GÖTEBORG
UNIVERSITY

Gothenburg June 2005

# Implementations of Parsing Algorithms for Linear Multiple Context-Free Grammars

Thesis for M.A. in Computational Linguistics


Håkan Burden
Dept. of Linguistics
Göteborg University
Supervisor: Peter Ljunglöf
Dept. of Computing Science
Chalmers University of Technology
Göteborg University


# Abstract

This thesis is an account of implementations of parsing algorithms for Linear Multiple Context-Free Grammars (LMCFG). The algorithms have originally been proposed for Parallel Multiple Context-Free Grammars (PMCFG), a superclass to LMCFG, by Peter Ljunglöf. LMCFG is a Mildly Context-Sensitive Grammar formalism.

The implementations are part of the work being carried out for the TALK project at the departements of Linguistics at Göteborg University and Computing Science at Chalmers University of Technology and Göteborg Univerity.

The Language Technology Group at Chalmers is currently conducting research round the grammar formalism Grammatical Framework (GF). The important subclass context-free GF is equivalent to PMCFG. This implies that a subset of the context-free GF grammars can be parsed as equivalent LMCFG grammars.

Four different algorithms for parsing LMCFG are implemented, using deductive agenda-driven chart-parsing. The first algorithm is a straightforward bottom-up strategy combining items with smaller cover of the input string to items with larger cover. The second algorithm uses a context-free approximation and then recovers the resulting chart. The third algorithm is an active algorithm with Earley and Kilbury prediction. And the last algorithm is incremental.

The algorithms have not been thoroughly tested as part of the work presented here. However, preliminary testing indicate that they seem faster than the existing parser for GF.

# Sammanfattning

Den här uppsatsen är en redogörelse för implementeringar av parsningsalgoritmer för Linear Multiple Context-Free Grammars (LMCFG). Algoritmerna har från början föreslagits av Peter Ljunglöf för Parallel Multiple Context-Free Grammars (PMCFG), en superklass till LMCFG. LMCFG är en milt kontextkänslig grammatikformalism.

Implementeringen är en del av det arbete som institutionerna för Lingvistik vid Göteborgs Universitet och Datavetenskap på Chalmers Tekniska Högskola och Göteborgs Universitet utför inom TALK projektet.

Språkteknologigruppen vid Chalmers bedriver bland annat forskning kring grammatik formalismen Grammatical Framework (GF). Den viktiga subklassen context-free GF är ekvivalent med PMCFG. Det innebär att vissa context-free GF grammatiker kan parsas som ekvivalenta LMCFG grammatiker.

Fyra olika algoritmer har implementerats utifrån deduktiv agenda-driven chart-parsning. Den första algoritmen är en enkel bottom-up algoritm som kombinerar den erhållna chartinformationen nerifrån och upp till större och större enheter. Den andra algoritmen utgår från en kontextfri uppskattning och filtrerar sen ut den information som överensstämmer med den ursprungliga LMCFG:n. Tredje algoritmen är en variant på aktiv parsning med både Earley och Kilbury filtrering som alternativ. Den sista algoritmen är en inkrementell algoritm.

Det har inte genomförts någon omfattande utvärdering av algoritmerna inom det arbete som presenteras här. Preliminära tester antyder dock att algoritmerna är snabbare än den nuvarande parsningsalgoritmen för GF.

# Acknowledgments

Time to get down to serious business. It always seems appropriate when one has neglected life for academic virtues.

First of all I have to give Peter my sincere thanks. He's responsible for introducing me to parsing algorithms in the first place. Parsing algorithms are more than science, they are art forms. I'm also very grateful for the way he's guided me through my first major solo project.

Thank you Silverbullit for Citizen Bird.

Thank you Liverpool for your Champions League adventures. Long nights at the computer pass by quick as anything when Juventus and Chelsea are knocked out by The Reds. And an extra round of applause for Carragher and Dudek, your performance in Istanbul will be a part of me for the rest of my life.

Thank you, my modernly extended family. I love you Burdens, Blåbergs, Bukowinskas and Josefsson. You've always believed in me and your support has often been vital.

A very big thanks to all my friends at school. I've enjoyed your company. I hope I won't have to miss you.

To my friends outside, I love you. For all we have done and everything we said we could do. I hope to see you soon.

And thank you Malva for keeping it real. Godspeed You Black Emperor. I love you and could never have done it without you. You combine the knowledge of a buddhist monk with a hedonists appetite for life, bringing order to chaos and chaos to order.

Finally, Ellen, I love you ♡. Thank you for all your love and support. Living with me hasn't always been that easy. I wish you the best and hope to be a part of your future, for ever and ever.

/Håkan

# Contents

# List of Figures

Figure 1: Notations used throughout the thesis

| | |
|---|---|
| $w$ | an input string s.t. $w = w_1 \dots w_n$ |
| $\epsilon$ | the empty sequence/string |
| $s$ | a substring in $w$, $s = w_i, \dots, w_j$ $0 \leq i \leq j \leq \|w\|$ |
| $G$ | a grammar, $G = (C, \Sigma, S, \mathcal{R})$ |
| $C$ | the set of non-terminals, also called the categories |
| $\Sigma$ | the set of terminal tokens (the alphabet) |
| $S$ | a start-symbol of a grammar s.t. $S \in C$ |
| $\mathcal{R}$ | the set of rules |
| $V$ | the set of symbols, $C \cup \Sigma$ |
| $A$, $B$ | elements in $C$ |
| $\vec{B}$ | a sequence of categories, $B_1, \dots B_n$ |
| $a$, $b$ | elements in $\Sigma$ |
| $\mathcal{L}$; $\mathcal{L}(A)$ | a language; the language of category $A$ |
| $f$, $g$ | function symbols |
| $\delta$ | arity of a function or rule |
| $\alpha, \beta, \gamma$ | sequences of linearizations or elements in $V$ |
| $\Phi$, $\Psi$ | linearization records or sequences of linearizations, $\Phi = \alpha_1 \dots \alpha_n$ |
| $^*$ | the Kleene star |
| $\cdot$ | concatenation of two sequences |
| $\rightarrow$ | $A \rightarrow \alpha \equiv (A, \alpha) \in \mathcal{R}$ |
| $\Rightarrow$ | $B \Rightarrow \alpha\beta\gamma$ whenever $B \rightarrow \beta$ |
| $\Rightarrow^*$ | the reflexive and transitive closure of a category |
| $\|x\|$ | the length/size of $x$ |
| $n$, $m$ | natural numbers we often use $n$ when $\|x\|$ is known |
| $\Gamma$ | a record of any type. Often used for range records |
| $\vec{\Gamma}$ | a sequence of records, $\vec{\Gamma} = \Gamma_1, \dots \Gamma_n$ |
| $r$, $s$ | record labels |
| $i$, $j$, $k$ | natural numbers used for indices |
| $(i, j)$ | the range $i$ to $j$ |
| $\rho$ | any range $(i, j)$ |
| $\rho^\epsilon$ | the range for the empty string |
| $\lceil (i, j) \rceil$ | the ceiling of a range, returns $(j, j)$ |
| $\lfloor (i, j) \rfloor$ | the floor of a range, returns $(i, i)$ |

# Chapter 1

# Introduction

## 1.1 Motivation

This thesis is the report for the implementations of parsing algorithms carried out during the spring of 2005. The parsing algorithms are proposed in Peter Ljunglöf's PhD thesis "Expressivity and Complexity of the Grammatical Framework" (2004). The implementations cover a subset of the proposed algorithms.

We have tried to follow the notations of Ljunglöf when possible, to make it easy to compare the proposed algorithms with the implemented.

Grammatical Framework (GF) is one of the areas of research at Chalmers University of Technology. The work around GF is also a part of the work being carried out for the TALK-project (Tools for Ambient Linguistic Knowledge) at Chalmers and Göteborg University. For information about GF and the TALK project, see GF (2004) and TALK (2004) respectively.

## 1.2 Structure

The chapters have the following structure

- Chapter 2 Background: Introduction and definition of some grammar formalisms. Explanation of decorated context-free parsing algorithms using deduction and a brief description of the proposed algorithms.
- Chapter 3 Implementation: A description of the algorithms as they have been implemented.
- Chapter 4 Small Evaluation: For several reasons there has not been any extensive evaluation conducted. Nevertheless it is possible to draw some conclusions, and to show some parse results.
- Chapter 5 Summary: Comments on the implementations and proposals for future work. Rounding off the thesis.
- Appendix A The Code: The source code of the implemented algorithms.

## 1.3 Haskell

The algorithms are implemented in Haskell as is most of the Grammatical Framework. It is not necessary to understand Haskell to read the report even if the code of the implementations is in Haskell. The bulk of the implemented code can be found in Appendix A.

Haskell is a functional programming language named after one of the pioneers in $\lambda$-calculus, Haskell B. Curry. It is based on $\lambda$-calculus and statically typed. This means that the implemented functions are defined for specific types; a function for doubling a `Float` will not take an `Integer` as an argument. Functions can be higher-order, meaning that a function can have other functions as arguments.

For more information on Haskell, see Hudak et al. (1999), Peyton Jones (2003) or Thompson (1999).

# Chapter 2

# Background

The background chapter can roughly be divided into two parts. The first part introduces the grammar formalisms that are background knowledge Linear Multiple Context-Free Grammars. It also gives a variant of Context-Free Grammars called Decorated Context-Free Grammars. The second part presents the tools for parsing. A brief account of the proposed algorithms is given at the end.

## 2.1 Preliminary definitions

### 2.1.1 Sets and strings

#### Concatenation of sets

Given that $X$ and $Y$ are sets then $X \cdot Y = \{x \cdot y \,|\, x \in X,\ y \in Y\}$. Further, $X^{n+1} = \{X \cdot X^n | \, x \in X\}$ and $X^0 = \{\epsilon\}$, where $\epsilon$ is an empty sequence.

#### The Kleene star

The Kleene star, $^*$, is used to denote all possible repetitions of a set $X$

$$X^* = X^0 \cup X^1 \cup \ldots \cup X^i = \bigcup_0^\infty X^i$$

#### Alphabet

A finite set of terminal tokens is called an alphabet and denoted $\Sigma$.

#### Strings and substrings

A string $w \in \Sigma^*$ is a sequence $w_1 \ldots w_n$ in which each $w_i \in \Sigma$. A substring is any continuous part of a string. This means that every terminal token of a string $w$ can be seen as a substring of $w$.

***Example*** Given the string $w = 1\,2\,3\,4\,5$; $1$, $3\,4$ and $2\,3\,4\,5$ are all substrings of $w$ but $1\,3$, $1\,2\,4\,5$ and $4\,5\,6$ are not.

#### Language

A language, $\mathcal{L}$, is a set of strings over an alphabet, $\Sigma$, i.e. $\mathcal{L} \in \Sigma^*$.

***Example*** The language

$$\{\, a^n b^i c^n \mid n, i \geq 0 \,\}$$

can be written as

$$a^n b^* c^n$$

### 2.1.2 Records and tuples

#### Record

A label is an atomic symbol and a record is a set of unique label-value pairs.

***Example*** If $r_1, \ldots, r_n$ are labels and $x_1, \ldots, x_n$ are values (such as ranges or sequences of symbols) then

$$\Gamma = \{r_1 = x_1; \ldots; r_n = x_n\}$$

is a record.

#### Record projection

Projection on a record $\Gamma$ with the label $r_i$ is written $\Gamma.r_i$. The projection will return the value paired with $r_i$.

Projections will either return a terminal value or another projection, giving record projection a recursive structure.

***Example*** Given the two records

$$\begin{aligned} \Gamma_1 &= \{r = \Gamma_2.r'\} \\ \Gamma_2 &= \{r' = a\} \end{aligned}$$

the projection $\Gamma_1.r$ returns the projection $\Gamma_2.r'$ which in turn will give the terminal value $a$.

**Record unification**

We define simple unification of records as

$$\Gamma_1 \sqcup \Gamma_2 = \Gamma_1 \cup \Gamma_2$$

Simple unification succeeds iff there is no $r$ s.t. $\Gamma_1.r \neq \Gamma_2.r$

**Record substitution**

We can substitute one record for another in a list of records. We write the operation as

$$\Gamma_1, \ldots, \Gamma_n[i := \Gamma]$$

meaning that in the list $\Gamma_1, \ldots \Gamma_n$ the $i$:th element is substituted by $\Gamma$. Substitution can also be performed on projections in records. The operation is denoted

$$\Gamma[B_k/\Gamma_k]$$

and every projection $B_k.r$ in $\alpha_1 \ldots, \alpha_n$ is substituted by the value given by $\Gamma_k.r$.

***Example*** $\Gamma_1$, $\Gamma_2$, $\Gamma_3[2 := \Gamma]$ will substitute the second record for $\Gamma$, returning $\Gamma_1$, $\Gamma$, $\Gamma_3$.

Given $\Gamma = \{r = a\, A_1.r\, a;\, r_2 = A_1.r'\}$ and $\Gamma_1 = \{r = a,\, r' = A_2.r''\}$ then $\Gamma[A_1/\Gamma_1] = a\, a\, a\, A_2.r''$.

**Tuples**

A tuple can be seen as a record since every record projection can be replaced by the corresponding tuple projection.

***Example*** As an example, the tuple $T = (x_1, \ldots, x_n)$ is equivalent to the record $\Gamma = \{1 = x_1; \ldots; n = x_n\}$ and the $i$:th element in $T$ is the same element as that given by the projection $\Gamma.i$.

## 2.2 Grammars

In the following section we will define Context-Free Grammars (CFG) and a variant of CFG called Decorated Context-Free Grammars (DCFG). We will also introduce the grammar formalisms Grammatical Framework (GF, Ranta, 2004) and Parallel Multiple Context-Free Grammars (PMCFG, Seki et al., 1991). The separation of syntax into an abstract and a concrete part will be introduced since this is the way both GF and PMCFG handle syntax.

Figure 2.1: An example of a Context-Free grammar
A context-free grammar (adapted from Ljunglöf (2004), page 17) where the rules are

$$\begin{array}{rcl} S & \rightarrow & NP,\ VP \\ NP & \rightarrow & D,\ N \\ NP & \rightarrow & N \\ VP & \rightarrow & V,\ NP \\ D & \rightarrow & a \\ D & \rightarrow & many \\ N & \rightarrow & lion \\ N & \rightarrow & lions \\ N & \rightarrow & fish \\ V & \rightarrow & eat \\ V & \rightarrow & eats \end{array}$$

and $S$ = sentence, $NP$ = noun phrase, $VP$ = verb phrase, $D$ = determiner, $N$ = noun and $V$ = verb. Only the rules are given since it follows from $\mathcal{R}$ what $C$, $S$ and $\Sigma$ are.

### 2.2.1 Context-Free Grammars

Context-Free Grammars (CFG) are a subclass of the Phrase Structure Grammars. They are called context-free since the rules have no context-dependent information on when they are allowed to be applied; the left-hand side of the rule is restricted to contain a single category, (Chomsky, 1959).

A context-free grammar is a four-tuple ($C$, $S$, $\Sigma$, $\mathcal{R}$), where

- $C$ is the set of non-terminal symbols,
- $\Sigma$ is the alphabet,
- $S$ is the start category of $G$ s.t. $S \in C$ and
- $\mathcal{R}$ is the set of rules: $\mathcal{R} \subseteq C \times V^*$ where $V = C \cup \Sigma$ is known as the set of symbols.

An example of a CFG, recognizing a small fragment of English, can be found in figure 2.1.

#### Some grammar notations

For most grammars $C$, $S$ and $\Sigma$ are obvious from $\mathcal{R}$ and therefore only the rules are given.

We use the Greek letters $\alpha$, $\beta$ and $\gamma$ to denote any sequence of symbols in $V$. It is common to use $A \rightarrow \beta$ instead of $(A, \beta) \in \mathcal{R}$, and we call $A$ the left-hand side

and $\beta$ the right-hand side of the rule. Elements in $\beta$ are the daughters of $A$. For a sequence of symbols, $\alpha B\gamma$, we can use the rewriting relation $\Rightarrow$ to write $\alpha B\gamma \Rightarrow \alpha\beta\gamma$ iff $B \rightarrow \beta$.

The empty string is denoted $\epsilon$ and the rule $A \rightarrow \epsilon$ is called an $\epsilon$-rule. The number of categories on the right-hand side of $\rightarrow$ is the arity of the rule, denoted $\delta$.

#### Expressivity of CFG

Expressivity features handled by a context-free grammar include

- nesting ($a^n b^n$) and
- reverse copying $\{ww^R | w \in (a \cup b)^*\}$ (where $abab^R = baba$)

For most practical uses the complexity of everyday language could be captured within the expressivity of Context-Free Grammars. There are however some linguistic features that do require more expressive power;

- multiple agreement ($a^m b^m c^m$),
- crossed agreement ($a^n b^m c^n d^m$) and
- duplication $\{ww \mid w \in (a \cup b)^*\}$.

For exampleShieber (1985) proposes that the subordinate clauses of Swiss German carry a syntax containing discontinuent constituents (crossed agreement). The same has been claimed for Dutch by Joshi (1985).

#### Language of a CFG

The reflexive and transitive closure of $\Rightarrow$ is written as $\Rightarrow^*$. The language of a category $A$ is then

$$\mathcal{L}(A) = \{w \in \Sigma^* | A \Rightarrow^* w\}$$

The language recognized by a grammar $G$ is $\mathcal{L}(G)$ which equals $\mathcal{L}(S)$ iff $S$ is the starting category of $G$.

### 2.2.2 Decorated Context-Free Grammar

The context-free approximation described in section 3.3 uses a form of CFG with decorated rules. The decoration consists of a name for the rule and subscripting each non-terminal in the right-hand side in order to facilitate implementation. The example CFG as a Decorated CFG is shown in figure 2.2.

In all other respects a Decorated CFG (DCFG) can be seen as any other straightforward CFG.

***Example*** The following context-free rule

Figure 2.2: A Decorated CFG
The example CFG in figure 2.1 as a Decorated CFG

$$
\begin{array}{rcl}
s : S & \rightarrow & NP_1,\ VP_2 \\
np : NP & \rightarrow & D_1,\ N_2 \\
np : NP & \rightarrow & N_1 \\
vp : VP & \rightarrow & V_1,\ NP_2 \\
d : D & \rightarrow & a \\
d : D & \rightarrow & many \\
n : N & \rightarrow & lion \\
n : N & \rightarrow & lions \\
n : N & \rightarrow & fish \\
v : V & \rightarrow & eat \\
v : V & \rightarrow & eats
\end{array}
$$

$$S \rightarrow NP,\ VP$$

can be decorated to

$$s : S \rightarrow NP_1,\ VP_2$$

*many lions eat fish* is an example of a sentence generated by the decorated grammar. See figure 2.3 its syntactical structure.

### 2.2.3 Mildly Context-Sensitive Grammars

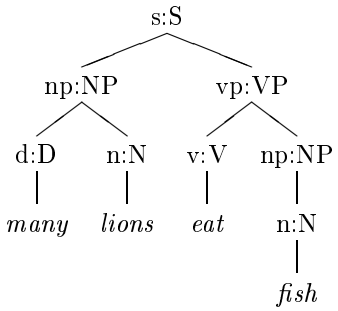
Several grammar formalisms have evolved under the name Mildly Context-Sensitive grammars, a term coined by Joshi (1985).

