M}_{\mathrm{sr}}, \tau_{\mathrm{sr}})$ verhält und der nicht mehr als k Eingabesymbole an Lookahead verwendet, da seine Reduce-Aktionen $\varphi \mid x \to \varphi' \mid x$ stets $|x| \leq k$ und seine Shift-Aktionen $\psi \mid ay \to \psi' \mid y$ stets $|y| \leq \max\{k \dot{-} 1, 0\}$ erfüllen, heißt *LR(k)-Parser zu G.* Und G heißt *LR(k)-Grammatik,* wenn ein deterministischer LR(k)-Parser zu G existiert (und $S \Rightarrow^+ S$ nicht möglich ist).[11] Man beachte, daß $(\tilde{M}_{\mathrm{sr}}, \tau_{\mathrm{sr}})$ ein (i. a. nichtdeterministischer) LR(0)-Parser zu G ist und daß jeder Kellertransduktor, der sich stark wie $(\tilde{M}_{\mathrm{sr}}, \tau_{\mathrm{sr}})$ verhält, ein Rechtsparser zu G ist.

2.4 Kanonische LR(k)-Maschinen

Die Standardkonstruktion deterministischer LR(k)-Parser mit dem Ergebnis sogenannter „kanonischer LR(k)-Parser“ ist, wie wir im nächsten Abschnitt sehen werden, an sehr zentraler Stelle mit starken Redundanzen behaftet. Um gleichartige alternative (insbesondere redundanzärmere) LR(k)-Parser nachweislich korrekt zu konstruieren, werden die Eigenschaften solcher noch zu formulierender Kellertransduktoren *relativ* zu denen des Standardkonstruktionsverfahrens der kanonischen LR(k)-Parser untersucht. Dies erfordert, die Konstruktion von kanonischen LR(k)-Parsern hier sauber und adäquat zu präsentieren. Es wird sich dabei für die theoretische Arbeit als geschickt erweisen, kanonische LR(k)-Parser über die Vorstufen nichtdeterministischer und deterministischer LR(k)-Maschinen zu konstruieren. Die folgende Präsentation weicht daher auch ab von der Herangehensweise in (Sippu und Soisalon-Soininen 1990), wo nichtdeterministische LR(k)-Maschinen erst *a posteriori* eingeführt werden; sie kann als eine Verbesserung der Konstruktion von Heilbrunner (1981) im notationellen Stil von Sippu und Soisalon-Soininen (1990) charakterisiert werden.

G ist eine LR(k)-Grammatik genau dann, wenn sich der Shift-reduce-Parser zu G unter Einhaltung der k-Lookahead-Einschränkung „deterministisch machen“ läßt. Ein allgemeines Verfahren, beliebige Kellertransduktoren zu äquivalenten deterministischen umzuformen, kann nicht existieren. Wie also erreichen wir dieses mit einer auf das Problem spezialisierten Methode ? Der Schlüssel zur Konstruktion eines deterministischen LR(k)-Parsers

[11] In anderer Formalisierung zeigt dies (Heilbrunner 1981).
Ferner sei hier angemerkt, daß die zusätzliche Forderung, daß $S \Rightarrow^+ S$ in G nicht gelten darf, dazu dient zu verhindern, daß Grammatiken mehrdeutig sein und dennoch deterministische LR(k)-Parser aufweisen können. Man erinnere sich in diesem Zusammenhang an die Bemerkung aus Fußnote 8 auf Seite 24.

liegt bekanntermaßen in der Analyse derjenigen (sogenannten *lebensfähigen*) Kellerinhalte, die in tatsächlich terminierenden Berechnungen des Shift-reduce-Parsers auftreten können. Das Analogon auf der Seite von G ist wie folgt definiert: Es ist $\gamma \in V^*$ genau dann ein *lebensfähiges Präfix* von G, wenn

$$S \underset{\text{rm}}{\Longrightarrow}^* \delta A y \underset{\text{rm}}{\Longrightarrow} \delta\alpha\beta y = \gamma\beta y \quad \text{in } G$$

für irgendwelche $\delta \in V^*$, $y \in T^*$ und eine Regel $A \to \alpha\beta$ von G gilt; das lebensfähige Präfix γ heißt *vollständig*, wenn $\beta = \varepsilon$ ist, und in der Rechtssatzform $\delta\alpha\beta y$ heißt $\alpha\beta$ *Handle*. Die Sprache der lebensfähigen Präfizes einer kfG G bezeichnen wir mit $\text{VP}(G)$.[12] Jedes Präfix eines lebensfähigen Präfix von G ist selbst wiederum ein lebensfähiges Präfix von G.[13] Zwischen lebensfähigen Kellerinhalten von $(\tilde{M}_{\text{sr}}, \tau_{\text{sr}})$ und lebensfähigen Präfizes von G besteht der Zusammenhang:

> Jeder lebensfähige Kellerinhalt von $\tilde{M}_{\text{sr}}$ ist entweder S oder ein lebensfähiges Präfix von G. Umgekehrt ist jedes lebensfähige Präfix von G ein lebensfähiger Kellerinhalt von $\tilde{M}_{\text{sr}}$, falls (wie vorausgesetzt) G reduziert ist.[14]

Es erweist sich im folgenden als günstig, das Eingabebegrenzungssymbol \$ in der betrachteten kfG explizit zu machen: Mit $V' = V \cup \{\$, S'\}$, $T' = T \cup \{\$\}$ und $P' = P \cup \{S' \to S\$\}$ definieren wir $G' = (V', T', P', S')$ als die \$-*Erweiterung von* G.[15] Gleichzeitig wird in G' damit S nun zu einem lebensfähigen Präfix: Es ist $\text{VP}(G') = \text{VP}(G) \cup \{S, S\$\}$.

Glücklicherweise stellt sich nun heraus, daß die Sprache der lebensfähigen Präfizes einer kfG regulär ist. Prinzipiell kommt daher in den Sinn, mit Hilfe eines entsprechenden DEA die Lebensfähigkeit von Kellerinhalten in Berechnungen des Shift-reduce-Parsers sozusagen zu bewachen. Und ferner unterscheiden sich Kellerinhalte von in Berechnungen aufeinanderfolgenden Konfigurationen des Shift-reduce-Parsers immer nur geringfügig am rechten Kellerrand.[16] Es gelingt daher, die Bewachung der Lebensfähigkeit der Kellerinhalte durch die Aktionen des Shift-reduce-Parsers selbst vorzunehmen, indem durch deren

[12] Das Akronym VP bezieht sich auf die englische Bezeichnung *viable prefix.*

[13] (Sippu und Soisalon-Soininen 1990), Lemma 6.4.

[14] (Sippu und Soisalon-Soininen 1990), Theorem 6.7.

[15] Sippu und Soisalon-Soininen (1990) fügen statt dessen die Regel $S' \to \$S\$$ hinzu, was unnötige Verkomplizierungen zur Folge hat.

[16] Als alternatives theoretisches Fundament für LR(k)-Parser kann hier auch auf die regulären kanonischen Systeme von Büchi zurückgegriffen werden, wie Langmaack (1971) zeigt.

verfeinernden Umbau gewährleistet wird, daß auf dem Keller in das jeweilige lebensfähige Präfix die dafür durchlaufene Zustandsfolge des bewachenden DEA hineingefaltet wird.

Der so gewonnene Kellertransduktor ist jedoch erst ein LR(0)-Parser zu G. Er weist deswegen immer noch solche Nichtdeterminismen auf, die erst durch die präzise Berücksichtigung der zu den lebensfähigen Präfizes jeweils möglichen Lookaheads geeigneter Länge vermieden werden können (falls überhaupt G eine LR(k)-Grammatik ist). Das entsprechende Wissen steht bereit, wenn die Konstruktion des dem Parser zugrunde liegenden DEA – wie seit langem bekannt und üblich – über das Konzept des „LR(k)-gültigen Items" geschieht.

Ein *k-Item (zu G)* ist ein 4-Tupel (A,α,β,y) mit $A \to \alpha\beta \in P'$ sowie $y \in k : T'^*$ und wird als $[A \to \alpha\cdot\beta, y]$ notiert, wobei ε auch weggelassen wird – z. B. in $[A \to \alpha\cdot\varepsilon, y] = [A \to \alpha\cdot, y]$ oder $[A \to \varepsilon\cdot\beta, \varepsilon] = [A \to \cdot\beta]$. $I_{G,k}$ (kurz I_k) bezeichnet die Menge der k-Items zu G. Es ist $[A \to \alpha\cdot\beta, y]$ ein *LR(k)-gültiges Item für $\gamma \in V'^*$* gdw.

$$S' \underset{\text{rm}}{\Longrightarrow}{}^* \delta A z \underset{\text{rm}}{\Longrightarrow} \delta\alpha\beta z = \gamma\beta z \quad \text{in } G' \quad \text{und} \quad k : z = y$$

für irgendwelche $\delta \in V'^*$ und $z \in T'^*$ gilt. Die Menge der LR(k)-gültigen Items für γ wird mit $\text{VALID}^G_{\text{LR}(k)}(\gamma)$ (kurz $\text{VALID}_k(\gamma)$ oder nur $\text{VALID}(\gamma)$) bezeichnet. Ein k-Item $[A \to \alpha\cdot\beta, y]$, in dem $\alpha \neq \varepsilon$ ist, wird gelegentlich auch *Kernelitem* genannt.

In Anlehnung an Sippu und Soisalon-Soininen (1990) definieren wir auf den k-Items zu G die Relationen

$$[A \to \alpha\cdot B\beta, y]\ \mathbf{desc}^G_{\text{LR}(k)}\ [B \to \cdot\omega, z]\,, \quad \text{falls } z \in \text{FIRST}'_k(\beta y), \qquad \text{und}$$
$$[A \to \alpha\cdot X\beta, y]\ \mathbf{passes\text{-}}X\ [A \to \alpha X\cdot\beta, y]$$

für jedes $X \in V$ und kürzen im folgenden $\mathbf{desc}^G_{\text{LR}(k)}$ zu **desc** ab sowie, wie eben schon geschehen, $\text{FIRST}_{G',k}$ zu FIRST'_k.

Weiter konstruieren wir nun zunächst die *kanonische nichtdeterministische LR(k)-Maschine $M^G_{\text{c,LR}(k)}$ (kurz M_{c}) zu G*.[17] Diese ist ein i. a. nichtdeterministischer, nicht ε-freier endlicher Automat mit der Zustands- und Endzustandsmenge I_k, dem Startzustand $[S' \to \cdot S\$, \varepsilon]$, dem Eingabealphabet V' und den Aktionen $pass_{\text{c}} \cup desc_{\text{c}}$ mit

$$pass_{\text{c}} = \{qX \to q' : q\ \mathbf{passes\text{-}}X\ q' \quad \text{für ein } X \in V'\}\,,$$
$$desc_{\text{c}} = \{q \to q' : q\ \mathbf{desc}\ q'\}\,.$$

[17] Die hier angegebene kanonische nichtdeterministische LR(k)-Maschine ist praktisch die rechtslineare „Itemgrammatik" von Heilbrunner (1981) in Form des entsprechenden nichtdeterministischen endlichen Automaten. Für den konzisen Beweis der behaupteten Eigenschaften dieses Automaten siehe daher dort.

Das Übergangsverhalten dieses Automaten verknüpft prägnant die Begriffe „lebensfähiges Präfix“ und „LR(k)-gültiges Item“, denn für jedes $\gamma \in V'^*$ und jedes k-Item q gilt:

$$[S' \to \cdot S\$, \varepsilon]\gamma \Rightarrow^* q \quad \text{in } M_c \quad \Leftrightarrow \quad \begin{array}{l} q \text{ ist ein LR}(k)\text{-gültiges Item für } \gamma \\ \text{und } \gamma \in \mathrm{VP}(G') \ . \end{array}$$

Also akzeptiert der Automat genau die Sprache $\mathrm{VP}(G')$, und es ist

$$\mathrm{VALID}(\gamma) = \Rightarrow^*([S' \to \cdot S\$, \varepsilon]\gamma) \cap I_k \quad \text{in } M_c \ .$$

Und bekanntermaßen erhalten wir nun ein deterministisches, ε-freies Äquivalent, indem wir zu M_c den sogenannten „sparsamen Potenzautomaten“[18] $P(M_c)$ konstruieren, der allgemein wie folgt erklärt ist.

Sei $M = (Q_M, T_M, P_M, q_{s,M}, F_M)$ ein endlicher Automat. Dann ist der *volle Potenzautomat zu M* der deterministische, ε-freie Automat $(2^{Q_M}, T_M, \widehat{P}_M, \widehat{q}_{s,M}, \widehat{F}_M)$ mit den Aktionen

$$\hat{q}X \to \hat{q}' \in \widehat{P}_M \quad \Leftrightarrow \quad \hat{q}' = \{q' : qX \Rightarrow q'' \Rightarrow^* q' \quad \text{in } M \quad \text{für } q \in \hat{q},\, q'' \in Q_M\}$$

für alle $X \in T_M$ und $\hat{q}, \hat{q}' \subseteq Q_M$ sowie

$$\begin{aligned} \widehat{q}_{s,M} &= \{q : q_{s,M} \Rightarrow^* q \quad \text{in } M\} \ , \\ \widehat{F}_M &= \{\hat{q} \subseteq Q_M : \hat{q} \cap F_M \neq \emptyset\} \ . \end{aligned}$$

Daraus entsteht der *sparsame Potenzautomat zu M*, $P(M)$, durch das Entfernen des „Fehlerzustands“ $\emptyset$ und unerreichter Zustände: $P(M) = (\hat{Q}_M, T_M, \hat{P}_M, \hat{q}_{s,M}, \hat{F}_M)$ mit

$$\begin{aligned} \hat{Q}_M &= \{\emptyset \neq \hat{q} \subseteq Q_M : \widehat{q}_{s,M} x \Rightarrow^* \hat{q} \quad \text{im vollen Potenzautom. zu } M \text{ f. e. } x \in T_M^*\} \ , \\ \hat{P}_M &= \widehat{P}_M \cap (\hat{Q}_M T_M \times \hat{Q}_M) \ , \\ \hat{q}_{s,M} &= \widehat{q}_{s,M} \ , \\ \hat{F}_M &= \widehat{F}_M \cap \hat{Q}_M \ . \end{aligned}$$

[18] Heilbrunner (1981) spricht an entsprechender Stelle von der *„well-known subset construction for nondeterministic automata“*, wie sie in (Gries 1971) präsentiert sei, und nennt das noch um unerreichbare Zustände zu bereinigende Zwischenergebnis den *„completely specified automaton“*. Eine frühere Quelle für den Begriff *„subset construction“* ist (Harrison 1965), wo der Ergebnisautomat als *„subset machine“* bezeichnet wird. In Übereinstimmung mit Brauer (1984) erscheint dem Autor jedoch der Terminus „Potenzautomat“ besser gewählt zu sein.

$P(M)$ kann im übrigen induktiv leicht auch direkt, d. h. ohne den Zwischenschritt über den vollen Potenzautomaten, aufgebaut werden. Ferner gilt

$$\hat{q} = \mathrm{GOTO}_{P(M)}(\hat{q}_{s,M}, x) \quad \Leftrightarrow \quad \hat{q} = \{q \in Q_M : q_{s,M}x \Rightarrow^* q \quad \text{in } M\} \neq \emptyset \tag{3}$$

für alle $x \in {T_M}^*$, und insbesondere akzeptieren $P(M)$ und M dieselben Sprache.

Den Automaten $\hat{M}_c = \hat{M}^G_{c,\mathrm{LR}(k)} = P(M_c)$ nennen wir die *kanonische (deterministische) LR(k)-Maschine zu G.* Den Startzustand von $\hat{M}_c$ werden wir einheitlich mit $\hat{q}_{s,c}$ bezeichnen, seine Zustandsmenge mit $\hat{Q}_c$ und seine Aktionenmenge mit $\hat{P}_c$. Da schon in M_c alle Zustände Endzustände sind, gilt dasselbe auch für $\hat{M}_c$; die Endzustandsinformation ist insofern redundant. Für $\hat{M}_c$ gilt mit $\gamma \in V'^*, X \in V'$:[19]

1. $\hat{q}_{s,c} = \mathbf{desc}^*([S' \to \cdot S\$, \varepsilon])$,
2. in einer Aktion $\hat{q}X \to \hat{q}'$ ist $\hat{q}' = \mathbf{passes\text{-}}X\ \mathbf{desc}^*(\hat{q})$,
3. $\hat{q} = \mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \gamma) \quad \Leftrightarrow \quad \hat{q} = \mathrm{VALID}(\gamma) \neq \emptyset$,
4. $L(\hat{M}_c) = L(M_c) = \mathrm{VP}(G')$.

Die Zustände von $\hat{M}_c$ prägen den lebensfähigen Präfizes von G' eine Äquivalenzstruktur auf: Es heißen γ und γ' aus $\mathrm{VP}(G')$ *kanonisch LR(k)-äquivalent,* geschrieben $\gamma\ \rho^G_{c,\mathrm{LR}(k)}\ \gamma'$, genau dann, wenn $\mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \gamma) = \mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \gamma')$ gilt.

2.5 Kanonische LR(*k*)-Parser

Mit Hilfe der deterministischen kanonischen LR(k)-Maschine $\hat{M}_c$ zu G verhindern wir nun durch adäquaten Umbau des Shift-reduce-Parsers $(\tilde{M}_{sr}, \tau_{sr})$ zu G, daß der Parser Aktionen anwenden kann, die in Sackgassen führen: Wir erzwingen, daß der Kellerinhalt stets einem lebensfähigen Präfix von G (oder abschließend S) entspricht, indem wir statt eines Kellerinhalts $\gamma = X_1X_2\ldots X_n$ systematisch $\hat{q}_0X_1\hat{q}_1X_2\hat{q}_2\ldots X_n\hat{q}_n$ verwenden, wobei $X_i \in V$ und $\hat{q}_i = \mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, X_1X_2\ldots X_i)$ ist für $0 \leq i \leq n$. Somit wird auf dem Keller gleichzeitig mitprotokolliert, welche Zustandsfolge $\hat{q}_0, \hat{q}_1, \hat{q}_2, \ldots, \hat{q}_n$ von $\hat{M}_c$ für γ durchlaufen wird,[20]

[19] Die Konstruktion der deterministischen LR(k)-Maschine von Sippu und Soisalon-Soininen (1990) ist direkt und algorithmisch und die Herleitung ihrer Eigenschaften dann auch erheblich umständlicher als für die hier gezeigte, an (Heilbrunner 1981) angelehnte Konstruktion.

[20] Bekanntermaßen kann der Kellerinhalt sogar ohne Informationsverlust zu $\hat{q}_0\hat{q}_1\hat{q}_2\ldots\hat{q}_n$ verdichtet werden, was in (Sippu und Soisalon-Soininen 1990) auch so formuliert ist. Dies ist jedoch eine Optimierung. Unsere Formulierung hält sich an (Knuth 1965) und (Aho und Ullman 1972).

und die neuen Shift- und Reduce-Aktionen transformieren dann auf dem Keller, blenden wir dort die $\hat{M}_c$-Zustände aus, lebensfähige Präfizes von G in ebensolche (oder in S).

Mit diesen Überlegungen ist der *kanonische LR(k)-Parser zu G* ein Kellertransduktor $(\tilde{M}_c, \tau_c) = (\tilde{M}^G_{c,\mathrm{LR}(k)}, \tau^G_{c,\mathrm{LR}(k)})$, dessen Kelleralphabet die Vereinigung aus V und der Zustandsmenge von $\hat{M}_c$ ist, sein Eingabealphabet ist T, seine Kellerinitialisierung ist $\hat{q}_{s,c}$, sein einziger finaler Kellerinhalt ist $\hat{q}_{s,c}\, S\, \mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, S)$, und seine Aktionen setzen sich aus den „kanonischen Shift- und Reduce-Aktionen" zusammen, die der Ausgabehomomorphismus τ_c auf Folgen von Regeln von G abbildet:

Für jeden Zustand $\hat{q}$ von $\hat{M}_c$ und jedes $a \in T$ ist

$$r = \quad \hat{q}\,|\,a\,x \to \hat{q}\,a\,\mathrm{GOTO}_{\hat{M}_c}(\hat{q}, a)\,|\,x \tag{4}$$

eine *kanonische Shift-Aktion (von a)* genau dann, wenn ein k-Item $[A \to \alpha \cdot a\beta, z] \in \hat{q}$ existiert und $x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)$ ist; dazu wird $\tau_c(r) = \varepsilon$ gesetzt.

$$r = \quad \hat{q}\,X_1\,\mathrm{GOTO}_{\hat{M}_c}(\hat{q}, X_1) \ldots X_m\,\mathrm{GOTO}_{\hat{M}_c}(\hat{q}, X_1 \ldots X_m)\,|\,y \\ \to \hat{q}\,A\,\mathrm{GOTO}_{\hat{M}_c}(\hat{q}, A)\,|\,y \tag{5}$$

ist eine *kanonische Reduce-Aktion (nach Regel r')* genau dann, wenn $m \geq 0$ ist, $X_1, \ldots, X_m \in V$ und $r' = A \to X_1 \ldots X_m \in P$ sind und ferner das k-Item $[A \to \cdot X_1 \ldots X_m, y] \in \hat{q}$ existiert; dazu wird $\tau_c(r) = r'$ gesetzt.

Der kanonische LR(k)-Parser $(\tilde{M}_c, \tau_c)$ zu G verhält sich stark wie der Shift-reduce-Parser $(\tilde{M}_{sr}, \tau_{sr})$ zu G. (Die Umkehrung gilt jedoch i. a. nicht, da $\tilde{M}_{sr}$ in „Sackgassen" geraten kann.) $(\tilde{M}_c, \tau_c)$ ist in der Tat ein LR(k)-Parser; und G ist eine LR(k)-Grammatik genau dann, wenn $(\tilde{M}_c, \tau_c)$ deterministisch ist (sofern, wie vorausgesetzt, $S \Rightarrow^+ S$ in G unmöglich ist).[21]

Beispiel: Die Beispielgrammatik $G = G_{ab\varepsilon}$, die auch von Sippu und Soisalon-Soininen (1990)[22] verwendet wird, hat die Regeln

$$S \to a\,A \mid b\,B$$
$$A \to \varepsilon \mid c\,A\,d$$
$$B \to \varepsilon \mid c\,B\,d$$

und das Startsymbol S. Die Abbildungen 1 und 2 zeigen ihre zugehörige nichtdeterministische bzw. deterministische LR(1)-Maschine M_c bzw. $\hat{M}_c$. Die Aktionen des kanonischen

[21] Theoreme 6.34 und 6.39 in (Sippu und Soisalon-Soininen 1990); (Heilbrunner 1981).

[22] Siehe dort Kapitel 6.

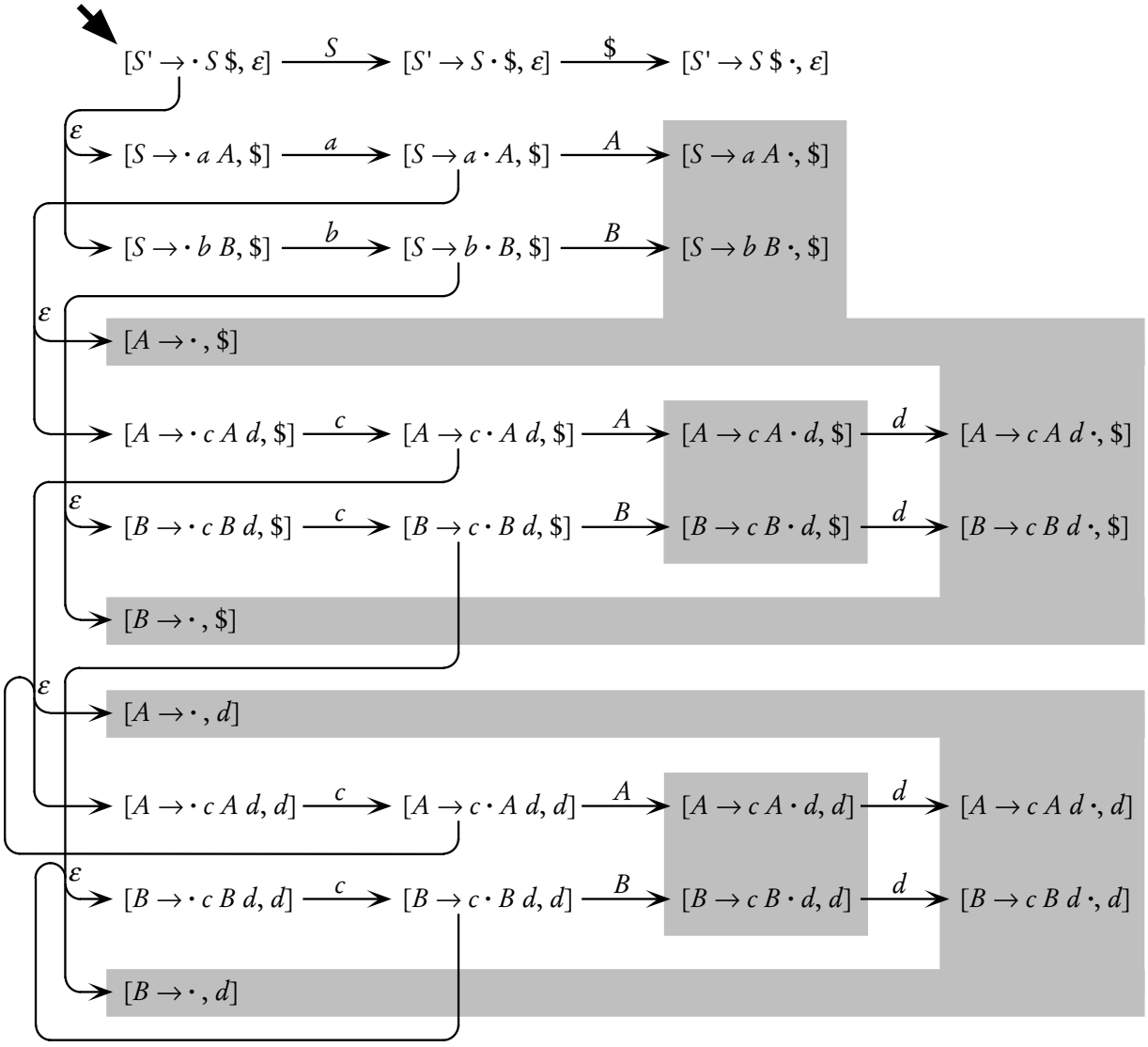


Abbildung 1: Der vom Startzustand $[S' \to \cdot\, S\$, \varepsilon]$ aus erreichbare Ausschnitt der kanonischen nichtdeterministischen LR(1)-Maschine M_c zu $G = G_{ab\varepsilon}$. Die Grauunterlegungen einiger Zustandsgruppen werden an späterer Stelle beschrieben.

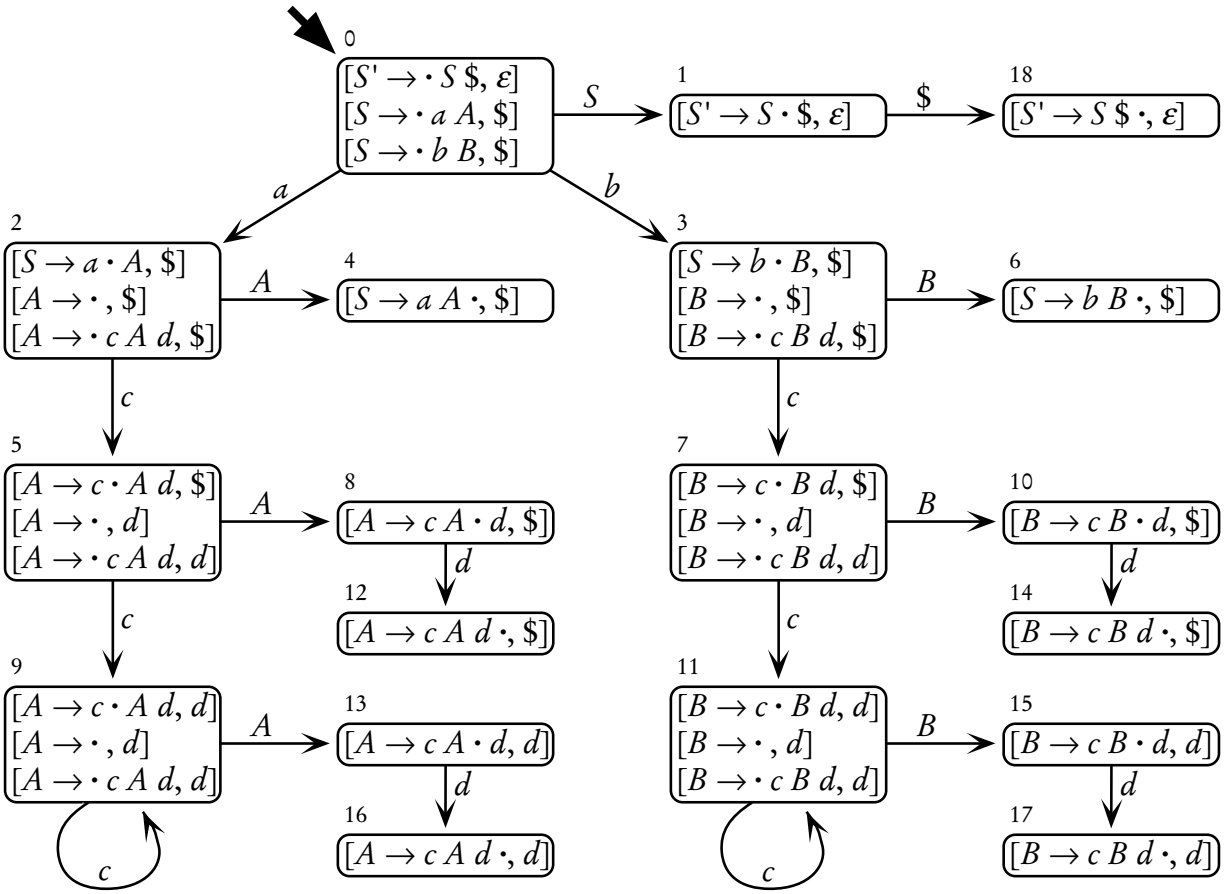


Abbildung 2: Die kanonische deterministische LR(1)-Maschine $\hat{M}_c$ zu $G = G_{ab\varepsilon}$.

LR(1)-Parsers $(\tilde{M}_c, \tau_c)$, in denen die relevanten Zustände von $\hat{M}_c$ gemäß den Annotationen in Abbildung 2 zu Zahlen abgekürzt wurden, sind:

$$
\begin{aligned}
r_1 &= 0\,a\,2\,A\,4\,|\,\$ \to 0\,S\,1\,|\,\$\,, & \tau_c(r_1) &= S \to a\,A\,.\\
r_2 &= 0\,b\,3\,B\,6\,|\,\$ \to 0\,S\,1\,|\,\$\,, & \tau_c(r_2) &= S \to b\,B\,.\\
r_3 &= 2\,|\,\$ \to 2\,A\,4\,|\,\$\,, & \tau_c(r_3) &= A \to \varepsilon\,.\\
r_4 &= 2\,c\,5\,A\,8\,d\,12\,|\,\$ \to 2\,A\,4\,|\,\$\,, & \tau_c(r_4) &= A \to c\,A\,d\,.\\
r_5 &= 3\,|\,\$ \to 3\,B\,6\,|\,\$\,, & \tau_c(r_5) &= B \to \varepsilon\,.\\
r_6 &= 3\,c\,7\,B\,10\,d\,14\,|\,\$ \to 3\,B\,6\,|\,\$\,, & \tau_c(r_6) &= B \to c\,B\,d\,.\\
r_7 &= 5\,|\,d \to 5\,A\,8\,|\,d\,, & \tau_c(r_7) &= A \to \varepsilon\,.\\
r_8 &= 5\,c\,9\,A\,13\,d\,16\,|\,d \to 5\,A\,8\,|\,d\,, & \tau_c(r_8) &= A \to c\,A\,d\,.\\
r_9 &= 7\,|\,d \to 7\,B\,10\,|\,d\,, & \tau_c(r_9) &= B \to \varepsilon\,.\\
r_{10} &= 7\,c\,11\,B\,15\,d\,17\,|\,d \to 7\,B\,10\,|\,d\,, & \tau_c(r_{10}) &= B \to c\,B\,d\,.\\
r_{11} &= 9\,|\,d \to 9\,A\,13\,|\,d\,, & \tau_c(r_{11}) &= A \to \varepsilon\,.\\
r_{12} &= 9\,c\,9\,A\,13\,d\,16\,|\,d \to 9\,A\,13\,|\,d\,, & \tau_c(r_{12}) &= A \to c\,A\,d\,.\\
r_{13} &= 11\,|\,d \to 11\,B\,15\,|\,d\,, & \tau_c(r_{13}) &= B \to \varepsilon\,.\\
r_{14} &= 11\,c\,11\,B\,15\,d\,17\,|\,d \to 11\,B\,15\,|\,d\,, & \tau_c(r_{14}) &= B \to c\,B\,d\,.\\
s_1 &= 0\,|\,a \to 0\,a\,2\,|\,, & \tau_c(s_1) &= \varepsilon\,.\\
s_2 &= 0\,|\,b \to 0\,b\,3\,|\,, & \tau_c(s_2) &= \varepsilon\,.\\
s_3 &= 2\,|\,c \to 2\,c\,5\,|\,, & \tau_c(s_3) &= \varepsilon\,.\\
s_4 &= 3\,|\,c \to 3\,c\,7\,|\,, & \tau_c(s_4) &= \varepsilon\,.\\
s_5 &= 5\,|\,c \to 5\,c\,9\,|\,, & \tau_c(s_5) &= \varepsilon\,.\\
s_6 &= 7\,|\,c \to 7\,c\,11\,|\,, & \tau_c(s_6) &= \varepsilon\,.\\
s_7 &= 8\,|\,d \to 8\,d\,12\,|\,, & \tau_c(s_7) &= \varepsilon\,.\\
s_8 &= 9\,|\,c \to 9\,c\,9\,|\,, & \tau_c(s_8) &= \varepsilon\,.\\
s_9 &= 10\,|\,d \to 10\,d\,14\,|\,, & \tau_c(s_9) &= \varepsilon\,.\\
s_{10} &= 11\,|\,c \to 11\,c\,11\,|\,, & \tau_c(s_{10}) &= \varepsilon\,.\\
s_{11} &= 13\,|\,d \to 13\,d\,16\,|\,, & \tau_c(s_{11}) &= \varepsilon\,.\\
s_{12} &= 15\,|\,d \to 15\,d\,17\,|\,, & \tau_c(s_{12}) &= \varepsilon\,.
\end{aligned}
$$

$(\tilde{M}_c, \tau_c)$ ist deterministisch und folglich $G_{ab\varepsilon}$ eine LR(1)-Grammatik. □

3 Allgemeine LR(k)-Parser

Die in einer beliebigen Situation von einem LR(k)-Parser $(\tilde{M},\tau)$ zu treffende Entscheidung ist von der dreifachen Frage „wann, was, wohin reduzieren ?“ geprägt, oder ausführlich: 1.) Ist zu reduzieren oder zu shiften ? 2.) Falls zu reduzieren ist, welches (bzw. wie lang) ist das Handle ? Und 3.) zu welcher linken Seite ist zu reduzieren ? Während ein implementierter LR(k)-Parser diese drei Teilfragen üblicherweise nacheinander behandelt, verschmelzen sie im formalen Modell, dem kanonischen LR(k)-Parser, zu singulären Aktionen, was dazu führt, daß die in (5) gezeigte Struktur einer kanonischen Reduce-Aktion nach einer Regel $A \to X_1 \ldots X_m$ den Eindruck erweckt, es müßten die obersten (hier) $2m+1$ Kellersymbole tatsächlich *alle* herangezogen werden, um über die Adäquatheit einer Reduktion nach $A \to X_1 \ldots X_m$ zu entscheiden. Das ist jedoch nicht der Fall. Es ist bekannt und wird in den Implementierungen stets genutzt, daß, in Kombination mit dem Lookahead, in das oberste Kellersymbol *alleine* schon die Antwort auf jede der drei oben skizzierten Fragen fokussiert ist.

Um dies kurz zu rekapitulieren, sei $[A \to \alpha \cdot \beta, y] \in \mathrm{VALID}(\gamma)$. Dann ist $\gamma = \delta\alpha$ für ein δ, und mit $\alpha = \alpha'\alpha''$ gilt $[A \to \alpha' \cdot \alpha''\beta, y] \in \mathrm{VALID}(\delta\alpha')$. Mithin folgt dann aus $\{[A_1 \to \alpha_1 \cdot \beta_1, y_1], [A_2 \to \alpha_2 \cdot \beta_2, y_2]\} \subseteq \mathrm{VALID}(\gamma)$, daß α_1 Suffix von α_2 oder α_2 Suffix von α_1 ist. Alle Items aus $\mathrm{VALID}(\gamma)$ teilen also eine „gemeinsame Geschichte“, und das Kellersymbol $\mathrm{VALID}(\gamma)$ codiert quasi die längste solche. Jeder $\hat{M}_c$-Zustand $\hat{q}$ außer dem Startzustand $\hat{q}_{s,c} = \mathrm{VALID}(\varepsilon)$ enthält mindestens ein Kernelitem der Form $[A \to \alpha X \cdot \beta, y]$ mit $X \in V'$ und codiert damit mindestens die eindeutige „1-Zeichen-Restgeschichte“ X, weswegen X auch *das Access-Symbol* von $\hat{q}$ genannt wird; genauer könnte die längste solche Itemkomponente α der Items aus $\hat{q}$ sogar als *die Access-Symbolfolge* von $\hat{q}$ bezeichnet werden, dieses wird jedoch in der Literatur nicht erwähnt.

Hier mag der Hinweis hilfreich sein, daß die Eigenschaft des eindeutigen Access-Symbols in den kanonischen deterministischen LR(k)-Maschinen nicht etwa Folge der Konstruktion des sparsamen Potenzautomaten ist. Letztere ist ein allgemeines Verfahren zur Bestimmung eines äquivalenten deterministischen zu einem nichtdeterministischen endlichen Automaten, dessen Ergebnisautomaten im allgemeinen mitnichten Zustände mit eindeutigem Access-Symbol aufweisen. Vielmehr ist dies im Falle der kanonischen deterministischen LR(k)-Maschine $\hat{M}_c$ eine Konsequenz der besonderen Struktur der zugrunde liegenden kanonischen nichtdeterministischen LR(k)-Maschine M_c, deren Konstruktionsweise mit

sich bringt, daß jedes k-Item $[A \to \alpha \cdot \beta, y]$ seine „individuelle Geschichte“ α mitcodiert.

Bezogen auf die in (5) gezeigte Struktur einer kanonischen Reduce-Aktion, ist damit plausibilisiert, wieso darin der Kellerabschnitt

$$X_1 \, \mathrm{GOTO}_{\hat{M}_c}(\hat{q}, X_1) \ldots \mathrm{GOTO}_{\hat{M}_c}(\hat{q}, X_1 \ldots X_{m-1}) \, X_m$$

tatsächlich eine reine Platzhalterrolle spielt und das darunterliegende Kellersymbol $\hat{q}$ lediglich für die Ermittlung des neu zu kellernden Zustands $\mathrm{GOTO}_{\hat{M}_c}(\hat{q}, A)$ konsultiert wird.

Analoge Zusammenhänge sind in angepaßter Form auch für abgeleitete LR-Parserverfahren wie etwa SLR(k)- und LALR(k)-Parser zu beobachten.

Angesichts der nun offensichtlichen Redundanz in der kanonischen Itemmengenkonstruktion wollen wir uns im folgenden damit beschäftigen, ob es nicht möglich ist, weniger redundante, mithin „kleinere“ als die kanonischen LR(k)-Parser zu konstruieren. Spontan mag einem Leser an dieser Stelle in den Sinn kommen, dies wäre vielleicht schlicht mit bekannten Methoden zur Minimierung des DEA zu erreichen, den wir die deterministische kanonische LR(k)-Maschine zu G nennen. Dies greift jedoch zu kurz! Um dieses einzusehen, machen wir uns klar, daß wir im Umgang mit den kanonischen LR(k)-Maschinen zu G zwei Sichten einnehmen, die wir die „Innensicht“ und die „Außensicht“ nennen wollen. Bei der Außensicht auf die Struktur einer kanonischen LR(k)-Maschine sind wir an ihrer Eigenschaft interessiert, lebensfähige Präfizes von G' zu erkennen, nicht dagegen an Information über zugehörige LR(k)-gültige Items. (Wir reden hier über ihre Zustände als „irgendwelche $\hat{q}$, $\hat{q}'$, … bzw. q, q', … “.) Diese Sicht unterliegt beispielsweise der sparsamen Potenzkonstruktion und auch der *Struktur* der kanonischen Parseraktionen. Bei der Innensicht hingegen, die der *Erstellung* der Parseraktionen unterliegt, interessiert uns – speziell für Lookahead-Information – gerade die Verbindung zwischen lebensfähigen Präfizes und ihren LR(k)-gültigen Items. Insbesondere führen längere Lookaheads bei gleicher Außenansicht zu einer detaillierteren Innenansicht, und die Größe der kanonischen LR(k)-Maschinen wächst i. a., obwohl die gleichen lebensfähigen Präfizes erkannt werden. Insgesamt wird damit klar, daß die Idee der DEA-Minimierung der deterministischen kanonischen LR(k)-Maschine zu G deswegen verworfen werden muß, weil sie nur die Außen-, nicht jedoch die Innensicht berücksichtigt.

Um für die Entwicklung von Verfahren zur Konstruktion weniger redundanter LR(k)-Parser als den kanonischen eine genügend konkrete Ausgangsposition zu schaffen, werden wir nun zunächst einen Rahmen für deren strukturellen Aufbau stecken.

Wir halten zunächst fest, daß wir uns zu $G = (V,T,P,S)$ nur für Kellertransduktoren mit „reinem LR(k)-Verhalten“ interessieren wollen, sprich: für Kellertransduktoren $(\tilde{M},\tau)$, die sich wie der kanonische LR(k)-Parser $(\tilde{M}_\mathrm{c},\tau_\mathrm{c})$ verhalten und umgekehrt. Und natürlich sollten in $\tilde{M}$ (wie in $\tilde{M}_\mathrm{c}$ der Fall) keine nutzlosen Aktionen vorkommen.

Da jeder Berechnungsschritt von $\tilde{M}_\mathrm{c}$ auf dem Keller eine Repräsentation eines lebensfähigen Kellerinhalts ψ des Shift-reduce-Parsers $(\tilde{M}_\mathrm{sr},\tau_\mathrm{sr})$ in die Repräsentation eines wiederum lebensfähigen Kellerinhalts φ transformiert, leistet dieses entsprechend auch $\tilde{M}$ – nur werden ψ und φ von $\tilde{M}$ möglicherweise anders repräsentiert: $\tilde{M}_\mathrm{c}$ stellt einen lebensfähigen Kellerinhalt $X_1 \ldots X_n$ von $\tilde{M}_\mathrm{sr}$ durch $\hat{q}_0 X_1 \hat{q}_1 \ldots X_n \hat{q}_n$ dar, wobei $\hat{q}_0, \hat{q}_1, \ldots, \hat{q}_n$ die von der deterministischen kanonischen LR(k)-Maschine $\hat{M}_\mathrm{c}$ zu G für $X_1 \ldots X_n$ nacheinander durchlaufenen Zustände sind, d. h. $\hat{q}_i = \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, X_1 \ldots X_i)$ für $0 \leq i \leq n$. Und wir beschränken uns im folgenden darauf, daß die von uns betrachteten Kellertransduktoren $(\tilde{M},\tau)$ selbiges $X_1 \ldots X_n$ analog repräsentieren, nämlich durch $\hat{q}_0 X_1 \hat{q}_1 \ldots X_n \hat{q}_n$, wobei auch hier $\hat{q}_0, \hat{q}_1, \ldots, \hat{q}_n$ Zustände eines buchstabierenden deterministischen endlichen Automaten $\hat{M}$ sind, der den Startzustand $\hat{q}_\mathrm{s}$ auf Symbolfolgen aus V'^* operiert und für den $\hat{q}_i = \mathrm{GOTO}_{\hat{M}}(\hat{q}_\mathrm{s}, X_1 \ldots X_i)$ für $0 \leq i \leq n$ gilt.

Wir wollen weiterhin, daß in $\tilde{M}$ genau dieselbe Menge an Kellerinformation zur Entscheidung über anzuwendende Aktionen herangezogen wird wie in $\tilde{M}_\mathrm{c}$. Daher fordern wir von den Aktionen von $\tilde{M}$ sowie dessen initialen und finalen Kellerinhalten einen zu $\tilde{M}_\mathrm{c}$ analogen Aufbau: Es sei $\hat{q}_\mathrm{s}$ der initiale und $\hat{q}_\mathrm{s} S\, \mathrm{GOTO}_{\hat{M}}(\hat{q}_\mathrm{s}, S)$ der einzige finale Kellerinhalt von $\tilde{M}$. Eine Shift-Aktion (von a) sei für $\tilde{M}$ von der zu (4) analogen Struktur

$$r = \quad \hat{q} \mid ax \to \hat{q}\, a\, \mathrm{GOTO}_{\hat{M}}(\hat{q}, a) \mid x \tag{6}$$

mit $\tau(r) = \varepsilon$, und eine Reduce-Aktion (nach $r' = A \to X_1 \ldots X_m$) sei von der zu (5) analogen Struktur

$$\begin{aligned} r = \quad & \hat{q}\, X_1\, \mathrm{GOTO}_{\hat{M}}(\hat{q}, X_1) \ldots X_m\, \mathrm{GOTO}_{\hat{M}}(\hat{q}, X_1 \ldots X_m) \mid y \\ & \to \hat{q}\, A\, \mathrm{GOTO}_{\hat{M}}(\hat{q}, A) \mid y \end{aligned} \tag{7}$$

mit $\tau(r) = r'$.

Und schließlich soll der Aufbau von $\hat{M}$ den von $(\tilde{M},\tau)$ in „derselben Art und Weise“ reflektieren, wie $\hat{M}_\mathrm{c}$ dies für $(\tilde{M}_\mathrm{c},\tau_\mathrm{c})$ tut: Auch $\hat{M}$ soll ein DEA sein, der genau die lebensfähigen Präfizes von G' akzeptiert, so daß in $\hat{M}_\mathrm{c}$ wie in $\hat{M}$ jeder T-Übergang je eine Gruppe von Shift-Aktionen und jeder $(V \setminus T)$-Übergang je eine Gruppe von Reduce-Aktionen in $\tilde{M}_\mathrm{c}$ respektive $\tilde{M}$ anzeigt.

Die Mitglieder $(\tilde{M}, \tau)$ der skizzierten Familie von Kellertransduktoren werden wir als „allgemeine LR(k)-Kellertransduktoren“ bezeichnen. Und natürlich wird auch der kanonische LR(k)-Parser $(\tilde{M}_c, \tau_c)$ selbst zu dieser Familie gehören. Daher und aufgrund des gemeinsamen Aufbauschemas aller ihrer Mitglieder bietet es sich dann an, die Herleitung der wesentlichen Eigenschaften dieser Familie durch Analogie aus den bekannten Eigenschaften der kanonischen Standardkonstruktion vorzunehmen.

Wir wollen zunächst die Familie der allgemeinen LR(k)-Kellertransduktoren präzise formulieren. Hier ist klar, daß für das Ablesen von Aktionen des Kellertransduktors aus dem zugrunde gelegten DEA, für den wir die Bezeichnung „allgemeine LR(k)-Maschine“ verwenden werden, Information über LR(k)-gültige Items nicht mehr verwendet werden soll. (Die Itemmengenkonstruktion soll ja gerade zur Disposition gestellt werden !)

Stellen wir uns also vor, daß in der kanonischen LR(k)-Maschine $\hat{M}_c$ die Zustände nunmehr nur noch voneinander unterscheidbare Entitäten sind, die keine weiteren Informationen durch „Hineinsehen“ preisgeben. Ist es immer noch möglich, aus $\hat{M}_c$ dann $\tilde{M}_c$ und τ_c abzulesen ? Dies gelingt tatsächlich noch für den Fall $k = 0$: Die Shift-Aktionen lassen sich aus den T-Übergängen von $\hat{M}_c$, die Reduce-Aktionen mit Hilfe von G aus den $(V \setminus T)$-Übergängen rekonstruieren. Für $k \geq 1$ ($k \geq 2$ bei Shift-Aktionen) ist jedoch die Lookahead-Komponente der abzulesenden Aktionen nicht zu ermitteln. Ein zur Konstruktion geeigneter DEA $\hat{M}$ muß daher neben der Eigenschaft, exakt die lebensfähigen Präfizes von G' zu akzeptieren, genau diese fehlenden Lookahead-Anteile als zusätzliche Stützinformation aufweisen.

3.1 LR(k)-Rechtskontext

Da alle lebensfähigen Kellerinhalte des Shift-reduce-Parsers $(\tilde{M}_{sr}, \tau_{sr})$ zu G lebensfähige Präfizes von G' sind, können wir die Wirkung einer jeden Aktion des aus $\hat{M}$ abzulesenden Parsers $(\tilde{M}, \tau)$ auch als Transformation eines lebensfähigen Präfix' ψ in ein anderes lebensfähiges Präfix φ von G' auffassen. Für eine Shift-Aktion von a wird ψ in $\varphi = \psi a$ und für eine Reduce-Aktion nach $A \to \omega$ wird $\psi = \delta\omega$ in $\varphi = \delta A$ transformiert.

Die für die Konstruktion von $\tilde{M}$ aus $\hat{M}$ relevanten Lookaheads zu einem lebensfähigen Präfix φ von G' charakterisieren wir daher wie folgt.

Der *(LR(k)-)Rechtskontext* von $\varphi \in \mathrm{VP}(G')$, $\mathrm{RC}(\varphi) = \mathrm{RC}_{G,\mathrm{LR}(k)}(\varphi)$, ist mit $A \in V' \setminus T'$,

$a \in T'$ und $\gamma \in V'^*$ definiert durch

$$\begin{aligned}
\mathrm{RC}(\varepsilon) &= \emptyset\,, \\
\mathrm{RC}(\gamma A) &= \{(k:z) : S' \underset{\mathrm{rm}}{\Longrightarrow}^{+} \gamma A z\}\,, \\
\mathrm{RC}(\gamma a) &= \bigcup\{\mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z) : S' \underset{\mathrm{rm}}{\Longrightarrow}^{*} \delta B z \underset{\mathrm{rm}}{\Longrightarrow} \delta\alpha a\beta z = \gamma a\beta z\} \\
&= \{(\max\{k \dot{-} 1, 0\} : z) : S' \underset{\mathrm{rm}}{\Longrightarrow}^{+} \gamma a z\}\,.
\end{aligned}$$

Oder wegen der geforderten Reduziertheit von G äquivalent, aber für unsere Zwecke suggestiver:

$$\begin{aligned}
\mathrm{RC}(\varepsilon) &= \emptyset\,, \\
\mathrm{RC}(\gamma A) &= \{y : \text{es ex. } [A \to \cdot\,\omega, y] \in \mathrm{VALID}_k(\gamma) \text{ mit } A \neq S'\}\,, \\
\mathrm{RC}(\gamma a) &= \{x : \text{es ex. } [B \to \alpha \cdot a\beta, z] \in \mathrm{VALID}_k(\gamma), \\
&\qquad\qquad \text{so daß } x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)\}\,.
\end{aligned} \tag{8}$$

Offensichtlich ist $\mathrm{RC}(\varphi)$ (ausgenommen den Grenzfall $\gamma a = S\$$) genau die Menge (i) der Lookahead-Anteile derjenigen Reduce-Aktionen und (ii) der über das zu shiftende Symbol hinausgehenden Lookahead-Verlängerungen derjenigen Shift-Aktionen des kanonischen LR(k)-Parsers zu G, die die Wirkung haben, irgendein anderes lebensfähiges Präfix ψ von G' gerade in φ zu transformieren. Und genau dieselbe Information wird in jedem allgemeinen LR(k)-Parser zu G benötigt.

Wie leicht zu sehen ist, gilt das

Lemma 3.1
Ist $\mathrm{VALID}_k(\gamma_1) = \mathrm{VALID}_k(\gamma_2) \neq \emptyset$ *und ist mit* $X \in V'$ *neben* γ_1 *und* γ_2 *auch* $\gamma_1 X \in \mathrm{VP}(G')$ *(und mithin* $\gamma_2 X \in \mathrm{VP}(G')$*), so ist* $\mathrm{RC}(\gamma_1 X) = \mathrm{RC}(\gamma_2 X)$. □

Da für zwei von der kanonischen LR(k)-Maschine $\hat{M}_c$ mittels Aktionenfolgen $\pi_1 r$ respektive $\pi_2 r$ akzeptierte lebensfähige Präfizes $\gamma_1 X$ und $\gamma_2 X$ von G' aber

$$\begin{aligned}
r &= \mathrm{VALID}_k(\gamma_1) X \to \mathrm{VALID}_k(\gamma_1 X) \\
&= \mathrm{VALID}_k(\gamma_2) X \to \mathrm{VALID}_k(\gamma_2 X)\,,
\end{aligned}$$

insbesondere also $\mathrm{VALID}_k(\gamma_1) = \mathrm{VALID}_k(\gamma_2)$ ist, muß auch $\mathrm{RC}(\gamma_1 X) = \mathrm{RC}(\gamma_2 X)$ sein. Hinterlegten wir also zu $\hat{M}_c$ die Rechtskontextinformation $\mathrm{RC}(\gamma X)$ *an* dem letzten Übergang r, mit dem γX akzeptiert wird, so wäre diese Etikettierung von r wohldefiniert:

Lemma 3.2
Akzeptiert $\hat{M}_c$ zwei Eingaben $\varphi, \varphi' \in V'^*$ mit Aktionenfolgen π respektive π' und gilt $\pi : 1 = \pi' : 1$, so ist $\mathrm{RC}(\varphi) = \mathrm{RC}(\varphi')$. □

Für unser Vorhaben der Definition allgemeiner LR(k)-Parser zu G auf der Grundlage allgemeiner LR(k)-Maschinen zu G bedeutet die obige Herleitung, daß die dem zu konstruierenden Parser $(\tilde{M}, \tau)$ zugrunde liegende Maschine $\hat{M}$ zu jedem letzten Übergang für ein von ihm akzeptiertes lebensfähiges Präfix $\varphi \neq \varepsilon$ von G' dessen Menge $\mathrm{RC}(\varphi)$ möglicher Rechtskontexte berechnen können muß.

3.2 Definition allgemeiner LR(k)-Parser

Prinzipiell bieten sich nun an einem DEA, der lebensfähige Präfizes erkennt, zwei verschiedene sinnvolle Orte an, wo die für die Parserkonstruktion notwendige Stützinformation der LR(k)-Rechtskontexte bereitgehalten werden kann: am letzten Übergang, mit dem ein nichtleeres lebensfähiges Präfix akzeptiert wird, oder am letzten Zustand unmittelbar vor diesem Übergang. Wir werden die letztere Modellierungsform wählen, und zwar wollen wir, wenn für eine akzeptierte Eingabe γX vom DEA die Zustandsfolge $\hat{q}_{s,c}, \ldots, \hat{q}, \hat{q}'$ durchlaufen wird, die $u \in \mathrm{RC}(\gamma X)$ gepaart mit X an $\hat{q}$ hinterlegen. Ein Paar (a, u) mit $a \in T$ zeigt dann an $\hat{q}$ eine Shift-Aktion von a bei Eingabepräfix au an; (A, u) mit $A \in V \setminus T$ zeigt an, daß bei Lookahead u der Zustand $\hat{q}$ im Parserkeller ein Handle ω nach unten begrenzt, das zur linken Seite A zu reduzieren ist.

Daß diese Modellierungsform und ihre Alternative, $\mathrm{RC}(\gamma X)$ dem *Übergang* $\hat{q}X \to \hat{q}'$ beizuordnen, direkt ineinander umgerechnet werden können, ist leicht einzusehen. Die gewählte Form der Verankerung an den DEA-*Zuständen* bietet den wesentlichen Vorteil, Stützinformation leichter als Abstraktion entsprechender Mengen LR(k)-gültiger Items ansatzweise auffassen zu können. Die wesentlichen Unterschiede allgemeiner zu kanonischen LR(k)-Maschinen (und dann allgemeiner LR(k)-Parser zu kanonischen LR(k)-Parsern) werden so kognitiv fokussiert. Als Konsequenz daraus wird später ein Verfahren zur direkten Konstruktion kompakter allgemeiner LR(k)-Parser abgeleitet werden können. Auch gelingt die Modellierung von später diskutierter weitergehender Stützinformation zu Optimierungszwecken ohne Bruch. Wir formulieren nun die folgende Definition einer allgemeinen LR(k)-Maschine als „DEA mit Zusatzinformation an den Zuständen“:

Eine *allgemeine (deterministische) LR(k)-Maschine zu G* ist ein System $(\hat{M}, \mathrm{RC}_{\hat{M}})$, in dem $\hat{M}$ ein vollständig spezifizierter, buchstabierender DEA ohne unerreichbare Zustände (mit

Startzustand $\hat{q}_s$) ist, der die Sprache $\mathrm{VP}(G')$ der lebensfähigen Präfizes von G' akzeptiert, und $\mathrm{RC}_{\hat{M}}$ ist eine Funktion, die jedem Zustand von $\hat{M}$ Rechtskontextinformation in Form von Paaren (X,u), notiert als $X \mid u$, zuordnet, so daß für alle $\gamma \in \mathrm{VP}(G')$ gilt:

$$\mathrm{RC}_{\hat{M}}(\mathrm{GOTO}_{\hat{M}}(\hat{q}_s,\gamma)) = \{X \mid u : X \in V',\ \gamma X \in \mathrm{VP}(G'),\ u \in \mathrm{RC}(\gamma X)\}\ . \tag{9}$$

Die suggestivere Schreibweise $X \mid u$ für ein Paar (X,u) charakterisiert prägnant jede Parserkonfiguration, die durch Anwendung einer solchen Parseraktion erreicht wird, welche aus der Information $X \mid u = (X,u) \in \mathrm{RC}_{\hat{M}}(\hat{q})$ heraus konstruiert wird: $a \mid u$ stilisiert die Wirkung eines Shifts von a bei Eingabepräfix au, $A \mid u$ die einer Reduktion zur linken Seite A, wobei $\hat{q}$ im Parserkeller das Handle nach unten begrenzt.

Wie angekündigt, ist die kanonische LR(k)-Maschine $\hat{M}_c$ zu G selbst auch eine allgemeine LR(k)-Maschine zu G; genauer können wir eine Funktion $\mathrm{RC}_c = \mathrm{RC}_{\hat{M}_c}$ angeben, die den Zuständen von $\hat{M}_c$ Rechtskontextpaare zuordnet, so daß $(\hat{M}_c, \mathrm{RC}_c)$ die Definition einer allgemeinen LR(k)-Maschine zu G erfüllt:

$$\begin{aligned}\mathrm{RC}_c(\hat{q}) = {} & \{A \mid y : \text{es ex. } [A \to \cdot\,\omega, y] \in \hat{q} \text{ mit } A \neq S'\} \quad \cup \\ & \{a \mid x : \text{es ex. } [A \to \alpha \cdot a\beta, z] \in \hat{q} \text{ mit } a \in T', \\ & \qquad \text{so daß } x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)\}\ .\end{aligned} \tag{10}$$

Jede Paarmenge $\mathrm{RC}_c(\hat{q})$ kann – für $k > 1$ unter Zuhilfenahme von G' für FIRST$'$ – direkt aus den k-Items in $\hat{q}$ extrahiert werden und stellt eine starke Abstraktion der in der Itemmenge $\hat{q}$ enthaltenen Gesamtinformation dar.

Lemma 3.3
$(\hat{M}_c, \mathrm{RC}_c)$ *ist eine allgemeine LR(k)-Maschine zu G.*

Beweis: Die kanonische LR(k)-Maschine $\hat{M}_c$ ist nach Konstruktion ein vollständig spezifizierter, buchstabierender DEA, der die Sprache der lebensfähigen Präfizes von G' akzeptiert. Weiter besitzt $\hat{M}_c$ keine unerreichbaren Zustände, und insbesondere gilt für $\hat{M}_c$, daß $\mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c},\gamma) = \mathrm{VALID}_k(\gamma)$ für alle $\gamma \in \mathrm{VP}(G')$ ist. Daher ist (10) äquivalent zu

$$\begin{aligned}\mathrm{RC}_c(\mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c},\gamma)) = {} & \{A \mid y : \text{es ex. } [A \to \cdot\,\omega, y] \in \mathrm{VALID}_k(\gamma) \text{ mit } A \neq S'\} \ \cup \\ & \{a \mid x : \text{es ex. } [A \to \alpha \cdot a\beta, z] \in \mathrm{VALID}_k(\gamma) \text{ mit } a \in T', \\ & \qquad \text{so daß } x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)\}\ .\end{aligned}$$

Mit geläufigem Strukturwissen über $\hat{M}_{\mathrm{c}}$ und mit der Definition (8) ergibt sich daraus

$$\begin{aligned}\mathrm{RC}_{\mathrm{c}}(\mathrm{GOTO}_{\hat{M}_{\mathrm{c}}}(\hat{q}_{\mathrm{s,c}},\gamma)) = {} & \{A \mid y : A \in V' \setminus T',\ \gamma A \in \mathrm{VP}(G'),\ y \in \mathrm{RC}(\gamma A)\} \quad \cup \\ & \{a \mid x : a \in T',\ \gamma a \in \mathrm{VP}(G'),\ x \in \mathrm{RC}(\gamma a)\} \\ = {} & \{X \mid u : X \in V',\ \gamma X \in \mathrm{VP}(G'),\ u \in \mathrm{RC}(\gamma X)\}\,,\end{aligned}$$

was zu zeigen war. □

Für die Beispielgrammatik $G = G_{ab\varepsilon}$ von Seite 34 stellt Abbildung 3 die kanonische LR(1)-Maschine ihrem Pendant in Gestalt der allgemeinen LR(1)-Maschine gegenüber und verdeutlicht die Abstraktion von der Information $\mathrm{VALID}_k(\gamma) = \mathrm{GOTO}_{\hat{M}_{\mathrm{c}}}(\hat{q}_{\mathrm{s,c}},\gamma)$ hin zu $\mathrm{RC}_{\mathrm{c}}(\mathrm{GOTO}_{\hat{M}_{\mathrm{c}}}(\hat{q}_{\mathrm{s,c}},\gamma))$. Man beachte, daß die abstraktere Information $\mathrm{RC}_{\mathrm{c}}(\hat{q})$ einen Zustand $\hat{q}$ nicht mehr eindeutig identifiziert.

Abbildung 4 zeigt eine kompaktere LR(1)-Maschine zu $G_{ab\varepsilon}$, die 8 Zustände im Gegensatz zu 19 Zuständen der kanonischen LR(1)-Maschine aufweist.

Sei $(\hat{M},\mathrm{RC}_{\hat{M}})$ eine allgemeine LR(k)-Maschine zu G mit Startzustand $\hat{q}_{\mathrm{s}}$. Dann kann aus dieser, nun ohne auf die Zustandsbezeichnungen von $\hat{M}$ Bezug nehmen zu müssen, ein Kellertransduktor $(\tilde{M},\tau)$ konstruiert werden, indem als dessen Kelleralphabet die Vereinigung aus V und der Zustandsmenge von $\hat{M}$ gewählt wird, T als Eingabealphabet, $\hat{q}_{\mathrm{s}}$ als Kellerinitialisierung, $\hat{q}_{\mathrm{s}}\,S\,\mathrm{GOTO}_{\hat{M}}(\hat{q}_{\mathrm{s}},S)$ als einziger finaler Kellerinhalt, und die Aktionen von $\tilde{M}$ sind „Shift-“ und „Reduce-Aktionen“, die wie folgt erklärt sind.

