2 MO_LEX, EINE REPRÄSENTATIONSSPRACHE FÜR *FST*s

2.1 Die Syntax von mo_lex

2.1.1 Die Syntax von mo_lex-Theorien[1]

Eine **mo_lex**-Theorie besteht aus zwei endlichen Alphabeten (Aus- und Eingabe-Alphabet (V_A und V_E)) und einer Folge von Sätzen. Sätze sind Abkürzungskonventionen[2] oder Constraints[3]. Jedes Constraint hat die Form "Spezifikation --> String". "String" ist eine beliebige Kette aus V_A*. Spezifikationen setzen sich aus einem (möglicherweise leeren) linken Kontext (*LK*), dem Spezifikations-Kern (*K*) und einem (möglicherweise leeren) rechten Kontext (*RK*) zusammen. *K* ist ein *Endlicher Regulärer Ausdruck* (*ERA*), *LK* ein *Links Negierter Regulärer Ausdruck* (*LRA*) und *RK* ein *Rechts Negierter Regulärer Ausdruck* (*RRA*). Die Syntax von *ERA*s, *LRA*s und *RRA*s ist bis auf das Fehlen von Stern und Kreuz-Operator (*, +) und die Verwendung spezifischer Negations-Operatoren ("~R, ~L") identisch zu der *Regulärer Ausdrücke* in **(f)lex** und wird in **2.1.2** ausführlicher beschrieben. (1) enthält die Syntax von **mo_lex** in Backus-Naur-Normalform:

(1)	Theorie	-->	Satz (Theorie)
	Satz	-->	Abkürzungskonvention
		|	Constraint
	Constraint	-->	Spezifikation "-->" String
	Spezifikation	-->	(LK) K (RK)
	LK	-->	LRA"\"
	RK	-->	"/" RRA
	K	-->	ERA
	Abkürzungskonvention	-->	"&LK" L_Def
		|	"&RK" R_Def
		|	"&DEF" Name Def
	L_Def	-->	L_Def L_Def
		|	LRA
		|	"{"Name"}"[4]

[1] Der Begriff "Theorie" wird hier wie für DATR (Evans & Gazdar:1990, 1996) üblich verwendet.

[2] Zur Verwendung von Abkürzungs-Konventionen s. **§1/1**. Hier nehme ich außer Abkürzungs-Konventionen für linke Kontexte ("&LK") auch solche für rechte Kontexte an ("&RK") und rechne dazu auch *reguläre Definitionen* ("&DEF").

[3] Ich spreche von "Constraints" anstatt von "Regeln", da **mo_lex**-Constraints anders als etwa kontextsensitive Regeln kein erlei Bezug auf Zwischenrepräsentationen erlauben. Damit sind sie deklarative Beschränkungen über Relationen.Daß diese Beschränkungen verletzbar sind, teilen sie mit Theorien wie Optimality Theory (Prince & Smolensky 1993, s. **§1/4**).

[4] Wie in **(f)lex** können *reguläre Definitionen* ineinander eingebettet werden. Die Interpretation als Abkürzungskonvention ist nur möglich, solange keine Definition direkt oder indirekt auf sich selbst zugreift.

```
R_Def       -->     R_Def  R_Def
                |   RRA
                |   "{"Name"}"
Def         -->     Def  Def
                |   RA
                |   "{"Name"}"

RA          -->     LRA
                |   RRA
                |   ERA
```

2.1.2 Die Syntax *Regulärer Ausdrücke* (*RA*s) in mo_lex

Natürliche Sprachen besitzen nur endliche Morpheminventare, und mir sind keine Analysen von Morphemen bekannt, die beliebig viele Features markieren. Es gibt z.B. sicherlich kein Morphem, nennen wir es HU, das "(1sg)+" (ein oder beliebig viele Male "1sg") ausdrückt. Dasselbe gilt für Kontextspezifikationen: Morphologische Allomorphie kennt keine Abhängigkeiten, die über unbegrenzt lange Distanzen gelten.[5] Aus diesem Grund gibt es in **mo_lex** anstatt *Regulärer Ausdrücke* nur *Endliche Reguläre Ausdrücke* (*ERA*s). *ERA*s sind *Reguläre Ausdrücke* ohne Stern- ("*") und Kreuz ("+")-Operator, also ohne Rekursion. Auch ε ist kein *ERA*. Dies bedeutet, daß Epenthese (etwa: "LK\ε/RK --> String") in **mo_lex** nicht ausdrückbar ist.[6]

2.1.3 *Endliche Reguläre Ausdrücke* (*ERA*s)

