Generating and Simplifying Sentences

Shashi Narayan

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Generating and Simplifying Sentences

THÈSE

présentée et soutenue publiquement le 7 Novembre 2014

pour l'obtention du

Doctorat de l'Université de Lorraine

(Mention Informatique)

par

Shashi Narayan

Composition du jury

Rapporteurs :	Anja Belz	Reader, University of Brighton, UK
	Mark Steedman	Professeur, University of Edinburgh, UK
Examinateurs :	Katja Filippova	Chercheur, Google, Zurich, Switzerland
	Claire Gardent	Directrice de Recherches CNRS, LORIA, Nancy, France
	Guy Perrier	Professeur, Université de Lorraine, Nancy, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Acknowledgements

This thesis would not be complete without thanking all the people who contributed directly or indirectly to it.

Most of all, I would like to thank my adviser and mentor, Claire. She happens to be one of the most passionate researchers and a wonderfully down-to-earth person. I was very fortunate to have her as my supervisor. You get the space while working with her, but at the same time, because of her presence, you feel very motivated and determined. I am very thankful to her for listening to my blobby ideas. Thanks for all interesting yoga classes. I have learned so much from her in past few years. And I hope she will always be there to guide me.

Countless thanks to my jury members, Anja Belz, Mark Steedman, Guy Perrier and Katja Filippova for showing interest in my research work. I am very thankful to them for reviewing my thesis and for providing me with valuable feedback.

Un merci spécial à Alex, Emmanuel et Christophe pour aider moi avec le résumé en français de ma thèse.

I had a wonderful time in Nancy. It would not have been possible without my friends and colleagues at LORIA. Special thanks to my office mates, Alejandra, Laura and Lina. Ale was always there to help with anything. With Laura, I had numerous stimulating discussions. Lina, she was always full of life. On a personal note, they are very sweet and kind-hearted persons. I am really going to miss our coffee discussions with Ale, Lina and Emmanuel. Merci beaucoup Emmanuel pour moi encourager à parler français. Thanks to Bikash for some abnormal laughs and to Ingrid for her wonderful cakes. Thanks to Mehwish and Sandro for helping me through both good and bad times.

मैं अपने परिवार के निःस्वार्थ प्रेम का काफी कृतज्ञ हूँ। मैं जहाँ भी था, माँ और पापा, दोनों सदा सहृदय मेरे साथ थे। रश्मि (मेरी छोटी बहन), तुम पर मुझे काफी गर्व है। तुम्हारी हरेक सफलता मुझे आनन्द देती है। साथ ही, मैं अपनी बड़ी बहन सुधा के अकथ सहयोग और प्रोत्साहन का काफी आभारी हूँ। Jag vill tacka min fru, Karna, för hennes kärlek, för att hon stärker min tro och gör mig till en bättre människa.

Résumé

Génération et Simplification des Phrases

Shashi Narayan

Superviseur: Claire Gardent

Mots-clés: Génération de la Langue Naturelle, Génération de MR à Texte, Génération de Texte à Texte, Réalisation de Surface, Grammaire d'Arbres Adjoints, Formalisme Grammatical, Extraction d'Erreur, Simplification de Phrase

Selon la représentation d'entrée, la génération de la langue naturelle peut être classée en deux catégories: la génération de texte à partir de représentation de sens (MR, *meaning representation*) et la génération de texte à partir de texte. Cette thèse étudie ces deux types d'entrées et est en conséquence divisée en deux parties: la première partie, la génération de texte à partir des représentation de sens ou "Génération des phrases", se concentre sur la tâche de génération du texte en langue naturelle à partir d'arbres de dépendance peu profondes. La seconde partie, la génération de texte à partir de texte ou "Simplification des phrases" tente de générer des phrases simples à partir de phrases complexes.

Génération des Phrases. Par comparaison avec les réalisateurs de surface statistiques, les réalisateurs de surface symboliques sont généralement plus fragiles et/ou inefficaces. Dans cette thèse, nous étudions comment effectuer la réalisation de surface symbolique à l'aide d'une grammaire robuste et efficace. Cette approche symbolique s'appuie sur une grammaire d'arbres adjoints basés sur les traits et prend en entrée des structures peu profondes décrites dans le format standard des arbres de dépendance de la tâche de réalisation de surface du Generation Challenge (The Surface Realisation Shared Task). Notre algorithme combine les techniques et les idées des approches guidées par la tête (head-driven) et lexicalistes. En outre, la structure d'entrée est utilisée pour filtrer l'espace de recherche initial à l'aide d'un concept *de filtrage local par polarité* afin de paralléliser les processus. Nous montrons que notre algorithme réduit considérablement le temps de génération comparé à une approche lexicaliste de référence qui explore l'ensemble de l'espace de recherche. Afin d'améliorer davantage la couverture, nous proposons deux algorithmes de fouille d'erreur: le premier, un algorithme qui exploite les arbres de dépendance plutôt que des données séquentielles et le second, un algorithme qui structure la sortie de la fouille d'erreur au sein d'un arbre (appelé l'arbre de suspicion) afin de représenter les erreurs de façon plus pertinente. Nous utilisons ces algorithmes de fouille d'erreur pour identifier les problèmes de notre système de génération. Nous montrons que nos réalisateurs combinés à ces algorithmes de fouille d'erreur améliorent leur couverture significativement. A l'issue de la thèse, nous nous concentrons sur la génération d'un phénomène linguistique plus complexe tel que la coordination elliptique. Nous étendons le réalisateur afin de considérer différents types d'ellipses telles que le phénomène de gapping, le partage du sujet, la montée du noeud droit, et la coordination

non-constituante.

Simplification des Phrases. D'un côté, les systèmes de simplification existants basés sur des règles écrites à la main sont limités à des règles purement syntaxiques, et de l'autre, les systèmes d'apprentissage automatique prennent en entrée une phrase ou son arbre d'analyse syntaxique. Dans cette thèse, nous proposons l'utilisation d'informations linguistiques plus riches sous la forme de représentations sémantiques profondes afin d'améliorer la tâche de simplification de phrase. Nous utilisons les structures de représentation du discours (DRS) pour la représentation sémantique profonde. Nous proposons alors deux méthodes de simplification de phrase: une première approche supervisée hybride qui combine une sémantique profonde à de la traduction automatique, et une seconde approche non-supervisée qui s'appuie sur un corpus comparable de Wikipedia. Les deux approches utilisent des modèles de simplification à l'aide des DRS afin de procéder à un découpage guidé par la sémantique ainsi qu'à des opérations de suppression. Nous montrons à la fois que notre approche supervisée est significativement en avance vis à vis des systèmes existants dans la production de phrases simples, grammaticales et conservant le sens; mais également que notre approche non-supervisée leur est comparable.

Abstract

Generating and Simplifying Sentences

Shashi Narayan

Supervisor: Claire Gardent

Keywords: Natural Language Generation, MR-to-Text Generation, Text-to-Text Generation, Surface Realisation, Tree-adjoining Grammar, Grammatical Formalism, Error Mining, Sentence Simplification

Depending on the input representation, Natural Language Generation can be categorized into two classes: MR-to-text (meaning representation to text) generation and text-to-text generation. This dissertation investigates issues from both classes. Accordingly, this dissertation is divided into two parts: the first part (MR-to-text generation, "Generating Sentences") circles around a task of generating natural language text from shallow dependency trees, and the second part (text-to-text generation, "Simplifying Sentences") tries to generate simple sentence(s) given a complex sentence.

Generating Sentences. In comparison with statistical surface realisers, symbolic surface realisers are usually brittle and/or inefficient. In this thesis, we investigate how to make symbolic grammar based surface realisation robust and efficient. We propose an efficient symbolic approach to surface realisation using a Feature-based Tree-Adjoining grammar and taking as input shallow structures provided in the format of dependency trees from the Surface Realisation Shared Task. Our algorithm combines techniques and ideas from the head-driven and lexicalist approaches. In addition, the input structure is used to filter the initial search space using a concept called *local polarity filtering*; and to parallelise

processes. We show that our algorithm drastically reduces generation times compared to a baseline lexicalist approach which explores the whole search space. To further improve our robustness, we propose two error mining algorithms: one, an algorithm for mining dependency trees rather than sequential data and two, an algorithm that structures the output of error mining into a tree (called, suspicion tree) to represent them in a more meaningful way. We use these error mining algorithms to identify problems in our generation system. We show that our realisers together with these error mining algorithms improves its coverage by a wide margin. At the end, we focus on generating a more complex linguistic phenomenon such as elliptic coordination. We extend our realiser to represent and generate different kinds of ellipsis such as gapping, subject sharing, right node raising and non-constituent coordination.

Simplifying Sentences. Earlier handcrafted rule-based simplification systems are limited to purely syntactic rules and machine learning systems either starts from the input sentence or its phrasal parse tree. In this thesis, we argue for using rich linguistic information in the form of deep semantic representations to improve the sentence simplification task. We use the Discourse Representation Structures (DRS) for the deep semantic representation of the input. We propose two methods for sentence simplification: a supervised approach to hybrid simplification using deep semantics and statistical machine translation, and an unsupervised approach to sentence simplification using the comparable Wikipedia corpus. Both approaches use DRS simplification models to do semantically governed splitting and deletion operations. We show that while our supervised approach is significantly ahead of existing state-of-the-art systems in producing simple, grammatical and meaning preserving sentences; our unsupervised approach is also competitive with them.

Je dédie ma thèse à la magnifique temps passé à Nancy.

Contents

Génération et Simplification des Phrases

Sommaire

La génération de la langue naturelle (NLG) vise à produire des textes dans les langues humaines en répondant à deux questions "Que dire?" (sélection de contenu) et "Comment le dire?" (réalisation de contenu) [Reiter and Dale, 2000]. Dans ce grand cadre du NLG, la réalisation de surface apparaît comme la dernière composante de la chaîne de traitement (sous-composante de la réalisation de contenu). Son objectif principal est de représenter le sens porté par les phrases sous la forme d'expressions en langage naturel. En partant d'une représentation donnée du sens de la phrase en entrée, la réalisation de surface essaie alors de générer une phrase grammaticalement correcte et qui a le même sens.

En fonction du type de représentation en entrée, la génération peut être classée en deux catégories: la génération de texte à partir de représentation de sens, ou la génération à partir de texte. La génération de texte à partir de représentation de sens vise à générer des phrases à partir de représentations plus ou moins non-linguistiques. Certaines approches de cette catégorie s'appuient ainsi sur des bases de données ou des bases de connaissances (par exemple, des ontologies) en entré en tandis que d'autres exploitent des représentations sémantiques plus ou moins profondes (par exemple, les F-structures, les formules de logique du premier ordre, les formules de récurrence sémantique minimales, les formules de logiques de description, des structures de représentation du discours, des arbres de dépendance). Au contraire, la tâche de génération à partir de texte, comme la simplification de la phrase, la compression de la phrase, le résumé de texte ou la génération de paraphrases, transforme un texte en langue naturelle en un autre texte en langage naturel.

Bien que ces deux catégories d'approches de NLG aient été déjà largement étudiées, il reste encore beaucoup à faire. Cette thèse aborde les deux aspects de génération de texte à partir dereprésentation de sens et de génération à partir de texte. Dans la première partie (Section F.1), nous nous concentrons sur la génération de texte à partir de représentation de sens, en optimisant à grande échelle des approches de génération de surface symboliques utilisant une grammaire d'arbres adjoints basée sur les traits (Feature-based Tree-adjoining Grammar) et prenant en entrée des structures peu profondes représentées sous la forme d'arbres de dépendance, dans le cadre d'une campagne d'évaluation de génération de surface (Surface Realisation Shared Task). La deuxième partie de cette thèse (Section F.2) concerne l'utilisation d'informations linguistiques riches en entrée, sous la forme de représentations sémantiques profondes, pour améliorer la tâche de simplification des phrases.