Für jeden Zustand $\hat{q}$ von $\hat{M}$ und jedes $a \in T$ ist

$$r = \hat{q} \mid a\,x \to \hat{q}\,a\,\mathrm{GOTO}_{\hat{M}}(\hat{q},a) \mid x$$

eine *Shift-Aktion (von a)* genau dann, wenn $a \mid x \in \mathrm{RC}_{\hat{M}}(\hat{q})$; dazu wird $\tau(r) = \varepsilon$ gesetzt. Weiter ist

$$\begin{aligned} r = \quad & \hat{q}\,X_1\,\mathrm{GOTO}_{\hat{M}}(\hat{q},X_1)\ldots X_m\,\mathrm{GOTO}_{\hat{M}}(\hat{q},X_1\ldots X_m) \mid y \\ & \qquad\qquad \to \hat{q}\,A\,\mathrm{GOTO}_{\hat{M}}(\hat{q},A) \mid y \end{aligned}$$

eine *Reduce-Aktion (nach Regel r′)* genau dann, wenn $m \geq 0$ ist, $X_1,\ldots,X_m \in V$ und $r' = A \to X_1\ldots X_m \in P$ sind und ferner $A \mid y \in \mathrm{RC}_{\hat{M}}(\hat{q})$ gilt; dazu wird $\tau(r) = r'$ gesetzt.

$(\tilde{M},\tau)$ heißt der aus $(\hat{M},\mathrm{RC}_{\hat{M}})$ konstruierte *allgemeine LR(k)-Kellertransduktor zu G.* Offensichtlich gilt das folgende

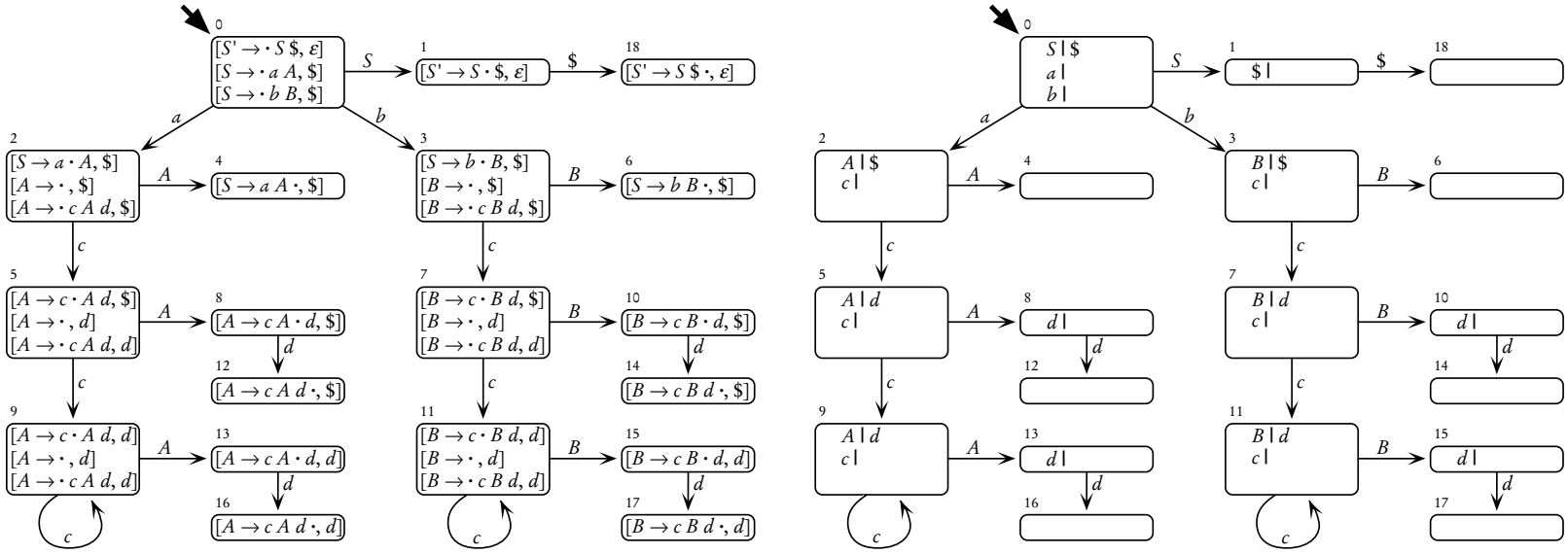


Abbildung 3: Die kanonische LR(1)-Maschine $\hat{M}_c$ zu $G = G_{ab\varepsilon}$ mit eingezeichneten Itemmengen (links) und ihr Pendant in Gestalt der allgemeinen LR(1)-Maschine $(\hat{M}_c, \mathrm{RC}_c)$ mit eingezeichneten Rechtskontextpaaren (rechts). $\hat{M}_c$ weist 19 Zustände auf. Der kanonische LR(1)-Parser dazu besitzt 12 Shift-Aktionen (je 1 von a und b, 6 von c, 4 von d) und 14 Reduce-Aktionen (je 1 nach $S \to a\,A$ und $S \to b\,B$, je 3 nach $A \to \varepsilon$, $A \to c\,A\,d$, $B \to \varepsilon$ und $B \to c\,B\,d$) und ist deterministisch.

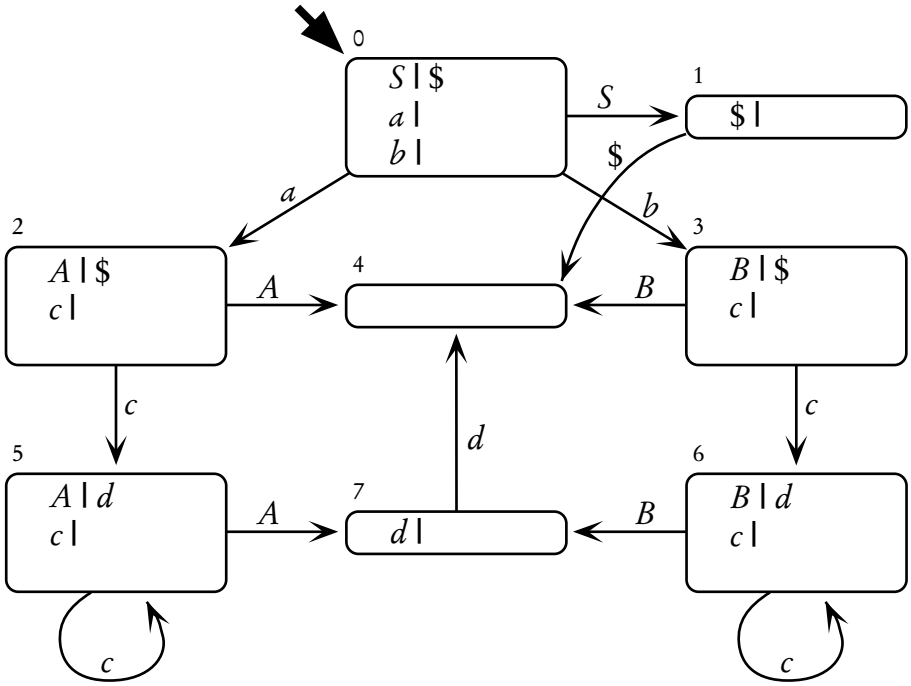


Abbildung 4: Die allgemeine LR(1)-Maschine zu $G = G_{ab\varepsilon}$ mit geringster Zustandszahl. Sie weist 8 Zustände auf. Der daraus konstruierte allgemeine LR(1)-Parser besitzt 7 Shift-Aktionen (je 1 von a, b und d, 4 von c) und 10 Reduce-Aktionen (je 1 nach $S \to a\,A$ und $S \to b\,B$, je 2 nach $A \to \varepsilon$, $A \to c\,A\,d$, $B \to \varepsilon$ und $B \to c\,B\,d$) und ist deterministisch. Die gezeigte LR(1)-Maschine ist isomorph zur LR(0)-Maschine mit geringster Zustandszahl. Der aus letzterer konstruierte allgemeine LR(0)-Parser weist ebenfalls 7 Shift- und 10 Reduce-Aktionen auf, ist jedoch nichtdeterministisch.

Lemma 3.4
Zum kanonischen LR(k)-Parser $\hat{M}_c$ zu G liefern Standardkonstruktion und die Konstruktion über $(\hat{M}_c, \mathrm{RC}_c)$ denselben Kellertransduktor: den kanonischen LR(k)-Parser $(\tilde{M}_c, \tau_c)$.
□

3.3 Eigenschaften allgemeiner LR(k)-Parser

Wir wollen nun zeigen, daß die obige Konstruktion LR(k)-Parser zu G liefert, und in einem weiteren Schritt, daß diese LR(k)-Parser, wie der kanonische zu G, genau dann deterministisch sind, wenn G eine LR(k)-Grammatik ist. Den Beweis ersterer Aussage bereiten wir vor mittels zweier Lemmata. Einem ersten, welches die Kellerinhalte erreichbarer Konfigurationen der allgemeinen LR(k)-Kellertransduktoren charakterisiert. Und einem weiteren, welches besagt, daß sich zu G zwei beliebige solche Kellertransduktoren stets gleich verhalten.

Lemma 3.5 (Kellerinhalte allgemeiner LR(k)-Kellertransduktoren)
Sei $(\hat{M}, \mathrm{RC}_{\hat{M}})$ eine allgemeine LR(k)-Maschine zu G mit Startzustand $\hat{q}_s$, und sei $(\tilde{M}, \tau)$ der daraus konstruierte allgemeine LR(k)-Kellertransduktor. Dann gilt:

(a) $\tilde{M}$ erreicht ausschließlich Konfigurationen $\$\varphi \mid z\$$ mit $\varphi = \hat{q}_0 X_1 \hat{q}_1 \ldots X_n \hat{q}_n$ für ein $n \geq 0$, $X_1, \ldots, X_n \in V$ und $\hat{q}_i = \mathrm{GOTO}_{\hat{M}}(\hat{q}_s, X_1 \ldots X_i)$ für $0 \leq i \leq n$.

(b) $X_1 \ldots X_n$ ist ein lebensfähiges Präfix von G'.

Beweis: Induktion über die Länge l der zu $\$\varphi \mid z\$$ führenden Berechnung von $\tilde{M}$. Die Gültigkeit von (a) ist anhand der Struktur der Aktionen von $\tilde{M}$ offensichtlich. Für (b) genügt der Hinweis, daß $L(\hat{M}) = \mathrm{VP}(G')$ und jeder Zustand von $\hat{M}$ Endzustand ist. □

Mit den Bezeichnungen aus dem Lemma sagen wir auch, daß von $\$\varphi \mid z\$$ oder auch φ die Zeichenfolge $X_1 \ldots X_n$ *buchstabiert* wird.

Lemma 3.6 (Verhaltensgleichheit allgemeiner LR(k)-Kellertransduktoren)
Seien $(\hat{M}, \mathrm{RC}_{\hat{M}})$ und $(\hat{M}', \mathrm{RC}_{\hat{M}'})$ allgemeine LR(k)-Maschinen zu G und $(\tilde{M}, \tau)$ respektive $(\tilde{M}', \tau')$ die daraus konstruierten allgemeinen LR(k)-Kellertransduktoren. Dann verhält sich $(\tilde{M}, \tau)$ wie $(\tilde{M}', \tau')$.

Beweis: Wir geben eine totale Funktion *explain* an, die erreichbare Konfigurationen von $\tilde{M}$ in Konfigurationen von $\tilde{M}'$ abbildet, und weisen nach, daß *explain* die geforderten Eigenschaften[23] aufweist. Es habe $\hat{M}$ den Startzustand $\hat{q}_s$ und $\hat{M}'$ den Startzustand $\hat{q}'_s$.

Laut vorangehendem Lemma ist in jeder erreichbaren Konfiguration $\$\varphi\,|\,z\$$ von $\tilde{M}$ der Kellerinhalt φ von der Form $\hat{q}_0 X_1 \hat{q}_1 \ldots X_n \hat{q}_n$ für ein $n \geq 0$, $X_1, \ldots, X_n \in V$ und $\hat{q}_i = \mathrm{GOTO}_{\hat{M}}(\hat{q}_s, X_1 \ldots X_i)$ für $0 \leq i \leq n$, und $X_1 \ldots X_n \in \mathrm{VP}(G') = L(\hat{M}) = L(\hat{M}')$. Mit $\hat{q}'_i = \mathrm{GOTO}_{\hat{M}'}(\hat{q}'_s, X_1 \ldots X_i)$ für $0 \leq i \leq n$ sei dann

$$\mathit{explain}(\$\hat{q}_0 X_1 \hat{q}_1 \ldots X_n \hat{q}_n\,|\,z\$) = \$\hat{q}'_0 X_1 \hat{q}'_1 \ldots X_n \hat{q}'_n\,|\,z\$ \;.$$

explain weist die behaupteten Eigenschaften auf:

1. (Startkonfigurationen): Zu $\$\varphi\,|\,z\$$ Startkonfiguration von $\tilde{M}$ ist

   $$\mathit{explain}(\$\varphi\,|\,z\$) = \mathit{explain}(\$\hat{q}_s\,|\,z\$) = \$\hat{q}'_s\,|\,z\$$$

   Startkonfiguration von $\tilde{M}'$.

3. (Endkonfigurationen):

   $$\begin{aligned}
   \$\varphi\,|\,z\$ \quad &\text{ist Endkonfiguration von } \tilde{M} && \Leftrightarrow \\
   \$\varphi\,|\,z\$ &= \$\hat{q}_s S\ \mathrm{GOTO}_{\hat{M}}(\hat{q}_s, S)\,|\,\$ && \Rightarrow \\
   \mathit{explain}(\$\varphi\,|\,z\$) &= \$\hat{q}'_s S\ \mathrm{GOTO}_{\hat{M}'}(\hat{q}'_s, S)\,|\,\$ && \Leftrightarrow \\
   \mathit{explain}(\$\varphi\,|\,z\$) \quad &\text{ist Endkonfiguration von } \tilde{M}' \,.
   \end{aligned}$$

2. (Übergänge): Maßgeblich für die auf die Konfigurationen $\$\varphi\,|\,z\$ = \$\hat{q}_0 X_1 \hat{q}_1 \ldots X_n \hat{q}_n\,|\,z\$$ von $\tilde{M}$ anwendbaren Aktionen sind nach Konstruktion die Rechtskontextpaare $\mathrm{RC}_{\hat{M}}(\hat{q}_i)$ für $0 \leq i \leq n$. Da aber $\hat{M}$ und $\hat{M}'$ beide nach Voraussetzung LR(k)-Maschinen zu G sind, ist $\mathrm{RC}_{\hat{M}}(\hat{q}_i) = \{X\,|\,u : X \in V',\ X_1 \ldots X_i X \in \mathrm{VP}(G'),\ u \in \mathrm{RC}(X_1 \ldots X_i X)\} = \mathrm{RC}_{\hat{M}'}(\hat{q}'_i)$ für $0 \leq i \leq n$ erfüllt. Gilt also

   $$\$\varphi\,|\,z\$ \overset{r}{\Longrightarrow} \$\psi\,|\,x\$ \quad \text{in } \tilde{M}\,,$$

   so muß $\tilde{M}'$ eine Aktion r' gleichen „Typs“ wie r (d. h. eine Shift-Aktion desselben Symbols bzw. eine Reduce-Aktion nach derselben Regel von G) besitzen, die auf

[23] Siehe S. 27.

$explain(\$\varphi \mid z\$)$ anwendbar ist. Und die Gültigkeit von

$$explain(\$\varphi \mid z\$) = \$\varphi' \mid z\$ \overset{r'}{\Longrightarrow} \$\psi' \mid x\$ = explain(\$\psi \mid x\$) \quad \text{in } \tilde{M}'$$
$$\text{und} \quad \tau(r) = \tau'(r')$$

ist leicht zu überprüfen. □

Aus Symmetriegründen folgt sofort:

Korollar 3.7 (Verhaltensgleichheit allgemeiner LR(k)-Kellertransduktoren)
Sind $(\tilde{M}, \tau)$ und $(\tilde{M}', \tau')$ zwei allgemeine LR(k)-Kellertransduktoren zu G, dann verhält sich $(\tilde{M}, \tau)$ stark wie $(\tilde{M}', \tau')$ und umgekehrt.

Insbesondere verhält sich jeder allgemeine LR(k)-Kellertransduktor $(\tilde{M}, \tau)$ zu G stark wie der kanonische LR(k)-Parser $(\tilde{M}_c, \tau_c)$ zu G. □

Theorem 3.8 (LR(k)-Parser-Eigenschaft allgemeiner LR(k)-Kellertransduktoren)
Jeder allgemeine LR(k)-Kellertransduktor $(\tilde{M}, \tau)$ zu G ist ein LR(k)-Parser zu G.

Beweis: Das vorangehende Korollar über die Verhaltensgleichheit allgemeiner LR(k)-Kellertransduktoren beinhaltet die Aussage, daß $(\tilde{M}, \tau)$ sich stark wie $(\tilde{M}_c, \tau_c)$ zu G verhält. Und $(\tilde{M}_c, \tau_c)$ *ist* ein LR(k)-Parser zu G, verhält sich also stark wie der Shift-Reduce-Parser $(\tilde{M}_{sr}, \tau_{sr})$ zu G (aber nicht umgekehrt). Dann verhält sich transitiv auch $(\tilde{M}, \tau)$ stark wie $(\tilde{M}_{sr}, \tau_{sr})$.

Daß darüber hinaus die Aktionen von $\tilde{M}$ korrekt beschränkt sind bezüglich der Länge des Lookaheads, ist leicht einzusehen. □

Allgemeine LR(k)-Kellertransduktoren zu G dürfen wir also mit Recht *allgemeine LR(k)-Parser zu G* nennen.

Lemma 3.9 (Typübertragung von Nichtdeterminismen)
Seien $(\tilde{M}, \tau)$ und $(\tilde{M}', \tau')$ allgemeine LR(k)-Parser zu G. Ist $\tilde{M}$ nichtdeterministisch in zwei Aktionen r_1 und r_2, so existieren in $\tilde{M}'$ Aktionen r_1' desselben „Typs" wie r_1 und r_2' desselben „Typs" wie r_2, in denen auch $\tilde{M}'$ nichtdeterministisch ist.

Beweis: Sei $(\tilde{M}, \tau)$ aus der allgemeinen LR(k)-Maschine $(\hat{M}, RC_{\hat{M}})$ konstruiert, und $\tilde{M}$ weise einen Nichtdeterminismus in den Aktionen r_1 und r_2 auf. Aufgrund der Uniformität der Lookaheads in allgemeinen LR(k)-Parsern müssen die Lookahead-Anteile der linken Seiten von r_1 und r_2 identisch sein. Der Nichtdeterminismus in r_1 und r_2 ist also Folge der Tatsache, daß auf den linken Seiten von r_1 und r_2 der eine Kelleranteil Suffix des anderen ist.

Nun enthält $\hat{M}$ keine unerreichbaren Zustände, und nach Konstruktion muß ein lebensfähiges Präfix γ von G' existieren sowie eine Konfiguration $\$\varphi \mid z\$$ von $\tilde{M}$, die γ buchstabiert, auf die sowohl r_1 wie auch r_2 anwendbar sind. Wie bereits im Beweis des Lemmas 3.6 über die Verhaltensgleichheit allgemeiner LR(k)-Kellertransduktoren ausgeführt, existiert dann aber eine Aktion r_1' von $\tilde{M}'$ desselben „Typs“ wie r_1 und ebenso eine Aktion r_2' von $\tilde{M}'$ desselben „Typs“ wie r_2, die beide auf diejenige Konfiguration $\$\varphi' \mid z\$$ von $\tilde{M}'$ anwendbar sind, welche ebenfalls γ buchstabiert. Und r_1' und r_2' sind verschiedene Aktionen (und mithin $\tilde{M}'$ nichtdeterministisch in r_1' und r_2'), da sie sich bereits über ihren „Typ“ unterscheiden lassen, denn offensichtlich können in allgemeinen LR(k)-Parsern zwei Aktionen, die einen Nichtdeterminismus aufweisen, nicht beide Shift-Aktionen und auch nicht beide Reduce-Aktionen nach derselben Regel von G sein. □

Theorem 3.10 (Determinismus allgemeiner LR(k)-Parser)
Ein allgemeiner LR(k)-Parser $(\tilde{M},\tau)$ zu G ist genau dann deterministisch, wenn G eine LR(k)-Grammatik ist.

Beweis: Wir erinnern uns, daß $G = (V,T,P,S)$ reduziert ist und $S \Rightarrow^+ S$ nicht gilt.

Der kanonische LR(k)-Parser $(\tilde{M}_c,\tau_c)$ zu G (genauer: davon $\tilde{M}_c$) ist, wie in den Grundlagen präsentiert wurde, bekanntlich genau dann deterministisch, wenn G eine LR(k)-Grammatik ist.

Und ferner kann, in beiden Richtungen auf $(\tilde{M}_c,\tau_c)$ und $(\tilde{M},\tau)$ angewandt, mit Hilfe des obigen Lemmas 3.9 über die Typübertragung von Nichtdeterminismen geschlossen werden, daß $\tilde{M}_c$ genau dann deterministisch ist, wenn $\tilde{M}$ deterministisch ist. □

4 Minimale allgemeine LR(k)-Maschinen und -Parser

In den zurückliegenden Seiten haben wir nach der Rekapitulation der Standardkonstruktion des kanonischen LR(k)-Parsers zur gegebenen Grammatik G dieses Konstruktionsverfahren kompatibel verallgemeinert in der Weise, daß als wesentliche Hilfsmaschinerie, auf deren Grundlage der resultierende Kellertransduktor konstruiert wird, nicht mehr nur die spezielle kanonische LR(k)-Maschine zugelassen wird, sondern eine im allgemeinen unendlich große Familie von geeigneten DEAs mit offengelassener Struktur, aber passender beigeordneter Zuordnungsvorschrift von Rechtskontextinformation an die jeweiligen DEA-Zustände. Die Mitglieder dieser Familie wurden „allgemeine LR(k)-Maschinen zu G“ genannt. Und schließlich haben wir zeigen können, daß ein aus einer allgemeinen LR(k)-Maschine konstruierter sogenannter „allgemeiner Kellertransduktor zu G“ ein vollwertiger LR(k)-Parser zu G ist und daher die Bezeichnung „allgemeiner LR(k)-Parser zu G“ verdient.

Schon aus anwendungsorientierten Gründen liegt es nun nahe, sich für „möglichst kleine“ Exemplare der allgemeinen LR(k)-Parser zu interessieren.[24] Und intuitiv ist klar, daß die Größe eines allgemeinen LR(k)-Parsers abhängig ist von der Anzahl der Zustände der allgemeinen LR(k)-Maschine, aus der er konstruiert ist. Daher rückt in den Mittelpunkt nun die Frage, ob wir eine allgemeine LR(k)-Maschine minimaler Zustandsanzahl bestimmen können.

4.1 Minimale allgemeine LR(k)-Maschinen

Eine allgemeine LR(k)-Maschine $(\hat{M}, \mathrm{RC}_{\hat{M}})$ zu G weist die beiden Eigenschaften auf, (i) genau die lebensfähigen Präfizes von G' zu erkennen (Außensicht) und (ii) zu diesem lebensfähigen Präfizes an den Zuständen von $\hat{M}$ korrekte Rechtskontextinformation zu berechnen (Innensicht). Wie erwähnt, greift es, die Minimierung von $(\hat{M}, \mathrm{RC}_{\hat{M}})$ betreffend, zu kurz, lediglich die Außensicht zu berücksichtigen und $\hat{M}$ mit bekannten Techniken[25] als

[24] Es sei ins Gedächtnis gerufen, daß wir (von pathologischen Fällen abgesehen) die Größe eines Kellertransduktors als die Summe der Längen der linken und rechten Seiten seiner Aktionen ansehen.

[25] Siehe z. B. (Hopcroft 1971).

DEA zu minimieren, da so Zustände ohne Rücksicht auf die dort berechnete Rechtskontextinformation zusammengefaßt werden: Das damit verbundene Konzept, zwei Zustände $\hat{q}$ und $\hat{q}'$ dann als äquivalent anzusehen, wenn die „ab $\hat{q}$“ bzw. „ab $\hat{q}'$“ erkannten Sprachen identisch sind, ist zu grob. Vielmehr dürfen $\hat{q}$ und $\hat{q}'$ dann nicht mehr als äquivalent angesehen werden, wenn die an $\hat{q}$ und $\hat{q}'$ berechnete Rechtskontextinformation $\mathrm{RC}_{\hat{M}}(\hat{q})$ bzw. $\mathrm{RC}_{\hat{M}}(\hat{q}')$ differiert, denn unterschiedliche Rechtskontextinformationen bedeuten unterschiedliche daraus erzeugte Parseraktionen, und dem DEA $\hat{M}'$, in dem gegenüber $\hat{M}$ die Zustände $\hat{q}$ und $\hat{q}'$ zusammengefaßt sind, kann keine Rechtskontextfunktion $\mathrm{RC}_{\hat{M}'}$ mehr beigeordnet werden, so daß $(\hat{M}', \mathrm{RC}_{\hat{M}'})$ immer noch eine allgemeine LR(k)-Maschine zu G wäre.

Zusammengefaßt ergibt sich, daß in $(\hat{M}, \mathrm{RC}_{\hat{M}})$ genau dann zwei Zustände $\hat{q}$ und $\hat{q}'$ als äquivalent angesehen werden dürfen – geschrieben $\hat{q} \sim \hat{q}'$ –, wenn für alle $w \in T'^*$ gilt:

> $\hat{q}_1 = \mathrm{GOTO}_{\hat{M}}(\hat{q}, w)$ existiert genau dann, wenn $\hat{q}'_1 = \mathrm{GOTO}_{\hat{M}}(\hat{q}', w)$ existiert,
> und dann ist $\mathrm{RC}_{\hat{M}}(\hat{q}_1) = \mathrm{RC}_{\hat{M}}(\hat{q}'_1)$.

Induktiv ist $\hat{q}$ äquivalent zu $\hat{q}'$, wenn $\mathrm{RC}_{\hat{M}}(\hat{q}) = \mathrm{RC}_{\hat{M}}(\hat{q}')$ ist und für jedes $a \in T'$, für das $\hat{M}$ eine Aktion $\hat{q}a \to \hat{q}_a$ aufweist, genau dann auch eine Aktion $\hat{q}'a \to \hat{q}'_a$ existiert und $\mathrm{RC}_{\hat{M}}(\hat{q}_a) = \mathrm{RC}_{\hat{M}}(\hat{q}'_a)$ ist.

Beispiel: Wie man leicht sieht, sind in der kanonischen LR(1)-Maschine zu $G_{ab\varepsilon}$ als allgemeine LR(1)-Maschine, die in Abbildung 3 gezeigt ist, zunächst die dortigen Zustände 4, 6, 12, 14, 16, 17 und 18 alle paarweise äquivalent. Daraus wiederum ergibt sich, daß auch die Zustände 8, 10, 13 und 15 und dann auch einerseits 5 und 9, andererseits 7 und 11 paarweise äquivalent sind. Das Ergebnis der entsprechenden Zustandszusammenfassungen ist in Abbildung 4 gezeichnet. □

Einen effizienten Algorithmus zur Berechnung der bis auf Zustandsumbenennungen eindeutigen allgemeinen LR(k)-Maschine zu G mit minimaler Zustandsanzahl erhalten wir nun, indem wir im klassischen DEA-Minimierungsalgorithmus von Hopcroft (1971), besser beschrieben von Gries (1973), die initiale Zustandspartition nicht binär gemäß der (hier ohnehin irrelevanten) Endzustandseigenschaft, sondern gemäß der unterschiedlichen Rechtskontextinformationen bestimmen, bevor wir diese Klassifizierung nach dem bekannten Verfahren auf Basis der Übergangsstruktur mit dem Zeitaufwand $O(|T|n\log n)$ zu $\sim$ verfeinern, wobei n die Anzahl der Zustände von $\hat{M}$ ist.[26]

[26] Es sei darauf hingewiesen, daß die Algorithmen in (Hopcroft 1971) und (Gries 1973) etwas zu modifizieren sind, da sie von einem vollständig spezifizierten DEA ausgehen.

Dem Leser mit tieferen Kenntnissen in Automatentheorie mag aufgefallen sein, daß auch der beschriebene modifizierte Minimierungsalgorithmus altbekannt ist: Es ist der Algorithmus zur Minimierung eines (unvollständig spezifizierten) sogenannten Zustands-Ausgabe- oder Moore-Automaten. Dieser ist quasi ein DEA mit Ausgabeinformation an den Zuständen anstatt einer Auszeichnung von Endzuständen und weist daher Antwort- statt Akzeptanzverhalten auf.

Ferner sei noch einmal erinnert an die erwähnte alternative Modellierungsmöglichkeit für allgemeine LR(k)-Maschinen, nämlich die Rechtskontextinformation den *Übergängen* des DEA zuzuordnen: Historisch entspricht diese Variante (unvollständig spezifizierten) sogenannten Übergangs-Ausgabe- oder Mealy-Automaten. Im übrigen stellt sich die beruhigende Tatsache heraus, daß im Vergleich der zwei Modellierungsalternativen die bei beiden Varianten jeweils resultierenden, bis auf Zustandsumbenennungen eindeutigen zustandsminimalen allgemeinen LR(k)-Maschinen strukturell gleichwertig sind; insbesondere weisen sie gleiche Zustandsanzahlen auf und führen zu (bis auf Zustandsbezeichnungen) gleichen allgemeinen LR(k)-Parsern.

Mit den erkannten Bezügen zu klassischer Automatentheorie ergeben sich unmittelbar weitere Ergebnisse über die Struktur der Familie der allgemeinen LR(k)-Maschinen zu einer gegebenen Grammatik G.

Sei $(\hat{M}_\mathrm{c}, \mathrm{RC}_\mathrm{c})$ zu G betrachtet, und $[\hat{q}]$ bedeute $\{\hat{q}' \in \hat{\mathrm{Q}}_\mathrm{c} : \hat{q} \sim \hat{q}'\}$. Dann können wir zunächst festhalten, daß $(\hat{M}_\mathrm{m}, \mathrm{RC}_\mathrm{m})$ mit dem wohldefinierten DEA $\hat{M}_\mathrm{m} = (\hat{\mathrm{Q}}_\mathrm{m}, T', \hat{P}_\mathrm{m}, \hat{q}_{\mathrm{s,m}}, \hat{\mathrm{Q}}_\mathrm{m})$,

$$\begin{aligned}
\hat{\mathrm{Q}}_\mathrm{m} &= \hat{\mathrm{Q}}_\mathrm{c}/\!\sim = \{[\hat{q}] : \hat{q} \in \hat{\mathrm{Q}}_\mathrm{c}\}\,, \\
\hat{P}_\mathrm{m} &= \{[\hat{q}]a \to [\hat{q}'] : a \in T',\ \hat{q}a \to \hat{q}' \in \hat{P}_\mathrm{c}\}\,, \\
\hat{q}_{\mathrm{s,m}} &= [\hat{q}_{\mathrm{s,c}}]
\end{aligned}$$

und der wohldefinierten Abbildung RC_m, die jedem $[\hat{q}] \in \hat{\mathrm{Q}}_\mathrm{m}$ die Rechtskontextpaare

$$\mathrm{RC}_\mathrm{m}([\hat{q}]) = \mathrm{RC}_\mathrm{c}(\hat{q})$$

zuordnet, eine allgemeine LR(k)-Maschine zu G ist, in der keine weiteren Zusammenfassungen äquivalenter Zustände mehr möglich sind. Nebenbei definieren wir $h_\mathrm{m} : \hat{\mathrm{Q}}_\mathrm{c} \to \hat{\mathrm{Q}}_\mathrm{m}$ mittels $h_\mathrm{m}(\hat{q}) = [\hat{q}]$. Sind weiter $(\hat{M}_1, \mathrm{RC}_1)$ und $(\hat{M}_2, \mathrm{RC}_2)$ zwei allgemeine LR(k)-Maschinen zu G mit den Zustandsmengen $\hat{\mathrm{Q}}_1$ resp. $\hat{\mathrm{Q}}_2$ und sind in $\hat{M}_2$ gegenüber $\hat{M}_1$ einige äquivalente Zustände zusammengefaßt, was durch die surjektive Abbildung $h_{12} : \hat{\mathrm{Q}}_1 \to \hat{\mathrm{Q}}_2$ beschrieben werde, dann wollen wir sagen, $(\hat{M}_2, \mathrm{RC}_2)$ *ist ökonomischer als* $(\hat{M}_1, \mathrm{RC}_1)$, da $\hat{M}_2$

(höchstens) weniger Zustände benötigt. Die Automatentheorie[27] sichert nun zu, daß modulo Zustandsumbenennungen die Familie der allgemeinen LR(k)-Maschinen zu G bezüglich der Relation „ist ökonomischer als“ eine partielle Ordnung bildet, in der $(\hat{M}_\mathrm{m}, \mathrm{RC}_\mathrm{m})$ kleinstes Element ist. Wir nennen daher $(\hat{M}_\mathrm{m}, \mathrm{RC}_\mathrm{m})$ *die (zustands)minimale allgemeine LR(k)-Maschine zu G.*

Beispiel: Für die bereits strapazierte Beispielgrammatik $G_{ab\varepsilon} : S \to a\,A \mid b\,B,\ A \to \varepsilon \mid c\,A\,d,\ B \to \varepsilon \mid c\,B\,d$ kennen wir bereits zwei allgemeine LR(k)-Maschinen: die kanonische LR(k)-Maschine $(\hat{M}_\mathrm{c}, \mathrm{RC}_\mathrm{c})$, die in Abbildung 2 dargestellt ist, und die minimale $(\hat{M}_\mathrm{m}, \mathrm{RC}_\mathrm{m})$, die Abbildung 4 zeigt. Zwei weitere allgemeine LR(k)-Maschinen zu G, $(\hat{M}_1, \mathrm{RC}_1)$ und $(\hat{M}_2, \mathrm{RC}_2)$, sind in Abbildung 5 gezeichnet. Es gilt:

$$(\hat{M}_\mathrm{m}, \mathrm{RC}_\mathrm{m}) \quad \text{ist ökonomischer als} \quad (\hat{M}_\mathrm{c}, \mathrm{RC}_\mathrm{c}) \quad \text{ist ökonomischer als} \quad (\hat{M}_2, \mathrm{RC}_2)$$

und

$$(\hat{M}_\mathrm{m}, \mathrm{RC}_\mathrm{m}) \quad \text{ist ökonomischer als} \quad (\hat{M}_1, \mathrm{RC}_1) \quad \text{ist ökonomischer als} \quad (\hat{M}_2, \mathrm{RC}_2)\,,$$

jedoch sind $(\hat{M}_1, \mathrm{RC}_1)$ und $(\hat{M}_\mathrm{c}, \mathrm{RC}_\mathrm{c})$ unvergleichbar. □

4.2 Minimale allgemeine LR(k)-Parser

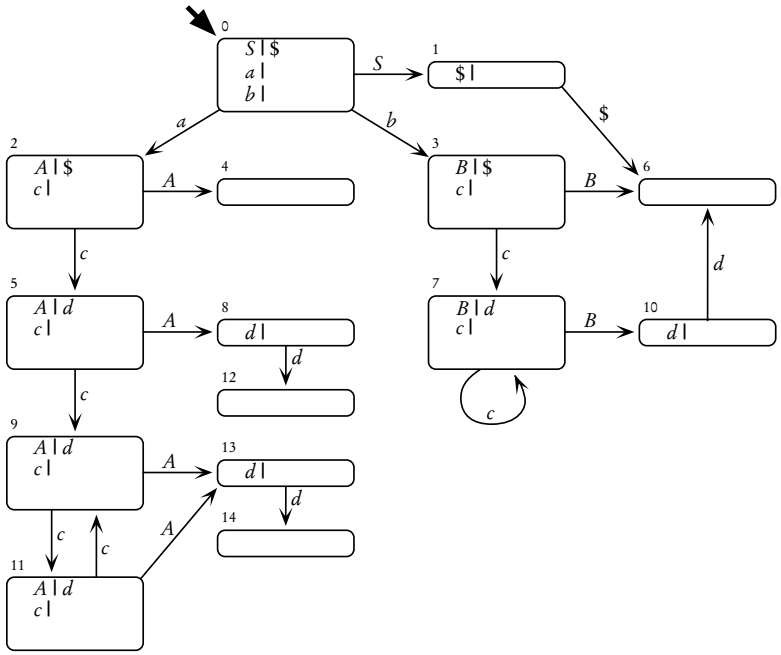
Was bedeutet, übertragen, die beschriebene Ordnungsstruktur der Familie der allgemeinen LR(k)-Maschinen zu G für die entsprechenden allgemeinen LR(k)-Parser zu G ?

Zwischen den Aktionenmengen allgemeiner LR(k)-Parser zu G lassen sich Beziehungen herstellen. Einen Ansatz hierzu bot bereits der Beweis von Lemma 3.6 über die Verhaltensgleichheit allgemeiner Kellertransduktoren, wo die aus $(\hat{M}, \mathrm{RC}_{\hat{M}})$ respektive $(\hat{M}', \mathrm{RC}_{\hat{M}'})$ konstruierten allgemeinen LR(k)-Parser $(\tilde{M}, \tau)$ und $(\tilde{M}', \tau')$ betrachtet wurden; die totale Funktion *explain*, welche erreichbare Konfigurationen von $\tilde{M}$ umrechnete in ebensolche von $\tilde{M}'$, war dort definiert worden durch

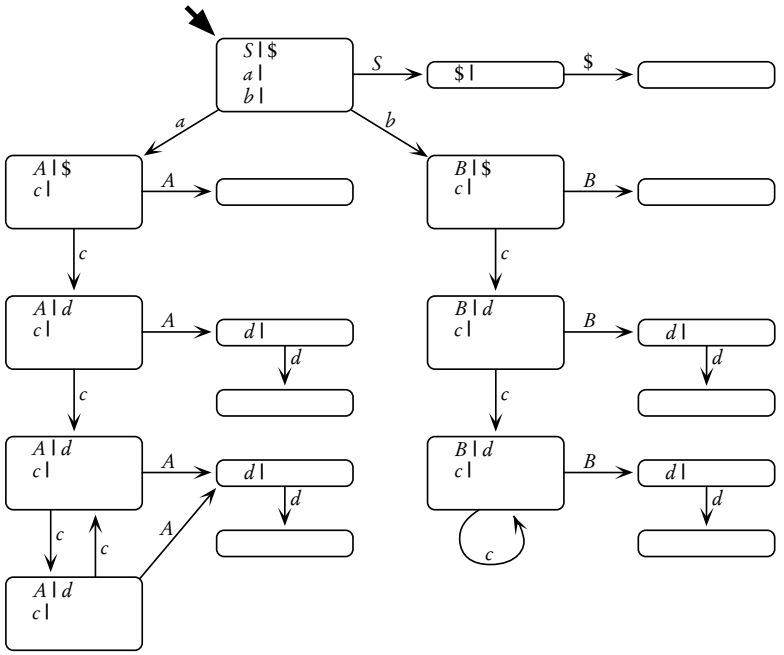
$$\mathit{explain}(\$\hat{q}_0 X_1 \hat{q}_1 \ldots X_n \hat{q}_n \,|\, z\,\$) = \$\hat{q}'_0 X_1 \hat{q}'_1 \ldots X_n \hat{q}'_n \,|\, z\,\$$$

mit $\hat{q}_i = \mathrm{GOTO}_{\hat{M}}(\hat{q}_\mathrm{s}, X_1 \ldots X_i)$ und $\hat{q}'_i = \mathrm{GOTO}_{\hat{M}'}(\hat{q}'_\mathrm{s}, X_1 \ldots X_i)$ für $0 \le i \le n$. Und zweifelsohne ist *explain* bijektiv.

[27] Siehe z. B. (Brauer 1984).



(a) Die allgemeine LR(1)-Maschine $(\hat{M}_1, \mathrm{RC}_1)$ zu $G = G_{ab\varepsilon}$ ist, im Vergleich mit $(\hat{M}_c, \mathrm{RC}_c)$, in ihrer „$a$-Hälfte“ „noch unökonomischer“ gebaut, ihre „b-Hälfte“ ist dagegen „maximal ökonomisch“.



(b) Die allgemeine LR(1)-Maschine $(\hat{M}_2, \mathrm{RC}_2)$ zu $G = G_{ab\varepsilon}$ ist gebaut wie $(\hat{M}_c, \mathrm{RC}_c)$, in ihrer „$a$-Hälfte“ jedoch „noch unökonomischer“.

Abbildung 5: Zwei weitere allgemeine LR(1)-Maschinen zu $G = G_{ab\varepsilon}$

Dann können wir nun zwischen der Aktionenmenge $P_{\tilde{M}}$ von $\tilde{M}$ und der von $\tilde{M}'$, $P_{\tilde{M}'}$, die folgende Relation $C(\tilde{M},\tilde{M}') \subseteq P_{\tilde{M}} \times P_{\tilde{M}'}$ definieren:

$$r\ C(\tilde{M},\tilde{M}')\ r' \quad :\Leftrightarrow \quad \begin{array}{l}\text{es existieren erreichbare Konfigurationen } \$\varphi\,|\,z\$ \text{ und} \\ \$\varphi'\,|\,z'\$ \text{ von } \tilde{M}, \text{ so daß } \$\varphi\,|\,z\$ \overset{r}{\Longrightarrow} \$\varphi'\,|\,z'\$ \text{ in } \tilde{M} \text{ und} \\ \mathit{explain}(\$\varphi\,|\,z\$) \overset{r'}{\Longrightarrow} \mathit{explain}(\$\varphi'\,|\,z'\$) \text{ in } \tilde{M}' \text{ gilt.}\end{array}$$

Offensichtlich ist $C(\tilde{M}',\tilde{M}) = C(\tilde{M},\tilde{M}')^{-1}$.

Damit können wir formulieren:

Lemma 4.1 (Aktionenverwandschaft zwischen allgemeinen LR(k)-Parsern)
Seien $(\tilde{M},\tau)$ und $(\tilde{M}',\tau')$ allgemeine LR(k)-Parser zu G und dazu $C(\tilde{M},\tilde{M}')$ wie soeben definiert. Dann gilt

(a) $C(\tilde{M},\tilde{M}')$ ist links- und rechtstotal.

(b) Zwei Aktionen r und r', für die $r\ C(\tilde{M},\tilde{M}')\ r'$ gilt, sind vom selben „Typ".

(c) Seien r_1, r_2 Aktionen von $\tilde{M}$ und r_1', r_2' Aktionen von $\tilde{M}'$, so daß $r_1\ C(\tilde{M},\tilde{M}')\ r_1'$ und $r_2\ C(\tilde{M},\tilde{M}')\ r_2'$ erfüllt ist. Dann ist $\tilde{M}$ nichtdeterministisch in r_1 und r_2 genau dann, wenn $\tilde{M}'$ in r_1' und r_2' nichtdeterministisch ist.

Beweis:
(a) Bekanntlich verhält sich $(\tilde{M},\tau)$ wie $(\tilde{M}',\tau')$ und umgekehrt, und weder $\tilde{M}$ noch $\tilde{M}'$ besitzen nutzlose Aktionen.

(b) Siehe den Beweis des Lemmas 3.6 über die Verhaltensgleichheit allgemeiner Kellertransduktoren.

(c) Siehe den Beweis des Lemmas 3.9 über die Typübertragung von Nichtdeterminismen.

□

Für die aus vergleichbaren allgemeinen LR(k)-Maschinen zu G konstruierten allgemeinen LR(k)-Parser ergibt sich dann das

Lemma 4.2
Zu G seien $(\hat{M},\mathrm{RC}_{\hat{M}})$ und $(\hat{M}',\mathrm{RC}_{\hat{M}'})$ allgemeine LR(k)-Maschinen und $(\tilde{M},\tau)$ resp. $(\tilde{M}',\tau')$ die daraus konstruierten allgemeinen LR(k)-Parser.

Ist $(\hat{M}',\mathrm{RC}_{\hat{M}'})$ ökonomischer als $(\hat{M},\mathrm{RC}_{\hat{M}})$, so gilt:

(a) $|(\tilde{M}', \tau')| = |\tilde{M}'| \leq |\tilde{M}| = |(\tilde{M}, \tau)|$.

(b) $(\tilde{M}', \tau')$ *weist höchstens weniger Nichtdeterminismen auf als* $(\tilde{M}, \tau)$.

Beweis: Drücke h die Zusammenfassung von $\hat{M}$- zu $\hat{M}'$-Zuständen aus. Mit Hilfe von h ist $C(\tilde{M}, \tilde{M}')$ gerade gegeben durch

$$
\begin{array}{rcl}
 & & r = \hat{q}_0 X_1 \hat{q}_1 \ldots X_n \hat{q}_n \,|\, u \to \hat{q}_0 X \hat{q}'_1 \,|\, u' \quad \text{und} \\
r\ C(\tilde{M}, \tilde{M}')\ r' & \Leftrightarrow & r' = h(\hat{q}_0) X_1 h(\hat{q}_1) \ldots X_n h(\hat{q}_n) \,|\, u \to h(\hat{q}_0) X h(\hat{q}'_1) \,|\, u' \\
 & & \text{mit den offensichtlichen Quantorisierungen.}
\end{array}
$$

Und es ist leicht einzusehen, daß $C(\tilde{M}, \tilde{M}')$ rechtseindeutig, d. h. eine (rechtstotale) *Abbildung* ist, woraus, zusammen mit den Punkten (b) und (c) des vorangehenden Lemmas 4.1, beide Aussagen des Theorems sofort herzuleiten sind. □

Insbesondere ergeben sich aus diesem Lemma für den aus der minimalen allgemeinen LR(k)-Maschine zu G konstruierten allgemeinen LR(k)-Parser die folgenden befriedigenden Eigenschaften:

Theorem 4.3 (Eigenschaften eines minimalen allgemeinen LR(k)-Parsers)
Zu G *sei* $(\tilde{M}_\mathrm{m}, \tau_\mathrm{m})$ *der aus der minimalen allgemeinen LR(k)-Maschine* $(\hat{M}_\mathrm{m}, \mathrm{RC}_\mathrm{m})$ *konstruierte allgemeine LR(k)-Parser. Dann gilt*

(a) Die Größe von $(\tilde{M}_\mathrm{m}, \tau_\mathrm{m})$, $|\tilde{M}_\mathrm{m}|$, *ist unter allen allgemeinen LR(k)-Parsern zu* G *minimal.*

(b) $(\tilde{M}_\mathrm{m}, \tau_\mathrm{m})$ *weist unter allen allgemeinen LR(k)-Parsern zu* G *die geringste Zahl an Nichtdeterminismen auf.*

($(\tilde{M}_\mathrm{m}, \tau_\mathrm{m})$ wird daher *minimaler allgemeiner LR(k)-Parser zu* G genannt.)

Beweis: $(\hat{M}', \mathrm{RC}_{\hat{M}'}) = (\hat{M}_\mathrm{m}, \mathrm{RC}_\mathrm{m})$ und $(\tilde{M}', \tau') = (\tilde{M}_\mathrm{m}, \tau_\mathrm{m})$ im vorigen Lemma. □

Beispiel: Mit Zuständen von $\hat{M}_\mathrm{m}$ zu Zahlen abgekürzt gemäß den Annotationen in Abbildung 4 weist der minimale allgemeine LR(1)-Parser $(\tilde{M}_\mathrm{m}, \tau_\mathrm{m})$ zu $G_{ab\varepsilon}$ die folgenden

Aktionen auf:

$$
\begin{aligned}
r'_1 &= 0\,a\,2\,A\,4\,|\,\$ \to 0\,S\,1\,|\,\$\,, & \tau_{\mathrm{m}}(r'_1) &= S \to a\,A\,. \\
r'_2 &= 0\,b\,3\,B\,4\,|\,\$ \to 0\,S\,1\,|\,\$\,, & \tau_{\mathrm{m}}(r'_2) &= S \to b\,B\,. \\
r'_3 &= 2\,|\,\$ \to 2\,A\,4\,|\,\$\,, & \tau_{\mathrm{m}}(r'_3) &= A \to \varepsilon\,. \\
r'_4 &= 2\,c\,5\,A\,7\,d\,4\,|\,\$ \to 2\,A\,4\,|\,\$\,, & \tau_{\mathrm{m}}(r'_4) &= A \to c\,A\,d\,. \\
r'_5 &= 3\,|\,\$ \to 3\,B\,4\,|\,\$\,, & \tau_{\mathrm{m}}(r'_5) &= B \to \varepsilon\,. \\
r'_6 &= 3\,c\,6\,B\,7\,d\,4\,|\,\$ \to 3\,B\,4\,|\,\$\,, & \tau_{\mathrm{m}}(r'_6) &= B \to c\,B\,d\,. \\
r'_7 &= 5\,|\,d \to 5\,A\,7\,|\,d\,, & \tau_{\mathrm{m}}(r'_7) &= A \to \varepsilon\,. \\
r'_8 &= 5\,c\,5\,A\,7\,d\,4\,|\,d \to 5\,A\,7\,|\,d\,, & \tau_{\mathrm{m}}(r'_8) &= A \to c\,A\,d\,. \\
r'_9 &= 6\,|\,d \to 6\,B\,7\,|\,d\,, & \tau_{\mathrm{m}}(r'_9) &= B \to \varepsilon\,. \\
r'_{10} &= 6\,c\,6\,B\,7\,d\,4\,|\,d \to 6\,B\,7\,|\,d\,, & \tau_{\mathrm{m}}(r'_{10}) &= B \to c\,B\,d\,. \\
s'_1 &= 0\,|\,a \to 0\,a\,2\,|\,, & \tau_{\mathrm{m}}(s'_1) &= \varepsilon\,. \\
s'_2 &= 0\,|\,b \to 0\,b\,3\,|\,, & \tau_{\mathrm{m}}(s'_2) &= \varepsilon\,. \\
s'_3 &= 2\,|\,c \to 2\,c\,5\,|\,, & \tau_{\mathrm{m}}(s'_3) &= \varepsilon\,. \\
s'_4 &= 3\,|\,c \to 3\,c\,6\,|\,, & \tau_{\mathrm{m}}(s'_4) &= \varepsilon\,. \\
s'_5 &= 5\,|\,c \to 5\,c\,5\,|\,, & \tau_{\mathrm{m}}(s'_5) &= \varepsilon\,. \\
s'_6 &= 6\,|\,c \to 6\,c\,6\,|\,, & \tau_{\mathrm{m}}(s'_6) &= \varepsilon\,. \\
s'_7 &= 7\,|\,d \to 7\,d\,4\,|\,, & \tau_{\mathrm{m}}(s'_7) &= \varepsilon\,.
\end{aligned}
$$

Die minimierende Zustandszusammenfassung h_{m} von $(\hat{M}_{\mathrm{c}}, \mathrm{RC}_{\mathrm{c}})$ aus Abbildung 3 auf $(\hat{M}_{\mathrm{m}}, \mathrm{RC}_{\mathrm{m}})$ ist

$\hat{q}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$h_{\mathrm{m}}(\hat{q})$	0	1	2	3	4	5	4	6	7	5	7	6	4	7	4	7	4	4	4

Mit den $\tilde{M}_{\mathrm{c}}$-Aktionenbezeichnungen von Seite 37 ist

$$
\begin{aligned}
C(\tilde{M}_{\mathrm{c}}, \tilde{M}_{\mathrm{m}}) = \{\, & (r_1, r'_1),\ (r_2, r'_2),\ (r_3, r'_3),\ (r_4, r'_4),\ (r_5, r'_5),\ (r_6, r'_6),\ (r_7, r'_7), \\
& (r_8, r'_8),\ (r_9, r'_9),\ (r_{10}, r'_{10}),\ (r_{11}, r'_7),\ (r_{12}, r'_8),\ (r_{13}, r'_9),\ (r_{14}, r'_{10}), \\
& (s_1, s'_1),\ (s_2, s'_2),\ (s_3, s'_3),\ (s_4, s'_4),\ (s_5, s'_5),\ (s_6, s'_6), \\
& (s_7, s'_7),\ (s_8, s'_5),\ (s_9, s'_7),\ (s_{10}, s'_6),\ (s_{11}, s'_7),\ (s_{12}, s'_7)\,\}
\end{aligned}
$$

eine Abbildung.

Dagegen waren $(\hat{M}_1, \mathrm{RC}_1)$ aus Abbildung 5 und $(\hat{M}_c, \mathrm{RC}_c)$ unvergleichbar. Einige Aktionen des aus $(\hat{M}_1, \mathrm{RC}_1)$ konstruierten allgemeinen LR(1)-Parsers $(\tilde{M}_1, \tau_1)$ sind (mit Zahlen für $\hat{M}_1$-Zustände gemäß Abbildung):

$$r_1'' = 9 \,|\, d \to 9\, A\, 13 \,|\, d\,, \qquad \tau_1(r_1'') = A \to \varepsilon\,.$$
$$r_2'' = 11 \,|\, d \to 11\, A\, 13 \,|\, d\,, \qquad \tau_1(r_2'') = A \to \varepsilon\,.$$
$$r_3'' = 7 \,|\, d \to 7\, B\, 10 \,|\, d\,, \qquad \tau_1(r_3'') = B \to \varepsilon\,.$$

Ausschnittsweise ist

$$C(\tilde{M}_c, \tilde{M}_1) = \{\ldots, (r_9, r_3''), \ldots, (r_{11}, r_1''), (r_{11}, r_2''), \ldots, (r_{13}, r_3''), \ldots\}$$

und weder $C(\tilde{M}_c, \tilde{M}_1)$ noch $C(\tilde{M}_1, \tilde{M}_c) = C(\tilde{M}_c, \tilde{M}_1)^{-1}$ eine Abbildung. □

5 Minimierung kanonischer LR(k)-Maschinen als Abstraktion von Itemmengen

Es ist bereits erwähnt worden, daß ein k-Item $[A \to \alpha \cdot \beta, y]$ im Rahmen seiner Rolle in der kanonischen nichtdeterministischen LR(k)-Maschine M_c zu einer Grammatik G – und dann auch in dem deterministischen Pendant $\hat{M}_c$ – in den „linken“ Komponenten A und α quasi seine „individuelle Geschichte“ codiert. Diese Beobachtung legt nahe, zur frühzeitigen Vermeidung von Redundanz schon bei der Konstruktion einer (dann noch zu minimierenden) allgemeinen LR(k)-Maschine eine Projektion jedes $[A \to \alpha \cdot \beta, y]$ auf (β, y) von vorneherein zu berücksichtigen. Als Voruntersuchung zum folgenden Abschnitt 6, in dem eine solche Projektionskonstruktion detailliert analysiert wird, wollen wir hier zunächst einige kurze Überlegungen anstellen, inwieweit sich die Wirkung der Zustandsminimierung in der kanonischen LR(k)-Maschine zustandslokal überhaupt als eine Abstraktion der Mengen LR(k)-gültiger Items verstehen läßt. Es wird sich herausstellen, daß dieses nur approximativ gelingt.

Wie wir an dieser Stelle wissen, existiert zur kanonischen LR(k)-Maschine $\hat{M}_c$ (bzw. $(\hat{M}_c, \mathrm{RC}_c)$) zu G stets eine Funktion h_m, die die Zusammenfassung von äquivalenten $\hat{M}_c$-Zuständen zu Zuständen der minimalen allgemeinen LR(k)-Maschine $(\hat{M}_m, \mathrm{RC}_m)$ zu G reflektiert. Diese Zustandsäquivalenzrelation sei $\equiv_m$, d. h.

$$\hat{q} \equiv_m \hat{q}' \quad \Leftrightarrow \quad h_m(\hat{q}) = h_m(\hat{q}') \; .$$

Welche Rückschlüsse erlaubt eine solche Äquivalenz zwischen zwei Itemmengen $\hat{q}$, $\hat{q}'$ auf deren jeweils enthaltene k-Items ?

Die Äquivalenz $\hat{q} \equiv_m \hat{q}'$ besagt, daß von $\hat{q}$ und $\hat{q}'$ aus gleiche Fortsetzungen lebensfähiger Präfizes erkannt werden (Außensicht) und zudem die zugehörigen Rechtskontextinformationen übereinstimmen (Innensicht). Innerhalb der Itemmengenkonstruktion mit dem Ergebnis $\hat{M}_c$ haben von jedem Item $[A \to \alpha \cdot \beta, y]$ aber nur dessen „in die Zukunft“ gerichteten, „rechten“ Komponenten β und y Einfluß auf die beiden Äquivalenzkriterien für $\hat{q} \equiv_m \hat{q}'$; die „rückwärts“ gerichteten Komponenten A und α sind diesbezüglich redundanter Ballast.

Definieren wir entsprechend eine „Rechts“-Projektionsabbildung $proj_{\mathrm{r}}$ durch

$$proj_{\mathrm{r}}(\hat{q}) = \{(\beta, y) \,:\, \text{es ex. ein } k\text{-Item } [A \to \alpha \cdot \beta, y] \in \hat{q}\}\,,$$

so ist nachvollziehbar, daß $proj_{\mathrm{r}}$ als die Zustandszusammenfassungsfunktion verwendet werden kann, mit der $(\hat{M}_{\mathrm{c}}, \mathrm{RC}_{\mathrm{c}})$ zu einer allgemeinen LR(k)-Maschine $(\hat{M}_{\mathrm{r}}, \mathrm{RC}_{\mathrm{r}}) = proj_{\mathrm{r}}(\hat{M}_{\mathrm{c}}, \mathrm{RC}_{\mathrm{c}})$ reduziert werden kann, die in naheliegender Weise aus $(\hat{M}_{\mathrm{c}}, \mathrm{RC}_{\mathrm{c}})$ gewonnen wird und – wie wir sagten – ökonomischer ist als $(\hat{M}_{\mathrm{c}}, \mathrm{RC}_{\mathrm{c}})$. (Der direkten Konstruktion dieser speziellen Maschine widmet sich der anschließende Abschnitt 6.)

Dieser große Kompaktifizierungsfortschritt läßt sich noch geringfügig weiter treiben: Wir können sogar, da in $\hat{M}_{\mathrm{c}}$ ein k-Item $[A \to \alpha \cdot \beta, y] \in \hat{q}$ dann keine Wirkung auf gültige Fortsetzungen lebensfähiger Präfizes ab $\hat{q}$ und zugehörige Rechtskontexte hat, wenn $\beta = \varepsilon$ ist, eine noch stärker kompaktifizierende Projektionsabbildung $proj_{\mathrm{r}\varepsilon}$ erklären durch

$$proj_{\mathrm{r}\varepsilon}(\hat{q}) = \{(\beta, y) \,:\, \text{es ex. ein } k\text{-Item } [A \to \alpha \cdot \beta, y] \in \hat{q} \text{ und } \beta \neq \varepsilon\}\,,$$

so daß selbst $(\hat{M}_{\mathrm{r}\varepsilon}, \mathrm{RC}_{\mathrm{r}\varepsilon}) = proj_{\mathrm{r}\varepsilon}(\hat{M}_{\mathrm{c}}, \mathrm{RC}_{\mathrm{c}})$ eine allgemeine LR(k)-Maschine zu G ist, die ökonomischer ist als $(\hat{M}_{\mathrm{c}}, \mathrm{RC}_{\mathrm{c}})$ und $(\hat{M}_{\mathrm{r}}, \mathrm{RC}_{\mathrm{r}})$.

Sind die Äquivalenzrelationen $\equiv_{\mathrm{r}}$ und $\equiv_{\mathrm{r}\varepsilon}$ auf den Zuständen von $\hat{M}_{\mathrm{c}}$ erklärt durch

$$\begin{aligned} \hat{q} \equiv_{\mathrm{r}} \hat{q}' \quad &\Leftrightarrow \quad proj_{\mathrm{r}}(\hat{q}) = proj_{\mathrm{r}}(\hat{q}') \qquad \text{und} \\ \hat{q} \equiv_{\mathrm{r}\varepsilon} \hat{q}' \quad &\Leftrightarrow \quad proj_{\mathrm{r}\varepsilon}(\hat{q}) = proj_{\mathrm{r}\varepsilon}(\hat{q}')\,, \end{aligned}$$

so ist offenkundig $\equiv_{\mathrm{m}} \supseteq \equiv_{\mathrm{r}\varepsilon} \supseteq \equiv_{\mathrm{r}}$, und für die in Kürze präsentierte Beispielgrammatik $G_{\mathrm{r}\varepsilon}$ ist $\equiv_{\mathrm{r}\varepsilon} \supsetneq \equiv_{\mathrm{r}}$. Die Frage liegt nun nahe, ob $proj_{\mathrm{r}\varepsilon}$ eine „echte“ Approximation von h_{m} bleibt oder ob sogar stets $\equiv_{\mathrm{m}} = \equiv_{\mathrm{r}\varepsilon}$ gilt. Wir werden jedoch einsehen müssen, daß die von der minimierenden Zustandszusammenfassung h_{m} charakterisierte Faktorisierung über die von $proj_{\mathrm{r}\varepsilon}$ i. a. hinausgeht.

Das in Abbildung 6 gezeigte Grammatikschema $G_{\mathrm{r}\varepsilon}$ skizziert Grammatiken, für die schon für den einfachen Fall $k = 0$, in dem Lookahead-Anteile in den k-Items keine Rolle spielen, $(\hat{M}_{\mathrm{r}\varepsilon}, \mathrm{RC}_{\mathrm{r}\varepsilon})$ nicht die minimale Zustandsanzahl aufweist, d. h. $\equiv_{\mathrm{m}} \supsetneq \equiv_{\mathrm{r}\varepsilon}$ ist.

Statt dessen weist uns eine Betrachtung der beiden $\equiv_{\mathrm{m}}$-äquivalenten 0-Itemmengen $\hat{p}_1 = \mathrm{GOTO}_{\hat{M}_{\mathrm{c}}}(\hat{q}_{\mathrm{s,c}}, \delta\alpha)$ und $\hat{p}'_1 = \mathrm{GOTO}_{\hat{M}_{\mathrm{c}}}(\hat{q}_{\mathrm{s,c}}, \eta\alpha)$, von denen wir wissen, daß $proj_{\mathrm{r}\varepsilon}(\hat{p}_1) = \{(A, \varepsilon), (\beta, \varepsilon), (\beta\gamma, \varepsilon)\} \neq \{(A, \varepsilon), (\beta\gamma, \varepsilon)\} = proj_{\mathrm{r}\varepsilon}(\hat{p}'_1)$ ist, darauf hin, daß selbst $proj_{\mathrm{r}\varepsilon}$ noch weiter verschärft werden kann, da – allerdings nur für $k = 0!$ – die gesamte Wirkung des

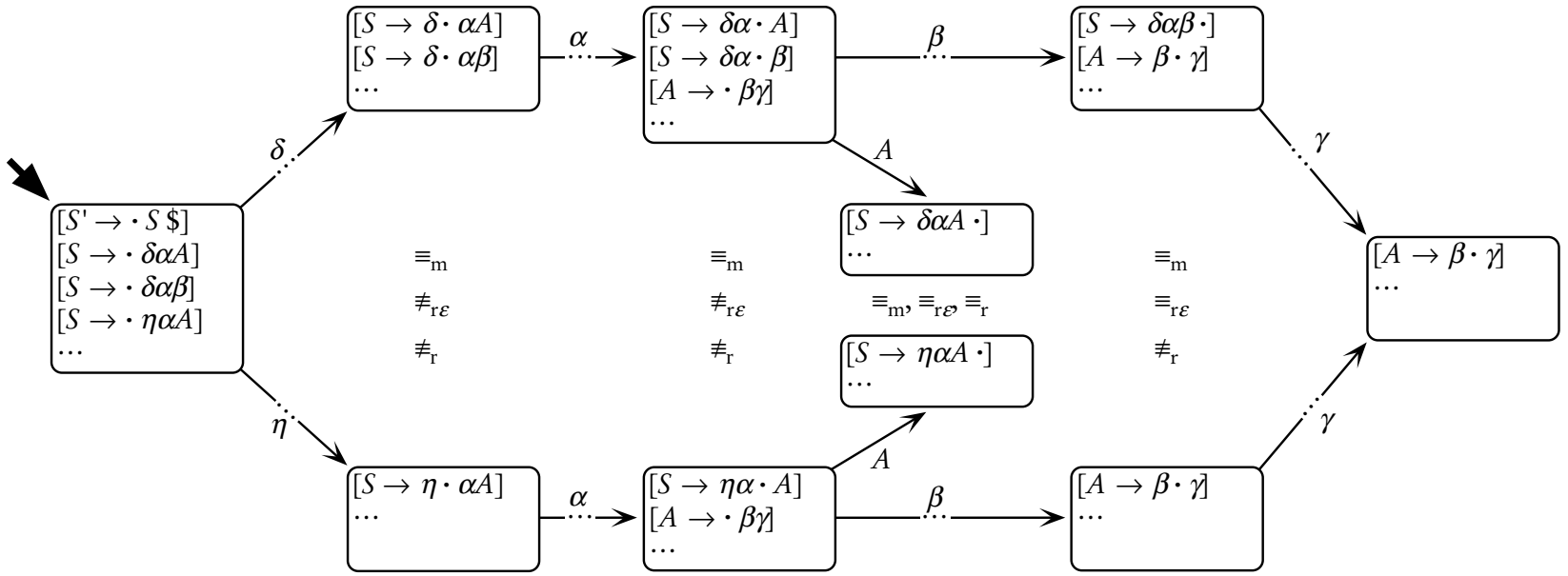


Abbildung 6: Eine Skizze wesentlicher Ausschnitte der kanonischen LR(0)-Maschine zum Grammatikschema $G = G_{r\varepsilon}$ mit den Regeln $S \to \delta\alpha A \mid \delta\alpha\beta \mid \eta\alpha A$, $A \to \beta\gamma$, … $(\alpha, \beta, \gamma, \delta, \eta \neq \varepsilon)$. Jeweils untereinander gezeichnete 0-Itemmengen sind $\equiv_m$-äquivalent, jedoch nicht alle $\equiv_r$- oder $\equiv_{r\varepsilon}$-äquivalent; die im einzelnen geltenden Beziehungen sind zwischen den Zuständen notiert.