Die Syntax *Regulärer Ausdrücke* in **(f)lex** ist ziemlich umfangreich, aber praktisch alle darin enthaltenen Konstruktionen lassen sich als Abkürzungskonventionen für Ausdrücke der folgenden Grundsyntax (2) bzw. für die durch diese denotierten Stringmengen (3) verstehen:[7][8]

(2) a. Jeder Buchstabe ist ein *Regulärer Ausdruck*.
b. Wenn X_1, ..., X_n *Reguläre Ausdrücke* sind, ist es auch $X_1 ... X_n$. (Verkettung).
c. Wenn X_1, ..., X_n *Reguläre Ausdrücke* sind, ist es auch (X_1 | ... | X_n) (Disjunktion).
d. Wenn X ein *Regulärer Ausdruck* ist, ist es auch (X)*.

(3) a. Jede Kette denotiert die Menge, die ausschließlich diese Kette enthält.
b. X (Y_1 | ... | Y_n) denotiert die Vereinigungsmenge der Mengen, die von XY_1, ..., XY_n denotiert werden.
c. X (Y)*Z denotiert die Vereinigungsmenge der Mengen, die von XZ, XYZ, XYYZ, XYYYZ,... denotiert werden.

Z.B. sind die *Regulären Ausdrücke* "(ab){2, 3}" ("zwei oder dreimal "ab") und "(abab | abababab)" ("abab oder abababab") semantisch äquivalent, ebenso "a+" ("1 oder mehrmals a") und "aa*"("a verkettet mit 0 oder mehreren as"). Die Grundsyntax für *Endliche RegulärеAusdrücke* ist in (2´) gegeben:

(2´) a. Jeder Buchstabe ist ein *Endlicher Regulärer Ausdruck*.
b. Wenn X_1, ..., X_n *Endliche Reguläre Ausdrücke* sind, ist es auch $X_1 ... X_n$.
c. Wenn X_1, ..., X_n *Endliche Reguläre Ausdrücke* sind, ist es auch (X_1 | ... |X_n).

---

[5] s. **4.1**. in **§1/4**.
[6] Für linke und rechte Kontexte ist das Fehlen von ε ohne Belang, da sie optional sind.
[7] Andere Komplikationen ergeben sich mit ESCAPE-Zeichen, z.B. "\n" für "Backspace" und der Ausschaltung von Sonderzeichen. So ist "(" ein Metazeichen, "\(" hingegen denotiert das Zeichen "(". *Reguläre Definitionen* lassen sich ebenfalls als Abkürzungskonventionen auffassen.
[8] Die Definitionen folgen Johnson (1972:16-17). Anders als bei Johnson und wie in **(f)lex** ist ε hier kein *Regulärer Ausdruck*. Die Notation ist an **(f)lex** -Konventionen angepaßt.

Für praktische Zwecke wie in **§2/5** gehe ich vom **(f)lex**-Inventar für die Syntax *Regulärer Ausdrücke* aus, soweit sich die entsprechenden Konstruktionen auf (2´) (bzw. die entsprechenden Sätze in (3)) reduzieren lassen (vgl. Anhang **C**), für die weitere theoretische Diskussion von den Definitionen in (2´) und (3).

2.1.4 *Disjunktive Normalform*

*ERA*s (bzw. *LRA*s und *RRA*s) lassen sich in eine äquivalente Normalform der Form ($S_1 | S_2 | ... | S_n$), für S_{1-n} Strings, bringen, indem wir alle Abkürzungen ("?" "[ ]") durch entsprechende Kombinationen aus Verkettung und Disjunktion ersetzen. Die so erhaltenen Ausdrücke lösen wir von innen her auf. Verkettungen von Verkettungen sind unproblematisch, da sie Strings ergeben, Disjunktionen von Disjunktionen, etwa (($S_1 | S_2 | S_3$) | $S_4 | S_5$) lösen wir auf, indem wir die Klammern von eingebetteten Disjunktions-Ausdrücken entfernen: ($S_1 | S_2 | S_3 | S_4 | S_5$). Verkettung mit disjunktiven Ausdrücken, z.B. ($S_1 | S_2$) ($S_3 | S_4$) formen wir um, indem wir jeden Teilausdruck der ersten Disjunktion mit jedem Teilausdruck der zweiten verketten und die Disjunktion daraus bilden: ($S_1S_3 | S_1S_4 | S_2S_3 | S_2 | S_2S_4$). Die Möglichkeit, *ERA*s in *disjunktive Normalform* zu bringen, ergibt sich unmittelbar aus (b) und (c) in (3) bzw. der Tatsache, daß *ERA*s endliche String-Mengen denotieren.

