UPPSALA
UNIVERSITET

IT 15 004
Examensarbete 15 hp
Januari 2015

A New SAT Encoding of Earley Parsing

Tobias Neil

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

A New SAT Encoding of Earley Parsing

Tobias Neil

While the Boolean satisfiability problem (SAT) lies in NP, prodigious work in SAT solvers has allowed for its use in modeling a multitude of practical problems. Stating a problem in SAT can be cumbersome though and so the demand for SAT encodings arises, providing a means to formulate problems or parts of problems in a more intuitive environment. Several algorithms have been proposed in the past to encode context-free grammars as SAT formulae, allowing for the comprehensive construction of many interesting constraints such as at-most k constraints or such ones pertaining to language syntax. In 2011 a new algorithm was proposed, differing from previous ones in it being based on Earley parsing instead of CYK parsing. Although it performed well for interesting groups of grammars it was later found to behave incorrectly for certain inputs. This thesis discusses the flaws in said algorithm, presents a revision of it and argues for the altered algorithm's correctness. The alterations come with a price, however, and both the running time and output size complexities suffer more-than-quadratic blowup. Since no empirical tests have been performed as of yet, it is still unclear what impact this blowup will have on practical instances.

Handledare: Tjark Weber
Ämnesgranskare: Lars-Henrik Eriksson
Examinator: Olle Gällmo
IT 15 004
Tryckt av: Reprocentralen ITC

Contents

1 Introduction

1.1 The SAT Problem

The Boolean satisfiability problem or SAT is the decision problem whether or not there is any assignment to the Boolean variables of a formula ϕ such that ϕ evaluates to true. It is a much studied one: In 1971 it was the first problem to be proven to be *NP-complete* [1], and as such many problems within NP have been formulated as SAT. Furthermore, dedicated work spanning the last half-century has resulted in some powerful tools to subjugate the theoretically hard problem (if $P \neq NP$ there is no polynomial-time algorithm to solve SAT) into one that can be practically solved even for large inputs.

Context-free grammars provide an intuitive way of expressing many useful patterns, e.g. syntax for both natural and programming languages, palindromes or at-most k structures (does the string $w \in \{a, b\}^*$ have at most k occurrences of the symbol a?). While *parsing* a given word, that is deciding whether or not it can be generated by a grammar G, can be done in polynomial time for context-free grammars there are reasons why we would want to encode this decision as a SAT input: It can be used to describe constraint problems, either on its own or as a part of a larger problem, modeling a subset of the desired constraints (such as the above mentioned at most k).

1.2 Previous Encoding

In 2011 Giannaros proposed an encoding of context-free grammars into the SAT problem [6]. The encoding produced good performance results, especially for practically occurring grammars such as those describing program language syntaxes. However, the encoding algorithm is not altogether sound, and will produce faulty output for some grammars. These flaws are presented in detail in Section 3.4. There are other encoding algorithms as well, most notably Axelsson et al from 2008 [7] and Quimper and Walsh's from 2007 [8]. What separates these from the algorithm laid out by Giannaros is that they mimic the methods of the CYK parsing algorithm whereas Giannaros' uses the parsing techniques introduced by J. Earley in his parsing algorithm from 1970 [3].

1.3 Contribution

The purpose of this paper is thus to reform Giannaros' algorithm so as to have it produce only correct output. In Section 4 I propose an altered version of the encoding that overcomes the problems described in 3.4. A proof for the correctness of the algorithm is given, but is coarsely written with several unproven assump-

tions. The algorithm is presented in a somewhat naive way but some optimization is hinted at; as of now the size of the output is very large, and while no empirical studies have been made it is highly unlikely that the promising results of Giannaros' faulty encoding carry over to this altered version.

2 Background

The reader is assumed to be familiar with basic complexity notation ($\Theta(n), O(n)$) and with Boolean algebra. If not, there are numerous textbooks covering these topics in various levels of depth. For the purposes of this paper a introductory level of understanding is sufficient. I recommend the standard work *Introduction to Algorithms* [5] for an introduction to complexity analysis and *Language, Proof and Logic* [4] for Boolean algebra.

2.1 Grammars

Grammars provide a powerful way to express structure in data and even computation itself. They can be used to find substrings in texts without knowing exactly what is being sought, being able to match not only on content but also on form. Some grammars are used to check whether a program is syntactically correct before attempting compilation, some are powerful enough to describe all the workings of a computer's processor. They come in different classes, building a hierarchy of expressive potential where a more powerful class of grammars can describe everything a less powerful class can, but not vice versa. This paper revolves around the context-free grammars, popular uses of which include modeling natural language grammars, syntax checks for compilers and structuring research data in biotechnology.

2.1.1 Languages

Definition 1. *(Strings) A string over an alphabet Σ of symbols is a sequence $w = w_1 w_2 ... w_k$ s.t. $\forall i : 1 \leq i \leq k : w_i \in \Sigma$.*

Definition 2. *(Languages) A set of strings is called a language.*

Much like sets over numbers, languages can be defined ostensively, i.e. by naming every member of the set, or by specifying a property that is common to the members but not to any non-members. In the case of the set of even numbers between 0 and 9, this could either be described by $\{0, 2, 4, 6, 8\}$ or by $\{n \in \mathbb{N} \mid 0 \equiv n \mod 2, 0 \leq n \leq 9\}$. When specifying languages, this discerning property is often a *grammar*.

Definition 3. *(Grammars) A grammar is a quadruple* (N, Σ, P, S).

- N *is a finite set on* non-terminal symbols
- Σ *is a finite set of* terminal symbols
- P *is a finite set of* production rules *on the form* $(\Sigma \cup N)^* N (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$
- $S \in N$ *is a unique* root symbol.

From here on, terminal symbols will be denoted by lowercase roman letters $(a, b, ...)$, non-terminals by uppercase roman letters $(A, B, ...)$ and sequences $(\Sigma \cup N)^*$ by lowercase Greek letters $(\alpha, \beta, ...)$. We say that β can be derived in one step from α, $\alpha \Rightarrow \beta$, *iff* there is a production rule $A \rightarrow \gamma \in P$ such that $\alpha = \alpha' A \alpha''$ and $\beta = \alpha' \gamma \alpha''$. A finite series of one-step derivations, $\alpha \Rightarrow \alpha_1 \Rightarrow ... \Rightarrow \alpha_k \Rightarrow \beta$ can be written as

$$\alpha \Rightarrow^* \beta$$

We can then use the grammar G to define the set L over an alphabet Σ as $L = \mathbb{L}(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$, i.e. the set of strings over Σ that can be derived from the start symbol S according to the rules in grammar G. We say that G *generates* the language $\mathbb{L}(G)$.

2.1.2 Context-Free Grammars

One class of grammars (and languages generated by them) is the *context-free grammars*. They have the property that all production rules in P are of the form

$$N \rightarrow (N \cup \Sigma)^*$$

A subset of context-free grammars are those written in *Chomsky normal form*, named after the linguist and philosopher Noam Chomsky. The Chomsky normal form grammars separate non-terminal right-hand sides from terminal right-hand sides. Each of their production rules follows one of these three schemes:

$$N \rightarrow N \times N$$

$$N \rightarrow \Sigma$$

$$S \rightarrow \varepsilon$$

The last production rule scheme is necessary in order to allow for the empty string to be a member of the generated language. Every CFG can be written in Chomsky normal form, although this might cause a quadratic blowup of the size of the grammar [10].

Backus-Naur Form Grammars (especially context-free grammars) are commonly written in the Backus-Naur Form (BNF). For a grammar where $S \Rightarrow^* Aba, A \Rightarrow^* Aa, A \Rightarrow^* \varepsilon$ we write:

$$S \to Aba$$

$$A \to Aa \mid \varepsilon$$

This grammar G would generate the language $\mathbb{L}(G) = \{ba, aba, aaba, ...\}$.

2.1.3 The Size of a Grammar

When discussing algorithms operating on grammars it is useful to relate the performance of said algorithms to some metric on the complexity of the grammars - intuitively, the CFG describing the syntax of the English language ought to be "bigger" than the example grammar in the previous section. One might be tempted to define such a metric as a sum or product of the sizes of the three components N, Σ, P. However, the alphabet of terminal symbols, or the set of non-terminals, could be arbitrary large and still not affect the potential member strings of the language. As a case in point, the alphabet employed in the example grammar $\{ba, aba, aaba, ...\}$ may very well include the symbols $c_1, ..., c_{100}$. A much more reasonable measure of grammar size would then be limited to the length of the production rules in P.

Definition 4. *(Size of a Grammar) The size of a grammar G is defined as the sum of the length of its production rules:*

$$|G| = \sum_{(A \to \alpha) \in P} |A\alpha|$$

2.2 Parsing

The objective of a *parser* is two-fold: Firstly, given a string w and a grammar G, to answer whether w is a member of the language generated by G, i.e. $w \in \mathbb{L}(G)$. Secondly - if G indeed recognizes w - to construct a *parse tree* of the string. Such a tree shows which production rules are employed in order to derive w from S, and in what order. Figure 2.2 shows a parse tree for a parenthesis matching grammar $S \to SS \mid (S) \mid \varepsilon$ and the string ()(()).

2.2.1 Earley

Earley's algorithm from 1970 [3] is a parsing algorithm for context-free grammars, taking as its inputs a grammar $G = (N, \Sigma, P, S)$ and a string w and in its most

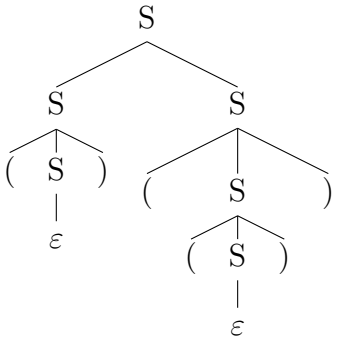

Figure 1: A parse tree

basic form decides whether or not $w \in \mathbb{L}(G)$. The algorithm scans the input string from left to right, constructing intermediate parsing states as it goes. These states are derived from the production rules of the grammar, marking how much of the rule's right-hand side has already been successfully matched against w. Such *Earley items* are of the form $(A \to \alpha \cdot \beta, j)$, where the dot denotes how much of the right-hand side has been matched thus far. The items are contained in sets X_0 to $X_{|w|}$, where $(A \to \alpha \cdot \beta, j) \in X_i$ means that $(A \to \alpha\beta) \in P$ and that $\alpha \Rightarrow^* w_{j+1}...w_i$. The integer j points back to the Earley item set where the current matching began, e.g. in the item $(A \to \alpha \cdot \beta, j) \in X_i$, it points to the the set X_j of which the yet-to-be-matched item $(A \to \cdot\alpha\beta, j)$ is a member.

For simplicity, and without loss of generality, a new start symbol S', along with the production rule $S' \to S$, is added to the grammar. Thus we can conclude that if $(S' \to S\cdot, 0) \in X_{|w|}$ then $S \Rightarrow^* w_1...w_{|w|} = w$ i.e. w is recognized by the grammar.

The sets X_1 to $X_{|w|}$ are initialized as empty, and X_0 as containing only the Earley item $(S' \to \cdot S, 0)$. Then, for each $i : 0 \leq i \leq |w|$, X_i is populated by applying these three rules repeatedly, until no more items can be added:

- *Predictor*: $\forall(A \to \alpha \cdot B\beta, j) \in X_i : \forall B \to \gamma \in P$ add $(B \to \cdot\gamma, i)$ to X_i
- *Scanner* (if $i < |w|$): $\forall(A \to \alpha \cdot w_{i+1}\beta, j) \in X_i$ add $(A \to \alpha w_{i+1} \cdot \beta, j)$ to X_{i+1}.
- *Completer*: $\forall(A \to \gamma\cdot, j) \in X_i : \forall(B \to \alpha \cdot A\beta', k) \in X_j$ add $(B \to \alpha A \cdot \beta, k)$ to X_i.