Itemrests (β,ε) auf die Äquivalenzkriterien auch vom Itemrest $(\beta\gamma,\varepsilon)$ abgedeckt wird. Auf nur noch „wirkungsvolle“ 0-Item-Reste projiziert

$$proj_{\mathrm{r}\varepsilon\mathrm{pf}}(\hat{q}) = \{(\beta,\varepsilon) : \text{es existiert ein 0-Item } [A \to \alpha\cdot\beta] \in \hat{q} \text{ mit } \beta \neq \varepsilon \text{ und } \hat{q} \text{ enthält kein 0-Item der Form } [A' \to \alpha'\cdot\beta\gamma] \text{ mit } \gamma \neq \varepsilon\}$$

mit der zugehörigen Äquivalenzrelation $\equiv_{\mathrm{r}\varepsilon\mathrm{pf}}$ auf den Zuständen von $\hat{M}_{\mathrm{c}}$

$$\hat{q} \equiv_{\mathrm{r}\varepsilon\mathrm{pf}} \hat{q}' \quad \Leftrightarrow \quad proj_{\mathrm{r}\varepsilon\mathrm{pf}}(\hat{q}) = proj_{\mathrm{r}\varepsilon\mathrm{pf}}(\hat{q}')\,,$$

für die $\equiv_{\mathrm{m}} \supseteq \equiv_{\mathrm{r}\varepsilon\mathrm{pf}} \supseteq \equiv_{\mathrm{r}\varepsilon}$ gilt.[28]

Und mit den obigen Zustandsbezeichnungen $\hat{p}_1$ und $\hat{p}_1'$ ist $proj_{\mathrm{r}\varepsilon\mathrm{pf}}(\hat{p}_1) = \{(A,\varepsilon), (\beta\gamma,\varepsilon)\} = proj_{\mathrm{r}\varepsilon\mathrm{pf}}(\hat{p}_1')$, d. h. $\equiv_{\mathrm{r}\varepsilon\mathrm{pf}} \supsetneq \equiv_{\mathrm{r}\varepsilon}$. Immer noch ist der Faktorisierungsgrad von $proj_{\mathrm{r}\varepsilon\mathrm{pf}}$ jedoch nur eine „echte“ Approximation dessen von h_{m}, denn die Betrachtung der $\equiv_{\mathrm{m}}$-äquivalenten Zustände $\hat{p}_2 = \mathrm{GOTO}_{\hat{M}_{\mathrm{c}}}(\hat{q}_{\mathrm{s,c}},\delta)$ und $\hat{p}_2' = \mathrm{GOTO}_{\hat{M}_{\mathrm{c}}}(\hat{q}_{\mathrm{s,c}},\eta)$ ergibt $proj_{\mathrm{r}\varepsilon\mathrm{pf}}(\hat{p}_2) = \{(\alpha A,\varepsilon), (\alpha\beta,\varepsilon)\} \neq \{(\alpha A,\varepsilon)\} = proj_{\mathrm{r}\varepsilon\mathrm{pf}}(\hat{p}_2')$. Also ist $\equiv_{\mathrm{m}} \supsetneq \equiv_{\mathrm{r}\varepsilon\mathrm{pf}}$. Daß hier trotz verschiedener Itemrestmengen $\{(\alpha A,\varepsilon), (\alpha\beta,\varepsilon)\}$ und $\{(\alpha A,\varepsilon)\}$ dennoch $\hat{p}_2$ und $\hat{p}_2'$ $\equiv_{\mathrm{m}}$-äquivalente Zustände sind, ist konkret damit zu begründen, daß das Vorhandensein des 0-Items $[S \to \delta\cdot\alpha A]$ in $\hat{p}_2$ *indirekt über die Regel* $A \to \beta\gamma$ die Verlängerung der in $\hat{p}_2$ erkannten lebensfähigen Präfizes um $\alpha\beta\gamma$ erlaubt. Erst durch Hinzuziehen der Regeln von $G_{\mathrm{r}\varepsilon}$, hier speziell der Regel $A \to \beta\gamma$, wird also klar, daß die Wirkung der Itemreste $(\alpha\beta,\varepsilon)$ bereits durch $(\alpha A,\varepsilon)$ vorweggenommen ist. Und es sind kompliziertere Situationen konstruierbar, in denen erst wiederholte Regel„anwendungen“ auf Itemreste die „Wirkungslosigkeit“ anderer Itemreste nachweisen.

Außer in rechtslinearen Grammatiken ist für den Fall $k > 0$ überdies die Generierung neuer Lookahead-Anteile in den k-Items zu berücksichtigen, die durch die Relation $\mathbf{desc} = \mathbf{desc}^{G}_{\mathrm{LR}(k)}$ beschrieben wird. Auch hier ist also für detailliertere Zustandsäquivalenzanalysen eine Kenntnis von Ableitungsstrukturen der betrachteten Grammatik vonnöten.

Noch bessere Approximationen von $\equiv_{\mathrm{m}}$ scheinen daher nicht mehr über die isolierte Betrachtung einzelner Itemmengen einer kanonischen LR(k)-Maschine, sondern nur noch durch Nutzung von globalerem Strukturwissen zu gewinnen zu sein, und die lokale Deutung der Wirkung der Zustandsminimierung als eine Abstraktion von Komponenten der kanonischen Itemmengen gelingt somit nur näherungsweise. Wir geben uns an dieser Stelle mit der erarbeiteten Approximationskette

$$\equiv_{\mathrm{m}} \supseteq \equiv_{\mathrm{r}\varepsilon} \supseteq \equiv_{\mathrm{r}}$$

[28] Die Buchstabenkombination „pf“ im Index steht für präfixfrei.

zufrieden bzw. mit

$$\equiv_{\mathrm{m}} \supseteq \equiv_{\mathrm{r}\varepsilon\mathrm{pf}} \supseteq \equiv_{\mathrm{r}\varepsilon} \supseteq \equiv_{\mathrm{r}}$$

für rechtslineare Grammatiken oder $k = 0$.

6 Direkte Konstruktion kompakter LR(k)-Maschinen

Minimale allgemeine LR(k)-Maschinen durch Aufbau und dann Minimierung kanonischer LR(k)-Maschinen zu konstruieren ist höchst unbefriedigend, da die als Zwischenergebnisse dienenden kanonischen LR(k)-Maschinen i. a. deutlich größer sind als die minimierten Endergebnisse und in der Praxis wegen ihrer erheblichen Zustandsanzahlen durchaus Probleme aufwerfen können. Auch wenn es dort nur näherungsweise gelingt, Zustandsminimierung als Abstraktion von kanonischen Itemmengen zu deuten, läßt doch der vorige Abschnitt die Idee keimen, Redundanz bereits frühzeitig zu reduzieren, indem nichtdeterministische LR(k)-Maschinen nur noch aus Itemresten (β,y) statt vollständigen k-Items $[A \to \alpha\cdot\beta,y]$ aufgebaut werden. Dies nimmt die Wirkung der kennengelernten Projektionsabbildung $proj_{\mathrm{r}}$ auf Item*mengen* bereits in den aus *einzelnen* Items bestehenden Zuständen der *nichtdeterministischen* LR(k)-Maschine vorweg. Um dann einzusehen, daß wir so in der Tat zu einer allgemeinen LR(k)-Maschine $(\hat{M}_{\mathrm{r}},\mathrm{RC}_{\mathrm{r}})$ zu G gelangen – nämlich zur Maschine $proj_{\mathrm{r}}(\hat{M}_{\mathrm{c}},\mathrm{RC}_{\mathrm{c}})$ aus dem vorigen Abschnitt –, ohne erst die kanonische LR(k)-Maschine $(\hat{M}_{\mathrm{c}},\mathrm{RC}_{\mathrm{c}})$ aufbauen zu müssen, werden wir im wesentlichen nachweisen, daß die der Potenzkonstruktion vorgezogene Itemprojektion diese „überlebt“.

Wir beginnen damit, die auf Mengen von k-Items wirkende Projektionsabbildung $proj_{\mathrm{r}}$ kompatibel zu „individualisieren“, indem wir mit gleichem Namen definieren:

$$proj_{\mathrm{r}}([A \to \alpha\cdot\beta,y]) = (\beta,y) = [\beta,y]\,,$$

wobei wir die $[\ldots]$-Schreibweise von Itemresten (β,y) aus Gründen notationeller Nähe einführen und $[\beta,\varepsilon]$ auch zu $[\beta]$ abkürzen werden. (Eine analoge „Individualisierung“ von $proj_{\mathrm{r}\varepsilon}$ aus dem vorigen Abschnitt scheint unmöglich.) Wir konstruieren nun zu G den nicht ε-freien und i. a. nichtdeterministischen endlichen Automaten $M^{G}_{\mathrm{r,LR}(k)}$ (kurz M_{r}) aus M_{c} mit der Zustands- und Endzustandsmenge $Q_{\mathrm{r}} = proj_{\mathrm{r}}(I_k)$, dem Startzustand $[S\,\$,\varepsilon] = proj_{\mathrm{r}}([S' \to \cdot\, S\,\$,\varepsilon])$, dem Eingabealphabet V' und der Aktionenmenge

$$P_{\mathrm{r}} = \{proj_{\mathrm{r}}(q)X \to proj_{\mathrm{r}}(q') : q,q' \in I_k,\ X \in V' \cup \{\varepsilon\} \text{ und } qX \to q' \in pass_{\mathrm{c}} \cup desc_{\mathrm{c}}\}\,.$$

M_r heißt die *redundanzreduzierte nichtdeterministische LR(k)-Maschine zu G.*[29] Der sparsame Potenzautomat $\hat{M}_r$ zu M_r heißt *redundanzreduzierte (deterministische) LR(k)-Maschine zu G.* Seinen Startzustand werden wir einheitlich mit $\hat{q}_{s,r}$ bezeichnen, seine Zustandsmenge mit $\hat{Q}_r$ und seine Aktionenmenge mit $\hat{P}_r$. Die Aktionen von M_r entsprechen genau denen von M_c nach Projektion der in M_c vorkommenden Zustände mittels $proj_r$. Dies beschreibt in natürlicher Weise ein ohne Verwechselungsgefahr ebenfalls $proj_r$ genannter Homomorphismus (bzgl. Konkatenation) von M_c-Aktionenfolgen in M_r-Aktionenfolgen, der gegeben ist durch

$$proj_r(qX \to q') = proj_r(q)X \to proj_r(q') \ .$$

$proj_r$ erhält hier den Aktionentyp in dem Sinne, daß $\hat{P}_r = pass_r \cup desc_r$ ist mit den disjunkten Mengen $pass_r = proj_r(pass_c)$ und $desc_r = proj_r(desc_c)$. Und offensichtlich ist der Aktionenhomomorphismus $proj_r$ „längenerhaltend“, d. h. $|proj_r(\pi)| = |\pi|$.

Beispiel: Abbildung 7 zeigt die redundanzreduzierte nichtdeterministische LR(1)-Maschine zu $G = G_{ab\varepsilon}$. M_c weist 29 erreichbare Zustände auf, M_r dagegen nur noch 19. □

Die folgenden zwei Lemmata beschreiben wesentliche allgemeine Verhaltensanalogien zwischen kanonischer und redundanzreduzierter nichtdeterministischer LR(k)-Maschine zu G.

Lemma 6.1 (M_c-nach-M_r-Verhaltensanalogie)
Seien $q, q' \in I_k$, $\gamma \in V'^*$ und π eine Aktionenfolge von M_c, so daß

$$q\gamma \stackrel{\pi}{\Longrightarrow} q' \quad \text{in } M_c$$

erfüllt ist. Dann gilt auch

$$proj_r(q)\gamma \stackrel{proj_r(\pi)}{\Longrightarrow} proj_r(q') \quad \text{in } M_r \ .$$

Beweis: Induktion über $|\pi|$. □

Lemma 6.2 (M_r-nach-M_c-Verhaltensanalogie)
Seien $q, q' \in proj_r(I_k)$, $\gamma \in V'^*$ und π eine Aktionenfolge von M_r, so daß

$$q\gamma \stackrel{\pi}{\Longrightarrow} q' \quad \text{in } M_r \tag{11}$$

[29] Natürlich wird M_r hier nur aus formalen Gründen *aus* M_c konstruiert.

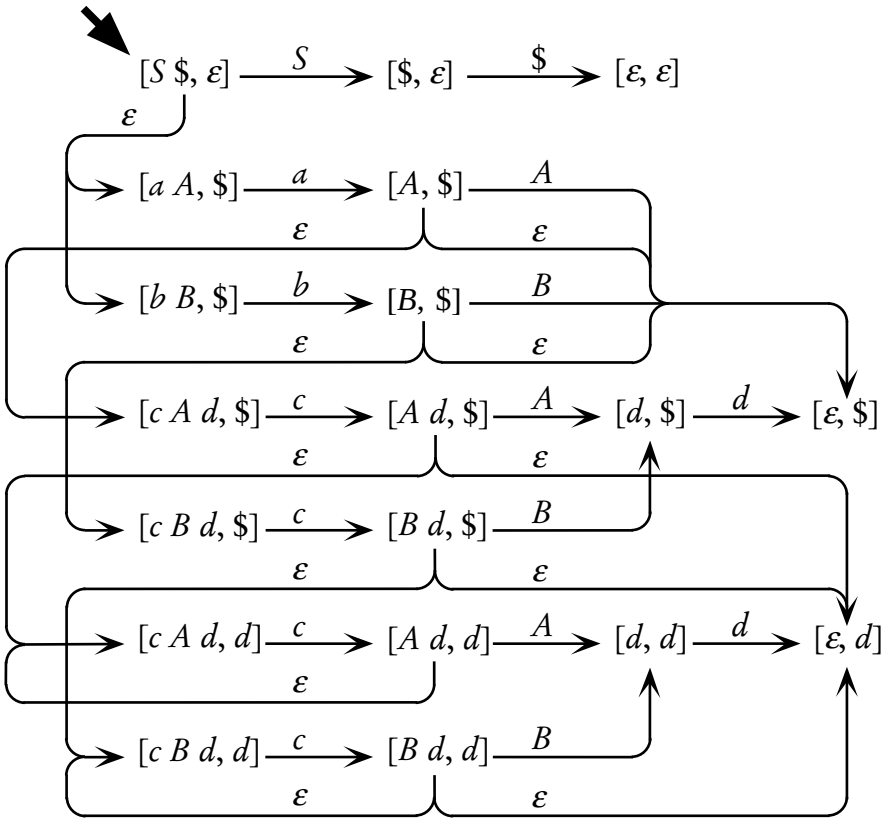


Abbildung 7: Der erreichbare Ausschnitt der redundanzreduzierten nichtdeterministischen LR(1)-Maschine zu $G = G_{ab\varepsilon}$. Die in Abbildung 1 der kanonischen nichtdeterministischen LR(1)-Maschine zu $G_{ab\varepsilon}$ grau unterlegten Zustandsgruppen kollabieren zu je einem einzigen Zustand.

erfüllt ist. Dann existieren Zustände $p_1, p_1' \in I_k$ *und eine Aktionenfolge* π_1 *von* M_c*, so daß*

$$\begin{aligned} &q = proj_r(p_1)\,, \quad q' = proj_r(p_1')\,, \quad \pi = proj_r(\pi_1) \quad \textit{und} \\ &p_1\gamma \stackrel{\pi_1}{\Longrightarrow} p_1' \quad \textit{in } M_c \end{aligned} \tag{12}$$

gilt. Sei ferner $p_2 \in I_k$ *mit* $proj_r(p_2) = q = proj_r(p_1)$*. Dann existieren* $p_2' \in proj_r(I_k)$ *und eine Aktionenfolge* π_2 *von* M_c*, so daß*

$$\begin{aligned} &proj_r(p_2') = q' = proj_r(p_1')\,, \quad proj_r(\pi_2) = \pi = proj_r(\pi_1) \quad \textit{und} \\ &p_2\gamma \stackrel{\pi_2}{\Longrightarrow} p_2' \quad \textit{in } M_c \end{aligned} \tag{13}$$

gilt.

Beweis: Wir betreiben Induktion über $|\pi|$.

Im Falle $|\pi| = 0$ ist $\pi = \varepsilon = \gamma$ und $q = q'$. Alle Zustände (Itemreste) von M_r sind Bild wenigstens eines Zustands (k-Items) von M_c, daher existiert ein Zustand $p_1 = p_1'$ von M_c mit $proj_r(p_1) = proj_r(p_1') = q = q'$, und (12) und (13) gelten mit der Wahl $\pi_1 = \pi_2 = \varepsilon$ und $p_2' = p_2$.

Im Induktionsschritt nehmen wir an, es sei $\pi = \pi' r$ und r eine Aktion von M_r. Ableitung (11) gliedert sich dann zunächst auf in

$$q\gamma = q\gamma' X \stackrel{\pi'}{\Longrightarrow} q'' X \stackrel{r}{\Rightarrow} q' \quad \text{in } M_r\,, \tag{14}$$

und für das linke Ableitungssegment liefert die Induktionsannahme, daß zum einen Zustände p_1, p_1'' und eine Aktionenfolge π_1' von M_c existieren, so daß

$$\begin{aligned} &q = proj_r(p_1)\,, \quad q'' = proj_r(p_1'')\,, \quad \pi' = proj_r(\pi_1') \quad \text{und} \\ &p_1\gamma = p_1\gamma' X \stackrel{\pi_1}{\Longrightarrow} p_1'' X \quad \text{in } M_c \end{aligned} \tag{15}$$

erfüllt ist, und zum anderen, daß für einen Zustand p_2 von M_c mit $proj_r(p_2) = q = proj_r(p_1)$ eine Aktionenfolge π_2' und ein Zustand p_2'' von M_c existieren, so daß

$$\begin{aligned} &proj_r(p_2'') = q'' = proj_r(p_1'')\,, \quad proj_r(\pi_2') = \pi' = proj_r(\pi_1') \quad \text{und} \\ &p_2\gamma = p_2\gamma' X \stackrel{\pi_2'}{\Longrightarrow} p_2'' X \quad \text{in } M_c \end{aligned} \tag{16}$$

gilt. Wir unterscheiden nun den Typ von $r = q'' X \to q'$.

Für $r \in pass_r$ ist $X \in V$, q'' ein Itemrest $[X\beta, y]$ und q' ein Itemrest $[\beta, y]$. Wegen $q'' = proj_r(p_1'')$ ist ferner p_1'' ein k-Item $[A_1 \to \alpha_1 \cdot X\beta, y]$. Sei p_1' dann das k-Item $[A_1 \to \alpha_1 X \cdot \beta, y]$.

Nach Konstruktion ist jetzt $r_1 = p_1'' X \to p_1' \in pass_c$, und (12) gilt mit der Wahl $\pi_1 = \pi_1' r_1$, denn $proj_r(\pi_1) = proj_r(\pi_1' r_1) = proj_r(\pi_1') proj_r(r_1) = \pi' r$. Weiterhin muß wegen $proj_r(p_2'') = q'' = proj_r(p_1'')$ aus (16) nun p_2'' ein k-Item $[A_2 \to \alpha_2 \cdot X\beta, y]$ sein. Dann existiert aber nach Konstruktion mit der Wahl $p_2' = [A_2 \to \alpha_2 X \cdot \beta, y]$ auch die Aktion $r_2 = p_2'' X \to p_2' \in pass_c$, und wir können die Ableitung aus (16) verlängern zu

$$p_2\gamma = p_2\gamma' X \overset{\pi_2'}{\Longrightarrow} p_2'' X \overset{r_2}{\Longrightarrow} p_2' \quad \text{in } M_c\,,$$

woraus mit der Wahl $\pi_2 = \pi_2' r_2$ und $proj_r(\pi_2) = proj_r(\pi_2' r_2) = proj_r(\pi_2') proj_r(r_2) = \pi' r = \pi$ dann (13) folgt.

Für $r \in desc_r$ ist $X = \varepsilon$, q'' ist ein Itemrest $[B\beta, y]$ und q' ein Itemrest $[\omega, z]$, wobei $B \to \omega \in P$ und $z = \text{FIRST}_k'(\beta y)$ ist. Wegen $q'' = proj_r(p_1'')$ ist ferner p_1'' ein k-Item $[A_1 \to \alpha_1 \cdot B\beta, y]$. Sei p_1' dann das k-Item $[B \to \cdot\, \omega, z]$. Nach Konstruktion ist jetzt $r_1 = p_1'' \to p_1' \in desc_c$, und (12) gilt mit der Wahl $\pi_1 = \pi_1' r_1$. Weiterhin muß wegen $proj_r(p_2'') = q'' = proj_r(p_1'')$ aus (16) nun p_2'' ein k-Item $[A_2 \to \alpha_2 \cdot B\beta, y]$ sein. Dann existiert aber nach Konstruktion auch die Aktion $r_2 = p_2'' \to p_1' \in desc_c$, und (13) folgt wie im vorigen Fall, wenn wir die Ableitung aus (16) um eine Anwendung von r_2 verlängern und $\pi_2 = \pi_2' r_2$ sowie $p_2' = p_1'$ setzen. □

Die Redundanz in der kanonischen Konstruktion schon in der frühen Stufe der kanonischen nichtdeterministischen LR(k)-Maschine M_c zu G äußert sich prägnant darin, daß allein mit Hilfe der Struktur der redundanzreduzierten nichtdeterministischen LR(k)-Maschine zu G für jeden Zustand (Itemrest) von M_r seine Urbilder bezüglich $proj_r$ in M_c (k-Items) rekonstruierbar sind: Es gilt

$$proj_r^{-1}([\beta, y]) = \{\, [A \to \alpha \cdot \beta, y] : \text{für irgendwelche } \alpha,\ A \text{ ist } [A \to \alpha \cdot \beta, y] \in I_k \,\}$$

(Grammatiksicht)

$$= \{\, [A \to \alpha \cdot \beta, y] : A = S' \text{ und es gilt } [S\,\$, \varepsilon]\alpha \Rightarrow^{|\alpha|} [\beta, y] \text{ in } M_r$$
$$\text{für irgendein } \alpha,$$
$$\text{oder aber } A \in V' \text{ und für irgendein } \alpha \text{ und}$$
$$\text{Zustände } q, q', q'' \text{ von } M_r \text{ gilt } q \Rightarrow q',\ qA \Rightarrow q''$$
$$\text{und } q'\alpha \Rightarrow^{|\alpha|} [\beta, y] \text{ in } M_r \,\}\,,$$

(Automatensicht)

wobei hilfreich für den hier nicht ausgeführten Beweis Lemma 6.2 über die M_r-nach-M_c-Verhaltensanalogie ist. Die Projektion von M_c auf M_r vermöge $proj_r$ verliert also strukturell keinerlei Information.

Theorem 6.3 (Übertragung der Redundanzreduktion auf die Potenzautomaten)
Es ist

$$L(M_c) = L(M_r) = L(\hat{M}_c) = L(\hat{M}_r) = \mathrm{VP}(G') \; . \tag{17}$$

Insbesondere gilt für jedes $\gamma \in V'^*$:

$$\mathrm{GOTO}_{\hat{M}_r}(\hat{q}_{s,r}, \gamma) \textit{ existiert} \quad gdw. \quad \mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \gamma) \textit{ existiert,} \tag{18}$$

und dann ist

$$\mathrm{GOTO}_{\hat{M}_r}(\hat{q}_{s,r}, \gamma) = proj_r(\mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \gamma)) \; . \tag{19}$$

Beweis: Um (18) und (19) zu zeigen, betrachten wir für ein $\gamma \in V'^*$

$$\hat{p} = \{p : [S' \to \cdot S\$, \varepsilon]\gamma \Rightarrow^* p \in I_k \quad \text{in } M_c\}$$

und

$$\hat{q} = \{q : [S\$, \varepsilon]\gamma \Rightarrow^* q \in proj_r(I_k) \quad \text{in } M_r\} \; .$$

Es gilt

$$\begin{aligned} proj_r(\hat{p}) &= \{proj_r(p) : [S' \to \cdot S\$, \varepsilon]\gamma \Rightarrow^* p \in I_k \quad \text{in } M_c\} \\ &= \hat{q} \; , \end{aligned} \tag{20}$$

denn:

Es ist $\hat{q} \supseteq proj_r(\hat{p})$ mit Lemma 6.1 über die M_c-nach-M_r-Verhaltensanalogie, wenn wir für die dortigen Bezeichner q und q' respektive die hiesigen $[S' \to \cdot S\$, \varepsilon]$ und p einsetzen.

Ferner ist $\hat{q} \subseteq proj_r(\hat{p})$ mit Teil (13) von Lemma 6.2 über die M_r-nach-M_c-Verhaltensanalogie, wenn wir für die dortigen Zustände q, q' und p_2 die hiesigen $[S\$, \varepsilon]$, q und $[S' \to \cdot S\$, \varepsilon]$ einsetzen.

Des weiteren sind alle Zustände von M_c und M_r Endzustände. Nach Konstruktion ist folglich $\hat{p} = \mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \gamma)$ Zustand und Endzustand von $\hat{M}_c$ genau dann, wenn $\hat{p} \neq \emptyset$ ist, und $\hat{q} = \mathrm{GOTO}_{\hat{M}_r}(\hat{q}_{s,r}, \gamma)$ Zustand und Endzustand von $\hat{M}_r$ genau dann, wenn $\hat{q} \neq \emptyset$ ist. Es folgt also (18) und mit (20) auch (19), und es ist $L(\hat{M}_c) = L(\hat{M}_r)$. Die restlichen Identitäten aus (17) müssen nicht mehr gezeigt werden. □

Da $\hat{M}_c$ und $\hat{M}_r$ als sparsame Potenzautomaten nur erreichbare Zustände enthalten, schließen wir auf

Korollar 6.4
$proj_{\mathrm{r}}$ auf k-Itemmengen bildet die Zustände von $\hat{M}_{\mathrm{c}}$ surjektiv in die Zustände von $\hat{M}_{\mathrm{r}}$ ab.
□

Und aus dem Fall $\gamma = \varepsilon$ im Theorem ergibt sich die Gültigkeit von

Korollar 6.5

$$proj_{\mathrm{r}}(\hat{q}_{\mathrm{s,c}}) = \hat{q}_{\mathrm{s,r}} \,. \qquad \square$$

Auch die Aktionen von $\hat{M}_{\mathrm{c}}$ und $\hat{M}_{\mathrm{r}}$ lassen sich vermöge $proj_{\mathrm{r}}$ leicht in Beziehung setzen:

Korollar 6.6
Ist $\hat{p}X \to \hat{p}'$ eine Aktion von $\hat{M}_{\mathrm{c}}$ für ein $X \in V'$, so ist $proj_{\mathrm{r}}(\hat{p})X \to proj_{\mathrm{r}}(\hat{p}')$ eine Aktion von $\hat{M}_{\mathrm{r}}$.

Beweis: Zu $\hat{p}X \to \hat{p}' = r$ läßt sich stets ein $\delta \in V'^{*}$ finden, so daß die Situation

$$\hat{q}_{\mathrm{s,c}}\delta X \Rightarrow^{*} \hat{p}X \stackrel{r}{\Longrightarrow} \hat{p}' \quad \text{in } \hat{M}_{\mathrm{c}} \quad \text{und}$$
$$\hat{q}_{\mathrm{s,r}}\delta X \Rightarrow^{*} \hat{q}X \stackrel{r'}{\Longrightarrow} \hat{q}' \quad \text{in } \hat{M}_{\mathrm{r}}$$

herstellbar ist, denn als sparsamer Potenzautomat enthält $\hat{M}_{\mathrm{c}}$ nur erreichbare Zustände, und wie Theorem 6.3 über die Übertragung der Redundanzreduktion auf die Potenzautomaten besagt, existiert $\hat{p} = \mathrm{GOTO}_{\hat{M}_{\mathrm{c}}}(\hat{q}_{\mathrm{s,c}}, \delta)$ gdw. $\hat{q} = \mathrm{GOTO}_{\hat{M}_{\mathrm{r}}}(\hat{q}_{\mathrm{s,r}}, \delta)$ existiert. Dann folgt mit der Wahl $\gamma = \delta$ im selben Theorem, daß $\hat{q} = proj_{\mathrm{r}}(\hat{p})$ ist. Und ebenso folgt mit der Wahl $\gamma = \delta X$, daß $\hat{q}' = proj_{\mathrm{r}}(\hat{p}')$ ist. Also ist $proj_{\mathrm{r}}(\hat{p})X \to proj_{\mathrm{r}}(\hat{p}')$ die Aktion $\hat{p}X \to \hat{p}' = r'$ von $\hat{M}_{\mathrm{r}}$. □

Auch von Aktionenfolgen von $\hat{M}_{\mathrm{c}}$ in Aktionenfolgen von $\hat{M}_{\mathrm{r}}$ können wir also (bezüglich Konkatenation) einen Homomorphismus $proj_{\mathrm{r}}$ angeben, der die Wirkung der Redundanzreduktion von M_{c} zu M_{r} auf $\hat{M}_{\mathrm{c}}$ und $\hat{M}_{\mathrm{r}}$ reflektiert. Dieser ist gegeben durch

$$proj_{\mathrm{r}}(\hat{p}X \to \hat{p}') = proj_{\mathrm{r}}(\hat{p})X \to proj_{\mathrm{r}}(\hat{p}') \,,$$

offensichtlich längenerhaltend und, als Abbildung von $\hat{M}_{\mathrm{c}}$- in $\hat{M}_{\mathrm{r}}$-Aktionen aufgefaßt, surjektiv.

Wir ordnen nun noch $\hat{M}_r$ die folgende Zuordnungsvorschrift RC_r von Rechtskontextpaaren an jeden Zustand $\hat{q}$ von $\hat{M}_r$ bei:

$$\begin{aligned} RC_r(\hat{q}) = \{A \,|\, y \,:\, & \text{es ex. } [A\beta, z] \in \hat{q} \text{ mit } A \in V' \setminus T', \beta \in V'^*, z \in T'^*, \\ & \text{so daß } y \in \text{FIRST}'_k(\beta z)\} \quad \cup \\ \{a \,|\, x \,:\, & \text{es ex. } [a\beta, z] \in \hat{q} \text{ mit } a \in T', \beta \in V'^*, z \in T'^*, \\ & \text{so daß } x \in \text{FIRST}'_{\max\{k-1,0\}}(\beta z)\} \,. \end{aligned} \tag{21}$$

Jede Paarmenge $RC_r(\hat{q})$ kann – für $k \geq 1$ unter Zuhilfenahme von G' – direkt aus den Itemresten in $\hat{q}$ extrahiert werden.

Theorem 6.7

(a) $(\hat{M}_r, RC_r)$ *ist eine allgemeine LR(k)-Maschine zu G.*

(b) $(\hat{M}_r, RC_r) = proj_r(\hat{M}_c, RC_c)$, *d. h.* $(\hat{M}_r, RC_r)$ *entsteht als Zusammenfassung von Zuständen von* $\hat{M}_c$ *gemäß* $proj_r$.

Beweis: (a) Nach Konstruktion ist $\hat{M}_r$ ein vollständig spezifizierter, buchstabierender DEA, zu dem Theorem 6.3 über die Übertragung der Redundanzreduktion auf die Potenzkonstruktion bestätigt, daß dieser die Sprache der lebensfähigen Präfizes zu G' akzeptiert. Weiter besitzt $\hat{M}_r$ keine unerreichbaren Zustände. Und um zu zeigen, daß auch RC_r den Forderungen genügt, seien $\gamma \in VP(G')$, $\hat{q} = GOTO_{\hat{M}_r}(\hat{q}_{s,r}, \gamma)$ und $\hat{p} = GOTO_{\hat{M}_c}(\hat{q}_{s,c}, \gamma)$. Wir wissen, daß $\hat{q} = proj_r(\hat{p})$ ist, womit dann (21) äquivalent ist zu

$$\begin{aligned} RC_r(\hat{q}) = \{A \,|\, y \,:\, & \text{es ex. } [A\beta, z] \in proj_r(\hat{p}) \text{ mit } A \in V' \setminus T', \\ & \text{so daß } y \in \text{FIRST}'_k(\beta z)\} \quad \cup \\ \{a \,|\, x \,:\, & \text{es ex. } [a\beta, z] \in proj_r(\hat{p}) \text{ mit } a \in T', \\ & \text{so daß } x \in \text{FIRST}'_{\max\{k-1,0\}}(\beta z)\} \,. \end{aligned}$$

Dies ist gleichbedeutend mit

$$\begin{aligned} RC_r(\hat{q}) = \{A \,|\, y \,:\, & \text{es ex. } [B \to \alpha \cdot A\beta, z] \in \hat{p} \text{ mit } A \in V' \setminus T', \\ & \text{so daß } y \in \text{FIRST}'_k(\beta z)\} \quad \cup \\ \{a \,|\, x \,:\, & \text{es ex. } [A \to \alpha \cdot a\beta, z] \in \hat{p} \text{ mit } a \in T', \\ & \text{so daß } x \in \text{FIRST}'_{\max\{k-1,0\}}(\beta z)\} \,. \end{aligned}$$

Und mit einer weiteren Umformung nach Konstruktion von $\hat{M}_c$ erhalten wir

$$\begin{aligned}\mathrm{RC_r}(\hat{q}) = {} & \{A \,|\, y : \text{es ex. } [A \to \cdot\,\omega, y] \in \hat{p} \text{ mit } A \neq S'\} \quad \cup \\ & \{a \,|\, x : \text{es ex. } [A \to \alpha \cdot a\beta, z] \in \hat{p} \text{ mit } a \in T', \\ & \qquad \text{so daß } x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)\} \,. \\ \overset{(10)}{=} {} & \mathrm{RC_c}(\hat{p}) \,.\end{aligned}$$

Da $(\hat{M}_c, \mathrm{RC_c})$ aber eine allgemeine LR(k)-Maschine zu G ist (Lemma 3.3), gilt auch

$$\mathrm{RC_r}(\hat{q}) = \mathrm{RC_r}(\mathrm{GOTO}_{\hat{M}_r}(\hat{q}_{s,r}, \gamma)) = \{X \,|\, u : X \in V',\, u \in \mathrm{RC}(\gamma X)\} \,,$$

was noch zu zeigen war.

(b) folgt aus (a) und Theorem 6.3 über die Übertragung der Redundanzreduktion auf die Potenzautomaten bzw. seinen Korollaren. □

Beispiel: Abbildung 8 zeigt zu $G = G_{ab\varepsilon}$ die redundanzreduzierte LR(1)-Maschine $\hat{M}_r$ neben der kanonischen LR(1)-Maschine $\hat{M}_c$. Die Wirkung der Redundanzreduktion auf die Potenzkonstruktion (Theorem 6.3) ist durch Grauunterlegung von Zustandsgruppen verdeutlicht: Zustände und Aktionen von $\hat{M}_r$ können durch Projektion vermöge $proj_r$ gewonnen werden.

Die Zustandsanzahl sinkt durch die Redundanzreduzierung von 19 auf 13 Zustände. $\tilde{M}_r$ weist noch 10 Shift-Aktionen ($\tilde{M}_c$: 12) und 14 Reduce-Aktionen ($\tilde{M}_c$: 14) auf. Der Vergleich mit der minimalen allgemeinen LR(1)-Maschine zu $G_{ab\varepsilon}$, die in Abbildung 4 gezeigt ist, und ein Blick auf die Zeichnung von $\hat{M}_r$ als allgemeine LR(1)-Maschine $(\hat{M}_r, \mathrm{RC_r})$ machen klar, welche unnötige Information in $\hat{M}_r$ dessen Zustandsminimalität verhindert: Die Zustände $\{[Ad, \$], [\varepsilon, d], [cAd, d]\}$ und $\{[Ad, d], [\varepsilon, d], [cAd, d]\}$ unterscheiden sich aufgrund einer systematisch mitgeführten Lookahead-Komponente, die für die Parserkonstruktion irrelevant ist. Entsprechendes gilt für die Zustände $\{[Bd, \$], [\varepsilon, d], [cBd, d]\}$ versus $\{[Bd, d], [\varepsilon, d], [cBd, d]\}$. □

Natürlich kann der aus $(\hat{M}_r, \mathrm{RC_r})$ konstruierte allgemeine LR(k)-Parser $(\tilde{M}_r, \tau_r)$, den wir den *redundanzreduzierten LR(k)-Parser zu G* nennen, auch direkt aus $\hat{M}_r$, d. h. ohne vorherigen Abstraktionsschritt von $\hat{M}_r$ zu $(\hat{M}_r, \mathrm{RC_r})$, aus den Itemrestmengen abgelesen werden, die die Zustände von $\hat{M}_r$ bilden. Die Aktionen von $(\tilde{M}_r, \tau_r)$ bestimmen sich dann wie folgt:

Für jeden Zustand $\hat{q}$ von $\hat{M}_r$ und jedes $a \in T$ ist

$$r = \quad \hat{q} \,|\, ax \to \hat{q}\, a\, \mathrm{GOTO}_{\hat{M}_r}(\hat{q}, a) \,|\, x$$

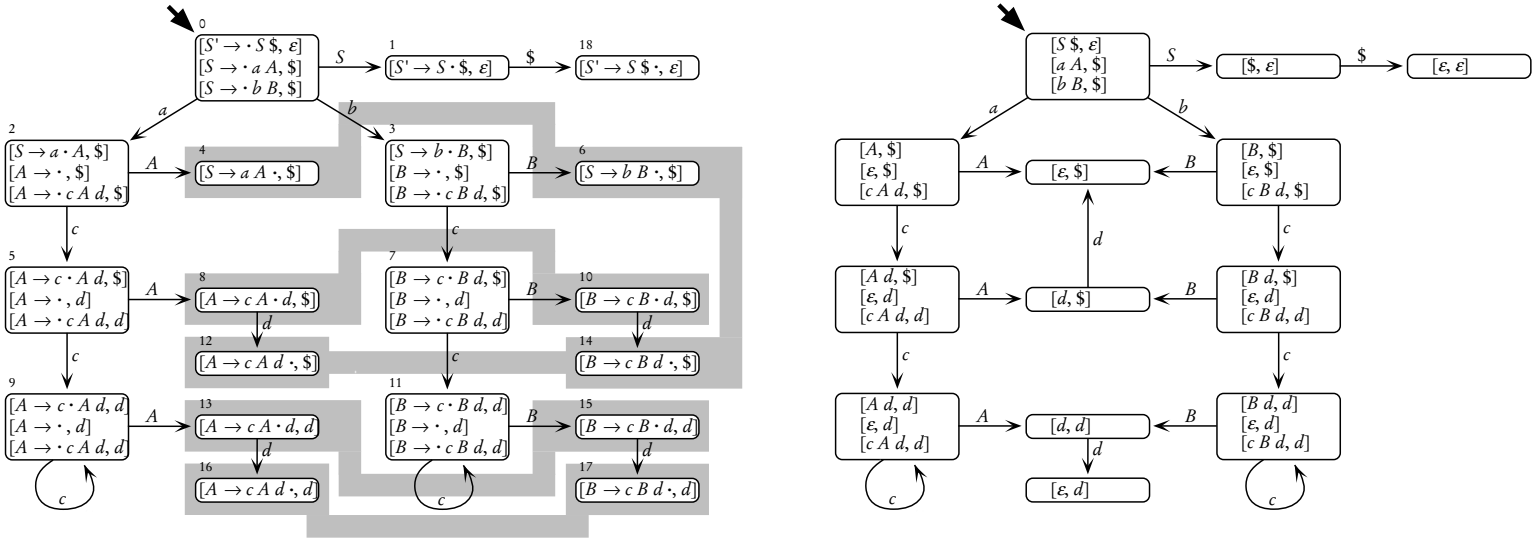


Abbildung 8: Die kanonische LR(1)-Maschine $\hat{M}_{\mathrm{c}}$ (links) und die redundanzreduzierte LR(1)-Maschine $\hat{M}_{\mathrm{r}}$ (rechts) zu $G = G_{abe}$. Grau unterlegte Zustandsgruppen in $\hat{M}_{\mathrm{c}}$ fallen zu je einem einzigen Zustand in $\hat{M}_{\mathrm{r}}$ zusammen.

eine Shift-Aktion (von a) genau dann, wenn ein Itemrest $[a\beta, z] \in \hat{q}$ existiert und zudem $x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)$ ist; dazu wird $\tau_{\mathrm{r}}(r) = \varepsilon$ gesetzt. Und

$$r = \quad \hat{q}\, X_1\ \mathrm{GOTO}_{\hat{M}_{\mathrm{r}}}(\hat{q}, X_1) \ldots X_m\ \mathrm{GOTO}_{\hat{M}_{\mathrm{r}}}(\hat{q}, X_1 \ldots X_m) \mid y \\ \rightarrow \hat{q}\, A\ \mathrm{GOTO}_{\hat{M}_{\mathrm{r}}}(\hat{q}, A) \mid y$$

ist eine Reduce-Aktion (nach Regel r') genau dann, wenn $m \geq 0$ ist, $X_1, \ldots, X_m \in V$ und $r' = A \rightarrow X_1 \ldots X_m \in P$ sind und ferner für ein $\beta \in V'^*$ und ein $z \in k : T'^*$ die Beziehung $[A\beta, z]\ \mathit{desc}_{\mathrm{r}}\ [X_1 \ldots X_m, y]$ besteht; dazu wird $\tau_{\mathrm{r}}(r) = r'$ gesetzt.

Mit den Ergebnissen dieses Abschnitts verfügen wir nun über eine Methode, kompaktere allgemeine LR(k)-Maschinen als die kanonischen, wenn auch nicht minimale, direkt zu konstruieren. Neben dem Vorteil der direkten Konstruierbarkeit bieten redundanzreduzierte allgemeine LR(k)-Maschinen in geringfügigem Maße auch Optimierungspotential für ihre Implementierung, worauf im folgenden Abschnitt eingegangen wird.

7 Allgemeine LR(k)-Parser mit besonderen Eigenschaften

Es ist an dieser Stelle recht naheliegend, wie wir erreichen können, eine allgemeine LR(k)-Maschine an ihren Zuständen *verfeinerte* Stützinformation berechnen zu lassen, d. h. solches Wissen, das über die unverzichtbaren LR(k)-Rechtskontexte hinausgeht. Zwar werden im allgemeinen derartige Informationsverfeinerungen dazu führen, daß bei einem Zustandsminimierungsprozeß seltener Zustände zusammengefaßt werden dürfen. Dennoch kann es gewollt sein, diesen Preis zu bezahlen, wenn die gewonnenen zusätzlichen Eigenschaften des abgelesenen Parsers dies rechtfertigen. Auf den folgenden Seiten wollen wir zwei Möglichkeiten sinnvoller Informationsanreicherungen vorstellen, die etwa bei der im Anhang vorgestellten allgemeinen Implementierungstechnik für allgemeine LR-Parser Vereinfachungen ermöglichen.

Aufgrund der Struktur ihrer Aktionen steht für allgemeine LR(k)-Parser mit dem obersten aktuellen Kellersymbol und dem k-Lookahead fest, ob eine Shift-Aktion anwendbar ist; ob jedoch irgendeine Reduce-Aktion anwendbar ist, kann aus derselben Information nicht geschlossen werden, sondern ergibt sich erst, wenn sukzessive auch tieferer Kellerkontext mitberücksichtigt wird. (Die erforderliche Kellerkontexttiefe ist situationsabhängig.) Und dabei kann nun der Fall auftreten, daß für den aktuellen Lookahead und den obersten Kellerzustand, alleine betrachtet, noch die Möglichkeit besteht, eine Reduce-Aktion anzuwenden, daß sich schließlich jedoch zum tieferen Kellerkontext dann doch keine einzige anwendbare Reduce-Aktion mehr finden läßt, sprich: ein Syntaxfehler vorliegt. Beispielsweise ist konkret im studierten Parser $(\tilde{M}_{\mathrm{m}}, \tau_{\mathrm{m}})$ zu $G = G_{ab\varepsilon}$ [30] noch $c\,5\,A\,7\,d\,4\,|\,\$$ Suffix einer linken Aktionenseite, $5\,c\,5\,A\,7\,d\,4\,|\,\$$ aber gerade nicht mehr (sehr wohl jedoch $5\,c\,5\,A\,7\,d\,4\,|\,d$), analog auch $6\,c\,6\,B\,7\,d\,4\,|\,\$$, $2\,c\,5\,A\,7\,d\,4\,|\,d$ und $3\,c\,6\,B\,7\,d\,4\,|\,d$ gerade nicht mehr.

Eine „frühe“ Syntaxfehlererkennung mittels dem alleinigen Wissen über k-Lookahead und obersten Kellerzustand ermöglichen wir einer Implementierung nun wie folgt:

Einen allgemeinen LR(k)-Parser $(\tilde{M}, \tau)$ zu G, der aus einer allgemeinen LR(k)-Maschine $(\hat{M}, \mathrm{RC}_{\hat{M}})$ konstruiert ist, nennen wir *reduktionsdeterminiert*, wenn für jede Reduce-

[30] Siehe S. 60.

Aktion

$$r = \ldots \hat{q} \,|\, y \to \ldots$$

von $\tilde{M}$ und jedes $\varphi \in V'^*$ für das $\mathrm{GOTO}_{\hat{M}}(\hat{q}_s, \varphi) = \hat{q}$ ist, eine Regel $A \to \omega$ von G existiert, so daß $\varphi = \varphi'\omega$ ist und mit $\hat{q}' = \mathrm{GOTO}_{\hat{M}}(\hat{q}_s, \varphi')$ dann $A \,|\, y \in \mathrm{RC}_{\hat{M}}(\hat{q}')$ gilt. (Nach Konstruktion enthält $\tilde{M}$ dann eine Reduce-Aktion

$$\hat{q}' X_1 \,\mathrm{GOTO}_{\hat{M}}(\hat{q}', X_1) \ldots X_m \,\mathrm{GOTO}_{\hat{M}}(\hat{q}', X_1 \ldots X_m) \,|\, y \to \hat{q}' A \,\mathrm{GOTO}_{\hat{M}}(\hat{q}', A) \,|\, y\,,$$

wobei $X_1, \ldots, X_m \in V'$ sind, so daß $X_1 \ldots X_m = \omega$ ist.)

Mit anderen Worten: Liegt in einer Berechnung von $\tilde{M}$ eine Konfiguration $\$ \ldots \hat{q} \,|\, z\$$ vor und *kann* der Zustand $\hat{q}$ von $\hat{M}$ bei vorliegendem k-Lookahead das zuoberst gekellerte Symbol einer (mit dem Wissen $\hat{q}$, y) möglicherweise anwendbaren Reduce-Aktion von $\tilde{M}$ (z. B. r) sein, so *ist* dann *garantiert* eine Reduce-Aktion von $\tilde{M}$ (nicht notwendigerweise r) anwendbar.

Lemma 7.1 (Syntaxfehlererkennung in reduktionsdeterminierten LR(k)-Parsern)
Ein reduktionsdeterminierter (allgemeiner) LR(k)-Parser erkennt vorliegende Syntaxfehler schon durch Inspektion von oberstem Kellersymbol und k-Lookahead.

Beweis: In Rede stehe der reduktionsdeterminierte LR(k)-Parser $(\tilde{M}, \tau)$, der aus $(\hat{M}, \mathrm{RC}_{\hat{M}})$ konstruiert sei. Aus einer Startkonfiguration erreiche $\tilde{M}$ eine Konfiguration $\$ \ldots \hat{q} \,|\, z\$$, $\hat{q}$ Zustand von $\hat{M}$, deren Kellerinhalt das lebensfähige Präfix φ von G' buchstabiere (Lemma 3.5). Es liege ein Syntaxfehler vor, d. h. keine der Aktionen von $\hat{M}$ sei anwendbar. Nehmen wir jetzt an, dies zu erkennen sei *nicht* alleine durch Inspektion von oberstem Kellersymbol $\hat{q}$ und k-Lookahead $y = k : z\$$ möglich. Dann muß $\tilde{M}$ eine (Reduce-)Aktion $r = \ldots \hat{q} \,|\, y \to \ldots$ besitzen, die nicht anwendbar ist. $(\tilde{M}, \tau)$ ist aber reduktionsdeterminiert, weswegen eine Regel $A \to \omega$ von G existieren muß, so daß die Zerlegung $\varphi = \varphi'\omega$ möglich ist und mit $\hat{q}' = \mathrm{GOTO}_{\hat{M}}(\hat{q}_s, \varphi')$ dann $A \,|\, y \in \mathrm{RC}_{\hat{M}}(\hat{q}')$ gilt. Mit $\omega = X_1 \ldots X_m$ für $X_1, \ldots, X_m \in V$, $m \geq 0$, ist nach Konstruktion nun

$$r' = \quad \hat{q}' X_1 \,\mathrm{GOTO}_{\hat{M}}(\hat{q}', X_1) \ldots X_m \,\mathrm{GOTO}_{\hat{M}}(\hat{q}', X_1 \ldots X_m) \,|\, y \to \hat{q}' A \,\mathrm{GOTO}_{\hat{M}}(\hat{q}', A) \,|\, y$$

eine Reduce-Aktion von $\tilde{M}$. Und r' ist anwendbar auf die in Rede stehende Konfiguration. Dies steht aber im Widerspruch zur Voraussetzung, daß ein Syntaxfehler vorliegt. Die obige Aktion $r = \ldots \hat{q} \,|\, y \to \ldots$ kann folglich nicht existieren; $\hat{q}$ und y genügen also zur Analyse des Syntaxfehlers. □

Es ist klar, daß der kanonische LR(k)-Parser $(\tilde{M}_\mathrm{c}, \tau_\mathrm{c})$ zu G reduktionsdeterminiert ist: Sei

$$r = \quad \ldots \hat{q} \,|\, y \to \ldots$$

eine Reduce-Aktion nach Regel $A \to \omega$ von G. Dann ist $[A \to \omega \cdot, y] \in \hat{q}$. Ferner codiert, wie diskutiert, $\hat{q}$ seine „Geschichte“ ω. Daher können alle lebensfähigen Präfizes φ, die $\mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, \varphi) = \hat{q}$ erfüllen, als $\varphi = \varphi'\omega$ geschrieben werden; und mit $\hat{q}' = \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, \varphi')$ gilt $[A \to \cdot\omega, y] \in \hat{q}' = \mathrm{VALID}_k(\varphi')$ und gemäß (10) folglich auch $A \,|\, y \in \mathrm{RC}_\mathrm{c}(\hat{q}')$.

Ebenso wie der kanonische ist aber auch der redundanzreduzierte LR(k)-Parser $(\hat{M}_\mathrm{r}, \mathrm{RC}_\mathrm{r})$ reduktionsdeterminiert: Sei

$$r = \quad \ldots \hat{q} \,|\, y \to \ldots$$

eine seiner Reduce-Aktionen, und sei $\varphi \in V'^*$ beliebig derart, daß $\mathrm{GOTO}_{\hat{M}_\mathrm{r}}(\hat{q}_{\mathrm{s,r}}, \varphi) = \hat{q}$ ist. Gemäß Theorem 6.3 über die Übertragung der Redundanzreduktion auf die Potenzautomaten ist dann $\hat{q} = proj_\mathrm{r}(\hat{p})$ für $\hat{p} = \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, \varphi)$. Nach Konstruktion ist $[\varepsilon, y] \in \hat{q}$, und $\hat{p}$ muß mindestens ein k-Item der Form $[A \to \omega \cdot, y]$ enthalten, so daß, wie auch schon oben, $\varphi = \varphi'\omega$ ist. Dann ist wieder $[A \to \cdot\,\omega, y] \in \hat{p}' = \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, \varphi')$ und mit $\hat{q}' = proj_\mathrm{r}(\hat{p}') = \mathrm{GOTO}_{\hat{M}_\mathrm{r}}(\hat{q}_{\mathrm{s,r}}, \varphi')$ somit $A \,|\, y \in \mathrm{RC}_\mathrm{c}(\hat{p}') = \mathrm{RC}_\mathrm{r}(\hat{q}')$ (Theorem 6.7). Egal welches lebensfähige Präfix also vom Kellerinhalt einer erreichbaren Konfiguration $\$ \ldots \hat{q} \,|\, z\,\$$ von $\tilde{M}_\mathrm{r}$, $y = k : z\,\$$, buchstabiert wird – es ist stets mindestens eine Reduce-Aktion von $\tilde{M}_\mathrm{r}$ anwendbar.

Warum zwar nicht jeder allgemeine, wohl aber der kanonische und der redundanzreduzierte LR(k)-Parser zu G reduktionsdeterminiert sind, ist, abstrakt formuliert, damit zu begründen, daß die letzteren Parsern zugrunde liegenden LR(k)-Maschinen anhand der Item(rest)informationen systematisch solche Zustände, die auf dem Parserkeller bei bestimmtem k-Lookahead das *Ende eines Handles* anzeigen, von anderen Zuständen unterscheiden: Im Falle eines kanonischen LR(k)-Parsers zeigt ein k-Item $[\ldots \to \ldots \cdot, y] \in \hat{q}$ an, daß auf dem Parserkeller bei k-Lookahead y ein Handle abschließt; im Fall eines redundanzreduzierten LR(k)-Parsers zeigt entsprechend ein Itemrest $[\varepsilon, y] \in \hat{q}$ eine solche Situation an.

Wie generell zu beliebigen allgemeinen LR(k)-Parsern gelangt werden kann, die reduktionsdeterminiert sind, liegt damit auf der Hand: Man führe eine weitere Stützinformationskomponente ein, mit deren Hilfe Handleenden angezeigt werden. Dazu definiere man

analog zu RC auf $\mathrm{VP}(G')$ zunächst

$$\mathrm{RD}(\gamma) = \{y : \text{es ex. irgendein } k\text{-Item } [A \to \omega\cdot, y] \in \mathrm{VALID}_k(\gamma)\} .$$

Sodann beschränkt man sich auf allgemeine LR(k)-Maschinen $(\hat{M}, \mathrm{RC}_{\hat{M}})$, denen eine weitere Stützinformationskomponente $\mathrm{RD}_{\hat{M}}$ hinzugefügt werden kann, die für jedes $\gamma \in \mathrm{VP}(G')$

$$\mathrm{RD}_{\hat{M}}(\mathrm{GOTO}_{\hat{M}}(\hat{q}_s, \gamma)) = \mathrm{RD}(\gamma)$$

erfüllt. Ein aus einer derartig erweiterten allgemeinen LR(k)-Maschine $(\hat{M}, \mathrm{RC}_{\hat{M}}, \mathrm{RD}_{\hat{M}})$ konstruierter allgemeiner LR(k)-Parser ist dann reduktionsdeterminiert. (Ohne Beweis.)

Allgemeine LR(k)-Parser, die mit oberstem Kellersymbol und k-Lookahead nicht nur (wie in reduktionsdeterminierten LR(k)-Parsern) zu entscheiden gestatten, *ob*, sondern sogar gegebenenfalls (wie in kanonischen LR(k)-Parsern), *nach welcher Regel von G* zu reduzieren ist, nennen wir *reduktionsregeldeterminiert.* Zu solchen Parsern gelangt man ebenso geradlinig:

Auf $\mathrm{VP}(G')$ definiere man

$$\mathrm{RR}(\gamma) = \{(A \to \omega, y) : \text{es ex. irgendein } k\text{-Item } [A \to \omega\cdot, y] \in \mathrm{VALID}_k(\gamma)\}$$

und betrachte allgemeine LR(k)-Maschinen $(\hat{M}, \mathrm{RC}_{\hat{M}})$, denen eine weitere Stützinformationskomponente $\mathrm{RR}_{\hat{M}}$ hinzugefügt werden kann, die für jedes $\gamma \in \mathrm{VP}(G')$

$$\mathrm{RR}_{\hat{M}}(\mathrm{GOTO}_{\hat{M}}(\hat{q}_s, \gamma)) = \mathrm{RR}(\gamma)$$

erfüllt. Die aus derartigen Maschinen $(\hat{M}, \mathrm{RC}_{\hat{M}}, \mathrm{RR}_{\hat{M}})$ konstruierten allgemeinen LR(k)-Parser weisen die genannte Eigenschaft dann auf. (Ohne Beweis.)

8 Abgeleitete LR-Parsertechniken: SLR(k)-, LALR(k)- und ILALR(k)-Parser

Die Vorteile der großen Mächtigkeit und der linearen Laufzeit[31] (implementierter) kanonischer LR(k)-Parser werden erkauft mit demjenigen Nachteil, der als Hauptkritik an diesen Parsern weithin bekannt ist: ihrer erheblichen Größe. In der Theorie ist wohl keine günstigere als die Abschätzung $O(2^{|T|^k \cdot |G| + k \cdot \log|T| + \log|G|})$ für die Größe des kanonischen LR(k)-Parsers von G bekannt.[32] Zwar scheinen kanonische LR(0)-Parser für Programmiersprachengrammatiken (und mithin auch entsprechende SLR(1)- oder LALR(1)-Parser) Purdom (1974) zufolge nur linear mit der Grammatikgröße zu wachsen. Selten jedoch sind solche Grammatiken LR(0), und bereits für $k = 1$ weisen die kanonischen LR(k)-Maschinen oft mehrere Tausend Zustände auf. Mit wachsendem k steigt diese Zustandsanzahl rapide, so daß in der Vergangenheit eine Implementierung von kanonischen LR(k)-Parsern ($k \geq 1$) sich verbot oder zumindest „schmerzte“.

Der am häufigsten begangene Weg zur Minderung der Parsergröße setzt abgewandelte LR(k)-Parserverfahren ein: Die von DeRemer (1969) eingeführten SLR(k)- und LALR(k)-Parser werden aus der Struktur kanonischer LR(0)-Maschinen gewonnen und weisen dementsprechend ungefähr deren Größe auf; die k-Lookaheads der Aktionen des abgelesenen Parsers sind jedoch gegenüber dem kanonischen LR(0)-Parser verfeinert. Beide Verfahren konstruieren Rechtsparser, die für weniger Grammatiken nichtdeterministisch sind als kanonische LR(0)-Parser, ihre Lookahead-Information ist jedoch nur eine mehr (LALR(k)) oder weniger (SLR(k)) gute Approximation der exakten Lookaheads von LR(k)-Parsern. Das von beiden mächtigere LALR(k)-Verfahren hat, besonders für $k = 1$, aufgrund des guten Mächtigkeits-Größe-Kompromisses seit langem eine anhaltend weite Verbreitung in existierenden Parsergeneratoren gefunden.[33]

---

[31] Siehe z. B. (Sippu und Soisalon-Soininen 1990), Theorem 6.34.
Daß auch allgemeine LR(k)-Parser eine bezüglich der Eingabelänge lineare Laufzeit aufweisen, ist offenkundig.

[32] (Sippu und Soisalon-Soininen 1990), Theorem 6.45.

[33] Chapman (1987) stellt in einem eigenen Kapitel einige verbreitete Parsergeneratoren etwas ausführlicher

Beiden Verfahren ist zu eigen, daß die von ihnen beschriebenen Grammatikklassen sehr „technisch“ charakterisiert sind.[34] Insbesondere geht bei der Parserkonstruktion die Bezugnahme auf die Itemmengen der kanonischen LR(0)-Maschinen so weit, daß sich aus den abstrakteren allgemeinen LR(0)-Maschinen, wie wir sehen werden, nicht generell gleichwertige SLR(k)- bzw. LALR(k)-Parser konstruieren lassen. Es ist es deshalb durchaus nicht sofort offensichtlich, wie „allgemeine SLR(k)- und LALR(k)-Parser“ adäquat zu definieren sind. Da die klassischen Definitionen von SLR(k)- und LALR(k)-Parsern fest mit klassischen kanonischen LR(0)-Maschinen verbunden sind, werden wir zur Terminologievereinheitlichung diese Parser fortan mit dem Adjektiv „kanonisch“ belegen.

8.1 Zur adäquaten Definition allgemeiner SLR(k)-Parser

Der *kanonische SLR(k)-Parser* $(\tilde{M},\tau) = (\tilde{M}^G_{\mathrm{c,SLR}(k)}, \tau^G_{\mathrm{c,SLR}(k)})$ *zu* G wird auf der Grundlage der kanonischen LR(0)-Maschine $\hat{M}_\mathrm{c}$ zu G konstruiert und besitzt für jeden Zustand $\hat{q}$ von $\hat{M}_\mathrm{c}$ je eine Shift-Aktion

$$r = \quad \hat{q}\,|\,a\,x \to \hat{q}\,a\,\mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q},a)\,|\,x$$

mit $x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta\,\mathrm{FOLLOW}'_k(A))$ für jedes $[A \to \alpha\cdot a\beta] \in \hat{q}$, $a \in T$, wobei $\tau(r) = \varepsilon$ gesetzt wird; ferner je eine Reduce-Aktion

$$r = \quad \hat{q}\,X_1\,\mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q},X_1)\ldots X_m\,\mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q},X_1\ldots X_m)\,|\,y \to \hat{q}\,A\,\mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q},A)\,|\,y$$

mit $y \in \mathrm{FOLLOW}'_k(A) = \mathrm{FOLLOW}_{G',k}(A)$ für jedes $[A \to \cdot\,\omega] = [A \to \cdot\,X_1\ldots X_m] \in \hat{q}$ ($m \geq 0$, $X_1,\ldots,X_m \in V$), wobei $\tau(r) = A \to \omega$ gesetzt wird. Die weiteren Komponenten von $\tilde{M}$ sind offensichtlich. G ist eine SLR(k)-Grammatik, wenn der kanonische SLR(k)-Parser zu G deterministisch ist und $S \Rightarrow^+ S$ in G nicht gilt. Kanonische SLR(0)- und LR(0)-Parser sind identisch.

---

vor: *LR* (LR(1)) von Whetherell und Shannon (1981); *yacc* (LALR(1)) von Johnson (1978); die Parsergeneratorkomponente des *HLP*-Systems (LALR(1)) von Räihä et al. (1983). Jüngeren Datums sind z. B. der *yacc*-kompatible *bison* des GNU Projekts, *PGS* (LALR(1)) von Klein und Martin (1989) und *Lalr* (LALR(1)) von Grosch (1990).