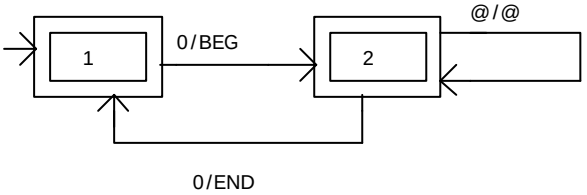
2.1.5 Negierte *Reguläre Ausdrücke*

Negation dient in **mo_lex** dazu, Kontexte in Constraints eleganter formulieren zu können. Z.B. taucht bei der **3sg-prs** des Hilfsverbs *jam* nur dann der irreguläre Stamm *ësh-* auf, wenn dem Verb *nicht* der Konjunktiv-Partikel *të* vorrausgeht. Obwohl Negation komplexer ist als etwa der "?"-Operator ist sie ebenfalls eine Abkürzungs-Konvention, die sich grundsätzlich auf die Kombination von Disjunktion und Verkettung zurückführen läßt.
Ein Constraint wie "~A\X --> Z" soll intuitiv besagen, daß X zu Z wird, wenn vor X entweder ein Zeichen steht, das nicht A ist (z.B. B) oder aber "nichts". "Nichts" kann hier nicht der leere String (ε) sein, z.B. steht in der Kette "AX" X rechts von ε, da AX = Aε X , und dies ist gerade der Fall, den wir ausschließen wollen. Es ist stattdessen sinnvoll, dieses "Nichts" in linken Kontexten als den *Anfang* und in rechten Kontexten als *Ende* der Eingabe zu interpretieren. Dazu führe ich für *Links Negierte RA*s das (reservierte) Symbol **BEG** (BEGIN) und für *Rechts Negierte RA*s das (reservierte) Symbol **END** ein.[9] Behandeln wir zuerst die Negation von Strings beliebiger Länge, z.B. steht in "X/~(ABC)" "~ABC" entweder (a) für das Eingabeende, oder (b) für einen String, der mit einem Buchstaben beginnt, der nicht A ist, oder (c) einen String, der mit A beginnt, aber mit einem Buchstaben, der nicht B ist oder dem Eingabeende weitergeht, oder (d) einen String, der mit AB beginnt, und von einem Buchstaben, der nicht C ist, oder vom Eingabeende weitergeführt wird:

---

[9] Wir können annehmen, daß jede Eingabe in der Form **BEG** ... **END** erfolgt, oder jede Eingabe mit Hilfe eines einfachen *Finite-State-Transducers* in diese Form bringen:

(1)



Da Komposition von *FST*s wiederum zu *FST*s führt, ist dieses Vorgehen für die weitere Argumentation ohne Probleme (s.u.)

(4) **String-Negation**

a. (Rechts)
$\sim_R$S, S ein String $x_1...x_n$, ist eine Kurzform für
(([^x_1] | **END**) | (x_1[^x_2] | **END**) | .... | ($x_1...x_{1n-1}$([^x_n] | **END**) (n-mal ")")

b. (Links)
$\sim_L$S, S ein String $x_1....x_n$, ist eine Kurzform für
(n-mal "(") (**BEG** | [^x_1]) $x_2...x_n$) | ... | ((**BEG** | [^x_{n-1}]) x_n) | ([^x_n] | **BEG**))[10]

Die Unterscheidung zwischen *Links-* und *Rechts- Negation* ist erstens nötig, da **BEG** nie links und **END** nie rechts von einem Buchstaben des Alphabets vorkommen wird. Zweitens arbeiten die Verneinungs-Operatoren spiegelverkehrt zueinander. So ist in $\sim_L$AB\X/$\sim_R$AB A in $\sim_L$AB, aber nicht in $_R\sim$AB, und B in $\sim_R$AB, aber nicht in $\sim_L$AB. Da die **mo_lex**-Syntax *Links Negierte ERA*s nur in linken und *Rechts Negierte* nur in rechten Kontexten zuläßt, ist das Negations-Symbol jeweils eindeutig und ich werde die Indizes {$_L$, $_R$} außer bei Definitionen weglassen (d.h. ~AB\X/~AB statt $\sim_L$AB\X/$\sim_R$AB schreiben).
Die Negation von disjunktiv verknüpften Strings läßt sich leider nicht durch die Negation der einzelnen Teil-Strings ersetzen, da dann ~(E_1 | E_2 | ... |E_n) die Menge der Strings denotiert, die entweder nicht in E_1, *oder* nicht in E_2 , *oder* ... nicht in E_n sind, aber nicht notwendigerweise Ketten, die in keinem der $E_1...E_n$ sind. Teilstrings, die (für $\sim_R$) echte Präfixe bzw. (für $_L\sim$) echte Suffixe unter den anderen Teil-strings haben, können wir von vorneherein entfernen, da etwa für XY und XZ, Y ≠ Z die (Rechts-) Negation von XZ sicher XY beinhalten wird und umgekehrt. ~(B|AB)\X/~(A|AB) ist also äquivalent zu ~B\X/~A. Ich gebe hier nur den Algorithmus, der für rechtsnegierte disjunktive *ERA*S einen nichtnegierten äquivalenten *ERA* liefert. Die linksnegierte Version verhält sich wieder genau spiegelbildlich dazu (einschließlich der Tatsache, daß **END** durch **BEG** ersetzt werden muß). Mit *First*(S) beziehe ich mich auf den ersten Buchstaben des Strings S, mit *Rest*(S) auf den Reststring, *L*(S) bezeichnet die String-Länge.