The original paper suggests the use of look ahead strings. However, this paper uses a revised algorithm proposed by Aycock and Horspool [2] where look ahead strings are not employed.

2.2.2 Ambiguity

Definition 5. *(Ambiguous grammars) A grammar that, for some string in its generated language, can yield more than one parse tree is called ambiguous.*

Definition 6. *(Infinitely ambiguous grammars) A grammar that, for some string in its generated language, can yield an infinite number of parse trees is called infinitely ambiguous.*

The parenthesis matching in Figure 2.2 is an example of an infinitely ambiguous grammar, since any occurrence of S can be used to derive $S \Rightarrow SS \Rightarrow \varepsilon S = S$ an arbitrary amount of times, thus creating arbitrary large parse trees for any string in its language.

2.3 Propositional Formulae, Satisfiability

A propositional formula consists of Boolean variables $z \in Z$ and constants ($\top$ (true) and $\bot$ (false)) put together with *Boolean connectives*, such as AND ($\wedge$) and OR ($\vee$). Depending on what values are being assigned to the variables in a formula it can evaluate to either $\top$ or $\bot$.

Definition 7. *(Propositional Formulae) We define a propositional formula inductively:*

- $\bot$ *is a propositional formula*
- $\top$ *is a propositional formula*
- *Any Boolean variable* $z \in Z$ *is a propositional formula*
- *If* A *is a propositional formula, then* $\neg A$ *is a propositional formula.*
- *If* A *and* B *are propositional formulae, then* $A \wedge B$ *is a propositional formula.*
- *If* A *and* B *are propositional formulae, then* $A \vee B$ *is a propositional formula.*
- *If* A *and* B *are propositional formulae, then* $A \Rightarrow B$ *is a propositional formula.*
- *If* A *and* B *are propositional formulae, then* $A \Leftrightarrow B$ *is a propositional formula.*

Each Boolean variable can be interpreted as either $\top$ or $\bot$. We can for example interpret the variables A and B as $A = \top$ and $B = \bot$, or $A = B = \bot$. Given a set Z' of Boolean variables there are $2^{|Z'|}$ number of *assignments*, specifying the Boolean value for each variable.

Definition 8. *(Boolean Assignment) An assignment σ is a mapping $Z \mapsto \{\top, \bot\}$.*

The *evaluation* of a formula for some assignment σ is the value of the formula, as dictated by the rules of the logical connectives, when all the variables are replaced with the constants they correspond to under σ.

Definition 9. *(Evaluation) The evaluation function $eval_\sigma$ maps, given an assignment σ, a propositional formula ϕ to either $\top$ or $\bot$. It is defined inductively:*

- $eval_\sigma(\bot) = \bot$
- $eval_\sigma(\top) = \top$
- $eval_\sigma(z) = \sigma(z)$
- $eval_\sigma(\neg A) = \top$ *iff* $eval_\sigma(A) = \bot$
- $eval_\sigma(A \wedge B) = \top$ *iff* $eval_\sigma(A) = \top$ *and* $eval_\sigma(B) = \top$
- $eval_\sigma(A \vee B) = \top$ *iff* $eval_\sigma(A) = \top$ *or* $eval_\sigma(B) = \top$
- $eval_\sigma(A \Rightarrow B) = \top$ *iff* $eval_\sigma(A) = \bot$ *or* $eval_\sigma(B) = \top$
- $eval_\sigma(A \Leftrightarrow B) = \top$ *iff* $eval_\sigma(A) = eval_\sigma(B)$

Definition 10. *(Satisfiable) A propositional formula ϕ is satisfiable iff there is at least one Boolean assignment σ s.t. $eval_\sigma(\phi) = \top$.*

SAT is the decision problem whether or not a formula is satisfiable. It was the first problem to be proven NP-complete. As such, the number of clauses as well as the number of variables outputted by the encoding algorithm is of practical interest.

AMO - ALO In the coming sections we will have a need for a propositional logic encoding of constraints such as "Exactly one of these variables must be assigned to $\top$". For this purpose we introduce the following two concepts:

- When *at most one (AMO)* of $n \geq 1$ variables can be true.
- When *at least one (ALO)* of $n \geq 1$ variables must be true.

When combined, these constraints can be used to model the above mentioned "Exactly one..."-criterion. ALO of a set $\{z_1, ..., z_n\} \subset Z$ is trivially implemented as a disjunction of the variables:

$$ALO(\{z_1, ..., z_n\}) = \bigvee_{1 \leq i \leq n} z_i \tag{1}$$

AMO is a bit more complex, but can for example be represented by a *binomial encoding*:

$$AMO(\{z_1, ..., z_n\}) = \bigwedge_{1 \leq i \leq n-1} \bigwedge_{i+1 \leq j \leq n} \neg z_i \vee \neg z_j \tag{2}$$

This concludes the background theory required of the reader in order to follow the upcoming discussions and proofs.

3 Old Encoding

3.1 Informal Description

In [6] the Earley algorithm is used to generate logical formulae reflecting the membership of strings for a given grammar. Instead of looking at any one specific string, the algorithm produces a formula that is satisfiable for any string of a certain length in the grammar's language. The algorithm first creates an extended version of the Earley sets, in the paper called *Earley spine sets*. The reason for this is that Giannaros' algorithm has no particular string in mind when performing the encoding, and so the scanner rule must be applicable on a larger set of Earley items. When the spine sets are fully constructed the algorithm explicitly lays out the dependencies among the Earley items, stating for each item under what conditions it would be constructed in the original algorithm.

3.2 Encoding

Given a CFG $G = (N, \Sigma, S, \rightarrow)$, where N is the set of non-terminals, Σ the alphabet, $S \in N$ the start symbol and $\rightarrow$ is the set of production rules, and a word length n, the encoding aims to output a Boolean formula $[\![G]\!]_n$, such that

$$\exists \sigma : eval_\sigma([\![G]\!]_n) = \top \Leftrightarrow \exists w \in \Sigma^* :: |w| = n \wedge w \in \mathbb{L}(G)$$

i.e. $[\![G]\!]_n$ is satisfiable iff there is some string in $\mathbb{L}(G)$ of length n.

We first produce the Earley spine sets Y_0 to Y_n by running the Earley algorithm described in 2.2.1, but with the scanner rule modified so as to accept any terminal symbol. We introduce two types of variables - *letter variables* and *Earley item variables*. For all $i : 1 \leq i \leq n$ and $a \in \Sigma$, the letter variable $w_i(a)$ is true iff $w_i = a$. The Earley item variable $X_i(A \rightarrow \alpha \cdot \beta, j)$ has a slightly more complicated semantics. It will not be described in detail here, but can be thought of as having the interpretation that X_i contains the Earley item $(A \rightarrow \alpha \cdot \beta, j)$. This notion is however not true in all cases and should rather be used as an informal description.

$[\![G]\!]_n$ is initialized as the formula $X_n(S' \to S\cdot, 0)$ and is then incrementally expanded by applying the following rules to every Earley item in every Earley spine set.

For all $i : 0 \leq i \leq n$ and $(A \to \alpha \cdot \beta, j) \in Y_i$:

- If $\alpha = \alpha' a$, add as a conjunct to $[\![G]\!]_n$:

$$X_i(A \to \alpha' a \cdot \beta, j) \Leftrightarrow X_{i-1}(A \to \alpha' \cdot a\beta, j) \wedge w_i(a) \tag{3}$$

- If $\alpha = \varepsilon$ and $A \neq S'$, add as a conjunct to $[\![G]\!]_n$:

$$X_i(A \to \cdot\beta, j) \Leftrightarrow \bigvee_{(B \to \alpha' \cdot A\beta', k) \in Y_i} X_i(B \to \alpha' \cdot A\beta, k) \tag{4}$$

- If $\alpha = \alpha' B$, add as a conjunct to $[\![G]\!]_n$:

$$X_i(A \to \alpha' B \cdot \beta, j) \Leftrightarrow \bigvee_{\substack{(B \to \gamma\cdot, k) \in Y_i \\ s.t.\ (B \to \gamma, k) \neq (A \to \alpha' B\beta, j)}} X_i(B \to \gamma\cdot, k) \wedge X_k(A \to \alpha' \cdot B\beta, j) \tag{5}$$

- If $X_i(A \to \alpha \cdot \beta, j) = X_0(S' \to \cdot S, 0)$, add as a conjunct to $[\![G]\!]_n$:

$$X_0(S' \to \cdot S, 0) \tag{6}$$

- We also need to ensure that every letter variable w_i maps to exactly one symbol, i.e. that for any index i, there is exactly one symbol a such that $w_i(a) \Leftrightarrow \top$

$$\bigwedge_{1 \leq i \leq n} AMO\{w_i(s) \mid s \in \Sigma\} \wedge ALO\{w_i(s) \mid s \in \Sigma\} \tag{7}$$

3.3 Example

In [6] the following example run on the infinitely ambiguous grammar

$$S \to SS \mid a \mid \varepsilon$$

is done to illustrate the point of the exclusion condition $(B \to \gamma, k) \neq (A \to \alpha' B\beta, j)$ in Rule 5. We will parse the string $w = aa$. The spine sets Y_0, Y_1, Y_2 are:

Y_0	Y_1	Y_2
$(S \to \cdot, 0)$	$(S \to \cdot, 1)$	$(S \to \cdot, 2)$
$(S \to \cdot SS, 0)$	$(S \to \cdot SS, 1)$	$(S \to \cdot SS, 2)$
$(S \to S \cdot S, 0)$	$(S \to S \cdot S, 0)$	$(S \to S \cdot S, 0)$
$(S \to SS\cdot, 0)$	$(S \to S \cdot S, 1)$	$(S \to S \cdot S, 1)$
$(S \to \cdot a, 0)$	$(S \to SS\cdot, 0)$	$(S \to S \cdot S, 2)$
$(S' \to \cdot S, 0)$	$(S \to SS\cdot, 1)$	$(S \to SS\cdot, 0)$
$(S' \to S\cdot, 0)$	$(S \to \cdot a, 1)$	$(S \to SS\cdot, 2)$
	$(S \to a\cdot, 1)$	$(S \to SS\cdot, 1)$
	$(S \to a\cdot, 0)$	$(S \to SS\cdot, 2)$
	$(S' \to S\cdot, 0)$	$(S \to \cdot a, 2)$
		$(S \to a\cdot, 1)$
		$(S' \to S\cdot, 0)$

Figure 2: Spine sets Y_0 to Y_2

When we come to the Earley item $(S \to S \cdot S, 0) \in Y_2$ we have to apply Rule 5. Since its pointer is 0 we need to find every item $(B \to \gamma\cdot, k) \in Y_i$ such that there is an item $(S \to \cdot SS, 0) \in Y_k$. The only such item is $(S \to SS\cdot, 0)$. However, if we look at the encoding of $(S \to SS\cdot, 0)$ we see that this item in turn could depend on $(S \to S \cdot S, 0)$. We thus get the following implications:

$$X_2(S \to S \cdot S, 0) \Rightarrow X_2(S \to SS\cdot, 0) \wedge X_0(S \to \cdot SS, 0)$$

$$\wedge X_2(S \to SS\cdot, 0) \Rightarrow (X_2(S \to \cdot, 2) \wedge X_2(S \to S \cdot S, 0)) \vee \mathbb{P}^1$$

$$\{Transitivity\} \Rightarrow$$

$$X_2((S \to S \cdot S, 0) \Rightarrow (X_2(S \to \cdot, 2) \wedge X_2(S \to S \cdot S, 0)) \vee \mathbb{P}) \wedge X_0(S \to \cdot SS, 0)$$

$$\{\textit{Distribution of } \wedge\} \Rightarrow$$

$$X_2(S \to S\cdot S, 0) \Rightarrow (X_2(S \to \cdot, 2) \wedge X_2(S \to S\cdot S, 0) \wedge X_0(S \to \cdot SS, 0)) \vee (\mathbb{P} \wedge X_0(S \to \cdot SS, 0))$$

$$\{\textit{Or-simplification}\} \Rightarrow$$

$$X_2(S \to S \cdot S, 0) \Rightarrow (X_0(S \to \cdot SS, 0) \wedge X_2(S \to \cdot, 2)) \vee \mathbb{Q}^2$$

This resulting clause is intuitively problematic, since the truth of the statement $(S \to S \cdot S, 0) \in X_2$, i.e. "The right-hand sequence S can be expanded into the symbol sequence $w_1 w_2$", is not related to any letter variable in that range, nor

[1] $\mathbb{P}$ is the remainder of the RHS disjunction, after $X_2(S \to \cdot, 2) \wedge X_2(S \to S \cdot S, 0)$ has been factored out.