[34] In besonderem Maße gilt dies für das LALR(k)-Verfahren. Aus (DeRemer und Pennello 1982) stammt das Zitat *„It is desirable to have a definition of LALR(1) grammar that does not involve the parser, but we know of no reasonable way to do this.“*

Liegt nun statt $\hat{M}_\mathrm{c}$ eine beliebige allgemeine LR(0)-Maschine $(\hat{M}, \mathrm{RC}_{\hat{M}})$ vor, so können daraus Reduce-Aktionen mit SLR(k)-Lookahead problemlos gewonnen werden: Im wesentlichen ist die Bedingung „$[A \to \cdot\, \omega] = [A \to \cdot\, X_1 \ldots X_m] \in \hat{q}$“ zu ersetzen durch „$A \,\mathbf{|}\, \in \mathrm{RC}_{\hat{M}}(\hat{q})$ und $A \to \omega = A \to X_1 \ldots X_m \in P$“. Der Lookahead-Anteil y ist nicht von der Itemmengen-Konstruktion abhängig! Jedoch können ähnlich Shift-Aktionen nicht erzeugt werden: Für die Bestimmung der zu „$a \,\mathbf{|}\, \in \mathrm{RC}_{\hat{M}}(\hat{q})$“ anstatt „$[A \to \alpha \cdot a\beta] \in \hat{q}$“ noch passenden Lookahead-Verlängerungen x sind die Komponenten β und A nicht verfügbar.

Forsche Leser könnten hier mit dem Vorschlag aufwarten, diese an der Itemmenge $\mathrm{VALID}_0(\gamma) = \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, \gamma)$ in $\hat{M}_\mathrm{c}$ vorhandene, aber am Zustand $\hat{q} = \mathrm{GOTO}_{\hat{M}}(\hat{q}_\mathrm{s}, \gamma)$ in $\hat{M}$ fehlende Information „herbeizuzaubern“, indem zu „$a \,\mathbf{|}\, \in \mathrm{RC}_{\hat{M}}(\hat{q})$“ die Lookahead-Verlängerungen zugehöriger Shift-Aktionen als $x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta\, \mathrm{FOLLOW}'_k(A))$ für alle $[A \to \alpha \cdot a\beta] \in \mathrm{VALID}_0(\gamma) = \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, \gamma)$ bestimmt werden. Dies ist allerdings problematisch, denn verschiedene lebensfähige Präfizes γ, die in $\hat{M}$ alle zu $\hat{q}$ führen, können in $\hat{M}_\mathrm{c}$ zu verschiedenen Zuständen $\mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, \gamma)$ führen, $\hat{M}_\mathrm{c}$ mag also Kontexte unterscheiden, die $\hat{M}$ nicht unterscheidet (und umgekehrt), und es können nach dieser Methode im Falle $k > 1$ in der Tat in $\tilde{M}$ Nichtdeterminismen vorhanden sein, die $\tilde{M}$ nicht aufweist (und wohl auch umgekehrt).

Ob auf diese Weise ein deterministischer oder nichtdeterministischer Parser konstruiert wird, hängt damit ab von der Struktur der konkreten zugrunde gelegten allgemeinen LR(0)-Maschine. Dies ist natürlich inakzeptabel! Abbildung 9 zeigt eine kleine SLR(2)-Grammatik, deren wie beschrieben versuchsweise aus der allgemeinen LR(0)-Maschine $\hat{M} = \hat{M}_\mathrm{m}$ aufgebaute „SLR(2)“-Shift- und -Reduce-Aktionen einen Nichtdeterminismus beinhalten. Für $k \leq 1$ tritt das Problem nicht auf. Allgemein jedoch darf, wie wir gesehen haben, nicht vollständig von den Komponenten β und A eines 0-Items $[A \to \alpha \cdot a\beta]$ abstrahiert werden, wenn „allgemeine SLR(k)-Parser zu G“ adäquat definiert werden sollen.

Ein gangbarer Weg ist, zu G eine „allgemeine SLR(k)-Maschine“ $(\hat{M}, \mathrm{RC}^{\mathrm{SLR}(k)}_{\hat{M}})$ als Ausgangspunkt zu wählen, welche in ihrer zweiten Komponente den $\hat{M}$-Zuständen in naheliegender Weise bereits korrekte „SLR(k)-Rechtskontexte“ zuordnet. Eine solche Maschine kann leicht aus den Itemmengen der kanonischen oder redundanzreduzierten LR(0)-Maschine gewonnen und sodann minimiert werden. (In Abbildung 9 ist die minimierte Ergebnismaschine für das dortige Beispiel mit gezeigt.) Daß der daraus wie für LR(k) abzulesende „allgemeine SLR(k)-Parser zu G“ genau dann deterministisch ist, wenn dasselbe auch für den klassischen kanonischen SLR(k)-Parser gilt, ist unschwer zu sehen. Wir

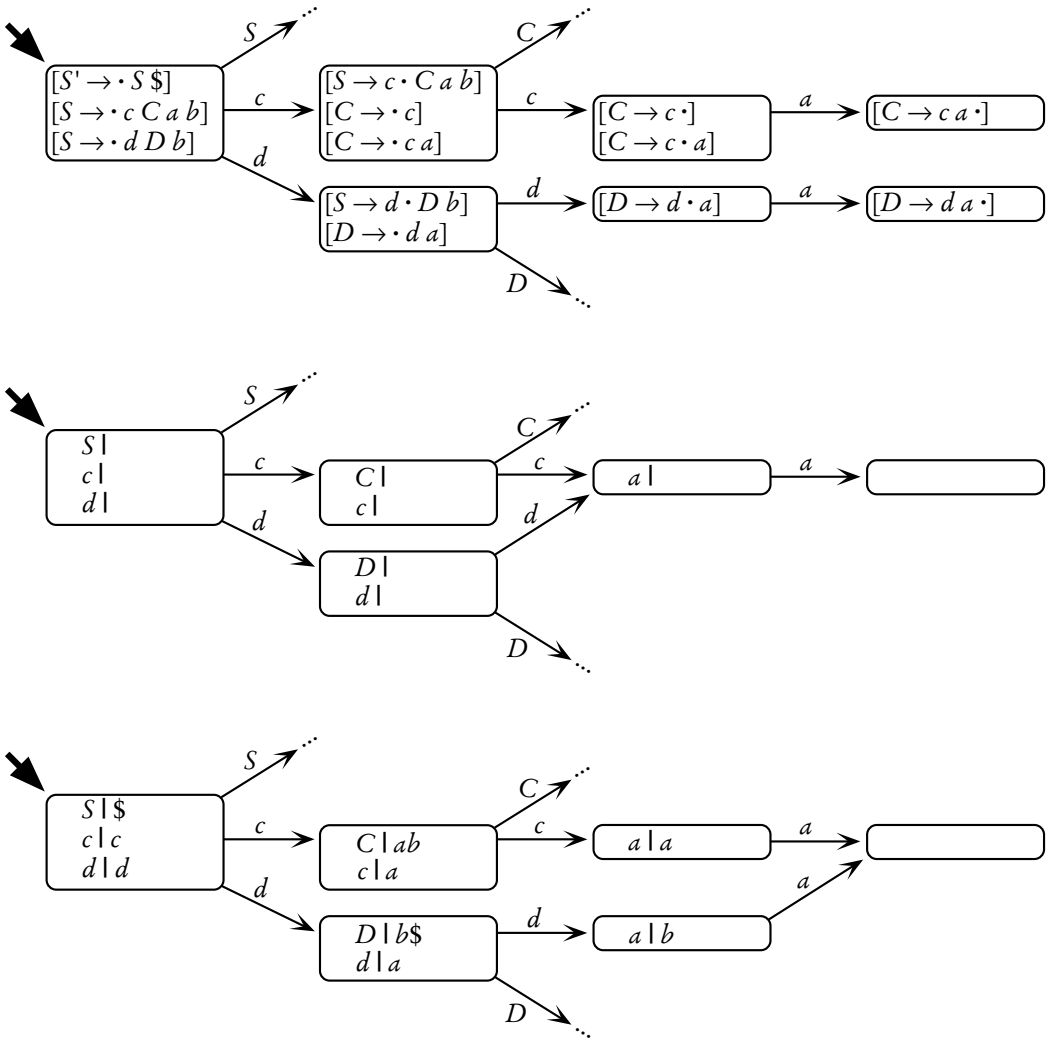


Abbildung 9: Ausschnitte der kanonischen LR(0)-Maschine $\hat{M}_{\mathrm{c}}$ (oben) und der minimalen allgemeinen LR(0)-Maschine $\hat{M}_{\mathrm{m}}$ (Mitte) zur Beispielgrammatik G mit den Regeln $S \to c\,C\,a\,b \mid d\,D\,b$, $C \to c \mid c\,a$, $D \to d\,a$ machen klar, daß die im Text beschriebene Technik der Gewinnung von „SLR(k)"-Aktionen aus allgemeinen LR(0)-Maschinen inakzeptabel ist: Der aus $\hat{M}_{\mathrm{c}}$ gewonnene kanonische SLR(2)-Parser ist deterministisch; der aus $\hat{M}_{\mathrm{m}}$ durch Aufsuchen von Iteminformation aus $\hat{M}_{\mathrm{c}}$ gewonnene ist nichtdeterministisch! (Shift-Aktion bei lebensfähigem Präfix $d\,d$ und Lookahead $a\,b$ versus Reduce-Aktion nach $C \to c$ bei lebensfähigem Präfix $c\,c$ und Lookahead $a\,b$.) Statt dessen ist für $k > 1$ eine allgemeine SLR(k)-Maschine (unten für $k = 2$) einzuführen, die die SLR(k)-Rechtskontexte korrekt widergibt.

werden jedoch auf Einzelheiten nicht mehr eingehen, da zum einen SLR(k)-Parser für $k > 1$ in der Praxis kaum von Interesse sind. Zudem wird der skizzierte, etwas mühsame Weg zur Gewinnung minimaler allgemeiner SLR(k)-Maschinen dem Adjektiv *simple* aus der Bedeutung des Akronyms SLR schwerlich noch gerecht.

8.2 Eine Variante des LALR(k)-Verfahrens und die adäquate Definition allgemeiner LALR(k)-Parser

Der *kanonische LALR(k)-Parser* $(\tilde{M},\tau) = (\tilde{M}^G_{\mathrm{c,LALR}(k)}, \tau^G_{\mathrm{c,LALR}(k)})$ *zu* G wird auf der Grundlage einerseits der kanonischen LR(0)-Maschine $\hat{M}_\mathrm{c} = \hat{M}^G_{\mathrm{c,LR}(0)}$ zu G (mit Startzustand $\hat{q}_{\mathrm{s,c}}$), andererseits der kanonischen LR(k)-Maschine $\hat{M}^k_\mathrm{c} = \hat{M}^G_{\mathrm{c,LR}(k)}$ zu G (mit Startzustand $\hat{q}^k_{\mathrm{s,c}}$) konstruiert. Dabei liefert die Struktur von $\hat{M}_\mathrm{c}$ die Shift- und Reduce-Aktionen mit Ausnahme der Lookahead-Anteile. Letztere werden aus der kanonischen LR(k)-Maschine bezogen, indem die dortigen Zuordnungen von k-Lookaheads zu lebensfähigen Präfizes auf die Struktur der LR(0)-Maschine herunterprojiziert werden, und zwar von der feineren Einteilung der lebensfähigen Präfizes durch die Itemmengen der LR(k)-Maschine $\hat{M}^k_\mathrm{c}$ auf die gröbere durch die der LR(0)-Maschine $\hat{M}_\mathrm{c}$. Einer k-Itemmenge I wird dabei die 0-Itemmenge

$$core(I) = \{[A \to \alpha \cdot \beta] : [A \to \alpha \cdot \beta, y] \in I\}$$

zugeordnet. Es gilt $core(\mathrm{GOTO}_{\hat{M}^k_\mathrm{c}}(\hat{q}^k_{\mathrm{s,c}}, \varphi)) = \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, \varphi)$ für alle $\varphi \in \mathrm{VP}(G')$.[35] (Wir beziehen uns unten bei der Formulierung der Parseraktionen nicht explizit auf *core*, sondern verwenden *core* nur implizit.)

In $\tilde{M}$ ist für jedes $\delta \in \mathrm{VP}(G')$

$$r = \quad \hat{q} \,|\, ax \to \hat{q}\, a\, \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}, a) \,|\, x$$

mit $\tau(r) = \varepsilon$ eine Shift-Aktion (von a) genau dann, wenn $\hat{q} = \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, \delta)$, $a \in T$, $x \in T'^*$ ist und ein LR(k)-gültiges Item $[A \to \alpha \cdot a\beta, z] \in \mathrm{VALID}_k(\delta)$ (d.h. $[A \to \alpha \cdot a\beta, z] \in \mathrm{GOTO}_{\hat{M}^k_\mathrm{c}}(\hat{q}^k_{\mathrm{s,c}}, \delta)$) existiert, so daß $x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)$ gilt. (Wir bemerken, daß $[A \to \alpha \cdot a\beta] \in core(\mathrm{GOTO}_{\hat{M}^k_\mathrm{c}}(\hat{q}^k_{\mathrm{s,c}}, \delta)) = \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}_{\mathrm{s,c}}, \delta)$ gilt.) Weiter ist

$$r = \quad \hat{q}\, X_1\, \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}, X_1) \ldots X_m\, \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}, X_1 \ldots X_m) \,|\, y \to \hat{q}\, A\, \mathrm{GOTO}_{\hat{M}_\mathrm{c}}(\hat{q}, A) \,|\, y$$

[35] (Sippu und Soisalon-Soininen 1990), Theorem 6.15.

mit $\tau(r) = r'$ eine Reduce-Aktion (nach Regel r') genau dann, wenn $\hat{q} = \mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \delta)$ ist, $m \geq 0$, $X_1, \ldots, X_m \in V$, $r' = A \to X_1 \ldots X_m \in P$ und $y \in T'^*$ sind und $[A \to X_1 \ldots X_m \cdot, y] \in \mathrm{VALID}_k(\delta' X_1 \ldots X_m)$ (d.h. $[A \to X_1 \ldots X_m \cdot, y] \in \mathrm{GOTO}_{\hat{M}_c^k}(\hat{q}_{s,c}^k, \delta' X_1 \ldots X_m)$) ist für irgendein $\delta' \in \mathrm{VP}(G')$, für das $\mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \delta X_1 \ldots X_m) = \mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \delta' X_1 \ldots X_m)$ gilt. (Man beachte, daß hier das k-Item $[A \to \cdot X_1 \ldots X_m, y]$ zwar stets in $\mathrm{VALID}_k(\delta')$, aber nicht immer in $\mathrm{VALID}_k(\delta)$ ist; dies wird unten noch wichtig werden.) Initialer Kellerinhalt von $\tilde{M}$ ist $\hat{q}_{s,c}$, einziger finaler Kellerinhalt ist $\hat{q}_{s,c} S\, \mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, S)$. Die weiteren Komponenten von $\tilde{M}$ sind offensichtlich. Kanonische LALR(0)-Parser und kanonische LR(0)-Parser sind identisch. G ist genau dann eine LALR(k)-Grammatik, wenn der kanonische LALR(k)-Parser zu G deterministisch ist und $S \Rightarrow^+ S$ nicht gilt.

Zu strukturell demselben Parser für eine gegebene Grammatik G gelangt man, indem nicht wie oben die Itemmengen der LR(k)-Maschine zu G – ihrer Lookaheads wegen – auf ihre jeweiligen LR(0)-Pendants mit gleichem *core* gedacht *projiziert* werden, sondern eine zur LR(0)-Maschine isomorphe sogenannte LALR(k)-Maschine durch *Vereinigung* jeweils der LR(k)-Itemmengen mit gleichem *core* konstruiert wird und aus dieser Maschine Shift- und Reduce-Aktionen wie für einen kanonischen LR(k)-Parser abgelesen werden.

Beispiel: Als ein Beispiel für eine Grammatik, die LR(k) für jedes $k \geq 1$, jedoch nicht LALR(k) für alle k ist, betrachten wir $G = G_{abc}$ mit den Regeln[36]

$$S \to a\,A\,a \mid b\,A\,b \mid a\,B\,b \mid b\,B\,a\,,$$
$$A \to c\,,$$
$$B \to c\,.$$

Abbildung 10 zeigt dazu die kanonische LR(k)-Maschine für $k \geq 2$. (Für $k = 1$ sind lediglich die Lookahead-Anteile der k-Items zu kürzen.) Mit den Zustandsabkürzungen aus der Abbildung fallen bei leeren Lookahead-Anteilen ($k = 0$) die Zustände 3 und 4 zu einem einzigen Zustand, sagen wir 3′, zusammen, was zur Folge hat, daß der kanonische LALR(k)-Parser ($k \geq 2$) zu G_{abc} in den Reduce-Aktionenpaaren

$$2\,c\,3' \mid a\$ \to 2\,A\,6 \mid a\$ \quad \text{und} \quad 2\,c\,3' \mid a\$ \to 2\,B\,7 \mid a\$\,,$$
$$2\,c\,3' \mid b\$ \to 2\,A\,6 \mid b\$ \quad \text{und} \quad 2\,c\,3' \mid b\$ \to 2\,B\,7 \mid b\$\,,$$
$$5\,c\,3' \mid a\$ \to 5\,A\,8 \mid a\$ \quad \text{und} \quad 5\,c\,3' \mid a\$ \to 5\,B\,9 \mid a\$ \quad \text{sowie}$$
$$5\,c\,3' \mid b\$ \to 5\,A\,8 \mid b\$ \quad \text{und} \quad 5\,c\,3' \mid b\$ \to 5\,B\,9 \mid b\$$$

[36] (Sippu und Soisalon-Soininen 1990), S. 67.

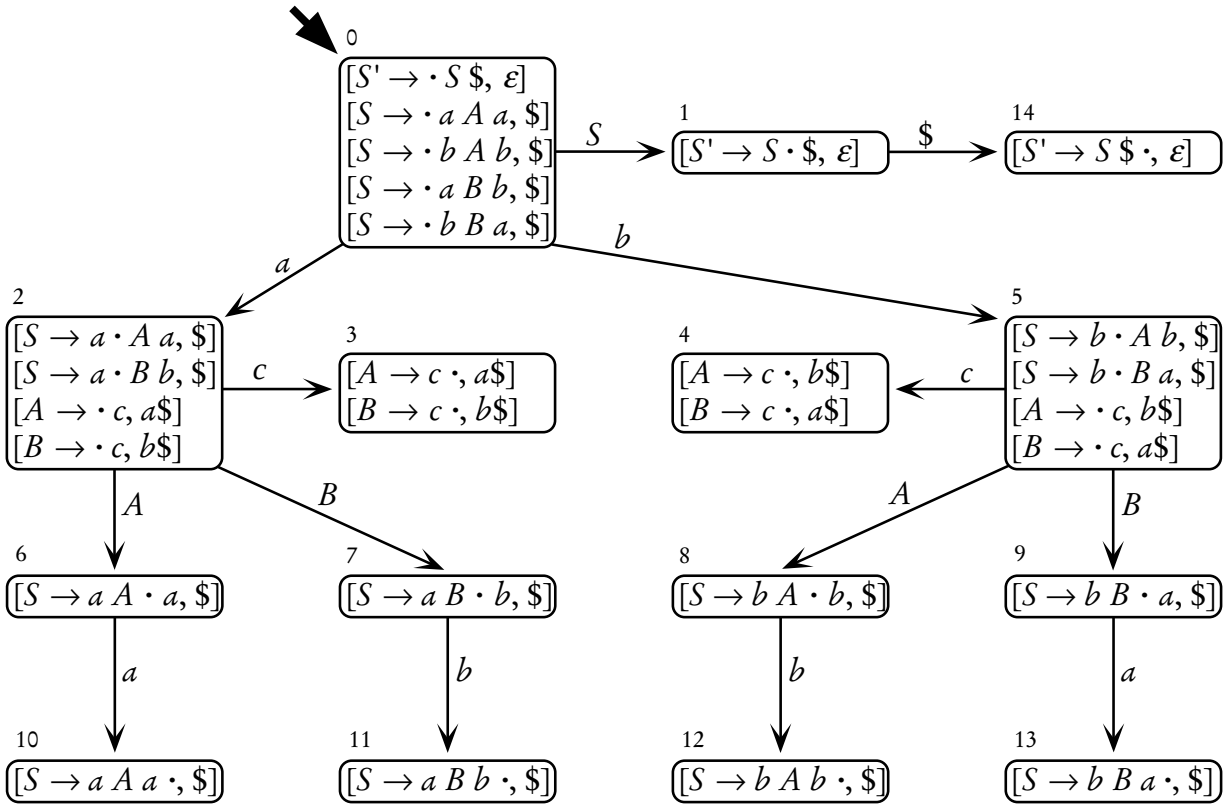


Abbildung 10: Die kanonische LR(k)-Maschine ($k \geq 2$) zur Grammatik G_{abc} mit den Regeln $S \to a\,A\,a \mid b\,A\,b \mid a\,B\,b \mid b\,B\,a$, $A \to c$, $B \to c$. Für die kanonische LR(1)-Maschine ist lediglich der Lookahead-Anteil der Items zu kürzen. Mit ausgeblendeten Lookahead-Anteilen ($k = 0$) fallen die Zustände 3 und 4 zusammen.
G_{abc} ist ein Beispiel für eine LR(1)-Grammatik, die für kein k eine LALR(k)-Grammatik ist.
Allerdings ist G_{abc} eine ILALR(1)-Grammatik.

nichtdeterministisch ist. (Der LALR(1)-Parser zu G_{abc} weist dieselben konfliktbehafteten Regelpaare auf, jedoch mit verkürzten Lookahead-Anteilen.)

Diese Nichtdeterminismen sind allgemein dadurch zu erklären, daß mittels der Projektion aus der filigraner strukturierten LR(k)- ($k \geq 1$) auf die gröber strukturierte LR(0)-Maschine Rechtskontexte von zwar kanonisch LR(0)-, aber nicht LR(k)-äquivalenten ($k \geq 1$) lebensfähigen Präfizes zusammenvereinigt werden. □

Bemerkenswert ist: LALR(k)-Parser, so wie von DeRemer (1969) formuliert, ignorieren präziseres vorhandenes Wissen über mögliche LR(k)-Rechtskontexte! Es ist möglich, auf derselben strukturellen Basis – der kanonischen LR(0)-Maschine für die Grundstruktur der Parseraktionen, der kanonischen LR(k)-Maschine für die Lookahead-Verfeinerung – einen Rechtsparser mit k-Lookahead zu konstruieren, der identisch mit DeRemers kanonischem LALR(k)-Parser ist bis auf die Tatsache, daß seine Aktionen nur eine Teilmenge der Aktionen des kanonischen LALR(k)-Parsers sind. Die Idee ist, die Lookahead-Information für Reduce-Aktionen anschaulich in der skizzierten LALR(k)-Maschine am *Beginn* der Handle-Erkennung abzugreifen statt an deren Ende, wo sie weniger präzise ist. Wir erhalten den verbesserten Kellertransduktor, indem wir in der LALR(k)-Bildungsvorschrift für Reduce-Aktionen die folgende Textersetzung vornehmen.

Alt: … eine Reduce-Aktion … genau dann, wenn … und $[A \to X_1 \ldots X_m \cdot, y] \in \mathrm{VALID}_k(\delta' X_1 \ldots X_m)$ … ist für irgendein $\delta' \in \mathrm{VP}(G')$, für das $\mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \delta X_1 \ldots X_m) = \mathrm{GOTO}_{\hat{M}_c}(\hat{q}_{s,c}, \delta' X_1 \ldots X_m)$ gilt.

Neu: … eine Reduce-Aktion … genau dann, wenn … und $[A \to X_1 \ldots X_m \cdot, y] \in \mathrm{VALID}_k(\delta X_1 \ldots X_m)$ ist.

Diese neue Bildungsvorschrift stellt formal sogar eine deutliche Vereinfachung dar, da wir darauf verzichten können, Lookaheads umständlich lebensfähigen Präfizes zuzuordnen, zu denen sie nicht „gehören“. Sie produziert damit für weniger Lookaheads y Reduce-Aktionen als die alte. Denn sind allgemein γ und γ' aus $\mathrm{VP}(G')$ kanonisch LR(k)-äquivalent (d.h. $\mathrm{VALID}_k(\gamma) = \mathrm{VALID}_k(\gamma') \neq \emptyset$), so gilt (insbesondere für $\gamma = \gamma'$) bekanntlich

$$\gamma' = \delta'\alpha' \wedge [A \to \alpha\alpha' \cdot \alpha'', y] \in \mathrm{VALID}_k(\gamma') \quad \Leftrightarrow \quad \gamma = \delta\alpha' \wedge [A \to \alpha \cdot \alpha'\alpha'', y] \in \mathrm{VALID}_k(\delta)\,.$$[37]

Sind allerdings γ und γ' nur kanonisch LR(0)-äquivalent, so gilt lediglich noch die Schlußrichtung „$\Leftarrow$“.

Beispielsweise sind für $G = G_{abc}$ und $k \geq 2$ die lebensfähigen Präfizes $\gamma = ac$ und $\gamma' = bc$ LR(0)-, nicht aber LR(k)-äquivalent ($k \geq 1$), und aus $[A \to c\cdot, a\$] \in \mathrm{VALID}_k(ac)$ folgt $[A \to \cdot c, a\$] \in \mathrm{VALID}_k(a)$, aber eben nicht $[A \to \cdot c, a\$] \in \mathrm{VALID}_k(b)$. Dennoch wird eine Reduce-Aktion nach $A \to c$ bei Lookahead $a\$$ überflüssigerweise auch für das lebensfähige Präfix bc erzeugt.

Die modifizierte Bildungsvorschrift produziert zu G_{abc} statt der obigen konfliktbehafteten Reduce-Aktionenpaare nur noch die Aktionen

$$\begin{aligned}
&2\,c\,3' \mid a\$ \to 2\,A\,6 \mid a\$\,,\\
&2\,c\,3' \mid b\$ \to 2\,B\,7 \mid b\$\,,\\
&5\,c\,3' \mid a\$ \to 5\,B\,9 \mid a\$ \quad \text{sowie}\\
&5\,c\,3' \mid b\$ \to 5\,A\,8 \mid b\$
\end{aligned} \tag{22}$$

für $k \geq 2$ bzw. selbige mit gekürzten Lookahead-Anteilen für $k = 1$, und die für $k \geq 1$ resultierenden Kellertransduktoren sind deterministisch!

Allgemein ist damit erklärt, weswegen die wie gesehen modifizierte Bildungsvorschrift für Reduce-Aktionen präzisere Lookahead-Anteile einsetzt, als in kanonischen LALR(k)-Parsern nach DeRemer verwendet werden. Den mit dieser Änderung resultierenden Kellertransduktor zu G wollen wir den *kanonischen ILALR(k)-Parser* $(\tilde{M}^G_{\mathrm{c,ILALR}(k)}, \tau^G_{\mathrm{c,ILALR}(k)})$ *zu* G nennen, wobei das Akronym „ILALR“ für „Improved Look-Ahead LR“ steht. G nennen wir eine *ILALR(k)-Grammatik,* wenn der kanonische ILALR(k)-Parser zu G deterministisch ist und $S \Rightarrow^+ S$ in G nicht gilt. Es ist unschwer zu zeigen, daß der kanonische ILALR(k)-Parser zu G ein Rechtsparser ist: Der kanonische LR(k)-Parser zu G verhält sich wie der kanonische ILALR(k)-Parser zu G und dieser wie der kanonische LR(0)-Parser zu G. Auf einen detaillierten Beweis verzichten wir.

Natürlich ist für $k \geq 1$ nicht jede LR(k)-Grammatik eine ILALR(k)-Grammatik. Um dies zu zeigen, kann die obige Grammatik G_{abc} geringfügig erweitert werden zur Grammatik

[37] Eine analoge Veränderung für LR(k)-Parser – eine Reduce-Aktion aufgrund der Gültigkeit von $[A \to X_1 \ldots X_m \cdot, y] \in \mathrm{VALID}_k(\gamma X_1 \ldots X_m)$ statt der von $[A \to \cdot X_1 \ldots X_m, y] \in \mathrm{VALID}_k(\gamma)$ zu bilden – bleibt daher ohne Folgen: Es resultieren die gleichen Parseraktionen.

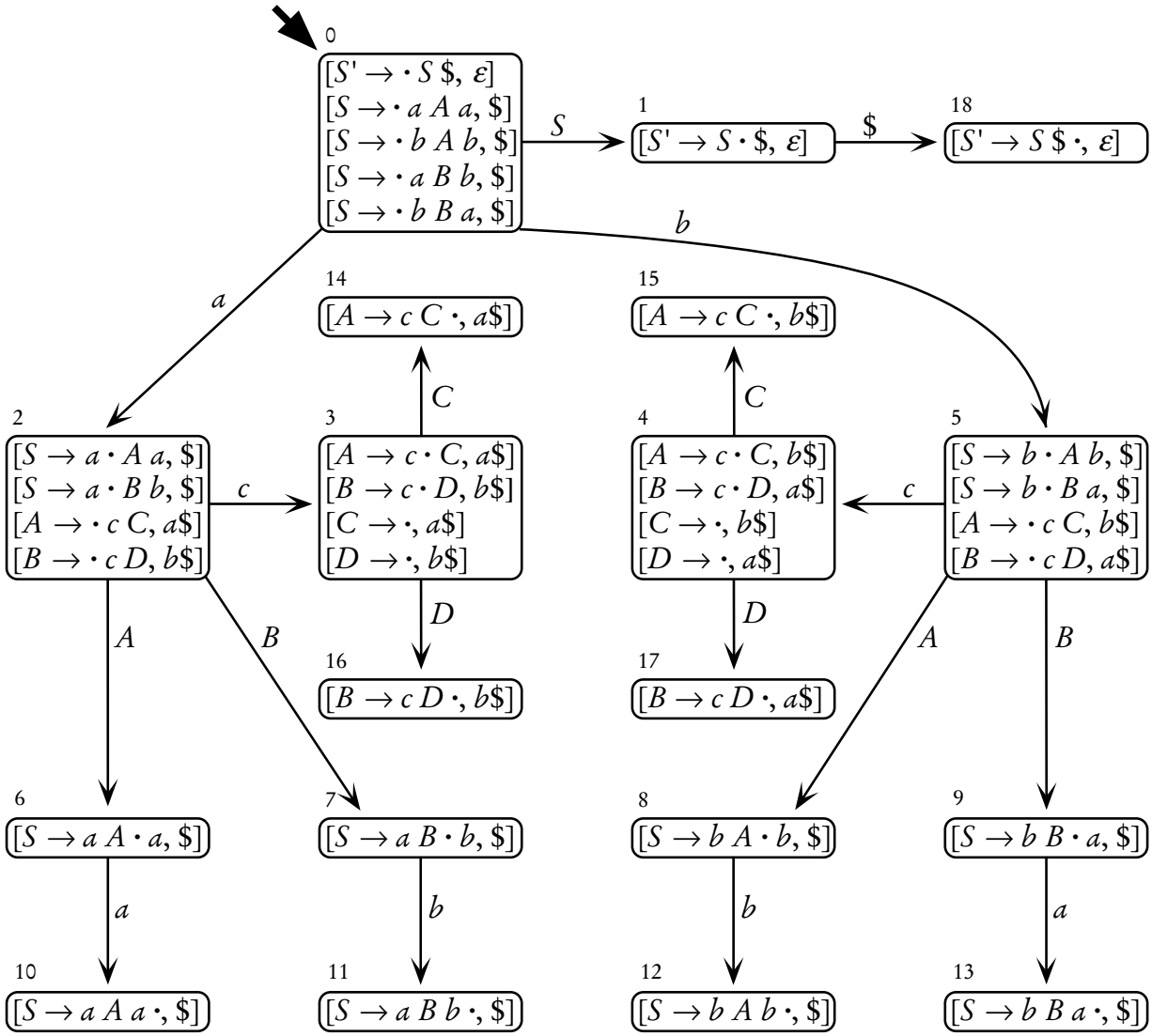


Abbildung 11: Die kanonische LR(k)-Maschine ($k \geq 2$) zur Grammatik $G_{abc\varepsilon}$ mit den Regeln $S \to a\,A\,a \mid b\,A\,b \mid a\,B\,b \mid b\,B\,a$, $A \to c\,C$, $B \to c\,C$, $C \to \varepsilon$, $D \to \varepsilon$. Für die kanonische LR(1)-Maschine ist lediglich der Lookahead-Anteil der Items zu kürzen. Mit wegfallenden Lookahead-Anteilen ($k = 0$) fallen die Zustände 3 und 4, 14 und 15 sowie 16 und 17 zusammen.
$G_{abc\varepsilon}$ ist ein Beispiel für eine LR(1)-Grammatik, die für kein k eine ILALR(k)-Grammatik ist.

$G_{abc\varepsilon}$ mit den Regeln

$$
\begin{aligned}
&S \to a\,A\,a \mid b\,A\,b \mid a\,B\,b \mid b\,B\,a\,,\\
&A \to c\,C\,,\\
&B \to c\,D\,,\\
&C \to \varepsilon\,,\\
&D \to \varepsilon\,.
\end{aligned}
$$

Die kanonische LR(k)-Maschine zu $G_{abc\varepsilon}$ zeigt Abbildung 11 für $k \geq 2$. (Für $k = 1$ sind wieder die Lookahead-Anteile zu kürzen.) Mit den Zustandsabkürzungen aus der Abbildung fallen bei leeren Lookahead-Anteilen ($k = 0$) die Zustände 3 und 4 zusammen, sagen wir zum Zustand $3'$; weiter fallen 14 und 15 zu $14'$ sowie 16 und 17 zu $16'$ zusammen. Die Zusammenfassung der Zustände 3 und 4 trifft hier den *Beginn* der Handle-Erkennung für die Regeln $C \to \varepsilon$ und $D \to \varepsilon$. Und während sich der kanonische LR(1)-Parser zu $G_{abc\varepsilon}$ als deterministisch erweist, ist nun der ILALR(k)-Parser ($k \geq 2$) zu $G_{abc\varepsilon}$ nichtdeterministisch in den Aktionenpaaren

$$
\begin{aligned}
&3'\,|\,a\,\$ \to 3'\,C\,14'\,|\,a\,\$ \quad \text{und} \quad 3'\,|\,a\,\$ \to 3'\,D\,16'\,|\,a\,\$ \quad \text{sowie}\\
&3'\,|\,b\,\$ \to 3'\,C\,14'\,|\,b\,\$ \quad \text{und} \quad 3'\,|\,b\,\$ \to 3'\,D\,16'\,|\,b\,\$\,.
\end{aligned}
$$

(Für $k = 1$ sind die Lookahead-Anteile zu kürzen, die Nichtdeterminismen bleiben dabei bestehen.)

Für die größere Mächtigkeit von deterministischen kanonischen ILALR(k)-Parsern im Vergleich zu deterministischen kanonischen LALR(k)-Parsern ist ein Preis zu bezahlen: Die Eigenschaft, reduktionsregeldeterminiert zu sein, weisen kanonische ILALR(k)-Parser im Unterschied zu den kanonischen LALR(k)-Parsern i. a. nicht mehr auf, wie man sich etwa an den Aktionenpaaren (22) des kanonischen ILALR(k)-Parsers zur Beispielgrammatik G_{abc} überzeugt. Kanonische ILALR(k)-Parser *müssen* also allgemein auch tieferen Kellerkontext als nur das oberste Kellersymbol betrachten. Eine in der Praxis gegenüber den üblichen tabellengesteuerten Implementierungen kaum ineffizientere Strategie dürfte sein, die Lookahead-Informationen am Handlebeginn statt am Handleende vorzuhalten und darüberhinaus für jeden zuoberst gekellerten Zustand bereitzuhalten, ob er ein Handleende anzeigt und, falls ja, wie lang die möglichen Handle sind. Es kann dann entsprechend gezielt ein tiefer gekellerter Zustand inspiziert werden, ob er einen passenden Handlebeginn anzeigt und für das aktuelle Lookahead eine geeignete Reduktion durchzuführen ist. Im Keller kann auf die Grammatiksymbolfolge verzichtet werden, die das lebensfähige Präfix

angibt, da kanonische ILALR(k)-Parser die Eigenschaft des eindeutigen Access-Symbols aufweisen. Ein Praxisvergleich zwischen implementierten kanonischen ILALR(k)- und LALR(k)-Parsern – in puncto Laufzeit- und Speichereffizienz und „wie oft“ ILALR(k)-Parser noch deterministisch sind, wo LALR(k)-Parser dies nicht mehr sind – verspricht spannend zu werden.

Analog unserer Behandlung von SLR(k)-Parsern sollten wir uns auch hier die Frage stellen, ob statt der zur Konstruktion verwendeten kanonischen LR(0)-Maschine nicht auch eine *beliebige allgemeine* LR(0)-Maschine eingesetzt werden könnte. Insbesondere verspräche die Verwendung der minimalen allgemeinen LR(0)-Maschine besonders kleine (I)LALR(k)-Parser. Das folgende vernünftige Verfahren, Lookaheads über ihre zugehörigen lebensfähigen Präfizes aus den k-Itemmengen der kanonischen LR(k)-Maschine in die gegebene allgemeine LR(0)-Maschine $(\hat{M}, \mathrm{RC}_{\hat{M}})$ zu G zu projizieren, bietet sich an: Wir ordnen $\hat{M}$ verbesserte Stützinformation bei, indem wir $\mathrm{RC}_{\hat{M}}$ versuchsweise redefinieren, und zwar für den LALR(k)-Fall als

$$\begin{aligned}\mathrm{RC}_{\hat{M}}(\hat{q}) = \{A \mid y \,:\, & \text{für irgendwelche } \omega,\delta,\delta' \in V'^{*} \text{ mit } \mathrm{GOTO}_{\hat{M}}(\hat{q}_s,\delta') = \hat{q} \\ & \text{und } \mathrm{GOTO}_{\hat{M}}(\hat{q}_s,\delta\omega) = \mathrm{GOTO}_{\hat{M}}(\hat{q}_s,\delta'\omega) \\ & \text{ex. ein } [A \to \omega\cdot, y] \in \mathrm{VALID}_k(\delta\omega)\} \quad \cup \\ \{a \mid x \,:\, & a \in T' \text{ und für ein } \gamma \in V'^{*} \text{ mit } \mathrm{GOTO}_{\hat{M}}(\hat{q}_s,\gamma) = \hat{q} \\ & \text{ex. ein } [A \to \alpha\cdot a\beta, z] \in \mathrm{VALID}_k(\gamma), \\ & \text{so daß } x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)\}\end{aligned} \tag{23}$$

und für den ILALR(k)-Fall merklich einfacher als

$$\begin{aligned}\mathrm{RC}_{\hat{M}}(\hat{q}) = \{A \mid y \,:\, & \text{für ein } \delta \in V'^{*} \text{ mit } \mathrm{GOTO}_{\hat{M}}(\hat{q}_s,\delta) = \hat{q} \\ & \text{ex. ein } [A \to \cdot\,\omega, y] \in \mathrm{VALID}_k(\delta)\} \quad \cup \\ \{a \mid x \,:\, & a \in T' \text{ und für ein } \gamma \in V'^{*} \text{ mit } \mathrm{GOTO}_{\hat{M}}(\hat{q}_s,\gamma) = \hat{q} \\ & \text{ex. ein } [A \to \alpha\cdot a\beta, z] \in \mathrm{VALID}_k(\gamma), \\ & \text{so daß } x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)\}\end{aligned} \tag{24}$$

Sodann wird aus $(\hat{M}, \mathrm{RC}_{\hat{M}})$ wie bekannt ein „allgemeiner (I)LALR(k)“-Parser konstruiert. Setzen wir für $\hat{M}$ die kanonische LR(0)-Maschine ein, so erhalten wir den kanonischen (I)LALR(k)-Parser.[38]

[38] Setzen wir für $\hat{M}$ die kanonische LR(l)-Maschine ein ($l \leq k$), so ergibt sich für die LALR-Projektion der (kanonische) „LA(k)LR(l)-Parser“ gemäß Anderson (1972). Natürlich könnten wir analog auch „kanonische ILA(k)LR(l)-Parser“ definieren.

Leider jedoch, wie schon ähnlich für SLR(k), hat dieser Ansatz Schwächen: Je nach Struktur der gewählten allgemeinen LR(0)-Maschine resultiert ein Kellertransduktor, der nichtdeterministisch (bzw. deterministisch) sein kann, während der kanonische (I)LALR(k)-Parser zu G deterministisch (bzw. nichtdeterministisch) ist. Wir belegen das Fehlschlagen dieses Ansatzes mit einem

Beispiel: Wir betrachten $G = G_{abc\varepsilon}$. Wir wir gesehen haben, ist diese Grammatik für kein k eine ILALR(k)-Grammatik, für $k \geq 1$ jedoch eine LR(k)-Grammatik. Dies bedeutet, daß die beschriebene Projektion von LR(k)-Lookaheads in die Struktur der kanonischen LR(0)-Maschine und erst recht der minimalen allgemeinen LR(0)-Maschine (Abbildung 12 oben) für den „ILALR(k)"-Parser Nichtdeterminismus zur Folge hat. Projizieren wir jedoch in die Struktur der als allgemeine LR(0)-Maschine aufgefaßten kanonischen LR(k)-Maschine, so sind für $k \geq 1$ entstehender „ILALR(k)"-Parser und „LALR(k)"-Parser natürlich deterministisch, denn sie sind identisch mit dem kanonischen LR(k)-Parser. Es genügt sogar, gegenüber der minimalen allgemeinen LR(0)-Maschine einen einzigen Zustand aufzubrechen (Abbildung 12 unten), so daß der abgelesene „ILALR(k)"- und selbst der „LALR(k)"-Parser deterministisch wird. □

Die Problematik des Ansatzes liegt darin begründet, daß durch die Projektion von LR(k)-Lookaheads in eine andere allgemeine LR(0)-Maschine als die kanonische im anderem Maße LR(k)-Rechtskontexte von lebensfähigen Präfizes zusammengelegt werden können, als es durch die von der speziellen Struktur der kanonischen LR(0)-Maschine induzierten Äquivalenzklassen lebensfähiger Präfizes bewirkt wird. Ist die allgemeine LR(0)-Maschine „zu kompakt", können so Nichtdeterminismen auftreten, die bei Projektion auf die kanonische LR(0)-Maschine nicht entstehen (wobei der *redundanzreduzierte (I)LALR(k)-Parser,* der aus der Projektion auf die redundanzreduzierte LR(0)-Maschine resultiert, stets dieselben Nichtdeterminismen wie der kanonische (I)LALR(k)-Parser aufweist). Auch der umgekehrte Fall wird eintreten können. Diese Beobachtungen zeigen gleichzeitig einen adäquateren Ansatz auf:

Zu den lebensfähigen Präfizes von G sind genau die von einem kanonischen (I)LALR(k)-Parser verwendeten zusammengelegten LR(k)-Rechtskontexte von lebensfähigen Präfizes zu beachten. Wie erreichen dies, indem wir den *LALR(k)-Rechtskontext von* $\varphi \in \mathrm{VP}(G')$,

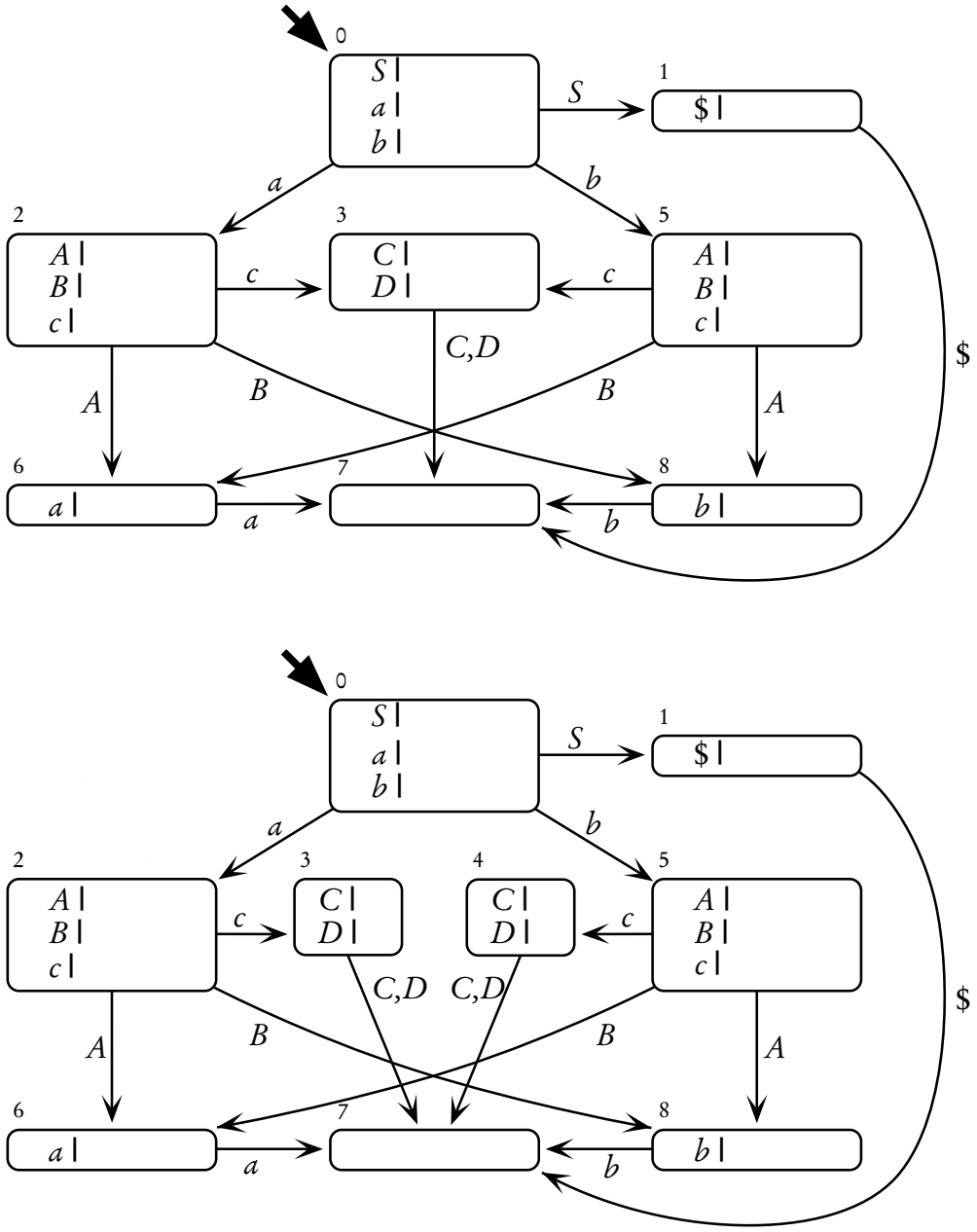


Abbildung 12: Die im Text beschriebene Projektion von ILALR(k)-Lookaheads auf die obere (minimale) allgemeine LR(0)-Maschine führt zu nichtdeterministischen Parseraktionen. Selbst die LALR(k)-Lookahead-Projektion auf die untere allgemeine LR(0)-Maschine führt hingegen zu einem deterministischen „LALR(k)“-Parser ($k \geq 1$).

$\mathrm{RC}_{G,\mathrm{LALR}(k)}(\varphi)$, mit $A \in V' \setminus T'$, $a \in T'$ und $\gamma \in V'^*$ definieren als

$$
\begin{aligned}
\mathrm{RC}_{G,\mathrm{LALR}(k)}(\varepsilon) &= \emptyset\,, \\
\mathrm{RC}_{G,\mathrm{LALR}(k)}(\gamma A) &= \{y : A \neq S' \text{ und es ex. } \gamma', \omega \in V'^*, \\
&\qquad \text{so daß } \mathrm{VALID}_0(\gamma\omega) = \mathrm{VALID}_0(\gamma'\omega) \\
&\qquad \text{und } [A \to \omega\cdot, y] \in \mathrm{VALID}_k(\gamma'\omega)\}\,, \\
\mathrm{RC}_{G,\mathrm{LALR}(k)}(\gamma a) &= \{x : \text{für ein } \gamma' \in V'^* \text{ mit } \mathrm{VALID}_0(\gamma) = \mathrm{VALID}_0(\gamma') \\
&\qquad \text{ex. ein } [B \to \alpha\cdot a\beta, z] \in \mathrm{VALID}_k(\gamma'), \\
&\qquad \text{so daß } x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)\}\,.
\end{aligned}
$$

Und den *ILALR(k)-Rechtskontext von* $\varphi \in \mathrm{VP}(G')$, $\mathrm{RC}_{G,\mathrm{ILALR}(k)}(\varphi)$, definieren wir als

$$
\begin{aligned}
\mathrm{RC}_{G,\mathrm{ILALR}(k)}(\varepsilon) &= \emptyset\,, \\
\mathrm{RC}_{G,\mathrm{ILALR}(k)}(\gamma A) &= \{y : A \neq S' \text{ und es ex. } \gamma', \omega \in V'^*, \\
&\qquad \text{so daß } \mathrm{VALID}_0(\gamma) = \mathrm{VALID}_0(\gamma') \\
&\qquad \text{und } [A \to \cdot\,\omega, y] \in \mathrm{VALID}_k(\gamma')\}\,, \\
\mathrm{RC}_{G,\mathrm{ILALR}(k)}(\gamma a) &= \{x : \text{für ein } \gamma' \in V'^* \text{ mit } \mathrm{VALID}_0(\gamma) = \mathrm{VALID}_0(\gamma') \\
&\qquad \text{ex. ein } [B \to \alpha\cdot a\beta, z] \in \mathrm{VALID}_k(\gamma'), \\
&\qquad \text{so daß } x \in \mathrm{FIRST}'_{\max\{k-1,0\}}(\beta z)\}\,. \\
&= \mathrm{RC}_{G,\mathrm{LALR}(k)}(\gamma a)
\end{aligned}
$$

Als eine *allgemeine (I)LALR(k)-Maschine zu* G bezeichnen wir dann ein System $(\hat{M}, \mathrm{RC}_{\hat{M}})$, in dem $\hat{M}$ ein vollständig spezifizierter, buchstabierender DEA ohne unerreichbare Zustände (mit Startzustand $\hat{q}_s$) ist, der die Sprache der lebensfähigen Präfizes von G' akzeptiert, und $\mathrm{RC}_{\hat{M}}$ ist eine Funktion, die jedem Zustand von $\hat{M}$ (I)LALR(k)-Rechtskontextinformation in Form von Paaren (X,u), notiert als $X \,|\, u$, zuordnet, so daß für alle $\gamma \in \mathrm{VP}(G')$ gilt:

$$
\mathrm{RC}_{\hat{M}}(\mathrm{GOTO}_{\hat{M}}(\hat{q}_s, \gamma)) = \{X \,|\, u : X \in V',\, \gamma X \in \mathrm{VP}(G'),\, u \in \mathrm{RC}_{G,(\mathrm{I})\mathrm{LALR}(k)}(\gamma X)\}\,.
$$

Es eignen sich stets die kanonische LR(0)-Maschine $\hat{M}_c$ zu G selbst und auch die redundanzreduzierte LR(0)-Maschine $\hat{M}_r$ zu G als der DEA $\hat{M}$ einer allgemeinen (I)LALR(k)-Maschine; die Stützinformation $\mathrm{RC}_{\hat{M}}$ kann mittels (23) bzw. (24) für $\hat{M} = \hat{M}_c$ aus der kanonischen oder für $\hat{M} = \hat{M}_r$ der redundanzreduzierten LR(k)-Maschine gewonnen werden und das Ergebnis sodann minimiert werden. Abbildung 13 zeigt die minimale ILALR(k)-Maschine ($k \geq 2$) zu $G = G_{abc\varepsilon}$.

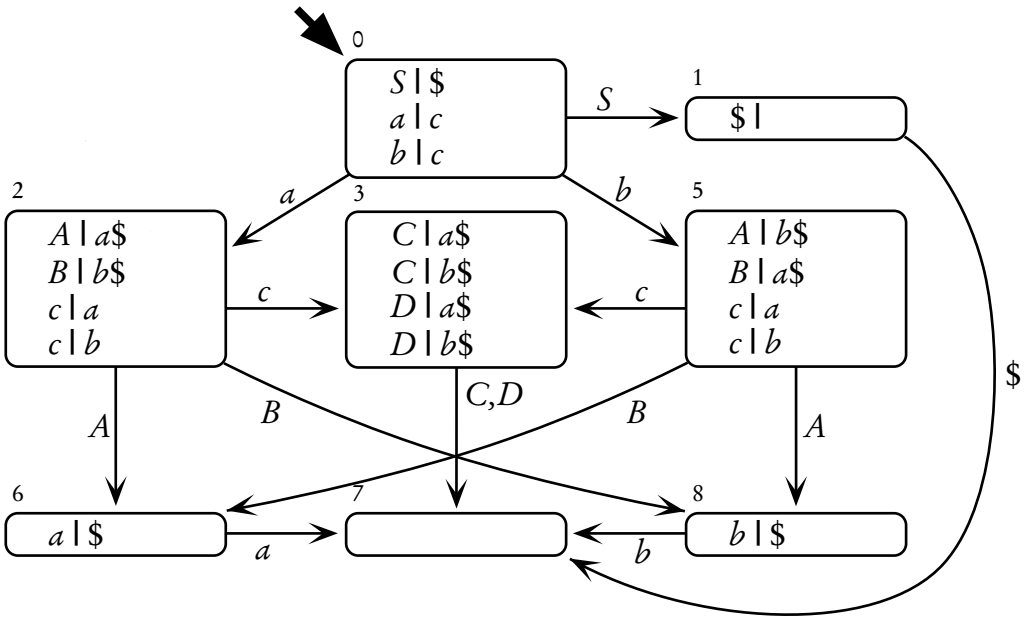


Abbildung 13: Die minimale allgemeine ILALR(k)-Maschine ($k \geq 2$) zu $G = G_{abc\varepsilon}$. (Für $k = 1$ ist in den Paaren $X \mid u$ jeweils u zu kürzen, und zwar zu $1 : u$, falls $X \in V' \setminus T'$, und zu ε, falls $X \in T'$ ist.)
Der abgelesene ILALR(k)-Parser ist für alle $k \geq 0$ nichtdeterministisch.

8.3 Effiziente Berechnung von ILALR(1)-Lookahead-Mengen

Im kanonischen LALR(1)-Parser zu G ist genau dann $a \in T'$ eine mögliche Lookahead-Vervollständigung einer Reduce-Aktion r des kanonischen LR(0)-Parsers zu G, wenn dieser LR(0)-Parser nach Anwendung von r und möglicherweise noch weiterer Reduce-Aktionen auf irgendeine erreichbare Parserkonfiguration dann entweder eine Shift-Aktion von $a \in T$ anwenden kann oder aber terminiert ($a = \$$). DeRemer und Pennello (1982) nutzen diese Beobachtung für die Entwicklung einer eleganten Methode, die Lookahead-Vervollständigungen von LR(0)- zu LALR(1)-Reduce-Aktionen (und somit LR(1)-Rechtskontexte) effizient *direkt aus der Übergangsstruktur der kanonischen LR(0)-Maschine* zu berechnen. Die kanonische LR(1)-Maschine zu G wird also nicht benötigt. Ihr Verfahren nimmt dabei auf die Inhalte der Itemmengen in der kanonischen LR(0)-Maschine keinen Bezug, weswegen es ebensogut auch auf beliebigen allgemeinen LR(0)-Maschinen operieren kann.

Es werden zu jeder LR(0)-Reduce-Aktion

$$r = \quad \ldots \hat{q} \mid \rightarrow \ldots ,$$

nach einer Regel $A \rightarrow \omega$ von G deren zugehörige LALR(1)-Lookahead-Vervollständigungen $\mathrm{LALR}(\hat{q}, A \rightarrow \omega) \subseteq T'$ mittels der Übergangsstruktur der LR(0)-Maschine $\hat{M}$ bestimmt als

$$a \in \mathrm{LALR}(\hat{q}, A \rightarrow \omega) \quad \Leftrightarrow \quad (\hat{q}, A \rightarrow \omega) \textbf{ has-LALR-lookahead } a ,$$

wobei **has-LALR-lookahead** gegeben ist durch den relationalen Ausdruck

$$\textbf{has-LALR-lookahead} = \textbf{lookback includes}^* \textbf{ reads}^* \textbf{ directly-reads} . \tag{25}$$

Die konstituierenden Hilfsrelationen hierin sind definiert als

$(\mathrm{GOTO}_{\hat{M}}(\hat{q}, \omega), A \rightarrow \omega)$ **lookback** $(\hat{q}, A)$, wenn $\mathrm{GOTO}_{\hat{M}}(\hat{q}, A)$ existiert und $A \rightarrow \omega \in P$ ist;

$(\mathrm{GOTO}_{\hat{M}}(\hat{q}, \alpha), A)$ **includes** $(\hat{q}, B)$, wenn $\mathrm{GOTO}_{\hat{M}}(\hat{q}, B)$ existiert und $B \rightarrow \alpha A \beta \in P$ und $\beta \Rightarrow^* \varepsilon$ in G gilt;

$(\hat{q}, A)$ **reads** $(\mathrm{GOTO}_{\hat{M}}(\hat{q}, A), B)$, wenn $A \in V \setminus T$ ist, $B \Rightarrow^+ \varepsilon$ in G gilt und zudem $\mathrm{GOTO}_{\hat{M}}(\hat{q}, AB)$ existiert;

$(\hat{q}, A)$ **directly-reads** a , wenn $a \in T'$, $A \in V \setminus T$ ist und $\mathrm{GOTO}_{\hat{M}}(\hat{q}, Aa)$ existiert.

Der Teilausdruck **includes*** **reads*** in (25) berücksichtigt, daß nach einer Reduktion nach $A \to \omega$ mit oberstem Kellerzustand $\hat{q}$ weitere Reduktionen möglich sind, bevor die nächste Shift-Aktion von a erfolgt. Für formale Beweise in der hier gewählten Notation siehe Abschnitt 7.3 in (Sippu und Soisalon-Soininen 1990), von woher auch die Anregung zu Abbildung 14 stammt, welche eine graphische Veranschaulichung für die Beziehung $a \in \mathrm{LALR}(\hat{q}, A \to \omega)$ bietet.

Der wesentliche Unterschied zwischen ILALR- und LALR-Verfahren wurde im Zusammenhang mit der Vorstellung des ILALR(k)-Verfahrens in Abschnitt 8.2 genau diskutiert. Und es ist vollkommen klar, daß sich im Zusammenhang mit der Berechnung von ILALR(1)-Lookahead-Vervollständigungen aus der kanonischen LR(0)-Maschine im Stil von DeRemer und Pennello dieser Unterschied gerade im Wegfall des Teilausdrucks **lookback** in (25) äußert, da ja für eine kanonische LR(0)-Reduce-Aktion

$$\hat{q} \ldots \mathrm{GOTO}_{\hat{M}_c}(\hat{q}, \omega) \,\mathbf{|} \to \hat{q}\, A\, \mathrm{GOTO}_{\hat{M}_c}(\hat{q}, A) \,\mathbf{|}$$

nach $A \to \omega$ die ILALR(1)-Lookahead-Vervollständigungen sich aus vermöge *core* projizierten k-Items für den Zustand $\hat{q}$ statt (wie im Falle LALR(1)) für den Zustand $\mathrm{GOTO}_{\hat{M}_c}(\hat{q}, \omega)$ bestimmen. Genauer werden für jede solche kanonische LR(0)-Reduce-Aktion deren ILALR(1)-Lookahead-Vervollständigungen $\mathrm{ILALR}(\hat{q}, A) \subseteq T'$ bestimmt als

$$a \in \mathrm{ILALR}(\hat{q}, A \to \omega) \quad \Leftrightarrow \quad (\hat{q}, A \to \omega) \textbf{ has-ILALR-lookahead } a \,,$$[39]

wobei **has-ILALR-lookahead** gegeben ist durch den relationalen Ausdruck

$$\textbf{has-ILALR-lookahead} = \textbf{includes}^* \textbf{ reads}^* \textbf{ directly-reads} \,.$$

Siehe nochmals Abbildung 14 für eine Veranschaulichung.

Die Berechnung von ILALR(1)-Lookahead-Vervollständigungen kann damit, wie von DeRemer und Pennello für LALR(1) beschrieben, über zwei Anwendungen einer um Mengenoperationen angereicherten Variante des Algorithmus von Tarjan (1972) zur Bestimmung der starken Zusammenhangskomponenten eines gerichteten Graphen vorgenommen werden; ein letzter Mengenvereinigungsschritt (**lookback**) entfällt jedoch. Für Einzelheiten wird auf (DeRemer und Pennello 1982) und (Sippu und Soisalon-Soininen 1990) verwiesen.

[39] Die von uns als $\mathrm{ILALR}(\hat{q}, A)$ bezeichnete Menge tritt übrigens in (DeRemer und Pennello 1982) als Hilfsmenge $\mathrm{Follow}(\hat{q}, A)$ in Erscheinung.

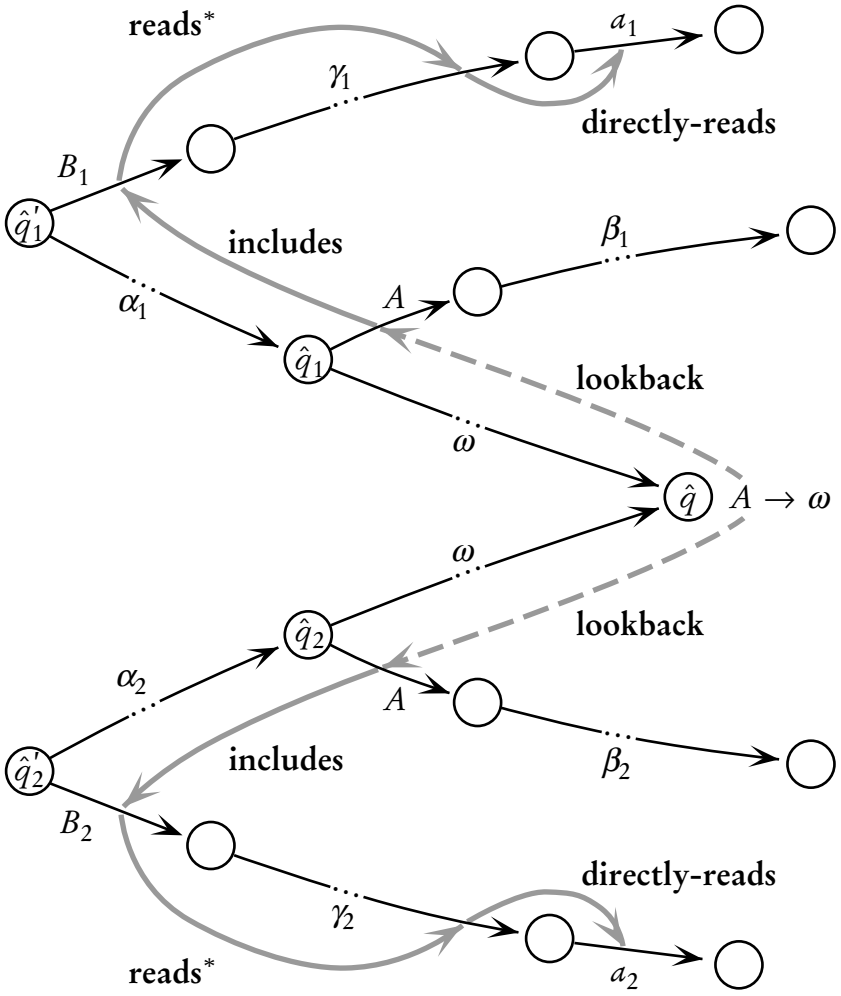


Abbildung 14: Ausschnitt einer allgemeinen LR(0)-Maschine. Es sind $A \to \omega$, $B_1 \to \alpha_1 A \beta_1$ und $B_2 \to \alpha_2 A \beta_2$ Regeln von G, und ferner gilt $\beta_1 \beta_2 \gamma_1 \gamma_2 \Rightarrow^* \varepsilon$. Die Abbildung visualisiert, warum $\{a_1, a_2\} \subseteq \mathrm{LALR}(\hat{q}, A \to \omega)$ ist: Es gilt

$(\hat{q}, A \to \omega)$ **lookback** $(\hat{q}_1, A)$ **includes** $(\hat{q}'_1, B_1)$ **reads*** **directly-reads** a_1

und

$(\hat{q}, A \to \omega)$ **lookback** $(\hat{q}_2, A)$ **includes** $(\hat{q}'_2, B_2)$ **reads*** **directly-reads** a_2 .