#### Expressivity and complexity of Mildly Context-Sensitive Grammars

Mildly Context-Sensitive grammars form a subclass of Context-Sensitive grammars (Chomsky, 1959) and have the following properties:

1. They can express any context-free language.
2. They have constant growth property (when ordered by increasing length the sentences of a language do not differ by more than a constant).
3. They can be parsed in polynomial time (with respect to the length of the input).
4. They can express multiple agreement, crossed agreement and duplication.

Figure 2.3: Parse tree
The string *many lions eat fish* is generated by the decorated grammar. The syntactical structure of the sentence is shown below.




The first and fourth of these properties are true for all Context-Sensitive grammars but the second and third properties are not. The benefit of a Mildly Context-Sensitive grammar is that it can express features beyond the expressivity of CFG:s without having the full-scale time-consumption of Context-Sensitive grammars.

In order to give formal bounds on expressivity the properties in the last point can be defined in the following way:

- $k$-multiple agreement: $a_1^m \dots a_k^m$
- $k$-crossed agreement: $a_1^{m_1} \dots a_k^{m_k} b_1^{m_1} \dots b_k^{m_k}$
- $k$-duplication: $\{w^k | w \in (a \cup b)^*\}$

With these definitions, a CFG is capable of expressing at most 2-multiple agreement, 1-crossed agreement and 1-duplication. The mildly context-sensitive grammar formalism Tree Adjoining Grammars (TAG, Joshi et al. 1975) can express 4-multiple agreement, 2-crossed agreement and 2-duplication and Multiple Context-Free Grammars (Seki et al., 1991) can express these properties for any given $k$.

There are limits to what a mildly context-sensitive grammar can handle. The language $a^{2^n}$, which gives all sequences of $a$ with length $2^n$, is such an example since it does not have a constant growth property.

### 2.2.4 Abstract and concrete syntax

Consider the context-free syntax rule for modifying a noun with an adjective

$$NP \rightarrow AP,\ N$$

(where $NP$ is the resulting noun phrase, $AP$ is the modifying adjective phrase and $N$ is the noun). The rule can be written in two ways, depending on what language the grammar shall generate.

Spanish modifies nouns by putting the adjective after the noun, *vino blanco*. In English the adjective comes first, as in *white wine*, and in French the order depends on the particular adjective in use: *bon vin* but *vin blanc*.

Thus we would need one more rule for the word order $N$, $A$ and a way of specifying when to use which rule. Alternatively, we can separate the syntax into an abstract and a concrete part.

The abstract rule would only specify which categories that can be combined into a noun phrase. The different ways of realising the abstract rule would then be described in concrete linearization rules. The grammars for Spanish, English and French would share the abstract rule but would each have their own concrete linearizations of it.

#### Advantages

There are some clear advantages of separating the abstract and the concrete syntax.

- One abstract syntax rule can have several concrete linearizations, allowing the abstract syntax to work like an interlingua between the concrete syntaxes. This works both for translating between natural languages but also between different kinds of output modes (plain text, XML documents, outputting speech synthesis etc.) for a certain piece of information.
- The abstract syntax can concentrate on the main issues and let the concrete linearizations take care of the details.

### 2.2.5 Grammatical Framework

Grammatical Framework (GF; Ranta, 2004) uses the type theory of Martin-Löf (1984) to express the semantics of natural languages, supporting higher-order functions and dependent types.

An important subclass of GF is obtained when the abstract rules are context-free, *i.e.* only contain first-order functions, and there are no dependent types. This subclass is therefore called Context-Free GF or cf-GF for short.

### 2.2.6 Generalized Context-Free Grammars

Generalized Context-Free Grammars (GCFG) were introduced as a way of describing Head Grammars (HG; Pollard, 1984). It is a Turing complete (Chomsky, 1959) formalism. Since the 1980's, GCFG has been used as a framework

for describing other grammar formalisms. One of these formalisms is Parallel Multiple Context-Free Grammars (PMCFG; Seki et al. 1991) and one of the conclusions in Ljunglöf (2004) is that it is possible to use GCFG and PMCFG to describe context-free GF.

GCFG separates the syntax into an abstract and a concrete part.

#### Abstract GCFG rules

The abstract syntax of GCFG is context-free and an abstract GCFG rule is written as

$$A \to f[A_1, \ldots, A_\delta]$$

There are two things that distinguish the abstract GCFG rule from an ordinary context-free rule. The first difference is that there can only be categories in the right-hand side of the rule. The second is the function name $f$ , which shows by which concrete rule the abstract rule is to be linearized.

#### Concrete GCFG linearizations

For every abstract function $f$ with arity $\delta$, there is one corresponding concrete linearization function $f^\circ$ defined on $\delta$ arguments

$$f^\circ(x_1, \ldots, x_\delta) = \alpha$$

The concrete syntax is made up of functions over linguistic objects. The objects are not defined in GCFG; it is up to the specific grammar formalism to define its own objects.

#### Combined GCFG rules

Sometimes it can be easier to write the abstract rule together with the concrete linearization. The combined rule is then written

$$A \to f[A_1, \ldots, A_\delta] := \alpha'$$

where $\alpha'$ is the result from substituting every $x_i$ in $\alpha$ for $A_i$.

### 2.2.7 Parallel Multiple Context-Free Grammars

Parallel Multiple Context-Free Grammars (PMCFG; Seki et al. 1991) are instances of Generalized Context-Free Grammars. In PMCFG the linguistic objects are defined as tuples of strings and the functions are defined using string concatenation. As we have seen, tuples can be replaced by equivalent records ( 2.1.2 on page 4), so we use records of linearization information as linguistic objects. An abstract PMCFG rule looks just like an abstract GCFG rule.

An example of a PMCFG can be found in figure 2.4 on page 13.

### Linearization records

A linearization record is a record of linearization rows. A linearization row is in turn a list of symbols, and a symbol is either a terminal or a projection of a category.

The terminals' linearization information depends on their types. Since the terminals are strings in PMCFG, the terminals will be linearized by concatenation. The categories are given their linearization information by record projections. And record projections have a recursive structure, in the end giving a category a string linearization.

A linearization record only containing terminals is a fully instantiated linearization record. We denote linearization rows by $\alpha$ or $\beta$. A linearization row has the same purpose as the right-hand side of a CFG rule: It tells us how the left-hand side is going to be linearized. A sequence of linearization rows is denoted by $\Phi$ or $\Psi$. For convenience we sometimes write the linearization record

$$\{s_1 = V.s_1 \;\; NP_2.s;\; s_2 = V.s_2 \; NP_2.s\}$$

as

$$s_1 = V.s_1 \; NP_2.s,\; s_2 = V.s_2 \; NP_2.s$$

***Example*** Consider the concrete linearization record (from figure 2.4)

$$s_1 = V.s_1 \; NP_2.s,\; s_2 = V.s_2 \; NP_2.s$$

it has two rows, one for the label $s_1$ and one for $s_2$. The projection $V.s_2$ is an unbound variable, dependent on the value paired with $s_2$ in the linearization row for $V$. Because of the recursive nature of projections, sooner or later the value will be a terminal and $V.s_2$ instantiated as a string.

### Concrete PMCFG linearizations

To every abstract function $f$ there is a linearization function $f^\circ$ returning a linearization record

$$f^\circ(x_1 \ldots x_\delta) = \{r_1 = \alpha_1; \ldots; r_n = \alpha_n\}$$

### Combined PMCFG rules

We can write the abstract rule and the concrete linearization as a combined rule. We then substitute every $x_i$ in $\alpha_k$ for $A_i$

$$\left.\begin{array}{r} A \to f[A_1,\, A_2] \\ f^\circ(x_1,\, x_2) = \{r = x_1.r'\; a \\ s = x_2.s'\; b\} \end{array}\right\} \quad \begin{array}{r} A \to f[A_1,\, A_2] := r = A_1.r'\; b \\ s = A_2.s'\; b \end{array}$$

Figure 2.4: An erasing PMCFG
The following grammar is taken from Ljunglöf (2004), page 59.

$$
\begin{array}{rcl}
S \to s_{sg}[NP_{sg},\ VP] & := & s = NP_{sg}.s\ VP.s_{sg} \\
S \to s_{pl}[NP_{pl},\ VP] & := & s = NP_{pl}.s\ VP.s_{pl} \\
NP_{sg} \to np_{dsg}[D_{sg},\ N] & := & s = D_{sg}.s\ N.s_{sg} \\
NP_{pl} \to np_{dpl}[D_{sg},\ N] & := & s = D_{pl}.s\ N.s_{pl} \\
NP \to np_{p}[N] & := & s = N.s_{pl} \\
VP \to vp_{csg}[V,\ NP_{sg}] & := & s_{sg} = V.s_{sg}\ NP_{sg}.s \\
 & & s_{pl} = V.s_{pl}\ NP_{sg}.s \\
VP \to vp_{cpl}[V,\ NP_{pl}] & := & s_{sg} = V.s_{sg}\ NP_{pl}.s \\
 & & s_{pl} = V.s_{pl}\ NP_{pl}.s \\
D_{sg} \to d_{a}[\,] & := & s = a \\
D_{pl} \to d_{m}[\,] & := & s = many \\
N \to n_{l}[\,] & := & s_{sg} = lion \\
 & & s_{pl} = lions \\
N \to n_{f}[\,] & := & s_{sg} = fish \\
 & & s_{pl} = fish \\
V \to v_{e}[\,] & := & s_{sg} = eats \\
 & & s_{pl} = eat
\end{array}
$$

We use subscripts to distinguish between the first and the second instance of the equivalent categories $A$ and $A$ in the rule's right-hand side. Actually all categories on the right-hand side are subscripted, so the rule

$$S \to f[A] := s = A.p\ A.q$$

is the shorthand notation for the rule

$$S \to f[A_1] := s = A_1.p\ A_1.q$$

However, since there is no way of confusing which $A$ is linearized by which label, there is no need to explicitly write out the subscripts.

### Linear grammars

If there can be at most one occurrence of each possible projection $A_i.r$ in a linearization record the PMCFG rule is linear. If all rules are linear the grammar is linear.

***Example*** In the grammar in figure 2.4 the rule

$$VP \rightarrow vp_{cpl}[V,\ NP_{pl}] \quad := \quad \begin{array}{l} s_{sg} = V.s_{sg}\ NP_{pl}.s, \\ s_{pl} = V.s_{pl}\ NP_{pl}.s \end{array}$$

is linear since no record projection occurs twice in the linearization.

#### Erasing grammars

A rule is erasing if there are argument projections that have no realization in the linearization. A grammar is erasing if it contains an erasing rule. Seki et al. (1991) have shown that it is possible to transform an erasing grammar to a non-erasing grammar. The non-erasing grammar can then be used for parsing instead of the erasing grammar.

***Example*** The grammar in figure 2.4 is erasing since the rule

$$S \rightarrow s_{sg}[NP_{sg},\ VP] := s = NP_{sg}.s\ VP.s_{sg}$$

only uses the $s_{sg}$ linearization of the $VP$:s linearization rows. The other row (labeled $s_{pl}$) is erased from the resulting linearization.

#### Linear Multiple Context-Free Grammars

If a grammar is linear it is called a Linear MCFG (LMCFG). If the grammar is non-erasing and linear it is called a Linear Context-Free Rewriting System (LCFRS, Vijay-Shanker et al. (1987)). Since there is an equivalent non-erasing grammar for every erasing grammar it is implied that LMCFG and LCFRS are equivalent grammar formalisms.

### 2.2.8 PMCFG and cf-GF are equivalent!

The result achieved by Ljunglöf (2004) is to show that cf-GF and PMCFG are equivalent formalisms. Consequently, a cf-GF can be reduced to a PMCFG and then we can use the PMCFG for parsing. However, we will not discuss how the equivalence can be proven.

## 2.3 Ranges

We use ranges in order to pinpoint partial structures for substrings in a sentence.

#### Range

A range is a pair of indices, $(i,\ j)$ in which $0 \leq i \leq j \leq |w|$, in an input string $w$. The entire string $w = w_1 \ldots w_n$ spans the range $(0, n)$. The word $w_i$ spans the range $(i-1, i)$ and the substring $w_i,\ \ldots, w_j$ spans the range $(i-1,\ j)$. A

range with identical indices, $(i, i)$, is called an empty range and spans the empty string.

We use $\rho$ to denote any range $(i, j)$.

***Example*** Given the input string *abcd*, the range for $a$ is (0, 1) and $bc$ has the range (1, 3).

#### Range records

If a record contains label-range pairs we call it a range record, $\Gamma = \{r_1 = \rho_1, \ldots, r_n = \rho_n\}$. All range records are fully instantiated, meaning there are no variables paired with the labels.

### 2.3.1 Some operations on ranges

Given the range $\rho = (i, j)$, the ceiling of $\rho$ returns an empty range for the right index

$$\lceil \rho \rceil = (j, j)$$

and the floor of $\rho$ does the same for the left index

$$\lfloor \rho \rfloor = (i, i)$$

### 2.3.2 Range concatenation

The result of concatenating two ranges $(i, j)$ and $(j', k)$ is non-deterministic, defined only when $j = j'$

$$(i, j) \cdot (j', k) = (i, k) \text{ iff } j = j'$$

### 2.3.3 Range restriction

In order to retrieve the ranges of any substring $s$ in a sentence $w = w_1 \ldots w_n$ we need to range restrict the sentence with respect to the linearization(s) for that token. Range restriction of a string $s$ with respect to $w$ is defined as:

$$\langle s \rangle^w = \{(i, j) \mid s = w_{i+1} \ldots w_j\}$$

If $w$ is understood from the context we simply write $\langle s \rangle$.

***Example*** Range restricting the terminal $a$ with respect to the string *abba* will give

$$\langle a \rangle = (0, 1) \text{ or } (3, 4)$$

Range restriction of a linearization record, $\Phi$, with respect to a sentence is written $\langle\Phi\rangle$. The result from range restricting a linearization record is that every terminal token $s$ is replaced by its range, $\langle s\rangle$. The result is of course non-deterministic since there can be several instances of a terminal in $w$, resulting in different replacements. The range restriction of two terminals next to each other fails if range concatenation fails for the resulting ranges. Any unbound variables in $\Phi$ are unaffected by range restriction.

The above holds for range restriction of any sequence of symbols. The terminals will be substituted by their ranges and the categories left as they are.

***Example*** Given the string $w = abba$ and the linearization record

$$\Phi = \{r_1 = a;\ r_2 = b; r_3 = A_1.r_4\}$$

range restriction would give

$$\begin{array}{rcl} \langle\Phi\rangle & = & \{r_1 = (0,1),\ r_2 = (1,2),\ r_3 = A_1.r'\} \\ & \text{or} & \{r_1 = (0,1),\ r_2 = (2,3),\ r_3 = A_1.r'\} \\ & \text{or} & \{r_1 = (3,4),\ r_2 = (1,2),\ r_3 = A_1.r'\} \\ & \text{or} & \{r_1 = (3,4),\ r_2 = (2,3),\ r_3 = A_1.r'\} \end{array}$$

Range restricting $\alpha = a, A, b, B$ with $w$ will return

$$\begin{array}{rcl} \langle\alpha\rangle & = & (0,\ 1),\ A,\ (1,\ 2),\ B \\ & \text{or} & (0,\ 1),\ A,\ (2,\ 3),\ B \\ & \text{or} & (3,\ 4),\ A,\ (1,\ 2),\ B \\ & \text{or} & (3,\ 4),\ A,\ (2,\ 3),\ B \end{array}$$

Range restricting $\Phi = \{r = a\, b\}$ with $abba$ gives

$$\langle\Phi\rangle \quad = \quad \{r = (0,\ 2)\}$$

The other possible solutions fail since they cannot be range concatenated.

### 2.3.4 Equivalent record types

A fully instantiated, range restricted linearization record will only contain ranges. It can therefore be seen as a range record. We say that the range record

$$\Gamma = \{r_1 = \rho_1; \ldots; r_n = \rho_n\}$$

is equivalent to the fully instantiated, range restricted linearization record

$$\Phi = \{r_1 = \rho_1; \ldots; r_n = \rho_n\}$$

## 2.4 Parsing

An introduction to parsing decorated context-free grammars using deductive agenda-driven chart-parsing.

### 2.4.1 Recognition vs Parsing

Recognition consists of determining whether the sentence $w$ is in the language generated by the grammar $G$ or not (*i.e.* $w \in \mathcal{L}(G)$ ). Parsing on the other hand consists of determining the syntactical structure of $w$ given $G$. The acquired syntactical information can in turn be used to simulate the generation of $w$.

It is obvious that the two are linked: If there is a way to generate $w$ from $G$ then $w \in \mathcal{L}(G)$. And correspondingly, if $w \in \mathcal{L}(G)$ then there is a way to generate $w$ from $G$. However recognition will return either $True$ or $False$ while parsing will return some representation of the possible syntactical structure(s) of the string.

### 2.4.2 Parsing as deduction

Parsing as deduction was introduced by Schieber, Schabes and Pereira (1995).

#### General form for inference rules

When viewing parsing as a deductive process new consequences are derived by inference rules from already acquired information. The inference rules are written as deduction rules and can have side conditions.

Given the antecedent items $A_1$ to $A_\delta$ and the side conditions *conds* the consequence item is $C$, which is written

$$\frac{A_1\ ,\ \ldots\ ,\ A_n}{C}\{conds$$

***Example*** If there is a context-free grammar rule $NP \rightarrow N$ and we already have an $N$ we can draw the conclusion that there is an $NP$

$$\frac{N}{NP}\{NP \rightarrow N$$

#### Axioms

A deduction without antecedents is always true, given that the conditions hold. Such a deduction is called an axiom. Axioms are vital for any deduction process since without them there will never be any antecedents for deriving the first consequences.

***Example*** When deductive parsing is started there are no items to derive consequences from. One way to get started is to predict from the grammar. These predictions would then be axioms. The axiom

$$\frac{}{S \rightarrow \alpha}\{S \rightarrow \alpha$$

is a prediction that says that we will find a way to linearize the context-free rule for the start category to match the input string.

### 2.4.3 Parse items

**Parse item** A parse item is a representation of a piece of information that the parsing algorithm has acquired. The items can be implemented in many ways, depending on which strategy is used for parsing.

#### Active and passive items

One way of representing the context-free rule $A \rightarrow \alpha, \beta$ is with the active item $[\rho; A \rightarrow \alpha \bullet \beta]$, where $\rho$ is a range $(i, j)$. This means that we have found everything to the left of the dot $\bullet$, $\alpha$, between $i$ and $j$, and are looking for everything to the right of the dot, $\beta$, in order to complete the entire range of $A$. An active item thus represents a partial analysis of the input and a prediction of what we might find later on.

If $\beta$ is empty, $[\rho; A \rightarrow \alpha\bullet]$, we can convert the active item to $[\rho; A]$ and call it a passive item since there is no longer anything left for it to find. A passive item represents a complete analysis of the input.

### 2.4.4 Decorated parse items

#### Decorated active items

A decorated active item has the form

$$[\rho; f : A \rightarrow [\alpha \bullet \beta]]$$

in which all categories in $\alpha$ are indexed and given with their range. Terminals are given as they are.