[2] $\mathbb{Q} \Leftrightarrow (\mathbb{P} \wedge X_0(S \to \cdot SS, 0))$

any Earley item variable implying any statement about that range, but can be satisfied by the truth of the variables $X_0(S \to \cdot SS, 0)$ and $X_2(S \to \cdot, 2)$ alone. We must therefore exclude self reference of this kind, although we will soon show that the exclusion condition of Rule 5 will not be enough.

3.4 Flaws

3.4.1 Semantics

In the previous encoding the Earley item variable $X_i(A \to \alpha \cdot \beta, j)$ had the interpretation $(A \to \alpha \cdot \beta, j) \in X_i$, where X_i is the i:th Earley set after a completed run of an Earley parse. Looking at the item variable $X_2(S \to S \cdot S, 0)$ in Section 3.3 we can see that this interpretation is faulty: by applying the completer rule on items $(S \to \cdot SS, 0) \in X_0$ and $(S \to SS\cdot, 0) \in X_2$ we add the item $(S \to S \cdot S, 0)$ to X_2. Yet, the encoding sets the variable $X_2(S \to S \cdot S, 0)$ to false. Clearly something is amiss in the semantics of the variables.

3.4.2 Indirect Recursions

The encoding rule reflecting the completer rule prevents the dependence of immediate recursions, as in the example of $X_2(S \to S \cdot S, 0)$, but does not prevent any *indirect* recursions. As an example, the production rules used in Section 3.3 can be extended with the trivial rule $S \to S$. We can now circumvent the recursion prevention by the derivation

$$X_2(S \to S \cdot S, 0) \Rightarrow X_2(S \to S\cdot, 0) \wedge X_0(S \to \cdot SS, 0)$$

$$\wedge X_2(S \to S\cdot, 0) \Rightarrow X_2(S \to SS\cdot, 0) \wedge X_0(S \to \cdot S, 0) \wedge \mathbb{P}^3$$

$$\wedge X_2(S \to SS\cdot, 0) \Rightarrow X_2(S \to \cdot, 2) \wedge X_2(S \to S \cdot S, 0) \wedge \mathbb{Q}$$

$$\{\textit{Several steps, not shown for the sake of brevity}\} \Rightarrow$$

$$X_2(S \to S \cdot S, 0) \Rightarrow (X_0(S \to \cdot SS, 0) \wedge X_0(S \to \cdot S, 0) \wedge X_2(S \to \cdot, 2)) \vee$$

$$(X_0(S \to \cdot SS, 0) \wedge X_0(S \to \cdot S, 0) \wedge \mathbb{Q}) \vee$$

$$X_0(S \to \cdot SS, 0) \wedge \mathbb{P}$$

As we can see in the first part of the disjunction, we have once again come upon a self satisfying clause.

[3] $\mathbb{P}$ and $\mathbb{Q}$ are the remainders of the RHS disjunctions, see the footnotes of Section 3.3

4 A New Encoding

I will now present a new encoding of context-free grammars. It is in many aspects like Giannaros' encoding, but resolves the problem of circular dependencies. The general idea is that we augment the production rules with the transitive, non-reflexive relation $A \gg B$, informally interpreted as A depends on B. The altered encoding is then executed much like that of Giannaros', without the exclusion condition of the completer rule encoding.

4.1 The Algorithm

The following encoding will, given a context-free grammar $G = (N, \Sigma, P, S)$ and word length n, output the Boolean formula $[\![G]\!]_n$, such that

$$\exists \sigma : eval_\sigma([\![G]\!]_n) = \top \Leftrightarrow \exists w \in \Sigma^* :: |w| = n \wedge w \in \mathbb{L}(G)$$

i.e. $[\![G]\!]_n$ is satisfiable iff there is some string in G of length n. We first construct a more general version of Earley sets, the *spine sets*, Y_0 to Y_n. The difference lies in that while the scanner rule used to construct Earley sets only applies to a terminal if it matches a symbol in the input string, the scanner rule for the spine sets is always applicable when a terminal is encountered.

We introduce *letter variables*, $w_i(a)$ for all $1 \leq i \leq n$ and $a \in \Sigma$, where $w_i(a)$ means that $w_i = a$. We also introduce *Earley set variables*, where $X_i(A \to \alpha \cdot \beta, j)$ means that the Earley item $(A \to \alpha \cdot \beta, j)$ can be added to X_i in a non-recursive way. Stylized letters $\mathcal{P}, \mathcal{Q}, \mathcal{R}$ denote Earley items.

$[\![G]\!]_n$ is initialized as the formula $X_n(S' \to S\cdot, 0)$ and is then incrementally expanded by applying the following rules to every Earley item in every Earley spine set.

For all $i : 0 \leq i \leq n$ and all $\mathcal{P} \in Y_i$:

- If $\mathcal{P} = (A \to \alpha a \cdot \beta, j)$, add as a conjunct to $[\![G]\!]_n$

$$X_i(\mathcal{P}) \Leftrightarrow X_{i-1}(\mathcal{Q}) \wedge w_i(a) \wedge (\mathcal{P}_i \gg \mathcal{Q}_{i-1}) \tag{8}$$

$$\textit{Where } \mathcal{Q} = (A \to \alpha \cdot a\beta, j)$$

- If $\mathcal{P} = (A \to \cdot\beta, i)$ and $A \neq S'$, add as a conjunct to $[\![G]\!]_n$:

$$X_i(\mathcal{P}) \Leftrightarrow \bigvee_{\mathcal{Q}=(B\to\alpha'\cdot A\beta', k)\in Y_i} X_i(\mathcal{Q}) \wedge (\mathcal{P}_i \gg \mathcal{Q}_i) \tag{9}$$

- If $\mathcal{P} = (A \to \alpha B \cdot \beta, j)$, add as a conjunct to $[\![G]\!]_n$:

$$X_i(\mathcal{P}) \Leftrightarrow \bigvee_{\mathcal{Q}=(B\to\gamma\cdot,k)\in Y_i} X_i(\mathcal{Q}) \wedge X_k(\mathcal{R}) \wedge (\mathcal{P}_i) \gg \mathcal{Q}_k) \wedge (\mathcal{P}_i \gg \mathcal{R}_i) \quad (10)$$

$$\textit{Where } \mathcal{R} = (A \to \alpha' \cdot B\beta', j)$$

- If $X_i(\mathcal{P}) = X_0(S' \to \cdot S, 0)$, add as a conjunct to $[\![G]\!]_n$:

$$X_0(S' \to \cdot S, 0) \quad (11)$$

AMO - ALO As with the previous encoding, we need to ensure that there is exactly one true letter variable w_i for each index i. We therefore introduce AMO- and ALO constraints to $[\![G]\!]_n$:

$$\bigwedge_{1\le i\le n} AMO\{w_i(s) \mid s \in \Sigma\} \wedge ALO\{w_i(s) \mid s \in \Sigma\} \quad (12)$$

Properties of $\gg$ We also need to encode the transitivity and non-reflexivity of the relation $\gg$. Here is presented a naive solution to the problem; one obvious optimization would be to keeping track of which dependencies are actually created in the previous phase. For the sake of brevity, this will not be described in this paper.

For all $i : 0 \le i \le n$ and all $\mathcal{P} \in Y_i$:
For all $i : 0 \le j \le n$ and all $\mathcal{Q} \in Y_j$:
For all $i : 0 \le k \le n$ and all $\mathcal{R} \in Y_k$, add as a conjunct to $[\![G]\!]_n$:

$$(\mathcal{P}_i \gg \mathcal{Q}_j) \wedge (\mathcal{Q} \gg \mathcal{R}_k) \Rightarrow (\mathcal{P}_i \gg \mathcal{R}_k) \quad (13)$$

For all $i : 0 \le i \le n$ and all $\mathcal{P} \in Y_i$, add as a conjunct to $[\![G]\!]_n$:

$$\neg(\mathcal{P}_i \gg \mathcal{P}_i) \quad (14)$$

4.2 Example

Let us revisit the case of the extended grammar from Section 3.4, $S \to SS \mid S \mid a \mid \varepsilon$, and the item $(S \to S \cdot S, 0) \in Y_2$. We will now have the following clauses:

$\mathcal{P}$	$(S \to S \cdot S, 0)$
$\mathcal{Q}$	$(S \to S\cdot, 0)$
$\mathcal{R}$	$(S \to \cdot SS, 0)$
$\mathcal{S}$	$(S \to SS\cdot, 0)$
$\mathcal{T}$	$(S \to \cdot S, 0)$
$\mathcal{U}$	$(S \to \cdot, 2)$

Our encoding gives us the following clauses, where $\mathbb{Q}$ and $\mathbb{S}$ are the rest of the disjunctions of Rule 10.