Dagegen werden für einen ILALR(1)-Parser die beiden Lookahead-Vervollständigungen $\mathrm{ILALR}(\hat{q}_1, A) \ni a_1$ und $\mathrm{ILALR}(\hat{q}_2, A) \ni a_2$ *nicht* an $\hat{q}$ vereinigt.

9 Zur Konstruktion von LR-Parsern für erweiterte kontextfreie Grammatiken

In Programmiersprachenreports haben sich gewisse Erweiterungsformen kontextfreier Grammatiken, in denen rechte Regelseiten reguläre Sprachen beschreiben, als Syntaxspezifikationsformalismen durchgesetzt. So meint beispielsweise Watt (1991):

> *„Niklaus Wirth introduced* EBNF*, or Extended Backus–Naur Form, and* syntax diagrams *to specify the syntax of Pascal (Jensen and Wirth 1974). These notations are generally more convenient than ordinary BNF, but not more powerful. BNF, EBNF, and syntax diagrams are now almost universally used to specify the syntax of programming languages, and understood by most programmers.“*

Gegenüber reinen kfGen – z. B. notiert in Backus–Naur Form (BNF) – bieten erweiterte Notationen wie EBNF oder Syntaxdiagramme den Vorteil, daß sich mit ihnen oft etliche Hilfsnichtterminale einsparen lassen, Spezifikationen allgemein kompakter werden und insbesondere iterative und „echt rekursive“ Strukturmuster einer zu beschreibenden Programmiersprache demjenigen, der sie erlernen will, deutlich leichter als solche erkennbar gemacht werden können. Für den Übersetzerbauer wiederum ist es hilfreich, sich – was die zugrunde gelegte kontextfreie Skelettgrammatik angeht – bei der Erstellung einer Parserkomponente für die in Rede stehende Sprache möglichst wenig von der Grundlage der Reportgrammatik entfernen zu müssen. Und falls die (noch um Attributierungen reicheren) Eingabespezifikations-Formalismen von Compiler-Generatoren die erweiterten Konstrukte der kontextfreien Sprachebene geeignet berücksichtigen, so profitieren auch sie von deren Nutzen hinsichtlich Prägnanz und Lesbarkeit. Verwiesen werden kann hier auf den Compiler-Generator *epsilon* auf der Grundlage Erweiterter Affix-Grammatiken, der in (Demuth, Weber, Kannapinn und Kröplin 1997) beschrieben wird.

Ganz besonders vielversprechend ist nun die Idee, die mächtige LR-Methode für die angedeuteten Erweiterungsformen kontextfreier Grammatiken nutzbar zu machen, womit wir uns in diesem Abschnitt beschäftigen wollen. Und natürlich hat es durchaus bereits

Versuche gegeben, dies zu leisten. Viele der in der Literatur geschilderten Verfahren erweisen sich jedoch aus gewichtigen Gründen als unbrauchbar für den Einsatz im generativen Übersetzerbau, fast immer fehlt ein überzeugender theoretischer Unterbau, und keines der Verfahren präsentiert letztlich eine formal ästhetische Lösung in Form von deterministischen Kellertransduktoren. Eine befriedigende solche Lösung zu entwickeln und sie mit bisherigen Ansätzen zu vergleichen ist Anliegen dieses Abschnitts, und wir werden sehen, daß die zuvor erzielten Ergebnisse zur theoretisch fundierten Konstruktion allgemeiner (insbesondere redundanzreduzierter) LR-Parser sich als nützlich erweisen werden.

Drei eingeführte Bezeichnungen bereichern unsere bis hierher noch vage gehaltene Ausdrucksweise: Die hier betrachteten Erweiterungsformen kontextfreier Grammatiken weisen auf ihren rechten Regelseiten nicht einfache Symbolfolgen auf, wie in den kfGen der Fall, sondern Beschreibungshilfsmittel, die reguläre Sprachen über den Grammatiksymbolen definieren, wobei wir o. B. d. A. zur Vereinfachung jedes Nichtterminal nur einmal auf linker Seite zulassen wollen. Übliche solche Hilfmittel sind hier reguläre Ausdrücke und endliche Automaten. Sind die rechten Regelseiten reguläre Ausdrücke, so wollen wir von einer *erweiterten kontextfreien Grammatik (ekfG)* sprechen. Liegen endliche Automaten vor, so reden wir von einer *RRPG,* wobei das Akronym die englische Bezeichnung *„regular right part grammar“* abkürzt, und ein Automat auf rechter Regelseite wird meist kurz als *RRPA („regular right part automaton“)* bezeichnet. Ein Oberbegriff für ekfGen und RRPGen existiert in der Literatur nicht. Nötigenfalls – wie z. B. in der Abschnittsüberschrift – reden wir von „ekfGen“ mit beiden Bedeutungen, da dies deutsch expandierbar ist.

Wie gelangen wir nun für solche Erweiterungsformen von kfGen zu nachweislich korrekten Parsern? Was sollen diese Parser sinnvollerweise überhaupt präzise leisten? Es ist vielleicht die Anpassung des LL(1)-Verfahrens an handgeschriebene Parser für ekfGen noch mit Intuition und Erfahrung durch „scharfes Hinsehen“ zu leisten. Im Falle des theoretisch komplexeren LR-Verfahrens dagegen sollten wir uns nicht mehr auf das Augenmaß verlassen. Schließlich würden wesentliche Eckpfeiler der LR-Theorie – z. B. Begriffe wie „lebensfähiges Präfix“ und „LR(k)-gültiges Item“ – durch den Übergang zu einem neuen Grammatiktypus redefiniert werden müssen, womit praktisch das gesamte theoretische Fundament der klassischen LR-Parser zu überarbeiten wäre – ein längerer Weg, und noch dazu ein komplizierter,[40] da die zugrunde liegende Grammatikform nun eine schwierigere ist. Wir werden einen einfacheren und insofern befriedigenderen Weg verfolgen, der insbesondere zu einem Syntaxanalyseverfahren mit den folgenden wesentlichen Eigenschaften

[40] Siehe (Madsen und Kristensen 1976).

führen wird:

Erstens wird das Verfahren *sauber theoretisch fundiert* sein. Zweitens werden wir – um den Bezug zur gefestigten Automatentheorie und Theorie der Formalen Sprachen zu wahren, aber auch, um effiziente Implementierbarkeit zu sichern – die konstruierten Syntaxanalysatoren als (implementierte) *deterministische Kellerautomaten* bzw. -transduktoren beschreiben. Und drittens ist klar, daß der avisierte Anwendungskontext der hier interessierenden Verfahren Übersetzer für Programmiersprachen sind. In diesen ist die Syntaxanalyse nur eine von mehreren Arbeitsphasen. Insbesondere fußen die ihr konzeptuell folgenden Phasen der Überprüfung von Kontextbedingungen und der Codeerzeugung auf der von ihr bestimmten syntaktischen Struktur des Quellprogramms. Um uns hier auch weiterhin auf das bewährte Kalkül der Attributgrammatiken (Knuth 1968) und damit verfahrensmäßig auf den bekannten Fächer effizienter Treewalk-Evaluatoren[41] stützen zu können, ist für uns unabdingbar, daß das *Ergebnis der Syntaxanalyse ein klassischer Ableitungsbaum* (z. B. als Rechtsparse) zu einer herkömmlichen (endlichen!) kfG sein muß.

Mit dem Gesagten ist bereits vorgezeichnet: Der hier beschriebene Ansatz, LR-Parser für ekfGen und RRPGen zu entwickeln, wird auf der naheliegenden und schlichten Idee beruhen, Exemplare dieser komplexeren Grammatiktypen in adäquater Weise mittels reiner kfGen zu modellieren. Dies ist nicht nur ökonomisch im Hinblick auf Kognition und zu leistende Theoriearbeit, denn wir können uns eines akzeptierten Begriffsapparats und umfangreichen und gereiften theoretischen Fundaments bedienen. Der Ansatz empfiehlt sich auch dadurch, daß er direkt auf dem theoretisch tragfähigen formalen Rahmenwerk von Heilbrunner (1979) zu ekfGen aufsetzt – auch wenn wir hier, was die präzise Definition angeht, wann eine ekfG eine „ELR(k)-Grammatik“ genannt werden soll, eine andere Konsequenz ziehen werden als Heilbrunner selbst.

Wir stellen nun Formalisierung und wesentliche Ergebnisse der erwähnten Arbeit von Heilbrunner (1979) vor, die sich ausschließlich mit der *Definition* von ELR(k)- und ELL(k)-Grammatiken im Rahmen von ekfGen befaßt, und beschränken uns dabei auf diejenigen Aussagen, die sich auf das LR-Verfahren beziehen. Dabei werden wir eigene Überlegungen zur Parserkonstruktion einschieben. Die Behandlung von RRPGen wird sich später mitergeben.

[41] Siehe (Deransart, Jourdan und Lorho 1988) für einen Überblick.

9.1 Transformationen von ekfGen in kfGen – Heilbrunner (1979)

Heilbrunner (1979) abstrahiert gleich zu Beginn seines Journalartikels zur Definition von ELR(k)- und ELL(k)-Grammatiken von der Struktur der rechten Regelseiten in den von ihm betrachteten ekfGen: Mit einer ekfG G_e *assoziiert* er zunächst eine kfG $G = (V, T, P, S)$, die die gleichen Symbolmengen V bzw. T und dasselbe Startsymbol S verwendet, jedoch insofern besonders ist, als die Regelmenge P von G auch unendlich sein darf – allerdings mit der Forderung, daß für alle $A \in V \setminus T$ jede Menge $R(A) = \{\omega : A \to \omega \in P\}$ gerade die von der rechten Regelseite zu A in G_e beschriebene reguläre Sprache ist. Wir wollen voraussetzen, daß G reduziert ist.

Bevor wir uns mit Heilbrunners Ergebnissen beschäftigen wollen, wie ekfGen LR-geeignet mittels reiner (endlicher) kfGen „repräsentiert“ werden können, verfolgen wir zunächst kurz – in Ergänzung zu Heilbrunner (1979) – ein Stück weit die Idee, beispielhaft aus einer assoziierten kfG mit unendlicher Regelmenge direkt einen LR(k)-Parser zu erzeugen. Dabei lohnt es sich, die kanonische und die redundanzreduzierte Konstruktion miteinander zu vergleichen.

Wir betrachten die Beispiel-ekfG aus Abbildung 17(a) auf Seite 114 mit der assoziierten kfG aus Abbildung 17(b) und beschränken uns auf den Fall $k = 0$.[42] Den Zwischenschritt der nichtdeterministischen LR(k)-Maschinen übergehen wir und betrachten sogleich die deterministischen Versionen.

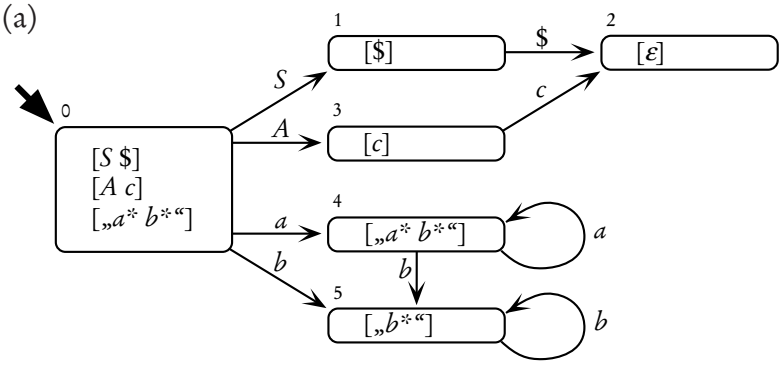
Abbildung 15 zeigt zunächst die redundanzreduzierte LR(0)-Maschine mit den daraus abgelesenen Shift- und Reduce-Aktionen des zugehörigen LR(0)-Parsers. Folgende Beobachtungen halten wir hierzu fest:

1. Die gezeigte redundanzreduzierte LR(0)-Maschine weist eine endliche Anzahl von Zuständen auf, wenn auch die Anzahl LR(0)-gültiger Items zu jedem lebensfähigen Präfix unendlich sein kann (Itemmengen 0, 4, 5). Daß dies auch allgemein gilt – auch für den Fall $k > 0$ –, ist eingedenk der diversen Abgeschlossenheitseigenschaften regulärer Sprachen leicht einzusehen.

2. Der abgelesene redundanzreduzierte LR(0)-Parser weist eine endliche Anzahl von Shift-Aktionen auf. Da mit der Überlegung aus 1. auch allgemein die redundanzreduzierte LR(k)-Maschine zu einer betrachteten assoziierten kfG eine endliche Anzahl von Zuständen aufweist, ist auch die Anzahl abgelesener Shift-Aktionen stets endlich.

[42] Wir verwenden übliche Syntax für reguläre Ausdrücke. Zumeist orientieren wir uns an (Brüggemann-Klein 1993): im Beispielausdruck $ab^*(c+d)^*$ kommen alle verwendeten Operatoren vor.

(a)



(b)

$$
\begin{aligned}
s_1 &= 0\,|\,a \to 0\,a\,4\,|\,, & \tau_c(s_1) &= \varepsilon\,.\\
s_2 &= 0\,|\,b \to 0\,b\,5\,|\,, & \tau_c(s_2) &= \varepsilon\,.\\
s_3 &= 3\,|\,c \to 3\,c\,2\,|\,, & \tau_c(s_3) &= \varepsilon\,.\\
s_4 &= 4\,|\,a \to 4\,a\,4\,|\,, & \tau_c(s_4) &= \varepsilon\,.\\
s_5 &= 4\,|\,b \to 4\,b\,5\,|\,, & \tau_c(s_5) &= \varepsilon\,.\\
s_6 &= 5\,|\,b \to 5\,b\,5\,|\,, & \tau_c(s_6) &= \varepsilon\,.\\
r_1 &= 0\,A\,3\,c\,2\,|\,\to 0\,S\,1\,|\,, & \tau_c(r_1) &= S \to A\,c\,.\\
r_2 &= 0\,|\,\to 0\,A\,3\,|\,, & \tau_c(r_2) &= A \to \varepsilon\,.\\
r_3 &= 0\,a\,4\,|\,\to 0\,A\,3\,|\,, & \tau_c(r_3) &= A \to a\,.\\
r_4 &= 0\,b\,5\,|\,\to 0\,A\,3\,|\,, & \tau_c(r_4) &= A \to b\,.\\
r_5 &= 0\,a\,4\,a\,4\,|\,\to 0\,A\,3\,|\,, & \tau_c(r_5) &= A \to a\,a\,.\\
r_6 &= 0\,a\,4\,b\,5\,|\,\to 0\,A\,3\,|\,, & \tau_c(r_6) &= A \to a\,b\,.\\
r_7 &= 0\,b\,5\,b\,5\,|\,\to 0\,A\,3\,|\,, & \tau_c(r_7) &= A \to b\,b\,.\\
r_8 &= 0\,a\,4\,a\,4\,a\,4\,|\,\to 0\,A\,3\,|\,, & \tau_c(r_8) &= A \to a\,a\,a\,.\\
r_9 &= 0\,a\,4\,a\,4\,b\,5\,|\,\to 0\,A\,3\,|\,, & \tau_c(r_9) &= A \to a\,a\,b\,.\\
r_{10} &= 0\,a\,4\,b\,5\,b\,5\,|\,\to 0\,A\,3\,|\,, & \tau_c(r_{10}) &= A \to a\,b\,b\,.\\
r_{11} &= 0\,b\,5\,b\,5\,b\,5\,|\,\to 0\,A\,3\,|\,, & \tau_c(r_{11}) &= A \to b\,b\,b\,.\\
&\ldots & &\ldots
\end{aligned}
$$

Abbildung 15: (a) Die deterministische redundanzreduzierte LR(0)-Maschine zur unendlichen kfG aus Abbildung 17(b) auf Seite 114. [„ … “] stellvertritt je ein Item für jede Expansion des regulären Ausdrucks in den Anführungszeichen. (b) Die endlich vielen Shift- und unendlich vielen Reduce-Aktionen des daraus abgelesenen LR(0)-Parsers.

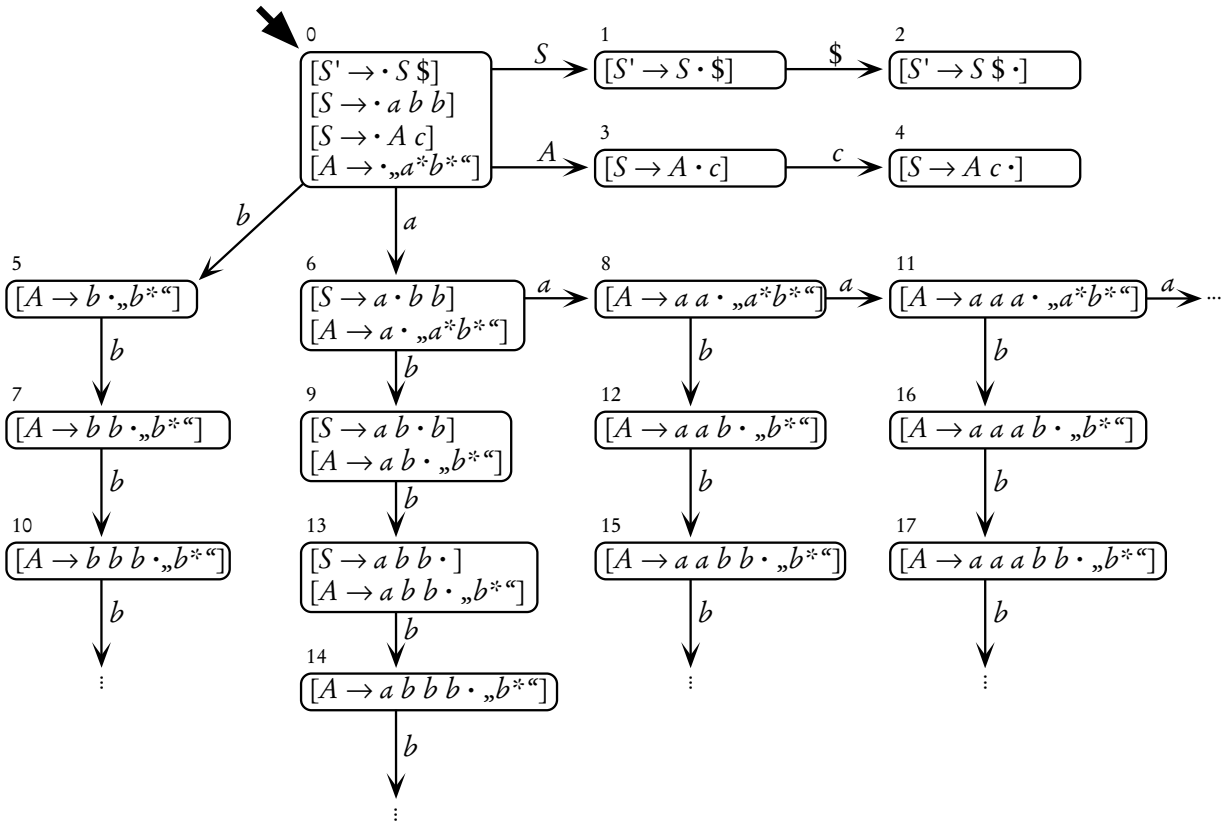


Abbildung 16: Die deterministische kanonische LR(0)-Maschine zur unendlichen kfG aus Abbildung 17(b) auf Seite 114. …[„ … “] stellvertritt je ein Item für jede Expansion des regulären Ausdrucks in den Anführungszeichen. Der abgelesene LR(0)-Parser besitzt unendlich viele Shift- und Reduce-Aktionen.

3. Obwohl die gezeigte LR(0)-Maschine eine endliche Anzahl von Zuständen aufweist, weist der abgelesene LR(0)-Parser unendlich viele Reduce-Aktionen auf. Dies ist natürlich aufgrund der unendlichen Regelmenge der Ausgangsgrammatik nicht verwunderlich.

Abbildung 16 zeigt zur selben Beispielgrammatik die kanonische LR(0)-Maschine. Hierzu beobachten wir:

1. Die gezeigte kanonische LR(0)-Maschine weist eine unendliche Anzahl von Zuständen auf: Die gegenüber den redundanzreduzierten Itemmengen hinzukommende Aufgabe der „Geschichtsverwaltung“ in den kanonischen Itemmengen be-

wirkt, daß einige der redundanzreduzierten Itemmengen in der kanonischen Konstruktion mit unendlich vielen „individuellen Geschichten“ vorkommen.

2. Der abgelesene kanonische LR(0)-Parser besitzt daher auch unendlich viele Shift-Aktionen.

3. Auch die Anzahl der Reduce-Aktionen des abgelesenen Parsers ist natürlich unendlich.

Quintessenz dieses kurzen Exkurses ist, daß zu ekfGen der Versuch, unter Abstraktion von der Struktur der vorkommenden rechten Regelseiten LR-Parser quasi direkt aus der jeweiligen assoziierten kfG zu konstruieren, zu Ergebnissen führt, die nur von theoretischem Interesse sind: Die entstehenden Kellertransduktoren mit ihren i. a. unendlichen Aktionenmengen eignen sich nicht als unmittelbare Implementierungsvorlage; und zwar die unendlichen kanonischen LR(k)-Parser, wenn man so will, noch weniger als die unendlichen redundanzreduzierten. Deutlich geworden ist aber auch, wo anzusetzen ist, wenn wir zu endlichen LR-Parsern gelangen wollen: Die Ausgangs-kfG, für die wir einen LR-Parser konstruieren, muß bereits eine endliche Regelmenge aufweisen. Daher ist nun von Interesse, wie einer i. a. unendlichen assoziierten kfG „LR-kompatibel“ eine „adäquate“ endliche kfG beiseite gestellt werden kann, so daß endliche LR-Parser resultieren.

Es wird dem Leser sicherlich klar sein, daß jede ekfG mittels eines vernünftigen, *ad hoc* gewählten Transformationsschemas in eine „äquivalente“ reine kfG überführt werden kann, indem nichtatomare Teilausdrücke auf der rechten Regelseite mittels eingeführter Hilfsnichtterminale kontextfrei nachgebildet werden. Die entstehende reine kfG spiegelt sogar die Struktur der ekfG wider. Daher ist auch eine zusätzliche Attributierung der ekfG sinnvoll möglich und kann mittransformiert werden mit dem Ergebnis einer attributierten kfG, so daß die attributierte ekfG quasi als deren notationelle Abkürzung angesehen werden könnte (Demuth et al. 1997). Allerdings sind unterschiedliche Transformationen von ekfGen in „äquivalente“ kfGen möglich, und nicht alle weisen wünschenswerte Eigenschaften auf.

So führen etwa die auf der Hand liegenden Transformationen einer Regel $A \to a(b)^* c$ in $A \to aA'c$ und $A' \to \varepsilon \mid A'b$ oder $A' \to \varepsilon \mid bA'$ zu kfGen, die entgegen der Natur von LR-Verfahren einen entsprechenden LR-Parser zu beispielsweise einem Handle $abbbc$ nicht erst *nach* dessen Kellern einschlägige Reduktionsentscheidungen treffen läßt, sondern bereits vorzeitig nach Kellern von a bzw. $abbb$. Solcherlei Transformationsschemata, verwendet etwa in (Demuth et al. 1997) und (Celentano 1981), sind somit ungeeignet im

Hinblick auf reine LR-Verfahren – wenn auch vielleicht in der Praxis ein gezieltes und benutzergesteuertes Abweichen von LR-geeigneten Transformationsmustern gelegentlich sinnvoll sein mag.

Folgen wir nun wieder Heilbrunner (1979) und nehmen wir an, wir verfügten für jedes $A \in V \setminus T$ über irgendeine kontextfreie (Sub-)Grammatik $H_A = (V_A, V, P_A, S_A)$ mit eigenen Nichtterminalen $V_A \setminus V$, die $L(H_A) = R(A)$ erfüllt, dann können wir die (endliche) *Repräsentation von G mit Subgrammatiken* H_A definieren durch $G^H = (V^H, T, P^H, S)$ mit

$$V^H = \bigcup_{A \in V \setminus T} V_A \qquad \text{und} \qquad P^H = \{A \to S_A : A \in V \setminus T\} \cup \bigcup_{A \in V \setminus T} P_A \,,$$

und offensichtlich ist $L(G) = L(G^H)$. Eine solche Repräsentationsgrammatik G^H ist mögliches Ergebnis einer Transformation von G_e, und wir sagen, daß die *Transformation vom Typ X* ist, wenn jede Subgrammatik H_A vom Typ X ist. Mit Blick auf die Klarheit dieser Subgrammatik-Struktur wollen wir von nun an erwarten, daß Transformationen grundsätzlich Ergebnis-kfGen mit dieser Struktur liefern; ggf. muß für die Trennung der Ebenen durch Regeln P^H gesorgt werden.

Das Kernergebnis von (Heilbrunner 1979) ist schließlich die Erkenntnis:

> *Ist G^H eine beliebige Repräsentation von G, deren Subgrammatiken H_A rechtslinear und eindeutig sind, dann ist G LR(k)-Grammatik gdw. G^H LR(k)-Grammatik ist.*[43]

Wesentliche Eigenschaft einer jeden auftretenden Subgrammatik H_A ist im Zusammenhang mit dem LR-Verfahren also, daß H_A rechtslinear ist und die Sprache $R(A)$ eindeutig beschreibt. Und wann immer mit *einer* solchen Familie von Subgrammatiken dann G^H zu einer endlichen LR(k)-Grammatik wird, wird es das auch mit jeder anderen solchen Familie. In der konkreten Wahl geeigneter Subgrammatiken besteht also eine gewisse Beliebigkeit.

---

[43] Wir reformulieren hiermit Theorem 2 aus (Heilbrunner 1979), welches irreführenderweise lautet: „*A grammar is an LR(k) grammar iff every representation using unambiguous right-linear grammars is an LR(k) grammar.*“ Statt dieser Aussage *beweist* Heilbrunner genau unsere Reformulierung.

Ferner wird dem aufmerksamen Leser vielleicht nicht entgangen sein, daß die in der vorliegenden Arbeit verwendete Definition einer kfG G als LR(k), wenn sich ihr Shift-reduce-Parser „deterministisch machen“ läßt, nicht einfach von endlichen auf unendliche kfGen übertragen läßt. Für unendliche kfGen verweisen wir daher auf die klassische LR(k)-Definition, wie sie z. B. auch bei Heilbrunner (1979) zu finden ist.

Diese Beliebigkeit ist schließlich Anlaß für Heilbrunner, die Ausgangs-ekfG G_e eine „ELR(k)-Grammatik“ dann zu nennen, wenn die mit G_e assoziierte (meist unendliche) kfG G eine LR(k)-Grammatik ist, d. h. irgendeine endliche Repräsentation von G mit rechtslinearen Subgrammatiken existiert, die die LR(k)-Eigenschaft aufweist. Diesem Definitionsvorschlag von Heilbrunner schließen wir uns nicht mehr an, was wir wie folgt ausführlich begründen:

Bekanntermaßen ist es stets möglich, zu einem regulären Ausdruck einen NEA zu konstruieren, der dessen beschriebene reguläre Sprache akzeptiert, indem er sozusagen die gültigen Wege durch den regulären Ausdruck verkörpert. Die „Eindeutigkeit“ des regulären Ausdrucks kann sodann mit der dieses NEA identifiziert werden (was wir später noch präzisieren). Ferner kann bekanntlich zu einem beliebigen NEA mühelos eine rechtslineare Grammatik angegeben werden, die dieselbe Sprache wie der NEA erzeugt und genau dann eindeutig ist, wenn dies auch der NEA ist: Die Zustände des NEA werden zu den Nichtterminalen dieser Grammatik, ihr Startsymbol wird der Anfangszustand, jede NEA-Aktion $q\alpha \to q'$ wird zu einer Regel $q \to \alpha q'$, und für jeden Endzustand p wird eine Regel $p \to \varepsilon$ aufgenommen.

Abbildung 17 zeigt für diese Umformung von G_e in eine endliche Repräsentations-kfG G^H, deren Subgrammatiken die Struktur von G_e reflektieren, ein Beispiel, an dem wir uns auch den besonderen Aufbau von Rechtsableitungen zu G^H vor Augen führen können. Diese Rechtsableitungen bestehen aus aneinandergesetzten Ableitungssegmenten, von denen jedes einzelne, je begrenzt durch eine „Einstiegs-“ und eine „Ausstiegsregel“, die Erzeugung gerade eines Handles von G_e (bzw. G) durch eine jeweilige Subgrammatik beinhaltet – oder anschaulich den Durchlauf durch den jeweiligen NEA. Die von einem Rechtsparser produzierten *umgekehrten* Rechtsableitungen reihen also *Rückwärts*-Durchläufe durch die NEAs aneinander.

Schließlich sollte, ohne daß wir dies schon hier im Detail aufzeigen können, klar sein, daß aus einem solchen Rückwärts-Durchlauf eines NEA dann rekonstruiert[44] werden kann, wie das in Rede stehende Handle von G_e durch einen passenden regulären Ausdruck auf einer rechten Regelseite von G_e *induktiv* erzeugt wird, da eben der zugehörige NEA die Struktur des Ausdrucks wiedergibt.

Natürlich wollen wir nun aber ekfGen im Kontext des Übersetzerbaus nutzen, weswegen unverzichtbar ist, daß wir mit den Regeln einer ekfG in praktisch sinnvoller Weise semantische Aktionen verbinden können. Und die einzig vernünftige Methode ist, dies

[44] Dies gilt nicht ganz uneingeschränkt, worauf wir noch zurückkommen.

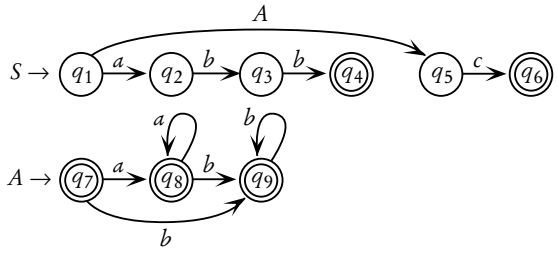
(a) Eine Beispiel-ekfG G_e:

$$S \to abb + Ac$$
$$A \to a^* b^*$$

(b) Die (unendliche) assoziierte kfG G zu G_e:

$$S \to abb \mid$$
$$\quad Ac$$
$$A \to \varepsilon \mid$$
$$\quad a \mid b \mid$$
$$\quad aa \mid ab \mid bb \mid$$
$$\quad aaa \mid aab \mid abb \mid bbb \mid$$
$$\quad \ldots$$

(c) G_e als RRPG mit Glushkov-NEAs:



(d) Die strukturbewahrende (endliche) Repräsentations-kfG G^H mit rechtslinearen Subgrammatiken:

$$\begin{aligned}
S \to q_1\,, \quad & q_1 \to aq_2\,, \\
& q_1 \to Aq_5\,, \\
& q_2 \to bq_3\,, \\
& q_3 \to bq_4\,, \quad q_4 \to \varepsilon\,, \\
& q_5 \to cq_6\,, \quad q_6 \to \varepsilon\,, \\
A \to q_7\,, \quad & q_7 \to aq_8\,, \quad q_7 \to \varepsilon\,, \\
& q_7 \to bq_9\,, \\
& q_8 \to aq_8\,, \quad q_8 \to \varepsilon\,, \\
& q_8 \to bq_9\,, \\
& q_9 \to bq_9\,, \quad q_9 \to \varepsilon\,.
\end{aligned}$$

(e) Eine beispielhafte Rechtsableitung in G^H mit 2 Segmenten:

$$\underbrace{S \underset{\text{rm}}{\Longrightarrow} q_1 \underset{\text{rm}}{\Longrightarrow} Aq_5 \underset{\text{rm}}{\Longrightarrow} Acq_6 \underset{\text{rm}}{\Longrightarrow} Ac}_{\text{1. Segment}} \underbrace{\underset{\text{rm}}{\Longrightarrow} q_7 c \underset{\text{rm}}{\Longrightarrow} aq_8 c \underset{\text{rm}}{\Longrightarrow} aaq_8 c \underset{\text{rm}}{\Longrightarrow} aabq_9 c \underset{\text{rm}}{\Longrightarrow} aabc}_{\text{2. Segment}}$$

Abbildung 17: Metamorphosen einer ekfG (a):
(b) ihre assoziierte (unendliche) kfG;
(c) eine gleichwertige RRPG mit Glushkov-NEAs auf rechten Regelseiten;
(d) eine aus dieser RRPG konstruierte Repräsentation der ekfG mit (eindeutigen) rechtslinearen Subgrammatiken; und dazu
(e) die Exemplifikation der Segmentstruktur von Rechtsableitungen.

orientiert am *induktiven Aufbau* der regulären Ausdrücke auf den rechten Regelseiten zu tun. Offensichtlich ist es im Übersetzerbau-Kontext also sinnlos, auf rechten Regelseiten von ekfGen mehrdeutige reguläre Ausdrücke zuzulassen.

In anderen Kontexten, in denen die von einem LR-Parser erwirtschafteten Feinstrukturinformationen nicht interessieren, scheint wiederum der konkrete Aufbau der regulären Ausdrücke auf rechten Regelseiten nebensächlich zu sein. Hier kann daher dem Grammatikautor zugemutet werden, stets eindeutige rechte Regelseiten zu formulieren, was immer möglich ist. Alternativ kann auch der Parsergenerator dahingehend modifiziert werden, daß die rechtslinearen Subgrammatiken durch den Umbau nicht der NEAs, sondern erst der entsprechenden Potenzautomaten gewonnen werden. Da die Potenzautomaten deterministisch sind, sind die Subgrammatiken dann eindeutig. Dieses Verfahren konstruiert deterministische Parser, wenn die zugrundeliegende ekfG eine ELR(k)-Grammatik im Sinne von Heilbrunner (1979) ist; allerdings mit dem Problem, daß die Ableitungsstrukturen dieser Parser in der Regel nicht mehr den direkten Rückschluß auf die Strukturen der regulären Ausdrücke gestatten.

Abweichend von Heilbrunner (1979), wollen wir daher – noch nicht ganz exakt – eine ekfG G_e eine „ELR(k)-Grammatik“ dann nennen, wenn die (möglicherweise unendliche) assoziierte kfG G eine LR(k)-Grammatik ist *und* die regulären Ausdrücke in G_e eindeutig sind; oder gleichbedeutend, wenn die auf dem beschriebenen Weg aus G_e erzeugte Grammatik G^H mit rechtslinearen Subgrammatiken, die eine endliche Repräsentation von G ist, eine LR(k)-Grammatik ist.

Diese Definition muß noch präzisiert werden in bezug auf den vorkommenden Begriff der Eindeutigkeit regulärer Ausdrücke, dem wir uns jetzt zuwenden wollen.

9.2 Endliche Automaten zu regulären Ausdrücken und Eindeutigkeitskonzepte

9.2.1 Brüggemann-Klein (1993)

Brüggemann-Klein beginnt die Zusammenfassung ihres feinen Journalbeitrags von 1993 zur Konstruktion endlicher Automaten aus regulären Ausdrücken mit der Feststellung

> *„It is a well-established fact that each regular expression can be transformed into a nondeterministic finite automaton (NFA) with or without ε-transitions, and all authors seem to provide their own variant of the construction. Of these,*

Berry and Sethi (1986) have shown that the construction of an ε-free NFA due to Glushkov (1961) is a natural *representation of the regular expression because it can be described in terms of the Brzozowski derivatives (Brzozowski 1964) of the expression.“*

Informell erklärt, legt bekanntlich die angesprochene Konstruktion eines ε-freien NEAs nach Glushkov (1961) ε-freie Wege durch die Vorkommen der Alphabetzeichen in einem regulären Ausdruck – Abbildung 17 zeigte bereits ein Beispiel. Brüggemann-Klein (1993) stellt einen neuen Algorithmus zur Konstruktion des Glushkov-NEAs M_E zu einem regulären Ausdruck E vor, dessen Effizienzsteigerung darauf fußt, daß er E zunächst in linearer Zeit (bezogen auf die Größe von E) in den regulären Ausdruck $E^\bullet$ in sogenannter *Stern-Normalform* transformiert.

Neben dieser ersten Rolle der Stern-Normalform in der Effizienzsteigerung der neuen Glushkov-NEA-Konstruktion spielt sie laut Brüggemann-Klein (1993) noch eine zweite Rolle – nämlich im Kontext unterschiedlicher Konzepte der Eindeutigkeit regulärer Ausdrücke. Wir studieren die Definition des Stern-Normalform-Operators $^\bullet$ nun näher, um Brüggemann-Klein (1993) gerade in Bezug auf diese zweite Rolle zu kritisieren.

$E^\bullet$ ist ein regulärer Ausdruck, der denselben Glushkov-NEA wie E ergibt – d. h. es ist $M_{E^\bullet} = M_E$ –, der jedoch die Eigenschaft aufweist, daß anschaulich für jeden gesternten Unterausdruck F^* von $E^\bullet$ das Sternen von F in den Glushkov-NEA $M_{E^\bullet}$ Rückwärts-Kanten einbringt, die darin garantiert nicht schon durch F selbst vorhanden sind (was dann zur Effizienzsteigerung der Konstruktion von $M_{E^\bullet}$ genutzt werden kann). Erreicht wird dies, indem gesternte Unterausdrücke von E „unterhalb“ der Sternoperatoren gegebenenfalls umgeformt werden, wozu der Hilfsoperator $^\circ$ dient. Ihn definiert Brüggemann-Klein (1993) induktiv durch

$$
\begin{array}{ll}
[E = \emptyset \text{ oder } E = \varepsilon] & E^\circ = \emptyset \\
[E = a] & E^\circ = E \\
[E = F + G] & E^\circ = F^\circ + G^\circ \\
[E = FG] & E^\circ = \begin{cases} FG\,, & \text{falls } \varepsilon \notin L(F) \cup L(G) \\ F^\circ G\,, & \text{falls } \varepsilon \in L(G) \setminus L(F) \\ FG^\circ\,, & \text{falls } \varepsilon \in L(F) \setminus L(G) \\ F^\circ + G^\circ\,, & \text{falls } \varepsilon \in L(F) \cap L(G) \end{cases} \\
[E = F^*] & E^\circ = F^\circ
\end{array}
$$

Der Stern-Normalform-Operator $^\bullet$ wird dann definiert durch

$$
\begin{array}{ll}
[E = \emptyset \text{ oder } E = \varepsilon \text{ oder } E = a] & E^\bullet = E \\
[E = F + G] & E^\bullet = F^\bullet + G^\bullet \\
[E = FG] & E^\bullet = F^\bullet G^\bullet \\
[E = F^*] & E^\bullet = F^{\bullet\circ *}
\end{array}
$$

Und E und $E^\bullet$ weisen in der Tat gleiche Glushkov-NEAs $M_E = M_{E^\bullet}$ auf; für den Beweis wird auf (Brüggemann-Klein 1993) verwiesen.

Ein reduziertes Beispiel ist der Ausdruck $(a^*b^*)^*$, für dessen Stern-Normalform sich ergibt:

$$
\begin{aligned}
(a^*b^*)^{*\bullet} &= (a^*b^*)^{\bullet\circ *} \\
&= (a^{*\bullet}b^{*\bullet})^{\circ *} \\
&= (a^{\bullet\circ *}b^{\bullet\circ *})^{\circ *} \\
&= (a^*b^*)^{\circ *} \\
&= (a^{*\circ} + b^{*\circ})^* \\
&= (a^\circ + b^\circ)^* \\
&= (a+b)^* \ .
\end{aligned}
$$

Der Leser überzeuge sich anhand von Abbildung 18, daß die Ausdrücke $(a^*b^*)^*$ und $(a+b)^* = (a^*b^*)^{*\bullet}$ zum selben Glushkov-NEA führen.

Die Autorin befaßt sich dann auch mit Konzepten der Eindeutigkeit von regulären Ausdrücken. Anknüpfend an den Beginn des Unterabschnitts, zitieren wir weiter aus der Zusammenfassung von (Brüggemann-Klein 1993):

> „*This concept [star normal form] is also useful for characterizing the relationship between two types of unambiguity that have been studied in the literature. Namely [...], modulo a technical condition, an expression is strongly unambiguous (Sippu and Soisalon-Soininen 1988) if and only if it is weakly unambiguous (Book et al. 1971) and in star normal form.*“

Die beiden verschiedenen angesprochenen Eindeutigkeitskonzepte wollen wir im folgenden näher betrachten. Dabei sind nur noch reguläre Ausdrücke zugelassen, die $\emptyset$ nicht enthalten.

Informell ist ein regulärer Ausdruck E genau dann *schwach eindeutig* (Book et al. 1971), wenn jedes Wort zu E auf eindeutige Weise auf einem „korrekten Weg“ durch die Vorkommen von Alphabetzeichen in E gelesen werden kann. Da der Glushkov-NEA M_E zu E

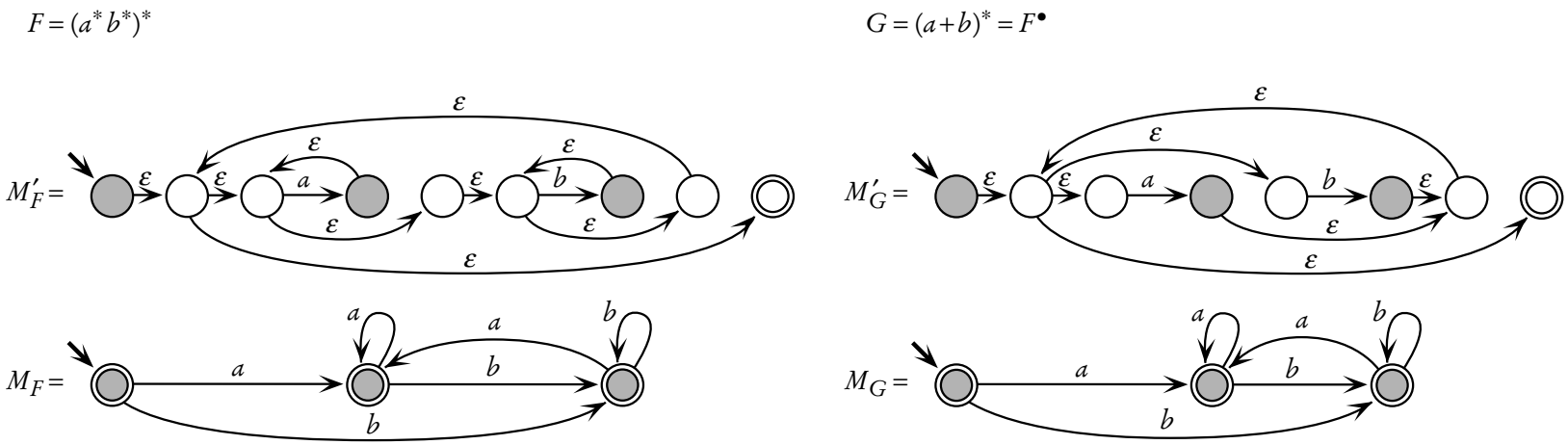


Abbildung 18: Gleiche Stern-Normalform garantiert gleiche Glushkov-NEAs:

Im gezeigten Beispiel besitzen die regulären Ausdrücke $F = (a^*b^*)^*$ und $G = (a+b)^* = F^\bullet$ unterschiedliche ε-NEAs M'_F bzw. M'_G (M'_F ist mehrdeutig, M'_G dagegen eindeutig). Ihre Stern-Normalform ist jedoch dieselbe ($G^\bullet = G = F^\bullet$), weswegen ihre Glushkov-NEAs M_F und M_G übereinstimmen.

Am selben Beispiel kann nachvollzogen werden, daß die Glushkov-NEAs gerade diejenigen Automaten sind, die durch die Stardardtechniken der Elimination von ε-Übergängen und Entfernung unerreichter Zustände und Übergänge aus den ε-NEAs M'_F bzw. M'_G gewonnen werden.

Und schließlich ist G in ε-Normalform ($G^\bullet = G$), und daher sind M'_G und M_G gleichzeitig eindeutig, denn die Verdichtung von M'_G zu M_G eliminiert dann nur eindeutige ε-Wege in M'_G.

gerade diese Wege modelliert, ist E genau dann schwach eindeutig, wenn M_E eindeutig ist. Dagegen ist E genau dann *stark eindeutig* (Sippu und Soisalon-Soininen 1988), wenn die Zerlegung jedes Wortes zu E in Teilwörter entsprechend dem induktiven syntaktischen Aufbau von E eindeutig ist.

Beispielsweise ist der Ausdruck $(a^*b^*)^*$ schwach eindeutig, da es natürlich eindeutig ist, welchem ihrer Vorkommen im Ausdruck die Zeichen a und b in jeder beliebigen Folge von ihnen zuzuordnen sind. Er ist aber nicht stark eindeutig, denn für (sogar) keine einzige Sequenz von Zeichen a und b ist deren Entstehen über Anwendungen der vorkommenden Stern-Operatoren eindeutig.

Beide Eindeutigkeitsbegriffe können auch induktiv über dem Aufbau von E definiert werden, d. h. ohne Rückgriff auf die Eindeutigkeit irgendeines zu E konstruierten Automaten; siehe dazu Definition 4.3 und Lemma 4.7 in (Brüggemann-Klein 1993). Für uns ist hier jedoch die Äquivalenz zu den Automateneindeutigkeiten von Interesse. Hier ist E stark eindeutig, wenn der mit einem der verschiedenen in der Literatur beschriebenen Verfahren – z. B. dem von Sippu und Soisalon-Soininen (1988) oder dem symmetrischeren Schema aus Abbildung 19, das wir in Beispielen verwenden – zu E konstruierte ε-NEA, den wir mit M'_E bezeichnen wollen, eindeutig ist.

Für die Beispielausdrücke $(a^*b^*)^*$ und $(a+b)^*$ sind die verschiedenen zugehörigen Automaten in Abbildung 18 gezeigt.

Um den Zusammenhang zwischen starker und schwacher Eindeutigkeit eines regulären Ausdrucks E präzise zu beschreiben, führt Brüggemann-Klein (1993) eine Charakterisierung ein, wann kein Unterausdruck von E das leere Wort mehrdeutig denotiert: Induktiv über dem Aufbau von E ist E in *ε-Normalform,* wenn:

$[E = \varepsilon$ oder $E = a]$	E ist in ε-Normalform
$[E = F + G]$	E ist in ε-Normalform, wenn F und G in ε-Normalform sind und $\varepsilon \notin L(F) \cap L(G)$.
$[E = FG]$	E ist in ε-Normalform, wenn F und G in ε-Normalform sind
$[E = F^*]$	E ist in ε-Normalform, wenn F in ε-Normalform ist und $\varepsilon \notin L(F)$.

Diese Eigenschaft kann in linearer Zeit überprüft werden.[45]

[45] Ob auch die *Transformation* eines Ausdrucks in ε-Normalform (mit einem geeigneten Transformationsschema) in linearer Zeit möglich ist, wird in (Brüggemann-Klein 1993) nicht beantwortet. Ein dabei

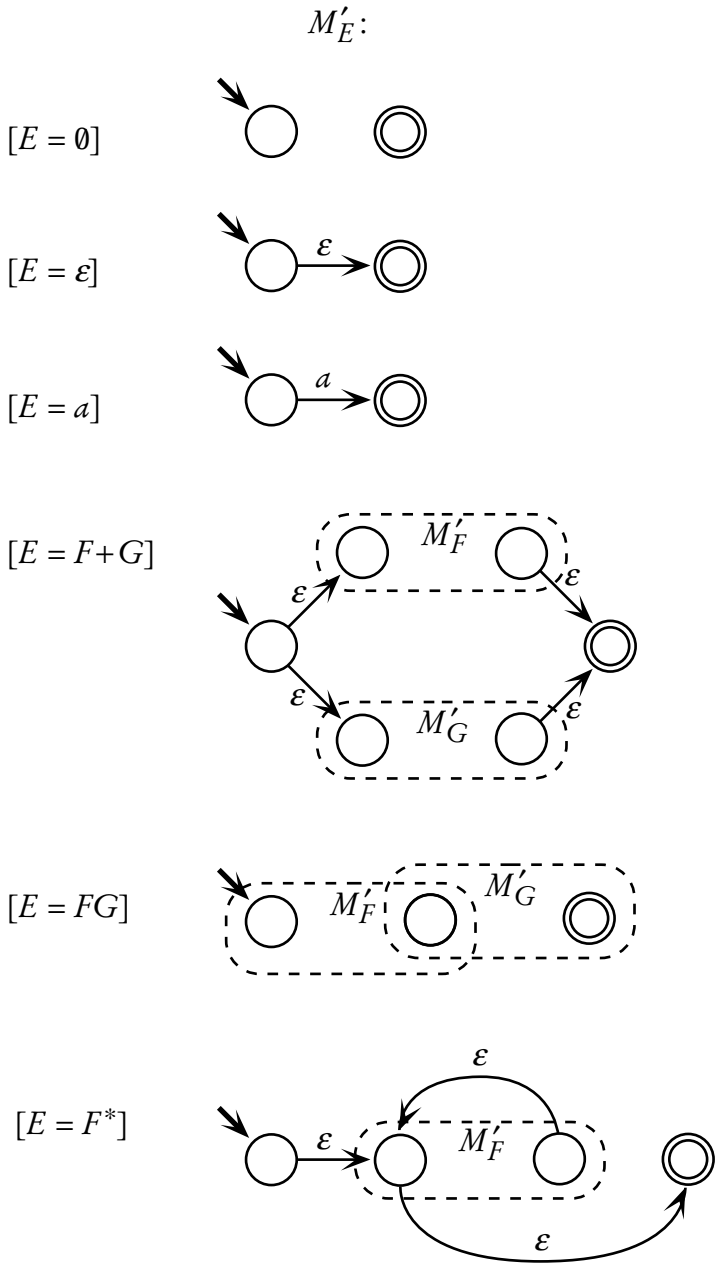


Abbildung 19: Der mit der gezeigten induktiven Konstruktion zu einem regulären Ausdruck E aufgebaute ε-NEA M'_E reflektiert die induktive Struktur von E.

Brüggemann-Klein (1993) arbeitet schließlich heraus, daß ein regulärer Ausdruck E genau dann stark eindeutig ist, wenn E

(1) schwach eindeutig und

(2) in Stern-Normalform und

(3) in ε-Normalform ist.

9.2.2 Verbesserungen

Wir werden im folgenden zeigen, daß sich der zuletzt gesehene Zusammenhang zwischen starker und schwacher Eindeutigkeit einfacher formulieren läßt. Es wird sich nämlich herausstellen, daß jeder ($\emptyset$-freie) reguläre Ausdruck in ε-Normalform bereits in Stern-Normalform ist. Insbesondere sollten wir also nicht die zitierte Sicht von Brüggemann-Klein (1993) übernehmen, ε-Normalform sei für den Zusammenhang zwischen starker und schwacher Eindeutigkeit eine *„technical condition"*. Vielmehr gelangen wir zu der Sicht, daß die Stern-Normalform eines Ausdrucks $E = E^\bullet$ alleine sicherstellt, daß sein ε-freier Glushkov-NEA $M_{E^\bullet}$ mit schlimmstenfalls quadratischem statt kubischem Aufwand erstellt werden kann; dagegen garantiert die *echt stärkere* Bedingung der ε-Normalform von E *alleine* die Äquivalenz der Eindeutigkeit von M_E und der des ε-NEAs M'_E, der den induktiven Aufbau von E wiedergibt, sprich: die Äquivalenz der starken und schwachen Eindeutigkeit von E.

Nicht nur für den Beweis *dieser* Behauptung ist das folgende Lemma nützlich:

Lemma 9.1
Für einen regulären Ausdruck E (der auch $\emptyset$ enthalten darf) gilt

$$\varepsilon \notin L(E) \quad \Rightarrow \quad E^\circ = E \; .$$

Beweis: Sei $\varepsilon \notin L(E)$. Die Fälle $E = \varepsilon$ und $E = F^*$ sind nicht möglich, und für $E = \emptyset$ sowie $E = a$ ist $E^\circ = E$ bereits nach Definition gegeben.

anfallendes Teilproblem ist die Frage, wie zu einem Ausdruck E ein Ausdruck E' mit $L(E') = L(E) \setminus \{\varepsilon\}$ gefunden werden kann. Ziadi (1996) bietet hierzu einen Algorithmus an. Dieser produziert allerdings einen Ausdruck E', dessen Größe $O(|E|\log|E|)$ ist, d. h. nichtlinear zur Größe $|E|$ von E wachsen kann. Allein E' auszugeben ist also nicht mit $O(|E|)$ Zeitaufwand möglich. Allerdings kann eine Baumrepräsentation von E' aus einer ebensolchen von E in linearer Zeit berechnet werden. Diese Resultate dürften sich auf die Problematik, E in ε-Normalform zu transformieren, übertragen.

Für $E = F + G$, d. h. $E^\circ = F^\circ + G^\circ$, ist $\varepsilon \notin L(E) \Leftrightarrow \varepsilon \notin L(F) \cup L(G)$, d. h. es ist sowohl $F^\circ = F$ als auch $G^\circ = G$ nach Induktionsvoraussetzung. Also auch $E^\circ = F^\circ + G^\circ = F + G = E$.

Es verbleibt der Fall $E = FG$, wo wieder $\varepsilon \in L(F) \cap L(G)$ nicht sein kann und bei $\varepsilon \notin L(F) \cup L(G)$ bereits $E^\circ = FG = E$ nach Definition gilt. Für $\varepsilon \in L(G) \setminus L(F)$, also $F^\circ = F$ nach Induktionsvoraussetzung, ist $E^\circ = F^\circ G = FG = E$. $\varepsilon \in L(F) \setminus L(G)$ symmetrisch. □

Lemma 9.2
Sei E ein $\emptyset$-freier regulärer Ausdruck.

Ist E in ε-Normalform, so ist E auch in Stern-Normalform, d. h. $E = E^\bullet$.

Beweis: Sei E $\emptyset$-frei und in ε-Normalform.

Im Falle $E = \varepsilon$ oder $E = a$ ist $E = E^\bullet$ nach Definition.

Für $E = F + G$ bzw. $E = FG$ sind F und G in ε-Normalform, nach Induktionsvoraussetzung ist $F = F^\bullet$ und $G = G^\bullet$, und $E^\bullet = F^\bullet + G^\bullet = F + G = E$ bzw. $E^\bullet = F^\bullet G^\bullet = FG = E$.

Schließlich ist für $E = F^*$ nun F in ε-Normalform und $\varepsilon \notin L(F)$, und es gilt $F = F^\bullet$ nach Induktionsvoraussetzung, weswegen $F^{\bullet\circ} = F^\circ$ ist. Dann aber gilt mit dem vorigen Lemma 9.1 $F^\circ = F$, und deshalb ist $E^\bullet = F^{\bullet\circ *} = F^{\circ *} = F^* = E$. □

Übrigens gilt die umgekehrte Richtung der Behauptung im Lemma nicht; beispielsweise ist $E = a^* + b^*$ zwar in Stern-Normalform, jedoch nicht in ε-Normalform.

Damit wissen wir nun, daß ε-Normalform nicht etwa ein konkurrierendes, sondern ein ein echt stärkeres Konzept als Stern-Normalform ist, weswegen sich der von Brüggemann-Klein (1993) beschriebene Zusammenhang zwischen starker und schwacher Eindeutigkeit regulärer Ausdrücke vereinfachen läßt:

Theorem 9.3 (Rolle der ε-Normalform)
Sei E ein $\emptyset$-freier regulärer Ausdruck. Dann ist E genau dann stark eindeutig, wenn E

(1) schwach eindeutig und

(2) in ε-Normalform ist. □

Daß, wie in Lemma 9.1 gezeigt, $\varepsilon \notin L(E)$ bereits $E^\circ = E$ impliziert, erlaubt übrigens noch weitere Vereinfachungen an der Präsentation von Brüggemann-Klein (1993). Zum einen

läßt sich der Operator $^\circ$ deutlich einfacher (Fall $[E = FG]$) formulieren durch

$$
\begin{array}{ll}
[E = \emptyset \text{ oder } E = \varepsilon] & E^\circ = \emptyset \\
[E = a] & E^\circ = E \\
[E = F + G] & E^\circ = F^\circ + G^\circ \\
[E = FG] & E^\circ = \begin{cases} FG\,, & \text{falls } \varepsilon \notin L(F) \cap L(G) \\ F^\circ + G^\circ\,, & \text{falls } \varepsilon \in L(F) \cap L(G) \end{cases} \\
[E = F^*] & E^\circ = F^\circ
\end{array}
$$

Und diese Definition „paßt“ dann auch viel klarer zu der von Brüggemann-Klein (1993) sehr plastisch mit *„remove feedback edges“* beschriebenen Wirkung dieses Operators.

Ferner läßt sich dann auch der kombinierte Operator $^{\bullet\circ}$ aus Lemma 3.7 von (Brüggemann-Klein 1993) induktiv simpler formulieren (Fall $[E = FG]$) durch

$$
\begin{array}{ll}
[E = \emptyset \text{ oder } E = \varepsilon] & \emptyset^{\bullet\circ} = \emptyset = \varepsilon^{\bullet\circ} \\
[E = a] & E^{\bullet\circ} = E \\
[E = F + G] & E^{\bullet\circ} = F^{\bullet\circ} + G^{\bullet\circ} \\
[E = FG] & E^{\bullet\circ} = \begin{cases} F^\bullet G^\bullet\,, & \text{falls } \varepsilon \notin L(F) \cap L(G) \\ F^{\bullet\circ} + G^{\bullet\circ}\,, & \text{falls } \varepsilon \in L(F) \cap L(G) \end{cases} \\
[E = F^*] & E^{\bullet\circ} = F^{\bullet\circ}
\end{array}
$$

9.2.3 Rekonstruktion induktiver Wortstruktur aus Zustandsfolgen

Wie wir wissen, spiegelt der ε-NEA M'_E zu einem regulären Ausdruck E unmittelbar die induktive Struktur von E wider. Es wird den Leser daher sicherlich nicht erstaunen, daß es unschwer möglich ist, zu einem von M'_E akzeptierten Wort $w \in L(E) = L(M'_E)$ eine Art Protokoll der Entstehung von w durch die Konstituenden des Ausdrucks E während eines linearen Durchlaufs durch eine akzeptierende Übergangsfolge von M'_E auf w zu erstellen – insbesondere die durchlaufenen ε-Übergänge von M'_E sind wesentliche Informationsstützen, und die diesbezügliche Symmetrie des in Abbildung 19 gezeigten Konstruktionsschemas für M'_E gestattet, daß hierfür ein Vorwärts- wie auch ein Rückwärts-Durchlauf durch die Übergangsfolge gleichermaßen gut geeignet ist.

Um dies genauer auszuführen, betrachten wir zu einer gegebenen ekfG G_e eine beliebige Regel $A \to E$; E ist also ein regulärer Ausdruck über den Symbolen V von G_e.

Wie die Expansion w von E (d. h. $w \in L(E)$) induktiv aus den Bestandteilen von E entsteht, kann intuitiv modelliert werden, indem wir zu E eine kontextfreie Subgrammatik $H_A^l = (V_A, V, P_A^l, S_A)$ konstruieren, in der V die Menge der Terminale ist und deren Regelmenge P_A^l und Symbolfolge V_A dadurch entsteht, daß, beginnend mit der (später) gemischt kontextfreien und erweitert-kontextfreien Regelmenge $\{S_A \to E\}$ sowie der initialen Symbolmenge $V \cup \{S_A\}$, solange „noch nicht rein kontextfreie" Regeln der Form $B \to \alpha(\beta)^* \gamma$ durch $B \to \alpha B' \gamma$ und *(links*rekursiv) $B' \to B'\beta \mid \varepsilon$ sowie solche der Form $B \to \alpha(\beta_1 \mid \ldots \mid \beta_n)\gamma$ durch $B \to \alpha B' \gamma$ und $B' \to \beta_1 \mid \ldots \mid \beta_n$ ersetzt und das bis dahin noch nicht vorkommende Nichtterminal B' aufgenommen wird, bis nur noch rein kontextfreie Regeln vorliegen.[46]

Analog können wir auch eine kontextfreie Subgrammatik $H_A^r = (V_A, V, P_A^r, S_A)$ konstruieren, die $*$-Operatoren in P_A^r nicht links-, sondern rechtsrekursiv modelliert.

Es ist leicht einzusehen, daß jedes der neu aufgenommenen Nichtterminale genau mit einem Vorkommen eines $+$- oder $*$-Operators in E korrespondiert und daß ein (bzw. der) Ableitungsbaum in H_A^l (oder auch H_A^r) über w strukturell eine (bzw. die) induktive Entstehung von w durch die Konstituenden von E unmittelbar wiedergibt.

Die in diesem Ableitungsbaum angewandten Regeln von H_A^l (bzw. H_A^r) können nun systematisch Regel für Regel während eines linearen Durchlaufs durch die entsprechende Übergangsfolge, mit der w vom ε-NEA M_E' akzeptiert wird, aufgezählt werden. Und zwar während eines *Vorwärts*-Durchlaufs in der Prä-Ordnung einer Top-down-*links-rechts*-Traversierung für den Ableitungsbaum von w bezüglich H_A^r, dagegen während eines *Rückwärts*-Durchlaufs in der Prä-Ordnung eines Top-down-*rechts-links*-Durchlaufs bezüglich H_A^l. Die Entscheidung über die jeweils nächste für den gesuchten Ableitungsbaum bestimmte Regel findet dabei systematisch an den Verzweigungen (bzgl. der Durchlaufrichtung) der Übergänge von M_E' statt. Und diese Entscheidungssituationen betreffen stets ε-Übergänge von M_E'. Abbildung 20 illustriert dies anhand eines Beispiels.

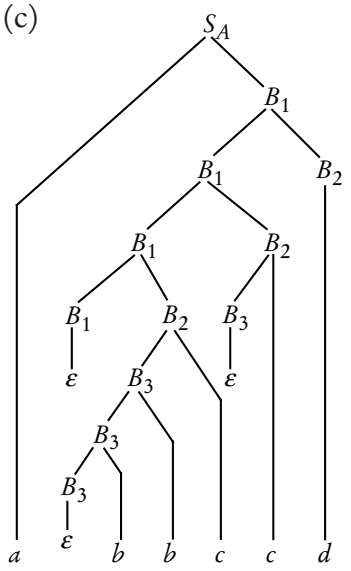
Obiges zusammenfassend, haben wir festgestellt, daß, gegeben ein linearer Durchlauf durch eine akzeptierende Übergangsfolge des ε-NEA M_E' auf w, die Rekonstruktion der entsprechenden induktiven Struktur von w durch die Konstituenden von E einfach möglich ist und daß die dabei durchlaufenen ε-Übergänge von M_E' wesentliche Informationsstützen sind.

[46] Mit Hilfe der so entstehenden kontextfreien Subgrammatiken zu allen Nichtterminalen von G_e könnte wiederum eine endliche Repräsentationsgrammatik G^H im Sinne von Heilbrunner (1979) für G_e bzw. dessen assoziierte kfG konstruiert werden, worauf wir hier jedoch nicht hinauswollen.