1 Génération les Phrases

La plupart des approches actuelles de la réalisation de surface (RS) sont statistiques [Langkilde and Knight, 1998; Ratnaparkhi, 2000; Bangalore and Rambow, 2000a; Bangalore and Rambow, 2000b; Langkilde, 2002; Nakanishi and Miyao, 2005; Zhong and Stent, 2005; Cahill and Van Genabith, 2006; White and Rajkumar, 2009; Konstas and Lapata, 2012a; Konstas and Lapata, 2012b]. Les approches symboliques sont quant à elles généralement fragiles (faible couverture) et/ou inefficaces (elles sont soit lentes ou ne parviennent pas à retourner un résultat en temps raisonnable sur des phrases plus longues) [Shieber *et al.*, 1990; Elhadad, 1993a; Elhadad and Robin, 1996; Bateman, 1997; Lavoie and Rambow, 1997; Butt *et al.*, 2002; White, 2004;

Carroll and Oepen, 2005; Gardent and Kow, 2007b; Gardent and Perez-Beltrachini, 2010]. Cependant la RS fondée sur des grammaires écrites à la main (i) est une approche transparente, c'est à dire qu'il existe une relation directe entre la grammaire et les changements dans le résultat; (ii) est linguistiquement riche, c'est à dire qu'elle associe des phrases générées avec des informations linguistiques détaillées, tels que les arbres de dérivation et dérivés, les traits morphosyntaxiques, ainsi que de l'information syntaxique (comme les règles de grammaire utilisées pour le sujet, l'objet, le verbe transitif etc.); et (iii) est utile pour les applications centrées sur la précision, par exemple l'apprentissage de la langue [Gardent and Perez-Beltrachini, 2012; Perez-Beltrachini *et al.*, 2012; Perez-Beltrachini, 2013] ou la génération de requêtes fondées sur la connaissance [Perez-Beltrachini *et al.*, 2014].

Dans cette thèse, nous explorons comment faire la réalisation de surface basée sur une grammaire symbolique de manière robuste et efficace. Dans la section F.1.1, nous introduisons nos ressources, la grammaire et les arbres de dépendance en entrée. Ensuite, nous proposons une approche symbolique, efficace et à grande échelle de la réalisation de surface qui s'appuie sur une grammaire d'arbres adjoints basée sur les traits et qui prend en entrée des structures peu profondes fournies dans le format des arbres de dépendance (Section F.1.2). Afin d'améliorer la couverture, nous proposons également deux algorithmes d'extraction d'erreur pour identifier les erreurs dans notre système de génération (Section F.1.3). Enfin, nous étendons notre système de génération afin de représenter et de générer un phénomène linguistique plus complexe comme la coordination elliptique (Section F.1.4).

1.1 Ressources: Grammaire et Représentation d'Entrée

La grammaire. La grammaire est une composante essentielle de la réalisation de surface, indépendamment du fait qu'elle soit statistique ou symbolique. Les approches statistiques, combinent l'utilisation de règles écrites à la main avec un modèle de langage (Nitrogen [Langkilde and Knight, 1998], [Ratnaparkhi, 2000] et HALogen [Langkilde, 2002]), utilisent des grammaires probabilistes (HPSG [Nakanishi and Miyao, 2005], TAG [Bangalore and Rambow, 2000a; Bangalore and Rambow, 2000b], CCG [White and Rajkumar, 2009] et CFG [Cahill and Van Genabith, 2006; Konstas and Lapata, 2012a; Konstas and Lapata, 2012b]), ou apprennent une grammaire de génération automatiquement en s'aidant d'un corpus [Zhong and Stent, 2005]. De même, les approches symboliques utilisent des grammaires symboliques orientées génération (FUF/SURGE [Elhadad, 1993a; Elhadad and Robin, 1996], KPML [Bateman, 1997] et RealPro [Lavoie and Rambow, 1997]) ou s'appuient sur des grammaires génériques réversibles (TAG [Koller and Striegnitz, 2002; Gardent and Kow,

2007b; Gardent and Perez-Beltrachini, 2010], CCG [White, 2004], HPSG [Carroll and Oepen, 2005] et LFG [Butt *et al.*, 2002]).

En suivant les approches de Koller and Striegnitz (2002), Gardent and Kow (2007b) et Gardent and Perez-Beltrachini (2010), notre algorithme de réalisation de surface repose sur une grammaire d'arbres adjoints lexicalisés à base de traits écrite à la main (FB-LTAG, [Vijay-Shanker and Joshi, 1988; Vijay-Shanker and Joshi, 1991]). Pour être plus précis, Koller and Striegnitz (2002), Gardent and Perez-Beltrachini (2010) et Gardent *et al.* (2011) ont démontré que dans une TAG, le traitement des arbres de dérivation plutôt que des arbres dérivés est plus efficace. Ainsi, en suivant Gardent and Perez-Beltrachini (2010) et Gardent and Perez-Beltrachini (2010), nous n'utilisons donc pas directement la grammaire FB-LTAG, mais plutôt la grammaire des arbres réguliers basée sur les traits (FB-RTG, [Schmitz and Roux, 2008]) qui elle, s'appuie sur les arbres de dérivation. Autrement dit, l'algorithme de réalisation de surface construit tout d'abord un arbre de dérivation puis s'appuie sur la grammaire FB-LTAG pour construire l'arbre dérivé et obtenir enfin la phrase à générer.

Nous utilisons une grammaire FB-LTAG à large couverture [Alahverdzhieva, 2008] pour l'anglais composée d'environ 1000 arbres et dont la couverture est similaire à XTAG [The XTAG Research Group, 2001]. Nous convertissons ensuite cette grammaire FB-LTAG vers une grammaire FB-RTG en utilisant la méthode détaillée dans [Schmitz and Roux, 2008]. Des descriptions plus détaillées sur notre grammaire FB-LTAG et sur la grammaire FB-RTG correspondante peuvent être trouvées dans le chapitre 2.

La représentation d'entrée. A cause de leur dépendance à différents formalismes grammaticaux, et bien qu'ils apprennent leur grammaire de la même source (le Penn Treebank), les réalisateurs de surface récents s'appuient tous sur des formats différents de représentation du sens. En fonction de la proximité avec la phrase de surface, certaines approches [Gardent and Kow, 2007b; Gardent and Perez-Beltrachini, 2010; Callaway, 2003; Carroll and Oepen, 2005; White, 2004; Basile and Bos, 2013] prennent en entrée des représentations sémantiques profondes (F-structures, des formules logiques du premier ordre, de la logique de description, des structures de représentation du discours) tandis que d'autres [Lavoie and Rambow, 1997; Bangalore and Rambow, 2000a; Bangalore and Rambow, 2000b; Narayan, 2011] s'appuient sur des structures peu profondes, comme par exemple, des structures de dépendance. Ces disparités créent un défi majeur pour l'évaluation des réalisateurs de surface à large échelle.

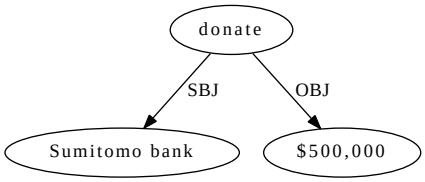
Afin de surmonter le problème décrit ci-dessus et de fournir une représentation d'entrée commune à un large éventail de réalisateurs de surface, une Tâche Commune

de Réalisation de Surface a été organisée Belz *et al.* (2011) - (Surface Realisation Shared Task), et les structures de dépendance ont été choisies comme format commun d'entrée. Les organisateurs ont extrait deux types de représentation du Penn Treebank: des arbres de dépendances peu profondes (avec étiquettes syntaxiques), et des graphes de dépendances profondes (avec étiquettes sémantiques). Dans les deux cas, les relations ont été arbitrairement ordonnées. La représentation peu profonde est une vue plus syntaxique de la phrase, tandis que la représentation profonde est plus proche de la représentation sémantique de la phrase correspondante.

Dans cette thèse, notre générateur s'appuie sur les arbres de dépendances peu profondes de la Tâche Commune de Réalisation de Surface [Belz *et al.*, 2011] comme représentation du sens. Nous décrivons l'ensemble de données plus en détail dans le chapitre 2. L'exemple (1) illustre une entrée de cette tâche commune ainsi qu'une sortie de la tâche de génération de texte à partir de représentation de sens. L'entrée est un arbre de dépendances peu profondes non-ordonnées (1**L'entrée**) et le résultat est une phrase qui verbalise cette entrée (1**Le résultat**).

(1) **Input:** Shallow dependency structure

```
SROOT 1 0 donate cpos=vbd
SBJ 2 1 Sumitomo bank cpos=nn
OBJ 3 1 $500,000 cpos=cd
```



Output: Sumitomo bank donated $500,000.

1.2 Optimiser la Réalisation de Surface

La réalisation de surface symbolique est très sensible au problème combinatoire. Chaque réalisateur de surface tente de faire face à ce problème d'une façon ou d'une autre. Selon le type de représentation du sens encodée par la grammaire, deux principaux types d'algorithmes ont été proposés pour générer des phrases avec des grammaires symboliques: les approches guidées par la tête (head-driven) et les approches lexicalistes. Pour les représentations sémantiques récursives telles que celles de formules logiques de premier ordre, les algorithmes guidés par la tête [Shieber *et al.*, 1990] fonctionnent mieux en évitant le problème de combinatoire grâce à des éléments lexicaux pour guider la recherche. D'autre part, pour les représentations sémantiques plates comme les MRS (la sémantique de récurrence minimale, [Copestake *et al.*, 2001]), les approches lexicalistes [Espinosa *et al.*, 2010; Carroll and Oepen, 2005; Gardent and Kow, 2005] ont été largement utilisées car elles imposent moins de contraintes sur la grammaire.

Nous proposons un algorithme [Narayan and Gardent, 2012b] qui combine les techniques et les idées des approches guidées par la tête et des approches lexicalistes. D'une part, la sélection de la règle est guidée, comme dans l'approche lexicaliste, par les unités élémentaires présentes dans l'entrée plutôt que par sa structure. D'autre part, la structure de l'entrée est utilisée pour fournir des indices descendants (top-down) pour la recherche et ainsi limiter la combinatoire. Pour améliorer encore l'efficacité, l'algorithme intègre trois techniques d'optimisation supplémentaires: (i) le filtrage par polarité de l'approche lexicaliste [Bonfante *et al.*, 2004; Gardent and Kow, 2007b]; (ii) l'utilisation d'un modèle de langage pour sélectionner les structures intermédiaires en compétition; et (iii) l'utilisation simultanée plutôt que séquentielle de prédictions descendantes (top-down) parallélisées.

Nous avons évalué notre algorithme sur les arbres de dépendances de la tâche partagée de réalisation de surface à grande échelle et avons montré que, sur ces données, l'algorithme réduit considérablement le temps d'exécution en comparaison à une approche lexicaliste simple qui explore l'ensemble de l'espace de recherche. Une description plus détaillée de cet algorithme et de son évaluation pourra être trouvée dans le chapitre 3.

1.3 Fouille d'Erreurs pour l'Amélioration de la Génération Symbolique

Au cours des dernières années, les méthodes de fouille d'erreurs [van Noord, 2004; Sagot and de la Clergerie, 2006; de Kok *et al.*, 2009] ont été développées afin d'identifier les sources les plus probables de défaillance d'analyse syntaxique dans les analyseurs qui s'appuient sur des grammaires écrites à la main et des lexiques. Cependant, les techniques utilisées pour énumérer et compter les n-grammes reposent sur la nature séquentielle d'un corpus textuel et ne s'étendent pas facilement à des données structurées. En outre, ces méthodes classiques génèrent une simple liste des formes suspectes classées par ordre décroissant de suspicion. Il n'y a aucune vue claire de la façon dont les différentes formes suspectes interagissent et, par conséquent, le linguiste doit analyser tous les cas d'erreur un par un au lieu de se concentrer sur l'amélioration des ensembles de cas d'erreur associés de façon linguistiquement pertinente.

Pour améliorer notre couverture, nous proposons deux algorithmes de fouille d'erreurs [Gardent and Narayan, 2012; Narayan and Gardent, 2012a] dans le chapitre 4: premièrement, un algorithme pour l'exploitation des arbres que nous appliquons pour détecter les sources les plus probables de défaillance de la génération et deuxièmement, un algorithme qui structure le résultat de la fouille d'erreurs dans un arbre

(appelé, arbre de suspicion), mettant en évidence les relations entre les formes suspectes. Nous montrons que ces algorithmes de fouille d'erreurs ont permis d'identifier non seulement les erreurs dans le système de génération (la grammaire et le lexique), mais aussi l'inadéquation entre les structures contenues dans l'entrée et les structures d'entrée attendues par notre générateur ainsi que quelques particularités/erreurs dans les données en entrée. Nous illustrons comment l'arbre de suspicion construit par notre algorithme permet de présenter les erreurs identifiées par la fouille d'erreurs de manière linguistiquement pertinente offrant ainsi un meilleur support pour l'analyse d'erreurs.

1.4 Génération de le Coordination Elliptique

Notre générateur symbolique accompagné de techniques de fouille d'erreurs conduit au développement en spirale de notre système de génération. Nous améliorons largement à la fois la couverture et les scores BLEU . L'amélioration globale de notre système de génération nous permet de ramener notre attention sur les subtilités liées à la représentation et la génération d'un phénomène linguistique plus complexe comme la coordination elliptique.