#### Decorated passive items

A decorated passive item is defined as having the form

$$[\rho; f : A]$$

***Example*** Given our example grammar in figure 2.2 and the sentence *many lions eat fish*, we can have the passive item $[(0, 1); d : D]$ claiming that $d : D$ has been found with the range $(0, 1)$. Or we can have the active item $[(2, 3); vp : VP \rightarrow V_1(2, 3) \bullet NP_2]$ for having found the verb in a verb phrase, with the prediction that there is an $np : NP$ starting at index 3.

Passive items for terminal rules (in which the right-hand side is empty) carry enough information to enable the construction of parse trees. Passive items for non-terminal rules do not since it is not possible to see how they came to achieve the parse information. For instance it cannot be derived from the grammar how the passive item $[(0, 4); s : S]$ came to have the range $(0, 4)$. For that we will have to use the corresponding active item. But it is possible to derive how the passive item $[(0, 1); d : D]$ came to have the range $(0, 1)$.

**Goals for recognition**

We use goal items to determine if a sentence belongs to the language of a grammar or not. This is achieved by first parsing the sentence and then checking if the goal item is in the chart. If it is, then recognition returns *True*, otherwise *False*.

Goal items are dependent on the grammar and on how the implementation of the parsing algorithm.

***Example*** In the decorated chart in figure 2.5 the passive item (40)

$$[(0,\ 4); s : S]$$

is a goal item. We could also use the corresponding active item (39)

$$[(0,\ 4); s : S \rightarrow NP_1(0,\ 2),\ VP_2(2,\ 4)\bullet]$$

### 2.4.5 Chart

In order to store the results of parsing we use a set of items called a chart. We denote the chart by $\mathcal{C}$. See figure 2.5 for an example of a decorated context-free parse chart.

Another way of looking at the chart is to describe it as a directed graph, $\mathcal{C} = (V,\ E)$, in which $V$ is the set of vertices, corresponding to the index positions, and $E$ corresponds to the parse items.

The chart will depend on both the input and the grammar. However, it will also depend on the parsing algorithm since the derived items will be different for different strategies. In figure 2.6 we give a directed graph of the passive items in figure 2.5.

The left parse tree in figure 2.3, the passive items in figure 2.5 and the directed graph 2.6 all represent the same syntactic structure. However in the chart and graph we also retain the structure with respect to the input positions.

### 2.4.6 Inference rules for Decorated CFG

There are three fundamental inference rules for a deductive chart-parsing algorithm (Kay, 1986; Wirén, 1992). The inference rules have been adapted for decorated context-free parsing. For convenience we add the inference rule Convert, which vonverts fully instantiated active items to passive ones. This makes it easier to define the inference rule Combine 2.2 and to search the chart for matching items since there will be fewer passive than active items.

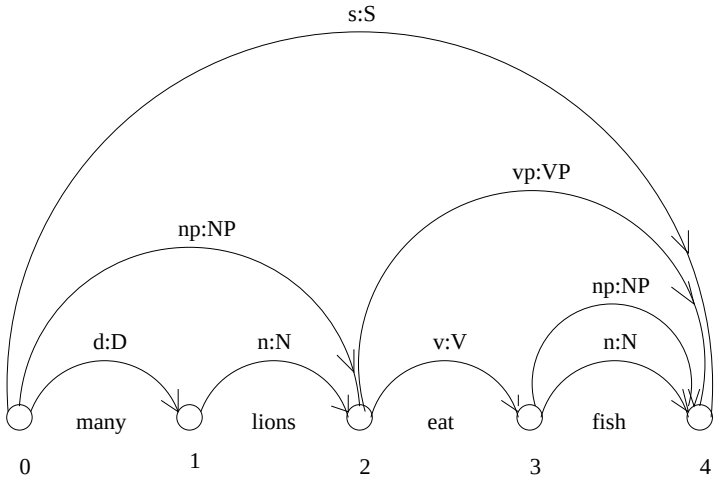
The items have the form defined in 2.4.3. It is important to remember that new items are only derived if the range concatenation succeeds. This is also the case for range restriction.

Figure 2.5: Example chart

Parsing the sentence *many lions eat fish* gives the following decorated context-free chart when using Earley filtering

| | | |
|---|---|---|
| 1 | $[(0,\ 0); s : S \rightarrow \bullet NP_1,\ VP_2]$ | *Predict* |
| 2 | $[(0,\ 0); np : NP \rightarrow \bullet N_1]$ | *Predict* |
| 3 | $[(0,\ 0); np : NP \rightarrow \bullet D_1,\ N_2]$ | *Predict* |
| 4 | $[(0,\ 0); n : N \rightarrow \bullet lion]$ | *Predict* |
| 5 | $[(0,\ 0); n : N \rightarrow \bullet lions]$ | *Predict* |
| 6 | $[(0,\ 0); n : N \rightarrow \bullet fish]$ | *Predict* |
| 7 | $[(0,\ 0); d : D \rightarrow \bullet a]$ | *Predict* |
| 8 | $[(0,\ 0); d : D \rightarrow \bullet many]$ | *Predict* |
| 9 | $[(0,\ 1); d : D \rightarrow many\bullet]$ | *Scan 8* |
| 10 | $[(0,\ 1); d : D]$ | *Convert* 9 |
| 11 | $[(0,\ 1); np : NP \rightarrow D_1(0,\ 1) \bullet N_2]$ | *Combine* 3, 10 |
| 12 | $[(1,\ 1); n : N \rightarrow \bullet lion]$ | *Predict* |
| 13 | $[(1,\ 1); n : N \rightarrow \bullet lions]$ | *Predict* |
| 14 | $[(1,\ 1); n : N \rightarrow \bullet fish]$ | *Predict* |
| 15 | $[(1,\ 2); n : N \rightarrow lions\bullet]$ | *Scan* 13 |
| 16 | $[(1,\ 2); n : N]$ | *Convert* 15 |
| 17 | $[(0,\ 2); np : NP \rightarrow D_1(0,\ 1),\ N_2(1,\ 2)\bullet]$ | *Combine* 11, 16 |
| 18 | $[(0,\ 2); np : NP]$ | *Convert* 17 |
| 19 | $[(0,\ 2); s : S \rightarrow NP_1(0,\ 2) \bullet VP_2]$ | *Combine* 1, 18 |
| 20 | $[(2,\ 2); vp : VP \rightarrow \bullet V_1,\ NP_2]$ | *Predict* |
| 21 | $[(2,\ 2); v : V \rightarrow \bullet eat]$ | *Predict* |
| 22 | $[(2,\ 2); v : V \rightarrow \bullet eats]$ | *Predict* |
| 23 | $[(2,\ 3); v : V \rightarrow eat\bullet]$ | *Scan* 21 |
| 24 | $[(2,\ 3); v : V]$ | *Convert* 23 |
| 25 | $[(2,\ 3); vp : VP \rightarrow V_1(2,\ 3) \bullet NP_2]$ | *Combine* 20, 24 |
| 26 | $[(3,\ 3); np : NP \rightarrow \bullet N_1]$ | *Predict* |
| 27 | $[(3,\ 3); np : NP \rightarrow \bullet D_1,\ N_2]$ | *Predict* |
| 28 | $[(3,\ 3); n : N \rightarrow \bullet lion]$ | *Predict* |
| 29 | $[(3,\ 3); n : N \rightarrow \bullet lions]$ | *Predict* |
| 30 | $[(3,\ 3); n : N \rightarrow \bullet fish]$ | *Predict* |
| 31 | $[(3,\ 3); d : D \rightarrow \bullet a]$ | *Predict* |
| 32 | $[(3,\ 3); d : D \rightarrow \bullet many]$ | *Predict* |
| 33 | $[(3,\ 4); n : N \rightarrow fish\bullet]$ | *Scan* 30 |
| 34 | $[(3,\ 4); n : N]$ | *Convert* 33 |
| 35 | $[(3,\ 4); np : NP \rightarrow N_1(3,\ 4)\bullet]$ | *Combine* 26, 34 |
| 36 | $[(3,\ 4); np : NP]$ | *Convert* 35 |
| 37 | $[(2,\ 4); vp : VP \rightarrow V_1(2,\ 3),\ NP_2(3,\ 4)\bullet]$ | *Combine* 25, 36 |
| 38 | $[(2,\ 4); vp : VP]$ | *Convert* 37 |
| 39 | $[(0,\ 4); s : S \rightarrow NP_1(0,\ 2),\ VP_2(2,\ 4)\bullet]$ | *Combine* 19, 38 |
| 40 | $[(0,\ 4); s : S]$ | *Convert* 39 |

Figure 2.6: The chart as directed graph
The edges are the passive items from the chart in figure 2.5. On top of the edge we have the left-hand side and underneath is the syntactical structure of the right-hand side.



**Predict**

$$\frac{}{[(i\,i); f : A \rightarrow \bullet\beta]} \left\{ \begin{array}{l} f : A \rightarrow \beta \\ 0 \leq i \leq |w| \end{array} \right. \tag{2.1}$$

The axioms as given by Predict. Prediction gives an item for each rule in $\mathcal{R}$ with an empty range for every input position $0 \leq i \leq |w|$.

**Combine**

$$\frac{[\rho'; f : A \rightarrow \alpha \bullet B_i\ \beta] \quad [\rho''; g : B]}{[\rho; f : A \rightarrow \alpha\ B_i \rho'' \bullet \beta]} \{ \rho \in \rho' \cdot \rho'' \tag{2.2}$$

If there is an item for the rule $f : A \rightarrow \alpha B \beta$ having found $\alpha$ within $\rho'$ and a passive item for the category $B$ spanning the range $\rho''$ we can add a new item to the chart, where $\alpha B$ has the range $\rho$.

**Scan**

$$\frac{[\rho'; f : A \rightarrow \alpha \bullet s\beta]}{[\rho; f : A \rightarrow \alpha s \bullet \beta]} \left\{ \begin{array}{l} \rho \in \rho' \cdot \langle s \rangle \end{array} \right. \tag{2.3}$$

If there is an item for the rule $f : A \rightarrow \alpha s \beta$ with the range $\rho'$, where the next token is a terminal, we can add a new item where $\alpha s$ spans $\rho' \cdot \langle s \rangle$.

For convenience, the fully instantiated active items are converted to passive items.

**Convert**

$$\frac{[\rho; f : A \rightarrow \beta\bullet]}{[\rho; f : A]} \tag{2.4}$$

Fully traversed active items are converted to passive items.

Prediction is very blunt. It predicts an item for every rule at every input position. This gives a vast number of useless items, especially if the number of rules and/or the size of the input is very large.

### 2.4.7 Earley prediction

This filtering technique was introduced by Earley (1970) and is a top-down strategy. Instead of predicting every possible rule at every possible input position Earley limits the predictions by only predicting a new item when an old one is looking for it.

**Predict**

$$\frac{[\rho'; g : C \rightarrow \gamma \bullet A\alpha]}{[\rho; f : A \rightarrow \bullet\beta]} \left\{ \begin{array}{l} f : A \rightarrow \beta \\ \rho = \lceil \rho' \rceil \end{array} \right. \tag{2.5}$$

Only predict an item for the rule $f : A \rightarrow \beta$ when there already is an active item looking for $A$. The new item's range is the ceiling of the antecedent item's range.

**Initial Prediction**

$$\frac{}{[(0,\,0) : f : S \rightarrow \bullet\alpha]}\{f : S \rightarrow \alpha \tag{2.6}$$

Predict an item spanning $(0, 0)$ for every rule in $\mathcal{R}$ where the left-hand side of the rule is a start-category.

Combine and Scan are included as inference rules numbers 2.2 and 2.3.

### 2.4.8 Kilbury prediction

Another filtering strategy is the one proposed by Kilbury (1985), using a bottom-up approach. An item is only predicted for a grammar rule if the rule looks for a category that already has been found.

This prediction strategy is also called left-corner parsing (as in Carroll, 2003).

**Predict+Combine**

$$\frac{[\rho; g : B]}{[\rho; f : A \rightarrow B\rho \bullet \beta]}\{f : A \rightarrow B_i\beta \tag{2.7}$$

Given a passive item for $B$ and a rule in which $B$ is the first element of the right-hand side we can add a new item for the rule, searching for the rest of the right-hand side.

**Predict+Scan**

$$\frac{}{[\rho; A \rightarrow s \bullet \beta]} \left\{ \begin{array}{l} f : A \rightarrow s\beta \\ \rho \in \langle s \rangle \end{array} \right. \tag{2.8}$$

For every rule with a substring as the first element in the right-hand side, add an active item for the rule spanning the substring, looking for the rest of the right-hand side.

Combine and Scan are included as inference rules numbers 2.2 and 2.3.

### 2.4.9 Implementing parsing as deduction

The actual implementation will depend on the grammar, the parsing algorithm and of course the goal for parsing.

As long as the deduction process enumerates all derivable items it is of no interest in which order they are produced. However, for efficiency reasons, we do not want to enumerate an item more than once. Therefore the chart has to be implemented as a set, only caching one instance of every item.

New items are added to the chart as they are derived by the inference rules. Since each new item can in itself have new items as it's consequence all new items are stored in a seperate data-structure called an agenda. When an item is removed from the agenda, all its consequences are derived. They are added to the chart and agenda, if they are not already in the chart. This procedure

Figure 2.7: An agenda-driven chart parsing algorithm for recognition

```
algorithm      : Agenda-driven Chart parsing
input          : Initial Items derived from Axioms
output         : True / False
data structures: Chart, a set of Items
                 Agenda, a collection of Items

initialize:
     Chart to set of Initial Items ;
     Agenda to collection of Initial Items ;

while Agenda not empty :
     remove a Trigger Item from Agenda ;
     compute all Consequence Items of Trigger Item ;
     for each Consequence Item :
          if Consequnce Item not in Chart :
               then: Add Consequence Item to Chart and Agenda ;

if Goal Item in Chart :
     then: True ;
     else: False ;
```

is iterated until there are no more items in the agenda. The resulting chart will then consist of all the syntactical information that can be derived from the sentence with respect to the grammar.

An algorithm for agenda-driven chart parsing can be found in figure 2.7.

## 2.5 Polynomial PMCFG parsing strategies

Ljunglöf (2004) proposes four main strategies for parsing PMCF grammars. The strategies have in turn different filtering techniques or versions. For an extensive description, see chapter 4 in Ljunglöf (2004).

### 2.5.1 Naïve algorithm

This is a naïve algorithm with a passive and an active version. The algorithm follows a straightforward bottom-up procedure, combining parse items with ranges covering smaller parts of the string to parse items with larger covering.

### 2.5.2 Context-free approximation algorithm

For this strategy the PMCFG is converted to a Decorated CFG. Parsing with the DCFG can then be carried out using any context-free algorithm. The decorated

context-free approximation might give items that are incorrect since the DCFG is overgenerating. Therefore the resulting chart needs to be filtered in a recovery step.

The complete but unsound decorated context-free chart is recovered in two steps. First the decorated context-free chart is transformed into a PMCFG chart. Then the items are combined into items with discontinuous constituents according to the original PMCFG in a way similiar to the one proposed for the Naïve algrotithm.

### 2.5.3 Active parsing algorithm

For the Active algorithm, an item is predicted for every possible range restriction of every linearization record. The linearization rows of the items are traversed by scanning and combining. Whenever a row has been fully instantiated, the next row in the linearization record is traversed until there are no more linearization rows.

Just as for context-free parsing, it can be unnecessary and time consuming to predict an item for every rule in the grammar, so adaptions of the two filtering strategies Earley and Kilbury to PMCF grammars are proposed.

### 2.5.4 Incremental parsing algorithm

An incremental parsing algorithm reads one token at a time from the input string and computes all possible consequences from that token before reading the next token.

The proposed strategy is similar to the Active parsing algorithm above with one important difference: For the Active algorithm an item is predicted for every possible range restriction of every linearization record. However, since the tokens are read incrementally (and therefore the order of the tokens is unknown) there has to be an item for every possible range restriction of a linearization row. The same procedure, and argument, goes for completion.

If massive and time consuming prediction was a problem for the Active algorithm it is an even bigger problem for the Incremental algorithm. Therefore a way of implementing Earley and Kilbury filtering is proposed. This should make the parsing process more time efficient.

# Chapter 3

# Implementation

There has not been enough time to implement all variants of the proposed algorithms. Both the Naïve and the Context-free approximation algorithms are proposed with an active and a passive version. Only the active versions have been implemented. The Active algorithm is implemented with both Earley and Kilbury predictions. The Incremental algorithm is implemented but none of the proposed prediction strategies are.

### Examples

All algorithms are explained with an example section, where we parse the sentence *abcd* with respect to the grammar in figure 3.1. For the Naïve, Approximative and Active algorithms the parse chart is given in full. However, for the Incremental algorithm this would take too much space so only an abbreviated example run is given.

The examples are given in the same notation as the algorithms. For those interested, the algorithms can be found in code in Appendix A.

### Items

Just as for the context-free parse items in 2.4.3 on page 18, it is not possible to derive parse trees from passive items for non-terminal rules, only for terminal rules.

In section 2.4.6 on page 19 we range restricted the terminals as they were scanned. For the implemented algorithms range restriction is carried out at the same time as prediction. This means that the items in the inference rules will have ranges instead of terminals in their linearization records. A consequence is that only rules that can be range restricted will be predicted as items, possibly making the chart smaller.

Figure 3.1: An interesting LMCFG

In order to have a small but interesting grammar for examples we use the following from Ljunglöf (2004), page 82.

$$\begin{aligned} S \to f[A] &:= s = A.p\ A.q \\ A \to g[A_1,\ A_2] &:= p = A_1.p\ A_2.p, \\ &\quad\ q = A_1.q\ A_2.q \\ A \to ac[\,] &:= p = a, \\ &\quad\ q = c \\ A \to bd[\,] &:= p = b, \\ &\quad\ q = d \end{aligned}$$

The grammar generates the language

$$\mathcal{L}(S) = \{s\ s_{hm} \mid s \in (a \cup b)^*\}$$

where $s_{hm}$ is the homomorphic mapping s.t. each $a$ in $s$ is translated to $c$, and each $b$ is translated to $d$. So, the homomorphic mapping of *abbab* equals *cddcd*. Examples of generated strings are *ac*, *abcd* and *bbaddc*. However, neither *abc* nor *abcdabcd* will be generated.
The language can not be described by a CFG since it contains a combination of multiple and crossed agreement with duplication. For instance the string *abbcdd* has multiple agreement on $a$, $b$, $c$ and $d$, crossed agreement on the pairs $a - c$ and $b - d$ respectively and a mapped duplication of the first part of the string *abb* to the second part *cdd*.

**Notations**

In some algorithms we choose to use the equivalent range record, $\Gamma$, for the fully instantiated, range-restricted linearization record, $\Phi$. This is written $\Gamma \equiv \Phi$. The equivalence is described in section 2.3.4.

A sequence $B_1, \ldots, B_\delta$ can be denoted by the more compact $\vec{B}$. The same goes for range records; $\Gamma_1, \ldots, \Gamma_n$ can be written as $\vec{\Gamma}$.

## 3.1 Adapting the algorithms to LMCFG

The original algorithms are designed for PMCFG, but since there are no such grammars in use at this time in the GF environment we have adjusted the algorithms for LMCFG. This also makes them more time efficient. The difference lies in how ranges are implemented. As we have seen (section 2.2.7 on page 13) PMCFG supports parallel linearizations for rules. In order to represent the

possibly multiple presence of the projection $A_i.r$ in the input, the proposed algorithms use sets of ranges.