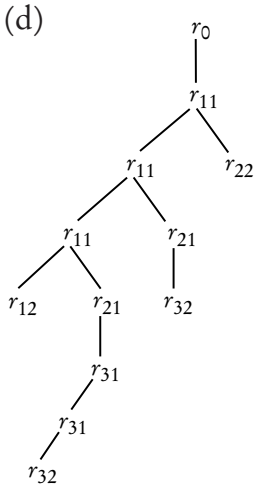
(a) $A \to a\,(\underbrace{\underbrace{\underbrace{b^*}_{B_3}\ c}_{B_2} + d}_{B_1})^*$

(b)
$$S_A \to aB_1 \qquad (r_0)$$
$$B_1 \to B_1B_2 \mid \varepsilon \qquad (r_{11}, r_{12})$$
$$B_2 \to B_3c \mid d \qquad (r_{21}, r_{22})$$
$$B_3 \to B_3b \mid \varepsilon \qquad (r_{31}, r_{32})$$

(c)



(d)



(e)

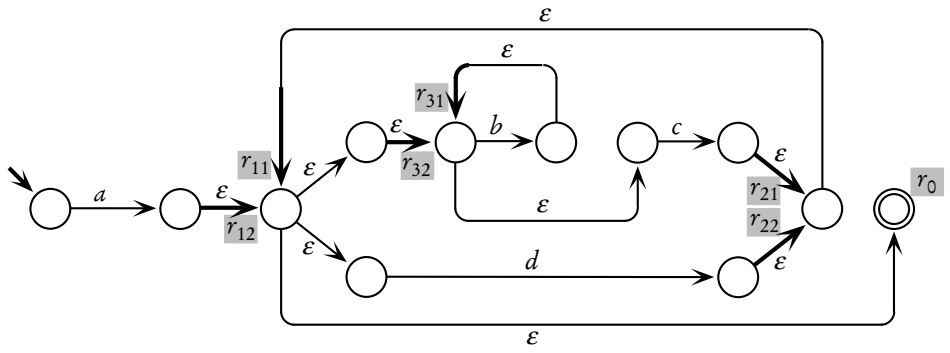


Abbildung 20: Beispielhafte Berechnung induktiver regulärer Struktur (Iteration linksrekursiv modelliert) aus einem ε-NEA-Durchlauf: Für die Expansion $w = abbccd$ des regulären Ausdrucks (a) modelliert der Ableitungsbaum (c) zur kfG (b) strukturell die induktive Enstehung von w aus den Konstituenden des regulären Ausdrucks. An den ε-Übergängen des zugehörigen ε-NEAs (e) können die kfG-Regeln systematisch so annotiert werden (grau unterlegt), daß sich daraus bei einem Rückwärts-Durchlauf durch die Übergänge, mit denen w akzeptiert wird, die im Ableitungsbaum angewandten Regeln in Top-down-rechts-links-Prä-Ordnung ergeben, nämlich $r_0, r_{11}, r_{22}, r_{11}, r_{21}, r_{32}, r_{11}, r_{21}, r_{31}, r_{31}, r_{32}, r_{12}$. Vgl. den Regelbaum (d).

Für den Anwendungskontext der ELR-Parserkonstruktion für ekfGen wiederum bedeuten unsere Überlegungen, daß es unproblematisch möglich ist, aus einem vom Parser für ein reduziertes Handle w gelieferten Rückwärts-Durchlauf durch den entsprechenden ε-NEA M'_E zur rechten Seite E (bzw. die zugehörige rechtslineare Subgrammatik) einen Ableitungsbaum zu erstellen, der den induktiven Aufbau von w wie gesehen strukturell mit den Mitteln einer intuitiven kfG wiedergibt. Eine solche kfG korrespondiert sicherlich eher mit der kognitiven Sicht eines Spezifikateurs auf eine ekfG, insbesondere dann, wenn diese mit einer sinnvollen Parametrisierung zu einer „2-Stufen-*ekfG*“ angehoben werden soll und dann per Angabe von Transformationsregeln zu klären ist, welche klassische „2-Stufen-*kfG*“ damit abkürzend gemeint ist (Demuth et al. 1997).

Wir sollten uns nun noch dafür interessieren, unter welchen Umständen die gleiche beschriebene Aufgabe – die Rekonstruktion der induktiven Wortstruktur eines Wortes $w \in L(E)$ aus einem Automatendurchlauf auf w – auch dann noch stets möglich ist, wenn wir nicht den ε-NEA M'_E, sondern den ε-freien Glushkov-NEA M_E heranziehen.

Die richtige Sicht darauf, warum und „um wieviel“ dieses Problem schwieriger ist, gewinnen wir, wenn wir nocheinmal Abbildung 18 betrachten. Dort ist zum einen an zwei regulären Ausdrücken $F = (a^*b^*)^*$ und $G = F^\bullet = (a+b)^*$ exemplifiziert, daß gleiche Stern-Normalform gleiche Glushkov-NEAs impliziert: Es ist dort $G^\bullet = G = F^\bullet$, und die ε-NEAs M'_F und M'_G unterscheiden sich zwar (M'_F ist mehrdeutig, M'_G dagegen eindeutig), jedoch sind die Glushkov-NEAs M_F und M_G gleich. Zum anderen kann am selben Beispiel nachvollzogen werden, daß die Glushkov-NEAs gerade diejenigen Automaten sind, die durch die Stardardtechniken der Elimination von ε-Übergängen und Entfernung unerreichter Zustände und Übergänge aus den ε-NEAs M'_F bzw. M'_G gewonnen werden.[47]

Rekapitulieren wir jetzt, daß ein beliebiger Ausdruck E genau dann stark eindeutig ist, wenn er schwach eindeutig und in ε-Normalform (und mithin in Stern-Normalform) ist, daß also M'_E genau dann eindeutig ist, wenn M_E eindeutig und E in ε-Normalform ist, so schließen wir, daß es genau die Eigenschaft der ε-Normalform von E ist, die sicherstellt, daß bei der Elimination von ε-Übergängen von M'_E nach M_E die gebrückten ε-Wege in M'_E garantiert eindeutig sind. Dies können wir für die Beispielausdrücke F (nicht in ε-Normalform) und G (in ε-Normalform) in der Abbildung auch nachvollziehen, wo in M'_F ein Zyklus über ε-Übergänge vorliegt, in M'_G jedoch zwischen je zwei Zuständen höchstens ein Weg über ε-Übergänge existiert.

Wir können dies nun so interpretieren, daß es – quasi durch Elimination von ε-Über-

[47] Siehe den Beweis von Lemma 4.6 in (Brüggemann-Klein 1993).

gängen – stets eindeutig möglich ist, aus einer auf $w \in L(E)$ vom ε-NEA M'_E durchlaufenen Übergangsfolge auf eine Übergangsfolge des Glushkov-NEAs M_E zu schließen. Der umgekehrte Schluß, also im wesentlichen die eindeutige Rekonstruktion der von M'_E zu M_E eliminierten ε-Übergänge in Übergangsfolgen, ist dagegen nur dann stets möglich, wenn der Ausdruck E in ε-Normalform vorliegt. Ist dies der Fall, so ist die Verdichtung von M'_E zu M_E eindeutig reversibel, ansonsten nicht. Das Ablegen der zu rekonstruierenden ε-Wege in M'_E kann günstig „an“ den einzelnen Übergängen von M_E erfolgen.

Gerade die eliminierte Information der ε-Übergänge von M'_E war es aber, die wir wesentlich für die Erklärung der Struktur eines Wortes $w \in L(E)$ in Terminologie der Operatoren in E auf Grundlage eines Durchlaufs von M'_E auf w ausgenutzt hatten. Damit ist klar, daß es nur dann stets möglich ist, dieselbe Aufgabe mit Hilfe eines Durchlaufs von M_E zu bewerkstelligen, wenn E in ε-Normalform ist!

Der reguläre Ausdruck $a(b^*c + d)^*$ aus Abbildung 20 weist die Eigenschaft der ε-Normalform auf, der Ausdruck $(a^*b^*)^*$ aus Abbildung 17(b) auf Seite 114 jedoch nicht.

9.3 Die Definition von ELR(k)-Grammatiken

An dieser Stelle verfügen wir nun endlich über die ausreichenden Kenntnisse, um eine gänzlich präzise Definition zu formulieren, wann wir eine ekfG eine ELR(k)-Grammatik nennen wollen. In unserer Vorabdefinition auf Seite 115 war unscharf geblieben, was wir unter der „Eindeutigkeit“ eines regulären Ausdrucks auf rechter Regelseite verstehen wollten; dies können wir nun klarstellen.

Dazu erinnern wir uns, daß unser Interesse für Konzepte der Eindeutigkeit regulärer Ausdrücke daher rührte, daß wir mit einer gegebenen ekfG über dem induktiven Aufbau der darin vorkommenden regulären Ausdrücke semantische Aktionen assoziieren können wollen. Mit den Ergebnissen aus dem vorangehenden Unterabschnitt ist jetzt klar, daß das für unsere Zwecke adäquate Eindeutigkeitskonzept das der starken Eindeutigkeit ist. Unsere präzise Definition des ELR(k)-Begriffs für ekfGen lautet daher:

Eine ekfG G_e heißt *ELR(k)-Grammatik,* wenn die (möglicherweise unendliche, reduzierte) assoziierte kfG G eine LR(k)-Grammatik ist und die ($\emptyset$-freien) regulären Ausdrücke in G_e stark (!) eindeutig sind.[48]

[48] Wir werden in Abschnitt 9.6.7 genauer beleuchten, daß der hiermit formulierte ELR(k)-Begriff für ekfGen effektiv identisch ist mit demjenigen, den schon Madsen und Kristensen (1976) beschreiben. Da ihre Journalveröffentlichung gleichzeitig die älteste zum Thema überhaupt ist, muß *dieser* ELR(k)-Begriff, den

Kombinieren wir unsere erarbeiteten Kenntnisse, so gelangen wir zum

Theorem 9.4 (ELR(k)-Eigenschaft einer ekfG)
Eine ekfG G_e ist genau dann eine ELR(k)-Grammatik, wenn

(1) diejenige Repräsentations-kfG G_1^H von G_e, deren rechtslineare Subgrammatiken aus den ε-NEAs der regulären Ausdrücke in G_e gewonnen werden, eine LR(k)-Grammatik ist;

oder alternativ, wenn

(2) diejenige Repräsentations-kfG G_2^H von G_e, deren rechtslineare Subgrammatiken aus den Glushkov-NEAs der regulären Ausdrücke in G_e gewonnen werden, eine LR(k)-Grammatik ist und alle regulären Ausdrücke in G_e in ε-Normalform sind.

Beweis: Unser Theorem 9.3 über die Rolle der ε-Normalform hatte den Zusammenhang zwischen starker und schwacher Eindeutigkeit eines regulären Ausdrucks etwas schlanker als Brüggemann-Klein (1993) formulieren können. Aus ihm und der Äquivalenz der Eindeutigkeit eines beliebigen NEAs und seiner gesehenen Umformulierung zu einer rechtslinearen Grammatik folgt, daß zu einer Regel $A \to E$ von G_e die aus dem ε-NEA M'_E gewonnene Subgrammatik von G_1^H genau dann eindeutig ist, wenn die aus dem Glushkov-NEA M_E gewonnene Subgrammatik von G_2^H eindeutig und zudem E in ε-Normalform ist.

Wir bemerken ferner, daß keine Repräsentation von G_e mit einer mehrdeutigen Subgrammatik eine LR(k)-Grammatik sein kann. (G_e ist reduziert.) Daß G_e nun eine ELR(k)-Grammatik genau dann ist, wenn G_1^H eine LR(k)-Grammatik ist, und ebenso auch genau dann, wenn G_2^H eine LR(k)-Grammatik und alle regulären Ausdrücke in G_e in ε-Normalform sind, folgt dann unter Nutzung des präsentierten Kernergebnisses von (Heilbrunner 1979). □

Die konstruktive Formulierung des ELR(k)-Begriffs für ekfGen im Theorem sieht vor der Gewinnung rechtslinearer Subgrammatiken den Zwischenschritt vor, die regulären Ausdrücke auf rechten Regelseiten – wenigstens gedacht – zunächst durch NEAs zu ersetzen. Ohne daß wir dies bisher betont hätten, wird hierdurch die ekfG zu einer RRPG,

Heilbrunner (1979) als MK-ELR(k) bezeichnet, als der originale angesehen werden. Wenn andere Autoren – wie LaLonde (1977, 1979), Purdom und Brown (1981), Chapman (1984), Nakata und Sassa (1986), Shin und Choe (1993) und Lee und Kim (1997) – einen anderen ELR(k)-Begiff prägen, ohne sogar den Unterschied zum Originalbegriff aufzuzeigen, so weist dies einmal mehr auf den allzu lässigen Umgang dieser Autoren mit der zugrundeliegenden Theorie hin.

was wir bedenken sollten, wenn wir nun auch für RRPGen einen tragfähigen ELR(k)-Begriff definieren wollen und anstreben, die Konstruktion von ELR(k)-Parsern für ekfGen auf die von ELR(k)-Parsern für RRPGen zurückführen zu können. Semantische Aktionen wollen wir daher auf seiten der RRPGen mit den Übergängen der RRPAs assoziieren können – was ohnehin naheliegt.

Den Begriff der (möglicherweise unendlichen) *assoziierten* kfG G für eine RRPG G_r übernehmen wir in der offensichtlichen Weise. Und so, wie wir in ekfGen auf reguläre Ausdrücke verzichtet haben, die $\emptyset$ enthalten, wollen wir nun auch RRPAs mit unerreichbaren oder toten Zuständen aussondern: Enthalten die RRPAs von G_r keine solchen pathologischen Zustände und ist zudem die assoziierte Grammatik G reduziert, so sagen wir, daß die RRPG G_r *reduziert* ist, und setzen diese Eigenschaft von nun an voraus. Wir gelangen zur folgenden Definition:

Eine reduzierte RRPG G_r heißt *ELR(k)-Grammatik,* wenn die assoziierte kfG G eine LR(k)-Grammatik ist und die RRPAs von G_r eindeutig sind.

Natürlich sind ekfGen und RRPGen bezüglich ihrer ELR(k)-Eigenschaften dann wie folgt verwandt:

Theorem 9.5 (ELR(k)-Zusammenhang zwischen ekfGen und RRPGen)
Sei G_e eine ekfG, und G_{r1} und G_{r2} seien diejenigen RRPGen, die dadurch entstehen, daß in den Regeln von G_e die regulären Ausdrücke auf rechten Regelseiten in der offensichtlichen Weise ersetzt werden durch ihre ε-NEAs (für G_{r1}) bzw. Glushkov-NEAs (für G_{r2}).

Dann ist G_e genau dann eine ELR(k)-Grammatik, wenn G_{r1} eine ELR(k)-Grammatik ist, oder alternativ, wenn G_{r2} eine ELR(k)-Grammatik ist und die regulären Ausdrücke in G_e in ε-Normalform sind. □

Wir bemerken noch, daß nicht jede RRPG durch Umbau einer ekfG entsteht: Beide betrachteten NEA-Varianten zu einem regulären Ausdruck E – der ε-NEA M'_E wie auch der Glushkov-NEA M_E – weisen beispielsweise die spezielle Eigenschaft auf, daß zu jedem einzelnen Zustand die in diesen verlaufenden Übergänge entweder alle mit ε oder aber alle mit einem für den Zustand eindeutigen Alphabetsymbol erfolgen. Eine weitere besondere Eigenschaft ist, daß niemals Übergänge zurück in den Startzustand führen. Beliebige -RRPGen müssen mitnichten diese speziellen Eigenschaften ihrer RRPAs aufweisen und können dennoch ELR(k)-Grammatiken sein.

9.4 Diskussion

Es bieten sich prinzipiell zwei Strategien an, für ekfGen die *Einbindung semantischer Aktionen* – beispielsweise Code zur automatischen Konstruktion eines Strukturbaums – in die zu generierenden ELR(k)-Parser zu realisieren:

(1) Die für die Konstruktion eines LR(k)-Parsers eingesetzte kfG mit rechtslinearen Subgrammatiken reflektiert die Übergänge der ε-NEAs zu den rechten Regelseiten der ekfG. Es ist dann der *Parser selbst,* der in den Einzelschritten während der Reduktion eines Handles die Übergangsfolge durch den passenden ε-NEA für das Handle offenlegt, mittels derer semantische Aktionen ausgewählt werden.

(2) Die für die Konstruktion eines LR(k)-Parsers eingesetzte kfG gibt in den rechtslinearen Subgrammatiken die ε-freien Glushkov-NEAs zu den rechten Regelseiten der ekfG wieder, und um die Anbindung semantischer Aktionen für ein reduziertes Handle zu leisten, wird auf die entsprechende Übergangsfolge durch den passenden ε-NEA *„außerhalb"* des eigentlichen Parsers rückgeschlossen. (Siehe den Unterabschnitt 9.2.3 über die Rekonstruktion induktiver Wortstruktur aus Zustandsfolgen.)

Des weiteren gibt es auch zwei Möglichkeiten, in einem ELR(k)-Parsergenerator mit der Überprüfung der starken Eindeutigkeit der regulären Ausdrücke auf den rechten Regelseiten einer ekfG bzw. der Eindeutigkeit der RRPAs einer RRPG umzugehen:

(1) Die Überprüfung der

- starken Eindeutigkeit der regulären Ausdrücke einer ekfG bzw.
- Eindeutigkeit der RRPAs einer RRPG

wird *separat* und vor der eigentlichen Generierung des LR(k)-Parsers zur repräsentierenden kfG vorgenommen.

(2) Oder die Überprüfung der Eindeutigkeiten wird in die Generierung des LR(k)-Parsers zur repräsentierenden kfG *integriert:* Ein nicht stark eindeutiger regulärer Ausdruck bzw. ein nicht eindeutiger RRPA ist an einem bestimmten Reduce-reduce-Konflikt zu identifizieren.

Geschieht für eine ekfG die Parsergenerierung über Glushkov-NEAs, so muß gesondert sichergestellt werden, daß alle regulären Ausdrücke der ekfG in ε-Normalform sind.

Die Problematik der Konstruktion von ELR(k)-Parsern ist damit prinzipiell gelöst. Dabei ist, erstens, unsere Herangehensweise theoretisch fundiert gewesen, die Ergebnisse sind

daher vertrauenswürdig. Zweitens stellen wir beruhigt fest: ELR(k)-Parser *sind* deterministische Kellertransduktoren. Und drittens liefern diese Parser klassische Ableitungsbäume (als Rechtsparse); bei Bedarf können die „rechtslastigen" Strukturen dieser Bäume mit einem geeigneten Homomorphismus umgerechnet werden in die entsprechenden Bäume (als Rechtsparse) zu einer für Benutzer intuitiveren kfG.

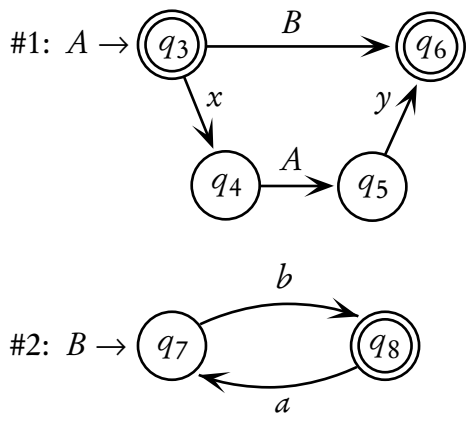
Offensichtlich ist nun auch, wie ESLR(k)- und ELALR(k)-Parser zu konstruieren sind; und von letzterem Verfahren wird auch noch die Rede sein.

9.5 Eine vergleichende Rekonstruktion des Verfahrens von Chapman (1984) mit kontextfreien Mitteln

Indem wir Teile des Journalartikels (Chapman 1984) analysieren und rekonstruieren, beginnen wir nun mit der Analyse von Arbeiten aus der Literatur, die sich mit der Konstruktion von ELR- und verwandten Parsern befassen. Bei dieser Literaturanalyse werden wir nicht rein chronologisch vorgehen. Vielmehr befassen wir uns zunächst mit denjenigen Arbeiten, die RRPGen einsetzen, dann mit der einzigen Arbeit, die ekfGen verwendet. Die RRPG-basierten Arbeiten wiederum behandeln wir ansonsten chronologisch. Unter ihnen jedoch betrachten wir die Arbeit von Chapman (1984) vorgezogen in diesem Abschnitt und mit besonderer Ausführlichkeit.

Dies hat folgende Begründung: Zum ersten sind die formalen Grundlagen der Arbeit von Chapman (1984) – im Gegensatz zu anderen RRPG-basierten – zumindest so tief und akkurat, daß eine Analyse auch ins Detail gehen kann. Zum zweiten können wir an dieser Arbeit am klarsten studieren, wie sogenannte Readback-Automaten konstruiert werden. Diese sind eine Art endlicher Automaten, mit deren Hilfe bei der Reduktion das jeweilige Handle auf dem Keller identifiziert wird, und werden auch in anderen Arbeiten eingesetzt – siehe Abschnitt 9.6 –, dort jedoch in unübersichtlicherer, „optimierter" Form. Und zum dritten weisen Chapmans und unsere eigene Arbeit trotz unterschiedlicher definitorischer Ansätze erstaunliche Analogien auf. Sie rechtfertigen ein näheres Hinsehen, und die anderen Arbeiten in der Literatur können danach schneller abgehandelt werden. Insbesondere sind die noch vor (Chapman 1984) erschienenen Arbeiten von LaLonde (1977, 1979) kaum theoretisch untermauert und eignen sich daher als Ausgangspunkt eines Literaturüberblicks weit weniger.

Hauptanliegen der Arbeit von Chapman (1984) ist, den effizienten Algorithmus von DeRemer und Pennello (1982) zur Bestimmung von LALR(1)-Lookaheads aus der Struktur



$S \to A\$$

$A \to q_3$	$q_3 \to Bq_6 \mid xq_4 \mid \varepsilon$
	$q_4 \to Aq_5$
	$q_5 \to yq_6$
	$q_6 \to \varepsilon$
$B \to q_7$	$q_7 \to bq_8$
	$q_8 \to aq_7 \mid \varepsilon$

Abbildung 21: Die in (Chapman 1984) für ein durchgehendes Beispiel verwendete RRPG und ihre $-erweiterte Reformulierung als kfG mit rechtslinearen Subgrammatiken.

der kanonischen LR(0)-Maschine einer kfG im Zusammenhang mit RRPGen nutzbar zu machen. Chapman setzt ε-freie (im Prizip auch mehrdeutige) RRPAs auf rechten Regelseiten ein, und wie Heilbrunners (1979) abstrahiert auch *sein* ELR(k)-Begriff von der Feinstruktur der reduzierten Handles.

Wir werden – mit unbedeutenden Anpassungen an den Stil der vorliegenden Arbeit – präsentieren, welchen ELR(0)-Parser Chapman zu seiner als Beispiel gewählten RRPG aus Abbildung 21 konstruiert. In derselben Abbildung ist dieser RRPG ihre Reformulierung als erweiterte kfG mit (eindeutigen) rechtslinearen Subgrammatiken gegenübergestellt. Wie wir dann sehen werden, wenn wir mit unseren rein kontextfreien Mitteln Chapmans Readback-Automaten rekonstruieren, ist unsere kfG aus Abbildung 21 eine LALR(1)-Grammatik, und wir würden die RRPG daher – analog unserem strengeren Begriff der ELR(k)-Grammatik – eine ELALR(1)-Grammatik nennen.

Chapmans etwas sorglose Herangehensweise an die Konstruktion eines LALR-Parsers offenbart sich schon auf seinen ersten Schritten:

> *„The classical construction for building LR(0) automata [(DeRemer 1971)] can be applied to RRP grammars by defining an item to be a right part state $q \in Q$, and identifying LR(0) states with sets of items.*

Define a relation $\downarrow$ on items by

$$p \downarrow q \qquad \textit{iff } \exists A : \delta(p,A) \neq \emptyset \wedge A \to q \ .$$

This models one step in the process of „moving the dot down“ to form the closure set of an item set; the entire closure is given by the reflexive transitive closure $\downarrow^*$. Thus a correct LR(0) automaton is given by putting

$$\begin{aligned} \tilde{q}_0 &= \text{closure}\, q_0 \\ \text{Next}(\tilde{q}, X) &= \text{closure} \cdot \text{succ}(\tilde{q}, X) \end{aligned}$$

and

$$\text{Reduce}\, \tilde{q} = \{(A,q) \mid \exists p \in \tilde{q}, \alpha \in V^* : p \in F \cap \hat{\delta}(q,\alpha)\}$$

where

$$\text{succ}(\tilde{q}, X) = \{q \mid \exists p \in \tilde{q} : q \in \delta(p,X)\}$$

and

$$\text{closure}\, \tilde{q} = \{q \mid \exists p \in \tilde{q} : p \downarrow^* q\}.$$[49]

The set $\tilde{Q}$ is identified with the set of item sets, and is thus the smallest $\tilde{Q}$ satisfying

$$\tilde{Q} = \{\text{closure}\, q_0\} \cup \{\text{closure}\, \tilde{q} \mid \exists X \in V, \tilde{p} \in \tilde{Q} : \tilde{q} = \text{succ}(\tilde{p}, X)\}.\text{“}$$

Kein Versuch der Übertragung eines so zentralen Begriffs wie dem des „lebensfähigen Präfix'“ aus der Begriffswelt der kfGen in die Verhältnisse bei RRPGen. Keine Definition der „LR(k)-Gültigkeit“ eines – nun ja doch andersartigen – Items. Die wesentlichen Aufgaben, die Chapmans LR(0)-Maschine erfüllen soll, bleiben unerwähnt, und damit geraten zitierte Wendungen wie *„the classical construction ... can be applied“* oder *„a correct LR(0) automaton is given by putting ...“* zu unbelegten oder leeren Behauptungen – *„correct“* bezüglich wessen ?!

Stellen wir nun in Abbildung 22 Chapmans LR(0)-Maschine für die Beispiel-RRPG derjenigen gegenüber, die wohl eher einer *„classical construction“* entspricht, nämlich der

[49] Chapmans – sicherlich versehentlich – fehlerhafte Formulierung von closure ist hier korrigiert.

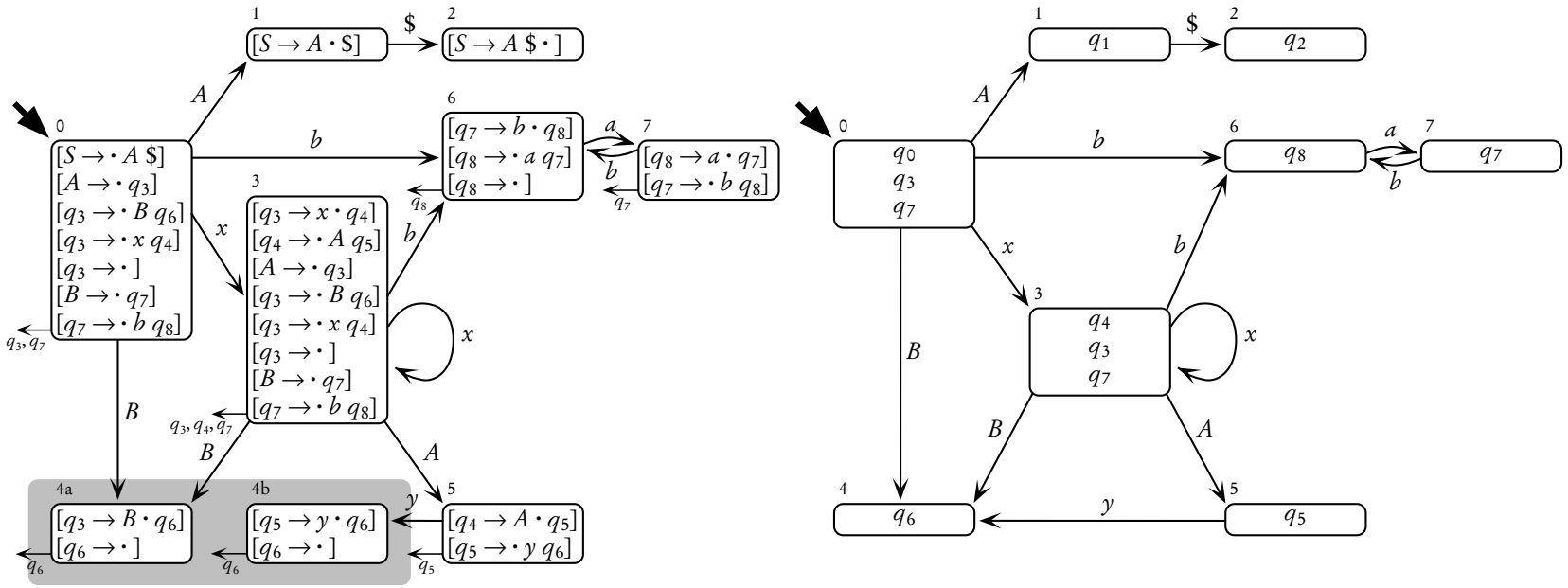


Abbildung 22: Der von Chapman konstruierten LR(0)-Maschine für seine Beispiel-RRPG, rechts gezeichnet, ist links die klassische kanonische LR(0)-Maschine für die geradlinige Reformulierung derselben RRPG als reine kfG gegenübergestellt. Es ist zu erkennen, daß Chapmans Konstruktion zu einem etwas kompakteren Ergebnis führt – siehe die Grauunterlegung von Zuständen links –, sie weist jedoch nicht mehr die Eigenschaft des eindeutigen Access-Symbols auf.

Anmerkungen:

1. Chapman erweitert in seiner Konstruktion die RRPG in der offensichtlichen Weise um eine neue Regel, deren RRPA Zustände q_0, q_1 und q_2 aufweist.

2. In der linken Konstruktion auf klassischerer Grundlage werden Items der Form $[\ldots \to \ldots q_i \cdot]$ und die naheliegenden Übergänge dorthin nur während der Reduktionen der komplexen Handles benötigt und sind zum Zwecke der besseren Vergleichbarkeit mit Chapmans Konstruktion hier nur angedeutet.

kanonischen LR(0)-Maschine für die gesehene kfG-Repräsentation der RRPG, so stellen wir Diskrepanzen fest: Chapmans Maschine ist etwas kompakter, und insbesondere besitzen nicht alle ihrer Zustände die altbekannte Eigenschaft des eindeutigen Access-Symbols !

Der Versuch, eine RRPG mit einer strukturell gleichwertigen kfG zu „simulieren“, insbesondere um die von Chapman für die Reduktionsphasen entworfenen Readback-Automaten mit *klassischen* Mitteln zu rekonstruieren und somit als korrekt einzusehen, bleibt hier offensichtlich stecken.

Woher rührt der Unterschied zwischen den beiden Maschinen? Ein näheres Hinsehen deckt auf, daß Chapmans Maschinen strukturell eine Zusammenfassung von Zuständen der Maschinen nach der klassischen Konstruktion sind, was in Abbildung 22 durch Grauunterlegung angedeutet ist: Die sich anbietende Wahl eines „Items“ als RRPA-Zustand führt bei der naheliegenden Anpassung der „Itemmengenkonstruktion“ dazu, daß darin die Items einer Itemmenge immer noch die möglichen Fortsetzungen der mit der Itemmenge verbundenen lebensfähigen Präfizes beschreiben, jedoch eben nicht mehr die „individuelle Geschichte“ eines jeden Items, wie dies bei der klassischen Konstruktion in der kanonischen LR(0)-Maschine zur kfG-Repräsentation der RRPG noch der Fall ist.

Dies weckt natürlich Erinnerungen an die „allgemeinen LR(k)-Parser“ und ihre „allgemeinen LR(k)-Maschinen“, deren Theorie wir im ersten Teil der vorliegenden Arbeit ausführlich behandelt haben. Und es bedarf keiner großen Anstrengungen mehr, um nun einzusehen, daß Chapmans LR(0)-Maschinen in bezug auf Gesamtstruktur und den Informationsgehalt der einzelnen Itemmengen identisch sind mit der jeweiligen redundanzreduzierten (!) LR(0)-Maschine zur kfG-Repräsentation der betrachteten RRPG, wenn wir Itemmengen zu den besonderen lebensfähigen Präfizes der Form $\ldots q_i$ (die nur während der Reduktionsphasen eine Rolle spielen) beim Vergleich unberücksichtigt lassen. (Auf diese besonderen Itemmengen kommen wir noch zurück.) Abbildung 23 zeigt die redundanzreduzierte LR(0)-Maschine zur kfG-Repräsentation der Beispiel-RRPG.

Wenden wir uns nun jener dedizierten Hilfsmaschinerie zu, die wir in Anlehnung an Chapman (1984) und LaLonde (1979) schon als Readback-Automaten bezeichnet haben. Deren Aufgabe, durch Rückwärts-Lesen in den Parserkeller hinein das jeweilige linke Handleende zu bestimmen, skizziert Chapman wie folgt:

> „*LaLonde's parsing algorithm for RRPGs is based on adding* readback states *to the usual LR parser: these constitute a finite state machine which examines the states stacked during previous shift moves and accepts the sequences which correspond to handles. Thus, all the states examined during readback can be*

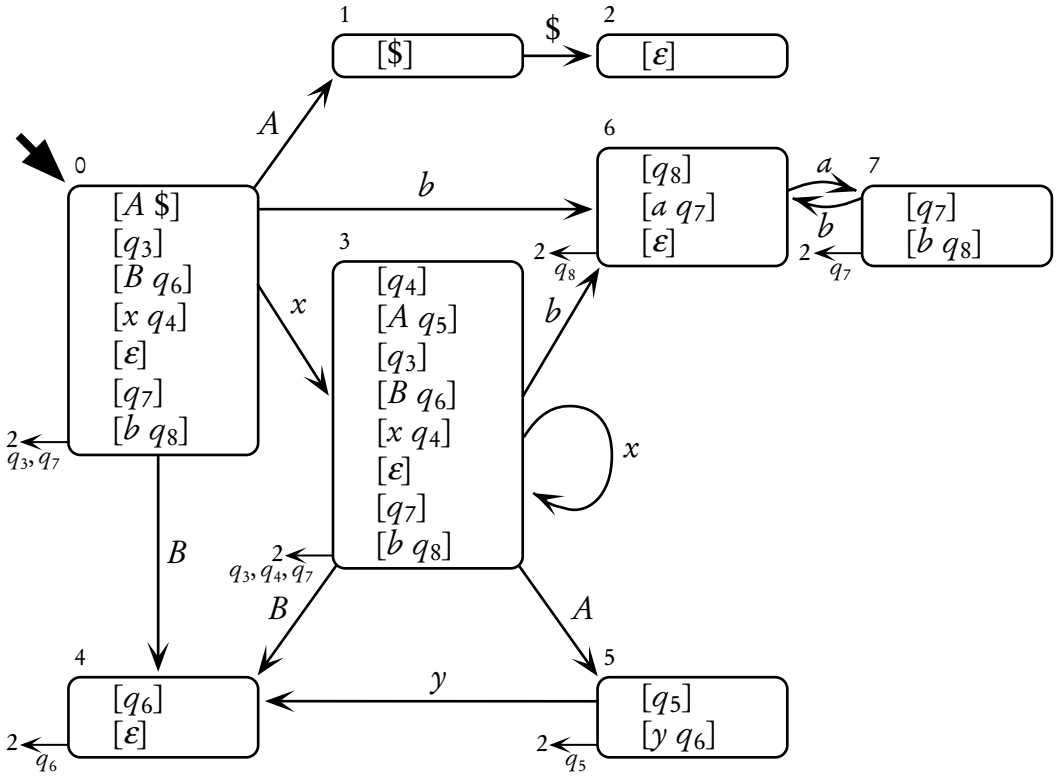


Abbildung 23: Die redundanzreduzierte LR(0)-Maschine zur Reformulierung der Beispiel-RRPG von Chapman als spezielle kfG ist im Kern strukturell identisch mit der von Chapman selbst konstruierten LR(0)-Maschine, die in Abbildung 22 gezeigt ist. Lediglich die besonderen Itemmengen zu lebensfähigen Präfizes $\ldots q_i$, die während der Reduktionen der komplexen Handles benötigt werden, sind in der hiesigen, theoretisch fundierten Konstruktion zusätzlich vorhanden und erlauben, eine entsprechende Steuerungs-Maschinerie für die komplexen Reduktionen systematisch abzulesen.

popped off to provide a correct reduction. Parsers which use k symbol lookahead and examine m stacked states to the left of the handle are known as LR(m,k) parsers.“

Die eigene Parsergenerierungstechnik nennt Chapman dann LALR(1,1).

Chapmans technische Umsetzung dieser Skizze betrachtet naheliegenderweise die Einbettungen der RRPA-Übergangsstrukturen in die Itemmengenkonstruktion, weswegen auch (Itemmenge, Item)-Paare zu den Zuständen der Readback-Automaten werden. Ein RRPA-Übergang $q_1 X \to q_2$, der sich wegen $q_1 \in \tilde{q}_1$ im Übergang $\tilde{q}_1 X \to \tilde{q}_2$ der LR(0)-Maschine wiederfindet, wird in der Readback-Maschinerie zum Übergang $(\tilde{q}_2, q_2)\tilde{q}_1 \to (\tilde{q}_1, q_1)$. Ein Paar $(\tilde{q}, q)$ wird Anfangszustand (auf den eine Reduktion an $\tilde{q}$ nach dem RRPA $\#j$ verweist), wenn q ein RRPA-Endzustand ist. $(\tilde{q}, q)$ wird Endzustand, wenn q ein RRPA-Startzustand ist und in $\tilde{q}$ tatsächlich mit der Handleerkennung begonnen wurde. Die entstehenden endlichen Automaten dieser Readback-Maschinerie fordert Chapman als deterministisch, und ein (noch um LALR(1)-Lookahead verfeinerter) Chapman'scher Parser liest bei einer Reduktion mit Hilfe eines ausgewählten solchen Readback-Automaten in die Itemmengenfolge auf dem Parserkeller hinein, um zu terminieren, sobald (!) ein Endzustand erreicht ist. Formalere Details können wir uns ersparen, wenn wir Abbildung 24 studieren, wo die beschriebenen Readback-Automaten zur Beispiel-RRPG aufgezeichnet sind.

Auch hier ist Chapmans Vorgehensweise etwas sorglos; einmal mehr jedoch müssen wir dies nicht nur ihm allein vorhalten, denn schon LaLonde (1979, S. 183) hatte zuvor die in Rede stehende Reduktions-Maschinerie mit endlichen Automaten gleichgesetzt:

„[...] construct an FA [finite automaton] which recognizes the reverse of the sequence of LR(k) items associated with the handle.“

Eindeutig greift dies zu kurz, denn bei genauerem Nachdenken bemerken wir die folgende Problematik: Es ist zwar richtig, daß diese umgekehrten Folgen von Itemmengen für alle möglichen Parserkellersituationen leicht durch Analyse der Einbettungen der RRPA-Übergangsstrukturen aus den Itemmengen extrahiert werden können. Auch ist offenkundig, daß sie reguläre Sprachen bilden und somit dedizierte DEAs konstruiert werden können, die genau diese Sprachen akzeptieren. Allerdings löst dies nicht das anstehende Problem, im Parserkeller das linke Handleende zu bestimmen, denn ein solcher Readback-DEA wäre lediglich in der Lage zu entscheiden, ob eine umgekehrte LR(k)-Itemfolge ein *komplettes*

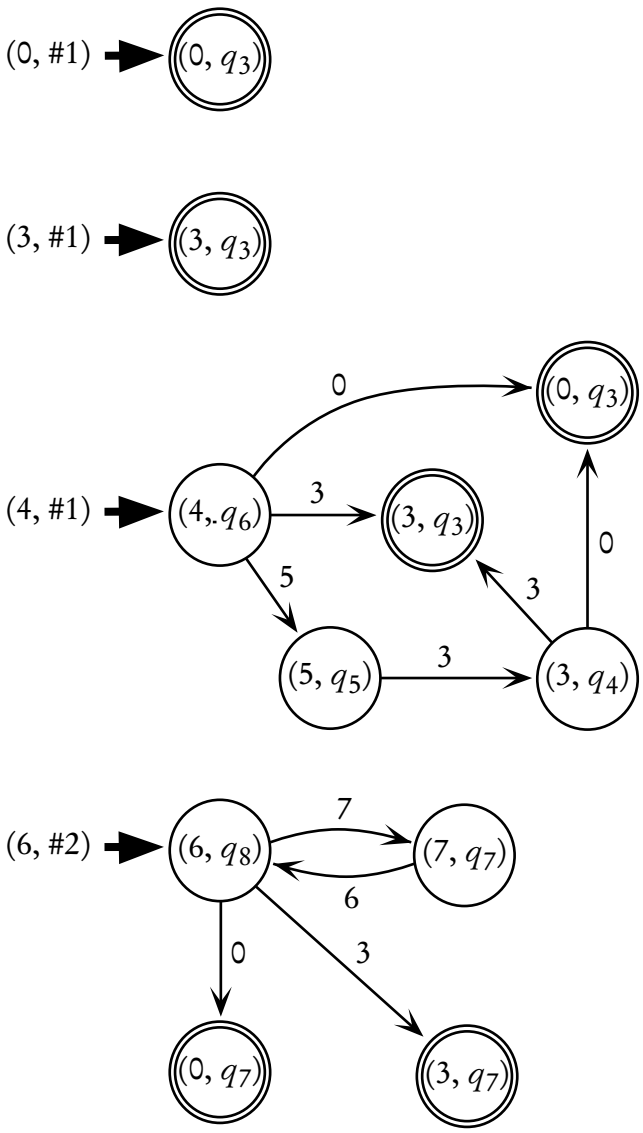


Abbildung 24: Die aus der Itemmengenkonstruktion in (Chapman 1984) abgelesenen Readback-Automaten sind reine endliche Automaten, die Chapman dann für seine LALR(1,1)-Parser als deterministisch einfordert. Die Abbildung zeigt diese Readback-Automaten für seine Beispiel-RRPG, welche wir im Text studieren. (Der in (Chapman 1984) gezeigte Readback-Automat für die Reduktion nach dem Startsymbol der erweiterten Grammatik ist überflüssig und hier nicht aufgeführt.)

Handle beschreibt oder, anders gesagt, das gerade im Keller vorliegende lebensfähige Präfix mit einem Handle *identisch* ist. Für die Aufgabe jedoch, soweit in eine umgekehrte LR(k)-Itemfolge *hinein* zu lesen, wie das überlesene *Anfangsstück* noch garantiert zum Handle gehört, ist ein herkömmlicher DEA so nicht angemessen. Und es ist überhaupt nicht klar, ob die Handleerkennungs-Problematik für *jede beliebige* ekfG, die im gemeinsamen Sinne von Heilbrunner (1979), LaLonde (1977, 1979), Chapman (1984) und Lee und Kim (1997) als ELR(k)-Grammatik angesehen wird,[50] überhaupt gelöst werden kann und mit einem Aufwand, der proportional zur Handlelänge ist. Tatsächlich kommt die angebotene theoretische Untermauerung seines Verfahrens bei Chapman (1984) nicht über Ansätze hinaus, und LaLonde (1979) geht über diesen Punkt mit dem arroganten Hinweis auf eine *„straightforward generalization of the Knuth technique"* gar gänzlich hinweg.

Wie aber gelangen wir nun zu einer *nachweislich korrekten* Readback-Maschinerie? Verfolgen wir dazu unseren eigenen, theoretisch fundierten Ansatz anhand des studierten Beispiels weiter, wobei wir uns mit Lookahead-Fragen erst später befassen wollen. Für unsere Beispiel-RRPG lesen wir aus Abbildung 23, ihrer redundanzreduzierten LR(0)-Maschine, folgende Shift-Aktionen des redundanzreduzierten LR(0)-Parsers ab, wobei wir wieder Zustände der LR(0)-Maschine gemäß der Abbildung zu Zahlen abgekürzt haben:

$$
\begin{aligned}
\mathit{shift}:\ & 0 \mid x \to 0\,x\,3 \mid ,\\
& 0 \mid b \to 0\,b\,6 \mid ,\\
& 3 \mid x \to 3\,x\,3 \mid ,\\
& 3 \mid b \to 3\,b\,6 \mid ,\\
& 5 \mid y \to 5\,y\,4 \mid ,\\
& 6 \mid a \to 6\,a\,7 \mid ,\\
& 7 \mid b \to 7\,b\,6 \mid .
\end{aligned}
$$

Die Reduce-Aktionen gliedern sich in die drei paarweise disjunkten Teilmengen $\mathit{reduce} = \mathit{red} \cup \mathit{rb} \cup \mathit{uce}$.[51] Aktionen aus *red* leiten dabei die Reduktionen der komplexen Handles

[50] Auf andere (teils eingeschränkte, teils fehlerhafte – siehe (Lee und Kim 1997)) Ansätze, namentlich die Arbeiten von Purdom und Brown (1981), Nakata und Sassa (1986) und Shin und Choe (1993), gehen wir erst an späterer Stelle ein.

[51] Die Namenswahl *red*, *rb* und *uce* ist durch (Chapman 1984) angeregt.

ein, indem sie den jeweils passenden RRPA-Endzustand sozusagen anwählen:

$$
\begin{aligned}
\mathit{red}: \quad & 0\rfloor \to 0\, q_3\, 2\rfloor\,, \\
& 3\rfloor \to 3\, q_3\, 2\rfloor\,, \\
& 4\rfloor \to 4\, q_6\, 2\rfloor\,, \\
& 6\rfloor \to 6\, q_8\, 2\rfloor\,.
\end{aligned}
$$

Die Aktionen aus *rb* beschreiben das Rückwärts-Durchlaufen eines angewählten RRPA und damit das eigentliche schrittweise Entfernen eines komplexen Handles vom Parserkeller:

$$
\begin{aligned}
\mathit{rb}: \quad & 0\, B\, 4\, q_6\, 2\rfloor \to 0\, q_3\, 2\rfloor\,, \\
& 0\, x\, 3\, q_4\, 2\rfloor \to 0\, q_3\, 2\rfloor\,, \\
& 0\, b\, 6\, q_8\, 2\rfloor \to 0\, q_7\, 2\rfloor\,, \\
& 3\, A\, 5\, q_5\, 2\rfloor \to 3\, q_4\, 2\rfloor\,, \\
& 3\, B\, 4\, q_6\, 2\rfloor \to 3\, q_3\, 2\rfloor\,, \\
& 3\, x\, 3\, q_4\, 2\rfloor \to 3\, q_3\, 2\rfloor\,, \\
& 3\, b\, 6\, q_8\, 2\rfloor \to 3\, q_7\, 2\rfloor\,, \\
& 5\, y\, 4\, q_6\, 2\rfloor \to 5\, q_5\, 2\rfloor\,, \\
& 6\, a\, 7\, q_7\, 2\rfloor \to 6\, q_8\, 2\rfloor\,, \\
& 7\, b\, 6\, q_8\, 2\rfloor \to 7\, q_7\, 2\rfloor\,.
\end{aligned}
$$

Die Aktionen aus *uce* schließlich stellen quasi die Ausstiege aus allen RRPA-Durchläufen an den Startzuständen dar und schließen die komplexen Reduktionen ab unter Kellerung des Nichtterminals auf der linken Seite der jeweiligen RRPA-Aktion:

$$
\begin{aligned}
\mathit{uce}: \quad & 0\, q_3\, 2\rfloor \to 0\, A\, 1\rfloor\,, \\
& 0\, q_7\, 2\rfloor \to 0\, B\, 4\rfloor\,, \\
& 3\, q_3\, 2\rfloor \to 3\, A\, 5\rfloor\,, \\
& 3\, q_7\, 2\rfloor \to 3\, B\, 4\rfloor\,.
\end{aligned}
$$

Die Einteilung von *reduce* in die Typen *red*, *rb* und *uce* ist dabei die natürliche Folge der regelmäßigen Struktur der Repräsentationsgrammatik, die die RRPG als kfG mit rechtslinearen Subgrammatiken ausdrückt.

Wir betrachten unsere Beispielaktionen *red*, *rb* und *uce* nun genauer. Augenfällige systematische Stütze in ihrem Zusammenwirken ist die von den *red*-Aktionen zuoberst gekellerte

Itemmenge (im Beispiel: 2), die, wenn man so will, einen Parsermodus „Reduktion begonnen“ ausdrückt: Dieselbe Itemmenge wird von allen *rb*- und *uce*-Aktionen zuoberst auf dem Keller erwartet; angewandte *rb*-Aktionen kellern dieselbe Itemmenge stets wieder zuoberst und belassen anschaulich den Parser im Modus „Reduktion begonnen“, während *uce*-Aktionen ihn wieder entkellern und den Parser zurück in den entgegengesetzten Zustand „Shift möglich“ schalten. Im allgemeinen können im Fall $k > 0$ in der jeweiligen redundanzreduzierten LR(k)-Maschine für unsere Repräsentations-Grammatiken mehrere solche Itemmengen existieren, die den Modus „Reduktion begonnen“ ausdrücken: Es sind dies genau die Itemmengen zu den lebensfähigen Präfizes der Form $\ldots q_i$, wobei q_i irgendein RRPA-Zustand ist. Diese Mengen enthalten grundsätzlich nur redundanzreduzierte Items der Art $[\varepsilon, x]$, im Falle $k = 0$ also nur das Item $[\varepsilon]$.

Sehen wir uns nun die *red*-Aktionen am Beispiel unserer Aktion $4\,| \rightarrow 4\,q_6\,2\,|$ näher an. In Prosa beschrieben, initiiert diese Aktion, falls der Parserkeller zuoberst die Itemmenge 4 enthält, eine komplexe Reduktion, indem sie zusätzlich zu dieser Itemmenge den RRPA-Zustand q_6 und den Modusindikator „Reduktion begonnen“ kellert. Mit der auf dem Parserkeller hinterlegten Information $4\,q_6$ „merkt“ sich der Parser unterhalb des Modusindikators, daß er nun beim schrittweisen Entfernen eines vollständigen komplexen Handles von hinten nach vorne beim RRPA-Endzustand q_6, eingebettet in die Itemmenge 4, aufzusetzen hat.

In der folgenden Readback-Phase, bestritten von den *rb*-Aktionen, wird die Einbettung des Rückwärts-Durchlaufs des handleaufspannenden RRPA (hier mit Endzustand q_6) in die Itemmengen verfolgt, während schrittweise das komplexe Handle auf dem Parserkeller entfernt wird. Dies geschieht allgemein, indem die von einer *red*-Aktion erzeugte Kombination aus „aktueller Itemmenge“, „aktuellem RRPA-Zustand“ und Modusindikator von den *rb*-Aktionen zuoberst auf dem Keller sowohl erwartet als auch systematisch fortgeschrieben wird. Beispielsweise deuten wir unsere *rb*-Aktion $3\,B\,4\,q_6\,2\,| \rightarrow 3\,q_3\,2\,|$, welche nach der obigen *red*-Aktion $4\,| \rightarrow 4\,q_6\,2\,|$ anwendbar sein könnte, in diesem Sinne so, daß die Identifizierung eines komplexen Handles, dessen RRPA-Einbettung bisher zum Zustand q_6 in Itemmenge 4 zurückverfolgt wurde, verlängert werden kann um das Grammatiksymbol B, falls darunter die Itemmenge 3 gekellert ist. (Die Kombination $3\,B$ bezeichnen wir in der Beispielaktion auch als den „Lookback“ der Aktion.) Neuer „Zustand“ der Handle-Rückverfolgung wird dann der „aktuelle RRPA-Zustand“ q_3, der in die dann „aktuelle Itemmenge“ 3 eingebettet ist, womit im passenden RRPA die Erzeugung von B rückverfolgt wird.

Nun könnte die *uce*-Aktion $3\,q_3\,2\,| \rightarrow 3\,A\,5\,|$ anwendbar sein. (Tatsächlich ist q_2 Startzu-

stand des RRPA zu A, und in Itemmenge 3 beginnen komplexe Handles zu A.) Ist also in der Readback-Phase der „Zustand“ der Handle-Rückverfolgung der RRPA-Zustand q_3, eingebettet in Itemmenge 3, so kann (!) die Readback-Phase enden, und A und die mit A aus Itemmenge 3 erreichte Itemmenge 5 wird gekellert. Insbesondere wird damit der Modusindikator, die Itemmenge 2, entfernt, und für den Parser ist wieder ein „Shift möglich“.

Abbildung 25 zeigt ein etwas umfangreicheres Beispiel, das insbesondere unsere an Mustern „Lookback“, „Zustand“ und „Modusindikator“ orientierte Beschreibung der Parseraktionen nocheinmal unterstreichen soll. Nach Studium dieser Veranschaulichung der Readback-Aktivitäten des Parsers sollte nun nicht mehr schwerfallen nachzuvollziehen, welche einzelnen Beiträge die Reduce-Aktionen zu ihrer Deutung als Familie von Readback-DEAs[52] leisten. Abbildung 26 stellt für die Reduce-Aktionen unseres Beispiels die Deutung als Readback-DEAs graphisch dar, wobei nocheinmal darauf hingewiesen sei, daß wir bisher die Betrachtung von irgendwelchem Lookahead ($k > 0$) ausgespart haben.

Strukturell ist das Ergebnis wohl derjenigen Readback-Maschine recht ähnlich, zu der Chapman durch einen Arbeitsschritt gelangt, den er als *„merging readback machines“* bezeichnet (aber nicht weiter erklärt). Unterschiede liegen jedoch im Detail. Bevor wir uns mit ihnen beschäftigen, sollten wir auch noch beleuchten, wie sich die Berücksichtigung von Lookahead in der Readback-Maschinerie auswirkt.

Zum Zwecke des direkten Vergleichs interessieren wir uns insbesondere für einen LALR(1)-Lookahead, und auch hier erweisen sich unsere Vorarbeiten im ersten Teil der vorliegenden Arbeit als sehr nützlich: Wir wissen, wie ein redundanzreduzierter LALR(1)-Parser konstruiert wird! Die um LALR(1)-Lookahead angereicherten Reduce-Aktionen unseres Beispielparsers sind in Abbildung 27 sogleich als Readback-Automaten aufgezeichnet.

In der Abbildung ist auffällig, daß offensichtlich ganze Gruppen von *rb*- und *uce*-Aktionen, die jeweils einen zusammenhängenden Readback-Automaten ergeben, für unterschiedliche Lookaheads komplett verdoppelt vorkommen. Diese Automatenduplikate sind tatsächlich systematische Folge der Art und Weise, in der im LALR-Verfahren die Aktionen der (hier) redundanzreduzierten LR(0)-Maschine um Lookahead angereichert werden.[53] Wir werden dies nicht vertiefen, bemerken jedoch, daß mit dieser Beobachtung klar ist, daß in einem redundanzreduzierten LALR(k)-Parser in allen *rb*- und *uce*-Aktionen auf die Lookahead-Anteile verzichtet werden kann und dennoch der Parser sich noch immer stark

[52] Anders als die üblichen DEAs können diese Readback-DEAs i. a. mehrere mögliche Startzustände haben und werden gezielt in einem davon gestartet.

[53] Siehe Abschnitt 8.2 der vorliegenden Arbeit.

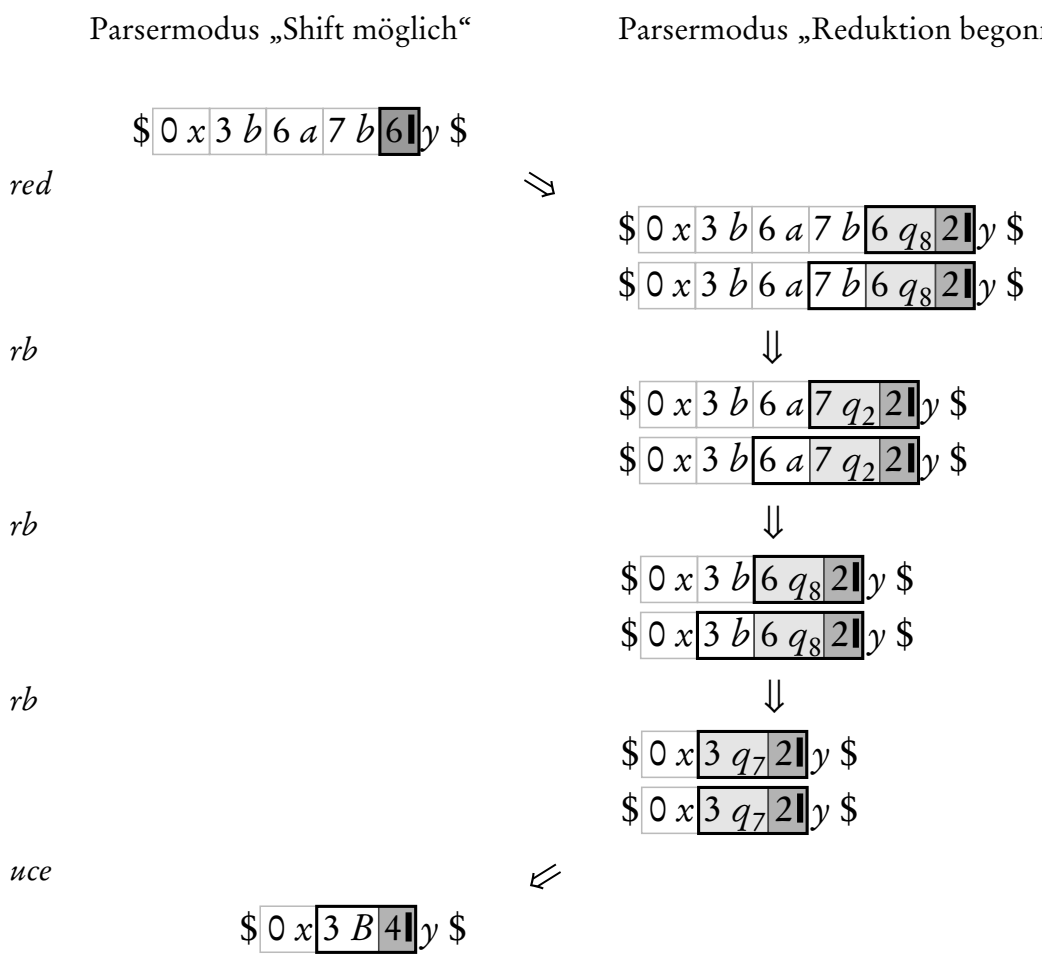


Abbildung 25: Gezeigt ist eine beispielhafte zusammenhängende Folge von Parseraktionen, welche der von uns im Text konstruierte redundanzreduzierte LR(0)-Parser (siehe Abbildung 23 und Seite 139) ausführen kann. Verdeutlicht werden soll, wie der Parser zum lebensfähigen Präfix $x\,b\,a\,b$ in Einzelschritten die Reduktion des komplexen Handles $b\,a\,b$ zum Nichtterminal B abwickelt im Stile eines eingebetteten, rückwärts gerichteten DEAs, welcher seinen Zustand (hellgrau unterlegt) stets unterhalb des Indikators (dunkelgrau unterlegt) für den Parsermodus „Reduktion begonnen“ im obersten Kellerabschnitt hält. Der zu Beginn angewählte Readback-DEA durchläuft die Zustände $6q_8$ (Startzustand), $7q_2$, $6q_8$ bis $3q_7$ (Endzustand), liest dabei die Lookbacks $7b$, $6a$ und $3b$ und hinterläßt mit Beendigung des Readback-DEAs zuoberst auf dem Keller die Folge 3 B 4, womit die linke Seite zum reduzierten komplexen Handle gekellert wird. Anschaulich verfolgt damit der Readback-DEA die Einbettung des handleerzeugenden RRPA in die Itemmengenkonstruktion rückwärts.

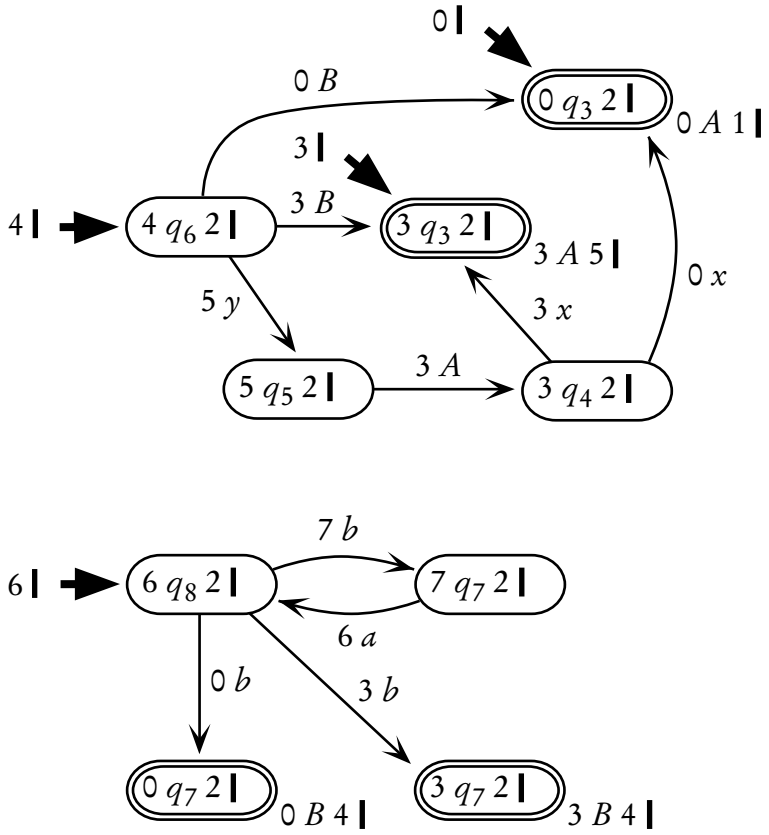


Abbildung 26: Die als Readback-DEAs interpretierten Reduce-Aktionen des redundanzreduzierten LR(0)-Parsers für die Reformulierung von Chapmans Beispiel-RRPG als spezielle kfG. Der mit Termination eines Readback-DEA herzustellende oberste Kellerinhalt ist jeweils rechts neben den Endzuständen angegeben.

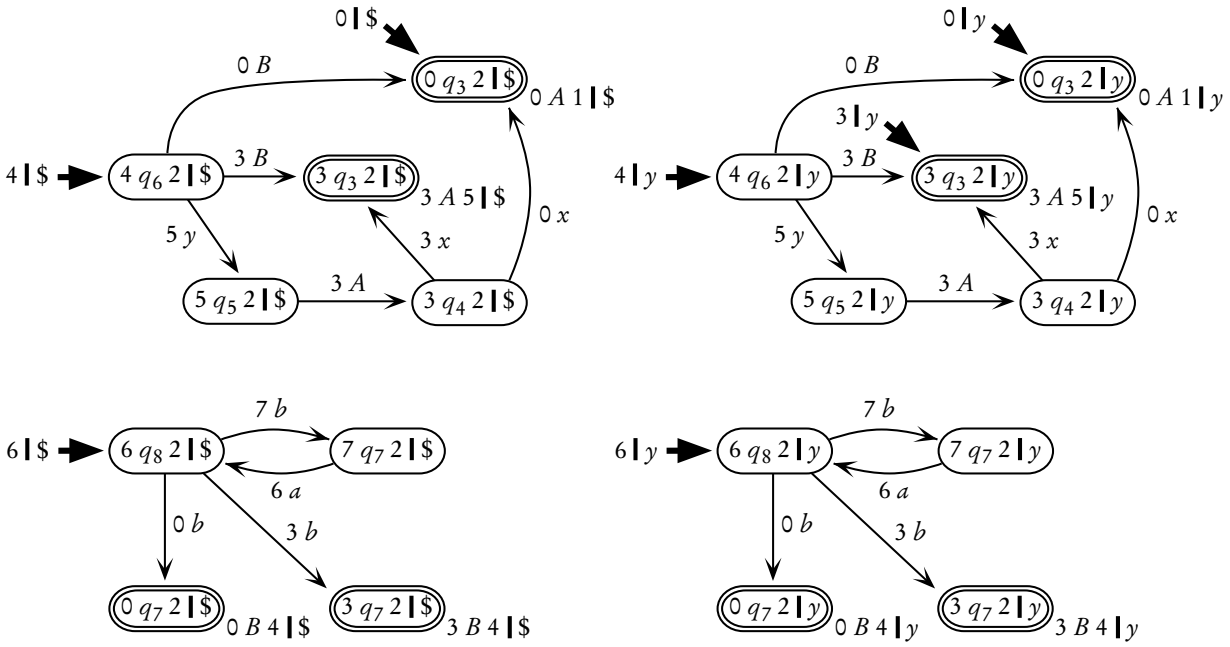


Abbildung 27: Wie Abbildung 26, jedoch für den redundanzreduzierten LALR(1)-Parser. Das vielfache Auftreten vollständiger einzelner Readback-DEAs für verschiedene Lookaheads ist systematisch in der Art und Weise begründet, wie im LALR-Verfahren die redundanzreduzierte LR(0)-Maschine mit Lookahead angereichert wird. (Siehe Abschnitt 8.2.)
Zum Vergleich sei auch auf Abbildung 29 verwiesen, wo die Readback-DEAs aufgeführt sind, die sich für den redundanzreduzierten ILALR(1)-Parser ergeben.