Dans les phrases elliptiques, il y a un sens sans le son. Ainsi, les mises en correspondances de forme/sens qui nous permettent, dans les phrases non-elliptiques d'associer les sons aux sens correspondants, ne fonctionnent plus. Nous extrayons, à partir des données fournies par la tâche de réalisation de surface [Belz *et al.*, 2011], 2398 entrées dont la phrase résultat correspondante contient une ellipse. Nous montrons qu'une quantité considérable de données contient une ellipse et que la couverture et le score BLEU diminuent sensiblement pour ces types d'entrées elliptiques. Nous soutenons le fait que le matériel omis devrait être représenté en utilisant des noeuds phonétiquement vides et nous introduisons alors un ensemble de règles de réécriture qui permet l'ajout de ces catégories vides à la représentation de l'arbre des dépendances. Enfin, nous évaluons notre réalisateur de surface sur l'ensemble de données résultant. Nous montrons que, après réécriture, le générateur améliore à la fois la couverture et le score BLEU pour les données elliptiques. Une description plus détaillée de notre approche [Gardent and Narayan, 2013] pourra être trouvée dans le chapitre 5.

2 Simplification des Phrases

La deuxième partie de cette thèse porte sur la génération de texte à partir de texte en se focalisant sur la simplification de phrase. La simplification de la phrase associe

une phrase complexe à une autre phrase plus simple, plus facile à lire et dont le contenu s'approche de la phrase d'origine. Par exemple, si nous considérons la phrase complexe (2**Complexe**) comme entrée et nous pouvons générer la version simplifiée présentée dans l'exemple (2**Simple**).

(2) **Complexe:** In 1964 Peter Higgs published his second paper in Physical Review Letters describing Higgs mechanism which predicted a new massive spin-zero boson for the first time.
Simple: Peter Higgs wrote his paper explaining Higgs mechanism in 1964. Higgs mechanism predicted a new elementary particle.

Les travaux précédents sur la simplification de phrase s'appuyaient sur des règles écrites à la main pour capturer la simplification syntaxique [Chandrasekar and Srinivas, 1997; Siddharthan, 2002; Canning, 2002; Siddharthan, 2006; Siddharthan, 2010; Siddharthan, 2011; Bott *et al.*, 2012; Siddharthan and Mandya, 2014]. Des approches plus récentes, cependant, utilisent un ensemble de données parallèles formées par l'alignement de phrases provenant de Wikipedia en anglais simple (Simple English Wikipedia) et Wikipedia en anglais traditionnel (Traditional English Wikipedia) [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Coster and Kauchak, 2011; Wubben *et al.*, 2012].

A notre connaissance, toutes les approches d'apprentissage automatique existantes (i) commencent à partir de la phrase d'entrée [Coster and Kauchak, 2011; Wubben *et al.*, 2012] ou de son arbre syntaxique [Zhu *et al.*, 2010; Woodsend and Lapata, 2011] et (ii) apprennent sur le corpus aligné de phrases simples et complexes. En comparaison, nous contribuons à la simplification de la phrase de deux façons. La première est que nous nous concentrons sur l'utilisation de l'information linguistique riche sous la forme d'une représentation sémantique profonde afin d'améliorer la tâche de simplification de la phrase. Nous utilisons les structures de représentation du discours [Kamp, 1981] définies par Boxer [Curran *et al.*, 2007] pour les représentations sémantiques profondes. La seconde est que nous proposons deux méthodes pour la simplification de la phrase : une approche supervisée pour la simplification hybride utilisant la sémantique profonde et la traduction automatique (Section F.2.1); ainsi qu'une approche non-supervisée pour la simplification de la phrase utilisant le corpus de Wikipedia comparable (Section F.2.2).

2.1 Simplification Hybride utilisant la Sémantique Profonde et la Traduction Automatique

D'un côté les approches d'apprentissage automatique basées sur les arbres syntaxiques [Zhu *et al.*, 2010; Woodsend and Lapata, 2011] sont confrontées aux prob-

lèmes de l'identification de la frontière de division, de la reconstruction de l'élément commun dans les phrases divisées tout en évitant de supprimer des arguments obligatoires. De l'autre côté, les approches d'apprentissage automatique basées sur la traduction automatique monolingue avec des phrases d'entrée en tant que source [Coster and Kauchak, 2011; Wubben *et al.*, 2012] n'atteignent pas une quantité significative de suppressions ou de divisions.

Notre approche supervisée [Narayan and Gardent, 2014] est entraînée sur l'ensemble de données parallèles alignées constitué de "Simple English Wikipedia" et de "Traditional English Wikipedia". Notre approche diffère des approches précédentes de deux façons. Premièrement, c'est une approche hybride qui combine un modèle qui encode les probabilités pour le découpage et la suppression avec un module de traduction automatique monolingue qui contrôle la réorganisation et la substitution. De cette façon, nous exploitons la capacité des systèmes de traduction automatique statistique (Statistical Machine Translation, SMT) à capturer à la fois la substitution lexicale et de phrases ainsi que le réarrangement, et ce tout en s'appuyant sur un module probabiliste afin de saisir les opérations de division et de suppression qui sont moins bien (la suppression) ou pas du tout (la division) capturées par les approches SMT. Deuxièmement, les probabilités de division et de suppression sont apprises en utilisant la représentation sémantique profonde des phrases complexes. Cette stratégie permet de motiver linguistiquement l'opération de découpage en ce que les éléments sémantiques partagés sont considérés comme les bases du découpage d'une phrase complexe en phrases plus simples. Cette stratégie facilite la complétion (la re-création de l'élément partagé dans les phrases découpées). En outre elle ofre un moyen naturel pour éviter la suppression des arguments obligatoires. A l'aune des méthodes supervisées actuelles [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Wubben *et al.*, 2012], notre modèle fournit une sortie beaucoup plus simple qui préserve à la fois la syntaxe et le sens . Le lecteur est invité à consulter le chapitre 7 pour une information détaillée sur notre approche de simplification supervisée de la phrase et de son évaluation complète.

2.2 Simplification Non-Supervisée de la Phrase utilisant Wikipedia

Construire un corpus aligné de phrases simples et complexes de bonne qualité pour des méthodes supervisées de simplification n'est pas une tâche facile, mais cette opération affecte aisément les performances des systèmes créés à partir de ceux-ci. En fait, Woodsend and Lapata (2011) débattent sur ce sujet et tentent d'améliorer les performances grâce à l'historique d'édition de "Simple Wikipedia". Nous avons en outre constaté qu'à cause du corpus utilisé pour l'apprentissage, notre approche

supervisée n'est pas aussi performante que la méthode de référence en ce qui concerne le nombre de division.

Nous présentons une approche non-supervisée de simplification de phrase qui ne nécessite ni règles écrites à la main, ni corpus d'entraînement de phrases complexes et simplifiées alignées. Au lieu de cela, nous exploitons "Simple English Wikipedia" et "Traditional English Wikipedia" non-alignés pour apprendre la probabilité des simplifications lexicales, de la sémantique des phrases simples et des phrases optionnelles, c'est à dire, des phrases qui peuvent être supprimées lors de la simplification. Encore une fois, le découpage des phrase est basé sur la sémantique dans le sens où il opère sur la structure sémantique profonde. Nous montrons (i) que le framework non supervisé que nous proposons est compétitif par rapport aux systèmes supervisés de référence et (ii) que notre approche basée sur la sémantique permet une manipulation du découpage de la phrase à la fois motivée linguistiquement et efficace. Le lecteur est invité à consulter le chapitre 8 pour une information complète sur notre approche de simplification non-supervisée de la phrase et son évaluation complète.

3 Résumé

Cette thèse se concentre autour de deux questions principales: l'une, comment faire la réalisation de surface de manière robuste et efficace à l'aide de grammaires symboliques, et l'autre, comment exploiter une information linguistique riche sous la forme d'une représentation sémantique profonde afin d'améliorer la simplification de phrases.

3.1 Contributions Principales

Dans ce qui suit, nous résumons les contributions principales de cette thèse :

Amélioration de l'efficacité de la RS symbolique. Nous présentons un nouvel algorithme efficace pour la réalisation de surface avec FB-LTAG qui prend en entrée des structures peu profondes fournies dans le format des arbres de dépendance. Lors de l'évaluation sur une grande quantité de données, nous avons montré que notre algorithme réduit considérablement les temps de génération en comparaison à une approche lexicaliste de référence qui explore l'ensemble de l'espace de recherche.

Amélioration de la couverture de la RS symbolique. Nous avons proposé deux nouveaux algorithmes de fouille d'erreur afin d'identifier des erreurs dans les réalisateurs de surface symboliques et par conséquent d'améliorer la robustesse

de notre réalisateur. Notre premier algorithme de fouille d'erreur nous permet efficacement d'identifier les arbres suspects. Notre deuxième algorithme permet de structurer le résultat de la fouille d'erreur dans un arbre de suspicion, ce qui constitue une analyse d'erreur ayant du sens d'un point de vue linguistique. A l'aide de ces algorithmes de fouille d'erreur, nous améliorons largement la couverture de notre réalisateur. Sur les données de test, nous avons atteint une couverture de 81 % avec un score BLEU de 0,72 pour les phrases générées.

Génération et développement du corpus de coordination elliptique. Nous avons montré que l'introduction de noeuds vides pour des phrases elliptiques améliore à la fois la couverture et la précision de notre réalisateur de surface. Pour évaluer notre réalisateur sur la capacité de générer de la coordination elliptique, nous avons recueilli 2398 représentations potentielles en entrée à partir des données de test. Sur cet ensemble de données, notre réalisateur de surface atteint une couverture de 76 % et un score BLEU de 0,74 sur les phrases générées. Nos données focalisées sur l'ellipse contiennent 384 cas potentiels de montée du noeud droit (Right Node Raising, RNR), 1462 cas potentiels de partage de sujet (Subject Sharing, SS), 456 cas potentiels de SS+RNR, 36 cas de gapping et 60 cas de coordinations non-constituantes (Non-Constituent Coordination, NCC). Nous estimons que cet ensemble de données pourrait être très utile pour d'autres générateurs afin de tester leurs propres capacités à générer des ellipses.
.

Sémantiques profondes pour la simplification des phrases. Nous avons proposé d'utiliser une information linguistique riche sous la forme de représentations sémantiques profondes afin d'améliorer la tâche de simplification des phrases. Nous utilisons la représentation DRS [Kamp, 1981] définie par Boxer [Curran *et al.*, 2007] pour les représentations sémantiques profondes. Nous avons alors proposé deux nouveaux algorithmes pour la simplification des phrases: l'un supervisé et l'autre non-supervisé. Les deux algorithmes utilisent la représentation sémantique profonde pour apprendre comment diviser une phrase complexe en plusieurs phrases simples et comment supprimer des modificateurs non-nécessaires dans une phrase sans modifier sensiblement son sens. Nous montrons que les modèles appris sur la représentation sémantique profonde facilitent la complétion (la re-création de l'élément partagé dans les phrases divisées) et fournissent un moyen naturel afin d'éviter la suppression d'arguments obligatoires.

Approche supervisée hybride pour la simplification. Nous avons proposé une approche hybride pour la simplification des phrases qui combine un modèle de simplification à base de DRS pour le découpage avec une méthode de suppression fondée sur un système de traduction automatique monolingue pour la substitution et le réordonnancement. Nous avons entraîné et évalué notre système à l'aide des corpora PWKP. Nous avons comparé notre système par rapport à trois méthodes de référence [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Wubben *et al.*, 2012] et nous avons montré que notre approche produit une sortie beaucoup plus simple et qui préserve à la fois la syntaxe et le sens.

Corpus comparables pour la simplification des phrases. En plus de notre approche supervisée, nous avons proposé une nouvelle approche de la simplification des phrases qui ne nécessite pas de corpus de phrases simples et complexes alignées. Notre approche n'est pas supervisée dans le sens où elle ne requiert qu'un large corpus composé d'un langage standard et d'un autre simplifié, mais pas d'alignement entre les deux. Les probabilités de simplification lexicale sont apprises par l'analyse des mots avec leur contexte à la fois dans "Wikipedia anglais simple" et " Wikipedia anglais traditionnel", tandis que les probabilités de la division et de la suppression sont apprises par l'analyse des structures fréquentes dans la représentation DRS avec uniquement "Wikipedia anglais simple". Nous avons évalué notre système sur le corpus d'évaluation PWKP et constaté que notre méthode est compétitive par rapport aux quatre méthodes supervisées de référence [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Wubben *et al.*, 2012; Narayan and Gardent, 2014].