For a LMCFG it is enough to represent every projection with a single range since it cannot occur more than once in any linearization record.

## 3.2 The Naïve algorithm

The first algorithm proposed by Ljunglöf is the 'Polynomial parsing for context-free GF' and it has two versions, a passive and an active. The passive version requires finding $\delta$ items for every rule $A \to f[B_1, \ldots, B_\delta] := \Phi$ in order to make a new item. Finding this subset of the chart is complicated and takes a lot of time. Therefore only the active version has been implemented.

### 3.2.1 Item form

There are two kinds of items, active and passive.

#### Active item

An active item for the rule

$$A \to f[\vec{B}] := \Psi$$

has the form

$$[A \to f[\vec{B'} \bullet \vec{B''}]; \Phi; \vec{\Gamma}]$$

in which the categories to the left of the dot $\bullet$, $\vec{B'}$, have been found with the linearizations in the list of range records $\vec{\Gamma}$. $\Psi$ is range restricted to $\Phi$.

#### Passive item

A passive item consists of a category and its range record

$$[A; \Gamma]$$

Use of passive items makes it easier to implement the algorithm and also helps when manually checking the parse result. They can be omitted with small changes to the inference rules.

### 3.2.2 Goals for recognition

Given the grammar in figure 3.1 we can now define a goal item for the Naïve algorithm for any input string $w$

$$[S;\ \{s = (0,\ |w|)\}]$$

### 3.2.3 Inference rules

The implemented rules are similiar to the ones proposed by Ljunglöf, but note that all range records are records over simple ranges.

**Predict**

$$\frac{}{[A \to f[\bullet \vec{B}]; \Phi; \quad ]} \begin{cases} A \to f[\vec{B}] := \Psi \\ \Phi \in \langle \Psi \rangle \end{cases} \tag{3.1}$$

Prediction gives an item for every rule in the grammar and the range restriction of its linearization is what it has found from the beginning. The sequence of range records is empty since none of the daughters in $\vec{B}$ have been found yet.

**Combine**

$$\frac{[A \to f[\vec{B} \bullet B_k \vec{B'}]; \Phi; \vec{\Gamma}] \quad [B_k; \Gamma_k]}{[A \to f[\vec{B}, B_k \bullet \vec{B'}]; \Phi'; \vec{\Gamma}, \Gamma_k]} \{\Phi' \in \Phi[B_k/\Gamma_k] \tag{3.2}$$

An active item looking for $B_k$ and a passive item that has found $B_k$ can be combined into a new active item. The new item has found $B_k$ and in its linearization record we substitute $B_k$ for its range. We also add the passive item's range record to the new item's record of daughters.

The active items with fully instantiated linearizations are converted to passive items.

**Convert**

$$\frac{[A \to f[\vec{B}\bullet]; \Phi; \vec{\Gamma}]}{[A; \Gamma]} \{\Gamma \equiv \Phi \tag{3.3}$$

Every fully instantiated Active item is converted into a Passive item. The fully instantiated linearization record is transformed into a range record with equivalent information.

### 3.2.4 Naïve parse chart

Figure 3.2 contains the parse chart for parsing the string *abcd* with the Naïve algorithm. Items 1 and 9 are examples of fully instantiated active items, 6 and 10 of the corresponding passive items. Prediction ensured that the four first items were added to the chart. Items 3 and 5 were combined into item 7. The active item 12 has been converted into item 13, which is the goal item for recognition. Item 11 is the combination of items 3 and 10, *i.e.* the predicted item for the rule $A \to g[A, A]$ and its corresponding passive item. It will never become fully instantiated since range concatenation always fails when the remaining projections in $\Phi_{\langle ab,cd \rangle A}$ are substituted for the ranges in the passive item's range record.

The linearization record $\Phi_{\langle a,,c \rangle b,d}$ is partially instantiated and $\Phi_g$ is the range restricted linearization record from the grammar rule $A \to g[A, A] := \Phi$. Since there are only unbound variables in $\Phi$ they carry the same information. The range record $\Gamma_{\langle b,d \rangle}$ contains the same parse information as the fully instantiated linearization record $\Phi_{\langle b,d \rangle}$.

Figure 3.2: Naïve parse chart

We get the following parse chart when parsing the string *abcd* with the grammar in figure 3.1 on page 28

| | | |
|---|---|---|
| 1 | $[A \rightarrow ac[\bullet]; \Phi_{\langle a,c\rangle}; ]$ | *Predict* |
| 2 | $[A \rightarrow bd[\bullet]; \Phi_{\langle b,d\rangle}; ]$ | *Predict* |
| 3 | $[A \rightarrow g[\bullet A,\ A]; \Phi_g; ]$ | *Predict* |
| 4 | $[S \rightarrow f[\bullet A]; \Phi_f; ]$ | *Predict* |
| 5 | $[A; \Gamma_{\langle a,c\rangle}]$ | *Convert* 1 |
| 6 | $[A; \Gamma_{\langle b,d\rangle}]$ | *Convert* 2 |
| 7 | $[A \rightarrow g[A \bullet A]; \Phi_{\langle a,c\rangle bd}; \Gamma_{\langle a,c\rangle}]$ | *Combine* 3, 5 |
| 8 | $[A \rightarrow g[A \bullet A]; \Phi_{\langle b,d\rangle ac}; \Gamma_{\langle b,d\rangle}]$ | *Combine* 3, 6 |
| 9 | $[A \rightarrow g[A,\ A\bullet]; \Phi_{\langle ab,cd\rangle}; \Gamma_{\langle a,,c\rangle}\Gamma_{\langle b,d\rangle}]$ | *Combine* 6, 7 |
| 10 | $[A; \Gamma_{\langle ac.bd\rangle}]$ | *Convert* 10 |
| 11 | $[A \rightarrow g[A \bullet A]; \Phi_{\langle ab,cd\rangle_A}; \Gamma_{\langle ac,bd\rangle}]$ | *Combine* 3, 10 |
| 12 | $[S \rightarrow f[A\bullet] : \Phi_{\langle abcd\rangle} : \Gamma_{\langle ac,bd\rangle}]$ | *Combine* 4, 10 |
| 13 | $[S; \Gamma_{\langle abcd\rangle}]$ | *Convert* 12 |

where the range records are the following

$$\begin{aligned}
\Gamma_{\langle a,c\rangle} &= \{p = (0,\ 1);\ q = (2,\ 3)\} \\
\Gamma_{\langle b,d\rangle} &= \{p = (1,\ 2);\ q = (3,\ 4)\} \\
\Gamma_{\langle ac,bd\rangle} &= \{p = (0,\ 2);\ q = (2,\ 4)\} \\
\Gamma_{\langle abcd\rangle} &= \{s = (0,\ 4)\}
\end{aligned}$$

and the range restricted linearization records are

$$\begin{aligned}
\Phi_{\langle a,c\rangle} &= \{p = (0,\ 1);\ q = (2,\ 3)\} \\
\Phi_{\langle b,d\rangle} &= \{p = (1,\ 2);\ q = (3,\ 4)\} \\
\Phi_{\langle ab,cd\rangle} &= \{p = (0,\ 2);\ q = (2,\ 4)\} \\
\Phi_{\langle abcd\rangle} &= \{s = 0,\ 4)\} \\
\Phi_{\langle a,c\rangle bd} &= \{p = (0,\ 1)\ A_1.p;\ q = (2,\ 3)\ A_1.q\} \\
\Phi_{\langle b,d\rangle ac} &= \{p = (1,\ 2)\ A_1.p;\ q = (3,\ 4)\ A_1.q\} \\
\Phi_{\langle ab,cd\rangle_A} &= \{p = (0,\ 2)\ A_1.p;\ q = (2,\ 4)\ A_1.q\} \\
\Phi_g &= \{p = A_1.p,\ A_2.p;\ q = A_1.q\ A_2.q\} \\
\Phi_f &= \{s = A_1.p,\ A_1.q\}
\end{aligned}$$

## 3.3 The Approximative algorithm

Parsing is performed in two steps in the Approximative algorithm. The first step is to parse the sentence with the LMCFG converted to a Decorated CFG. The resulting chart is then recovered in step two to a LMCFG chart.

### 3.3.1 The context-free approximation

In order to obtain the initial axioms for the deduction process, the LMCFG is converted into a DCFG which is used to make an approximative parse. The grammar conversion is done by creating a decorated context-free rule for every row in the linearization record. This means that any rule

$$A \to f[\vec{B}] := r_1 = \alpha_1, \ldots, r_n = \alpha_n$$

will give $n$ new rules

$$f : A.r_i \to \alpha_i$$

The parsing can then be completed as described in section 2.4.

***Example*** The rule

$$A \to f[\vec{B}] := r_1 = \alpha_1, r_2 = \alpha_2\,, r_3 = \alpha_3$$

will give the following context-free rules

$$\begin{aligned} f : A.r_1 &\to \alpha_1 \\ f : A.r_2 &\to \alpha_2 \\ f : A.r_3 &\to \alpha_3 \end{aligned}$$

Since the DCFG is over-generating compared to the LMCFG the returned parse chart is unsound. We therefore need to retrieve the passive items from the DCFG parse chart and check them against the LMCFG to get the discontinuous constituents and mark them for validity.

The chart of passive DCFG items is then extended by adding the items from prediction, to give the complete set of axioms.

The Approximative algorithm never range restricts. The ranges for the tokens in the input are given by the decorated context-free parsing.

A consequence of reducing a context-free GF grammar to a LMCFG is that all function names are unique. This means that every combination of an abstract rule with a concrete linearization will be distinguishable by the function name.

### 3.3.2 Items for the context-free approximation

There are two items involved when we convert the chart from the approximative parsing into axioms for the recovery step.

Figure 3.3: The LMCFG converted to a CFG

The rules of the example grammar 3.1 looks like this when converted to a Decorated CFG

$$\begin{array}{rcl} f : S.s & \rightarrow & A.p\ A.q \\ g : A.p & \rightarrow & A_1.p\ A_2.p \\ g : A.q & \rightarrow & A_1.q\ A_2.q \\ ac : A.p & \rightarrow & a \\ ac : A.q & \rightarrow & b \\ bd : A.p & \rightarrow & c \\ bd : A.q & \rightarrow & d \end{array}$$

The subscripted numbers are for distinguishing the two categories from each other, since they are equivalent. Here $A_1.q$ is a category of its own, not a record projection.

**Decorated item**

The items returned from the approximative parsing have the same form as that defined in 2.4.3 for active items

$$[\rho; f : A \rightarrow [\alpha \bullet \beta]]$$

**PreMCFG item**

We only need the function name in the item since every combination of abstract rule and concrete linearization has a unique function name

$$[f; r = \rho; \vec{\Gamma}]$$

$\vec{\Gamma}$ is extracted from a decorated item.

### 3.3.3 Converting the DCFG forest

The items in the DCFG chart are converted to preMCFG items, using the following rule

**Make PreMCFG items**

$$\frac{[\rho; f : A.r \rightarrow \beta]}{[f; r = \rho; \vec{\Gamma}]} \tag{3.4}$$

$\vec{\Gamma}$ is a partition of the daughters in $\beta$ such that,

$$\Gamma_i \quad \Leftrightarrow \quad \{r = \rho \mid B_i.r\rho \in \beta\}$$

where $\Gamma_i$, the $i$:th range record in $\vec{\Gamma}$, will consist of the label $r$ from the projection $B_i.r$ in $\beta$ and the range corresponding to $B_i.r$ in the final linearization.

***Example*** Given $\beta = A_1.r' \, \rho_{A_1}, \; A_2.r'' \, \rho_{A_2}$ then $\Gamma_1 = \{r' = \rho_{A_1}\}$ and $\Gamma_2 = \{r'' = \rho_{A_2}\}$

For the terminal rules with empty right-hand sides, $\vec{\Gamma}$ will be empty since there are no projections. For a rule with a non-empty right-hand side $\Gamma_i$ will consist of the information for the $i$:th category in the right-hand side. In total, $\vec{\Gamma}$ will have a range record for every daughter in the right-hand side.

### 3.3.4 Items for the recovery step

The recovery step uses three items.

#### Pre item

The items derived from the LMCFG have the following form

$$[A \to f[\vec{B}]; \Gamma \bullet r_i, \ldots, \, r_n; \vec{\Gamma}_\delta]$$

where $r_i \ldots r_n$ is a list of labels, $\vec{\Gamma}_\delta$ is a list of $|\vec{B}|$ range records, and $\Gamma$ is a range record for the labels $r_1, \ldots, r_{i-1}$

**Mark item** In order to recover the chart we use mark items with dotted rules and dotted records

$$[A \to f[\vec{B} \bullet \vec{B}']; \Gamma; \vec{\Gamma} \bullet \vec{\Gamma}']$$

The idea is to move categories from the right-hand side of the dot, $\bullet$, to the left at the same time as we check if the corresponding range record can be marked for correctness.

**Passive item** A passive item consists of a category and its range record

$$[A : \Gamma]$$

### 3.3.5 Goals for recognition

Given the grammar in figure 3.1 and the input string $w$ we get the goal item

$$[S; \{s = (0, \, |w|)\}]$$

### 3.3.6 Inference rules for the recovery step

#### Pre-Predict

$$\frac{}{[A \to f[\vec{B}]; \bullet r_1, \ldots, \, r_n; \vec{\Gamma_\delta}]} \left\{ \; A \to f[\vec{B}] := \Phi \right. \tag{3.5}$$

Every rule in the grammar is predicted as a Pre item. The context-free approximation gives the ranges for every token in the input, so we never need to range restrict. Instead, we use the labels $r_1, \ldots, \, r_n$ in $\Phi$ to retrieve the ranges given by the preMCFG items. $\vec{\Gamma_\delta}$ is a list of $\delta$ range records in which all records are empty.

**Pre-Combine**

$$\frac{[A \rightarrow f[\vec{B}]; \Gamma \bullet r, r_i, \ldots\ r_n; \vec{\Gamma}]\ [f; r = \rho; \vec{\Gamma}']}{[A \rightarrow f[\vec{B}]; \Gamma, r = \rho \bullet r_i, \ldots\ r_n; \vec{\Gamma}'']} \left\{ \vec{\Gamma}'' \in \vec{\Gamma} \sqcup \vec{\Gamma}' \right. \tag{3.6}$$

If there is a PreMCFG item for the function $f$ with a range for the label $r$, we can combine that PreMCFG item with a Pre item where $f$ is the function name and the next label is $r$. Then we move the dot forward. The new item has the unification of the antecedents record structures as its own structure of range records.

**Mark-Predict**

$$\frac{[A \rightarrow [\vec{B}]; \Gamma\bullet; \vec{\Gamma}]}{[A \rightarrow [\bullet\vec{B}]; \Gamma; \bullet\vec{\Gamma}]} \tag{3.7}$$

When all record labels have been found and given a range, we can start to check if the items have been derived in a valid way by marking the daughters' range records for correctness.

**Mark-Combine**

$$\frac{[A \rightarrow f[\vec{B} \bullet B_\mathrm{i}, \vec{B}']; \Gamma; \vec{\Gamma} \bullet \Gamma_\mathrm{i}, \vec{\Gamma}']\quad [B_\mathrm{i}; \Gamma_\mathrm{i}]}{[A \rightarrow f[\vec{B}, B_\mathrm{i} \bullet \vec{B}']; \Gamma; \vec{\Gamma}, \Gamma_\mathrm{i} \bullet \vec{\Gamma}']} \tag{3.8}$$

Record $\Gamma_i$ can be marked for correctness if there is a passive item for category $B_i$ that has found $\Gamma_i$.

**Convert**

$$\frac{[A \rightarrow f[\vec{B}\bullet]; \Gamma; \vec{\Gamma}\bullet]}{[A; \Gamma]} \tag{3.9}$$

Fully instantiated active items are converted to passive items.

### 3.3.7 Example of Approximative parsing

An example chart from top-down parsing the string *abcd* with the DCFG can be seen in figure 3.4. Item 8 is an example of the decorated grammar being overgenerating. The chart will have one corresponding preMCFG item for every decorated context-free item, which is given in the same figure.

Parsing *abcd* gives a chart of 32 items if the decorated context-free parsing is carried out with top-down filtering. For bottom-up filtering the resulting chart has 38 items. The chart can be seen in figure 3.5, except for the pre items. These can instead be found in figure 3.4 together with the fully instantiated items from the context-free approximation.

The preMCFG item 8 gets as far as becoming a mark item, but it will never be mark-combined since there are no passive items with the range record $\Gamma_{ab,c}$.

## 3.4 The Active algorithm

The active algorithm parses without context-free approximation.