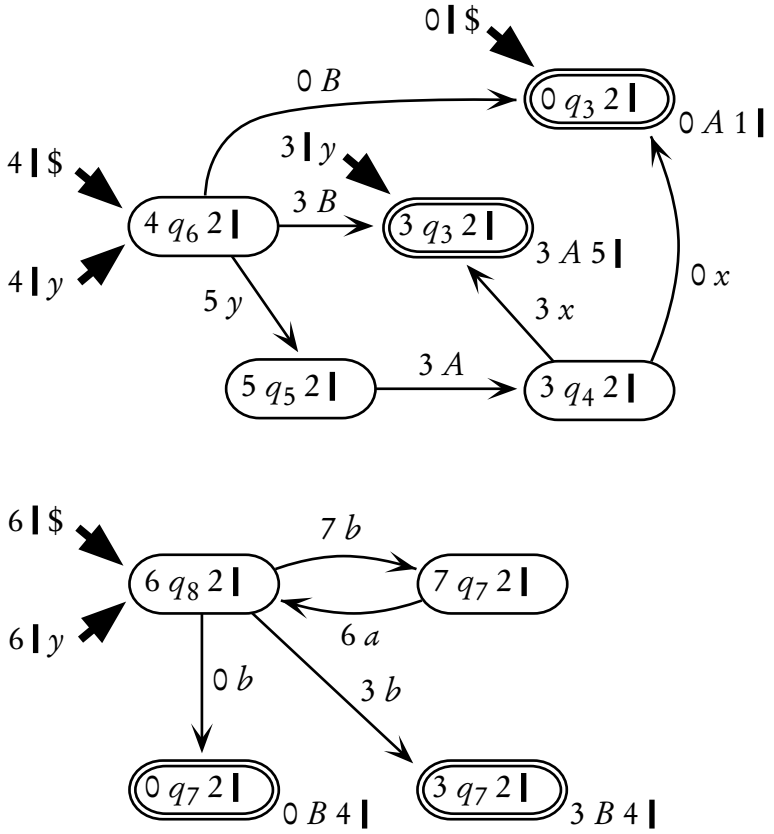


Abbildung 28: Wie im Text erläutert, können in den *rb*- und *uce*-Aktionen unserer redundanzreduzierten LALR(k)-Parser für ekfGen die Lookahead-Anteile gestrichen werden, was anschaulich der Zusammenlegung verdoppelter Readback-DEAs entspricht. Der entstehende Parser verhält sich stark wie der Parser mit noch vollen Lookahead-Anteilen.
Gezeigt sind die Readback-DEAs aus Abbildung 27 nach dieser Zusammenlegung.
Der entstandene Parser ist das Endergebnis unserer Rekonstruktion von Chapmans beispielhaft konstruiertem LALR(1,1)-Parser. (Siehe Abbildung 24.)

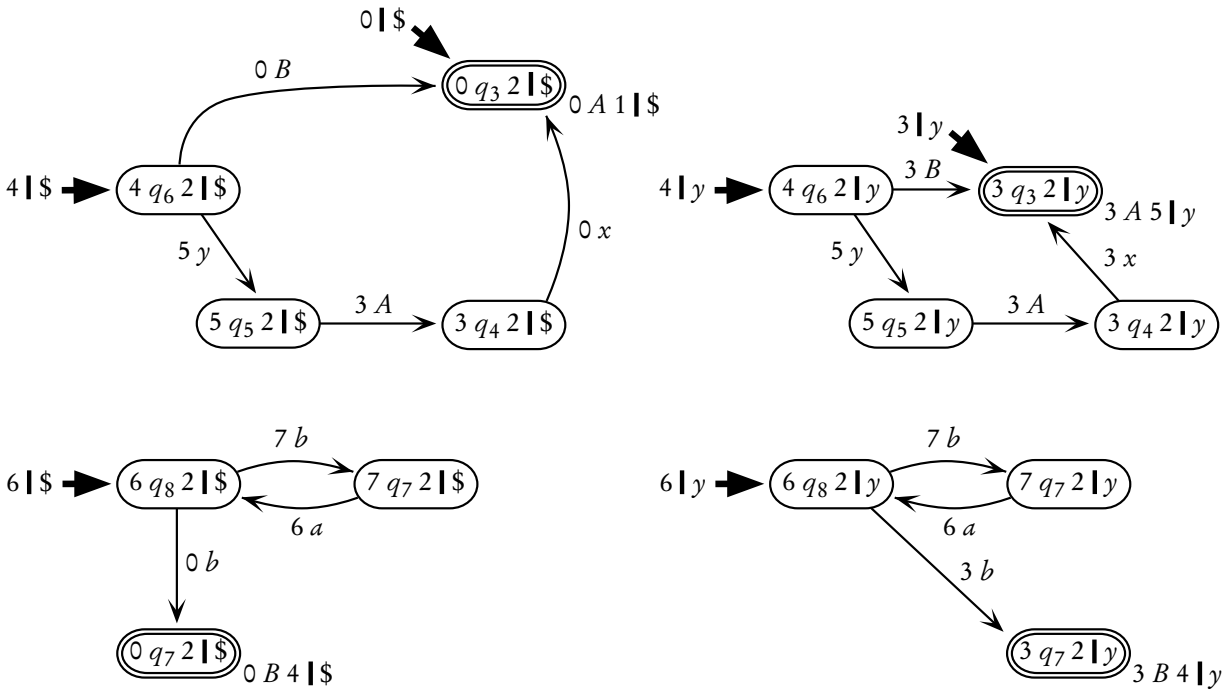


Abbildung 29: Wie Abbildung 27, jedoch für den redundanzreduzierten ILALR(1)-Parser. Der Vergleich mit Abbildung 27 veranschaulicht – hier für ekfGen – nocheinmal, daß die ILALR(k)-Methode seltener nichtdeterministische Parser hervorbringt als die LALR(k)-Methode (für $k > 0$): Der in Rede stehende ILALR(1)-Parser weist nur einen Teil der Reduce-Aktionen des vergleichbaren LALR(1)-Parsers aus; entsprechend weisen seine Readback-DEAs weniger Zustände und Übergänge auf als ihre LALR(1)-Pendants.

gleich verhält.[54] Anschaulich können also die Duplikate von Readback-Automaten aufeinandergelegt werden und in den „Ausgaben“ an den Endzuständen (also den rechten Seiten von *uce*-Aktionen) die Lookahead-Anteile gestrichen werden. Verschiedene Startzustandselektionen (also linke Seiten von *red*-Aktionen) verweisen auf den nach Zusammenlegung gemeinsamen Startzustand (also die gemeinsame rechte Seite derselben *red*-Aktionen nach Streichen des Lookahead-Anteils). Das Ergebnis dieser Zusammenlegung zeigt Abildung 28.

[54] Hier sei zum ersten darauf hingewiesen, daß *shift*-Aktionen nur mit *red*-Aktionen, nicht jedoch mit *rb*- und *uce*-Aktionen einen Nichtdeterminismus aufweisen können. Zum zweiten sei angemerkt, daß die geschilderten Beobachtungen natürlich auch auf kanonische LALR(k)-Parser zutreffen.

Interessant ist an dieser Stelle ein kurzer Blick auf den redundanzreduzierten ILALR(1)-Parser und dessen Readback-Maschinerie, die in Abbildung 29 gezeichnet ist. Beim Vergleich mit dem LALR(1)-Pendant aus Abbildung 27 fällt sofort auf, daß die im LALR(1)-Parser systematisch vorkommenden verdoppelten Readback-Automaten für verschiedene Lookaheads, die uns erlaubten, solche Automaten-Duplikate aufeinanderzuschieben, im ILALR(1)-Fall nicht mehr zu sehen sind: Im allgemeinen treten, abhängig vom hinzugefügten Lookahead, nur noch zusammenhängende Ausschnitte der LR(0)-Readback-Automaten als ILALR(1)-Readback-Automaten auf. Entsprechendes gilt auch verallgemeinert für $k > 0$ und ist Folge der Tatsache, daß – wie die vorliegende Arbeit bereits gezeigt hat – das ILALR(k)-Verfahren dazu neigt, schärfere Aktionenmengen und seltener Nichtdeterminismus zu produzieren als das LALR(k)-Verfahren.

Doch kommen wir nun dazu, Unterschiede zwischen Chapmans LALR(1,1)- und unserer eigenen LALR(1)-Readback-Maschinerie (nach Streichen der Lookahead-Anteile in *rb*- und *uce*-Aktionen) bzw. zwischen den entsprechenden Parsern zu diskutieren.

Fehlende Ausgrenzung von *rb*/*uce*-Konflikten

Konstruieren wir auf dem von uns präferierten Weg einen redundanzreduzierten LR-Parser für eine gegebene ekfG, so lassen sich, wie wir wissen, die Parseraktionen anhand ihrer Rollen in die disjunkten Aktionenklassen *shift*, *red*, *rb* und *uce* zerlegen. Wie wir teilweise ebenfalls bereits erwähnt haben, können zwischen diesen 4 Klassen nun nicht alle 10 prinzipiell denkbaren, sondern tatsächlich nur 4 verschiedene Typen von Nichtdeterminismen vorkommen, nämlich *shift*/*red*-, *red*/*red*-, *rb*/*rb*- und *rb*/*uce*-Konflikte.

Nicht mit allen diesen Konflikttypen geht Chapman korrekt um: Er nennt eine RRPG genau dann eine LALR(1,1)-Grammatik, wenn der seiner LR(0)-Maschine hinzugefügte LALR(1)-Lookahead zu keinen Konflikten führt und seine Readback-Automaten deterministisch sind. Ersteres schließt Konflikte der Typen *shift*/*red* und *red*/*red* aus, letzteres solche vom Typ *rb*/*rb*.

Die Möglichkeit von *rb*/*uce*-Konflikten jedoch scheint Chapman überhaupt nicht bedacht zu haben. Konflikte dieses Typs bedeuten, daß Parsersituationen möglich sind, in denen bei identischem Lookahead der Parser mit einer $red \cdot rb^*$-Aktionenfolge Grammatiksymbole entkellert hat, die sowohl ein vollständiges wie auch ein unvollständiges komplexes Handle zur selben linken Seite darstellen. Der dazu selektierte Readback-Automat kann nicht entscheiden, ob er anhalten soll (*uce*) oder nicht (*rb*). Offensichtlich hat Chapman dies

übersehen bei seiner Behauptung, der von ihm formal skizzierte Parser *„can easily be re-cast to give a DPDA [deterministic pushdown automaton] of the familiar kind“*.

Fehlende Ausgrenzung bestimmter *red/red*-Konflikte

Chapman (1984) betrachtet in seiner Arbeit die bekannte graphalgorithmische Technik von DeRemer und Pennello (1982) zur effizienten Berechnung der LALR(1)-Lookahead-Anreicherungen von LR(0)-Reduce-Aktionen aus der Struktur einer kanonischen LR(0)-Maschine zu einer kfG und überträgt die Idee des Algorithmus auf die von ihm geschaffenen Verhältnisse bei RRPGen. Obwohl es in seiner Arbeit an theoretischem Fundament fehlt, ist dennoch diese Übertragung in ihrem algorithmischen Kern korrekt. Schwierigkeiten kündigen sich jedoch im letzten Schritt der Lookahead-Mengen-Berechnung an, wo Chapman sogenannte Follow-Mengen[55] zu den endgültigen Lookahead-Mengen vereinigt: Er berechnet Lookahead-Mengen je zuoberst gekellerter Itemmenge und RRPG-*Regel.* Auf dieser Grundlage dann in seiner Definition der Klasse von LALR(1,1)-RRPGen einzufordern, daß *„parsing conflicts in inadequate LR(0) states can be resolved ...“*, kann nur so verstanden werden, daß es Chapman genügt, bei der Einleitung von komplexen Reduktionen die handleaufspannende RRPG-*Regel* eindeutig über das Lookahead selektieren zu können.

Dagegen haben wir selbst, indem wir auf den vorangehenden Seiten RRPGen als spezielle kfGen modelliert haben, gesehen, daß über die *red*-Aktionen des Parsers nicht eine RRPG-*Regel,* sondern präziser ein RRPA-*Endzustand* deterministisch anzuwählen ist. Beispielsweise kann ein LALR(1)-Parser für die RRPG mit den Regeln

bei lebensfähigem Präfix $a\,b$ und Lookahead \$ offenkundig nicht entscheiden, ob für die zu beginnende komplexe Reduktion anschaulich q_6 oder q_7 anzuwählen ist. Dennoch ist in derselben Situation eindeutig, daß nach Regel #2 zu reduzieren ist, und die RRPG ist nach Chapmans Definition, so wie wir diese verstehen, eine LALR(1,1)-Grammatik.

[55] Siehe Abschnitt 8.3 der vorliegenden Arbeit.

Nicht gekellerte Grammatiksymbole

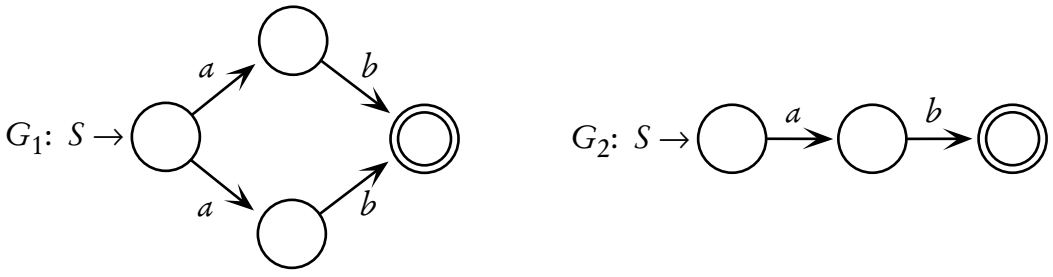
Ein auffälliger Unterschied zwischen dem vorgestellten Beispielparser von Chapman und dem Ergebnis unseres eigenen Ansatzes ist, daß Chapmans Parser die Grammatiksymbole des lebensfähigen Präfix' nicht mit auf dem Parserkeller ablegt. Die Konsequenzen dieses Unterschieds scheinen allerdings eher gering zu sein: Abweichungen aufgrund dessen bezüglich akzeptierter Sprache und Determinismuseigenschaft konnten nicht festgestellt werden. Wir wollen jedoch festhalten, daß ein deterministischer Chapman'scher Parser mit einer komplexen Reduktion „nur" die *Zustands*folge rekonstruieren könnte, die ein RRPA für das komplexe Handle durchläuft. Unser eigenes Verfahren hingegen rekonstruiert in der analogen Situation die entsprechende *Übergangs*folge des RRPA, mit der, wie wir meinen, semantische Aktionen sinnvoller verbunden werden können. Und natürlich ist allgemein nur der Schluß von der Übergangs- auf die Zustandsfolge möglich, aber nicht umgekehrt.

Wir erinnern uns hier aber auch daran, daß Chapmans Definition einer ELR(k)-Grammatik davon abstrahiert, *wie* in einem Ableitungsschritt die für die linke Seite einer RRPG-Regel eingesetzte Grammatiksymbolfolge mit Hilfe des RRPA der rechten Seite entsteht. Seine Parser beanspruchen daher auch nicht, feinere Strukturinformation als nur eine umgekehrte Rechtsableitung, bestehend aus erkannten RRPG-*Regeln,* zu liefern. Darauf kommen wir auch im folgenden Abschnitt zurück.

Eine grundsätzliche Kritik an Chapmans Verfahren

Den Vergleich von Chapmans direktem LALR(1,1)-Parserkonstruktionsverfahren aus einer RRPG und unserem eigenen Vorschlag, einen redundanzreduzierten LALR(1)-Parser indirekt zu erstellen über die Simulation der RRPG durch eine spezielle kfG, wollen wir nun in den Hintergrund stellen. Statt dessen wollen wir im folgenden noch zwei grundsätzliche Punkte an Chapmans Verfahren kritisieren.

Zur Verdeutlichung unseres ersten Kritikpunkts betrachten wir die beiden RRPGen G_1 und G_2 mit jeweils der folgenden einzigen Regel:



Diese beiden RRPGen sind ein triviales Beispiel für zwei reduzierte RRPGen mit gleichen Alphabeten und Startsymbolen, die in dem Sinne äquivalent sind, daß die von Chapman (und anderen Autoren) definierte Rechtsableitungsrelation, welche nur auf die aufgespannten Sprachen der RRPAs, nicht jedoch deren Struktur Bezug nimmt, exakt dieselbe ist. Derart äquivalente RRPGen generieren natürlich gleiche Sprachen, und insbesondere ist nach Chapmans (und anderer Autoren) Definition für jedes $k > 0$ stets die eine RRPG genau dann ELR(k), wenn dies auch die andere ist. Die obigen RRPGen G_1 und G_2 sind beide ELR(0)-Grammatiken im Sinne von Chapman.

Wie aber verhält es sich mit der Chapman'schen LALR(1,1)-Eigenschaft von G_1 und G_2? G_1 weist, anders als G_2, einen mehrdeutigen RRPA auf, was zu einem nichtdeterministischen Readback-Automaten für G_1 führt. G_1 ist somit nach Chapmans Definition keine LALR(1,1)-Grammatik, G_2 aber sehr wohl.

Diese Diskrepanz ist in zweierlei Hinsicht unbefriedigend. Zum einen bedeutet sie einen Bruch gegenüber der bekannten Grammatikhierarchiebeziehung LR(0) $\subset$ LALR(1) auf seiten der kfGen. Und zum zweiten ist ganz offensichtlich die Determinismuseigenschaft eines LALR(1,1)-Parsers abhängig von der Struktur der beteiligten RRPAs, während Chapmans Rechtsableitungs- und ELR(k)-Begriff von ihr aber gerade abstrahieren.

Unser zweiter Kritikpunkt betrifft die Frage, ob denn die Klassen der Chapman'schen LALR(m,k)-Grammatiken für festgehaltenes k überhaupt eine echte Hierarchie bilden, oder, anders ausgedrückt, ob es überhaupt Sinn macht, wie Chapman eine parametrische Länge m des Lookbacks zu betrachten. Unsere Antwort lautet: nein.

Tatsächlich mußte in unserer theoretisch fundierten Rekonstruktion des Chapman'schen Verfahrens mit rein kontextfreien Mitteln ein solcher Parameter für die Lookback-Länge m gar nicht erst eingeführt werden, sondern der von Chapman untersuchte Spezialfall $m = 1$ ergab sich bei uns schlicht aus der „natürlichen" Struktur unserer *rb*-Aktionen; und hier für die Realisierung einer Lookback-Länge von $m > 1$ entsprechend tieferen Kellerkontext in die *rb*- und die *uce*-Aktionen einzubeziehen kann, wie wir wissen, gegenüber dem redundanzreduzierten LALR(k)-Parser keinen etwaigen Nichtdeterminismus beseitigen. Eine von Chapman vermutete echte Hierarchie von LALR(m,k)-Grammatiken für festgehaltenes $k \geq 0$ und wachsendes $m > 0$ existiert nicht; der Fall $m = 0$ ist nur von theoretischem Belang.[56]

[56] Siehe auch (LaLonde 1977).

9.6 Erweiterte kontextfreie Grammatiken in weiterer Literatur

An die im vorangehenden Abschnitt vorgezogene Untersuchung von (Chapman 1984) schließen wir nun zunächst in den Abschnitten 9.6.1–9.6.6 die sonstigen Arbeiten auf der Basis von RRPGen in der Reihenfolge des Veröffentlichungsjahrs an und befassen uns dann in Abschnitt 9.6.7 noch mit der einzigen ekfG-basierten Arbeit.

9.6.1 LaLonde (1977, 1979)

Mit zu den ersten Autoren, die sich mit der Konstruktion von LR-Parsern für ekfGen befaßt haben, gehört Wilf LaLonde. Wir konzentrieren uns in der Diskussion auf seine wichtigsten Beiträge zum Thema: die beiden Journalbeiträge (LaLonde 1977) und (LaLonde 1979), die die wesentlichen Ergebnisse seiner Dissertation (LaLonde 1975) präsentieren.

Gegenstand der Untersuchungen darin sind RRPGen, auf die der Autor die bekannte, auf Knuth (1965) zurückgehende LR(k)-Parserkonstruktionstechnik für kfGen übertragen will. Der frühere der beiden Journalbeiträge beschreibt im Kern zunächst RRPGen und einen ELR(k)-Begriff dafür und stellt sodann einen tabelleninterpretierenden Syntaxanalysealgorithmus – dort „LR(m,k)-Parser“ genannt – vor. LaLondes ELR(k)-Begriff stimmt dabei mit dem bei Chapman (1984) und Heilbrunner (1981) gesehenen überein, abstrahiert also von den Strukturen der RRPAs. (LaLonde 1977) ist damit die früheste Journalveröffentlichung, die *diese* Auffassung vom ELR(k)-Begriff präsentiert. Die Bedeutung des Parameters m ist uns bereits bekannt: Chapmans (1984) und LaLondes (1979) Itemmengenkonstruktion ist dieselbe, und wie später Chapman (1984) verwendet bereits LaLonde (1977, 1979) Readback-Automaten zur Bestimmung des linken Handleendes im Keller. Grammatiksymbole werden ebenfalls nicht gekellert, und die Readback-Automaten verwenden insofern ebenso einen Lookback von m Itemmengen in den Parserkeller hinein. LaLondes (1979) Technik zur Konstruktion dieser Readback-Automaten ist allerdings nicht mit der von Chapman (1984) identisch. Vielmehr scheint die undurchsichtige Konstruktion von LaLonde (1979) im Kern darauf hinauszulaufen, daß anstatt der möglicherweise nichtdeterministischen Readback-Automaten, die Chapman aus den Itemmengen abliest, erst gewisse Potenzkonstruktionen davon für die Bestimmung des linken Handleendes eingesetzt werden.

Kritisieren müssen wir an beiden genannten LaLonde'schen Arbeiten zuallererst das Theorievakuum, das sie darstellen: Sie erschöpfen sich in Definitionen, Konstruktionsvorschriften und Beispielen. Das völlige Fehlen von Theorie vernebelt LaLonde (1979) durch

Formalismusdickicht und überhebliche Beruhigungen (*„straightforward generalization of the Knuth technique“*).

Der Autor der vorliegenden Dissertation will sich mit Spekulationen über die mögliche Korrektheit der LaLonde'schen Parserkonstruktionstechnik zurückhalten. Die entsprechenden Nachweise hätten natürlich von LaLonde selber erbracht werden müssen. Einen Ansatz zu notwendigen Beweisen lieferten jüngst Lee und Kim (1997); noch immer offen ist jedoch mindestens der Nachweis der korrekten Funktionsweise der Readback-Maschinerie für die Abwicklung der komplexen Reduktionen.

Wir weisen an dieser Stelle darauf hin, daß es mit Kenntnis der von uns vorgestellten Ergebnisse von Heilbrunner (1979), offensichtlich ist, wie wir zu einer gegebenen RRPG auf einfache Weise einen im Sinne von LaLonde (1977), Heilbrunner (1979), Lee und Kim (1997) u. a. korrekten ELR(k)-Parser konstruieren können: Wir ersetzen alle RRPAs durch eindeutige (z. B. deterministische) Pendants, betrachten dann die Umformulierung zu einer reinen kfG mit rechtslinearen Subgrammatiken und konstruieren schließlich einen klassischen kanonischen (oder auch einen allgemeinen) LR(k)-Parser. Dieser zeigt das gewünschte Verhalten und ist genau dann deterministisch, wenn die RRPG eine ELR(k)-Grammatik im genannten Sinne ist.

Wir können ferner eine Lehre, die wir aus der ausführlichen Diskussion der ELALR-spezifischen Parserkonstruktionstechnik von Chapman (1984) gezogen haben, sofort auch auf die Konstruktion von ELR-Parsern übertragen: Wir bemerkten nämlich, daß mit festgehaltenem Lookahead k eine echte Hierarchie der Chapman'schen LALR(m,k)-RRPGen für Lookback $m > 0$ gar nicht existiert. Und die Überlegung, die uns zu dieser Einsicht führte, gilt auch für die LaLonde'schen LR(m,k)-RRPGen. Die Klasse der LR($1,k$)-RRPGen ist folglich identisch mit der der ELR(k)-RRPGen nach LaLonde (1977) u. a. !

Und schließlich wollen wir auch noch darauf aufmerksam machen, wie offenkundig wenig durchdacht die Vorschläge von LaLonde (1979) zur Anbindung von Semantik an die von ihm konstruierten Parser sind. So betont er zum einen, daß sein Begriff von ELR(k)-RRPGen, bei dem ja von der Struktur der RRPAs abstrahiert wird, allgemeiner ist, als wenn auf rechten Regelseiten Eindeutigkeit gefordert wird. Zum anderen schlägt er vor, semantische Aktionen mit den RRPA-Übergängen zu verbinden, was für mehrdeutige RRPAs offensichtlich für die Übersetzerbaupraxis unbrauchbar ist. Auch seine Idee, Routinen mit Shift-Aktionen des Parsers zu koppeln, ist natürlich problematisch.

Wir ersparen uns eine detailliertere Kritik.

9.6.2 Purdom und Brown (1981)

Im Gegensatz zu den soeben diskutierten Arbeiten von LaLonde (1977, 1979) zeichnet sich der Journalbeitrag von Purdom und Brown (1981) durch die Eingänglichkeit und Klarheit der Präsentation aus und durch das Fehlen überheblicher Formulierungen. Die Autoren beschäftigen sich im Prinzip[57] nur mit RRPGen, deren RRPAs sie als deterministisch fordern. Ihr Begriff einer ELR(k)-Grammatik stimmt jedoch ansonsten mit demjenigen überein, den wir von LaLonde (1977), Chapman (1984) und Heilbrunner (1979) kennen, abstrahiert also von der Struktur der RRPAs.[58]

In ihrem Umgang mit diesem Begriff sind die Autoren recht unsensibel: Auf mögliche Einschränkungen, die sich aus ihrer Forderung nach deterministischen RRPAs ergeben, gehen sie nicht ein, sondern vergleichen ihren eigenen und die Ansätze von Madsen und Kristensen (1976), LaLonde (1977, 1979) und Heilbrunner (1979), als behandelten alle Autoren denselben ELR(k)-Begriff. Besonders die Arbeit von Madsen und Kristensen (1976), in der reguläre Ausdrücke auf rechten Regelseiten eingesetzt werden und *nicht* von der Struktur der rechten Regelseiten abstrahiert wird, schätzen Purdom und Brown (1981) falsch ein. Auch der Beitrag von Heilbrunner (1979) scheint nicht wirklich von ihnen verstanden worden zu sein.

Die Kernidee ihres Verfahrens ist, den Parser die „aktuelle" Itemmenge überhaupt nur dann kellern zu lassen, wenn beim Übergang zu einer Folge-Itemmenge das Übergangssymbol eindeutig den Anfang eines neuen komplexen Handles darstellt. (Das leere Wort als Handle muß gesondert behandelt werden.) Grammatiksymbole werden nicht gekellert. Die Implementierung der Reduktionen gerät damit sehr simpel (und zudem laufzeiteffizient[59]), da die Anzahl zu entkellernder Itemmengen nicht mehr variabel ist. Die von den Autoren verwendete Itemmengenkonstruktion kennen wir von der ausführlichen Diskussion von (Chapman 1984).

Natürlich existieren RRPGen mit ELR(k)-Eigenschaft, für die so nicht ohne weiteres ein Parser konstruiert werden kann, da für sie an mindestens einem Übergang der Item-

[57] Purdom und Brown (1981) führen zwar zunächst noch ein, was wir als eine ekfG bezeichnen, wechseln jedoch sofort hinüber zu dem, was wir eine RRPG nennen: *„An* extended context free grammar *is a context free grammar in which the right sides of productions are regular expressions over the terminal and nonterminal symbols of the grammar. (We represent the regular expressions by deterministic finite state machines.)"*

[58] Die „Definition" ihrer Ableitungsrelation muß dabei allerdings im Kopf des Lesers erfolgen!

[59] Tatsächlich handelt es sich um die Übertragung einer Optimierung von DeRemer (1969) für klassische LR-Parser auf die Verhältnisse bei RRPGen.

mengenkonstruktion nicht eindeutig ist, ob damit ein neues komplexes Handle begonnen wird oder nicht. In diesem Fall transformieren Purdom und Brown (1981) gezielt die Ausgangs-RRPG, um den *„stacking conflict"* zu beheben. Wie sie beweisen (mit Verweis auf Heilbrunner (1979)), geht dabei die etwaige ELR(*k*)-Eigenschaft der RRPG nicht verloren.

Die Möglichkeit der sinnvollen Anbindung semantischer Aktionen muß bei (Purdom und Brown 1981) etwas weniger kritisch beurteilt werden als bei den meisten anderen Arbeiten zum Thema, da zum ersten die konstruierten Parser einfach dahingehend erweitert werden könnten, daß ihre Reduktionen auch die komplexen Handles bestimmen; zum zweiten bewirkt die Einschränkung auf deterministische RRPAs, daß, anders als bei möglicherweise mehrdeutigen RRPAs, semantische Aktionen, die mit den RRPA-Übergängen verbunden sind, in eindeutiger Weise abgerufen werden können. Die Autoren selbst gehen auf diesen Punkt nicht ein. Als Kritikpunkt verbleibt, daß die Forderung nach deterministischen RRPAs unnötige Einschränkungen bewirkt. Für ekfGen bedeutet sie die Beschränkung auf reguläre Ausdrücke mit deterministischen Glushkov-NEAs.[60]

9.6.3 Nakata und Sassa (1986)

Der Journalbeitrag von (Nakata und Sassa 1986) weist von den Ansätzen her merkliche Ähnlichkeiten zur soeben diskutierten Arbeit von Purdom und Brown (1981) auf: Ebenfalls werden RRPGen mit deterministischen RRPAs betrachtet; die verwendete Itemmengenkonstruktion ist dieselbe; und auch die von Nakata und Sassa konstruierten Parser kellern nur solche Itemmengen, die auf dem Parserkeller nach „unten" ein Handle begrenzen können. Allerdings vermeiden die Autoren die von Purdom und Brown (1981) eingesetzte Transformation von solchen RRPGen, für die Itemmengen existieren, die auf dem Parserkeller ein Handle nach „unten" begrenzen *können,* aber nicht müssen: Die Reduktionen der Parser von Nakata und Sassa (1986) entkellern den Parserkeller nicht nur einmal, sondern sooft, bis eine Itemmenge zuoberst liegt, die aus einer bei Reduktionsbeginn bestimmten, statisch berechneten Kandidatenmenge entstammt. Ihre konstruierten Parser nennen die Autoren ELALR(1)-Parser.

Leider scheitert ein Vergleich ihrer Parserkonstruktionsmethode mit derjenigen, die wir selbst vorgeschlagen haben, schon am völligen Fehlen wesentlicher Grundlagen in ihrer Arbeit. So führen Nakata und Sassa nicht einmal einen Ableitungsbegriff für RRPGen ein. Als einzige „Definition" ihres Begriffs der ELR(*k*)-Grammatik finden wir:

[60] Siehe hierzu (Brüggemann-Klein 1993) für weiterführende Hinweise.

„An extended LR(k) (ELR(k)) grammar is an extended context free grammar which can be parsed from left to right with a lookahead of k symbols."

Auch die Anforderungsdefinition der von ihnen konstruierten ELALR(1)-Parser

„An RRP grammar is said to be an ELALR(1) grammar if and only if (i) parsing conflicts in inadequate LR(0) states can be resolved by using lookahead symbols [Anm. des Autors: welche ?!] *and (ii) at any reduction during LALR parsing the handle to be reduced can be uniquely determined."*

ist unbrauchbar, da eine Bildungsvorschrift für die darin angesprochenen Lookahead-Symbole nirgendwo erklärt wird.

Und zu allem Überfluß stützen die Autoren sich in Beweisen auf die mögliche Eigenschaft von Items, „Kernelitems" zu sein – ihre Funktion Kernel, die die Kernelitems einer Itemmenge angeben soll, ist jedoch nicht wohldefiniert.

Daß Nakata und Sassa (1986) keinen Nachweis für die Korrektheit ihrer Parserkonstruktionsmethode angeben, ist nun nicht anders zu erwarten. Statt dessen beschäftigen sich die Autoren schließlich noch mit einer Verfahrensverfeinerung, die eine leichte Effizienzsteigerung der generierten Parser bewirken kann, Die hier eingesetzte Energie wäre besser in akkuratere Grundlagenarbeit investiert worden.

9.6.4 Shin und Choe (1993)

Shin und Choe (1993) verstehen ihren Ansatz als eine Verbesserung der soeben in ihren Grundzügen vorgestellten Arbeit von Nakata und Sassa (1986): Sie bemühen sich in ihrem Journalbeitrag ebenfalls um die Konstruktion von ELALR-Parsern für RRPGen, erlauben jedoch nichtdeterministische RRPAs, und ihre Konstruktionsmethode, die ansonsten im wesentlichen die von Nakata und Sassa (1986) ist, stützt sich nur auf „Kernelitems", was in der Generierung geringere Speicherplatzanforderungen verspricht. Die von ihnen eingesetzte Itemmengenkonstruktion ist die übliche – wir kennen sie aus der ausführlichen Diskussion von (Chapman 1984). Wie üblich wird sie in Form einer mehr oder weniger unmotivierten Verfahrensvorschrift präsentiert.

Ähnlich wie bei der Diskussion von (Nakata und Sassa 1986) ist auch hier ein Vergleich der von uns vorgestellten Technik zur ELALR(k)-Parserkonstruktion mit der von Shin und Choe (1993) aufgrund ihres völlig unbrauchbaren theoretischen Fundaments

unmöglich: Shin und Choe (1993) präsentieren tatsächlich keinen Ableitungsbegriff und keinen ELR(k)-Begriff ! Und außer dem Satz

> *„The ELALR(k) parser can be constructed by computing the collection of sets of ELR(0) items and augmenting each item with set of lookahead strings of length k.“*

finden wir keinen weiteren Hinweis, mit *welchen* k-Lookaheads gearbeitet werden soll.

Wie schon bei (Nakata und Sassa 1986) spielt auch bei (Shin und Choe 1993) der Begriff des „Kernelitems“ eine kruziale Rolle; Shin und Choe scheitern jedoch nicht wie Nakata und Sassa an der Wohldefiniertheit der Begriffsdefinition – sie bieten überhaupt keine Definition an ! Natürlich ist auch bei ihnen kein Ansatz zu einem Nachweis der Korrektheit der konstruierten Parser zu finden.

Wir ersparen uns auch hier die weitere Diskussion.

9.6.5 Fortes Gálvez (1994)

Anliegen des sehr kurzen Artikels von Fortes Gálvez (1994) ist, auf einige Fehler in (Shin und Choe 1993) hinzuweisen. Der Autor diskutiert oberflächlich einige Schwächen in (Shin und Choe 1993), präsentiert für eine Beispiel-RRPG den gemäß (Shin und Choe 1993) konstruierten ELALR(1)-Parser[61] und stellt fest, daß dieser Parser konfliktfrei zu sein scheint. Andererseits ist diese RRPG, wie sich Fortes Gálvez ausdrückt, *„clearly ambiguous“*. Der Autor belegt seine Behauptung mittels zweier Beispiel-Ableitungen, die sich der von den meisten Autoren verwendeten Ableitungsrelation bedient, welche von den RRPA-Strukturen abstrahiert. Erheiternd ist hieran, daß, wie wir oben erwähnten, in der kritisierten Arbeit von (Shin und Choe 1993) eine solche Ableitungsrelation überhaupt nicht definiert wurde !

9.6.6 Lee und Kim (1997)

Die jüngste einschlägige Arbeit zur Thematik der ELR-Grammatiken und -Parser ist ein Journalbeitrag von Lee und Kim (1997). Bemerkenswert ist an ihm vor allem die darin erstmals geäußerte Kritik an der theoretischen Fundierung der meisten älteren Arbeiten zum Thema (s. u.).

[61] Wie der Lookahead dabei bestimmt wird, ist unklar.

Die Autoren beschäftigen sich mit der konstruktiven maschinellen Entscheidbarkeit, wann eine (in unserer Sprechweise:) RRPG eine ELR(k)-Grammatik ist. Ihr Verständnis von der ELR(k)-Eigenschaft entspricht – wie in den meisten Veröffentlichungen zum Thema – der von uns vorgestellten Definition von Heilbrunner (1979), also einer Sicht, in der die „Struktur" der Handles in den Rechtsableitungen keine Rolle spielt, ja sogar mehrdeutig sein kann, ohne daß dies von Einfluß auf die ELR(k)-Eigenschaft der Grammatik wäre. Nicht vollständig behandelt wird von Lee und Kim allerdings die Konstruktion eines Parsers für ihre ELR(k)-Grammatiken: Zwar wird das Parserverhalten formal mit einer Konfigurationsübergangsrelation beschrieben; die Reduktionen der komplexen Handles sind dabei jedoch makroartig als ein einziger Schritt charakterisiert, und wie nun (bzw. ob überhaupt) dieses Makroverhalten effizient implementiert werden kann – noch dazu hoffentlich mit den Mitteln eines deterministischen Kellerautomaten –, wird von den Autoren bewußt offengelassen:

> *„The subject of efficient techniques for locating the left end of a handle is beyond the scope of this paper and will not be discussed further."*

Wie bereits angekündigt, ist aus dieser Arbeit besonders eine kurze Passage erwähnenswert, die die fehlende theoretische Untermauerung in anderen Veröffentlichungen kritisiert:

> *„For ELR(k) parsing, several approaches [(Chapman 1984), (Heilbrunner 1979), (LaLonde 1977, 1979), (Madsen and Kristensen 1976), (Nakata and Sassa 1986), (Purdom and Brown 1981), (Shin and Choe 1993)] have been suggested, but most of them did not precisely characterize the parsable grammar class. Although a few approaches [(Nakata and Sassa 1986), (Shin and Choe 1993)] tried to present some lemmas deciding whether a given grammar is parsable or not, the lemmas have been found to be incorrect [(Fortes Gálvez 1994)]. As known well, such testing is crucial for reliable use of the suggested parsers, but there have been no noticeable efforts to correctly characterize ELR(k) or ELALR(k) grammars since then. The work described here was motivated by this situation."*

Diesem Verweis auf die teils krassen theoretischen Defizite der meisten einschlägigen Arbeiten können wir uns nur anschließen.

Auch Lee und Kim (1997) müssen sich allerdings vorwerfen lassen, die seit langem vorhandene theoretische Vorarbeit gerade von Heilbrunner (1979), auf dessen Arbeit sie selbst

verweisen, nicht ausreichend zur Kenntnis genommen zu haben. Denn wie die Ortung des linken Handleendes dann mit den traditionellen Mitteln eines deterministischen Kellerautomaten in ihrem Szenario, in dem die Struktur der RRPAs keine Rolle spielt, zu lösen ist, liegt nach Lektüre von (Heilbrunner 1979) auf der Hand: Wie auf Seite 153 bereits beschrieben, ersetze man sämtliche RRPAs durch eindeutige (z. B. deterministische) Pendants, formuliere diese als rechtslineare Subgrammatiken und generiere sodann einen klassischen LR(k)-Parser. Falls dieser deterministisch ist, so ist auch die Ausgangs-RRPG ELR(k), und die Handleanfänge sind leicht zu bestimmen.

Lee und Kim (1997) formulieren dagegen eine Itemmengen-Konstruktion direkt auf Grundlage der Ausgangs-RRPG. Ob sich das entstehende Ergebnis überhaupt eignet, um daraus auch bei mehrdeutigen RRPAs dennoch ggf. einen deterministischen ELR(k)-Parser abzulesen, ist unklar. (LaLonde (1977, 1979) hat zwar denselben Weg schon lange zuvor vorgeschlagen, eine theoretische Fundierung dieses Ansatzes jedoch nicht einmal in Grundzügen geliefert.)

9.6.7 Madsen und Kristensen (1976)

Wir wenden uns nun der letzten der nennenswerten Arbeiten zu, die wir noch diskutieren wollen: der Arbeit von Madsen und Kristensen (1976), die die früheste Journalveröffentlichung zum Thema ist. Sie und die Publikation von Heilbrunner (1979) sind nicht nur die gewinnbringendsten, sondern, was ihren eigentlichen Beitrag angeht, auch die beiden verkanntesten einschlägigen Arbeiten. Madsen und Kristensen (1976) behandeln ausschließlich ekfGen, für die schwerpunktmäßig reine ELR(k)-Parser konstruiert werden.

Wie wir es schon von Heilbrunner (1979) kennen, führen auch Madsen und Kristensen (1976) über das Konzept der *unendlichen* Produktionenmengen, die über reguläre Ausdrücke auf rechten Regelseiten *repräsentiert* werden, in die Thematik ein. Ihre Definition des Begriffs der ELR(k)-Grammatik fordert – wie auch unsere eigene –, daß diese repräsentierenden regulären Ausdrücke stets „eindeutig“ sein müssen. Sie tut dies allerdings auf indirekte Weise, denn interessanterweise definieren die Autoren gar nicht, was sie unter einem ein- bzw. mehrdeutigen regulären Ausdruck verstehen. Ihr offenkundiges Verständnis davon geht jedoch aus ihrem Konzept der „Instanz“ eines regulären Ausdrucks hervor. Dies ist eine mit Klammern derart versehene Expansion α' eines regulären Ausdrucks α, daß durch Streichen dieser Klammern ein vom Ausdruck beschriebenes Wort α'' entsteht und dabei die Klammern exakt rekonstruieren lassen, wie dieses Wort mit Hilfe der vorkommenden Operatoren und Zeichenvorkommen des Ausdrucks gebildet

wird. Implizit fassen Madsen und Kristensen einen regulären Ausdruck als mehrdeutig auf, zu dem ein Wort mit zwei verschiedenen Instanzen gebildet werden kann. Dieses Verständnis von der Ein- bzw. Mehrdeutigkeit eines regulären Ausdrucks ist nun insofern bemerkenswert, als es gewissermaßen seiner Zeit voraus ist: Es entspricht nämlich dem, was wir mit Brüggemann-Klein (1993) als *starke* Eindeutigkeit bezeichnet haben und wofür Brüggemann-Klein die viel spätere Quelle (Sippu und Soisalon-Soininen 1988) angibt, und nicht etwa der *schwachen* Eindeutigkeit (Book, Even, Greibach und Ott 1971), die zum Zeitpunkt des Erscheinens von (Madsen und Kristensen 1976) aus der Literatur bekannt war. Damit ist eine ekfG nach unserer eigenen Definition genau dann eine ELR(k)-Grammatik, wenn sie auch im Sinne von Madsen und Kristensen (1976) eine ELR(k)-Grammatik ist.[62]

Auf dieser Grundlage geben Madsen und Kristensen (1976) nun zwei Verfahren an, Syntaxanalysatoren für ihre ELR(k)-Grammatiken zu konstruieren.

Das erste Verfahren formt über ein festes Transformationsschema die komplex strukturierten regulären Ausdrücke auf den rechten Regelseiten der ekfG durch Einführen von Hilfsnichtterminalen sukzessive um in eine reine kfG, die dieselbe Sprache aufspannt. Madsen und Kristensen beweisen überzeugend, daß diese kfG genau dann eine LR(k)-Grammatik ist, wenn die Ausgangs-ekfG eine ELR(k)-Grammatik ist. Wir können uns Einzelheiten hier ersparen, wollen aber darauf hinweisen, daß alleine schon diese Leistung der Autoren in ihrer theoretischen Tragweite von sämtlichen diskutierten Folgearbeiten mit Ausnahme von (Heilbrunner 1979) ignoriert und/oder nicht verstanden wurde.

Das zweite Verfahren, das Madsen und Kristensen (1976) präsentieren, konstruiert einen Syntaxanalysator für ELR(k)-Grammatiken ohne den transformierenden Zwischenschritt direkt auf der Grundlage der ekfG. Die Autoren skizzieren die wesentlichen Elemente einer Itemmengenkonstruktion, in der die regulären Ausdrücke auf den rechten Regelseiten, mit einem *„LR-marker (dot)“* versehen, die Rolle der Items übernehmen:

> *„The LR-items are constructed in the normal way (Aho and Ullman 1972, algorithm 5.8, 5.9) using the following additional rules (which extend rules proposed by DeRemer (1974) and Earley (1970)):*

[62] Siehe dazu auch (Heilbrunner 1979). Daß Madsen und Kristensen (1976) mit einem anderen als dem zum damaligen Zeitpunkt üblichen Konzept der Eindeutigkeit regulärer Ausdrücke arbeiten, ist übrigens scheinbar auch Heilbrunner entgangen.

(1) Any item of the form

$$[A \to \alpha \cdot \{\beta\}^* \gamma, v]$$

is replaced by

$$[A \to \alpha \# \{\beta\}^* \gamma, v],$$
$$[A \to \alpha \{ \cdot \beta\}^* \gamma, v],$$
$$[A \to \alpha \{\beta\}^* \cdot \gamma, v].$$

(2) Any item of the form

$$[A \to \alpha \{\beta \cdot \}^* \gamma, v]$$

is replaced by

$$[A \to \alpha \{\beta \# \}^* \gamma, v],$$
$$[A \to \alpha \{\beta\}^* \cdot \gamma, v],$$
$$[A \to \alpha \{ \cdot \beta\}^* \gamma, v].$$

(3) Any item of the form

$$[A \to \alpha \cdot \{\omega_1 \mid \omega_2 \mid \ldots \mid \omega_n\} \beta, v]$$

is replaced by

$$[A \to \alpha \{\omega_1 \mid \omega_2 \mid \ldots \mid \cdot \omega_i \mid \ldots \mid \omega_n\} \beta, v]$$

for $i = 1, 2, \ldots, n$.

(4) Any item of the form

$$[A \to \alpha \{\omega_1 \mid \omega_2 \mid \ldots \mid \omega_i \cdot \mid \ldots \mid \omega_n\} \beta, v]$$

is replaced by

$$[A \to \alpha \{\omega_1 \mid \omega_2 \mid \ldots \mid \omega_i \# \mid \ldots \mid \omega_n\} \beta, v],$$
$$[A \to \alpha \{\omega_1 \mid \omega_2 \mid \ldots \mid \omega_n\} \cdot \beta, v].$$

The idea behind these rules is to keep track of which productions are applicable at a given point in the input string, in a manner consistent with the LR construction technique and with the meanings of regular expressions. # is a new symbol meaning that the corresponding item is not evaluated further. An item containing a # is called a reduce item. *The purpose of the # is to distinguish between the items introduced in cases (1) and (2) and between the items introduced for different i's in case (4).“*

Der Beweis der Korrektheit dieser Konstruktion erfordert die Einführung einiger sperriger Definitionen, um mit dem an beliebigen Stellen geteilten regulären Ausdrücken formal sauber umzugehen. Teilweise leistet dies (Madsen und Kristensen 1975), während (Madsen und Kristensen 1976) hier an der Oberfläche bleibt.

Nicht sofort einleuchtend an dieser Konstruktion ist, welcher Nutzen letztendlich aus nunmehr zwei Sorten von Items gezogen werden soll. Aus Madsens und Kristensens Erklärung *„# is a new symbol meaning that the corresponding item is not evaluated further“* erwächst dem Leser zunächst der Eindruck, das #-Symbol diene in erster Linie der Steuerung oder Effizienzsteigerung der Itemmengenkonstruktion. Dieser Eindruck lenkt jedoch ab. Auch die Erläuterung der Autoren, der Zweck des #-Symbols sei *„to distinguish between the items introduced in cases (1) and (2) and between the items introduced for different i's in case (4)“*, trifft den Nagel zuwenig auf den Kopf: Der Leser überzeuge sich anhand der oben und im folgenden zitierten Passagen selbst, daß im Verfahren von Madsen und Kristensen ohne weiteres auf die gesonderten #-Markierungen verzichtet werden kann, ohne dessen Funktionieren zu beeinträchtigen.

Der folgende längere Ausschnitt aus (Madsen und Kristensen 1976) beschreibt die von den Autoren im Parser eingesetzte Technik zur Abwicklung der komplexen Reduktionen.[63] Insbesondere achte man bei seiner Lektüre auf die Rolle der #-markierten *„reduce items“* (von uns #-Items genannt):

„Turning to the parsing problem, a set of LR-tables is constructed as in (Aho and Ullman 1972, algorithm 5.11). In addition, each LR-table is extended with a

[63] Mit der von ihnen selbst eingeführten Bezeichnungskonvention, die einen regulären Ausdruck α, eine Instanz α' von α und ein von α' beschriebenes Wort α'' unterscheidet, gehen Madsen und Kristensen (1976) hier selbst nicht präzise um. Beispielsweise sind keinesfalls Instanzen vom Parserkeller zu entfernen, sondern Wörter, deren sie strukturierende Instanzen gleichzeitig zu ermitteln sind!

Im zitierten Text bedeutet $I(\alpha)$ die Menge der Instanzen eines regulären Ausdrucks α, und $\}_i$ sind spezielle Klammersymbole, die in Instanzen verwendet werden.

description of the reduce items contained in the corresponding set of items.

The parsing algorithm works as normal (Aho and Ullman 1972, algorithm 5.7) except when a reduction is applied.

Assume that the production $A \to \alpha$ is applied. Let α' be the instance of α actually read. $|\alpha'|$ symbols on top of the pushdown list have to be popped. α can be split into $\delta_1, \delta_2, \ldots, \delta_m$ ($m \geq 1$) such that: $\alpha = \delta_1\delta_2\ldots\delta_m$ and each δ_i has the form

(1) $\delta_i \in (N \cup \Sigma)^*$ or

(2) $\delta_i = \{\beta\}^*$ for some β or

(3) $\delta_i = \{\omega_1 \mid \omega_2 \mid \ldots \mid \omega_n\}$ for some ω_j, $j = 1,2,\ldots n$.

(Note, the splitting is unique if δ_i and δ_{i+1} are not both of the form (1) for any $i = 1,2,\ldots,m \Leftrightarrow 1$.)

It follows that $\alpha' = \delta'_1\delta'_2\ldots\delta'_m$ where δ'_i is an instance of δ_i. The popping of the pushdown list can be split into popping first $|\delta'_m|$ symbols, then $|\delta'_{m-1}|$ symbols etc., and finally $|\delta'_1|$ symbols. Assume that $|\delta'_m|, |\delta'_{m-1}|, \ldots, |\delta'_{i+1}|$ symbols have been popped and that we are going to reduce an instance of δ_i. Let $\varphi = \delta_1\ldots\delta_{i-1}$, $\psi = \delta_{i+1}\ldots\delta_m$.

(a) If δ_i has the form (1) then $|\delta'_i|$ symbols have to be popped.

(b) If δ_i has the form (2), then $\delta'_i = \{\beta_1\beta_2\ldots\beta_s\}_s$, $s \geq 0$ and $\beta_i \in I(\beta)$, $i = 1,2,\ldots,s$. To pop $|\delta'_i|$ symbols consists of repeatedly popping instances of β using the following rules. If the item $[A \to \varphi\{\beta\#\}^*\psi, v]$ is in the LR-table on top of the pushdown list and v matches the next k input symbols then an instance of β can be popped. If the item $[A \to \varphi\#\{\beta\}^*\psi, v]$ is in the topmost LR-table we can stop the popping of instances of β. If only one of the items is in the topmost LR-table at a time no conflict exists but if both are present then we in general don't know whether to stop or to continue popping instances of β. To pop an instance of β we have to make a similar splitting of β as we did with α.

(c) If δ_i has the form (3), then $\delta'_i = \{\omega'_r\}_r$ $(1 \leq r \leq n)$, $\omega'_r \in I(\omega_r)$. If the topmost LR-table contains the item $[A \to \varphi\{\omega_1 \mid \ldots \mid \omega_s\# \mid \ldots \mid \omega_n\}\psi, v]$ and v matches the next k input symbols, then an instance of ω_s may be valid, i.e. $r = s$. If the LR-table also contains an item $[A \to \varphi\{\omega_1 \mid \ldots \mid \omega_j\# \mid \ldots \mid \omega_n\}\psi, v]$, $j \neq s$, then we in general don't know what to do. As in (b) popping an instance of ω_s involves a splitting of ω_s."

Wesentliche Teile eines Korrektheitsnachweis des Verfahrens enthält der technische Bericht (Madsen und Kristensen 1975), der damit den akkurat dargelegten Journalbeitrag (Madsen und Kristensen 1976) in theoretischer Tiefe ergänzt (aber ebenfalls die Notwendigkeit der #-Markierungen nicht schlüssig erklärt). Mit den folgenden Anmerkungen aus unserer heutigen Sicht wollen wir die Diskussion schließen:

1. Der von den Autoren eingeschlagene Weg, eine direkte ELR(k)-Parserkonstruktion darüber vorzunehmen, daß sie Items mit geteilten regulären Ausdrücken definieren, ist in seinem Korrektheitsnachweis formal sperrig. Dasselbe über den von uns selbst beschriebene Weg zu leisten stellt einen zwar längeren, nie jedoch sperrigen Weg dar, der zudem die Zusammenhänge zwischen der ELR(k)-Parserkonstruktion für ekfGen und für RRPGen aufdeckt.

2. Das im obigen Zitat beschriebene Verfahren von Madsen und Kristensen zur Abwicklung der komplexen Reduktionen ist essentiell eine rekursive Prozedur,[64] die während der sukzessiven Untersuchung der gekellerten Itemmengen auf dem Parserkeller ermittelt, welche Instanz eines regulären Ausdrucks einer rechten Regelseite die Struktur des komplexen Handles bestimmt. Die Instanz wird dabei nur gedacht konstruiert und währenddessen anschaulich von rechts nach links durchlaufen. Als wesentliche Hilfsinformation für die Umsetzung dieser Aufgabe benötigt die Prozedur zu jeder auf dem Parserkeller analysierten Itemmenge die Kenntnis von deren #-Items zum aktuellen Lookahead. Insbesondere benötigt eine Implementierung immer wieder Zugriff auf strukturelle Bestandteile des #-geteilten regulären Ausdrucks in den #-Items, um die Reduktion fortsetzen zu können. Diese repetierte Strukturinterpretation dürfte die Effizienz des Parsers von Madsen und Kristensen merklich negativ beeinträchtigen. Das von uns selbst angegebene Verfahren weist diese Ineffizienz nicht auf.

3. Es ist eine für das Verständnis wichtige Beobachtung, daß die soeben hervorgehobenen Strukturinterpretationen von Items während der Reduktionen gerade die #-Items betreffen, während auf die sonstigen (sozusagen normalen) Items nicht zugegriffen werden muß. Genau *dies* hätten Madsen und Kristensen ihren Lesern als Motivation für die ungewöhnlichen #-Markierungen vermitteln sollen.

4. Madsen und Kristensen (1976) orientieren sich, wie wir gesehen haben, an (Aho und Ullman 1972), wenn sie ihre Parser für ELR(k)-Grammatiken als tabellenge-

[64] Siehe auch (Madsen und Kristensen 1975).

steuerten Algorithmus präsentieren (und verstehen) statt als einen deterministischen Kellertransduktor. Besonders die Beschreibung der komplexen Reduktionen als eine Art *rekursive* Prozedur macht eine prinzipielle Realisierbarkeit eines solchen Parsers als deterministischer Kellertransduktor alles andere als offensichtlich und rief beim Autor der vorliegenden Dissertation über längere Zeit Zweifel hervor.

10 Zusammenfassung und Ausblick

Im ersten Teil der vorliegenden Arbeit konnten wir zeigen, wie aus „allgemeinen LR(k)-Maschinen“, die durch Abstraktion von irrelevanter Information in den bekannten kanonischen LR(k)-Maschinen gewonnen werden können, „allgemeine LR(k)-Parser“ aufgebaut werden, die von identischer formaler Bauweise wie die üblichen kanonischen LR(k)-Parser, jedoch potentiell von erheblich kleinerer formaler Größe sind. Die effizient bestimmbaren allgemeinen LR(k)-Parser minimaler Größe sind starke Faktorisierungen der kanonischen LR(k)-Parser und immer noch von linearer Laufzeit. Darüber hinaus sind alle allgemeinen LR(k)-Parser genau dann deterministisch, wenn auch der kanonische LR(k)-Parser zur selben Grammatik deterministisch ist.

Ihre attraktivere Größe erzielen allgemeine LR(k)-Parser, indem sie neben dem k-Lookahead die *gesamte* vorhandene Parserkellerinformation entlang zu reduzierender Handles für die Entscheidungsfindung bezüglich anzuwendender Parseraktionen nutzen anstatt praktisch nur den obersten Kellerzustand. Dabei handeln diese Parser gewonnene Kompaktheit gegen aufwendigeren Code zur Implementierung der Parseraktionen und machen gleichzeitig die in den traditionellen LR-Parsern unverzichtbaren Konstruktionsbestandteile klar, tragen also auch in didaktischer Hinsicht zum Verständnis der Konstruktion solcher Parser bei.

Ebenfalls studiert haben wir, wie SLR(k)-, LALR(k)- und die neuen, noch etwas mächtigeren ILALR(k)-Parser konstruiert werden können, wobei letztere gegenüber LALR(k)-Parsern dort ungenutztes Kellerkontextwissen nutzen, um zu schärferen Lookahead-Mengen zu gelangen. Und schließlich haben wir vorgestellt, wie ILALR(1)-Lookaheads beinahe wie LALR(1)-Lookaheads im Stile von DeRemer und Pennello (1982) effizient aus der Struktur der kanonischen LR(0)-Maschine berechnet werden können.

Insgesamt konnten wir herausarbeiten, welche Bestandteile der klassischen Konstruktionen für LR-Parser unverzichtbar sind für das korrekte Funktionieren der produzierten LR-Parser und welche verzichtbar sind und mit Redundanz in der Konstruktion bezahlt werden. Nicht zuletzt tragen wir damit auch in didaktischer Hinsicht zum Verständnis der Konstruktion von LR-Parsern bei.

Als vielversprechende zukünftige Arbeiten zu diesem ersten Themenkomplex bieten sich die folgenden an:

- Ergebnisse von Ukkonen (1983, 1985) zur Größe von deterministischen LR-Parsern sollten für minimale allgemeine LR-Parser reinspiziert werden.

- Implementierungserfahrungen mit einem sorgfältig implementierten Generator für allgemeine LR(k)- und/oder (I)LALR(k)-Parser könnten belegen, von welcher Größenordnung für Programmiersprachengrammatiken in der Praxis der Handel von Größe versus Laufzeit gegenüber kanonischen LR-Parsern ist. Auch wären geschickte Techniken, die Tabelleninterpretationssequenzen für Parseraktionen durch direkt ausführbaren Code zu ersetzen – siehe etwa (Pennello 1981) – von Interesse.

- Speziell für die *kanonischen* ILALR(k)-Parser haben wir eine Strategie zur Implementierung skizziert, von der wir vermuteten, daß sie ähnlich laufzeit- und speichereffizient ist wie übliche Strategien zur Implementierung kanonischer LALR(k)-Parser. Diese Vermutung sollte überprüft werden.

- Das LALR(k)-Verfahren projiziert bekanntlich LR(k)-Lookahead-Information auf die *kanonische* LR(0)-Maschine. Mit den in der vorliegenden Arbeit vorgestellten *allgemeinen* LR(k)-Maschinen stehen nun auch andere Projektionsziele zur Verfügung – beispielsweise kann auf die *minimale* allgemeine LR(0)-Maschine projiziert werden. Möglicherweise erweisen sich entsprechende Varianten des LALR- oder ILALR-Verfahrens als nützliche Erweiterungen des verfügbaren Spektrums an Kompromissen zwischen Implementierungsgröße, -laufzeit und Verfahrensmächtigkeit.

- Die Theorie kanonischer LL(k)-Parser weist viele Dualitäten zur Theorie kanonischer LR(k)-Parser auf – siehe Kapitel 8 in (Sippu und Soisalon-Soininen 1990) für eine ausführliche Behandlung, auf die wir uns terminologisch beziehen, und für zahlreiche bibliographische Hinweise. Die in der vorliegenden Arbeit vorgestellte Herangehensweise an (minimale) allgemeine LR(k)-Parser ist prinzipiell übertragbar auf „(minimale) allgemeine LL(k)-Parser“ (und dann ihre LA(k)LL(l)-Varianten). Dazu sei daran erinnert, daß kanonische LL(k)-Maschinen zu „lebensfähigen Suffizes“ deren „LL(k)-gültige Items“ berechnen, und diese enthalten teilweise verzichtbare Information: In einem Item $[A \to \alpha X \cdot \beta, y]$ ist die Komponente β unnötiger Ballast; ebenso auch A, wenn auf dem Parserkeller zwischen den Zuständen der LL(k)-Maschine auch das aktuelle lebensfähige Suffix mitgeführt wird.

Im zweiten Teil der vorliegenden Arbeit haben wir uns mit der Problematik befaßt, „ELR-Parser“, d. h. LR-Parser für erweiterte kontextfreie Grammatiken, theoretisch fundiert zu

konstruieren. Wie es schon in dem im ersten Teil behandelten Themenkomplex der klassischen LR-Theorie der Fall ist, ist dieses Arbeitsgebiet so alt, wie es (aus Gründen, die wir diskutiert haben) für den Bau von Übersetzern für Programmiersprachen von Relevanz ist. Im Gegensatz zur klassischen LR-Theorie jedoch, die bereits seit langem theoretisch wohldurchdrungen ist, weisen mit wenigen Ausnahmen die einschlägigen Veröffentlichungen zum zweiten Thema bis heute schwerwiegende Schwächen auf, wie unsere Literaturanalyse ergeben hat: Ausgenommen die früheste solche Arbeit überhaupt, die Arbeit von Madsen und Kristensen (1976), beachtet keine Arbeit, daß es im Bereich des Übersetzerbaus dringend geboten erscheint, einen Ableitungsbegriff zu verwenden, der klassische Ableitungsbäume impliziert, d. h. Ableitungsbäume, die über einer geeigneten eindeutigen und endlichen kontextfreien Grammatik in der bekannten Art und Weise gebildet werden können. Insbesondere muß es, damit erweiterte kontextfreie Grammatiken im Kontext des Übersetzerbaus sinnvoll eingesetzt werden können, möglich sein, semantische Aktionen zur Erklärung einer eindeutigen Compilersemantik zu formulieren. Zu diesem Zweck ist es unabdingbar, die *Struktur* der auf der rechten Regelseite verwendeten regulären Beschreibungshilfsmittel in den verwendeten Ableitungsbegriff einfließen zu lassen. Eine Transformation der regulären Ausdrücke oder endlichen Automaten auf rechten Regelseiten in äquivalente rein kontextfreie Grammatikkonstrukte ist ein naheliegender Schritt, der konsequent jedoch nur von Madsen und Kristensen (1976) vollzogen worden ist.

Statt dessen ist der in der Literatur verwendete Ableitungsbegriff fast immer ein für den tradierten Übersetzerbau unbrauchbarer: Er abstrahiert zumeist von den konkreten Strukturen der rechten Regelseiten, eine klare theoretische Fundierung ist kaum vorhanden. Und auch die Frage, ob die verschiedenen konstruierten Parser prinzipiell als deterministische Kellerautomaten bzw. -transduktoren formuliert werden können, wird nie thematisiert, so daß meist unklar bleibt, welche genaue Einordnung in die Theorie der Formalen Sprachen mit ihren Beziehungen zur Automatentheorie besteht. Erst recht erschwert wird diese Einordnung in einigen Fällen durch krasse definitorische Schwächen oder Versäumnisse.