3.2 Pointeurs pour de Futures Recherches

Cette thèse offre différentes orientations possibles pour de futures recherches : la génération à partir de structures de graphes (au lieu des arbres dépendances) des représentations sémantiques, la fouille d'erreur pour les suspects de surgénération, l'utilisation de corpus de coordination elliptique pour une évaluation plus approfondie, l'évaluation de la simplification des phrases pour la simplification au niveau du discours, l'exploitation d'une base de données de paraphrases pour la simplification, l'exploitation d'autres représentations pour la sémantique profonde etc. Le lecteur est invité à se reporter au chapitre 9.2 pour une liste complète des directions pour des travaux futurs.

List of Figures

List of Tables

Chapter 1

Introduction

NLG

NATURAL LANGUAGE GENERATION (NLG) aims at producing texts in human languages by solving two questions "What to say?" (content selection) and "How to say it?" (content realization) [Reiter and Dale, 2000]. In this big frame of NLG, surface realisation appears as a last component (sub-component of content realization) with the special task of mapping meaning representations to natural language expressions at sentence level. Given some meaning representation as input, surface realisation tries to generate a sentence licensed by the grammar capturing this meaning.

SURFACE REALIZATION

Depending on the input representation, NLG can be categorised into two classes: MR-to-text (meaning representation to text) generation and text-to-text generation. MR-to-text generation focuses on generating sentences from more or less non-linguistic representations. Some approaches in this class take as input databases or knowledge bases (e.g., ontologies) while others assume deep or shallow semantic representations (e.g., F-structures, First Order Logic formulae, Minimal Recursion Semantics formulae, Description Logic formulae, Discourse Representation Structures, Dependency trees). By contrast, the task of text-to-text generation, such as sentence simplification, sentence compression, text summarisation and paraphrase generation, maps natural language text to natural language text.

MR-TO-TEXT

TEXT-TO-TEXT

While both classes of NLG have already been widely studied, much remains to be done. This dissertation investigates issues from both MR-to-text generation and text-to-text generation. Accordingly, we divide this document into two parts. In the first part (MR-to-text generation, "Generating Sentences"), we focus on MR-to-text generation, optimising a large-scale symbolic approaches to surface realisation using a Feature-based Tree-Adjoining grammar and taking as input shallow structures provided in the format of dependency trees by the Surface Realisation Shared Task

[Belz *et al.*, 2011]. The second part of this dissertation (text-to-text generation, "Simplifying Sentences") focuses on the use of rich linguistic information in the form of deep semantic representation to improve the sentence simplification task.

Motivation behind "Generating Sentences"

Most current approaches to surface realisation (SR) are statistical [Langkilde and Knight, 1998; Ratnaparkhi, 2000; Bangalore and Rambow, 2000a; Bangalore and Rambow, 2000b; Langkilde, 2002; Nakanishi and Miyao, 2005; Zhong and Stent, 2005; Cahill and Van Genabith, 2006; White and Rajkumar, 2009; Konstas and Lapata, 2012a; Konstas and Lapata, 2012b]. In contrast, symbolic SR approaches are usually brittle (low coverage) and/or inefficient (they are either slow or time-out on longer sentences) [Shieber *et al.*, 1990; Elhadad, 1993a; Elhadad and Robin, 1996; Bateman, 1997; Lavoie and Rambow, 1997; Butt *et al.*, 2002; White, 2004; Carroll and Oepen, 2005; Gardent and Kow, 2007b; Gardent and Perez-Beltrachini, 2010]. However, hand-written grammar based SR (i) is a glass-box approach, i.e., there is a direct relation between the grammar and output changes; (ii) is linguistically rich, i.e., it associates generated sentences with detailed linguistic information, e.g., derivation and derived trees, morphosyntactic features, and syntactic information (e.g., used grammar rules for subject, object, transitive verb etc); and (iii) is useful for precision focused applications, e.g., language learning [Gardent and Perez-Beltrachini, 2012; Perez-Beltrachini *et al.*, 2012; Perez-Beltrachini, 2013] or knowledge-based query generation [Perez-Beltrachini *et al.*, 2014].

In this thesis, we explore how to make symbolic grammar based surface realisation robust and efficient. We propose an efficient and large scale symbolic approach to surface realisation using a Feature-based Tree-Adjoining grammar and taking as input shallow structures provided in the format of dependency trees. To improve coverage, we further propose two error mining algorithms to identify errors in our generation system. Finally, we extend our generation system for representing and generating more complex linguistic phenomena such as elliptic coordination. In what follows, we situate and motivate our research decisions and choices.

Grammar. Grammar is an essential component of surface realization, irrespective of whether statistical or symbolic. Statistical approaches either combine the use of handwritten rules with a language model (Nitrogen [Langkilde and Knight, 1998], [Ratnaparkhi, 2000] and HALogen [Langkilde, 2002]), use probabilistic grammars (HPSG [Nakanishi and Miyao, 2005], TAG [Bangalore and Rambow, 2000a; Bangalore and Rambow, 2000b], CCG [White and Rajkumar, 2009] and CFG [Cahill

and Van Genabith, 2006; Konstas and Lapata, 2012a; Konstas and Lapata, 2012b]), or learn a generation grammar automatically from corpora [Zhong and Stent, 2005]. Similarly, symbolic approaches either use NLG-geared symbolic grammars (FUF/ SURGE [Elhadad, 1993a; Elhadad and Robin, 1996], KPML [Bateman, 1997] and RealPro [Lavoie and Rambow, 1997]) or general purpose reversible grammars (TAG [Koller and Striegnitz, 2002; Gardent and Kow, 2007b; Gardent and Perez-Beltrachini, 2010], CCG [White, 2004], HPSG [Carroll and Oepen, 2005] and LFG [Butt *et al.*, 2002]).

Following Koller and Striegnitz (2002), Gardent and Kow (2007b) and Gardent and Perez-Beltrachini (2010), our surface realisation algorithm builds on a hand-written non-probabilistic FB-LTAG [Vijay-Shanker and Joshi, 1988; Vijay-Shanker and Joshi, 1991]. We use a large scale FB-LTAG [Alahverdzhieva, 2008] for English consisting of roughly 1000 trees and whose coverage is similar to XTAG [The XTAG Research Group, 2001].

Input Representation. Despite the fact that the recent surface realisers learn their grammar from the Penn Treebank, because of their dependence on different grammar formalisms, each realiser uses different meaning representations as input. Depending on the closeness to the surface sentence, some approaches [Gardent and Kow, 2007b; Gardent and Perez-Beltrachini, 2010; Callaway, 2003; Carroll and Oepen, 2005; White, 2004; Basile and Bos, 2013] take as input deep, semantically oriented representations (F-structures, First Order Logic formulae, Description Logic formulae, Discourse Representation Structures) while others [Lavoie and Rambow, 1997; Bangalore and Rambow, 2000a; Bangalore and Rambow, 2000b; Narayan, 2011] assume more shallow structures such as, for instance, dependency structures. This creates a major challenge towards a large scale evaluation.

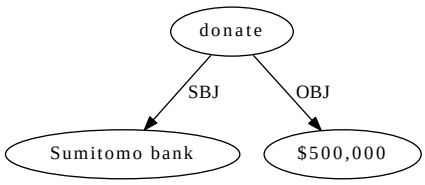
In order to overcome the above described problem and to provide a common-ground input representation for the wide range of surface realisers, Belz *et al.* (2011) organised a Surface Realisation Shared Task and chose dependency structures as their common-ground input representation. They extracted dataset from Penn Treebank with two different representations: shallow dependency trees (with syntactic labels) and deep dependency graphs (with semantic labels). For both representations, relations were arbitrarily ordered.

In this dissertation, our generator takes shallow dependency trees from Surface Realisation Shared Task [Belz *et al.*, 2011] as their meaning representation. We describe the dataset in more detail in Chapter 2. Example (3) shows an input of this shared task and an output of the MR-to-text generation task. The input is an unordered shallow dependency tree (3**Input**) and the output is a sentence verbalising

this input (3**Output**).

(3) **Input:** Shallow dependency structure

```
SROOT 1 0 donate cpos=vbd
SBJ 2 1 Sumitomo bank cpos=nn
OBJ 3 1 $500,000 cpos=cd
```



Output: Sumitomo bank donated $500,000.

Improving Efficiency. Symbolic surface realisation is very prone to the combinatorial problem [Kay, 1996]. Every surface realiser tries to deal with this bottleneck one way or another. Depending on the type of meaning representation encoded by the grammar, two main types of algorithms have been proposed for generating sentences with symbolic grammars: head-driven and lexicalist approaches. For recursive semantic representations such as first-order logic formulae, head-driven algorithms [Shieber *et al.*, 1990] have been argued to perform best in avoiding the combinatorics problem by using lexical items to guide the search. On the other hand, for flat semantic representations such as MRSs (Minimal Recursion Semantics, [Copestake *et al.*, 2001]), lexicalist approaches [Espinosa *et al.*, 2010; Carroll and Oepen, 2005; Gardent and Kow, 2005] have been used extensively because they impose fewer constraints on the grammar.

We propose an algorithm which combines techniques and ideas from the head-driven and the lexicalist approaches. On the one hand, rule selection is guided, as in the lexicalist approach, by the elementary units present in the input rather than by its structure. On the other hand, the structure of the input is used to provide top-down guidance for the search and thereby restrict the combinatorics. To further improve efficiency, the algorithm integrates three additional optimisation techniques: (i) polarity filtering from the lexicalist approach [Bonfante *et al.*, 2004; Gardent and Kow, 2007b]; (ii) the use of a language model to prune competing intermediate substructures; and (iii) simultaneous rather than sequential parallelised top-down predictions.

Improving Coverage using Error Mining. In recent years, error mining approaches [van Noord, 2004; Sagot and de la Clergerie, 2006; de Kok *et al.*, 2009] were developed to help identify the most likely sources of parsing failures in parsing systems using handcrafted grammars and lexicons. However the techniques they use to enumerate and count n-grams builds on the sequential nature of a text corpus and do not easily extend to structured data. In addition, they generate a flat list of suspicious forms ranked by decreasing order of suspicion. There is no clear overview of how the various suspicious forms interact and as a result, the linguist must analyse

all error cases one by one instead of focusing on improving sets of related error cases in a linguistically meaningful way.

To improve our coverage, we propose two error mining algorithms: one, an algorithm for mining trees which we apply to detect the most likely sources of generation failure and two, an algorithm that structures the output of error mining into a tree (called, suspicion tree), highlighting the relationships between suspicious forms. We show that these error mining algorithms permit identifying not only errors in the generation system (grammar, lexicon) but also mismatches between the structures contained in the input and the input structures expected by our generator as well as a few idiosyncrasies/error in the input data. We illustrate how the suspicion tree built by our algorithm helps to present the errors identified by error mining in a linguistically meaningful way thus providing better support for error analysis.

Generating Elliptic Coordination. Our symbolic generator together with error mining initiatives leads to the spiral development of our generation system. We improve in both coverage and BLEU scores. The global improvement of our generation system helps us to narrow down our focus towards the intricacies of representing and generating a more complex linguistic phenomenon such as elliptic coordination.

In elliptic sentences, there is meaning without sound. Thus the usual form/meaning mappings that in non-elliptic sentences allow us to map sounds onto their corresponding meanings, break down. We extract from the data provided by the Surface Realisation Task [Belz *et al.*, 2011] 2398 inputs whose corresponding output sentence contain an ellipsis. We show that a significant amount of the data contains an ellipsis and that both coverage and BLEU score markedly decrease for elliptic input. We argue that elided material should be represented using phonetically empty nodes and we introduce a set of rewrite rules which permits adding these empty categories to the dependency tree representation. Finally, we evaluate our surface realiser on the resulting dataset. We show that, after rewriting, the generator improves both on the coverage and the BLEU score for the elliptical data.

Motivation behind "Simplifying Sentences"

The second part of this dissertation focuses on text-to-text generation in particular sentence simplification. Sentence simplification maps a sentence to a simpler, more readable one approximating its content. For example, it takes the complex sentence (4**Complex**) as input and generates the simplified version shown in (4**Simple**).

(4) **Complex:** In 1964 Peter Higgs published his second paper in Physical Review Letters describing Higgs mechanism which predicted a new massive spin-zero boson for the first

time.

Simple: Peter Higgs wrote his paper explaining Higgs mechanism in 1964. Higgs mechanism predicted a new elementary particle.