Figure 3.4: Decorated context-free chart and equivalent preMCFG items
The following chart of fully instantiated items is derived by parsing *abcd* with the DCFG in figure 3.3.

$$
\begin{array}{ll}
1 & [(3,\,4); bd : A.q \to d] \\
2 & [(2,\,4); g : A.q \to A_1.q\,(2,\,3),\ A_2.q\,(3,\,4)] \\
3 & [(2,\,3); ac : A.q \to c] \\
4 & [(1,\,2); bd : A.p \to b] \\
5 & [(0,\,2); g : A.p \to A_1.p\,(0,\,1),\ A_2.p\,(1,\,2)] \\
6 & [(0,\,1); ac : A.p \to a] \\
7 & [(0,\,4); f : S.s \to A_1.p\,(0,\,2),\ A_1.q\,(2,\,4)] \\
8 & [(0,\,3); f : S.s \to A_1.p\,(0,\,2),\ A_1.q\,(2,\,3)]
\end{array}
$$

Converted to preMCFG items the decorated context-free items look like

$$
\begin{array}{ll}
1^* & [bd; \{q = (3,\,4)\};\,] \\
2^* & [g; \{q = (2,\,4)\}; \{q = (2,\,3)\}, \{q = (3,\,4)\}] \\
3^* & [ac; \{q = (2,\,3)\};\,] \\
4^* & [bd; \{p = (1,\,2)\};\,] \\
5^* & [g; \{p = (0,\,2)\}; \{p = (0,\,1)\}, \{p = (1,\,2)\}] \\
6^* & [ac; \{p = (0,\,1)\};\,] \\
7^* & [f; \{s = (0,\,4)\}; \{p = (0,\,2)\}, \{q = (2,\,4)\}] \\
8^* & [f; \{s = (0,\,3)\}; \{p = (0,\,2)\}, \{q = (2,\,3)\}]
\end{array}
$$

Figure 3.5: A chart for the Approximative algorithm
The chart from parsing *abcd* when the decorated context-free approximation is applied top-down. The preMCFG items are numbered $i^*$ and found in figure 3.4.

| | | |
|---|---|---|
| 1 | $[A \to bd[\,]; \bullet\{p,\, q\};\ ]$ | $Pre - Predict$ |
| 2 | $[A \to ac[\,]; \bullet\{p,\, q\};\ ]$ | $Pre - Predict$ |
| 3 | $[A \to g[A,\, A]; \bullet\{p,\, q\}; \{\}, \{\}]$ | $Pre - Predict$ |
| 4 | $[S \to f[A]; \bullet\{s\}; \{\}]$ | $Pre - Predict$ |
| 5 | $[A \to bd[\,]; \Gamma_b \bullet \{q\};\ ]$ | $Pre - Combine$ $4^*,\ 1$ |
| 6 | $[A \to ac[\,]; \Gamma_a \bullet \{q\};\ ]$ | $Pre - Combine$ $6^*,\ 2$ |
| 7 | $[A \to g[A,\, A]; \Gamma_{ab} \bullet \{q\}; \Gamma_a, \Gamma_b]$ | $Pre - Combine$ $5^*,\ 3$ |
| 8 | $[S \to f[A]; \Gamma_{\langle abcd \rangle} \bullet; \Gamma_{\langle ab,cd \rangle}]$ | $Pre - Combine$ $8^*,\ 4$ |
| 9 | $[S \to f[A]; \Gamma_{abc} \bullet; \Gamma_{ab,c}]$ | $Pre - Combine$ $7^*,\ 4$ |
| 10 | $[A \to bd[\,]; \Gamma_{\langle b,d \rangle} \bullet;\ ]$ | $Pre - Combine$ $1^*,\ 5$ |
| 11 | $[A \to ac[\,]; \Gamma_{\langle a,c \rangle} \bullet;\ ]$ | $Pre - Combine$ $3^*,\ 6$ |
| 12 | $[A \to g[A,\, A]; \Gamma_{\langle ab,cd \rangle} \bullet; \Gamma_{\langle a,c \rangle}, \Gamma_{\langle b,d \rangle}]$ | $Pre - Combine$ $2^*,\ 7$ |
| 13 | $[A \to bd[\bullet]; \Gamma_{\langle b,d \rangle}; \bullet]$ | $Mark - Predict$ 10 |
| 14 | $[A \to ac[\bullet]; \Gamma_{\langle a,c \rangle}; \bullet]$ | $Mark - Predict$ 11 |
| 15 | $[A \to g[\bullet A,\, A]; \Gamma_{\langle ab,cd \rangle}; \bullet\Gamma_{\langle a,c \rangle}, \Gamma_{\langle b,d \rangle}]$ | $Mark - Predict$ 12 |
| 16 | $[S \to f[\bullet A]; \Gamma_{\langle abcd \rangle}; \bullet\Gamma_{\langle ab,cd \rangle}]$ | $Mark - Predict$ 8 |
| 17 | $[S \to f[\bullet A]; \Gamma_{abc}; \bullet\Gamma_{ab,c}]$ | $Mark - Predict$ 9 |
| 18 | $[A; \Gamma_{\langle b,d \rangle}]$ | $Convert$ 12 |
| 19 | $[A; \Gamma_{\langle a,c \rangle}]$ | $Convert$ 13 |
| 20 | $[A \to g[A \bullet A]; \Gamma_{\langle ab,cd \rangle}; \Gamma_{\langle a,c \rangle} \bullet \Gamma_{\langle b,d \rangle}]$ | $Mark - Combine$ 15, 19 |
| 21 | $[A \to g[A,\, A\bullet]; \Gamma_{\langle ab,cd \rangle}; \Gamma_{\langle a,c \rangle}\Gamma_{\langle b,d \rangle}\bullet]$ | $Mark - Combine$ 20, 18 |
| 22 | $[A; \Gamma_{\langle ab,cd \rangle}]$ | $Convert$ 21 |
| 23 | $[S \to f[A\bullet]; \Gamma_{\langle abcd \rangle}; \Gamma_{\langle ab,cd \rangle}\bullet]$ | $Mark - Combine$ 17, 22 |
| 24 | $[S; \Gamma_{\langle abcd \rangle}]$ | $Convert$ 23 |

where the range records, $\Gamma_{\langle \ldots \rangle}$ are the same as in figure 3.2. The other range records are as follows

$$\begin{aligned}
\Gamma_a &= \{p = (0,\, 1)\} \\
\Gamma_b &= \{p = (1,\, 2)\} \\
\Gamma_{ab} &= \{p = (0,\, 2)\} \\
\Gamma_{abc} &= \{s = (0,\, 3)\} \\
\Gamma_{ab,c} &= \{p = (0,\, 2),\ q = (2,\, 3)\}
\end{aligned}$$

### 3.4.1 The range for $\epsilon$

For this algorithm we use a special kind of range, $\rho^\epsilon$, which denotes simultaneously all empty ranges $(i,\ i)$. There is an important difference between the range $(i,\ i)$ and the variable $\rho^\epsilon$ since $(i,\ i)$ is a range with identical indices, but $\rho^\epsilon$ is a constant for all empty ranges.

#### Operations on the epsilon-range

The range restriction of $\epsilon$ gives $\langle\epsilon\rangle = \rho^\epsilon$. Range concatenation of any range $\rho$ with the $\epsilon$-range gives

$$\rho \cdot \rho^\epsilon = \rho^\epsilon \cdot \rho = \rho$$

For the $\epsilon$-range, $\rho^\epsilon$, both the ceiling and the floor will return $\rho^\epsilon$.

### 3.4.2 Item form

#### Active item

Active items for the rule

$$A \rightarrow f[\vec{B}] := \Phi, r = \alpha\beta, \Phi'$$

have the form

$$[A \rightarrow f[\vec{B}]; \Gamma, r = \rho \bullet \beta', \Psi; \vec{\Gamma}]$$

where $\Gamma$ is the equivalent range record for the linearization rows in $\Phi$ and $\alpha$ has been recognized as the range $\rho$. We are still looking for the rest of the row, $\beta'$, and the remaining linearization rows $\Psi$. Both $\beta$ and $\Phi'$ are range restricted to $\beta'$ and $\Psi$ respectively. $\vec{\Gamma}$ is the list of range records containing the information about the daughters in $[\vec{B}]$.

#### Passive item

Passive items say that we have found $A$ inside $\Gamma$

$$[A; \Gamma]$$

### 3.4.3 Goals for recognition

Given the grammar in figure 3.1 and the input string $w$ we use the following goal item

$$[S;\ \{s = (0,\ |w|)\}]$$

### 3.4.4 Inference rules

**Predict**

$$\frac{}{[A \to f[\vec{B}];\ r = \rho^{\epsilon} \bullet \alpha', \beta'; \vec{\Gamma_{\delta}}]} \begin{cases} A \to f[\vec{B}] := r = \alpha, \Phi \\ \alpha', \Phi' \in \langle \alpha, \Phi \rangle \end{cases} \tag{3.10}$$

For every rule in the grammar, predict a corresponding item that has found the empty range. $\vec{\Gamma_{\delta}}$ is a list of $|\vec{B}|$ range records. All range records are empty since nothing has been found yet.

**Complete**

$$\frac{[A \to f[\vec{B}]; \Gamma, r = \rho\bullet, r' = \alpha, \Phi; \vec{\Gamma}]}{[A \to f[\vec{B}]; \Gamma, r = \rho, r' = \rho^{\epsilon} \bullet \alpha, \Phi; \vec{\Gamma}]} \tag{3.11}$$

When an item has found an entire linearization row we continue with the next row by starting it off with the empty range.

**Scan**

$$\frac{[A \to f[\vec{B}]; \Gamma, r = \rho' \bullet \rho''\alpha, \Phi; \vec{\Gamma}]}{[A \to f[\vec{B}]; \Gamma, r = \rho \bullet \alpha, \Phi; \vec{\Gamma}]} \{\rho \in \rho' \cdot \rho'' \tag{3.12}$$

Scanning is applied when the next symbol to read is a range. This range might be concatenated with the range for what the row has found so far. If range concatenation succeeds, there will be a new item with the resulting concatenation as range.

**Combine**

$$\frac{[A \to f[\vec{B}]; \Gamma, r = \rho' \bullet B_i.r'\alpha, \beta; \vec{\Gamma}] \quad [B_i; \Gamma']}{[A \to f[\vec{B}]; \Gamma, r = \rho \bullet \alpha, \beta; \vec{\Gamma}[i := \Gamma']]} \begin{cases} \rho \in \rho' \cdot \Gamma'.r' \\ \Gamma_i \subseteq \Gamma' \end{cases} \tag{3.13}$$

If the next thing to find is a projection on $B_i$, and there is a passive item where $B_i$ is the category, combination can be applied. The dot will then be moved past the projection. If $\Gamma'$ is consistent with the information the active item has for its $i$:th daughter, record substitution can be used. The range for $r$ is the concatenation of $\rho$ and the range corresponding to the projection $\Gamma'.r'$.

**Convert**

$$\frac{[A \to f[\vec{B}]; \Gamma, r = \rho\bullet; \vec{\Gamma}]}{[A; \Gamma, r = \rho]} \tag{3.14}$$

An active item that has fully recognized all its linearization rows is converted to a passive item.

### 3.4.5 Earley filtration for the Active algorithm

Earley filtration is an adaption from 2.4.7. There are three rules for Earley prediction. The Earley predictions give passive items for the terminal rules with fully range restricted linearizations. The rest of the rules are predicted as

active items. All rules with the start category as left-hand side are assumed to be non-terminal rules, so initial prediction will only give active items.

**Predict Passive**

$$\frac{[\ldots;\ldots,r=\rho'\bullet A.r\ldots;\ldots]}{[A;\Gamma]}\left\{\begin{array}{l}A\to f[]:=\Psi\\ \Gamma\equiv\langle\Psi\rangle\end{array}\right. \tag{3.15}$$

We only predict from the grammar if there is already an item looking for the left-hand side of the rule. The Passive item has the range record corresponding to the fully instantiated linearization record of the rule.

**Predict Active**

$$\frac{[\ldots;\ldots,r=\rho'\bullet A.r'\ldots;\ldots]}{[A\to f[\vec{B}];r=\rho\bullet\alpha',\Gamma;\vec{\Gamma_\delta}]}\left\{\begin{array}{l}A\to f[\vec{B}]:=r''=\alpha,\Phi\\ \alpha',\Gamma\in\langle\alpha,\Phi\rangle\\ \rho=\lceil\rho'\rceil\end{array}\right. \tag{3.16}$$

This version of prediction is applied if the right-hand side is non-empty. The new range is the ceiling of $\rho$.

**Initial prediction**

$$\frac{}{[S\to f[\vec{B}];s=(0,\,0)\bullet\Gamma;\vec{\Gamma_\delta}]}\left\{\begin{array}{l}S\to f[\vec{B}]:=s=\alpha\\ \Gamma\in\langle\alpha\rangle\end{array}\right. \tag{3.17}$$

Since there are no items at first, the parsing is initiated by predicting an item for every rule with a start category as left-hand side. $\vec{\Gamma_\delta}$ is a list of range records in which all records are empty.

Complete, Scan, Combine and Convert are included as inference rules 3.11-3.14.

### 3.4.6 Kilbury filtration for the Active algorithm

Kilbury prediction is an adaption from 2.4.8. The Kilbury predictions are limited to grammars in which terminals only occur in rules with empty right-hand sides. However, Seki et al. (1991) have shown that any PMCFG that does not fulfill this requirement can be converted to an equivalent grammar that does. An alternative would be to slightly alter the inference rules.

There are two new rules, while Complete, Combine and Convert are included as the rules 3.11, 3.13 and 3.14. Scan is replaced by Terminal.

**Predict**

$$\frac{[B_i;\Gamma_i]}{[A\to f[\vec{B}];r=\rho\bullet\alpha',\Gamma;\vec{\Gamma_\delta}[I:=\Gamma_i]]}\left\{\begin{array}{l}A\to f[\vec{B}]:=r=B_i.r'\alpha,\Phi\\ \alpha',\Gamma\in\langle\alpha,\Phi\rangle\\ \rho=\Gamma_i.r'\end{array}\right. \tag{3.18}$$

We only predict a new item for a rule, if there is a Passive item for the first category in the first linearization row. We move the dot past the category and add the Passive item's record to the new item's structure of records. The new item recieves its range from the projection $\Gamma_i.r'$.

**Terminal**

$$\frac{}{[A;\Gamma]} \begin{cases} A \rightarrow f[\,] := \phi \\ \Gamma \in \langle \phi \rangle \end{cases} \tag{3.19}$$

Every terminal rule is predicted as a passive item.

### 3.4.7 Example for the Active algorithm

The Active algorithm can be used with Earley or Kilbury filtering, or without filtering. Parsing the string *abcd* gives the following table for chart size

| Filter | Size |
|---|---|
| None | 25 |
| Earley | 20 |
| Kilbury | 15 |

The chart after parsing without prediction filters can be seen in figure 3.6.

#### Comments on the chart

Items 11 and 12 are examples of passive items; they are converted from the active items 9 and 10 respectively. Item 5 has fully traversed its first linearization row and has been completed to give item 7. Scanning item 8 gives an example of concatenation with $\rho^{\epsilon}$. The result can be seen in item 10. Item 24 is the result of combining the passive item 21 with the active item 23. Prediction gave the first four items.

Both prediction strategies result in fewer items since the terminal rules are predicted as passive items.

#### Earley

The use of Earley prediction gives a chart without items 1, 2, 4, 5, 6, 7, 8, 9, 10 and 15. Instead we get 4 items for the non-terminal rule $A \rightarrow g \ldots$ where $\rho^{\epsilon}$ is substituted for the empty ranges $(0, 0) \ldots (3, 3)$. The predicted item for the start rule will be predicted with the range (0, 0) instead of $\rho^{\epsilon}$.

#### Kilbury

If we instead use Kilbury prediction the same items are filtered out with two exceptions; item 15 will be included and item 3 will not be predicted. Instead the passive item 12 will trigger the prediction of item 14 (the combination of 3 and 12).

## 3.5 The Incremental algorithm

An incremental algorithm reads one token at a time. However our implementation does not, due to how we defined range restriction.

Figure 3.6: Active parse chart

This is the chart for parsing *abcd* with the Active algorithm without filtering. The range records $\Gamma_{\ldots}$ are the same as in figure 3.2. We also use

$$\Phi_{g_q} = \{q = A_1.q\ A_2.q\}$$

and

$$\Gamma_{ab} = \{p = (0,\ 2)\}$$

In order to fit the table on the page, the following notations are used for the inference rules $P$ = Predict, $S$ = Scan, $Cv$ = Convert, $Cp$ = Complete and $C$ = Combine.

| | | |
|---|---|---|
| 1 | $[A \to bd[\,]; p = \rho^{\epsilon} \bullet (1,\ 2)\ q = (3,\ 4);\ ]$ | $P$ |
| 2 | $[A \to ac[\,]; p = \rho^{\epsilon} \bullet (0,\ 1)\ q = (2,\ 3);\ ]$ | $P$ |
| 3 | $[A \to g[A,\ A]; p = \rho^{\epsilon} \bullet A_1.p,\ A_2.p\ \Phi_{g_q}; \{\}, \{\}]$ | $P$ |
| 4 | $[S \to f[A]; s = \rho^{\epsilon} \bullet A_1.p,\ A_2.q; \{\}]$ | $P$ |
| 5 | $[A \to bd[\,];\ p = (1,\ 2(3,\ 4);\ ]$ | $S$ 1 |
| 6 | $[A \to ac[\,]; p = (0,\ 1) \bullet q = (2,\ 3);\ ]$ | $S$ 2 |
| 7 | $[A \to bd[\,]; \{p = (1,\ 2)\},\ q = \rho^{\epsilon} \bullet (3,\ 4);\ ]$ | $Cp$ 5 |
| 8 | $[A \to ac[\,]; \{p = (0,\ 1)\},\ q = \rho^{\epsilon} \bullet (2,\ 3);\ ]$ | $Cp$ 6 |
| 9 | $[A \to bd[\,]; \{p = (1,\ 2)\},\ q = (3,\ 4)\bullet;\ ]$ | $S$ 7 |
| 10 | $[A \to ac[\,]; \{p = (0,\ 1)\},\ q = (2,\ 3)\bullet;\ ]$ | $S$ 8 |
| 11 | $[A; \Gamma_{\langle b,d\rangle}]$ | $Cv$ 9 |
| 12 | $[A; \Gamma_{\langle a,c\rangle}]$ | $Cv$ 10 |
| 13 | $[A \to g[A,\ A]; p = (1,\ 2) \bullet A_2.p\ \Phi_{g_q}; \Gamma_{\langle b,d\rangle}, \{\}]$ | $C$ 3, 11 |
| 14 | $[A \to g[A,\ A]; p = (0,\ 1) \bullet A_2.p\ \Phi_{g_q}; \Gamma_{\langle a,c\rangle}, \{\}]$ | $C$ 3, 12 |
| 15 | $[S \to f[A]; s = (1,\ 2) \bullet A_1.q; \Gamma_{\langle b,d\rangle}]$ | $C$ 4, 11 |
| 16 | $[S \to f[A]; s = (0,\ 1) \bullet A_1.q; \Gamma_{\langle a,c\rangle}]$ | $C$ 4, 12 |
| 17 | $[A \to g[A,\ A]; p = (0,\ 2) \bullet \Phi_{g_q}; \Gamma_{\langle a,c\rangle}, \Gamma_{\langle b,d\rangle}]$ | $C$ 11, 14 |
| 18 | $[A \to g[A,\ A]; \Gamma_{ab}, q = \rho^{\epsilon} \bullet A_1.q\ A_2.q; \Gamma_{\langle a,c\rangle}, \Gamma_{\langle b,d\rangle}]$ | $Cp$ 17 |
| 19 | $[A \to g[A,\ A]; \Gamma_{ab}, q = (2,\ 3) \bullet A_2.q; \Gamma_{\langle a,c\rangle}, \Gamma_{\langle b,d\rangle}]$ | $C$ 12, 18 |
| 20 | $[A \to g[A,\ A]; \Gamma_{ab}, q = (2,\ 4)\bullet; \Gamma_{\langle a,c\rangle}, \Gamma_{\langle b,d\rangle}]$ | $C$ 11, 19 |
| 21 | $[A; \Gamma_{\langle ab,cd\rangle}]$ | $Cv$ 20 |
| 22 | $[A \to g[A,\ A]; p = (0,\ 2) \bullet A_2.p\ \Phi_{g_q}; \Gamma_{\langle ab,cd\rangle}, \{\}]$ | $C$ 3, 21 |
| 23 | $[S \to f[A]; s = (0,\ 2) \bullet A_1.q; \Gamma_{\langle ab,cd\rangle}]$ | $C$ 4, 21 |
| 24 | $[S \to f[A]; s = (0,\ 4)\bullet; \Gamma_{\langle ab,cd\rangle}]$ | $C$ 23, 21 |
| 25 | $[S; \Gamma_{\langle abcd\rangle}]$ | $Cv$ 24 |

### 3.5.1 Incrementality and range restriction

Range restriction, as we have defined it, cannot handle partial results. Either the symbol is a terminal and can be substituted by its range or it is an unbound variable and is left as it is. When parsing incrementally this definition is not sufficient. For instance, the range restriction $\langle p = a,\ q = c \rangle^a$ will fail since there is no range for $c$ in the string $a$. We would need a new definition of range restriction, allowing partial results.

To get round the problem of not having partial range restriction we range restrict the entire input from the start. This means that the algorithm will no longer be truly incremental. This compromise has advanteges. The efficiency of an incremental algorithm depends to some degree on how fast the user provides the input. In order to test how the inference rules compare to the other algorithms it can be easier if it is in a static environment, and not given input a token at a time.