$$X_2(\mathcal{P}) \Rightarrow X_2(\mathcal{Q}) \wedge X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0)$$
$$X_2(\mathcal{Q}) \Rightarrow X_2(\mathcal{S}) \wedge X_0(\mathcal{T}) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0) \vee \mathbb{Q}$$
$$X_2(\mathcal{S}) \Rightarrow X_2(\mathcal{U}) \wedge X_2(\mathcal{P}) \wedge (\mathcal{S}_2 \gg \mathcal{U}_2) \wedge (\mathcal{S}_2 \gg \mathcal{P}_2) \vee \mathbb{S}$$

$$\{\textit{See appendix}\} \Rightarrow$$

$$X_2(\mathcal{P}) \Rightarrow (X_0(\mathcal{R}) \wedge X_0(\mathcal{T}) \wedge X_2(\mathcal{U}) \wedge X_2(\mathcal{P}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{S}_2 \gg \mathcal{P}_2) \wedge \mathbb{R}^4) \vee$$
$$(X_0(\mathcal{R}) \wedge X_0(\mathcal{T}) \wedge \mathbb{P}^5 \wedge \mathbb{S}) \vee$$
$$(X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge \mathbb{Q})$$

$$\{\textit{Rule 13 and repeated application of modus ponens}\} \Rightarrow$$

$$X_2(\mathcal{P}) \Rightarrow (X_0(\mathcal{R}) \wedge X_0(\mathcal{T}) \wedge X_2(\mathcal{U}) \wedge X_2(\mathcal{P}) \wedge (\mathcal{P}_2 \gg \mathcal{P}_2) \wedge \mathbb{R} \vee$$
$$(X_0(\mathcal{R}) \wedge X_0(\mathcal{T}) \wedge \mathbb{P} \wedge \mathbb{S}) \vee$$
$$(X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge \mathbb{Q})$$

$$\{\neg(\mathcal{P}_2 \gg \mathcal{P}_2) \textit{ from Rule 14 }\} \Rightarrow$$

$$X_2(\mathcal{P}) \Rightarrow \bot \vee$$
$$(X_0(\mathcal{R}) \wedge X_0(\mathcal{T}) \wedge \mathbb{P} \wedge \mathbb{S}) \vee$$
$$(X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge \mathbb{Q})$$

$$\{\textit{or-simplification}\} \Rightarrow$$

$$X_2(\mathcal{P}) \Rightarrow (X_0(\mathcal{R}) \wedge X_0(\mathcal{T}) \wedge \mathbb{P} \wedge \mathbb{S}) \vee$$
$$(X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge \mathbb{Q})$$

[4] $\mathbb{R} \Leftrightarrow (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0) \wedge (\mathcal{S}_2 \gg \mathcal{U}_2)$
[5] $\mathbb{P} \Leftrightarrow (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0)$

Not only does this preclude any circular satisfaction of $X_2(\mathcal{P})$, but also opens up the possibility of the variable being set to true, if any of the remaining dependencies can be set to true given the additional $(\mathcal{P}_2 \gg \mathcal{W}_i)$-statements, guaranteeing through the Rules 13 and 14 that any such dependency could not in turn imply $X_2(\mathcal{P})$.

4.3 Translation into CNF

The new encoding is slightly more tricky to translate into CNF than that presented by Gianarros. If we are content with only establishing $\Rightarrow$, i.e. the necessity of the existence of certain Earley items for the LHS item to exist[6], we can get away with fewer clauses, but Rules 9 and 10 will still suffer from blowups in clauses and number of variables.

The scanner encoding, Rule 8, is easily transformed into CNF:

$$A \Rightarrow B \wedge C \wedge D \Leftrightarrow (A \Rightarrow B) \wedge (A \Rightarrow C) \wedge (A \Rightarrow D)$$

Applying this to Rule 8, where $\mathcal{P} = (A \rightarrow \alpha a \cdot \beta, j)$ and $\mathcal{Q} = (A \rightarrow \alpha \cdot a\beta, j)$, yields clauses:

$$X_i(\mathcal{P}) \Rightarrow X_{i-1}(\mathcal{Q})$$
$$X_i(\mathcal{P}) \Rightarrow w_i(a)$$
$$X_i(\mathcal{P}) \Rightarrow (\mathcal{P}_i \gg \mathcal{Q}_{i-1})$$

When translating Rules 9 and 10 we run in to the problem of them being disjunctions of conjunctions - rather the opposite of the conjunctive normal form. A straightforward translation would cause an exponential blowup in the number of clauses and we elect instead to use the Tseitin transformation [9], where an auxiliary variable is added for every conjunction. The Tseitin transformation of the following formula

$$A \Rightarrow \bigvee_{1 \leq i \leq n} x_{i1} \wedge ... \wedge x_{im_i}$$

requires us to introduce the variables $v_1, ..., v_n$, and then the following clauses:

For all $i : 1 \leq i \leq n$:
    For all $j : 1 \leq j \leq m_i$:
        Add $v_i \Rightarrow x_{ij}$ to $[\![G]\!]_n$

[6] Giannaros provides good arguments for the viability of such an approach in Section 3.1.3 in [6], but as with the encoding in general no proof is provided for the correctness of this simplification.

Finally, we add the clause

$$A \Rightarrow \bigvee_{1 \leq i \leq n} v_i$$

The Tseitin transformation gives us the tools to convert the encoding of the scanner and completer rules into CNF. For Rule 9 we get, for $X_i(\mathcal{P}) = X_i(A \to \cdot\beta, i)$

Algorithm 1 Tseitin translation of Rule 9

$V \leftarrow \emptyset$
for $\mathcal{Q} = (B \to \alpha' \cdot A\beta', k) \in Y_i$ **do**
    Introduce a new variable v
    Add clause $v \Rightarrow X_i(\mathcal{Q})$ to $[\![G]\!]_n$
    Add clause $v \Rightarrow (\mathcal{P}_i \gg \mathcal{Q}_i)$ to $[\![G]\!]_n$
    Add variable v to V
Add clause $X_i(\mathcal{P}) \Rightarrow \bigvee_{v \in V} v$ to $[\![G]\!]_n$

To encode Rule 10 for $X_i(\mathcal{P}) = X_i(A \to \alpha B \cdot \beta, j)$, we perform an analogous operation:

Algorithm 2 Tseitin translation of Rule 10

$V \leftarrow \emptyset$
for $\mathcal{Q} = (B \to \alpha' \cdot A\beta', k) \in Y_i$ **do**
    Introduce a new variable v
    Add clause $v \Rightarrow X_i(\mathcal{Q})$ to $[\![G]\!]_n$
    Add clause $v \Rightarrow X_k(\mathcal{R})$ to $[\![G]\!]_n$ ▷ Where $\mathcal{R} = (A \to \alpha' \cdot B\beta', j)$
    Add clause $v \Rightarrow (\mathcal{P}_i \gg \mathcal{Q}_k)$ to $[\![G]\!]_n$
    Add clause $v \Rightarrow (\mathcal{P}_i \gg \mathcal{R}_i)$ to $[\![G]\!]_n$
    Add variable v to V
Add clause $X_i(\mathcal{P}) \Rightarrow \bigvee_{v \in V} v$ to $[\![G]\!]_n$

The ALO part of 12 is already in CNF as one single clause, and translating AMO is trivially done by separating the conjunctions into $O(n^2)$ clauses of the form $\neg z_i \vee \neg z_j$.

Having transformed all clauses of $[\![G]\!]_n$ into disjunctions, we can now use the resulting formula as an input to most major SAT-solvers.

5 Output Size

There are two major candidates for the establishment of an upper bound on number of clauses and variables, and runtime complexity: The application of Rules 8 through 10 when written in CNF, and Rule 13. Rules 11, 12 and 14 produce an obviously smaller number of variables and clauses, and are therefore excluded from this comparison.

5.1 Number of variables

In Section 4 we define a naive implementation of the transitive relationship where we first for every Earley item in every spine set introduce one variable for every Earley item in every spine set. From [6] we get that the number of Earley items in all of the spine set, i.e. $|Y| = \sum_{0 \leq i \leq n} |Y_i|$ is $O(|G^2|n^2)$. The number of variables introduced by the transitivity of the $\gg$-relation is therefore

$$|Y|^2 = O((|G|^2 n^2)^2) = O(|G|^4 n^4)$$

The number of variables created when translating the formula into CNF is also bounded by $O(|G|^4 n^4)$ - in fact, an even lower bound can be established: Translating either Rules 9 and 10 for a variable $X_i(\mathcal{P})$ creates at most $|Y_i|$ number of new variables, and Rule 8 produces none (if we assume the letter variables to be already instantiated). For all $|Y_i|$ items in a given set Y_i, we therefore get an upper bound $O(|Y_i|^2)$ on variables introduced by the above rules. The total sum of variables created through the translation into CNF is thus

$$O(\sum_{0 \leq i \leq n} |Y_i|^2) = O(|Y|^2 - \sum_{0 \leq i \leq n} \sum_{0 \leq j \leq n, i \neq j} |Y_i||Y_j|) = O(|G|^4 n^4 - \sum_{0 \leq i \leq n} \sum_{0 \leq j \leq n, i \neq j} |Y_i||Y_j|)$$

And so, $[\![G]\!]_n$ can be asserted to have $O(|G|^4 n^4)$ variables.

5.2 Number of clauses

Encoding the transitivity of $\gg$ requires one clause to be added for every triplet (x,y,z), where x,y,z are Earley item variables. Since we have $|Y|$ Earley item variables, there are $|Y|^3$ such triplets and the encoding of Rule 13 therefore adds

$$|Y|^3 = O((|G|^2 n^2)^3) = O(|G|^6 n^6)$$

clauses to $[\![G]\!]_n$.

The number of clauses introduced when translating Rules 9 and 10 into CNF is linear in the number of Earley item variables appearing in the RHS disjunction: For

Rule 9 we add 2 clauses per variable in addition to the final clause $X_i(\mathcal{P}) \Rightarrow \bigvee_{v \in V} v$; in translating Rule 10 we add 4. Since the RHS disjunction for an application of Rules 9 and 10 on a variable $X_i(\mathcal{P})$ range over the items in Y_i, the encoding of such a rule will produce $O(|Y_i|)$ new clauses. Since each instance of Rule 8 translation adds a constant number of clauses, the number of clauses added by an arbitrary rule encoding for an item $X_i(\mathcal{P})$ is $O(|Y_i|)$. The number of clauses added for all variables in Y_i is then $O(|Y_i|^2)$, which results in a grand total of

$$O(\sum_{0 \leq i \leq n} |Y_i|^2) = O(|G|^4 n^4 - \sum_{0 \leq i \leq n} \sum_{0 \leq j \leq n, i \neq j} |Y_i||Y_j|)$$

new clauses added to $[\![G]\!]_n$. Since the upper bound on clauses added from Rule 13 also gives an upper bound on those created by the CNF-translation the final bound on clauses for $[\![G]\!]_n$ is $O(|G|^6 n^6)$.

5.3 Comparison with previous encodings

Quimper and Walsh's encoding [8] produces outputs with $O(|G|^2 n^3)$ clauses and the encoding presented by Axelsson et al. [7] produces an output with $O(|G| n^3)$ clauses. These are both significantly lower bounds than $O(|G|^6 n^6)$, but since the state of the art SAT-solvers employ a multitude of heuristics and micro-optimizations, and display a general air of black magic, actual performance is not easily deduced from theoretical ditto. That being said, the discrepancy between the output sizes of previous encodings and those of that proposed here is quite overwhelming. It would be reasonable to think that these theoretical results carry over to the practical realm.

6 Correctness

Caveat Lector *This section is written in a much less lucid style than the rest of the paper. It has been tacked on in a late stage of the process, and the author has had very little time to polish the language and the disposition; the steps taken in the proof are possibly too large to hold up to a reasonable standard of rigor. Several assumptions are made and never proven to hold. It is to be regarded as a bonus feature and nothing essential will be lost, should the reader choose to skip to Section 8*

We have shown specific examples of where the new encoding succeeds when the old one did not, and the augmented algorithm sure seems like it is guarding itself from any sort of "empty implications". The algorithm is however still not proven correct, nor shall a complete formal proof be given here, but we will at least present

a fairly detailed sketch of a proof. This sketch could - and should - be fleshed out and rigorously described at some later point, but will in the meantime give us strong justifications for believing the algorithm to be correct.

6.1 Induction proofs

We will start by defining a general sort of induction that we will later use to show the relationship between the assignments of Earley item variables and the membership of their corresponding Earley items in the Earley sets created in a successful parse.