(5) **Negation von Disjunktionen**

Für $\sim_R$ (S_1 | S_2 | ... | S_n)

Entferne alle S_x, die Präfixe eines anderen S_y sind

Op(S_1,S_2, ..., S_n).
(
drucke("(");
drucke ("[^x_1.....x_n]"); wobei x_i = *First*(S_i)
drucke("|END");
Für jedes x_i, für das es mindestens ein S_{ij} gibt, so daß x_i = First(S_{ij}) und *L*(S_{ij}) > 1
(
drucke ("|(x_i");
für alle S_{ij} (j = 1....z), so daß x_i = *First*(S_{ij}) und *L*(*Rest*(S)) > 0
Op(*Rest*(S_{i1})...*Rest*(S_{iz})),
)
drucke(")");
)

[10] Die Notation "[^xyz]" ist eine **(f)lex**-Abkürzungskonvention und wird in Anhang **C** genauer beschrieben.

Schließlich gilt noch, daß $\sim_R\sim_R X = X$, und $\sim_L\sim_L X = X$, $\sim_L$**BEG** = $\sim_R$**END** = "."(d.h. ein beliebiges Zeichen ungleich **END** oder **BEG**). Die Definition für *RRA*s ergibt sich aus der für *ERA*s in (4´) durch Hinzufügen der Klauseln in (6) zur Definition von *ERA*s:

(6)

a. Die Kette aus dem Symbol **END** ist ein *RRA*.
b. Wenn R ein *RRA* ist, ist $\sim_R$R ein *RRA*.

Die Definition für *Links Negierte RA*s ist wieder spiegelverkehrt dazu.

2.2 Die Interpretation von mo_lex

2.2.1 *String-Relationen* und *Finite-State-Transducer*

mo_lex-Theorien sind Spezifikationen für binäre R*eguläre String-Relationen*[11], also für Mengen M von geordneten Stringpaaren $\{(E_1, A_1), (E_2, A_2), ..., (E_n, A_n)\}$. Das erste Element eines solchen String-Paares ist eine Abfolge von (Morphem-)Symbolen über dem Eingabealphabet, das zweite eine Kette von Symbolen des Ausgabealphabets (also von Phonemen), z.B. ((S(jam) 1sg, *jam*). N-stellige *Reguläre Relationen* sind, wie wohlbekannt ist (s. Johnson 1972, Kaplan & Kay 1994), äquivalent zu *Finite-State-Maschinen* mit N Bändern. Ich spreche daher im Folgenden je nach Bedarf von *String-Relationen* oder *Finite-State-Transducern* (*FST*s).
M ist eine teilweise Funktion, d.h. für jedes erste Element E eines String-Paares gibt es null oder ein Element A, so daß (E, A) in der Relation ist. M.a.W. **mo_lex**-Theorien liefern für jeden Eingabe-String - deterministisch, d.h. rechtseindeutig - höchstens einen Ausgabe-String. Im Folgenden skizziere ich unter Verwendung von kontextsensitiven Regeln die Abbildung von **mo_lex**-Theorien auf String-Relationen. Dabei zeige ich, daß die durch **mo_lex**-Theorien festgelegten Stringrelationen *regulär* und in der genannten Hinsicht deterministisch sind.