Earlier work on sentence simplification relied on handcrafted rules to capture syntactic simplification [Chandrasekar and Srinivas, 1997; Siddharthan, 2002; Canning, 2002; Siddharthan, 2006; Siddharthan, 2010; Siddharthan, 2011; Bott *et al.*, 2012; Siddharthan and Mandya, 2014]. More recent approaches, however, use a parallel dataset formed by aligning sentences from Simple English Wikipedia and traditional English Wikipedia [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Coster and Kauchak, 2011; Wubben *et al.*, 2012].

To the best of our knowledge, all existing machine-learning approaches (i) starts from the input sentence [Coster and Kauchak, 2011; Wubben *et al.*, 2012] or its parse tree [Zhu *et al.*, 2010; Woodsend and Lapata, 2011] and (ii) train on the aligned corpus of complex and simple sentences. In contrast, we contribute to sentence simplification in two ways. First, we focus on using rich linguistic information in the form of deep semantic representation to improve the sentence simplification task. We use the Discourse Representation Structures [Kamp, 1981] assigned by Boxer [Curran *et al.*, 2007] for the deep semantic representations. Second, we propose two methods for sentence simplification: a supervised approach to hybrid simplification using deep semantics and machine translation; and an unsupervised approach to sentence simplification using the comparable Wikipedia corpus.

Supervised Approach. On one hand, the machine-learning approaches based on parse trees [Zhu *et al.*, 2010; Woodsend and Lapata, 2011] face the problems of identifying the split boundary, of reconstructing the shared element in split sentences and of avoiding the deletion of obligatory arguments. On the other hand, the machine-learning approaches based on monolingual machine translation with input sentences as source [Coster and Kauchak, 2011; Wubben *et al.*, 2012] do not achieve a significant amount of deletion or splitting at all.

Our supervised approach is trained on the parallel aligned dataset formed by Simple English Wikipedia and traditional English Wikipedia. It differs from previous approaches in two main ways. First, it is a hybrid approach which combines a model encoding probabilities for splitting and deletion with a monolingual machine translation module which handles reordering and substitution. In this way, we exploit the ability of statistical machine translation (SMT) systems to capture phrasal/lexical substitution and reordering while relying on a dedicated probabilistic module to capture the splitting and deletion operations which are less well (deletion) or not at all (splitting) captured by SMT approaches. Second, the splitting and

deletion probabilities are learned using the deep semantic representation of complex sentences. This permits a linguistically principled account of the splitting operation in that semantically shared elements are taken to be the basis for splitting a complex sentence into several simpler ones. This facilitates completion (the recreation of the shared element in the split sentences). And this provides a natural means to avoid deleting obligatory arguments. When compared against current state of the art supervised methods [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Wubben *et al.*, 2012], our model yields a significantly simpler output that is both grammatical and meaning preserving.

Unsupervised Approach. Constructing a good quality aligned corpus of complex and simple sentences for supervised simplification methods is not an easy task but it easily affects the performance of systems trained on them. In fact, Woodsend and Lapata (2011) debate the issue and try to improve the performance by using the edit history of Simple Wikipedia. We also found that because of the corpus used for training, our supervised approach lags behind the gold standard in terms of the number of times it splits the input sentence.

We present an unsupervised approach to sentence simplification which requires neither hand written rules nor a training corpus of aligned standard and simplified sentences. Instead, we exploit non-aligned Simple and traditional English Wikipedia to learn the probability of lexical simplifications, of the semantics of simple sentences and of optional phrases i.e., phrases which may be deleted when simplifying. Again, sentence splitting is semantically based in that it operates on deep semantic structure. We show (i) that the unsupervised framework we propose is competitive with state-of-the-art supervised systems and (ii) that our semantically based approach allows for a principled and effective handling of sentence splitting.

Roadmap of the thesis

We divide this thesis into two parts: "Generating Sentences" and "Simplifying Sentences". Each part then follows a standard format: first a background chapter and then our contribution chapters. In what follows, we summarise the content of the remaining chapters of this thesis.

Part I: Generating Sentences

Chapter 2 (Background) describes various resources related to this part such as input representations of the Generation Challenge: Surface Realisation Shared Task [Belz *et al.*, 2011] used for generation (Section 2.1), Feature-based Lexicalised

Tree-Adjoining Grammar (Section 2.2.1) and its variant Feature-based Regular Tree Grammar (Section 2.2.2). Because of the self-contained nature of research addressed in the chapters of this part, we do not discuss related works here as a whole, instead we create a section for related work in each chapter separately.

Chapter 3 (Optimising Surface Realisation) describes our efficient surface realisation algorithm. In Section 3.2, we describe key concepts used for optimization in head-driven approaches, lexicalist approaches and statistical approaches. Section 3.3 presents the surface realisation algorithm we developed. Section 3.4 describes the evaluation setup and the results obtained. Section 3.5 concludes with pointers for future research.

Chapter 4 (Error Mining for Improving Symbolic Generation) presents our error mining approaches to improve the coverage of our generator. In Section 4.2, we discuss existing error mining approaches in parsing [van Noord, 2004; Sagot and de la Clergerie, 2006; de Kok *et al.*, 2009] in detail. Section 4.3 presents our algorithm to error mine dependency trees. In Section 4.3.1, we give a brief overview of the HybridTreeMiner algorithm, a complete and computationally efficient algorithm developed by Chi *et al.* (2004) for discovering frequently occurring subtrees in a database of labelled unordered trees. Section 4.3.2 shows how to adapt this algorithm to mine the dependency trees for subtrees with high suspicion rate. Section 4.3.3 shows how it permits mining the data for tree patterns of arbitrary size using different types of labelling information (POS tags, dependencies, word forms and any combination thereof). Section 4.4 presents our second algorithm which structures the output of error mining into a *suspicion tree*. We discuss in detail the error analysis using suspicion trees and their advantages over the ranked flat list of suspicious forms (Section 4.4.4). In Section 4.5, we show how these error mining algorithms help improving our surface realiser. Section 4.6 concludes with pointers for future research.

Chapter 5 (Generating Elliptic Coordination) describes how surface realisation handles elliptical sentences given an input where repeated material is omitted. We extract from the SR data 2398 inputs whose corresponding output sentence contains an ellipsis. Based on previous work on how to annotate and to represent ellipsis, we argue that elided material should be represented using phonetically empty nodes (Section 5.2). In Section 5.3, we introduce a set of rewrite rules which permits adding these empty categories to the SR data. We then evaluate our surface realiser on the resulting dataset (Section 5.4). Section 5.5 discusses related work on generating elliptic coordination. Section 5.6 concludes.

Part II: Simplifying Sentences

Chapter 6 (Background and Related Work) starts with introducing the sentence simplification task (Section 6.1) and its potential societal[1] and NLP applications (Section 6.1.1). In Section 6.2, we describe related work surrounding our research. In particular, we discuss handcrafted rules for syntactic simplification (Section 6.2.1) and four statistical frameworks [Zhu *et al.*, 2010; Coster and Kauchak, 2011; Woodsend and Lapata, 2011; Wubben *et al.*, 2012] for sentence simplification (Section 6.2.2). In Section 6.2.2.1, we describe Zhu *et al.* (2010)'s Parallel Complex-Simple sentence-aligned Wikipedia (PWKP) Corpus and its construction from Simple English Wikipedia and traditional English Wikipedia. Our supervised method uses the PWKP corpus for training and evaluation purposes.

Chapter 7 (Hybrid Simplification using Deep Semantics and Machine Translation) describes our supervised simplification framework with an example (Section 7.2). In Section 7.2.2, we formally define our simplification model combining probabilities from a DRS simplification model and an SMT based simplification model. Section 7.3 describes our automatic and human evaluation. We compare our outputs with the outputs of existing state-of-the-art methods on the PWKP evaluation corpus. Section 7.4 concludes with pointers for future research.

Chapter 8 (Unsupervised Sentence Simplification using Wikipedia) presents our unsupervised framework for sentence simplification (Section 8.2). It has three dedicated modules: lexical simplification, splitting and deletion. In Section 8.2.1, we show an example describing these three modules in function. Sections 8.2.2, 8.2.3 and 8.2.4 describe all three modules in detail. Section 8.3 compares our approach with other state-of-the-art methods on the PWKP evaluation corpus. Section 8.4 concludes with pointers for future research.

Chapter 9 (Conclusions) draws our conclusions on both parts "Generating Sentences" and "Simplifying Sentences" in Section 9.1. In Section 9.2, we give pointers for future research.

[1] Sentence simplification for people with reading disabilities, individuals with low-literacy, children and non-native speakers.

Part I

Generating Sentences

Chapter 2

Background

Contents

In this chapter, we describe the linguistic resources that we will use throughout Part I of this thesis. As described in Chapter 1, our generator takes shallow dependency trees from the Generation Challenge: Surface Realisation Shared Task [Belz *et al.*, 2011] as their input meaning representation. We start this chapter by explaining these dependency trees. Later, we discuss the Feature-based Lexicalised Tree-Adjoining Grammar formalism and its variant, Feature-based Regular Tree Grammar, used by our symbolic generator. We do not discuss related work in this chapter. This will be discussed separately in each of the following chapters.

2.1 The Surface Realisation Shared Task

SR Task

Belz *et al.* (2011) organised *The Generation Challenge: Surface Realisation Shared Task* (SR Task, in short) to provide a common-ground input representation for surface realisers. They constructed two datasets of input representations: one, shallow dependency structures (where relations are syntactic labels) and deep dependency structures (where relations are semantic labels). For both representations, relations were arbitrarily ordered. The shallow input representation is a more syntactic representation of the sentence. In contrast, the deep input representation is closer to a

semantic representation of the meaning of the corresponding sentence.

The SR Task datasets were constructed by post-processing the CoNLL 2008 Shared Task data [Surdeanu *et al.*, 2008]. The CoNLL 2008 Shared Task data was prepared by converting the Penn Treebank [Marcus *et al.*, 1993] to syntactic dependencies using the LTH Constituent-to-Dependency Conversion Tool for Penn-style Treebanks (PennConverter, [Johansson and Nugues, 2007]). The SR shallow dependency structures are then derived from the Pennconverter dependencies. For deep dependency structures, the Pennconverter dependencies are merged with the NomBank [Meyers *et al.*, 2004] and the PropBank [Palmer *et al.*, 2005] dependencies.

Here we assume as input to surface realisation, the shallow dependency structures provided by the SR task. In what follows, we describe these shallow dependency structures in more detail.

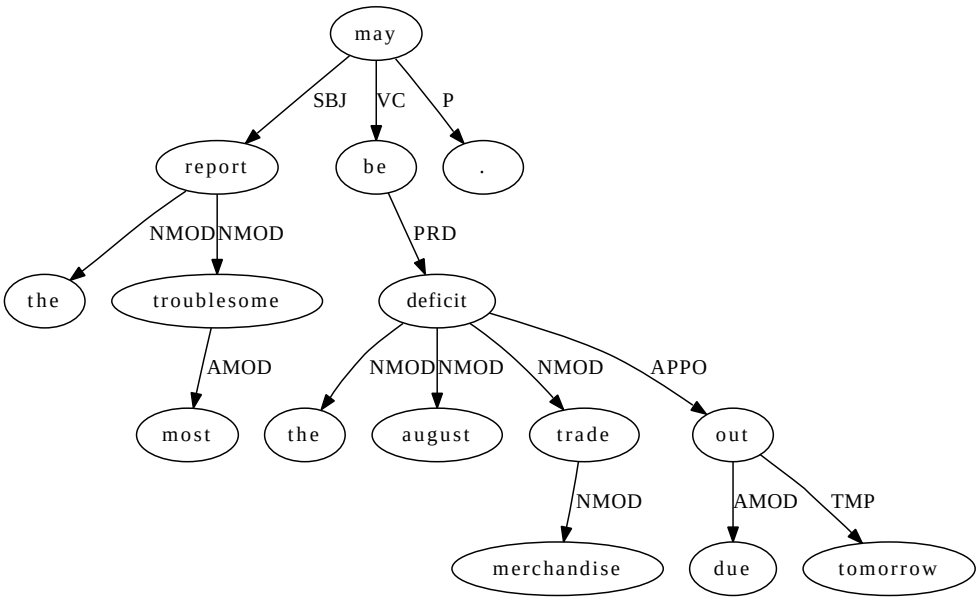


Figure 2.1: Shallow dependency tree from the SR Task for Sentence 5.