Definition 11. *(Well-founded Relation) A relation $\prec$ is well-founded* iff *it does not contain any* infinite descending chains*, i.e. there is no infinite sequence $x_1, x_2, ...$ s.t. $x_{n+1} \prec x_n$ for all $n \in \mathbb{N}$.*

For example, the relation $<$ over the natural numbers is well-founded, since no chain can go below 0, but the same relation over the set of all integers is not.

Theorem 1. *(Well-founded Induction) Given a well-founded relation $\prec$ on X, the theorem of* well-founded induction *states that if we can show that for any $x \in X$, the fact that the property $P(y)$ holds for all $y \in X$ s.t. $y \prec x$ implies that it also holds for $P(x)$, then that property holds for every member of X:*

$$(\forall x \in X : (\forall y \in X : (y \prec x \Rightarrow P(y))) \Rightarrow P(x)) \Rightarrow (\forall x \in X : P(x))$$

This tells us that if we can find a partial order over our Earley item variables, preferably one that reflects on the production rules in Section 4, and can show that the desired property (i.e. the above mentioned connection between Earley item variables and Earley items in a run of the parser) by necessity carries over from the right-hand side to the left-hand side in all the rules, we have shown that the property holds for all Earley item variables. The theorem is proven in [11] (Chapter 3).

Previously we said that we want to relate Earley item variable $X_i(\mathcal{P})$ to the membership of the corresponding Earley item and Earley set, $\mathcal{P} \in X_i$. However, that membership query could change during the course of a parse; initially, almost all Earley sets are empty and it's only later that they are incrementally expanded. When asking whether or not $(S' \rightarrow S\cdot, 0) \in X_n$, we obviously do not mean in an arbitrary intermediate state of the parsing. We therefore need to define the concept of the sets in a final state of the parsing algorithm.

Definition 12. *(Final State) A run of the Earley parsing algorithm proposed in [2] is in its* final state *when no Earley item can be added to any set.*

Definition 13. *(Final State Set) We define $E_i^{w,G}$ as the set X_i in the final state of a parse on string w and grammar G.*

Note that the above definitions also allow us to specify which string we parsed in order to obtain the Earley set. This is useful to us since the encoding does not operate on a single string, but rather on all strings in $\mathbb{L}(G)$ of a certain length.

6.2 Production Trees and Derivation Sets

We shall now attempt a definition of a well-founded relation. First we introduce the concept of *production trees*. We can see them informally as acyclic, directed graphs of *item nodes* $N_i(\mathcal{P})^W$ indicating how an item can be added to a set, very much in the same way as the rules in Section 4. Note that the same Earley item variable may exist in several different locations of the graph; we call these different occurrences *instances* of Earley item variables.

Since we want the graph to be acyclic, we need to exclude edges running to an item node higher up in the tree. We do this by introducing *derivation sets* to each node that record what nodes lie along the path down to it from the root node. We then stipulate the following invariant[7]:

$$\textit{A production tree rooted at node } N_i(\mathcal{P})^W \textit{ has no node } N_j(\mathcal{Q})^P, \mathcal{Q}_j \in W \tag{15}$$

Both the node set N and the edge set E of the production tree is defined inductively:

- $N_n(S' \to S\cdot, 0)^{\varnothing} \in N$
- If $N_i(\mathcal{P})^W \in N, \mathcal{P} = (A \to \alpha a \cdot \beta, j)$ then if $\mathcal{Q}_{i-1} \notin W$, where $\mathcal{Q} = (A \to \alpha \cdot a\beta, j)$

$$\begin{gathered} N_{i-1}(Q)^{W \cup \{\mathcal{P}_i\}} \in N, \\ (N_i(\mathcal{P})^W, N_{i-1}(\mathcal{Q})^{W \cup \{\mathcal{P}_i\}}) \in E \end{gathered} \tag{16}$$

- If $N_i(\mathcal{P})^W \in N, \mathcal{P} = (A \to \cdot\beta, i)$ then for all $\mathcal{Q} = (B \to \alpha \cdot A\beta, j) \in Y_i$ s.t. $\mathcal{Q}_i \notin W$

$$\begin{gathered} N_i(\mathcal{Q})^{W \cup \{\mathcal{P}_i\}} \in N, \\ (N_i(\mathcal{P})^W, N_{i-1}(\mathcal{Q})^{W \cup \{\mathcal{P}_i\}}) \in E \end{gathered} \tag{17}$$

[7]For the sake of brevity the invariant remains unproven, but it's quite clear that it holds for the base case, and that the edges maintain it.

- If $N_i(\mathcal{P})^W \in N, \mathcal{P} = (A \to \alpha B \cdot \beta, j)$ then for all $\mathcal{Q} = (B \to \gamma \cdot, k) \in Y_i$ and all $\mathcal{R} = (A \to \alpha \cdot B\beta, j) \in Y_k$ s.t. $\mathcal{Q}_i, \mathcal{R}_k \notin W$

$$\begin{aligned} N_i(\mathcal{Q})^{W \cup \{\mathcal{P}_i\}} &\in N, \\ N_k(\mathcal{R})^{W \cup \{\mathcal{P}_i\}} &\in N, \\ (N_i(\mathcal{P})^W, N_i(\mathcal{Q})^{W \cup \{\mathcal{P}_i\}}) &\in E, \\ (N_i(\mathcal{P})^W, N_k(\mathcal{R})^{W \cup \{\mathcal{P}_i\}}) &\in E \end{aligned} \tag{18}$$

The fact that the derivation sets are strictly increasing when we traverse the edges of the graph shows, along with the invariant (15) and the fact that the derivation sets can not grow infinitely, that the paths are finitely long. We can therefore conclude that the graph is acyclic (since any cycles would allow for infinite paths). Furthermore, we can at last define a well-founded relation on the set $\{(\mathcal{P}, i, W) \mid \mathcal{P} \in Y, 0 \leq i \leq n, W \in Y^*\}$.

Definition 14. *(The relation $\prec$)* $(\mathcal{Q}, j, U) \prec (\mathcal{P}, i, W)$ *iff there is a path in* T *from* $N_i(\mathcal{P})^W$ *to* $N_j(\mathcal{Q})^U$

We also want to relate the concept of derivation sets to the items and sets in an actual Earley parse. If an item $\mathcal{P}$ can be added to some set X_i, then naturally it can be done so without $\mathcal{P}$ to be in X_i in the first place - if we already have $\mathcal{P} \in X_i$ then we have no need for further rule applications in order for $\mathcal{P} \in X_i$ to be true. This also extends to the production children of $\mathcal{P} \in X_i$: If $\mathcal{P}$ can be added to X_i through either of the memberships $\mathcal{Q}_1 \in X_{j_1}, ..., \mathcal{Q}_f \in X_{j_f}$, then we can be assured that there is some $\mathcal{Q}_g \in X_{j_g}$, $1 \leq g \leq f$ for which $\mathcal{P} \in X_i$ is not a prerequisite, lest we shall arrive at a contradiction. With this in mind, we revisit the $E_i^{w,G}$-notation previously defined, but with derivation sets added as a taboo list of item/set pairs:

Definition 15. *We define* $E_i^{w,G,W}$ *as the i:th final state set of a modified Earley parse, where the rules are changed so that* $\mathcal{P}$ *cannot be added to set* X_i *if* $\mathcal{P}_i \in W$*:*

- *Predictor*: $\forall (A \to \alpha \cdot B\beta, j) \in X_i : \forall (B \to \gamma) \in P :$ If $(B \to \cdot\gamma, i)_i \notin W$ then add $(B \to \cdot\gamma, i)$ to X_i)
- *Scanner* (if $i < |w|$): $\forall (A \to \alpha \cdot w_{i+1}\beta, j) \in X_i :$ If $(A \to \alpha w_{i+1} \cdot \beta, j)_{i+1} \notin W$ then add $(A \to \alpha w_{i+1} \cdot \beta, j)$ to X_{i+1}.
- *Completer*: $\forall (A \to \gamma \cdot, j) \in X_i : \forall (B \to \alpha \cdot A\beta', k) \in X_j :$ If $(B \to \alpha A \cdot \beta, k)_i \notin W$ then add $(B \to \alpha A \cdot \beta, k)$ to X_i.

Note that if $W = \varnothing$ then the modified rules won't exclude any items and $E_i^{w,G,W} = E_i^{w,G}$ for all w, G, i.

6.3 The Relation $\gg$ and Derivation Sets

The transitive relation $\gg$ makes the encoding reflect how derivation sets propagate downwards in the production tree: If we interpret $X_j(\mathcal{Q}) \wedge (\mathcal{P}_i \gg \mathcal{Q}_j)$ as "this instance of $X_j(\mathcal{Q})$ has $X_i(\mathcal{P})$ in its derivation set", we can clearly see how the derivation sets accumulate with successive applications of modus ponens: No matter what encoding rule we apply on a left-hand side variable $X_i(\mathcal{P})$, each disjunction $X_j(\mathcal{Q})$ on the right-hand side will not only come with the property $X_i(\mathcal{P}) \gg X_j(\mathcal{Q})$ but will also, through the application of modus ponens and the transitive property of $\gg$, inherit the derivation set of the instance it was produced from. We can write any of the rules of the form

$$X_i(\mathcal{P}) \Leftrightarrow \mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge (\mathcal{P}_i \gg \mathcal{Q}_j) \vee \mathbb{P}_d$$

where $\mathbb{P}_c$ is the conjunctive part in the disjunction: $w_i(a)$ for some symbol a in the case of Rule 8, $\top$ in the case of Rule 9 and $X_k(\mathcal{R})$ for some $k, \mathcal{R}$ in the case of Rule 10. $\mathbb{P}_d$ is the remaining disjunction; for Rule 8 this is $\bot$. We therefore get the equivalence

$$X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)$$

$$\Leftrightarrow$$

$$\mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge (\mathcal{P}_i \gg \mathcal{Q}_j) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i) \vee (\mathbb{P}_d \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i))$$

$$\Leftrightarrow$$

$$\mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_j) \vee (\mathbb{P}_d \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{Q}_j))$$

In the first disjunction we can see that $\{\mathcal{P}_i\}$ has been added to the original derivation set W.

6.4 Proof

We start by defining a lemma stating that if an instance of a disjunct on the right-hand side of some rule application is not in the production tree of an instance of the left-hand side variable, then that disjunct is equivalent to $\bot$. The lemma remains unproven due to the limited scope of the thesis, but should be fairly self-evident.

Lemma 1. [8] *Given the rule*

$$X_i(\mathcal{P}) \Leftrightarrow (\mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge (\mathcal{P}_i \gg \mathcal{Q}_j)) \vee \mathbb{P}_d$$

[8] This lemma remains unproven for the sake of brevity, but ought to be fairly self-evident.

$(\mathcal{Q}, j, W \cup \{\mathcal{P}_i\}) \not\prec (\mathcal{P}, i, W)$ *implies that*

$$(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} \mathcal{S}_l \gg \mathcal{P}_i) \Leftrightarrow \bot \vee (\mathbb{P}_d \wedge \bigwedge_{\mathcal{S}_l \in W} \mathcal{S}_l \gg \mathcal{P}_i)$$

Lemma 2. $\mathcal{P} \in E_i^{w,G,W}$ *implies that there is a pair* $(\mathcal{Q}, j)$ *s.t.* $(\mathcal{Q}, j, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W)$.