2.2.2 Konflikt-Auflösung revisited

Die Semantik von **mo_lex** ist bis auf die Auflösung von Regelkonflikten am praktischen Verhalten von **(f)lex** orientiert. Der *einzige* Auflösungs-Mechanismus, den ich in dieser Arbeit verwende, beruht auf der willkürlichen Anordnung von Constraints (die Regelpräzedenz-Strategie aus **§1/1**). Es liegt also nahe, sich für **mo_lex** auf diesen Mechanismus zu beschränken, und - falls dies aus theoretischen Gründen nicht genügt - andere Auflösungs-Strategien nur für die Fälle zu verwenden, in denen dies unbedingt notwendig ist. Einige konstruierte Beispiele zeigen, daß die Regel-Präzedenz-Strategie nicht hinreicht, um beliebige Konflikte *innerhalb* von Constraints aufzulösen. Z.B. würde das Constraint "a{1,3} --> A" aaa ohne die Präferenz für längere Strings nach A, AA *und* AAA übersetzen. Die Theorie in (7) würde ohne die Links-Rechts-Strategie aaa auf AB *und* BA abbilden:

(7)

C1 aa --> A **C2** a --> B

Davon ausgehend, bleiben drei Alternativen, um eine weitgehende Beschränkung auf die Regel-Präzedenz-Strategie zu erreichen: (1) Wir schließen **mo_lex**-Constraints aus, die potentiell interne Konflikte induzieren. (2) Wir wenden die Bevorzugung und die Links-Rechts-Strategie nur an, um constraintinterne Konflikte aufzulösen, m.a.W. wir kehren die Auflösungs-Hierarchie aus **§1/1** um. (3) **mo_lex** darf keinen "mehrdeutigen" Input akzeptieren. Die Optionen, die ich im Weiteren verfolge sind (2) und (3), da Constraints, die potentiell Konflikte induzieren, zumindest für phonologische Anwendungen (vgl. **§2/5**) sinnvoll sein können, etwa für Constraints über zwei aufeinanderfolgende Vokale der Form "VV --> X".

[11] Für eine genauere Behandlung solcher Relationen und ihrer formalen Eigenschaften s. Kaplan & Kay (1994).

2.2.3 mo_lex und kontextsensitive Regeln

Für die Modellierung der Semantik von **mo_lex** beziehe ich mich auf Arbeiten von Kaplan & Kay (1994, im Weiteren: [K&K]) und Johnson (1972), die zeigen, daß kontextsensitive Regeln im SPE-Format (Chomsky & Halle 1968) als *Finite State Transducer* interpretiert werden können, wenn man die Anwendung von Regeln auf ihren eigenen Output ausschließt. Grammatiken mit geordneten kontextsensitiven Regeln können dann als Komposition der einzelnen Transducer verstanden werden. Da Komposition die *Regularität* von Relationen erhält, beschreiben auch die vollständigen Grammatiken *Reguläre Relationen*. Im Folgenden skizziere ich ein Verfahren, um **mo_lex**-Theorien auf Grammatiken mit geordneten kontextsensitiven Regeln und damit auf *Reguläre Relationen* abzubilden.
Die Interpretation einzelner kontextsensitiver Regeln zeigt - grob gesprochen- eine gewisse Verwandtschaft zu der von **mo_lex**-Constraints. Teil-Strings von Strings sollen in bestimmten Kontexten durch andere Strings ersetzt werden:

(8)

	a. Kontextsensitive Regeln				**b. mo_lex-Constraints**		
C1	B	-->	E/A__C	**R1**	A\B/C	-->	E
C2	B	-->	F	**R2**	B	-->	F

In beiden Fällen würde B in ABC nach E und B in XBY nach F übersetzt.

2.2.4 Defaults und die Auflösung von Konflikten in mo_lex

Wie in **mo_lex**, spielt bei kontextsensitiven Regeln die Anordnung der Klauseln eine wichtige Rolle[12], und wir können dadurch eine Art Default-Effekt erreichen: Drehen wir die Reihenfolge der Regeln in (12) um, erhalten wir in (8a) und (8b) für ABC und XBY die Übersetzung von B nach F. Kontextsensitive Regeln können optional oder obligatorisch angewendet werden ([K&K]:356). Um wie in **(f)lex** Determinismus zu erreichen, beschränke ich die Diskussion auf die obligatorische Anwendung von Regeln. Ferner unterliegen kontextsensitive Regeln normalerweise Beschränkungen im Hinblick auf ihre Anwendungsrichtung. Z.B. führt die Anwendung der Regel in (13) auf AAA von links nach rechts zu BA und von rechts nach links zu AB:

(9) AA --> B

Dies entspricht einem weiteren Konfliktlösungs-Mechanismus in **(f)lex** (der Links-Rechts-Strategie, s. **§1/1**), den wir bei der Modellierung von **mo_lex** durch kontextsensitive Regeln übernehmen können, indem wir alle Regeln obligatorisch von links nach rechts anwenden. Die *Finite-State*-Formalisierungen dieser Regeln berücksichtigen die Anwendungsrichtung von Regeln ([K&K]:355, Johnson 1972:58), so daß dies auch formal unproblematisch ist.[13]