2.1.1 Shallow Dependency Structures

The shallow dependency structures provided by the SR task are unordered syntactic dependency trees. All words including punctuation markers of the original sentence are represented by a node in the tree. An example of the shallow dependency trees for the sentence (5) is given in Figure 2.1.

(5) The most troublesome report may be the August merchandise trade deficit due out tomorrow.

Nodes of the shallow dependency trees are labelled with lemmas, part of speech tags, partial morphosyntactic information such as tense and number and, in some cases, a sense tag identifier. The part of speech tag set is almost the same as the Penn Treebank POS tag set (Table 2.1) except that the tags VBP (Verb, non-3rd person singular present) and VBZ (Verb, 3rd person singular present) are treated indifferently and the agreement decision is left for the realisers to decide.

Penn Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Table 2.1: The Penn Treebank tag set.

Syntactic Label	Description
ADV	General Adverbial
AMOD	Modifier of adjective or adverbial
APPO	Apposition
BNF	Benefactor complement (for) in dative shift
CONJ	Second conjunct (dependent on conjunction)
COORD	Coordination
DEP	Unclassified
DIR	Adverbial of direction
DTV	Dative compelent (to) in dative shift
EXT	Adverbial of extent
EXTR	Extraposed element in cleft
HMOD	Token inside a hyphenated word (dependent on the head of the hyphenated word)
HYPH	Token part of a hyphenated word (dependent on the preceeding part of the hyphenated word)
IM	Infinitive verb (dependent on infinitive marker to
LGS	Logical subject of a passive verb
LOC	Locative adverbial or nominal modifier
MNR	Adverbial of manner
NAME	Name-internal link
NMOD	Modifier of nominal
OBJ	Object
OPRD	Predicative complement of raising/control verb
P	Punctuation
PMOD	Modifier of preposition
POSTHON	Posthonorific modifier of nominal
PRD	Predicative complement
PRN	Parenthetical
PRP	Adverbial of purpose or reason
PRT	Particle (dependent on verb)
PUT	Complement of the verb put
SBJ	Subject
SUB	Subordinated clause (dependent on subordinating conjunction)
SUFFIX	Possessive suffix (dependent on possessor)
SROOT	Root
TITLE	Title (dependent on name)
TMP	Temporal adverbial or nominal modifier
VC	Verb Chain
VOC	Vocative

Table 2.2: Atomic syntactic labels produced by the Pennconverter.

Edges or relations of the shallow dependency trees are labeled with syntactic labels. Table 2.2 shows the list of atomic syntactic labels produced by the Pennconverter. In general, edges are labeled with these atomic syntactic labels. However, edges can be labeled with non-atomic syntactic labels, which consists of multiple atomic labels or an atomic label marked with an additional information such as GAP (used to mark a gapping, cf. Chapter 5).

Belz *et al.* (2011) divided the SR data into three sets: the training set is taken from Sections 02-21 of the Penn Treebank, the development set from Section 24 and

the test set is a randomly selected subset of 100 sentences from Section 24.

The training set is obviously much larger compared to the test set. In our experiments, we do not need any training data to train our generator as it is a symbolic generator based on a hand-written non-probabilistic feature-based lexicalised Tree-Adjoining grammar. Instead, to facilitate the study of the coverage and the development of our grammar, we try to generate the whole training set of the SR data. In what follows, we just use the term "SR data" to refer to the training set of the SR data.

SR Data

2.2 Lexicalised Tree-adjoining Grammar

TAG

Tree-adjoining Grammar (TAG, [Joshi and Schabes, 1997] - originally introduced in [Joshi *et al.*, 1975] and [Joshi, 1985]) is a tree generating system which consists of elementary trees (initial trees and auxiliary trees) and two compositional operations - substitution and adjunction. The Lexicalised Tree-adjoining Grammar (LTAG) variant of TAG, in addition, anchors each elementary tree with a lexical item (lexicalisation). Derivation in TAG yields two trees: a *derived tree* which is, like for context free grammars, the tree produced by combining the grammar rules (here, the elementary trees) licensed by the input; and a *derivation tree* which indicates how the derived tree was built i.e., which elementary trees were used and how they were combined.

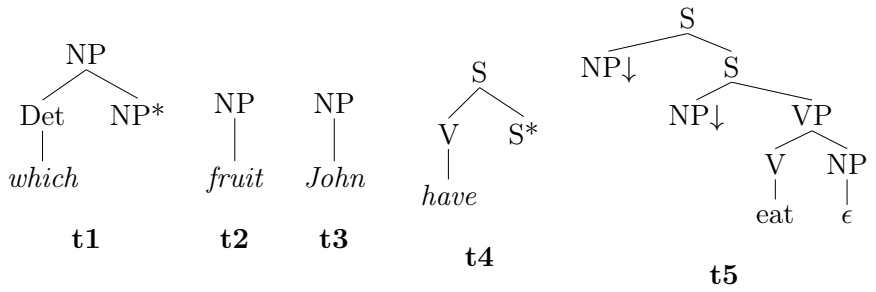


Figure 2.2: A toy lexicalised tree-adjoining grammar.

Formally, a tree-adjoining grammar is a tuple (Σ, N, I, A, S) where Σ is a finite set of terminals; N is a finite set of non-terminals ($\Sigma \cap N = \emptyset$); I is a finite set of finite trees, called initial trees; A is a finite set of finite trees, called auxiliary trees and S is a distinguished non-terminal symbol ($S \in N$). Initial trees are trees whose interior nodes are labelled with non-terminals and leaves are marked with substitution nodes (non-terminals marked with a down arrow $\downarrow$) or terminals. Auxiliary trees are trees

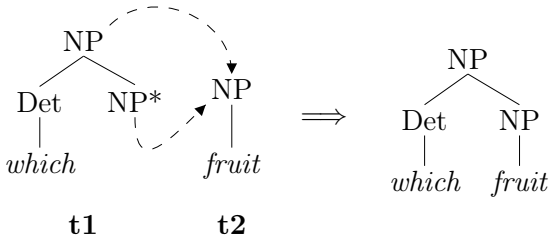


Figure 2.3: TAG adjunction operation.

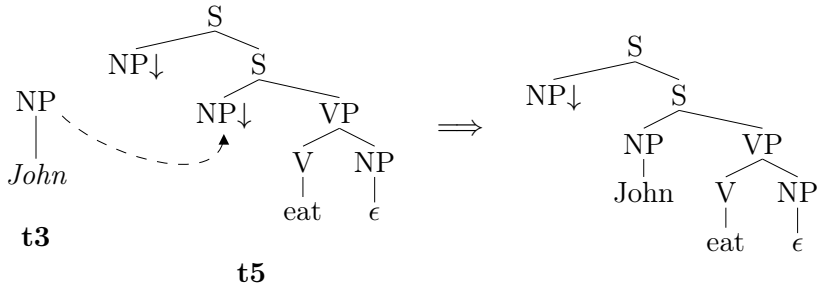


Figure 2.4: TAG substitution operation.

whose interior nodes are labelled with non-terminals, one leaf node is marked with a foot node (non-terminal with a star $*$) with the same category as its root node and the rest of the leaves are labelled with terminals.

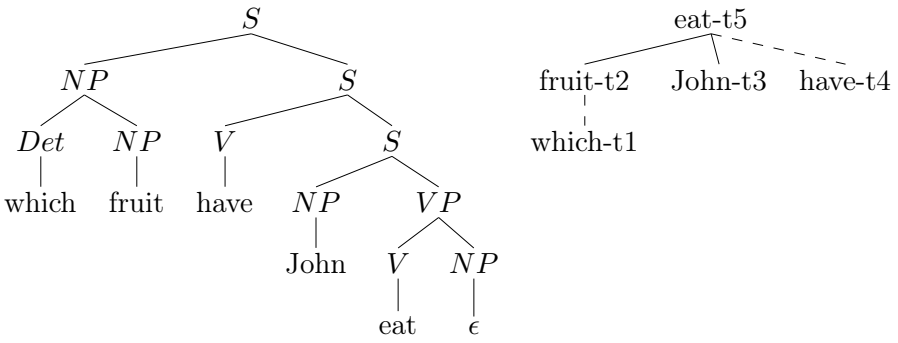


Figure 2.5: Derived and Derivation trees for the sentence *Which fruit has John eaten?*.

Figure 2.2 shows elementary trees from a toy lexicalised tree-adjoining grammar. Trees t2, t3 and t5 are initial trees whereas trees t1 and t4 are auxiliary trees. Figure 2.4 shows a substitution operation in TAG where the initial tree t3 gets substituted at one of the substitution site of the elementary tree t5. Figure 2.3 shows an adjunction operation where the auxiliary tree t1 adjoins at the root of the elementary tree t2.

Figure 2.5 shows derived and derivation trees for the sentence *Which fruit has John eaten?*.

2.2.1 Feature-based Lexicalized Tree-adjoining Grammar

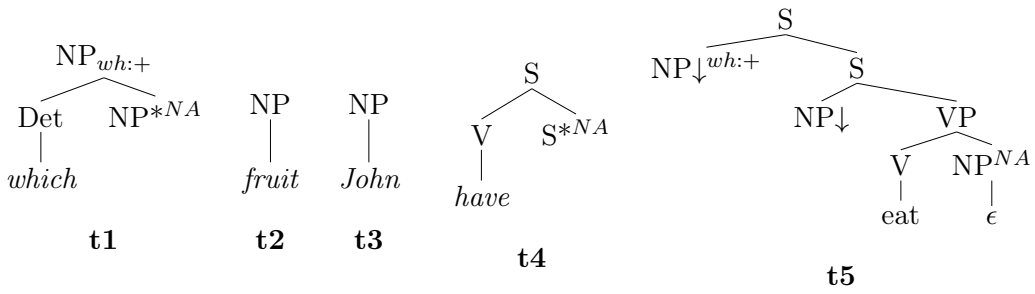


Figure 2.6: A toy feature-based lexicalised tree-adjoining grammar.

FB-LTAG

Feature-based Lexicalised Tree-adjoining Grammar (FB-LTAG, [Vijay-Shanker and Joshi, 1988]) is a variant of TAG where the nodes in the elementary trees are decorated with two feature structures called *top* and *bottom* which are unified during derivation. Figure 2.6 shows an example FB-LTAG version of the TAG grammar shown in Figure 2.2. Note that some of the features in Figure 2.6 are abbreviated for the clarity of representation. Figure 2.7 and Figure 2.8 show the feature unifications carried out along during substitution and adjunction respectively. Substitution unifies the top feature structure of a substitution node with the top feature structure of the root node of the tree being substituted in. The adjunction of an auxiliary tree to a tree node (adjunction node) unifies the top and bottom feature structures of the adjunction node with the top feature structure of the root node of the auxiliary tree and the bottom feature structure of its foot node respectively. Finally, at the end of the derivation, the top and bottom feature structures of all nodes in the derived tree are unified.

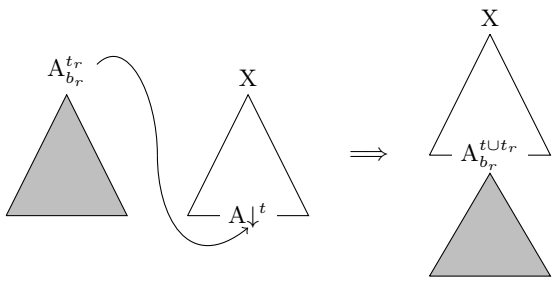


Figure 2.7: Feature unifications during substitution in FB-LTAG.

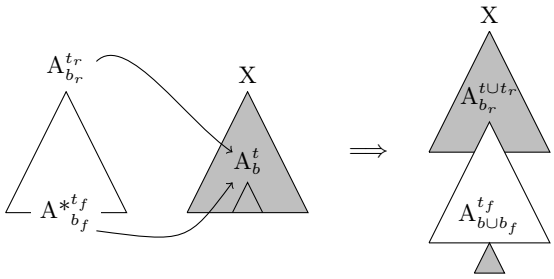


Figure 2.8: Feature unifications during adjunction in FB-LTAG.

Features in FB-LTAG enhance the descriptive capacity of the grammar compared to the TAG formalism but retain the formal properties of the TAG formalism. Hence, an FB-LTAG will be equivalent to a TAG from the point of view of generative capacity but one with an enhanced descriptive capacity. This enhanced descriptive capacity forthrightly helps generation by disallowing invalid operations along derivation which are restricted by feature unification constraints.

The grammar [Alahverdzhieva, 2008] underlying the surface realisation algorithm presented in Chapter 3 is an FB-LTAG for English consisting of roughly 1000 trees and whose coverage is similar to XTAG [The XTAG Research Group, 2001].