### 3.5.2 Item form

#### Active items

The only item form is active

$$[A \to f[\vec{B}]; \Gamma, r = \rho \bullet \beta, \psi; \vec{\Gamma}]$$

The items have the same form as the active items used in the Active algorithm. However we use the notation $[A; \Gamma, r = \rho]$, where $r = \rho$ is the latest recognized row, for the item $[A \to \ldots; \Gamma, r = \rho \bullet \phi; \ldots]$ and call the item passive. Note that there are no passive items implemented and that any item with a fully instantiated row is called passive, even if there are more rows to instantiate.

### 3.5.3 Goals for recognition

For the example grammar 3.1 on page 28 we get the following goal item

$$[S \to f[A];\ s = (0,\ |w|); \{p = (0,\ \frac{|w|}{2});\ q = (\frac{|w|}{2},\ |w|)\}]$$

We use an active item as goal item since there are no passive items.

### 3.5.4 Inference rules

#### Notations

If we only want to be sure that two items have the same abstract rule, we denote the rule by $R$.

**Predict**

$$\frac{}{[A \to f[\vec{B}]; r = (i,\, i) \bullet \alpha', \Phi', \Psi'; \vec{\Gamma}_\delta]} \left\{ \begin{array}{l} A \to f[\vec{B}] := \Phi, r = \alpha, \Psi \\ \alpha', \Phi', \Psi' \in \langle \alpha, \Phi, \Psi \rangle \\ 0 \leq i \leq |w| \end{array} \right. \tag{3.20}$$

An item is predicted for every linearization row and every input position. $\vec{\Gamma}_\delta$ is a list of range records of length $\delta$ in which all records are empty.

**Complete**

$$\frac{[R; \Gamma, r = \rho \bullet \Phi, r' = \alpha, \Psi; \vec{\Gamma}]}{[R; \Gamma, r = \rho, r' = (k,\, k) \bullet \alpha, \Phi, \Psi; \vec{\Gamma}]} \left\{ \begin{array}{l} \rho = (i,\, j) \\ j \leq k \leq |w| \end{array} \right. \tag{3.21}$$

Whenever a linearization row is fully traversed completion is applied. This means that an item is predicted for every remaining linearization row and every remaining input position between the range of the traversed row and the end of the input.

**Scan**

$$\frac{[R; \Gamma, r = \rho' \bullet \rho'', \alpha, \Phi; \vec{\Gamma}]}{[R; \Gamma, r = \rho \bullet \alpha, \Phi; \vec{\Gamma}]} \{\rho \in \rho' \cdot \rho'' \tag{3.22}$$

If the next item in the linearization row is a range, it is concatenated to the range for the partially recognized row.

In the Active algorithm the inference rule Convert 3.14 added the last label-range pair to the range record for the passive item. In the absence of passive items we just have to remember that there is such a pair when combining.

**Combine**

$$\frac{[R; \Gamma, r = \rho' \bullet B_i.r'\, \alpha, \Phi; \vec{\Gamma}]\; [B_i; \Gamma', r' = \rho'']}{[R; \Gamma, r = \rho \bullet \alpha, \Phi; \vec{\Gamma}[i := (\Gamma', r' = \rho'')]]} \left\{ \begin{array}{l} \rho \in \rho' \cdot \rho'' \\ \Gamma_i \subseteq (\Gamma', r' = \rho'') \end{array} \right. \tag{3.23}$$

Combining is applied if the next item is a record projection and there is a passive item for the corresponding category. The information in the $i$:th range record of $\vec{\Gamma}$ must be consistent with the information found for the passive item. This can be checked by a subset check since the range record of the passive item must be fully instantiated.

### 3.5.5 Example run

Parsing the sentence *abcd* with the Incremental algorithm results in a chart with 78 items. Therefore the example run will only briefly explain the inference rules. The large number of items is a consequence of using $(i,\, i)$-ranges instead of $\rho^\epsilon$(section 3.4.1), and of predicting items for every linearization row.

### Prediction

Prediction is crude. The grammar rule

$$A \rightarrow g[A\,A] := p = A_1.p\,A_2.p,\ q = A_1.q\,A_2.q$$

will be predicted as ten different items, one item for every row and for every input position $0 \leq i \leq 4$. Examples of such items are

$$[A \rightarrow g[A\,A]; p = (2,\,2) \bullet A_1.p\,A_2.p\,\{q = A_1.q\,A_2.q\}; \{\}, \{\}]$$

and

$$[A \rightarrow g[A\,A]; q = (2,\,2) \bullet A_1.q\,A_2.q\,\{p = A_1.p\,A_2.p\}; \{\}, \{\}]$$

This holds for every rule, all in all predicting 35 items. The terminal rules will have ranges instead of projections in the linearization record.

### Complete

The above holds also for completion. When a linearization row is fully instantiated to a range, an item is predicted for every remaining row and input position. For example if the last row was instantiated to the range (1, 3), then in our case this would give two possible ranges, (3, 3) and (4, 4), for every row. The chart contains 16 items as a consequence of completion.

### Scan

Scanning is carried out in exactly the same way as in the Active algorithm.

### Combine

Combining is also performed in the same way as in the Active algorithm, but with an important difference. In the incremental algorithm the range of the row to be traversde is always known. Therefore it is always possible to give a partial index for the items to combine. Thus the active item

$$[\ldots; \ldots r = (i,\,3) \bullet \{B_i.r' \ldots\,; \ldots]$$

can only be combined with a passive item

$$[B \rightarrow \ldots; \ldots r' = (3,\,j) \bullet \phi; \ldots]$$

In the Active algorithm it was not always the case that the range was known. Therefore we could not be as explicit in looking for items to combine. This makes the Incremental inference rule for combining more efficient since we can limit our search space. However, this will not show in runtimes and chart size until predicting is more economic. Until then there will be far more items to combine with at every input position.

### 3.5.6 Proposed prediction strategies

There was not enough time to implement the proposed Earley or Kilbury filter.

#### Earley

The Earley prediction consists of three rules for predicting, completion and initial prediction. Initial prediction is the same as the rule number 3.17, returning items for every rule where the left-hand side is a start category. Prediction and completion predict new items for grammar rules when the left-hand side of the rule is searched for by an existing item.

#### Kilbury

New items are only predicted for linearization rows in which the first symbol has already been found. At the same time the dot can be moved forward. There are two rules for predicting to be combined with both Scan and Combine, giving four new inference rules for Kilbury filtering.

# Chapter 4

# Small-scale evaluation

The algorithms have not been tested for realistic grammars and large corpi as part of this thesis. This is due to that there was not enough time to create big enough grammars and corpuses to test against. Some preliminary tests have been conducted and it is possible to show how the algorithms compare for the grammar in figure 3.1.

It is not possible to draw any conclusions from the tables on the general performance of the algorithms. They are for showing how they perform compared to each other for a very small grammar.

## 4.1 Preliminary testing

The parser used today in the GF-library converts the given grammar to a CFG in order to make an approximate parse. As we have seen the CFG will at most certainly be overgenerating. Therefore a recovery step is used just as in the Approximative algorithm ( 3.3 on page 32). However, instead of using the strategy we used, the GF-parser recovers the parse-result tree-by-tree.

In preliminary tests the implemented algorithms are more efficient than the original GF-parser. A grammar for English was used in the tests. It consists of roughly 500 GF rules. Converted to LMCFG rules this gives a grammar with approximately 22.000 rules or 20.000 rules if converted to a CFG.

***Example*** Parsing the randomely generated sentence *you had begged to die here* gives roughly 60.000 context-free trees. After recovery only 6 remain. It is the recovering of the context-free chart that makes the GF parser slower than our implemented algorithms.

## 4.2 Parse table

### 4.2.1 Efficiency for correct sentences

In figure 4.1 we give the result from parsing sentences of lengths 6, 12 and 24 terminal tokens. Parsing is carried out with respect to the grammar in figure 3.1. This is by no means an extensive evaluation of the algorithms, but does illustrate how they perform for a small grammar. Their performance could very well turn out to be quite different when using a larger or more context-free grammar. The used grammar generates sentences with multiple and crossed agreement combined with duplication, all features outside the expressivity of a CFG.

Figure 4.1: Evaluation of valid sentences
Chart sizes and running times for parsing strings of various lengths. The strings are valid, such as *abbacddc*. All strings are parsed with respect to the LMCFG on page 28. Times are in milliseconds and chart size is given in number of items.

| | | Length | | |
|---|---|---|---|---|
| | | 6 | 12 | 24 |
| Naïve | Chart | 31 | 137 | 834 |
| | Time | <1 | 10 | 110 |
| Approx | Chart | 116 | 2980 | 170216 |
| bottom-up | Time | 20 | 70 | 9910 |
| Approx | Chart | 96 | 2888 | 169818 |
| top-down | Time | <1 | 100 | 9670 |
| Active | Chart | 78 | 663 | 8207 |
| no filter | Time | <1 | 40 | 2650 |
| Active | Chart | 59 | 574 | 7778 |
| Earley | Time | 10 | 120 | 7630 |
| Active | Chart | 56 | 589 | 7917 |
| Kilbury | Time | <1 | 40 | 2560 |
| Incremental | Chart | 254 | 2375 | 34813 |
| | Time | 10 | 150 | 7760 |

### 4.2.2 Comments

#### The Naïve algorithm

The Naïve algorithm is by far the most efficient algorithm. It would be interesting to see how it performs for much larger grammars.

#### The Approximative algorithm

It seems to make no difference if we implement the decorated context-free approximation with bottom-up or top-down prediction. Even if the resulting charts are very big compared to the other algorithms, the run-times do not grow to the same extent. One reason for the large chart size is that we use four different kinds of items. A lot of information is duplicated as it is passed from one kind of item to another. Many of the items are also derived from context-free parsing which is quicker than parsing mildly context-sensitive grammars.

#### The Active algorithm

Using Earley prediction for the Active algorithm gives fewer items but makes parsing a lot slower. Kilbury gives the same reduction on chart size and a slightly quicker parsing compared to using no filtering. Remember, it is not possible to say anything about the performance of the different prediction strategies until they have been tested on much larger grammars.

Both prediction strategies result in fewer items as a consequence of predicting passive items for terminal rules. This can turn out to be even more efficient for grammars with a big percentage of terminal rules. A possible explanation for the poor behaviour of Earley can be that the gain of top-down prediction is lost on such a small grammar and the use of empty ranges, $(i, i)$, instead of the $\epsilon$-range, $\rho^{\epsilon}$.

#### The Incremental algorithm

The performance of the incremental algorithm improves as the size of the input grows, compared to the Approximative algorithm. Otherwise it is slow and memory demanding.

### 4.2.3 Efficiency for incorrect sentences

Chart sizes and runtimes are not only interesting when the sentence is valid. It is just as interesting to have quick, memory efficient parsing algorithms if the sentence is invalid. In the table in figure 4.2 all the implemented algorithms are faster and derive less items when recognizing invalid sentences.

Especially the Approximative algorithm is much faster for rejecting sentences in which a $c$ or $d$ has been substituted for $a$ or $b$ respectively, in an otherwise valid sentence.

Figure 4.2: Evaluation of invalid sentences
The strings have the form *abbacdbc*, *i.e.* somewhere in the second half of the sentence a $c$ or $d$ is substituted for $a$ or $d$. Parsing is carried out with respect to the grammar in figure 3.1 on page 28. Times are in milli-seconds and chart sizes in number of items.

| | | Length | | |
|---|---|---|---|---|
| | | 6 | 12 | 24 |
| Naïve | Chart | 16 | 79 | 449 |
| | Time | <1 | 10 | 40 |
| Approx | Chart | 48 | 614 | 34063 |
| bottom-up | Time | <1 | 30 | 1390 |
| Approx | Chart | 28 | 449 | 21744 |
| top-down | Time | <1 | 20 | 810 |
| Active | Chart | 32 | 316 | 4215 |
| no filter | Time | <1 | 20 | 870 |
| Active | Chart | 11 | 238 | 3837 |
| Earley | Time | <1 | 80 | 3120 |
| Active | Chart | 14 | 246 | 3929 |
| Kilbury | Time | <1 | 20 | 890 |
| Incremental | Chart | 128 | 1538 | 21689 |
| | Time | 10 | 120 | 2490 |

# Chapter 5

# Summary

## 5.1 Future work

#### Further implementations

The implementations do not cover all proposed algorithms. There are passive versions of the Naïve and Approximative algorithms still left to do. It would be interesting to implement them for the sake of comparison.

Neither of the prediction filters have been implemented for the Incremental algorithm. The proposed Earley prediction should be easy to implement, it is very similiar to the version of Earley used for the Active algorithm. The implementation of Kilbury is probably more demanding.

Re-implementing the Incremental algorithm in a dynamic environment is also an intersting future development. This will mean that an extended range restriction has to be implemented, able to cope will partial results.

#### Evaluation

Further tests are necessary. For now, all we know is that the Naïve algorithm is suitable for very small grammars. Which algorithm to use for larger grammars cannot be decided before the algorithms have been tested on large grammars.

#### Readapting to PMCFG

The algorithms are implemeted for LMCFG. If we want to use the technique of reducing a context-free GF to a PMCFG in order to get faster parsing algorithms it is necessary to readapt the strategies to PMCFG. It might very well be that the algorithms will parse PMCFG faster than the GF parser parses cf-GF.

#### Complexity

It would be a Master thesis in its own right todetermine the complexity of the algorithms. Until proven we will just have to hope the complexity is polynomial.

#### Correctness

There has not been time to give formal proofs of the algorithms being correct. The proposed PMCFG algorithms are proved both complete and sound by Ljunglöf (2004). A comparison of his discussion with the implemented algorithms indicates that the differences are too large for just copying his proofs to our work.

For now, all we can say is that they seem to be correct.

## 5.2 Conclusion

We have implemented four algorithms for parsing Linear Multiple Context-Free Grammars. A thorough testing of the algorithms with grammars of varying sizes is necessary before any conclusions can be drawn on their overall performance. However preliminary testing indicates that the implemented algorithms parse an LMCF grammar faster than the existing parser for GF parses an equivalent context-free GF grammar. It seems promising...

# Bibliography


Carroll, J. (2003). Parsing. In Mitkov, R., editor, *The Oxford Handbook of Computational Linguistics*, chapter 12, pages 233–248. Oxford University Press.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, 2:137–167.

GF (2004). The Grammatical Framework homepage. Located at `http://www.cs.chalmers.se/~aarne/GF/`

Hudak, P., Peterson, J., and Fasel, J. (1999). A gentle introduction to Haskell 98. Technical report, Yale University. Available from the Haskell web site: `http://www.haskell.org/tutorial`

Joshi, A. (1985). How much context-sensitivity is necessary for characterizing structural descriptions — tree adjoining grammars. In Dowty, D., Karttunen, L., and Zwicky, A., editors, *Natural Language Processing: Psycholinguistic, Computational and Theoretical Perspectives*, pages 206–250. Cambridge University Press, New York.

Joshi, A. K., Levy, L. S., and Takahashi, M. (1975). Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

Kay, M. (1986). Algorithm schemata and data structures in syntactic processing. In Grosz, B., Jones, K., and Webber, B., editors, *Readings in Natural Language Processing*, pages 35–70. Morgan Kaufman Publishers, Los Altos, CA.

Ljunglöf, P. (2004). *Expressivity and Complexity of the Grammatical Framework*. PhD thesis, Göteborg University and Chalmers University of Technology.

Martin-Löf, P. (1984). *Intuitionistic Type Theory*. Bibliopolis, Napoli.

Okasaki, C. (1998). *Purely Functional Data Structures*. Cambridge University Press.

Peyton Jones, S. (2003). *Haskell 98 Language and Libraries*. Cambridge University Press, New York.

Pollard, C. (1984). *Generalised Phrase Structure Grammars, Head Grammars and Natural Language*. PhD thesis, Stanford University.

Ranta, A. (2004). Grammatical Framework, a type-theoretical grammar formalism. *Journal of Functional Programming*, 14(2):145–189.

Seki, H., Matsumara, T., Fujii, M., and Kasami, T. (1991). On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Shieber, S. (1985). Evidence against the context-freeness of natural language. *Computational Linguistics*, 20(2):173–192.

Shieber, S., Schabes, Y., and Pereira, F. (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36.

TALK (2004). The Talk project homepage. Located at `http://www.talk-project.org/`

Thompson, S. (1999). *The Craft of Functional Programming, 2nd ed.* Addison-Wesley.

Vijay-Shanker, K., Weir, D., and Joshi, A. (1987). Characterizing structural descriptions produced by various grammatical formalisms. In *25th Meeting of the Association for Computational Linguistics*.

Wirén, M. (1992). *Studies in Incremental Natural-Language Analysis*. PhD thesis, Linköping University, Linköping, Sweden.

# Appendix A

# The code

To fully understand the code the reader will probably need to know at least some Haskell. There is a difference between the code developed for this thesis and the actual code implemented into GF. The main reason being that it is a lot easier to develop outside of GF and that GF has some features that extend Haskell.

The function `recognize` is grammar dependent for all algorithms as a consequence of the goal items being grammar dependent. This is apparent in the type declaration for `recognize` and the same holds for the function `parse` in algorithm `3.3`, since the information passed to the CF parser is grammar dependent. Haskell supports dependent types.

All charts are implemented as `RedBlackMap`:s, a RedBlackTree structure with key-value pairs as leaves (see Okasaki, 1998 for more information on functional data structures).

The type of a grammar is `Grammar n c l t` where `n` is the type of the function names, `c` is the type for the categories, `l` is the type for lables and `t` is the type for tokens. Hence all objects used in grammars or for parsing are dependent on one or more of these four types. Referring to the table in figure A.1 the type `Lin c l t` is the type for a linearization row. Its final type is dependent on the type of the categories, lables and tokens used in the grammar.

In the implementaion of the Example grammar in figure 3.1, the type is `String` for the function names, the lables and the tokens while the categories are of the user-defined type `NT`.

When a linearization record is range restricted, the type is redefined from `LinRec c l t` to `LinRec c l Range` and the corresponding instantiation of types occurs for the tokens; `Tok t` becomes `Tok Range` .

The `Nondet` type is used when a function can return several solutions for the given arguments. The functions for range restriction can give a number of different ranges, all depending on the arguments (see 2.3.3). The reurn value for $\langle s \rangle$ is therefore `Nondet Range`.

For every algorithm an example of the items in code are given. Some items are

Figure A.1: Types and code

The four first are the basic types of the grammar. All other types depend on the basic types, except `Nondet`.

| Code: | Used for: |
|---|---|
| `c` | Variable type for elements in $C$ |
| `l` | Variable type for labels |
| `n` | Variable typ for function names |
| `t` | Variable type for elements in $\Sigma$ |
| `Tok t` | A token of type `t` |
| `Tok Range` | A token of type `Range` |
| `Range` | The constructor for `Range ( Int, Int )` |
| `Grammar n c l t` | `= [ Rule n c l t ]` |
| `Rule n c l t` | `= Rule c [ c ] ( LinRec c l t ) n` |
| `Lin c l t` | A linearization row, |
| | `Lin l [ Symbol ( c, l Int ) t ]` |
| `LinRec c l t` | A linearization record, `[ Lin c l t ]` |
| `Symbol ( c, l, Int ) t` | `= Cat ( c, l, Int ) \| Tok t` |
| `RangeRec l` | A range record, `[( l, Range )]` |
| `NT` | The type for categories in 3.1 |
| `Cat ( A, "p", 0 )` | The category $A_0.p$, where $A$ is of the type `NT` |
| `AbstractRule n c` | `= ( n, c, [ c ] )` |
| `DottedRule n c` | `= ( n, c, [ c ], [ c ] )` |
| `Nondet` | Used when a function is non-deterministic |

so long that they are written on several lines, following the layout of how the items are defined in the algorithm.