Proof for Lemma 2 First off: $\mathcal{P}_i \notin W$; otherwise $\mathcal{P}$ could not be added to X_i for any parse run (by the definition of the $E_i^{w,G,W}$-structure). Secondly, there must be at least one $\mathcal{Q}_j$ among the potential producers of $\mathcal{P}_i$ that can be produced without the item/set pairs in W: If not, then we obviously cannot add $\mathcal{P}$ to X_i without the same pairs. Therefore there must be some potential producing item/set pair $\mathcal{Q}_j$ (or two such, should we deal with an instance of the completer rule) such that $\mathcal{Q}_j \notin W$. Rules 16 to 18 tell us that we then have an edge $(N_i(\mathcal{P})^W, N_j(\mathcal{Q}^{W \cup \{\mathcal{P}_i\}})$. The definition of $\prec$ tells us then that $(\mathcal{Q}, j, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W)$.

We will use induction over the order established by $\prec$ to show the relationship between satisfiability of logical formulae and membership of their corresponding Earley item/set pairs. We shall, for the induction part of the proof, exclude the clause $X_n(S' \rightarrow S\cdot, 0)$ from $[\![G]\!]_n$ and reintroduce it further on. We call the reduced formula $[\![G']\!]_n$.

Theorem 2. *For all* $\mathcal{P}, i, W$*:*

$$\exists \sigma : eval_\sigma(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{Q}_j \in W} (\mathcal{Q}_j \gg \mathcal{P}_i)) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

$$\Leftrightarrow$$

$$\exists w \in \Sigma^* : \mathcal{P} \in E_i^{w,G,W}$$

Proof Assume that the above property holds for all $(\mathcal{R}, k, U)$, *s.t.* $(\mathcal{R}, k, U) \prec (\mathcal{P}, i, W)$. We will show that it holds for $(\mathcal{P}, i, W)$ by first showing how, for each rule used to construct the production tree, the above property will be preserved and then showing that it holds for all minimal elements in the order established by $\prec$.

6.4.1 Scanner Rule (8)

If $\mathcal{P} = (A \rightarrow \alpha a \cdot \beta, j)$ then $\mathcal{Q} = (A \rightarrow \alpha \cdot a\beta, j)$. In order to show that the property persists when we move upwards in the production tree by the application of Rule 16, we need to prove that

$$(\mathcal{Q}, i-1, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge \exists w \in \Sigma^* : \mathcal{Q} \in E_{i-1}^{w,G,W \cup \{\mathcal{P}_i\}}$$

$$\Leftrightarrow \tag{19}$$
$$\exists w \in \Sigma^* : \mathcal{P} \in E_i^{w,G,W}$$

and that

$$(\mathcal{Q}, i-1, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge \exists \sigma : eval_\sigma(X_{i-1}(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_{i-1})) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

$$\Leftrightarrow \tag{20}$$
$$\exists \sigma : eval_\sigma(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

If $(\mathcal{Q}, i-1, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W)$ then we either have

$$\exists w \in \Sigma^* : \mathcal{Q} \in E_{i-1}^{w,G,W \cup \{\mathcal{P}\}}$$

or

$$\neg \exists w \in \Sigma^* : \mathcal{Q} \in E_{i-1}^{w,G,W \cup \{\mathcal{P}\}}$$

20 $\Rightarrow$ In the first case we know that at some point in the parse of a word w, $\mathcal{Q} \in X_{i-1}$, and that a scanner rule application on the same item will add $\mathcal{P}$ to the set X_i. We also know that $\mathcal{P}_i \notin W$, since $(\mathcal{P}, i, W)$ is the root of a subtree and the invariant 14 thus prohibits $\mathcal{P}_i \in W$. Therefore, if we can add $\mathcal{Q}$ to X_{i-1} without the use of the item/set pairs in $W \cup \{\mathcal{P}_i\}$, then we can add $\mathcal{P}$ to X_i without the use of the item/set pairs in W. We can therefore conclude that $\mathcal{P} \in E_i^{w,G,W}$.

20 $\Leftarrow$ The other case is quite straightforward: If there is no word w for which $(A \to \alpha \cdot a\beta, j) \in E_{i-1}^{(w,G,W \cup \{\mathcal{P}_i\})}$ then there is no item to apply the scanner rule to in order to produce $(A \to \alpha a \cdot \beta, j) \in E_i^{(w,G,W)}$.

19 $\Rightarrow$ Looking at the logical formula, we will have to apply Rule 8 and so get the equivalence

$$X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)$$
$$\Leftrightarrow$$
$$X_{i-1}(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_{i-1}) \wedge w_i(a)$$

Naturally, if there is no σ, such that

$$eval_\sigma(X_{i-1}(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_{i-1})) = \top$$

then nor can there be a σ satisfying that formula with the constraint $w_i(a)$ added, since $eval_\sigma(A \wedge B) = \top$ if and only if both $eval_\sigma(A) = \top$ and $eval_\sigma(B) = \top$, and any σ satisfying the more constrained formula would also satisfy the less constrained one - contrary to our assumption.

We can thus conclude that

$$\neg\exists\sigma : eval_\sigma(X_{i-1}(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_{i-1})) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

$$\Rightarrow$$

$$\neg\exists\sigma : eval_\sigma(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{Q}_j \in W} (\mathcal{Q}_j \gg \mathcal{P}_i)) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

19 $\Leftarrow$ If, on the other hand, there is a σ satisfying the less constrained formula, then we only need to show that any such σ also can satisfy the constraint $w_i(a)$ to show that

$$\exists\sigma : eval_\sigma(X_{i-1}(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_{i-1})) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

$$\Rightarrow$$

$$\exists\sigma : eval_\sigma(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

The only reason that $eval_\sigma(w_i(a)) = \bot$ is if

$$(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)) \Rightarrow w_i(b), \quad b \neq a$$

since the AMO constraint is the only constraint keeping us from setting all $w_k(c)$ to $\top$. However, no movement upwards (that is applying the rules from right to left) in the production tree requires us to set any letter variables to true[9]; this only happens when we move downwards in the tree. We thus need to show that no series of left-to-right rule applications will cause the undesired implication. Since we only mention $w_i(b)$ (for some b) when the variable on the left-hand side is on the form $X_i(\mathcal{R})$, and the only left-to-right implication of the above mentioned $X_i(\mathcal{P})...$ is $X_{i-1}(\mathcal{Q})...$ the implications would have to be on the form

$$(X_i(\mathcal{P})...) \Rightarrow (X_{i-1}(\mathcal{Q})...)$$

$$(X_{i-1}(\mathcal{Q})...) \Rightarrow (X_i(\mathcal{R})...)$$

[9]Once again, as the scope of this thesis is limited this statement lacks formal proof

$$(X_i(\mathcal{R})...) \Rightarrow (w_i(b)...)$$

But since $m \leq n$ for all $(\mathcal{S}, m, U) \prec (\mathcal{T}, n, V)$, we cannot reach (R, i, W') from $(\mathcal{Q}, i-1, W \cup \{\mathcal{P}_i\})$. We therefore have no conflicting letter variable $w_i(b)$, and are free to set $w_i(a)$ to $\top$.

We have yet to show what happens when $(\mathcal{Q}, i-1, W \cup \{\mathcal{P}_i\}) \not\prec (\mathcal{P}, i, W)$, i.e. we're dealing with the scanner rule: Lemma 1 tells us that

$$(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} \mathcal{S}_l \gg \mathcal{P}_i) \Leftrightarrow \bot \vee (\mathbb{P}_d \wedge \bigwedge_{\mathcal{S}_l \in W} \mathcal{S}_l \gg \mathcal{P}_i)$$

and so $X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W}(\mathcal{S}_l \gg \mathcal{P}_i)$ is not satisfiable, since $\mathbb{P}_d \Leftrightarrow \bot$ for the scanner case.

Likewise, if $(\mathcal{Q}, i-1, W \cup \{\mathcal{P}_i\}) \not\prec (\mathcal{P}, i, W)$ it can only be because $\mathcal{Q}_{i-1} \in W$ (this being the only criterion for $(\mathcal{Q}, i-1, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W)$). If we then modify the parser rules so as not to add the items in $W \cup \{\mathcal{P}_i\}$ to their corresponding sets, and $\mathcal{Q}_i \in W$, we will at no point have $\mathcal{Q} \in X_i$ for any parse run. We can therefore conclude that $\mathcal{Q} \notin E_{i-1}^{w,G,W\cup\{\mathcal{P}_i\}}$ and so we have no item to apply the scanner rule to in order to add $\mathcal{P}$ to $E_i^{w,G,W}$.

We have now shown both $A \Rightarrow B$ and $\neg A \Rightarrow \neg B$, which is the same as $A \Leftrightarrow B$ for both the parsing semantics and the encoding. Using our induction assumption we can conclude that for $\mathcal{P} = (A \rightarrow \alpha a \cdot \beta, j)$, the property is preserved:

$$\exists \sigma : eval_\sigma(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

$$\Leftrightarrow$$

$$(\mathcal{Q}, i-1, W\cup\{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge \exists \sigma : eval_\sigma(X_{i-1}(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W\cup\{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_{i-1})) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

$$\Leftrightarrow$$

$$(\mathcal{Q}, i-1, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge \exists w \in \Sigma^* : \mathcal{Q} \in E_{i-1}^{w,G,W\cup\{\mathcal{P}_i\}}$$

$$\Leftrightarrow$$

$$\exists w \in \Sigma^* : \mathcal{P} \in E_i^{w,G,W}$$

6.4.2 Predictor Rule (9)

In order to show that the property persists when we move upwards in the production tree by the application of Rule 17, we need to prove that for $\mathcal{P} = (A \rightarrow \cdot\gamma, i)$:

$$\exists \sigma : eval_\sigma(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

$$\Leftrightarrow \tag{21}$$

$$\exists \sigma, \mathcal{Q} : (\mathcal{Q}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge eval_\sigma(X_i(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_i)) = \top$$

$$\wedge\ eval_\sigma([\![G']\!]_n) = \top$$

$$Where\ \mathcal{Q} = (B \rightarrow \alpha \cdot A\beta, j)$$

and that

$$\exists w \in \Sigma^* : \mathcal{P} \in E_i^{w,G,W}$$

$$\Leftrightarrow \tag{22}$$

$$\exists w \in \Sigma^*, \mathcal{Q} : (\mathcal{Q}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge |w| = n \wedge \mathcal{Q} \in E_i^{w,G,W \cup \{\mathcal{P}_i\}}$$

$$Where\ \mathcal{Q} = (B \rightarrow \alpha \cdot A\beta, j)$$

We will begin with noting that $\mathcal{Q} \in Y_i$ for all $\mathcal{Q} = (B \rightarrow \alpha \cdot A\beta, j)$ such that $(\mathcal{Q}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W)$. This is easily realized from the way we construct the production tree, more precisely how Rule 17 is formulated: $(\mathcal{R}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W)\}$ for all $\mathcal{R} \in Y_i$ such that $\mathcal{R}_i \notin W$.