2.2.5 Die Modifikation von Kontexten

Es bleibt ein wichtiger Unterschied zwischen der intendierten Interpretation von **mo_lex**-Constraints und kontextsensitiven Regeln. Diese werden auf den Output anderer Regeln angewendet. Z.B. erzeugt die Anwendung von **R1** in (10b) auf ABC GBC. Darauf läßt sich **R3**, nicht aber **R2** anwenden, und wir erhalten GFC:

[12] soweit man, wie hier vorausgesetzt, von extrinsischer Regelanordnung ausgeht.

[13] Ich beschränke die Diskussion hier auf Option (2) aus **2.2.2**. Eine mögliche Implementation für Option (3) behandle ich in **2.2.8**.

(10) **a. mo_lex-Constraints** **b. Kontextsensitive Regeln**

C1	A/B	-->	G	**R1**	A	-->	G/__B
C2	A\B/C	-->	E	**R2**	B	-->	E/A___C
C3	B	-->	F	**R3**	B	-->	F

Dies ist für **mo_lex** sicher nicht das gewünschte Ergebnis. Nehmen wir an, daß ABC morphosyntaktische Features (Morpheme) sind und G, E, F Phonem-Ketten. Die folgende kleine Theorie beschreibt die **aor-prs-1sg**-Form von *zë*: *zura*.

(11) **mo_lex-Constraints**

C1	S(zë)/aor	-->	zu
C2	S(zë)\aor/[1-3]sg	-->	r
C3	aor\1sg	-->	a

"S(zë)aor" würde jedoch, wenn wir die Constraints unmittelbar wie kontextsensitive Regeln interpretieren, durch **C1** zu "zu aor", d.h. der angemessene Kontext für **C2** wird durch **C1** "zerstört ". Wir befinden uns in einem Dilemma: Der Default-Effekt, den wir in **2.2.3** gesehen haben, kommt gerade dadurch zustande, daß Teile des Ausgangs-Strings nach deren Modifikation durch eine Regelanwendung nicht mehr sichtbar sind. Genau diese Unsichtbarkeit von zu grundeliegendem Material bewirkt aber, daß relevante Kontexte wie in (11) nicht mehr identifiziert werden können. Dieses Dilemma läßt sich umgehen, wenn wir die Tatsache ausnützen, daß durch Regel-Anwendungen modifizierte Strings als Kontexte sichtbar sein sollten, nicht aber als Regel-Kerne.
Wir schaffen für jede Ausgangsregel mit dem Index i und jedes Symbol S des Alphabets das Hilfssymbol S_i. Wir ersetzen alle Regeln der Form "K_i --> E_i /LK_i__RK_i" durch geordnete Paare von Regeln "K_i --> E_i´/LK_i´___RK_i´" und "E_i´ --> E_i. E_i´erhalten wir, indem wir jedes Symbol S in K_i durch S_i ersetzen. LK´und RK´, indem wir jedes Symbol S durch $[SS_1...S_n]$ (bei n Ausgangsregeln) ersetzen. Die Regeln ordnen wir in zwei Blöcken an. Der erste Block enthält alle Regeln, die das erste Element in einem Regel-Paar sind, der zweite Block alle Regeln, die an zweiter Stelle in einem Regel-Paar stehen. In beiden Blöcken sind die Regeln linear wie ihre Ausgangs-Regeln geordnet. (13) zeigt die Anwendung des Verfahrens auf (12):

(12) V = {A, G, B, E, F, C}

R1	A	-->	G/__B
R2	B	-->	E/A___C
R3	B	-->	F

(13) V = {A, A_1, A_2, A_3, B, B_1, B_2, B_3, C, C_1, C2, C_3, E, E_1, E_2, E_3, F, F_1, F_2, F_3, G, G_1, G_2, G_1}

R1/1	A	-->	A_1/__[B$B_1$$B_2$$B_3$]
R2/1	B	-->	B_2/[A$A_1$$A_2$$A_3$]___[C$C_1$$C_2$$C_3$]
R3/1	B	-->	B_3
R1/2	A_1	-->	G
R2/2	B_2	-->	E
R3/2	B_3	-->	F

ABC wird zu A_1BC (**R1/1**), dann zu A_1B_2C. A_1 wird durch G und B_2 durch E ersetzt, so daß wir GEF erhalten. Die Anwendung von Regeln des 2. Blocks ist trivial, da die relevanten Teile von Input-Strings durch genau eine Regel im ersten Block eingeführt werden und genau durch eine Regel im 2. Block gematcht werden. Das erweiterte Alphabet wird um einiges größer sein als das Ausgangs-Alphabet, aber immer noch endlich, da es nur endlich viele Regeln gibt und die Zahl der neu eingeführten Symbole der Regelanzahl mal der Zahl der Ausgangssymbole entspricht.