## A.1 ExampleGrammar

This is the example grammar in figure 3.1, written in Haskell. The projections

$$A_1.p,\ A_2.q$$

are implemented as

```
[ Cat ( A, "p", 0 ), Cat ( A, "q", 1 )]
```

using the type `NT` for the categories and `String` for the labels.

All categories are indexed explicitly in the code while it was an implicit feature in the text. The indices are reduced by one to match Haskell's list indexing.

```
{-- Module ------------------------------------------------------------------
  Filename   : ExampleGrammar.hs
  Author     : Håkan Burden
  Time-stamp : <2005-03-03, 16:00>
  Description: Implementation of Example grammar 4.1
               as described in Ljunglöf 2004
-------------------------------------------------------------------------------}

module Examples where


-- imported GF modules
import MCFGrammar
import Parser


-- Following Non-Terminals are used: S, A ---------------------------------

data NT = S | A
          deriving( Eq, Ord, Show )


-- Example grammar 4.1 ----------------------------------------------------

ex41 = [ Rule S [ A ] [ Lin "s" [ Cat ( A, "p", 0 ),
                                  Cat ( A, "q", 0 ) ]] "f",
         Rule A [ A, A ] [ Lin "p" [ Cat ( A, "p", 0 ),
                                     Cat ( A, "p", 1 ) ],
                           Lin "q" [ Cat ( A, "q", 0 ),
                                     Cat ( A, "q", 1 ) ]] "g",
         Rule A [] [ Lin "p" [ Tok "a" ],
                     Lin "q" [ Tok "c" ]] "ac",
         Rule A [] [ Lin "p" [ Tok "b" ],
                     Lin "q" [ Tok "d" ]] "bd" ]
```

## A.2 Ranges

The module for all functions on ranges. Even those functions only used by one algorithm are placed in the Ranges module and not as helper functions in the algorithm's module. $\rho^{\epsilon}$ is written as `ERange`.

```
{-- Module --------------------------------------------------------------
  Filename   : Ranges.hs
  Author     : Håkan Burden
  Time-stamp : <2005-02-12, 18:52>
  Description: Functions for Ranges
------------------------------------------------------------------------}

module Ranges where



-- imported Haskell modules
import List
import Monad

-- imported GF modules
import MCFGrammar
import Nondet
import Parser



-- Declared new types: Linearization- and Range records as lists ----------

type LinRec c l t = [ Lin c l t ]

type RangeRec   l = [( l, Range )]



{-- Functions -----------------------------------------------------------
  Ceiling              : Returns the ceiling of a Range
  Concatenation        : Concatenation of Ranges, Symbols and
                         Linearizations and records of Linearizations
  Record transformation: Makes a Range record from a fully instantiated
                         Linearization record
  Record projection    : Given a label, returns the corresponding Range
  Range restriction    : Range restriction of Tokens, Symbols,
                         Linearizations and Records given a list of Tokens
  Record replacment    : Substitute a record for another in a list of Range
                         records
  Argument substitution: Substitution of a Cat cat to a Tok Range, where
                         Range is the cover of cat
                         Note: The argument is still a Symbol c Range
  Record Subsumation   : Checks if a Range record subsumes another Range
                         record
  Record unification   : Unification of two Range records
------------------------------------------------------------------------}

--- Ceiling ------------------------------------------------------------

ceil :: Range -> Range
ceil ERange           = ERange
ceil ( Range ( i, j ) = ( Range ( j, j )
```

```
--- Concatenation ------------------------------------------------------

concRanges :: Range -> Range -> Nondet Range
concRanges ERange ( Range ( i, j ))         =
    return ( Range ( i, j ))
concRanges ( Range ( i, j )) ( Range ( j', k )) =
    do guard ( j == j' )
       return (Range ( i, k ))


concSymbols :: [ Symbol c Range ] -> Nondet [ Symbol c Range ]
concSymbols ( Tok rng:Tok rng':toks ) = do rng'' <- concRanges rng rng'
                                           concSymbols ( Tok rng'':toks )
concSymbols ( sym:syms )              = do syms' <- concSymbols syms
                                           return ( sym:syms' )
concSymbols []                        = return []


concLin :: Lin c l Range -> Nondet ( Lin c l Range )
concLin ( Lin lbl syms ) = do syms' <- concSymbols syms
                              return ( Lin lbl syms' )


concLinRec :: LinRec c l Range -> Nondet ( LinRec c l Range )
concLinRec = mapM concLin


--- Record transformation ----------------------------------------------

makeRangeRec :: LinRec c l Range -> RangeRec l
makeRangeRec lins = map (\( Lin lbl [ Tok rng ]) -> ( lbl, rng )) lins


--- Record projection --------------------------------------------------

projection :: Eq l => l -> RangeRec l -> Nondet Range
projection l rec = maybe failure return $ lookup l rec


--- Range restriction --------------------------------------------------

rangeRestTok :: Eq t => [ t ] -> t -> Nondet Range
rangeRestTok toks tok = do i <- member ( elemIndices tok toks )
                           return ( makeRange ( i, i + 1 ))


rangeRestSym :: Eq t => [ t ] -> Symbol a t -> Nondet ( Symbol a Range )
rangeRestSym toks ( Tok tok ) = do rng <- rangeRestTok toks tok
                                   return ( Tok rng )
rangeRestSym _ ( Cat cat )    = return ( Cat cat )


rangeRestLin :: Eq t => [ t ] -> Lin c l t -> Nondet ( Lin c l Range )
rangeRestLin toks ( Lin lbl syms ) =
    do syms' <- mapM ( rangeRestSym toks ) syms
```

```
       return ( Lin lbl syms' )


rangeRestRec :: Eq t => [ t ] -> LinRec c l t
             -> Nondet ( LinRec c l Range )
rangeRestRec toks = mapM ( rangeRestLin toks )


-- Record replacment ----------------------------------------------------

replaceRec :: [ RangeRec l ] -> Int -> RangeRec l -> [ RangeRec l ]
replaceRec recs i rec = ( fst tup ) ++ [ rec ] ++ ( tail $ snd tup )
    where tup = splitAt i recs


--- Argument substitution ------------------------------------------------

substArgSymbol :: Eq l => Int -> RangeRec l -> Symbol ( c, l, Int ) Range
               -> Symbol ( c, l, Int ) Range
substArgSymbol i rec ( Tok rng ) = ( Tok rng )
substArgSymbol i rec ( Cat ( cat, lbl, j ))
    | i==j      = maybe ( Cat ( cat, lbl, j )) Tok $ lookup lbl rec
    | otherwise = (Cat ( cat, lbl, j ))


substArgLin :: Eq l => Int -> RangeRec l -> Lin c l Range
            -> Lin c l Range
substArgLin i rec ( Lin lbl syms ) =
    ( Lin lbl ( map (substArgSymbol i rec ) syms ))


substArgRec :: Eq l => Int -> RangeRec l -> LinRec c l Range
            -> LinRec c l Range
substArgRec i rec lins = map ( substArgLin i rec ) lins


--- Record Subsumation ---------------------------------------------------

subsumes :: Eq l => RangeRec l -> RangeRec l -> Bool
subsumes rec rec' = and [ elem r rec' | r <- rec ]


--- Record unification ---------------------------------------------------

unifyRangeRecs :: Ord l => [ RangeRec l ] -> [ RangeRec l ]
               -> Nondet [ RangeRec l ]
unifyRangeRecs recs recs' = zipWithM unify recs recs'
    where unify rec [] = return rec
          unify [] rec = return rec
          unify rec1'@( p1@( l1, r1 ):rec1 ) rec2'@( p2@( l2, r2 ):rec2 )
              = case compare l1 l2 of
                LT -> do rec3 <- unify rec1 rec2'
                         return ( p1:rec3 )
                GT -> do rec3 <- unify rec1' rec2
                         return ( p2:rec3 )
                EQ -> do guard ( r1 == r2 )
                         rec3 <- unify rec1 rec2
                         return ( p1:rec3 )
```

## A.3 NaiveParse

The active item 11 in the naïve parse chart in figure 3.2 on page 31 is written in code as

```
Active ("g",A,[A],[A])
       [Lin "p" [Tok (Range (1,2)),Cat (A,"p",1)],
        Lin "q" [Tok (Range (3,4)),Cat (A,"q",1)]]
       [("p",Range (1,2)),("q",Range (3,4))]
```

where the dot in the `DottedRule` is represented as two lists of categories, `[A],[A]`. The passive item 13 is in turn written as

```
Passive S
        [("s",Range (0,4))]
```

This is also the goal item for recognition.

```
{-- Module ------------------------------------------------------------
  Filename   : NaiveParse.hs
  Author     : Håkan Burden
  Time-stamp : <2005-02-24, 14:43>
  Description: An agenda-driven implementation of the algorithm 4.2.1,
               "Polynomial parsing for context-free GF",
               as described in Ljunglöf (2004)
-----------------------------------------------------------------------}

module NaiveParse where


-- imported GF modules

import ExampleGrammar
import GeneralChart
import MCFGrammar
import MCFParser
import Nondet
import Parser
import Ranges



{-- Datatypes and types -----------------------------------------------
  NChart    : A RedBlackMap with Items and NKeys
  Item      : The parse Items are either Active or Passive
  NKey      : One key for Active Items, one for Passive Items and one for
              Active Items converted to Passive Items
-----------------------------------------------------------------------}

type NChart   n c l = ParseChart ( Item n c l ) ( NKey c )

data Item     n c l = Active ( DottedRule n c )
                             ( LinRec c l Range )
                             ( RangeRec l )
                    | Passive c
                              ( RangeRec l )
```

```
                      deriving ( Eq, Ord, Show )

data NKey         c = Act c
                    | Pass c
                    | Final
                      deriving ( Eq, Ord, Show )


{-- Parsing -------------------------------------------------------------
  recognize: Returns 'True' if the goal Item is in the parse-chart
             otherwise 'False'
  parse    : Builds a chart from the initial agenda (given by prediction)
             and the inference rules
  keyof    : Given an Item returns an appropriate NKey for storing the
             Item in the Chart
------------------------------------------------------------------------}

recognize :: Grammar String NT String String -> [ String ] -> Bool
recognize mcfg toks =
    chartMember ( parse mcfg toks )
                  ( Passive S [( "s", Range ( 0, n ))]
                  ( Pass S )


parse :: ( Eq t, Ord n, Ord c, Ord l ) => Grammar n c l t -> [ t ]
      -> NChart n c l
parse mcfg toks = buildChart keyof [ convert, combine ]
                               ( predict mcfg toks )


keyof :: Item n c l -> NKey c
keyof ( Active ( _, _, _, ( next:_ )) lins _ ) = Act next
keyof ( Passive cat _ )                        = Pass cat
keyof _                                        = Final


{--Inference rules ---------------------------------------------------
  predict: Creates an Active Item of every Rule in the Grammar to give the
           initial agenda
  combine: Creates an Active Item every time it is possible to combine an
           Active Item from the agenda with a Passive Item from the Chart
  convert: Active Items with nothing to find are converted to Passive Items
------------------------------------------------------------------------}

predict :: ( Eq t, Eq c ) => [ t ] -> Grammar n c l t -> [ Item n c l ]
predict mcfg toks =
    [ Active ( f, cat, [], rhs ) lins' [] | Rule cat rhs lins f <- mcfg,
      lins' <- solutions $ rangeRestRec toks lins ]


combine :: ( Ord n, Ord c, Ord l ) => ParseChart ( Item n c l ) ( NKey c )
        -> Item n c l -> [ Item n c l ]
combine chart ( Active ( f, cat', found, (cat:toFind)) lins rec ) =
    [ Active ( f, cat', found ++ [ cat ], toFind ) lins'' ( rec ++ rec' ) |
      Passive cat rec' <- chartLookup chart ( Pass cat ),
      lins'' <- solutions $ concLinRec $ substArgRec ( length found )
      rec' lins ]
combine chart ( Passive cat rec ) =
    [ Active ( f, cat', found ++ [ cat ], toFind ) lins'' ( rec'++ rec ) |
      (Active ( f, cat', found, ( cat:toFind )) lins' rec')
      <- chartLookup chart ( Act cat ),
```

```
      lins'' <- solutions $ concLinRec $ substArgRec ( length found )
      rec lins' ]
combine _ _ = []


convert :: ( Ord n, Ord c, Ord l ) => ParseChart ( Item n c l ) ( NKey c )
        -> Item n c l -> [ Item n c l ]
convert _ ( Active ( f, cat, rhs, [] ) lins _ ) =
    [ Passive cat ( makeRangeRec lins ) ]
convert _ _ = []
```

## A.4 ApproxParse

There are four different kinds of items for the Approximative algorithm, pre-mcg, pre, mark and passive. The pre-mcfg item

$$[g; \{q = (2,\ 4); \{q = (2,\ 3),\ q = (3,\ 4)\}]$$

is written as

```
PreMCFG "g"
        [("q",Range (2,4))]
        [[("q",Range (2,3))],[("q",Range (3,4))]]
```

The pre item

$$[A \rightarrow bd[]; \{p = (1,\ 2)\}; \{q\}; \{\}]$$

looks as follows

```
Pre ("bd",A,[]) [("p",Range (1,2))] ["q"] []
```

A mark item uses DottedRules and has two lists for separating marked range records from unmarked ones

$$[A \rightarrow g[A \bullet A]; \{p = (0,\ 2), q = (2,\ 4)\}; \{p = (0,\ 1), q = (2,\ 3)\}; \{p = (1,\ 2), q = (3,\ 4)\}]$$

This item will look like

```
Mark ("g",A,[A],[A])
     [("p",Range (0,2)),("q",Range (2,4))]
     [("p",Range (0,1)),("q",Range (2,3))]
     [[("p",Range (1,2)),("q",Range (3,4))]]
```

when written in code. Finally we have the passive items, that look just like the passive items for the Naïve algorithm.

```
Passive S [("s",Range (0,4))]
```

is the goal item

$$[S; \{s = (0, 4)]$$

```
{-- Module ------------------------------------------------------------
  Filename   : ApproxParse.hs
  Author     : Håkan Burden
  Time-stamp : <2005-03-08 16:36:26>
  Description: An agenda-driven implementation of the algorithm 4.3.4,
               "Parsing through context-free approximation",
               as described in Ljunglöf (2004)
------------------------------------------------------------------------}

module ApproxParse where


-- imported Haskell modules
import List
import Monad

-- imported GF modules
import ConvertMCFGtoDecoratedCFG
import qualified DecoratedCFParser as CFP
import qualified DecoratedGrammar as CFG
import ExampleGrammar
import GeneralChart
import qualified MCFGrammar as MCFG
import MCFParser
import Nondet
import Parser
import Ranges


{-- Datatypes ---------------------------------------------------------
  AChart: A RedBlackMap of Items and AKeys
  Item  : Four different Items are used:
          * PreMCFG for MCFGPre-Items,
          * Pre-Items are the Items returned by the pre-Functions,
          * Mark-Items are the corresponding Items for the mark-Functions,
          * correctly marked Mark-Items are converted to Passive Items.
  AKey  : One AKey for every kind of Item and one for Items to be converted
------------------------------------------------------------------------}

data Item   n c l = PreMCFG n
                            ( RangeRec l )
                            [ RangeRec l ]
                  | Pre ( AbstractRule n c )
                        ( RangeRec l)
                        [ l ]
                        [ RangeRec l ]
                  | Mark ( DottedRule n c )
                         ( RangeRec l )
                         ( RangeRec l )
                         [ RangeRec l ]
                  | Passive c
```

```
                        ( RangeRec l )
                    deriving ( Eq, Ord, Show )

type AChart n c l = ParseChart ( Item n c l ) ( AKey n c l )

data AKey   n c l = Pm n l
                  | Pr n l
                  | Mk c ( RangeRec l )
                  | Ps c ( RangeRec l )
                  | Final
                    deriving ( Eq, Ord, Show )


{-- Parsing -------------------------------------------------------------
  recognize: Returns 'True' if the goal Item is the parse-chart,
             otherwise 'False'
  parse    : Builds a parse-chart from the agenda and the inference rules.
             The agenda consists of the Passive Items from context-free
             approximation (as PreMCFG-Items) and the Pre-Items inferred by
             pre-prediction. The Context-Free parsing is done by either
             bottom-up or top-down filtering
  keyof    : Given an Item returns an appropriate Key for storing the Item
             in the Chart
------------------------------------------------------------------------}

recognize :: ( Ord t ) => Strategy
          -> MCFG.Grammar String NT String t -> [t] -> Bool
recognize strategy mcfg toks =
    chartMember ( parse strategy mcfg toks ) ( Passive S rec ) ( Ps S rec )
        where rec = [( "s" , MCFG.Range ( 0, length toks ))]



parse :: ( Ord t ) => CFP.Strategy -> MCFG.Grammar String NT String t
      -> [ t ] -> AChart String NT String
parse strategy mcfg toks =
    buildChart keyof [ preCombine, markPredict, markCombine, convert ]
                  (( makePreItems ( CFP.parse strategy
                                    ( CFG.pInfo ( convertGrammar mcfg ))
                                    [( S, "s" )] toks )) ++
                   ( prePredict mcfg ))


keyof :: Item n c l -> AKey n c l
keyof ( PreMCFG f [( lbl, rng )] _ )              = Pm f lbl
keyof ( Pre ( f, _, _ ) _ ( lbl:_ ) _ )           = Pr f lbl
keyof ( Mark ( _, _, _, ( cat:_ )) _ _ ( rec:_ )) = Mk cat rec
keyof ( Passive cat rec )                         = Ps cat rec
keyof _                                           = Final


{-- Initializing agenda -------------------------------------------------
makePreItems: Every Passive Item from the Context-Free chart is made into a
              PreMCFG-Item
------------------------------------------------------------------------}

makePreItems :: ( Eq c, Ord i ) => CFG.Grammar n ( Edge ( c, l )) i t
             -> [ Item n c l ]
makePreItems cfchart =
```

```
    [ PreMCFG fun [( lbl, MCFG.makeRange ( i, j ))] ( symToRec beta ) |
      CFG.Rule ( Edge i j ( cat,lbl )) beta fun <- cfchart ]


{-- Inference rules -------------------------------------------------------
prePredict : Predicts a Pre-Item for every Rule in the MCF grammar
preCombine : Combines a Pre-Item looking for the lable l with a
             PreLCFG-Item for l into a new Pre-Item
markPredict: Predicts a Mark-Item for every Pre-Item with no lables left to
             look for
markCombine: Combines a Mark-Item looking for a record rec with a Passive
             Item for rec into a new Mark-Item
convert    : Converts a fully marked Mark-Item into a Passive Item
------------------------------------------------------------------------}

prePredict :: ( Ord n, Ord c, Ord l ) => MCFG.Grammar n c l t
           -> [ Item n c l ]
prePredict mcfg = [ Pre ( f, cat, rhs ) [] ( getLables lins )
                    ( replicate ( length rhs ) [] ) |
                    MCFG.Rule cat rhs lins f <- mcfg ]


preCombine :: ( Ord n, Ord c, Ord l )
           => ParseChart ( Item n c l ) ( AKey n c l ) -> Item n c l
           -> [ Item n c l ]
preCombine chart ( Pre head@( f, _, _ ) rec ( l:ls ) recs ) =
    [ Pre head ( rec ++ [( l, r )]) ls recs'' |
      PreMCFG f [( l, r )] recs' <- chartLookup chart ( Pm f l ),
      recs'' <- solutions ( unifyRangeRecs recs recs' ) ]
preCombine chart ( PreMCFG f [( l, r )] recs ) =
    [ Pre head ( rec ++ [( l, r )]) ls recs'' |
      Pre head rec ( l:ls ) recs' <- chartLookup chart ( Pr f l ),
      recs'' <- solutions ( unifyRangeRecs recs recs' ) ]
preCombine _ _ = []


markPredict :: ( Ord n, Ord c, Ord l )
            => ParseChart ( Item n c l ) ( AKey n c l ) -> Item n c l
            -> [Item n c l]
markPredict _ ( Pre ( f, cat, rhs ) rec [] recs ) =
    [ Mark ( f, cat, [], rhs ) rec [] recs ]
markPredict _ _ = []


markCombine :: ( Ord n, Ord c, Ord l )
            => ParseChart ( Item n c l ) ( AKey n c l ) -> Item n c l
            -> [ Item n c l ]
markCombine chart ( Mark ( f, cat', found, ( cat:toFind )) rec' marked
                    ( rec:toMark )) =
    [ Mark ( f, cat', found ++ [ cat ], toFind ) rec'
      ( marked ++ rec ) toMark |
      Passive cat rec <- chartLookup chart ( Ps cat rec ) ]
markCombine chart ( Passive cat rec ) =
    [ Mark ( f, cat', found ++ [ cat ], toFind ) rec' ( marked ++ rec )
      toMark |
      Mark ( f, cat', found, ( cat:toFind )) rec' marked ( rec:toMark )
      <- chartLookup chart ( Mk cat rec ) ]
markCombine _ _ = []
```

```
convert :: ( Ord n, Ord c, Ord l )
        => ParseChart ( Item n c l ) ( AKey n c l ) -> Item n c l
        -> [Item n c l]
convert _ ( Mark ( _, cat, _, [] ) rec rec' [] ) = [ Passive cat rec ]
convert _ _                                      = []


{-- Helper functions ------------------------------------------------------
getLables: Returns the list of lables in LinRec
symToRec : Gives a RangeRec from the lables and ranges in the Context-Free
           chart
--------------------------------------------------------------------------}

getLables :: LinRec c l t -> [ l ]
getLables lins = [ l | MCFG.Lin l syms <- lins ]


symToRec :: Ord i => [ Symbol ( Edge ( c, l ), i ) d ]
         -> [[( l, MCFG.Range )]]
symToRec beta =
    map makeLblRng $ groupBy ( \( _, d ) ( _, d' ) -> ( d == d' ))
                   $ sortBy sBd [ ( Edge i j ( c, l) , d ) |
                     Cat ( Edge i j ( c, l ), d ) <- beta ]
    where makeLblRng edges =
              [ ( l, ( MCFG.makeRange ( i, j ))) |
                  ( Edge i j ( _, l ), _ ) <- edges]
          sBd ( _, d ) ( _, d' )
              | d < d' = LT
              | d > d' = GT
              | otherwise = EQ
```