21 $\Leftarrow$ Rule 9 gives us that

$$X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)$$

$$\Leftrightarrow$$

$$\bigvee_{\mathcal{Q}=(B \rightarrow \alpha \cdot A\beta, j) \in Y_i} (X_i(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_i)))$$

Since all $\mathcal{Q} = (B \rightarrow \alpha \cdot A\beta, j)$ such that $(\mathcal{Q}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W)$ are in Y_i, any σ satisfying

$$X_i(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_i)$$

will also satisfy

$$\bigvee_{\mathcal{Q}=(B \rightarrow \alpha \cdot A\beta, j) \in Y_i} (X_i(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_i)))$$

We therefore only need to find one such pair of $\mathcal{Q}$ and σ in order to know that $X_i(\mathcal{P})$... can be satisfied:

$$\exists \sigma, \mathcal{Q} : (\mathcal{Q}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge eval_\sigma(X_i(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_i)) = \top$$

$$\wedge\ eval_\sigma([\![G']\!]_n) = \top$$

$$\Rightarrow$$

$$\exists \sigma : eval_\sigma(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

$$Where\ \mathcal{Q} = (B \to \alpha \cdot A\beta, j)$$

21 $\Rightarrow$ In the opposite case, where no satisfying $\sigma/\mathcal{Q}$-pair can be found, the right-hand side of the equivalence (and consequently the left-hand side) will have to rely on those disjuncts

$$X_i(\mathcal{R}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_i)$$

where $(\mathcal{R}, i, U) \not\prec (\mathcal{P}, i, W)$. But Lemma 1 tells us that these disjuncts will result in $\perp$ - which is intrinsically unsatisfiable - when introduced in the context of $X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)$. Since the right-hand side of the equality cannot be satisfied by neither those disjuncts in the production tree of $(\mathcal{P}, i, W)$, nor those not in it, the absence of satisfying $\sigma/\mathcal{Q}$-pairs will result in the absence of a satisfying σ for the $(\mathcal{P}, i, W)$-clause, or:

$$\neg\exists \sigma, \mathcal{Q} : (\mathcal{Q}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge eval_\sigma(X_i(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_i)) = \top$$

$$\wedge\ eval_\sigma([\![G']\!]_n) = \top$$

$$\Rightarrow$$

$$\neg\exists \sigma : eval_\sigma(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

$$Where\ \mathcal{Q} = (B \to \alpha \cdot A\beta, j)$$

22 $\Leftarrow$ If we, in parsing some word w, can add an item $\mathcal{Q} = (B \to \alpha \cdot B\beta, j)$ to the set X_i without the use of the item/set pairs in $W \cup \{\mathcal{P}_i\}$, then relaxing the taboo constraint to just W and applying the predictor rule (9) on $\mathcal{Q}$ will add $\mathcal{P}$ to X_i, and we thus get $\mathcal{P} \in E_i^{w,G,W}$.

22 $\Rightarrow$ If we assume that there is no $w \in \Sigma^*$ and $\mathcal{Q} = (B \to \alpha \cdot A\beta, j)$ such that $(\mathcal{Q}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge |w| = n \wedge \mathcal{Q} \in E_i^{w,G,W\cup\{\mathcal{P}_i\}}$, then the occurrence of $\mathcal{P} = (A \to \cdot\gamma, i) \in X_i$ in the parsing of any w must have been produced by some $\mathcal{R} = (B \to \alpha \cdot A\beta, j) \in X_i$ for the same w, where $(\mathcal{R}, i, U) \not\prec (\mathcal{P}, i, W)$. In order to ensure that no item/set pair in W has been employed in producing $\mathcal{P} \in X_i$, this constraint must obviously be fulfilled by the producing item $\mathcal{R} \in X_i$. In addition, there must be at least some item/set pair producing $\mathcal{P} \in X_i$ which is in turn producible without employing $\mathcal{P}_i$, lest $\mathcal{P} \in X_i$ should be dependent on itself. Thus, the most lax constraint we can put on $\mathcal{R} \in X_i$ is that the item/set pairs in $W \cup \{\mathcal{P}_i\}$ have not been employed in its production. This in turn means that U must encompass all those pairs, i.e. $W \cup \{\mathcal{P}_i\} \subseteq U$. We also note that all the items we can add to a set X_i while ignoring the pairs in $V \cup V'$ we also can add to X_i if we relax the constraint to only ignore the pairs in V, thus $E_i^{w,G,V\cup V'} \subseteq E_i^{w,G,V}$. We use this observation to state that if $\mathcal{R} \in E_i^{w,G,U}$ and $W \cup \{\mathcal{P}_i\} \subseteq U$, then $E_i^{w,G,U} \subseteq E_i^{w,G,W\cup\{\mathcal{P}_i\}}$ and consequently $\mathcal{R} \in E_i^{w,G,W\cup\{\mathcal{P}_i\}}$. But looking at Rule 17, the only reason $(\mathcal{R}, i, U) \not\prec (\mathcal{P}, i, W)$ would be that $\mathcal{R}_i \in W$, and so $\mathcal{R}$ by the definition of the derivation sets cannot be added to X_i, and so $\mathcal{R} \notin E^{w,G,W\cup\{\mathcal{P}_i\}}$. If $\mathcal{R}$ cannot be added to X_i without employing the item/set pairs in $W \cup \{\mathcal{P}_i\}$, then nor can it be used if we want to add $\mathcal{P}$ to X_i without employing the pairs in W. Thus, if

$$\neg\exists w \in \Sigma^*, \mathcal{Q} : (\mathcal{Q}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge |w| = n \wedge \mathcal{Q} \in E_i^{w,G,W\cup\{\mathcal{P}_i\}}$$

$$Where\ \mathcal{Q} = (B \to \alpha \cdot A\beta, j)$$

and since no $\mathcal{R} \in E_i^{w,G,U}$ such that $(\mathcal{R}, i, U) \not\prec (\mathcal{P}, i, W)$ can be used to construct $\mathcal{P} \in E_i^{w,G,W}$, we have

$$\neg\exists w \in \Sigma^* : \mathcal{P} \in E_i^{w,G,W}$$

Once again, we have shown that $A \Rightarrow B$ and $\neg A \Rightarrow \neg B$ for both the Earley set membership and the encoding rules. We therefore conclude that for $\mathcal{P} = (A \to \cdot\gamma, i)$:

$$\exists\sigma : eval_\sigma(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$

$$\Leftrightarrow$$

$$\exists\sigma, \mathcal{Q} : (\mathcal{Q}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge eval_\sigma(X_i(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W\cup\{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_i)) = \top$$

$$\wedge\ eval_\sigma([\![G']\!]_n) = \top$$

$$Where\ \mathcal{Q} = (B \to \alpha \cdot A\beta, j)$$

$$\Leftrightarrow$$

$$\exists w \in \Sigma^*, \mathcal{Q} : (\mathcal{Q}, i, W \cup \{\mathcal{P}_i\}) \prec (\mathcal{P}, i, W) \wedge |w| = n \wedge \mathcal{Q} \in E_i^{w,G,W \cup \{\mathcal{P}_i\}}$$
$$Where\ \mathcal{Q} = (B \to \alpha \cdot A\beta, j)$$
$$\Leftrightarrow$$
$$\exists w \in \Sigma^* : \mathcal{P} \in E_i^{w,G,W}$$

6.4.3 Completer Rule (10)

The proof for the completer rule is so analogous to that for the predictor rule that it will not be presented here.

6.4.4 Base cases

There are two possible situations for which we cannot further extend the production tree from $N_i(\mathcal{P})^W$; either there are no producers of the node - this happens for $(\mathcal{P}, i) = ((S' \to \cdot S, 0), 0)$ - or all the potential producers of $\mathcal{P}_i$ are in W. In the first case, $(S' \to \cdot S, 0)$ is always a member of $E_0^{w,G,W}$ iff $(S' \to \cdot S, 0) \notin W$ and $X_0(S' \to \cdot S, 0) \wedge \bigwedge_{\mathcal{S}_l \in W}(\mathcal{S}_l \gg (S' \to \cdot S, 0)_0)$ is true iff $(S' \to \cdot S, 0) \notin W$, since $X_0(S' \to \cdot S, 0)$ is given as true in $[\![G]\!]_n$.

In the second case Lemma 1 shows that every disjunct

$$\mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_j)$$

on the right-hand side of

$$X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{S}_l \in W} (\mathcal{S}_l \gg \mathcal{P}_i)$$
$$\Leftrightarrow$$
$$((\mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge \bigwedge_{\mathcal{S}_l \in W \cup \{\mathcal{P}_i\}} (\mathcal{S}_l \gg \mathcal{Q}_j)) \vee ((\mathbb{P}_d \wedge \bigwedge_{\mathcal{S}_l \in W} \mathcal{S}_l \gg \mathcal{P}_i) \vee \mathbb{R})$$

will turn out to be false, rendering the left-hand side unsatisfiable.

Similarly, taking the inverse of Lemma 2 gives us that

$$\neg\exists(\mathcal{Q}, j, U) : (\mathcal{Q}, j, U) \prec (\mathcal{P}, i, W) \Rightarrow \mathcal{P} \notin E_i^{w,G,W}$$

and so the property holds for all the base cases.

6.4.5 In conclusion

We have shown that for all $\mathcal{P}, i, W$:

$$\exists\sigma : eval_\sigma(X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{Q}_j \in W} (\mathcal{Q}_j \gg \mathcal{P}_i)) = \top \wedge eval_\sigma([\![G']\!]_n) = \top$$
$$\Leftrightarrow$$
$$\exists w \in \Sigma^* : \mathcal{P} \in E_i^{w,G,W}$$

We defined $[\![G']\!]_n$ as the formula $[\![G]\!]_n$ produced by the encoding rules in Section 4, without the clause $X_n(S' \to S\cdot, 0)$, that is

$$[\![G]\!]_n \Leftrightarrow [\![G']\!]_n \wedge X_n(S' \to S\cdot, 0)$$

We can extend the right-hand side with an empty conjunction, since

$$\bigwedge_{\mathcal{S}_l \in \varnothing} (\mathcal{S}_l \gg \mathcal{T}_j) \Leftrightarrow \top$$

and thus get

$$[\![G]\!]_n \Leftrightarrow [\![G']\!]_n \wedge X_n(S' \to S\cdot, 0) \wedge \bigwedge_{\mathcal{S}_l \in \varnothing} (\mathcal{S}_l \gg X_n(S' \to S\cdot, 0))$$

From this follows that

$$\exists\sigma : eval_\sigma([\![G]\!]_n) = \top$$
$$\Leftrightarrow$$
$$\exists\sigma : eval_\sigma([\![G']\!]_n \wedge X_n(S' \to S\cdot, 0) \wedge \bigwedge_{\mathcal{S}_l \in \varnothing} (\mathcal{S}_l \gg X_n(S' \to S\cdot, 0))) = \top$$
$$\Leftrightarrow$$
$$\exists w \in \Sigma^* : (S' \to S\cdot, 0) \in E_n^{w,G,\varnothing}$$

And since for all $w, G, i : E_i^{w,G,\varnothing} = E^{w,G}$, i.e. if $W = \varnothing$ then none of the parsing rules will be modified and the set $E_i^{w,G,W}$ is the same as it would be on a normal run of the Earley parse. The statement $(S' \to S\cdot, 0) \in E_n^{w,G}$ tells us that there is some word w, the parsing of which will in its final state have $(S' \to S\cdot, 0) \in X_n$, which in turn tells us that $\exists w \in \Sigma^* : w \in \mathbb{L}(G) \wedge |w| = n$. We therefore get the desired equality

$$\exists\sigma : eval_\sigma([\![G]\!]_n) = \top$$
$$\Leftrightarrow$$
$$\exists w \in \Sigma^* : (S' \to S\cdot, 0) \in E_n^{w,G,\varnothing}$$
$$\Leftrightarrow$$
$$\exists w \in \Sigma^* : w \in \mathbb{L}(G) \wedge |w| = n$$

That is, $[\![G]\!]_n$ is satisfiable if and only if there is a word w in the language generated by G of length n. $\square$

7 Discussion

A major shortcoming of this thesis is lack of a fully fleshed out proof. If indeed there ever was a point to this thesis, it would be to prove that an encoding algorithm based on Earley parsing can work. As it stands now the proof sketch might be intuitively convincing, but is certainly not unassailable. This and the absence of empirical data are due to a shortage of time largely caused by the author having followed too many red herrings and having spent too much time searching for other, better solutions: The current solution was in fact conceived at an early stage, but due to its large output size the author looked for ways to a priori weed out any circular dependencies. At one point the project revolved solely around chartering the dependencies in the variable graph. Ultimately this approach was scrapped since it failed to yield any improvements regarding run time or output size, and the current solution was finally decided upon. Another result of these time-consuming diversions is the rather lengthy and cumbersome style in which the proof sketch has been written. In the words of Pascal: *I would have written a shorter letter, but I did not have the time.*

8 Conclusion

8.1 Summary

A SAT encoding presented by Giannaros in [6] has been scrutinized, and a flaw has been found and discussed in Section 3.4. A new encoding aimed at remedying said flaw is suggested in Section 4, and through the example in Section 4.2 it is reasonable to believe that the new encoding is successful in this; furthermore a proof sketch is presented. This, however, rests on a handful of lemmas that, though intuitively acceptable, remain unproven.