2.2.6 Präferenz für längere Strings

Bis jetzt haben wir zwei von drei nötigen Konfliktlösungs-Strategien in **mo_lex** modelliert. Es fehlt die Präferenz für längere Strings. Erinnern wir uns zu diesem Zweck daran, daß sich die Kerne von **mo_lex**-Constraints in disjunktive Normalform bringen lassen (s. **2.1.4**). Die entsprechenden Constraints haben dann die Form LK\(S_1 | S_2 | ... | S_n)/RK --> E, für S_1, ..., S_n Strings. Dem entspricht jeweils eindeutig eine Menge von Regeln {LK\$S_1$/RK --> E, LK\$S_2$RK --> E, ..., LK\$S_n$/RK --> E}. Der ursprüngliche Regel-Kern enthält nur endlich viele disjungierte Strings, also ist diese Menge endlich. Die Constraints in solchen Mengen nenne ich *Elementar-Constraints*. Wenn wir annehmen, daß **mo_lex** Theorien nicht direkt interpretiert werden, sondern ausgehend von einer bestimmten Anordnung ihrer *Elementar-Constraints*, und daß die *Elementar-Constraints* eines Constraints der Länge nach geordnet sind, modellieren wir damit Konfliktauflösung durch die Bevorzugung längerer Strings.
Es bleibt das Problem, wie wir in *disjunktiver Normal-Form* zwischen disjunktiv verknüpften Strings (bzw. den zugehörigen Regeln) wählen, die gleichlang sind. Führt z.B. (14a) zu (14b) oder zu (14c):

(14)

a. CD\(AB|BA)/EF --> X

b. CD\AB/EF --> X
CD\BA/EF --> X

c. CD\BA/EF --> X
CD\AB/EF --> X

Es bietet sich an, auf die Links-Rechts-Strategie zurückzugreifen, die wir mit Hilfe sogenannter Batch-Regeln modellieren können ([K&K]:349). Batch-Regeln sind Komplexe aus mehreren (untereinander ungeordneten) Sub-Regeln. Bei der Anwendung von Batch-Regeln wird an jedem Punkt der Ableitung eine jeweils passende Teilregel angewendet. In obligatorischen Batch-Regeln, die von links nach rechts angewendet werden, kommt immer *die* Regel zuerst zur Anwendung, die einen String möglichst weit links matcht.
Zu Batch-Regeln bündeln wir alle *Elementar-Constraints* eines Constraints, deren Kerne gleichlang sind. Die Batch-Regeln eines Constraints ordnen wir für die Interpretation der **mo_lex**-Theorie, wie gehabt, nach der Länge der Kerne, z.B. wird (15a) zu (15b) ("**{,}**" klammern Sub-Regeln einer Batch-Regel):

(15)

a.
QR\(AB|DEF|HIJ|K|LM)/OP --> Z

b.
{QR\DEF/OP --> Z
QR\HIJ/OP --> Z**}**

{QR\LM/OP --> Z
QR\AB/OP --> Z**}**

QR\K/OP --> Z

Damit hat die Bevorzugung längerer Strings (innerhalb von Constraints) Präzedenz über die Links-Rechts-Strategie, was eine weitere Abweichung von **(f)lex** (mit der umgekehrten Präzedenz) bedeutet.

2.2.7 Negative Ergänzung

Grammatiken kontextsensitiver Regeln lassen die Teile von Input-Strings, für die keine expliziten Regeln angegeben sind, unverändert. Das entspricht dem Normal-Verhalten von **(f)lex**, das unspezifizierten Input mit einer Default-ECHO-Anweisung "abfängt" und unverändert wiedergibt. Negative Ergänzung können wir dadurch modellieren, daß wir in der kontextsensitiven Umsetzung einer **mo_lex**-Theorie zwischen die beiden Regel-Blöcke (s. **2.2.4**) einen Transducer setzen, der die Erweiterungs-Buchstaben des Eingabe-Alphabets auf sich selbst abbildet, hingegen keinen String akzeptiert, der Buchstaben des (nichterweiterten)

Eingabe-Alphabets enthält.[14] Der Effekt dieser Maßnahme ist, daß der Gesamt-Transducer nur Strings akzeptieren wird, bei denen jeder einzelne Buchstabe der Eingabe durch eine Regel-Anwendung modifiziert wird.