## A.5 ActiveParse

For the Earley function initial the rules with a start symbol as left-hand side is hard-coded. In the actual GF implementation it is substituted for the result from the function `pInfo`, which returns the parse-information of the grammar. The two rules for Earley prediction are combined into one rule in the code.

The active item 13 in figure 3.6 looks like

```
Active  ("g",A,[A,A])
        []
        (Range (1,2))
        (Lin "p" [Cat (A,"p",1)])
        [Lin "q" [Cat (A,"q",0),Cat (A,"q",1)]]
        [[("p",Range (1,2)),("q",Range (3,4))],[]]
```

written in code. When fully instantiated and converted to a passive item it looks like

```
Passive A
       [("p",Range (0,2)),("q",Range (2,4))]
```

The inference rules for Kilbury and Earley prediction are given in the end of the module.

```
{-- Module ------------------------------------------------------------
  Filename   : ActiveParse.hs
  Author     : Håkan Burden
  Time-stamp : <2005-03-24, 14:43>
  Description: An agenda-driven implementation of algorithm 4.6,
               "Active parsing of PMCFG",
               as described in Ljunglöf (2004)
----------------------------------------------------------------------}

module ActiveParse where


-- imported GF modules
import ExampleGrammar
import GeneralChart
import MCFGrammar
import MCFParser
import Nondet
import Parser
import Ranges


{-- Datatypes ---------------------------------------------------------
  AChart: A RedBlackMap with Items and AKeys
  Item  : Items are either Active or Passive
  AKey  : One key for every kind of Item and one for Active Items converted
          to Passive Items
----------------------------------------------------------------------}

data Item   n c l = Active ( AbstractRule n c )
                           ( RangeRec l )
                           Range
                           ( Lin c l Range )
                           ( LinRec c l Range )
                           [ RangeRec l ]
                  | Passive c
                            [ RangeRec l ]
                    deriving ( Eq, Ord, Show )

type AChart n c l = ParseChart ( Item n c l ) ( AKey c )

data AKey       c = Act c
                  | Pass c
                  | Final
                    deriving ( Eq, Ord, Show )


{-- Parsing -----------------------------------------------------------
recognize: If the goal Item is in the parse-chart: 'True',
           otherwise: 'False'
parse    : Builds a Chart from the initial agenda, given by prediction, and
           the inference rules. Parsing can be done with either Earley or
           Kilbury filtering, or without filtering
keyof    : Given an Item returns an appropriate Key for storing the Item in
           the Chart
----------------------------------------------------------------------}

recognize :: Strategy -> Grammar String NT String String -> [ String ]
          -> Bool
```

```
recognize strategy mcfg toks =
    chartMember ( parse strategy mcfg toks ) item ( keyof item )
    where item = Passive S [( "s", Range ( 0, n ))]
          n    = length toks


parse :: ( Ord n, Ord c, Ord l, Eq t ) => Strategy -> Grammar n c l t
      -> [ t ] -> ParseChart ( Item n c l ) ( AKey c )
parse (False, False) mcfg toks =
    buildChart keyof
               [ complete, scan, combine, convert ]
               ( predict mcfg toks )
parse (True, False) mcfg toks =
    buildChart keyof
               [ predictKilbury mcfg toks, complete, combine, convert ]
               ( terminal mcfg toks )
parse (False, True) mcfg toks =
    buildChart keyof
               [ predictEarley mcfg toks,
                 complete, scan, combine, convert ]
               ( initial ( take 1 mcfg ) toks )


keyof :: Item n c l -> AKey c
keyof ( Active _ _ _ ( Lin _ (( Cat ( next, _, _ )):_ )) _ _ ) = Act next
keyof ( Passive cat _ )                                        = Pass cat
keyof _                                                        = Final


{--Inference rules -------------------------------------------------------
predict : Creates an Active Item of every Rule in the Grammar to give the
          initial Agenda
complete: Predicts an Active Item for the next linearization row, if the
          previous row is fully satisfied.
scan    : If the next symbol to read is a range for a token, concatenate
          the range for what is found so far with the range for the token
combine : Creates an Active Item every time it is possible to combine
          an Active Item from the agenda with a Passive Item from the Chart
convert : Active Items with nothing to find are converted to Passive Items
-------------------------------------------------------------------------}

predict :: Eq t => Grammar n c l t -> [ t ] -> [ Item n c l ]
predict grammar toks = [ Active ( f, cat, rhs) [] ERange lin' lins'
                         ( replicate ( length rhs ) [] ) |
                         Rule cat rhs lins f <- grammar,
                         ( lin':lins' )
                         <- solutions $ rangeRestRec toks lins ]


complete :: ( Ord n, Ord c, Ord l ) => ParseChart ( Item n c l ) ( AKey c )
         -> Item n c l -> [ Item n c l ]
complete _ ( Active rule found ( Range ( i, j )) ( Lin l [] )
             ( lin:lins ) recs ) =
    [ Active rule ( found ++ [( l, Range ( i,j ))] ) ERange lin
      lins recs ]
complete _ _ = []


scan :: ( Ord n, Ord c, Ord l ) => ParseChart ( Item n c l ) ( AKey c )
     -> Item n c l -> [ Item n c l ]
```

```
scan _ ( Active rule found rng ( Lin l (( Tok rng' ):syms )) lins recs ) =
    [ Active rule found rng'' ( Lin l syms ) lins recs |
      rng'' <- solutions $ concRanges rng rng' ]
scan _ _ = []


combine :: ( Ord n, Ord c, Ord l ) => ParseChart ( Item n c l ) ( AKey c )
        -> Item n c l -> [ Item n c l ]
combine chart ( Active rule found rng ( Lin l (( Cat ( c, r, d )):syms ))
               lins recs ) =
    [ Active rule found rng'' ( Lin l syms ) lins
      ( replaceRec recs d found' ) |
      Passive _ found' <- chartLookup chart ( Pass c ),
      rng' <- solutions $ projection r found',
      rng'' <- solutions $ concRanges rng rng',
      subsumes ( recs !! d ) found' ]
combine chart ( Passive c found ) =
    [ Active rule found' rng ( Lin l syms ) lins
      ( replaceRec recs' d found ) |
      Active rule found' rng' ( Lin l (( Cat ( c, r, d )):syms ))
      lins recs'
      <- chartLookup chart ( Act c),
      rng'' <- solutions $ projection r found,
      rng <- solutions $ concRanges rng' rng'',
      subsumes ( recs' !! d ) found ]
combine _ _ = []


convert :: ( Ord n, Ord c, Ord l ) => ParseChart ( Item n c l ) ( AKey c )
        -> Item n c l -> [ Item n c l ]
convert _ ( Active ( f, cat, rhs ) found rng ( Lin l [] ) [] recs ) =
    [ Passive cat ( found ++ [( l, rng )] ) ]
convert _ _ = []


{-- Earley Filtering ------------------------------------------------------
initial      : Predict an Active Item for every rule in the grammar where
               the left-hand side of the rule is a start symbol
predictEarley: If there is an Active Item looking for a category and a rule
               where that category is the left-hand side of a rule, predict
               a new Item
--------------------------------------------------------------------------}

initial :: Eq t => [ Rule n c l t ] -> [ t ] -> [ Item n c l ]
initial starts toks =
    [ Active ( f, s, rhs ) [] ( Range ( 0, 0 )) lin' lins'
      ( replicate ( length rhs ) [] ) |
      Rule s rhs lins f <- starts,
      ( lin':lins' ) <- solutions $ rangeRestRec toks lins ]


predictEarley mcfg toks _ ( Active _ _ rng
                          ( Lin _ (( Cat ( cat, _, _ )):_ )) _ _ ) =
    concat [ earley rng rule | rule@( Rule cat' _ _ _ )
             <- mcfg, cat == cat' ]
        where earley _ ( Rule cat [] lins f ) =
                  [ Passive cat ( makeRangeRec lins' ) |
                    lins' <- solutions $ rangeRestRec toks lins ]
              earley rng ( Rule cat rhs lins f ) =
                  [ Active ( f, cat, rhs ) [] ( ceil rng ) lin' lins'
```

```
                  ( replicate ( length rhs ) [] ) |
                  ( lin':lins' ) <- solutions $ rangeRestRec toks lins ]
predictEarley _ _ _ _ = []


{-- Kilbury Filtering ----------------------------------------------------
predictKilbury: Predict an Active Item for a rule if there already is a
                Passive Item for the first category in the first
                linearization row
terminal      : Predict a Passive Item for every rule with empty right-hand
                side
-------------------------------------------------------------------------}

predictKilbury mcfg toks _ ( Passive ( _, cat, _ ) found _ ) =
    [ Active ( f, a, rhs ) [] rng lin' lins' daughters |
      Rule a rhs (( Lin l (( Cat ( cat', r, i )):syms )):lins ) f <- mcfg,
      cat == cat',
      lin' <- solutions $ rangeRestLin toks ( Lin l syms ),
      lins' <- solutions $ rangeRestRec toks lins,
      rng <- solutions $ projection r found,
      let daughters =
            ( replaceRec ( replicate ( length rhs ) [] ) i found ) ]
predictKilbury _ _ _ _ = []

terminal mcfg toks =
    [ Passive cat ( makeRangeRec lins' ) |
      Rule cat [] lins f <- mcfg,
      lins' <- solutions $ rangeRestRec toks lins ]
```

## A.6 IncrementalParse

The active item

$$[A \to g[A,\, A]; \{p = (0,\, 2)\}, q = (2,\, 4)\bullet; \Gamma_{\langle ac,bd\rangle}]$$

where $\Gamma_{\langle ac,bd\rangle}$ is taken from figure 3.2, will be written as

```
Active ("g",A,[A,A])
       [("p",Range (0,2))]
       (Range (2,4))
       (Lin "q" [])
       []
       [[("p",Range (0,1)),("q",Range (2,3))],
        [("p",Range (1,2)),("q",Range (3,4))]]
```

The layout follows the definition of an active item in the code.

```
{-- Module ---------------------------------------------------------------
  Filename   : IncrementalParse.hs
  Author     : Håkan Burden
  Time-stamp : <2005-04-29, 14:10>
  Description: An agenda-driven implementation of algorithm 4.6,
               "Incremental PMCFG parsing",
```

```
               as described in Ljunglöf (2004)
-----------------------------------------------------------------------}

module IncrementalParse where


-- imported Haskell modules
import List

-- imported GF modules
import ExampleGrammar
import GeneralChart
import MCFGrammar
import MCFParser
import Parser
import Ranges
import Nondet


{-- Datatypes ------------------------------------------------------------
  IChart: A RedBlackMap with Items and IKeys
  Item  : One kind of Item since the Passive Items not necessarily need to
          be fully saturated, they can still have rows to recognize.
  IKey  : Three kind s of IKeys; one for Items investigating an unsaturated
          row, one for Items who have saturated an entire row and one for
          fully saturated Items
-----------------------------------------------------------------------}

type IChart n c l = ParseChart ( Item n c l ) ( IKey c l )

data Item   n c l = Active ( AbstractRule n c )
                           ( RangeRec l )
                           Range
                           ( Lin c l Range )
                           ( LinRec c l Range )
                           [ RangeRec l ]
                    deriving ( Eq, Ord, Show )

data IKey     c l = Act c l Int
                  | Pass c l Int
                  | Final
                    deriving ( Eq, Ord, Show )


{-- Parsing --------------------------------------------------------------
recognize: Reurns 'True' if the goal Item is in the Chart,
           otherwise 'False'
parse    : Builds a Chart from the initial agenda, given by prediction, and
           the inference rules
keyof    : Given an Item returns an appropriate IKey for storing the Item
           in the Chart
-----------------------------------------------------------------------}

recognize mcfg toks = chartMember ( parse mcfg toks ) item ( keyof item )
    where item = Active ( "f", S, [ A ])
                 [] ( Range ( 0, n )) ( Lin "s" [] ) []
                 [[( "p", Range ( 0, n2 )), ( "q", Range ( n2, n ))]]
          n = length toks
          n2 = n `div` 2
```

```
parse :: ( Ord n, Ord c, Ord l, Eq t ) => Grammar n c l t -> [ t ]
      -> IChart n c l
parse mcfg toks = buildChart keyof
                  [ complete toks n, scan, combine ]
                  ( predict mcfg toks n )
    where n = length toks


keyof :: Item n c l -> IKey c l
keyof ( Active _ _ ( Range ( _, j ))
        ( Lin _ (( Cat ( next, lbl, _ )):_ )) _ _ )
    = Act next lbl j
keyof ( Active ( _, cat, _ ) found ( Range ( i, _ )) ( Lin lbl [] ) _ _ )
    = Pass cat lbl i
keyof _
    = Final


{-- Inference Rules -------------------------------------------------------
predict : Predicts an Item for every linearization row in every rule in the
          grammar
complete: Predicts a new item for every remaining linearization row, when
          the previous row is fully saturated
scan    : Range concatenates the range for what is found so far with the
          range of the next symbol, if it is a linearized token
combine : Combines an Active Item looking for the category cat with a
          Passive Item for cat
--------------------------------------------------------------------------}

predict :: ( Eq n, Eq c, Eq l, Eq t ) => Grammar n c l t -> [ t ] -> Int
        -> [ Item n c l ]
predict mcfg toks n =
    [ Active ( f, c, rhs ) [] ( Range ( k, k )) lin' lins''
      ( replicate ( length rhs ) [] ) |
      Rule c rhs lins f <- mcfg,
      lins' <- solutions $ rangeRestRec toks lins,
      ( lin', lins'' ) <- select lins',
      k <- [ 0..n ] ]


complete :: (Ord n, Ord c, Ord l) => [ t ] -> Int
         -> ParseChart (Item n c l) ( IKey c l ) -> Item n c l
         -> [Item n c l ]
complete toks n _ ( Active rule found rng@( Range ( _, j )) ( Lin l [] )
                    lins recs ) =
    [ Active rule ( found ++ [( l, rng )]) ( Range ( k, k ))
      lin lins' recs | ( lin, lins' ) <- select lins,
      k <- [ j..n ] ]
complete _ _ _ _ = []


scan :: ( Ord n, Ord c, Ord l ) => ParseChart ( Item n c l ) ( IKey c l )
     -> Item n c l -> [ Item n c l ]
scan _ ( Active rule found rng ( Lin l (( Tok rng' ):syms )) lins recs ) =
    [ Active rule found rng'' ( Lin l syms ) lins recs |
      rng'' <- solutions $ concRanges rng rng' ]
scan _ _ = []
```

```
combine :: ( Ord n, Ord c, Ord l )
        => ParseChart ( Item n c l ) ( IKey c l ) -> Item n c l
        -> [ Item n c l ]
combine chart ( Active rule found rng@( Range ( _, j ))
                ( Lin l (( Cat ( c, r, d )):syms )) lins recs ) =
    [ Active rule found rng'' ( Lin l syms ) lins
      ( replaceRec recs d ( found' ++ [( l', rng' )]) ) |
      Active _ found' rng' ( Lin l' [] ) _ _
      <- chartLookup chart ( Pass c r j ),
      subsumes ( recs !! d ) ( found' ++ [( l', rng' )]),
      rng'' <- solutions $ concRanges rng rng' ]
combine chart ( Active ( _, c, _ ) found rng'@( Range ( i, _ ))
                ( Lin l [] ) _ _ ) =
    [ Active rule found' rng'' ( Lin l' syms ) lins
      ( replaceRec recs d ( found ++ [( l, rng' )])) |
      Active rule found' rng ( Lin l' (( Cat ( c, r, d )):syms ))
      lins recs <- chartLookup chart ( Act c l i ),
      subsumes ( recs !! d ) ( found ++ [( l,rng' )]),
      rng'' <- solutions $ concRanges rng rng' ]
combine _ _ = []
```