A way to algorithmically translate the encoding formula into CNF is laid out, and an upper bound for the sizes of the resulting formulae is established in Section 5. It is found to be thus: $O(|G|^6 n^6)$ number of clauses, with $O(|G|^4 n^4)$ number of variables. In Section 5.3 this result is compared with two previous encodings - that of Quimper and Walsh [8] and that of Axelsson et al [7] - and is found to give a worse theoretical bound on the output size. No empirical tests have been made, but an argument is made for the guess that the poor theoretical performance of the new encoding also yields a poor practical performance.

Three goals for future work are stipulated in Section 8.2, addressing the problems of the large upper bound on clause size as well as the lack of correctness proof and empirical data.

8.2 Future Work

Theoretical performance As the results in Section 5 shows, the most pressing optimization required for this algorithm to be of any practical use is that of finding a leaner way to represent the transitivity of the $\gg$-relation. The optimization hinted at in Section 4 would be a reasonable first step, but there is no doubt more work to be done here. An alternative approach would be to only encode the production trees introduced in Section 6 - this should yield strictly less variables than the current method.

Actual performance As has been discussed in previous sections, the correlation between theoretical and actual performance is rather weak. In [6] Giannaros presents test data for a variety of grammars - some of theoretical interest (like the infinite ambiguous grammar shown in 2.2) and some of practical applicability (Java syntax, at most k-grammars). Anyone endeavoring to run empirical tests on this encoding would do well in studying the results presented therein.

Proof The proof, such as it is, rests on several unproven assumptions, most notably Lemma 1 and the invariant (14). The current proof also omits how the letter variables are related to letters in a parsed word.

References

[1] S. Cook *The complexity of theorem-proving procedures* Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, 1971

[2] J. Aycock and R. N. Horspool, *Practical Earley parsing*, The Computer Journal, vol. 45, no. 6, 2002

[3] J. Earley, *An efficient context-free parsing algorithm.* Communications of the ACM, vol. 13, no. 2, 1970

[4] J. Barwise and J. Etchemendy *Language, Proof and Logic* CSLI Publications, 2002

[5] Cormen, Leiserson, Rivest, Stein *Introduction to Algorithms* The MIT Press, 2009

[6] P.A. Giannaros, *A new SAT encoding of context-free grammars.* Master's thesis, University of Cambridge, submitted June 15th 2011

[7] R. Axelsson, K. Heljanko, and M. Lange, *Analyzing context-free grammars using an incremental SAT solver* Lecture Notes in Computer Science, vol. 5126, pp. 410 - 422, 2008

[8] C. G. Quimper and T. Walsh, *Decomposing global grammar constraints* Principles and Practice of Constraint Programming, pp. 590 - 604, 2007

[9] A. Biere, M. Heule, H. van Maaren and T. Walsh, *Handbook of Satisfiability* IOS Press, 2009

[10] M. Lange and H. Leiss, *To CNF or not to CNF? An efficient yet presentable version of the CYK algorithm*, Informatica Didactica, vol. 8, 2009

[11] G. Winskel *The Formal Semantics of Programming Languages* MIT Press, 2001

9 Appendix

9.1 Logical derivation from Section 4.2

$$X_2(\mathcal{P}) \Leftrightarrow X_2(\mathcal{Q}) \wedge X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0)$$
$$X_2(\mathcal{Q}) \Leftrightarrow X_2(\mathcal{S}) \wedge X_0(\mathcal{T}) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0) \vee \mathbb{Q}$$
$$X_2(\mathcal{S}) \Leftrightarrow X_2(\mathcal{U}) \wedge X_2(\mathcal{P}) \wedge (\mathcal{S}_2 \gg \mathcal{U}_2) \wedge (\mathcal{S}_2 \gg \mathcal{P}_2) \vee \mathbb{S}$$

We apply equivalence substitution on the last two clauses

$$X_2(\mathcal{P}) \Leftrightarrow X_2(\mathcal{Q}) \wedge X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0)$$

$$X_2(\mathcal{Q}) \Leftrightarrow ((X_2(\mathcal{U}) \wedge X_2(\mathcal{P}) \wedge (\mathcal{S}_2 \gg \mathcal{U}_2) \wedge (\mathcal{S}_2 \gg \mathcal{P}_2) \vee \mathbb{S}) \wedge$$
$$X_0(\mathcal{T}) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0)) \vee \mathbb{Q}$$

...and again on the remaining two clauses

$$X_2(\mathcal{P}) \Leftrightarrow (((X_2(\mathcal{U}) \wedge X_2(\mathcal{P}) \wedge (\mathcal{S}_2 \gg \mathcal{U}_2) \wedge (\mathcal{S}_2 \gg \mathcal{P}_2) \vee \mathbb{S}) \wedge$$

$$X_0(\mathcal{T}) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0)) \vee \mathbb{Q})$$
$$\wedge X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0)$$

We then distribute the last conjunction

$$X_2(\mathcal{P}) \Leftrightarrow (X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge$$

$$(X_2(\mathcal{U}) \wedge X_2(\mathcal{P}) \wedge (\mathcal{S}_2 \gg \mathcal{U}_2) \wedge (\mathcal{S}_2 \gg \mathcal{P}_2) \vee \mathbb{S}) \wedge$$
$$X_0(\mathcal{T}) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0)) \vee$$
$$(X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge \mathbb{Q})$$

We use commutativity and associativity of $\wedge$ for some rearranging

$$X_2(\mathcal{P}) \Leftrightarrow ((X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge X_0(\mathcal{T}) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0)) \wedge$$

$$(X_2(\mathcal{U}) \wedge X_2(\mathcal{P}) \wedge (\mathcal{S}_2 \gg \mathcal{U}_2) \wedge (\mathcal{S}_2 \gg \mathcal{P}_2) \vee \mathbb{S}) \vee$$
$$(X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge \mathbb{Q})$$

And then distribute the first clause over the disjunction $(X_2(\mathcal{U})... \vee \mathbb{S})$

$$X_2(\mathcal{P}) \Leftrightarrow (X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge X_0(\mathcal{T}) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge$$

$$(\mathcal{Q}_2 \gg \mathcal{T}_0) \wedge X_2(\mathcal{U}) \wedge X_2(\mathcal{P}) \wedge (\mathcal{S}_2 \gg \mathcal{U}_2) \wedge (\mathcal{S}_2 \gg \mathcal{P}_2)) \vee$$

$$(X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge X_0(\mathcal{T}) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0)) \wedge \mathbb{S}) \vee$$
$$(X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge \mathbb{Q})$$

We do some reshuffling within the disjuncts and use the shortenings $\mathbb{P}$[10]and $\mathbb{R}$[11], and so get

$$X_2(\mathcal{P}) \Leftrightarrow (X_0(\mathcal{R}) \wedge X_0(\mathcal{T}) \wedge X_2(\mathcal{U}) \wedge X_2(\mathcal{P}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{S}_2 \gg \mathcal{P}_2) \wedge \mathbb{R}) \vee$$
$$(X_0(\mathcal{R}) \wedge X_0(\mathcal{T}) \wedge \mathbb{P} \wedge \mathbb{S}) \vee$$
$$(X_0(\mathcal{R}) \wedge (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge \mathbb{Q})$$

9.2 Logical derivation from Section 6

$$X_k(\mathcal{R}) \Leftrightarrow (X_i(\mathcal{P}) \wedge \bigwedge_{\mathcal{W}_l \in A} \mathcal{W}_l \gg \mathcal{P}_i) \vee \mathbb{R}$$
$$X_i(\mathcal{P}) \Leftrightarrow \mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge (\mathcal{P}_i \gg \mathcal{Q}_j) \vee \mathbb{P}_d$$

And by modus ponens we get

$$X_k(\mathcal{R}) \Leftrightarrow ((\mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge (\mathcal{P}_i \gg \mathcal{Q}_j) \vee \mathbb{P}_d) \wedge \bigwedge_{\mathcal{W}_l \in A} \mathcal{W}_l \gg \mathcal{P}_i) \vee \mathbb{R}$$

Which we can expand to

$$X_k(\mathcal{R}) \Leftrightarrow ((\mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge (\mathcal{P}_i \gg \mathcal{Q}_j) \wedge \bigwedge_{\mathcal{W}_l \in A} \mathcal{W}_l \gg \mathcal{P}_i) \vee (\mathbb{P}_d \wedge \bigwedge_{\mathcal{W}_l \in A} \mathcal{W}_l \gg \mathcal{P}_i)) \vee \mathbb{R}$$

We distribute "and"

$$X_k(\mathcal{R}) \Leftrightarrow ((\mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge (\mathcal{P}_i \gg \mathcal{Q}_j) \wedge \bigwedge_{\mathcal{W}_l \in A} ((\mathcal{W}_l \gg \mathcal{P}_i) \wedge (\mathcal{P}_i \gg \mathcal{Q}_j))) \vee \mathbb{S}$$[12]

Rule 13 (transitivity of $\gg$), along with some application of modus ponens, gives us

$$X_k(\mathcal{R}) \Leftrightarrow ((\mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge (\mathcal{P}_i \gg \mathcal{Q}_j) \wedge \bigwedge_{\mathcal{W}_l \in A} (\mathcal{W}_l \gg \mathcal{Q}_j)) \vee \mathbb{S}$$

Which we can simplify to

$$X_k(\mathcal{R}) \Leftrightarrow ((\mathbb{P}_c \wedge X_j(\mathcal{Q}) \wedge \bigwedge_{\mathcal{W}_l \in A \cup \{\mathcal{P}_i\}} (\mathcal{W}_l \gg \mathcal{Q}_j)) \vee \mathbb{S}$$

[10] $\mathbb{P} \Leftrightarrow (\mathcal{P}_2 \gg \mathcal{Q}_2) \wedge (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge (\mathcal{Q}_2 \gg \mathcal{S}_2) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0)$
[11] $\mathbb{R} \Leftrightarrow (\mathcal{P}_2 \gg \mathcal{R}_0) \wedge (\mathcal{Q}_2 \gg \mathcal{T}_0) \wedge (\mathcal{S}_2 \gg \mathcal{U}_2)$
[12] $\mathbb{S} \Leftrightarrow (\mathbb{P}_d \wedge \bigwedge_{\mathcal{W}_l \in A} \mathcal{W}_l \gg \mathcal{P}_i) \vee \mathbb{R}$