2.2.2 Feature-based Regular Tree Grammar

Koller and Striegnitz (2002), Gardent and Perez-Beltrachini (2010) and Gardent *et al.* (2011) have shown that processing the derivation trees of a given TAG rather than its derived trees is more efficient. Following Gardent and Perez-Beltrachini (2010) and Gardent and Perez-Beltrachini (2010), we therefore use not the FB-LTAG grammar described in the previous section, but the Feature-based Regular Tree Grammar (FB-RTG, [Schmitz and Roux, 2008]) of derivation trees that can be derived from it. That is, the surface realisation algorithm first builds a derivation tree. The generated sentence is then extracted from the derived tree which can be reconstructed from this derivation tree using the original FB-LTAG. FB-RTG

Figure 2.9 shows the FB-RTG corresponding to the FB-LTAG shown in Figure 2.6. The conversion from FB-LTAG to FB-RTG is described in detail in [Schmitz and Roux, 2008]. Intuitively, the FB-RTG representation of an FB-LTAG elementary tree t, is a rule whose LHS describes the syntactic requirement satisfied by t (e.g., S_S for an initial tree (subscript S from **s**ubstitution) rooted in S and VP_A for an auxiliary tree (subscript A from **a**djunction) rooted in VP) and whose RHS describes t's requirements. Adjunction is handled as an optional requirement which can be satisfied by the adjunction of an empty string and subscripts indicates the

$$
\begin{array}{lllll}
\textbf{r1} & NP_A^{[t:T]} & \rightarrow & which(NP_A^{[t:T,b:[wh:+]]}) \\
\textbf{r2} & NP_S^{[t:T]} & \rightarrow & fruit(NP_A^{[t:T]}) \\
\textbf{r3} & NP_S^{[t:T]} & \rightarrow & John(NP_A^{[t:T]}) \\
\textbf{r4} & S_A^{[t:T]} & \rightarrow & have(S_A^{[t:T]}) \\
\textbf{r5} & S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:+]]}\ S_A\ NP_S\ VP_A) \\
\textbf{r6} & X_A^{[t:T,b:T]} & \rightarrow & \epsilon
\end{array}
$$

Figure 2.9: A toy feature-based regular tree grammar corresponding to the grammar shown in Figure 2.6.

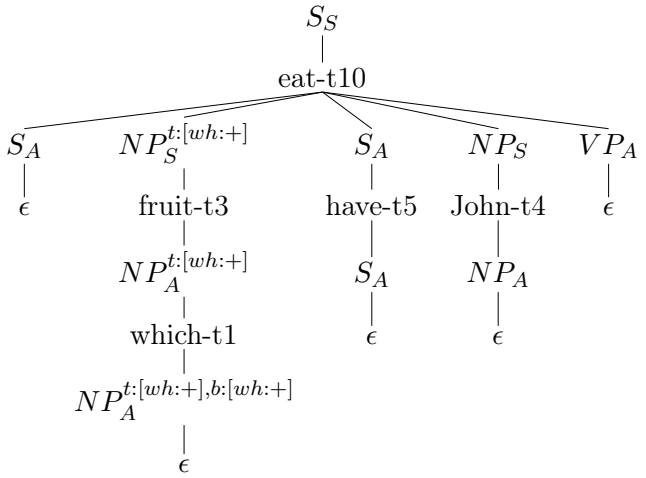


Figure 2.10: The FB-RTG derivation for "*Which fruit has John eaten*".

nature of the requirement (S for a substitution and A for adjunction). For instance, the rule **r5** in Figure 2.9 describes the contribution of the elementary tree **t5** in Figure 2.6 lexicalised with the lemma *eat* to a derivation tree as follows: **t5** can satisfy a requirement for a substitution on a node labelled with the S category (LHS with category S_S) and requires two substitutions both labelled with the NP category (NP_S and NP_S) and three optional adjunctions of category S, S and VP respectively (S_A, S_A, VP_A). At the end, the rule **r6** (not present in the original FB-LTAG) implements optional adjunction for arbitrary categories with X, a variable ranging over all syntactic categories.

The derivation process in FB-RTG produces trees that are almost identical to the FB-LTAG derivation trees. Figure 2.10 shows the FB-RTG derivation tree for the sentence "*Which fruit has John eaten?*". When abstracting away from the categorial nodes, the FB-RTG derivation tree mirrors the derivation tree of the original FB-

LTAG. The A and S subscripts indicate which operation was used for combining; and the nodes at which each FB-LTAG elementary tree adjoins or substitutes is encoded by features in these trees. Note that the FB-RTG derivation tree is unordered but there is a one-to-one correspondence between the FB-RTG derivation tree (Figure 2.10) and the FB-LTAG derivation tree (Figure 2.5, right) and hence, the FB-LTAG derived tree (Figure 2.5, left).

Chapter 3

Optimising Surface Realisation

Contents

This chapter is based on the the following paper:

Shashi Narayan and Claire Gardent. *Structure-driven lexicalist generation*, in Proceedings of the 24th International Conference on Computational Linguistics (COLING), pages 2027–2042, Mumbai, India, December 2012.

In this chapter, we present a novel algorithm for surface realisation with lexicalised grammars. In this algorithm, the structure of the input is used both top-down to constrain the selection of applicable rules and bottom-up to filter the initial search space associated with local input trees. In addition, parallelism is used to recursively pursue the realisation of each child node in the input tree. We evaluate the algorithm on the input data provided by the Generation Challenge Surface Realisation Shared Task and show that it drastically reduces processing time when compared with a simpler, top-down driven, lexicalist approach.

3.1 Introduction

Depending on the type of semantic representation encoded by the grammar, two main types of algorithms have been proposed for generating sentences with bi-directional, unification-based grammars such as CCG (Combinatory Categorial Grammar, [Espinosa *et al.*, 2010]), HPSG (Head-Driven Phrase Structure Grammar, [Carroll *et al.*, 1999a]) and TAG (Tree Adjoining Grammar, [Gardent and Kow, 2005]).

For recursive semantic representations such as first-order logic formulae, head-driven algorithms [Shieber *et al.*, 1990] have been argued to be best in restricting the combinatorics inherent to bottom-up search. They avoid non-termination by using lexical items to guide the search and they allow for semantically nonmonotonic grammars (i.e., grammars where the semantics of a rule's left-hand side need not be subsumed by the semantics of the rule's right-hand side). One main issue with this approach however is the so-called logical form equivalence problem [Shieber, 1993] where generators fail to produce natural language expressions for all the logically equivalent formulae. A logic formula may have several logically equivalent but syntactically distinct formulae. For instance $p \wedge q$ is logically equivalent to $q \wedge p$. In general though, a grammar will associate with natural language expressions only one of these logically equivalent formula. Hence a generator will be able to produce the natural language expression E only when given the formula ϕ associated by the grammar with E. For all other formulae logically equivalent to ϕ, it will fail. Since the problem of computing logical equivalence for e.g., first order logic is undecidable, the problem is quite deep.

For flat semantic representations such as MRSs (Minimal Recursion Semantics, [Copestake *et al.*, 2001]) on the other hand, lexicalist approaches [Espinosa *et al.*, 2010; Carroll and Oepen, 2005; Gardent and Kow, 2005] have extensively been used because they impose few constraints on the grammar thereby making it easier to maintain bi-directional grammars that can be used both for parsing and for genera-

tion; and the approach lessens the impact of the logical form equivalence problem – since the semantic representations are less structured, there are fewer logical equivalences to deal with [Copestake *et al.*, 2001]. One known drawback of lexicalist approaches, however, is that they generally lack efficiency. Indeed, previous work [Gardent and Kow, 2005; Gardent and Kow, 2007b; Gardent *et al.*, 2011] has shown that the high combinatorics of lexicalist approaches stem from (i) strong lexical ambiguity (each input element is usually associated with a large number of grammatical structures thereby inducing a very large initial search space); (ii) the lack of order information in the input (as opposed to parsing where the order of words in the input string restricts the number of combinations to be explored); and (iii) intersective modifiers (given n modifiers applying to the same constituent, there are $n!$ ways to combine these together).

In this chapter, we present an algorithm for surface realisation that combines techniques and ideas from the head-driven and lexicalist approaches. On the one hand, rule selection is guided, as in the lexicalist approach, by the elementary units present in the input rather than by its structure. On the other hand, the structure of the input is used to provide top-down guidance for the search and thereby restrict the combinatorics.

To further improve efficiency, the algorithm integrates three additional optimisation techniques. From the lexicalist approach, it adapts two techniques designed to prune the search space, namely a so-called polarity filter [Moortgat, 1997; Bonfante *et al.*, 2004] on local input trees; and the use of a language model to prune competing intermediate substructures. In addition, the algorithm is parallelised to explore the possible completions of the top-down predictions simultaneously rather than sequentially.

The algorithm was implemented using a FB-LTAG [Alahverdzhieva, 2008] (Chapter 2) for English and tested on the Surface Realisation Shared Task data (Chapter 2). We compare our algorithm with a baseline lexicalist approach [Narayan, 2011] which processes the input tree top down. The results show that the algorithm we propose drastically improves on the baseline, reducing generation time for sentences longer than 6 words w.r.t. this baseline.

Section 3.2 situates our approach with respect to related work. Section 3.3 presents the surface realisation algorithm we developed. Section 3.4 describes the evaluation setup and the results obtained. Section 3.5 concludes with pointers for future research.

3.2 Related Work

Most of the recent proposals on optimising surface realisation with unification grammars focus on lexicalist approaches because as mentioned above, these approaches place minimal requirements on the grammar and lessen the impact of the logical form equivalence problem. We now summarise symbolic and statistical optimisation techniques that were previously introduced to improve SR efficiency.

3.2.1 Symbolic Approaches to Surface Realisation

3.2.1.1 Head-driven Approach

Shieber *et al.* (1990) have proposed a head-driven algorithm for generation from the recursive semantic representations such as first-order logic formulae. We discuss two salient features of their algorithm.

Lexically guided search. Shieber *et al.* (1990) avoid non-termination by using lexical items to guide the search. For example, the verb gets generated before any of its complements. This makes full information of the subject to be available before it was generated. This avoids the nondeterminism inherent to left-to-right processing in bottom-up Earley based generators.

Semantically nonmonotonic grammars. Earlier, Shieber (1988) proposed a uniform approach to parsing and generation using semantically monotonic grammar[2]. Shieber *et al.* (1990) improve on Shieber (1988) by proposing a weaker constraint on the grammar and allowing semantically nonmonotonic grammars. With the new algorithm, the sentence *John calls him up* could be generated even though information for *up* appears nowhere in the goal semantics *call(john, him)*.

3.2.1.2 Lexicalist Approaches

HPSG Grammars. Carroll and Oepen (2005) present an chart-based bottom-up, lexicalist, surface realiser for wide-coverage unification-based HPSG grammars. In addition to multiple small refinements in their generation algorithm, they introduce two novel techniques: the integration of subsumption-based local ambiguity factoring, and a procedure to selectively unpack the generation forest according to a probability distribution given by a conditional, discriminative model.

[2]Semantic monotonic grammars are grammars in which the semantic component of each right-hand-side nonterminal subsumes some portion of the semantic component of the left-hand-side. [Shieber *et al.*, 1989]

Subsumption-based local ambiguity factoring and Selective unpacking Chart parsers with context-free grammars compute the parse forest in polynomial time by packing the sub-parses dominated by same nonterminal and covering the same segment of the input sentence into a single unitary representation. Carroll and Oepen (2005) adapted this technique to unification grammar based, chart-based realisation. They used feature structure subsumption to pack derivations which cover the same segment of input MRS semantics and have feature structures standing in a subsumption relation. For example the phrases in (6) would have equivalent feature structures and therefore packed into one representation.

(6) "*young Polish athlete*" and "*Polish young athlete*"

The selective unpacking procedure allows Carroll and Oepen to extract a small set of n-best realizations from the generation forest at minimal cost. The global rank order is determined by a conditional, discriminative Maximum Entropy (ME) model.

Carroll and Oepen's algorithm is evaluated on the *hike* treebank, a collection of 330 sentences of instructional text taken from Norwegian tourism brochures with an average length of 12.8 words. Practical generation times average below or around one second for outputs of 15 words [Carroll and Oepen, 2005].

TAG Grammars. Gardent and Kow (2007b) propose a three step surface realisation algorithm for FB-LTAG where first, a so-called polarity filter is used to prune the initial search space; second, substitution is applied to combine trees together; and third, adjunction is applied.