2.2.8 Two-Level-Constraints anstelle kontextsensitiver Regeln

Arbeiten wir statt mit kontextsensitiven Regeln mit Two-Level-Constraints, ergibt sich eine restriktivere Lösung für regelinterne Konflikte in der Art von Option (2) in **2.2.2**. Wir modellieren jedes **mo_lex**-Constraint durch eine *Surface Coercion Rule* (vgl. **4.2.3** in **§1/4**), etwa (16a) durch (16b):

(16) a. A\(AB|ABC)/C --> D b. (AB|ABC):D <= A:@__C:@

(20a) führt in Strings wie AABCC zu einem Konflikt, da sowohl die Übersetzung von AB als auch von ABC nach D möglich ist. (16b) hingegen fordert für jede Übersetzung von AABCC, daß alle Strings, die (AB|ABC) matchen, nach D übersetzt werden müssen, was offensichtlich unmöglich ist, da es keinen String gibt, der gleichzeitig ADCC und ADC ist. Eine Two-Level-Grammatik, die (16b) enthält, liefert für AABCC also keinen Output. Interne Regelkonflikte führen zur Nichtakzeptanz der Eingabe. Auf die Konstruktion von Batch-Regeln und die Anordnung der Elementarconstraints nach ihrer Länge können wir im Two-Level-Format also verzichten, da alle Regelkonflikte bei sich überschneidenden Regelkernen zum selben negativen Ergebnis führen. Ich lasse die Frage hier offen, ob die Verwendung von Two-Level-Regeln in diesem Zusammenhang besser ist als die kontextsensitiver Regeln, da es meines Wissens keinerlei empirische Evidenz für Allomorphie-Constraints gibt, die sich derart "überschneiden", daß Konflikte, wie die hier besprochenen, auftauchen.[15] Aus Platzgründen gehe ich hier nicht im Detail auf die Möglichkeit einer Implementation durch Two-Level-Regeln ein und verfolge im weiteren die Modellierung durch kontextsensitive Regeln.

2.2.9 mo_lex ist deterministisch

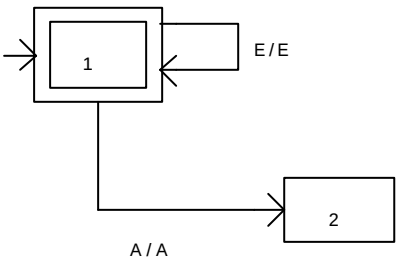
Aus den Abschnitten **2.2.3** bis **2.2.7** ergibt sich, daß jede **mo_lex**-Theorie zu genau einer Repräsentation als kontextsensitive Grammatik führt. [K&K]:(358) nennen zwei Faktoren, die dazu führen können, daß kontextsensitive Regeln wie (17) auch dann für eine Eingabe mehrere Ausgabe-Strings bereithalten, wenn sie obligatorisch und in festgelegter Abarbeitungs-Richtung angewendet werden:

(17) X --> Y /A__B

(1) Y ist ein *Regulärer Ausdruck*, der mehr als einen String denotiert und (2) X denotiert mindesten zwei Strings S_1 und S_2, so daß $S_1 = S_2S_3$ und $S_3 \neq \varepsilon$. Beide Fälle sind für die kontextsensitive Repräsentation von **mo_lex**-Theorien ausgeschlossen. (1) durch die **mo_lex**-Syntax: Ersetzungen (Y in (17)) sind Strings und nicht beliebige *Reguläre Ausdrücke*, (2) durch das Verfahren zur Bildung kontextsensitiver Re präsentationen, das Batch-Regeln erzeugt. Deren Teilregeln enthalten als Kerne ebenfalls nur Strings und die

---

14



E steht für beliebige Buchstaben des Erweiterungs-, A für beliebige Buchstaben des Ausgangs-Alphabets.

15 Der Gebrauch, den ich hier von Two-Level-Regeln mache, entspricht nicht dem von Two-Level-Grammatiken, sondern von Grammatiken geordneter kontextsensitiver Regeln, d.h. Regeln folgen sukzessiv aufeinander und gelten nicht parallell. Da beide Regelformate auf *FST*s abbildbar sind, ist dies unproblematisch.

Kerne dieser Teil-Regeln sind für jede Batch-Regel gleich lang. Aus dem Determinismus der einzelnen Transducer ergibt sich in trivialer Weise, daß ihre Komposition deterministisch ist und **mo_lex**-Theorien für einen beliebigen Eingabe-String höchstens einen Ausgabe-String vorsehen.