Um die diskutierten Ansätze aus der Literatur konstruktiv kritisieren zu können, haben wir ein eigenes Verfahren zur Konstruktion von ELR-Parsern vorgestellt, das reguläre Ausdrücke auf rechten Regelseiten umformt in äquivalente endliche Automaten und diese wiederum in äquivalente rechtslineare Subgrammatiken. Auf diese Weise kann die Konstruktion von ELR-Parsern für Grammatiken mit regulären Ausdrücken und solche mit endlichen Automaten auf rechten Regelseiten vereinheitlicht und auf die Konstruktion von LR-Parsern für reine kontextfreie Grammatiken zurückgeführt werden.

Diese Vorgehensweise ist theoretisch fundiert und bedient sich gleichzeitig eines vorhan-

denen Begriffsapparats und Theoriegerüsts. Dabei wird gerade nicht von der Struktur der regulären Ausdrücke bzw. endlichen Automaten auf den rechten Regelseiten abstrahiert. Daß die resultierenden Parser deterministische Kellertransduktoren sind, folgt unmittelbar.

Für Grammatiken mit regulären Ausdrücken auf rechten Regelseiten stellt sich heraus, daß nach unseren Vorstellungen die *Eindeutigkeit* der regulären Ausdrücke notwendige Voraussetzung für die ELR(k)-Eigenschaft der Grammatik ist. Präziser benötigen wir die „starke Eindeutigkeit" der regulären Ausdrücke, die nach Brüggemann-Klein (1993) auf Sippu und Soisalon-Soininen (1988) zurückgeht und strenger ist als die „schwache Eindeutigkeit" (Book, Even, Greibach und Ott 1971). Ferner konnten wir zeigen, daß der Zusammenhang zwischen diesen beiden Eindeutigkeitsbegriffen, den bekanntlich Brüggemann-Klein (1993) über die Konjunktion der beiden Eigenschaften eines regulären Ausdrucks, in sogenannter Stern- *und* ε-Normalform zu sein, charakterisiert, einfacher formuliert werden kann: Es stellt sich heraus, daß jeder reguläre Ausdruck in ε-Normalform allemal in Stern-Normalform ist. Dieses Ergebnis vereinfacht das Verständnis des Zusammenhangs zwischen den beiden Eindeutigkeitskonzepten.

In unserer Literaturanalyse haben wir uns ausführlicher mit der Arbeit von Chapman (1984) befaßt, dessen die Reduktionsphasen abwickelnde „Readback-Automaten" wir mit rein kontextfreien Mitteln theoretisch fundiert rekonstruiert haben. Dabei konnten wir von der verallgemeinerten LR-Theorie aus dem ersten Teil der vorliegenden Arbeit profitieren und einige Fehler in Chapmans Konstruktion aufdecken. Auch haben wir einsehen können, daß für festes k die von Chapman formulierten Klassen der LALR(m,k)-Parser ($m \geq 1$) bzw. ihrer entsprechenden Grammatiken keine echte Hierarchie bilden, sondern zusammenfallen. Analoges gilt auch für die von LaLonde (1977) formulierten LR(m,k)-Parser bzw. ihre zugehörigen Grammatiken.

Die größten Gemeinsamkeiten weist unser Verfahren mit einem Verfahren von Madsen und Kristensen (1976) auf; ihre formale Herangehensweise ist jedoch teilweise sperrig, im Detail mitunter auch fragwürdig, und die Reduktionsphasen ihrer ELR-Parser sind als rekursive Prozeduren formuliert, die nicht nur aufwendiger zu sein scheinen als unsere Readback-Automaten, sondern auch den Bezug zur Automatentheorie unklar lassen.

Die Ergebnisse, die die vorliegende Arbeit zum Themenkomplex der ELR-Parser präsentiert hat, könnten sinnvoll besonders durch ihre Erprobung in anspruchsvollen Implementierungsprojekten umgesetzt werden. Wir skizzieren eine Möglichkeit:

- Es sollten Implementierungs- und Nutzungserfahrungen im Zusammenhang mit Compiler-Generierungssystemen gesammelt werden. Eine geeignete Plattform

hierfür wäre beispielsweise das kompakte Compiler-Generierungssystem *epsilon* auf der Grundlage erweiterter Affixgrammatiken, das als Nachfolger des Compiler-Generierungssystems *eta* an der Technischen Universität Berlin entwickelt wurde (Demuth, Weber, Kannapinn und Kröplin 1997): Mit dem Ziel eines prägnanteren Spezifikationsstils und auch, um als kontextfreie Skelettsyntax (die sogenannte Hypergrammatik) möglichst originale Report-Grammatiken von Programmiersprachen einsetzen zu können, wurde dieses Werkzeug darauf ausgelegt, als Eingabespezifikationen Erweiterte Affixgrammatiken in einer Notation zu akzeptieren, in der auf rechten Hyperregelseiten reguläre Ausdrücke verwendet werden dürfen. Affixinformation muß dann nicht nur über die Hypernichtterminale fließen, sondern auch über die regulären Iterations- und Alternativen-Operatoren. Ohne den geschlossenen Kalkül der Erweiterten Affixgrammatiken wirklich modifizieren zu müssen, kann dies geleistet werden, indem die regulären Konstrukte in der Hypergrammatik geradlinig in rein kontextfreie Strukturen transformiert werden (wobei die Affixebene geeignet mitzutransformieren ist).

Und unter bestimmten Bedingungen, die sich zum einen durch das zugrunde gelegte Parserverfahren ergeben und zum anderen durch das Schema, mit dem ekfG- in kfG-Regeln (unter Berücksichtigung der Affixebene) zu transformieren sind, kann sogar auf die eigentliche Transformation ganz verzichtet werden und effizienter Code generiert werden, der die syntaktische Analyse direkt auf der Skelett-ekfG aufsetzt und die Affixberechnungen in den Parsercode integriert. Der soeben genannte Technische Bericht beschreibt dies für einen zugrunde gelegten ELL(1)-Parser nach dem Verfahren des rekursiven Abstiegs. Ein entsprechender Ansatz für einen ELR-Parser, der für die Reduktionsphasen das Durchlaufen der Readback-Automaten sozusagen auscompiliert, um in den generierten Parsercode die Affixberechnungen zu integrieren, steht jedoch noch aus. Grundsätzlich kann für die Integration der Affixberechnungen möglicherweise (Jullig und DeRemer 1984) Anregungen liefern.

A Anhang: Implementierung allgemeiner LR-Parser

Alle in dieser Arbeit vorkommenden deterministischen LR-Parser sind spezielle Kellertransduktoren (im Stil von Sippu und Soisalon-Soininen (1988, 1990)) oder auch sogenannte *extended pushdown transducers*[65] (mit der Terminologie von Aho und Ullman (1972)), und ihre Implementierung müßte hier nicht eigens diskutiert werden, wenn aus der Literatur – z. B. aus so detaillierten Abhandlungen über Parsertheorie wie denen von Aho und Ullman (1972), Chapman (1987) und Sippu und Soisalon-Soininen (1988, 1990)) – verbreitet bekannt wäre, wie allgemein Exemplare dieser formalen Modelle geschickt implementiert werden können. Keines der erwähnten Werke jedoch behandelt dieses Implementierungsproblem ausführlich; statt dessen wird in der Literatur üblicherweise nur die spezielle Implementierung von kanonischen LR-Parsern durch sogenannte Parseraktions- und GOTO-Tabellen behandelt, die gerade deswegen so einfach gelingt, weil sie eine besondere Eigenschaft dieser Parser ausnutzt, nämlich – wie wir es in Abschnitt 7 bezeichnet hatten – „reduktionsregeldeterminiert" zu sein, was zu Beginn des folgenden Absatzes auch nocheinmal erläutert wird. Und natürlich weist diese Eigenschaft nicht jeder beliebige Kellertransduktor bzw. *extended pushdown transducer* auf; auch nicht generell die in dieser Arbeit vorgestellten

[65] Diese Aussage ist, streng genommen, nicht ganz korrekt, da die Originaldefinition des *extended pushdown transducers* von Aho und Ullman (1972) nur einen 1-Lookahead ermöglicht. Tatsächlich unterscheiden die Autoren auch zwischen „LR(k)-Parser"/„LR(k)-Parser-Algorithmus" und deren Formulierung als *extended pushdown transducer:*

> *„An LR(k) parser for a CFG G is nothing more than a set of rows in a large table [. . .]"* (S. 374).
>
> *„A DPDT [deterministic pushdown transducer] [. . .] can be constructed to implement the LR(k) parsing algorithm. Once we realize that we can store the lookahead string in the finite control of the DPDT, it should be evident how an extended DPDT can be constructed to implement [. . .] the LR(k) parsing algorithm.*
>
> *We leave the proofs of these observations for the Exercises."* (S. 395).

Diese Trennung können wir vermeiden, da die in der vorliegenden Arbeit verwendeten Kellertransduktoren von Sippu und Soisalon-Soininen (1988) nicht derart störend eingeschränkt sind. Viel griffiger haben wir daher auch definiert, ein LR(k)-Parser *ist* ein Kellertransduktor mit speziellen Eigenschaften. (Man beachte, daß Sippu und Soisalon-Soininen (1990) keinen allgemeinen Begriff des LR(k)-Parsers definieren, sondern nur den speziellen Begriff des kanonischen LR(k)-Parsers.)

allgemeinen LR-Parser. Nun sind deterministische allgemeine LR-Parser als spezielle Kellertransduktoren aufgrund des uniformen Aufbaus aller Lookahead-Komponenten in den Aktionen etwas einfacher zu implementieren als der allgemeine Fall eines beliebigen deterministischen Kellertransduktors. Der im folgenden beschriebene algorithmische Umgang mit den Kelleranteilen variablen Aufbaus auf den linken Seiten der Automatenaktionen kann jedoch für den allgemeinen Fall offensichtlich in analoger Manier auch auf den Umgang mit Lookahead-Anteilen variablen Aufbaus ausgedehnt werden. Wir wollen daher die Diskussion problembezogen auf die Implementierung allgemeiner LR-Parser einschränken und hier o. B. d. A. stellvertretend speziell LR(k)-Parser behandeln.

Wie auch eingangs von Abschnitt 3 ausführlich erläutert wurde, erlaubt bekanntlich in (deterministischen) kanonischen LR(k)-Parsern jeweils alleine schon das oberste Kellersymbol in Kombination mit dem k-Lookahead nicht nur zu entscheiden, *ob* zu shiften oder zu reduzieren ist, sondern sogar gegebenenfalls *nach welcher Regel* zu reduzieren ist („reduktionsregeldeterminiert"). Dasselbe ist nicht generell möglich für (deterministische) allgemeine LR(k)-Parser: Es ist abhängig von den Längen der rechten Seiten der jeweils betroffenen Grammatikregeln, wie tief hinab in den Keller hinein der Kellerkontext reicht, den die linken Seiten von Reduce-Aktionen einbeziehen. Daher ist es naheliegend, die jeweils anzuwendende Reduce-Aktion dadurch zu bestimmen, daß nur nötigenfalls und nach und nach tiefere Kellersymbole betrachtet werden. Vorausgesetzt, die in Rede stehende Grammatik erfüllt die LR(k)-Bedingung (wovon wir ausgehen wollen), kristallisiert sich dabei parallel aus einer schrumpfenden Menge noch in Frage kommender schließlich höchstens eine anwendbare Parseraktion heraus. Wie wir auch bereits in Abschnitt 7 erfahren haben, ist, auch wenn auf diese Weise bereits ein größerer (aber natürlich konstant beschränkter) Kellerabschnitt im Entscheidungsprozeß begutachtet wurde, dennoch nicht allgemein garantiert, daß überhaupt eine anwendbare Parseraktion resultiert. Mit anderen Worten ist allgemein während des *gesamten* Entscheidungsprozesses zu berücksichtigen, daß der Parser auch Fehlersituationen korrekt zu erkennen hat. Abschnitt 7 hatte sich unter anderem damit befaßt, wie eine allgemeine LR(k)-Maschine derart mit weiterer Stützinformation ausgestattet werden kann, daß der daraus konstruierte (implementierte) allgemeine LR(k)-Parser Aufwand zur korrekten Erkennung von Syntaxfehlern auf den *Beginn* des Entscheidungsprozesses beschränken kann („reduktionsdeterminiert"). Dieser Umstand ermöglicht hier eine etwas einfachere Implementierung des Parsers, worauf wir aber nicht näher eingehen wollen.

Im beschriebenen Prozeß der Entscheidungsfindung, welche der Parseraktionen anzuwenden ist, sind die zu treffenden Teilentscheidungen planbar: Zu einem vorliegenden

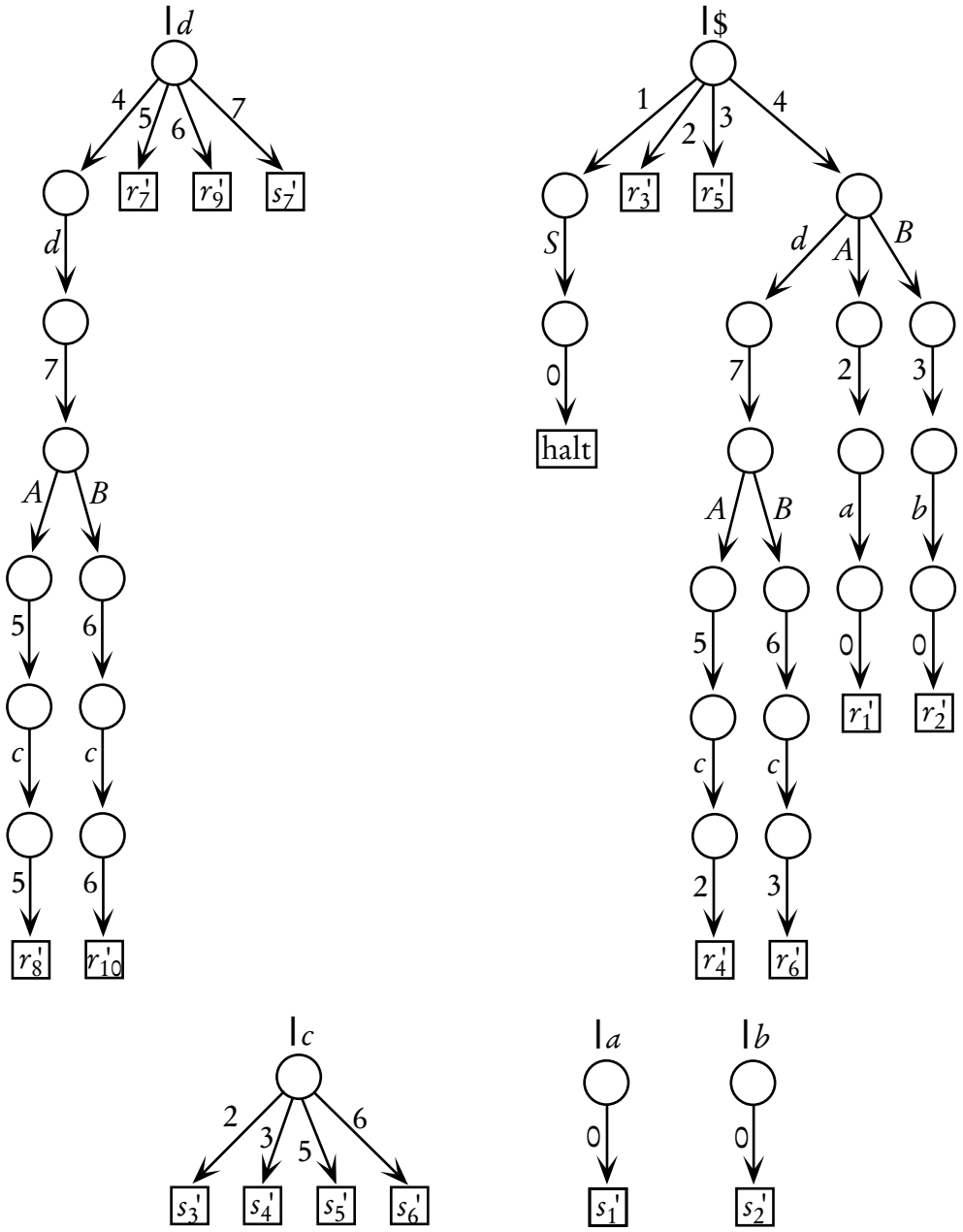


Abbildung 30: Die Suffixstrukturen von Endkonfiguration und den linken Seiten der Parseraktionen am Beispiel des minimalen allgemeinen LR(1)-Parsers zu $G = G_{ab\varepsilon}$. Der die Baumwurzel bestimmende Lookahead-Anteil und die Kantenmarkierungen an den absteigenden Pfaden identifizieren allgemein im Falle eines deterministischen Parsers eine eindeutige anzuwendende Aktion an einem Blatt.

k-Lookahead y und einem gerade betrachteten Kellerkontext φ sind genau diejenigen Aktionen noch „Kandidaten“, von deren linker Seite $\varphi \mid y$ ein Suffix ist. Verlängern wir den betrachteten Kellerkontext um ein Symbol Z zu $Z\varphi$, so schrumpft diese Kandidatenmenge zu denjenigen Reduce-Aktionen, deren linke Seite auch noch das längere Suffix $Z\varphi \mid y$ aufweisen. Lassen wir ferner derartige Suffizes die Knoten eines gerichteten Graphen bezeichnen und legen zwischen je zwei verwandten Suffizes $\varphi \mid y$ und $Z\varphi \mid y$ eine mit Z markierte Kante, so ergibt sich eine Waldstruktur, die für das Beispiel $G = G_{ab\varepsilon}$ in Abbildung 30 gezeigt ist. Die Wurzeln der darin vorkommenden Bäume geben die Lookahead-Suffizes der Form $\mid y$ an.

Jeden solchen Baum können wir auch als die Übergangsstruktur eines besonderen deterministischen „Entscheidungs“-Automaten interpretieren: Dessen Startzustand entspricht der Wurzel des Baumes und wird zu Beginn des Entscheidungsprozesses anhand des k-Lookaheads ausgewählt. Sodann liest der Automat gekellerte Symbole „rückwärts“ in den Keller hinein und tätigt die den passend markierten Kanten entsprechenden Zustandsübergänge. Anschaulich steigt er dabei in „seinem“ Baum ab. Es gelingt genau dann stets eindeutig, die anzuwendende Aktion über den erreichten Endzustand zu bestimmen, wenn der formale allgemeine LR(k)-Parser deterministisch ist.

Zwei wesentlich verschiedene Implementierungsstrategien für kanonische LR(k)- und verwandte Parser sind üblich, nämlich (i) die Interpretation von „Parseraktionstabellen“ (was letztlich auf (Knuth 1965) zurückgeht) und (ii) die Compilation der Tabelleninterpretation in ausführbaren Code (z. B. (Aho und Johnson 1974), (Pennello 1981)). Beide Wege sind prinzipiell auch für die Implementierung des Entscheidungsprozesses von allgemeinen LR(k)- und verwandten Parsern gangbar. Wir wollen uns hier jedoch nur in Ansätzen mit Tabelleninterpretationstechniken beschäftigen.

Jede einzelne der an den verschiedenen Baumknoten zu treffenden Abstiegsentscheidungen kann durch einen Vektor repräsentiert werden, welcher mit dem jeweils angetroffenen nächsttieferen Kellersymbol indiziert wird und dessen Einträge entweder (in Form einer anzuwendenden Parseraktion) die erfolgreiche Termination des Entscheidungsprozesses anweist oder (in Form eines Verweises auf dessen zugehörigen Vektor) einen Abstieg zu einem Nachfolgerknoten im steuernden Baum anzeigt oder aber, was nicht vernachlässigt werden darf, einen vorliegenden Syntaxfehler moniert.

Würden wir allerdings jeden solchen Vektor separat speichern, so stellte dies eine erhebliche Ressourcenverschwendung dar, denn die überhaupt antreffbaren Kellersymbolfolgen sind nicht beliebig aus dem Kelleralphabet zusammengesetzt, sondern haben systematischen

Aufbau, und folglich werden i. a. etliche Vektoreneinträge nachweislich nie referenziert werden. Nun wird von einer Implementierung eines aus $(\hat{M}, \mathrm{RC}_{\hat{M}})$ konstruierten allgemeinen LR(k)-Parsers $(\tilde{M}, \tau)$ in aller Regel nicht erwartet, daß sie tatsächlich präzise eine von $\tilde{M}$ auf eine vorliegende Konfiguration anwendbare Aktion r zu identifizieren in der Lage wäre, sondern lediglich, daß die von r bewirkte Konfigurationstransformation nebst Teilausgabe $\tau(r)$ korrekt nachvollzogen wird. So genügt es im Falle von Reduce-Aktionen i. a. festzustellen, (i) nach welcher Regel von G zu reduzieren ist, um zunächst die korrekte Anzahl von Symbolen zu entkellern und anschließend die linke Regelseite zu kellern, und (ii) welcher Zustand von $\hat{M}$ für die linke Regelseite noch zu kellern ist. Im Falle von Shift-Aktionen r ist lediglich festzustellen, welcher Zustand von $\hat{M}$ zusätzlich zum vorliegenden Eingabesymbol zu kellern ist. Beispielsweise sind die Aktionen s'_3 und s'_5 sowie auch s'_4 und s'_6 in Abbildung 30 nach diesem Abstraktionsschritt gleich. Und allgemein mutieren durch entsprechende Knotenverschmelzungen die beschriebenen Suffixstrukturbäume zu (in diesem Stadium nur an den Blättern) kollabierten Bäumen.

Auf deren Grundlage kann die Entscheidung über die nächste anzuwendende Parseraktion dadurch gefällt werden, daß zunächst mit dem vorliegenden Eingabe-Lookahead der Wurzelknoten des zugehörigen (kollabierten) Entscheidungsbaums ausgewählt wird. Sodann wird solange der Kellerinhalt symbolweise in den Keller hinein gelesen und mit jedem angetroffenen Symbol über eine weitere passend markierte Kante abgestiegen, wie eine solche existiert (und dann eindeutig ist). Wird ein Blatt erreicht, so ist aus diesem die für die Konfigurationstransformation notwendige Information zu ermitteln; anderenfalls liegt in der Eingabe ein Syntaxfehler vor. Die Entscheidungsbäume können in einer einzigen Matrix repräsentiert werden, in der zu jedem inneren Knoten eine mit Kellersymbolen spaltenindizierte Zeile die Abstiegsverweise bzw. Blattinformation angibt oder aber Fehlereinträge enthält.

Diese Matrix ist von erheblicher Größe: Als Zeilen- *und* Spaltenindizes treten die Zustände von $\hat{M}$ in Erscheinung, in den Spalten zusätzlich noch die Grammatiksymbole $V \cup \{\$\}$. (Die übliche Steuerungstabelle kanonischer LR(k)-Parser wird in den Spalten nur mit $V \cup \{\$\}$ indiziert.) Jedoch werden die meisten ihrer Fehlereinträge nie referenziert werden, denn wie Lemma 3.5 über die Kellerinhalte von allgemeinen LR(k)-Parsern besagt, sind diese von einem systematischen Aufbau, welcher durch die Struktur von $\hat{M}$ geprägt wird.

Beispielsweise können, wie der allgemeinen LR(1)-Maschine $(\hat{M}_\mathrm{m}, \mathrm{RC}_\mathrm{m})$ zu $G = G_{ab\varepsilon}$ aus Abbildung 4 zu entnehmen ist und auch in Abschnitt 7 bereits erläutert wurde, unterhalb von abwärts in dieser Reihenfolge angetroffenen Kellersymbolen 4, d, 7, A, 5, c als nächsttieferes Kellersymbol die $\hat{M}_\mathrm{m}$-Zustände 2 und 5 vorgefunden werden, und nichts

sonst. Zum 1-Lookahead \$ ist davon 2 eine gültige Suffixverlängerung zu $2\,c\,5\,A\,7\,d\,4\,|\,\$$, was die gesamte linke Seite der Parseraktion r_4' ist. Wird dagegen 5 angetroffen, so liegt ein Syntaxfehler vor. Zum 1-Lookahead d ist dagegen gerade umgekehrt 5 eine Suffixverlängerung, und 2 bedeutet einen Syntaxfehler.

Wir nehmen nun auch die vom Parser zu berücksichtigenden Syntaxfehlerentscheidungen (unter Nutzung von Lemma 3.5) in die kollabierten Entscheidungsbäume auf, indem „Syntaxfehlerblätter“ und entsprechende Übergänge dorthin an den notwendigen Stellen hinzugefügt werden. Abbildung 31 zeigt die entstehenden kollabierten Entscheidungsbäume für den studierten Beispielparser $(\tilde{M}_{\mathrm{m}}, \tau_{\mathrm{m}})$ zu $G = G_{ab\varepsilon}$.

Bevor wir eigentliche Kompressionstechniken für die Darstellung von derartigen kollabierten Entscheidungsbäumen als dünn besetzte Matrizen diskutieren, liegt es nahe, zuvor noch solche Optimierungen in Betracht zu ziehen, die die Struktur der Entscheidungsbäume nutzen. Glücklicherweise können wir hier auf bewährte Techniken zurückgreifen: Die Implementierung sogenannter Bottom-up-Baumautomaten, wie sie in der Arbeit von Börstel, Möncke und Wilhelm (1991) geschildert ist, führt ebenfalls auf kollabierte Entscheidungsbäume, die zunächst semantikerhaltend strukturell optimiert und sodann, als dünn besetzte Matrizen repräsentiert, mit Standardtechniken komprimiert werden können. Der folgende Abschnitt faßt die von den drei Autoren verwendete Technik zusammen.

A.1 Exkurs: Tabellenkompression für Bottom-up-Baumautomaten

Endliche Bottom-up-Baumautomaten können im generativen Compilerbau in unterschiedlichen Problemfeldern eingesetzt werden. Solche Automaten propagieren in Bottom-up-Manier Zustände durch die Knoten eines Baumes, und zwar je eine Generation weit pro Schritt. In (Wilhelm und Maurer 1992) werden sie beispielsweise für die Gewinnung von globaler Attributabhängigkeitsinformation eingesetzt und besonders nutzbringend für die Generierung effizienter Code-Selektoren. Zu letzterem Zweck wird „optimale“ Code-Auswahl als ein auf (mehrdeutigen) Baumgrammatiken gestütztes Parsing-Problem auf den Operatorbäumen der Zwischensprache verstanden mit der zusätzlichen Forderung der Minimalität der kumulativen Kosten jeder Regelanwendung, wobei die Regeln der eingesetzten Baumgrammatik die Zielmaschinenarchitektur wiedergeben. Leider sind die angesprochenen Passagen in (Wilhelm und Maurer 1992), dem fast zeitgleich erschienenen, noch tiefergehenden Konferenzbeitrag (Ferdinand, Seidl und Wilhelm 1992) und auch dessen späterer Journal-Veröffentlichung (Ferdinand, Seidl und Wilhelm 1994) nicht ganz fehlerfrei, alle drei Arbeiten verweisen jedoch auf einschlägige Literatur zu Baumauto-

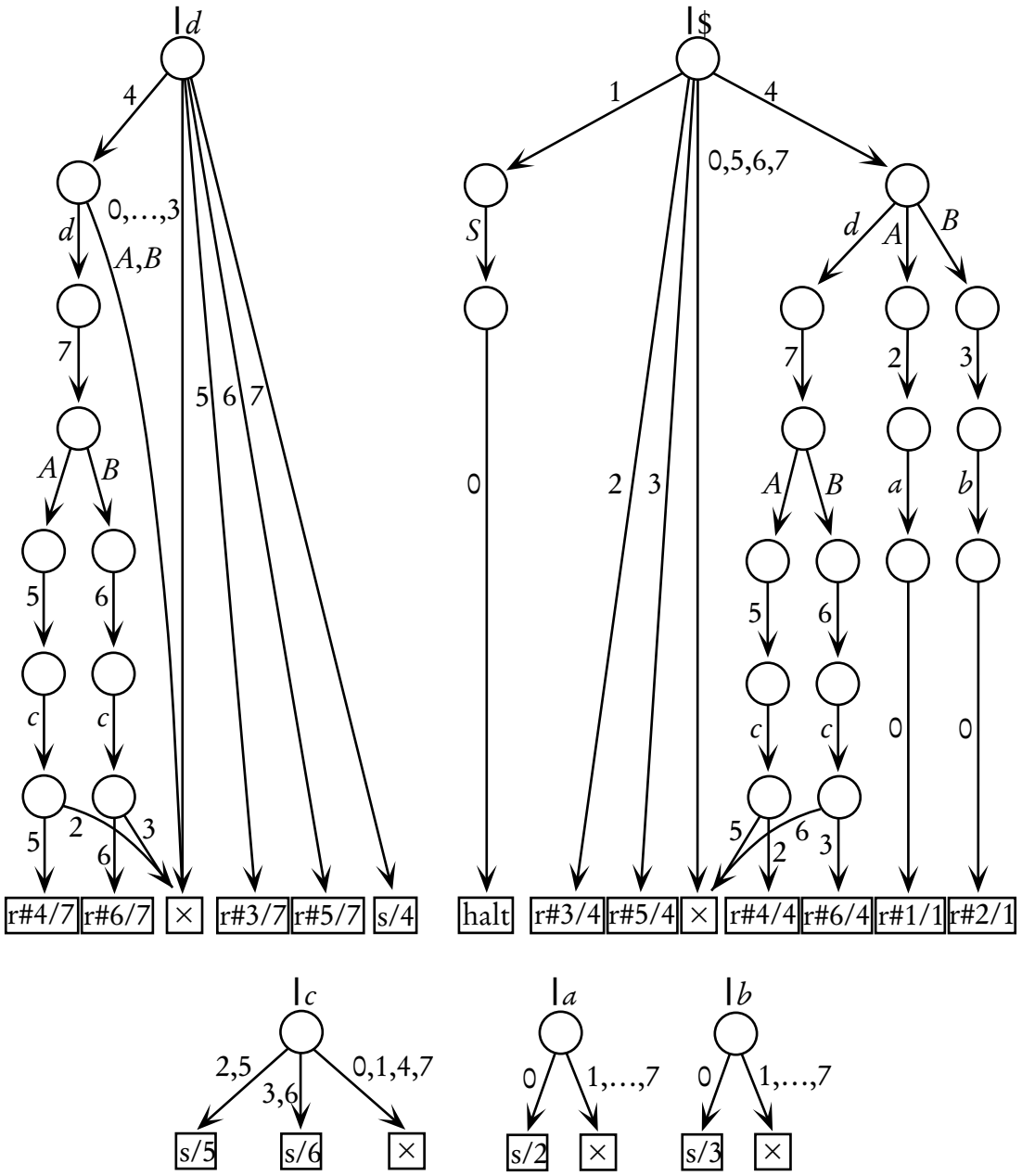


Abbildung 31: Die Entscheidungsbäume für den minimalen allgemeinen LR(1)-Parser zu $G = G_{ab\varepsilon}$ berücksichtigen gezielt auch Syntaxfehlersituationen und bestimmen an den Blättern nur noch die für die Implementierung der Parseraktionen notwendige Information: „halt“, „$\times$“ (Syntaxfehler), „s/q“ (Shift) und „r #p/q“ (Reduce nach Regel #p); q ist ein neu zu kellernder Zustand der LR(1)-Maschine. Regelabkürzungen: #1 : $S \to a\,A$, #2 : $S \to b\,B$, #3 : $A \to \varepsilon$, #4 : $A \to c\,A\,d$, #5 : $B \to \varepsilon$, #6 : $B \to c\,B\,d$.

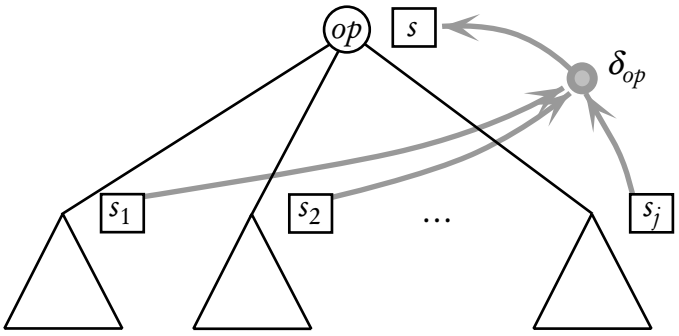


Abbildung 32: Visualisierung des Informationsflusses bei einem Arbeitsschritt eines Bottom-up-Baumautomaten an einem mit *op* markierten Baumknoten: Der dekorierende Zustand $s = \delta_{op}(s_1, s_2, \ldots, s_j)$ dieses Knotens wird aus den dekorierenden Zuständen seiner Nachfolger *op*-spezifisch bestimmt.

maten. Für uns ist hier die geschickte Implementierung eines einzelnen Arbeitsschritts eines Bottom-up-Baumautomaten von Interesse.

Grundlage ist ein Baum, dessen mit Operatoren eines Operatorenalphabets markierte Knoten mit Zuständen eines Zustandsalphabets konsistent gemäß fester Berechnungsvorschriften zu dekorieren sind und der als wohlgeformt vorausgesetzt wird in dem Sinne, daß alle Operatoren von jeweils festgelegter Stelligkeit sind und die Anzahlen der jeweiligen Nachfolgerknoten diesen Stelligkeiten entsprechen. In einem Berechnungsschritt eines endlichen Bottom-up-Baumautomaten an einem mit *op* markierten Knoten ist dessen Zustandsdekoration s aus den vorliegenden Zuständen $s_1, \ldots, s_j$ seiner j Nachfolger *op*-spezifisch zu bestimmen: $s = \delta_{op}(s_1, \ldots, s_j)$. Abbildung 32 visualisiert den Informationsfluß.[66] Eine erste Implementierungsmöglichkeit dieses Berechnungsschritts ist die Repräsentation von δ_{op} über eine j-dimensionale Matrix. Dabei kommen *garantiertermaßen* nicht alle, oft sogar nur wenige Kombinationen der $s_1, \ldots, s_j$ vor, und die j-dimensionale Matrix ist entspre-

[66] Aus (Börstel, Möncke und Wilhelm 1991).

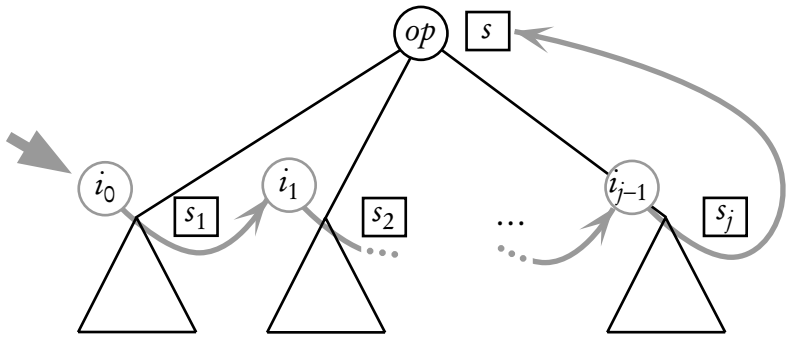


Abbildung 33: Schrittweise Berechnung von $s = \delta_{op}(s_1, s_2, \ldots, s_j)$ durch einen „horizontalen Automaten“ für *op*.

chend dünn besetzt. Für Kompressionszwecke erweist sich daher eine Modellierung in Form etwa von j zweidimensionalen Tabellen als günstiger, mit der die Berechnung von s aus $s_1, \ldots, s_j$ argumentweise organisiert wird.

Diese Tabellen können als die Repräsentation eines „horizontalen Automaten“ für *op* angesehen werden, welcher in einem Links-rechts-Gang durch die Nachfolger des *op*-markierten Knotens Übergänge gemäß der dort vorgefundenen dekorierenden Zustände in neu eingeführte Zwischenzustände tätigt und so schließlich in j Schritten den Ergebniszustand s am übergeordneten Knoten bestimmt. Abbildung 33 skizziert die Vorgehensweise.[67]

Gleichfalls können diese Tabellen als Repräsentationen eines *op*-Entscheidungsbaums der Tiefe j interpretiert werden, dessen Knoten entweder (an den inneren) mit den neu eingeführten Zwischenzuständen des horizontalen Automaten bzw. (an den Blättern) den dekorierenden Ergebniszuständen des Baumautomaten-Berechnungsschritts markiert sind und dessen Kanten gerade mit möglichen Zustandsdekorierungen $s_1, \ldots, s_j$ der zu durchlaufenden j Unterbaumwurzeln so markiert sind, daß für einen Kantenzug von der Wurzel des Entscheidungsbaums zu einem seiner Blätter, welches mit s markiert sei, stets $s = \delta_{op}(s_1, \ldots, s_j)$ gilt. Gleich markierte Blätter werden zusammengelegt, der Entscheidungsbaum zu *op* ist also initial an den Blättern kollabiert. Er stellt die (von links nach rechts) argumentweise Berechnung des Funktionswerts $s = \delta_{op}(s_1, \ldots, s_j)$ und gleichzeitig die Übergangsstruktur des horizontalen Automaten für einen mit *op* markierten Baum-

[67] Aus (Börstel, Möncke und Wilhelm 1991).

knoten dar.

Ebenso wie die oben erwähnte j-dimensionale Matrix für die Berechnung der δ_{op}-Werte aus Argumenten $s_1,\ldots,s_j$ meist dünn besetzt ist, sind dies auch die alternativen j zweidimensionalen Tabellen: Auch auf sie wird garantiert nur mit Eingangswerten $s_1,\ldots,s_j$ zugegriffen, für die $\delta_{op}(s_1,\ldots,s_j)$ definiert ist, was dann für ihre komprimierte Speicherung genutzt wird.

Zuvor jedoch werden von Börstel, Möncke und Wilhelm auf jeden horizontalen Automaten semantikerhaltende Verkleinerungstechniken (ggf. wiederholt) angesetzt:

(a) Ein horizontaler Automat zu einem Operator *op* mit Stelligkeit ≥ 1, der nur ein einziges Ergebnis s berechnet, wird ersetzt durch einen Automaten, der nur noch einen Start- und gleichzeitigen Endzustand s besitzt.

(b) Ein Zwischenzustand i wird mit einem Endzustand s identifiziert, wenn alle i verlassenden Übergänge zu s führen. Im visualisierenden Entscheidungsbaum wird ein innerer Knoten, dessen ausgehende Kanten alle zu demselben Blatt führen, eliminiert, indem die einlaufenden Kanten zu diesem Blatt umgelegt und damit Entscheidungswege gekürzt werden.

(c) Zwei verschiedene nach gleicher Schrittzahl des horizontalen Automaten angenommene Zwischenzustände (und entsprechende inzidente Übergänge) werden zusammengelegt, wenn mit gleichen Eingaben stets gleiche Zustände erreicht werden. Im Entscheidungsbaum werden entsprechend zwei innere Knoten derselben Tiefe zusammengelegt, wenn die Kantenmarkierungen der auslaufenden Kanten beider Knoten gleich sind und gleich markierte Kanten stets gleiche Knoten erreichen.

Für ein kleines handsimuliertes Beispiel und für Zahlenmaterial, das Aufschluß über die Wirkung dieser Verkleinerungsschritte zu einigen Fällen aus unterschiedlichen Anwendungsfeldern von Baumautomaten gibt, muß auf (Börstel, Möncke und Wilhelm 1991) verwiesen werden. Die resultierenden Repräsentationen der verkleinerten horizontalen Automaten (bzw. ihrer Veranschaulichung als gekürzte und kollabierte Entscheidungsbäume) in Form mehrerer dünn besetzter zweidimensionaler Tabellen werden von den Autoren dann mit Standardkompressionstechniken platzökonomischer gespeichert. Zum einen stellen Börstel, Möncke und Wilhelm ein Zeilenversatz-Schema vor *(row displacement scheme),* für das als wesentliche Literaturquelle (Tarjan und Yao 1979) genannt wird. Zum anderen wird auch ein Zeilen-Spalten-Schema *(row column scheme)* diskutiert, das

von Dencker, Dürre und Heuft (1984) für die kompakte Speicherung von Parsertabellen vorgeschlagen wird. Letztere Kompressionstechnik produziert komprimierte Ergebnistabellen, die in der Regel kleiner sind als die mit Zeilenversatz erreichten; der zu zahlende Preis ist der Verlust des direkten Zugriffs. Wir gehen auf beide Kompressionstechniken hier nicht mehr näher ein, sondern verweisen auf (Börstel, Möncke und Wilhelm 1991).

A.2 Adaption der Kompressionstechniken für Baumautomaten an die Implementierung allgemeiner LR(k)-Parser

Die Entscheidungsbäume, die wir im vorangehenden Abschnitt als die Übergangsstrukturen „horizontaler Automaten" für die Durchführung eines Baumautomaten-Berechnungsschritts kennengelernt haben, weisen (unoptimiert) große Ähnlichkeiten zu denjenigen, die wir als die Steuerstrukturen für die Bestimmung der nächsten anzuwendenden Aktion eines zu implementierenden allgemeinen LR(k)-Parsers entwickelt haben: In beiden Fällen handelt es sich um endliche, unoptimiert zunächst nur an den Blättern kollabierte Bäume mit ausgezeichneter Wurzel, deren Kanten mit Elementen einer endlichen Menge markiert sind, und zwar so, daß die von einem beliebigen inneren Knoten ausgehenden Kanten eine eindeutige Markierung tragen. Unterschiedlich ist:

- Im Gegensatz zu unseren Entscheidungsbäumen für Parseraktionen sind in unoptimierten Entscheidungsbäumen für Baumautomaten-Berechnungsschritte die Kantenmarkierungen (die Eingabealphabete horizontaler Automaten) mit den möglichen Blattmarkierungen (den möglichen Endzuständen horizontaler Automaten) identisch. Für die Tabellenrepräsentation dieser Bäume bedeutet dies, daß diejenigen möglichen Tabelleneinträge, die Blätter anzeigen, gleichzeitig auch die möglichen Spaltenindizes der Tabellen sind – ein Umstand, der in den verschiedenen eingesetzten Kompressionstechniken keine Rolle spielt.
- In dem unoptimierten Entscheidungsbaum für den Berechnungsschritt eines Baumautomaten an einem mit op markierten Knoten sind alle Kantenzüge von der Wurzel zu den Blättern von der gleichen Länge: der Stelligkeit von op. Auch dieses ist nicht der Fall in Entscheidungsbäumen für Parseraktionen: Hier hängen die Blattiefen von den Längen der linken Seiten der verschiedenen Parseraktionen ab. Der Umstand der einheitlichen Blattiefen spielt jedoch in den verschiedenen eingesetzten Kompressionstechniken ebenfalls keine Rolle, die Verkleinerungsschritte (a) und (b) aus dem vorigen Abschnitt zerstören diese Eigenschaft sogar.

Auch die gemeinsame Eigenschaft, daß Kollabierungen nur an den Blättern der Entscheidungsbäume vorkommen, wird übrigens von den Kompressionstechniken nicht weiter genutzt, sondern mit dem Verkleinerungsschritt (a) aus dem vorigen Abschnitt zerstört.

Die vorgestellten Kompressionstechniken für die Implementierung von Baumautomaten-Berechnungsschritten sind somit auf die Implementierung von allgemeinen LR(k)-Parsern übertragbar, ja sogar noch verfeinerbar. Nocheinmal zusammenfassend, gelangen wir zu folgender Methode zur strukturellen Verkleinerung unserer Entscheidungsbäume für Parseraktionen:

(a) Entscheidungsbäume mit nur einem einzigen Blatt werden ersetzt durch solche trivialen Entscheidungsbäume, die nur noch aus diesem Blatt als gleichzeitige Wurzel bestehen.

(b) Ein innerer Knoten, dessen ausgehende Kanten sämtlich zu demselben Blattknoten führen, kann mitsamt der auslaufenden Kanten eliminiert werden, nachdem alle zu ihm verlaufenden Kanten zu diesem Blatt umgelegt wurden.

(c) Zwei verschiedene innere Knoten derselben Tiefe werden zusammengelegt, wenn die Mengen der Markierungen der auslaufenden Kanten der beiden Knoten gleich sind und gleich markierte Kanten jeweils gleiche Knoten erreichen.

Verfeinern können wir die Technik von Börstel, Möncke und Wilhelm noch, indem wir sie gleichzeitig auf alle vorhandenen Entscheidungsbäume anwenden. Verschiedene Entscheidungsbäume können nämlich durchaus gemeinsame Blätter aufweisen, da unsere abstrakten Blattmarkierungen unabhängig von den die Entscheidungsbäume auswählenden Lookaheads sind. In der Summe können auf diese Weise zum Teil ganze Entscheidungsbäume für verschiedene Lookaheads zusammengelegt werden, was in der Praxis sogar häufig vorkommen wird, da mit Programmiersprachengrammatiken oft Situationen auftreten, in denen für ganze Gruppen „gleichrangiger“ Lookaheads (z. B. Multiplikationsoperatoren, Additionsoperatoren in arithmetischen Ausdrücken) bis auf Lookahead gleiche Aktionengruppen anwendbar sind.

Leider stellt sich heraus, daß das vorliegende Fallbeispiel der Entscheidungsbäume aus Abbildung 31 für den studierten Beispielparser $(\tilde{M}_{\mathrm{m}}, \tau_{\mathrm{m}})$ zu $G = G_{ab\varepsilon}$ nicht reichhaltig genug ist, um daran die vorgestellten Verkleinerungstechniken erschöpfend demonstrieren zu können.[68] Lediglich im Entscheidungsbaum zum Lookahead \$ können

[68] Dies ist auf die einfache Struktur der Grammatik $G_{ab\varepsilon}$ zurückzuführen. Beispielgrammatiken, die zu komplexeren Entscheidungsbäumen führen, sprengen hier jedoch den Rahmen.

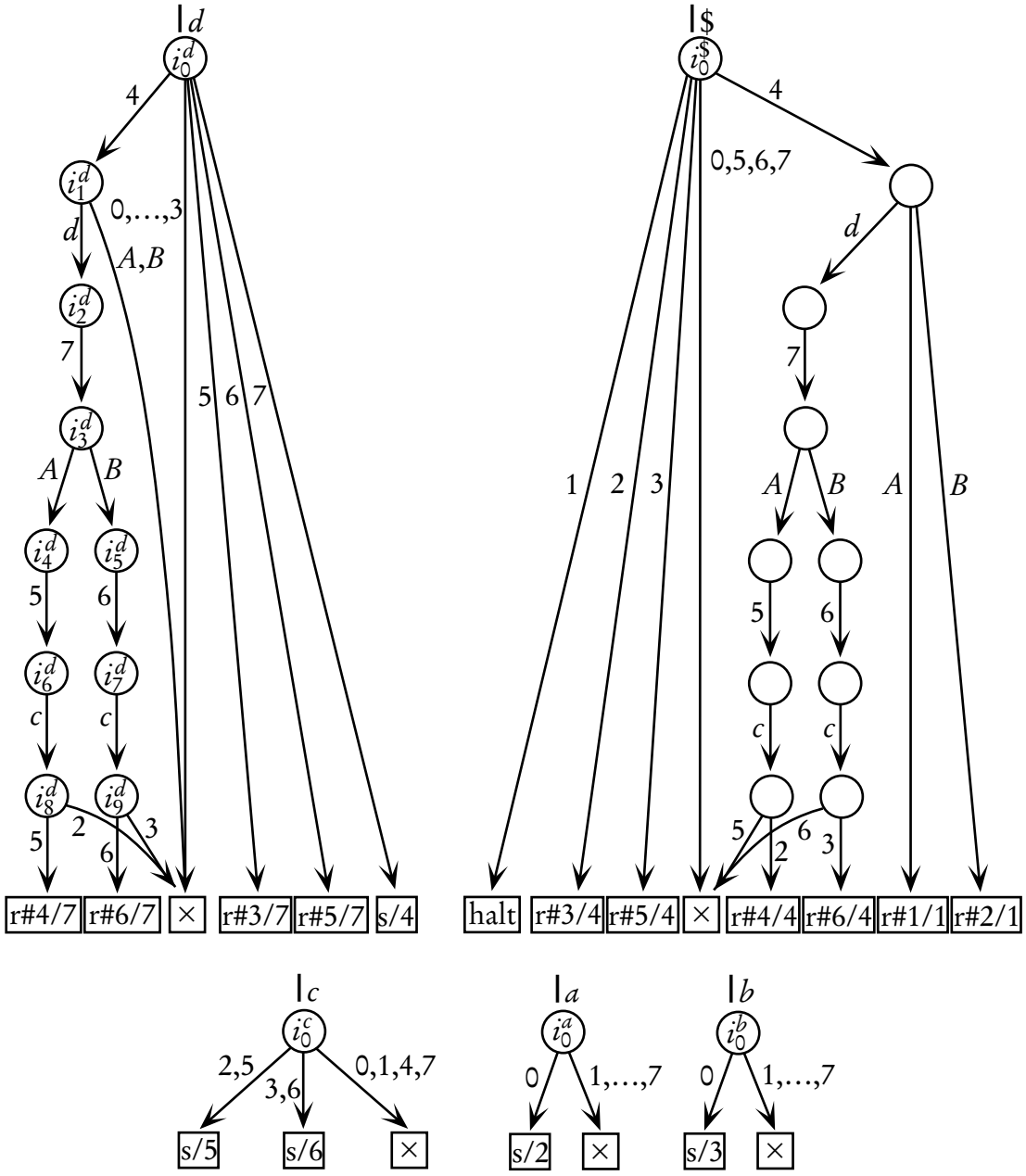


Abbildung 34: Die verkleinerten Entscheidungsbäume für den minimalen allgemeinen LR(1)-Parser zu $G = G_{ab\varepsilon}$. (Knotenverschmelzungen sind hier nicht möglich.)

mehrfach innere Knoten eliminiert und damit Entscheidungswege verkürzt werden (Regel (b)). Das Ergebnis zeigt Abbildung 34. Zwar kommen auch in diesem kleinen Fallbeispiel durchaus gemeinsame Blätter verschiedener Entscheidungsbäume vor, Zusammenlegungsmöglichkeiten für weitere Knoten aus unterschiedlichen Entscheidungsbäumen ergeben sich allerdings nicht mehr.

In der Abbildung sind im Entscheidungsbaum zum Lookahead d die inneren Knoten mit Hilfsbezeichnungen $i_0^d, i_1^d, \ldots, i_9^d$ versehen worden. Diese indizieren Zeilen einer unkomprimierten, zweidimensionalen, dünn besetzten Haupttabelle, wo eine mit i_l^d indizierte Zeile die Abstiege am i_l^d-markierten Baumknoten angibt (Abbildung 35), wobei wir die Tabellenrepräsentationen aller Entscheidungsbäume zu dieser Haupttabelle zusammengezogen haben. Diese Tabellendarstellung kann mit dem angedeuteten Zeilenversatz- oder dem Zeilen-Spalten-Schema komprimiert werden, was hier nicht mehr demonstriert wird.

Selbstverständlich können wir das geschilderte Verfahren zur Implementierung von Entscheidungsbäumen für Parseraktionen auch auf kanonische LR(k)-Parser anwenden. Repräsentieren wir hier die nichtterminalen Übergänge der zugrunde liegenden LR(k)-Maschine nicht – wie in Abbildung 35 vorgeschlagen – mittels Blattmarkierungen „r #p/q“ (als die Information q), sondern in einer separaten Tabelle, so daß sich die Blattmarkierungen zu „r #p“ vereinfachen, so sind alle Entscheidungsbäume mit den vorgestellten Verkleinerungstechniken stets zur Höhe 1 verkürzbar, da kanonische LR(k)-Parser reduktionsregeldeterminiert sind und somit alle Entscheidungsbaumäste unterhalb der Wurzeln entscheidungsfrei sind. Weiterhin müssen auf dem Keller nie Grammatiksymbole inspiziert werden, weswegen entsprechende Tabellenspalten entfallen können, und wir gelangen, bezogen auf unkomprimierte Parsertabellen, insgesamt mehr oder weniger zur bekannten Standardimplementierung kanonischer LR(k)-Parser.

innere Baumknoten	Kellersymbole															
	0	1	2	3	4	5	6	7	S	A	B	\$	a	b	c	d
i_0^d	×	×	×	×	i_1^d	a_1	a_2	s/4								
i_1^d										×	×					i_2^d
i_2^d								i_3^d								
i_3^d										i_4^d	i_5^d					
i_4^d						i_6^d										
i_5^d							i_7^d									
i_6^d															i_8^d	
i_7^d															i_9^d	
i_8^d			×			a_3										
i_9^d				×			a_4									
⋮								…								

Index	a_1	a_2	a_3	a_4
Regel	#3	#5	#4	#6
q	7	7	7	4

Regel	#1	#2	#3	#4	#5	#6
linke Seite	S	S	A	A	B	B
Länge rechter Seite	2	2	0	3	0	3

k-Lookahead	$\|a$	$\|b$	$\|c$	$\|d$	$\|\$$
Entscheidungsbaumwurzel	i_0^a	i_0^b	i_0^c	i_0^d	$i_0^\$$

Abbildung 35: Die wesentlichen unkompaktifizierten Tabellen zur Steuerung der Entscheidungsfindung für anzuwendende Parseraktionen. In der oberen zweidimensionalen Tabelle, die die gekürzten Entscheidungsbäume repräsentiert, sind nur die Einträge für den Baum zum Lookahead d (mit Anfangsknotenbenennung i_0^d) aufgeführt. Nicht gefüllte Einträge werden nie referenziert. Einträge × bedeuten vorliegende Syntaxfehler. Einträge a_j weisen Reduce-Aktionen an und stellen Indizes in die zweite Tabelle dar, der die erkannte Grammatikregel und der neu zu kellernde Zustand zu entnehmen sind. Das ebenfalls zu kellernde Nichtterminal der linken Regelseite und die für die vorherige Entkellerung benötigte Länge der rechten Regelseite ist der dritten Tabelle zu entnehmen. Die vierte Tabelle weist den k-Lookaheads die Anfangsknotenbenennung ihres Entscheidungsbaums zu. Die unvollständig besetzte zweidimensionale Tabelle kann auf unterschiedliche Weise linearisiert und kompaktifiziert werden.

Literatur

Aho, A. V. und S. C. Johnson (1974). LR parsing. *ACM Computing Surveys 6*, S. 99–124.

Aho, A. V. und J. D. Ullman (1972). *The Theory of Parsing, Translation and Compiling. Vol.1: Parsing*. Englewood Cliffs, N.J.: Prentice-Hall.

Anderson, T. (1972). *Syntactic Analysis of LR(k) Languages*. Ph. D. thesis, University of Newcastle upon Tyne.

Berry, G. und R. Sethi (1986). From regular expressions to deterministic automata. *Theoretical Computer Science 48*, S. 117–126.

Book, R., S. Even, S. Greibach und G. Ott (1971). Ambiguity in graphs and expressions. *IEEE Transactions on Computers C20*, S. 149–153.

Börstel, J., U. Möncke und R. Wilhelm (1991). Table compression for tree automata. *ACM Transactions on Programming Languages and Systems 13*(2), S. 295–314.

Brauer, W. (1984). *Automatentheorie*. Teubner.

Brüggemann-Klein, A. (1993). Regular expressions into finite automata. *Theoretical Computer Science 120*, S. 197–213.

Brzozowski, J. A. (1964). Derivatives of regular expressions. *Journal of the ACM 11*, S. 481–494.

Celentano, A. (1981). An LR parsing technique for extended context-free grammars. *Computer Languages 6*, S. 95–107.

Chapman, N. P. (1984). LALR(1,1) parser generation for regular right part grammars. *Acta Informatica 21*, S. 29–45.

Chapman, N. P. (1987). *LR Parsing: Theory and Practice*. Cambridge University Press.

Demuth, J., S. Weber, S. Kannapinn und M. Kröplin (1997). Echte Compilergenerierung – Effiziente Implementierung einer abgeschlossenen Theorie. Forschungsbericht des Fachbereichs Informatik 1997/6, Technische Universität Berlin, Deutschland.

Dencker, P., K. Dürre und J. Heuft (1984). Optimization of parser tables for portable compilers. *ACM Transactions on Programming Languages and Systems 6*(4), S. 546–572.

Deransart, P., M. Jourdan und B. Lorho (1988). *Attribute Grammars*, Band 323 von *Lecture notes in computer science*. Springer.

DeRemer, F. L. (1969). *Practical Translators for LR(k)-languages*. Ph. D. thesis, Massachusetts Institute of Technology, Cambridge, Mass.

DeRemer, F. L. (1971). Simple LR(*k*) grammars. *Communications of the ACM 14*, S. 453–460.

DeRemer, F. L. (1974). Lexical analysis. In F. L. Bauer und J. Eickel (Hrsg.), *Compiler construction, an advanced course*, Band 21 von *Lecture notes in computer science*, S. 109–120. Springer.

DeRemer, F. L. und T. Pennello (1982). Efficient computation of LALR(1) look-ahead sets. *ACM Transactions on Programming Languages and Systems 24*(4), S. 615–649.

Earley, J. C. (1970). An efficient context-free parsing algorithm. *Communications of the ACM 13*, S. 94–102.

Ferdinand, C., H. Seidl und R. Wilhelm (1992). Tree automata for code selection. In R. Giegerich und S. L. Graham (Hrsg.), *Code Generation—Concepts, Tools, Techniques: Proceedings of the International Workshop on Code Generation*, Dagstuhl, Germany, S. 30–50.

Ferdinand, C., H. Seidl und R. Wilhelm (1994). Tree automata for code selection. *Acta Informatica 31*(8), S. 741–760.

Fortes Gálvez, J. (1992). Generating LR(1) parsers of small size. In P. Pfahler und U. Kastens (Hrsg.), *Proceedings of the 4th International Conference on Compiler Construction, CC '92*, Paderborn, Germany, S. 16–29.

Fortes Gálvez, J. (1994). A note on a proposed LALR parser for extended context-free grammars. *Information Processing Letters 50*, S. 303–305.

Glushkov, V. M. (1961). The abstract theory of automata. *Russian Mathematical Surveys 16*, S. 1–53.

Gries, D. (1971). *Compiler Construction for Digital Computers*. New York: Wiley.

Gries, D. (1973). Describing an algorithm by Hopcroft. *Acta Informatica 2*, S. 97–109.

Grosch, J. (1990). Lalr—A generator for efficient parsers. *Software—Practice and Experience 20*(11), S. 1115–1135.

Harrison, M. (1965). *Introduction to Switching and Automata Theory*. McGraw-Hill.

Heilbrunner, S. (1979). On the definition of ELR(k) and ELL(k) grammars. *Acta Informatica 11*, S. 169–176.

Heilbrunner, S. (1981). A parsing automata approach to LR theory. *Theoretical Computer Science 15*, S. 117–157.

Hopcroft, J. (1971). An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, S. 189–196. New York: Academic Press.

Jensen, K. und N. Wirth (1974). *PASCAL: User Manual and Report*, Band 18 von *Lecture notes in computer science*. Springer.

Johnson, S. C. (1978). Yacc: Yet Another Compiler-Compiler. Computing Science Report 32, Bell Laboratories, Murray Hill, N. J.

Jullig, R. K. und F. DeRemer (1984). Regular right-part attribute grammars. *Proceedings of the SIGPLAN 1984 Symposium on Compiler Construction, SIGPLAN Notices 19*, S. 171–178.

Klein, E. und M. Martin (1989). The parser generating system PGS. *Software—Practice and Experience 19*(11), S. 1015–1028.

Knuth, D. E. (1965). On the translation of languages from left to right. *Information and Control 8*(6), S. 607–639.

Knuth, D. E. (1968). Semantics of context-free languages. *Mathematical Systems Theory 2*(2), S. 127–145.

LaLonde, W. R. (1975). *Practical LR analysis of regular right part gramars*. Ph. D. thesis, University of Waterloo, Waterloo, Ontario, Canada.

LaLonde, W. R. (1977). Regular right part grammars and their parsers. *Communications of the ACM 20*(10), S. 731–741.

LaLonde, W. R. (1979). Constructing LR parsers for regular right part grammars. *Acta Informatica 11*, S. 177–193.

Langmaack, H. (1971). Application of regular canonical systems to grammars translatable from left to right. *Acta Informatica 1*, S. 111–114.

Lee, G.-O. und D.-H. Kim (1997). Characterization of extended LR(k) grammars. *Information Processing Letters 64*, S. 75–82.

Madsen, O. L. und B. B. Kristensen (1975). On extended context free grammars and LR-parsing. Technical Report DAIMI PB-53, Department of Computer Science, University of Aarhus, Denmark.

Madsen, O. L. und B. B. Kristensen (1976). LR-parsing of extended context free grammars. *Acta Informatica 7*, S. 61–73.

Nakata, I. und M. Sassa (1986). Generation of efficient LALR parsers for regular right part grammars. *Acta Informatica 23*, S. 149–162.

Pager, D. (1977). A practical general method for constructing LR(*k*) parsers. *Acta Informatica 7*, S. 249–268.

Pennello, T. J. (1981). Very fast LR parsing. *Proceedings of the SIGPLAN 1986 Symposium on Compiler Construction, SIGPLAN Notices 21*, S. 145–151.

Purdom, P. (1974). The size of LALR(1) parsers. *BIT 14*, S. 326–337.

Purdom, P. W. und C. A. Brown (1981). Parsing extended LR(*k*) grammars. *Acta Informatica 15*, S. 115–127.

Räihä, K.-J., M. Saarinen, M. Sarjakoski, S. Sippu, S.-S. E. und M. Tienari (1983). Revised report on the compiler writing system HLP. Department of Computer Science Report A-1983-1, University of Helsinki, Helsinki.

Reiser, M. und N. Wirth (1992). *Programming in Oberon: Steps beyond Pascal and Modula*. New York: ACM Press.

Shin, H.-C. und K.-M. Choe (1993). An improved LALR(*k*) parser generation for regular right part grammars. *Information Processing Letters 47*, S. 123–129.

Sippu, S. und E. Soisalon-Soininen (1988). *Parsing Theory. Vol.1: Languages and Parsing*, Band 15 von *EATCS Monographs on Theoretical Computer Science*. Springer.

Sippu, S. und E. Soisalon-Soininen (1990). *Parsing Theory. Vol.2: LR(k) and LL(k) Parsing*, Band 20 von *EATCS Monographs on Theoretical Computer Science*. Springer.

Tarjan, R. E. (1972). Depth first search and linear graph algorithms. *SIAM Journal on Computing 1*, S. 146–160.

Tarjan, R. E. und A. C.-C. Yao (1979). Storing a sparse table. *Communications of the ACM 22*(11), S. 606–611.

Ukkonen, E. (1983). Lower bounds on the size of deterministic parsers. *Journal of Computer and System Sciences 26*, S. 153–170.

Ukkonen, E. (1985). Upper bounds on the size of LR(*k*) parsers. *Information Processing Letters 20*, S. 99–103.

Watt, D. A. (1991). *Programming Language Syntax and Semantics*. International series in computer science. Prentice-Hall.

Whetherell, C. und A. Shannon (1981). LR – automatic parser generator and LR(1) parser. *IEEE Transactions on Software Engineering SE-7*, S. 274–278.

Wilhelm, R. und D. Maurer (1992). *Übersetzerbau: Theorie, Konstruktion, Generierung*. Springer.

Ziadi, D. (1996). Regular expression for a language without empty word. *Theoretical Computer Science 163*, S. 309–315.

Lebenslauf

Sönke Kannapinn, geboren am 12.5.1964 in Essen

Schule

1982	Abitur am Gymnasium Luisenschule in Mülheim an der Ruhr

Ausbildung und Berufstätigkeit (1)

1982–1984	Ausbildung zum Technischen Assistenten Informatik am Bildungszentrum für informationsverarbeitende Berufe e. V. in Paderborn
1984–1989	Programmierassistent und Systemprogrammierer bei der Firma Nixdorf Microprocessing Engineering GmbH in Berlin

Universität

1986–1993	Studium der Informatik an der Technischen Universität Berlin
1989–1993	Tätigkeit als Tutor an der Lehreinheit „Programmiersprachen und Compiler“ des Fachbereichs Informatik der Technischen Universität Berlin
1993–1998	Wissenschaftlicher Mitarbeiter mit Lehraufgaben an der Lehreinheit „Programmiersprachen und Compiler“ des Fachbereichs Informatik der Technischen Universität Berlin

Berufstätigkeit (2)

1998–2000	„Berater“ bei der Firma software design & management AG in Frankfurt am Main
2000–	„Systems Engineer“ bei der Firma Wincor Nixdorf GmbH & Co. KG in Berlin