Polarity filtering The notion of polarity comes from the Categorial Grammar "count invariant" on argument and results [Moortgat, 1997]. Later, Bonfante *et al.* (2004) used polarisation for the abstraction of grammatical formalisms and showed how an abstract formalism can be used efficiently for lexical disambiguation. Following Bonfante *et al.* (2004), in our case, polarity filtering filters out combinations of FB-LTAG elementary trees which cover the input semantics but cannot yield a valid parse tree either because a syntactic requirement cannot be satisfied or because a syntactic resource cannot be used. In this way, the exponential impact of lexical ambiguity can be reduced.

POLARITY FILTERING

For example, Figure 3.1 shows the initial search space after lexical selection. Each tree is associated with polarity signature describing its syntactic resources or syntactic requirements. The tree for *the picture* is a syntactic resource of the category NP, hence has polarity (+np), whereas the tree for *the cost of* is itself a syntactic resource of the category NP but also requires an NP, hence has polarity (+np -np).

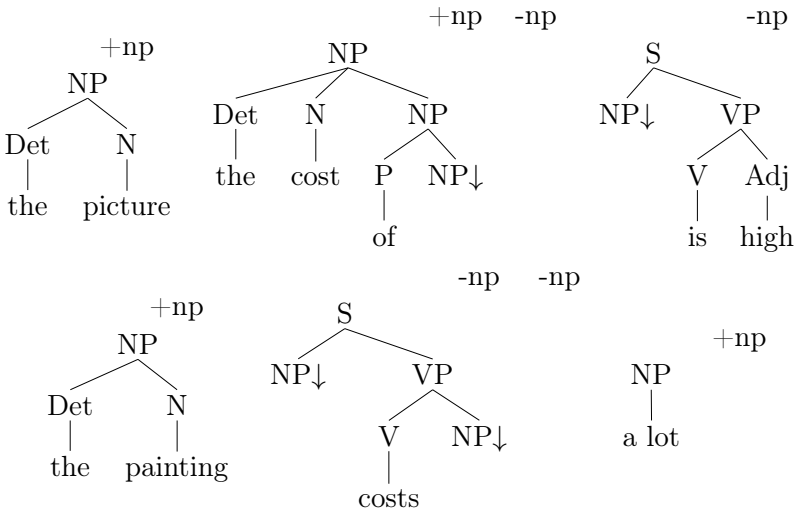


Figure 3.1: TAG elementary trees with their polarities.

Polarity filtering only lets through those combinations of elementary trees whose polarity signature is empty syntactic requirements. All other combinations of the elementary trees are dropped out because such combinations can never lead to the syntactically complete derived trees. The generator will allow generating "*the cost of the picture is high*" ($\emptyset$) but it will not allow the derivation for "*the cost of the picture a lot*" ([+2np]).

Delayed adjunction Applying substitution before adjunction means that first a skeleton sentence is built before modifiers are adjoined. This permits reducing the combinatorics introduced by intersective modifiers as the multiple intermediate structures they may license do not propagate to the rest of the sentence tree.

Gardent and Kow (2007b) and Gardent and Kow (2005) use this approach to generate paraphrases from flat semantics. In practice however, evaluation is restricted to short input and the algorithm fails to scale up [Gardent and Perez-Beltrachini, 2010].

Exploring derivation trees for generation For flat semantics, Koller and Striegnitz (2002) present a surface realisation algorithm where (i) the XTAG FB-LTAG grammar [The XTAG Research Group, 2001] is converted to a dependency grammar capturing the derivation trees of XTAG and (ii) a constraint-based dependency parser is used to construct derivation trees from semantic representations. The parser used was specifically developed for the efficient parsing of free word order languages

and is shown to efficiently handle both the lexical ambiguity and the lack of order information in the input that are characteristic of surface realisation from a flat semantics. The evaluation however is restricted to a few hand constructed example inputs; and the grammar conversion ignores feature structure information.

To address these shortcomings, Gardent and Perez-Beltrachini (2010) present an approach which makes use of the procedure for converting an FB-LTAG to an FB-RTG introduced by Schmitz and Roux (2008). As in [Koller and Striegnitz, 2002], the initial FB-LTAG is converted to a grammar of its derivation trees. However in this case, the grammar conversion and the resulting feature-based RTGs accurately translates the full range of unification mechanisms employed in the initial FB-LTAG. An Earley, bottom-up algorithm is developed and the approach is tested on a large benchmark of artificially constructed examples illustrating different levels of linguistic complexity (different input lengths, different numbers of clauses and of modifiers). The approach is shown to outperform the algorithm presented by Gardent and Kow (2007b) in terms of space. Speed is not evaluated however and the algorithm is not evaluated on real life data.

3.2.2 Probabilistic Approaches to Surface Realisation

Probabilistic techniques have been proposed in surface realisation to improve e.g., lexical selection, the handling of intersective modifiers and the selection of the best output.

Integrating n-gram scores In CCG based White's system [White, 2004], the best paraphrase is determined on the basis of n-gram scores. To address the fact that there are $n!$ ways to combine any n modifiers with a single constituent, White (2004) proposes to use a language model to prune the chart of identical edges representing different modifier permutations, e.g., to choose between *fierce black cat* and *black fierce cat*. Similarly, Bangalore and Rambow (2000b) assume a single derivation tree that encodes a word lattice (*a {fierce black, black fierce} cat*), and uses statistical knowledge to select the best linearisation.

Adapting supertagging to generation Bangalore and Rambow (2000b) adapted the supertagging techniques first proposed for parsing [Bangalore and Joshi, 1999] to surface realisation. While generating from dependency trees using FB-TAG, Bangalore and Rambow (2000b) used a tree model to produce a single most probable lexical selection for words in a tree. The tree model made an assumption that the choice of a tree for a node only depends on its children nodes. Later, Espinosa *et*

al. (2008) adapted the supertagging for CCG based realisers. Given a treebank in the appropriate format, this technique permits filtering the initial search space by using a model trained on that treebank. Only the most probable sequences of FB-LTAG elementary trees are considered. Although supertagging improves the performance of symbolic parsers and generators significantly, chances of assigning the wrong categories and failing to explore the best output remains.

In sum, various symbolic and statistical techniques have been developed to improve the efficiency of surface realisation. The algorithm we propose departs from these approaches in that it does not require training data; it is optimised by combining parallel processing, top-down prediction and local bottom-up polarity filtering; and it was evaluated on a large scale using the input data provided by the Generation Challenge SR Task.

3.3 Optimising a TAG-based Surface Realisation Algorithm

Taking inspiration from [Gardent *et al.*, 2011], our surface realisation algorithm builds on an FB-RTG grammar [Schmitz and Roux, 2008] derived from an FB-LTAG Grammar [Alahverdzhieva, 2008]. In what follows, we start with presenting an example input shallow dependency tree and an example FB-RTG along with its FB-LTAG. We then go on to present the surface realisation algorithm.

3.3.1 Input, Grammar and Example Derivation

We have already discussed both FB-RTG and shallow dependency trees in detail in Chapter 2. Here, we show an example input shallow dependency tree (Figure 3.2) from the SR Task data and an example FB-RTG along with its FB-LTAG (Figure 3.3). These are used throughout this chapter to explain the various optimisations integrated in our SR algorithm.

Our surface realisation algorithm builds on an FB-RTG describing the derivation trees of an FB-LTAG rather than the FB-LTAG itself. Given the input (Figure 3.2), our surface realisation algorithm first builds a derivation tree (Figure 3.4, top) using the FB-RTG (Figure 3.3, bottom). The generated sentence is then extracted from the derived tree which can be reconstructed from this derivation tree using the original FB-LTAG. Morphological realisation is carried out in a post-processing step from the list of lemmas and feature structures decorating the yield of the FB-LTAG derived tree.

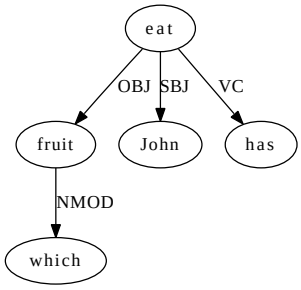
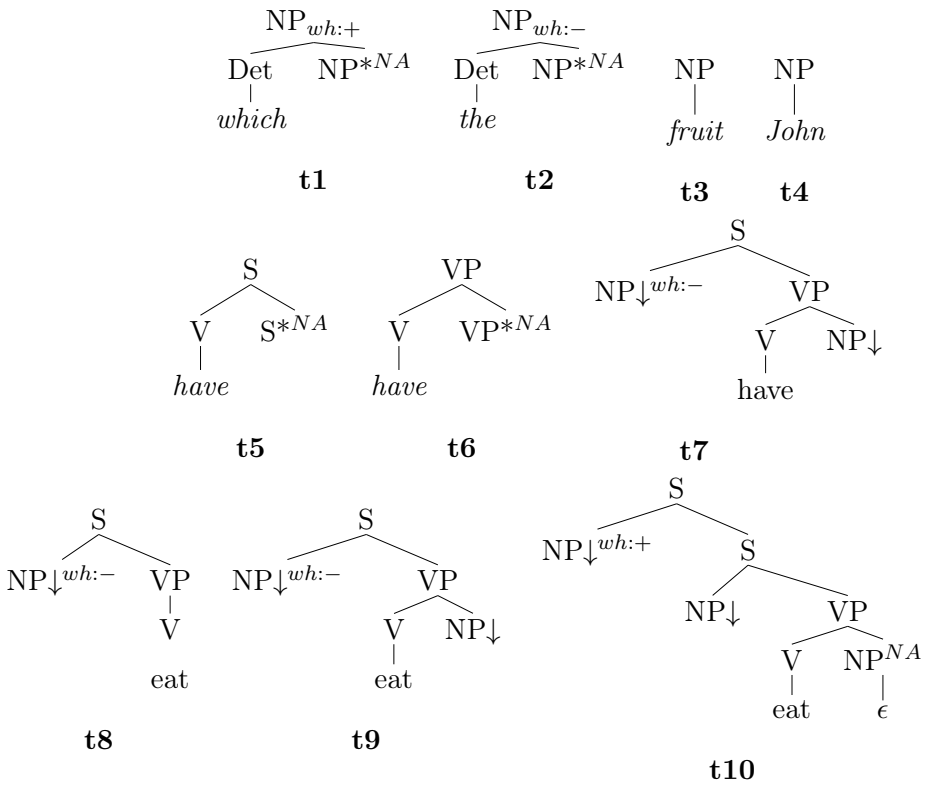


Figure 3.2: Shallow dependency tree for "*Which fruit has John eaten*".

Note that contrary to the flat semantic representations often used by surface realisers, the SR data has a clear tree structure. It has two advantages. One, the combinatorics induced by the lack of order in flat semantic representations is less in this task. Indeed, the algorithm we present exploits this structure to minimize the combinatorics. Similarly, [White, 2006] applies chunking constraints to the graph structure of flat semantic representation to constrain the generation of coordinate structures and address the issue of semantically incomplete phrases. Second, the nodes at which each FB-LTAG elementary tree adjoins or substitutes is encoded by features in these trees: for instance, the subject node of **t9** will have the feature *subject* while its object node will have the feature *object*. By comparing the dependency relations present in the input dependency tree with the feature values given by the grammar, it is thus possible to determine on which nodes of the parent tree in the derivation tree, its children trees should combine.

3.3.2 The Surface Realisation Algorithm

Our surface realisation starts from the root node of the input tree and processes all children nodes in parallel by spreading the lexical selection constraints top-down and completing the FB-RTG rules bottom-up. Figure 3.5 shows the architecture of the surface realiser. The controller provides the interface to our surface realization system. It takes a shallow dependency tree as input and produces a ranked list of sentences as output. More specifically, the controller defines a process pool such that each process present in this pool represents a node (a lemma) in the input dependency tree and the communication scheme among processes reflects the dependency relations among nodes in the input dependency tree. In this way, generation is guided by the structure of the input dependency tree. Each process node executes Algorithm 1 at its core. In Algorithm 1, the function $listen(N_1, N_2, ...)$ listens to the messages from the nodes $N_1, N_2, ...$ and the function $send(N_1, N_2, ...)$ sends messages



r1	$NP_A^{[t:T]}$	$\rightarrow$	$which(NP_A^{[t:T,b:[wh:+]]})$
r2	$NP_A^{[t:T]}$	$\rightarrow$	$the(NP_A^{[t:T,b:[wh:-]]})$
r3	$NP_S^{[t:T]}$	$\rightarrow$	$fruit(NP_A^{[t:T]})$
r4	$NP_S^{[t:T]}$	$\rightarrow$	$John(NP_A^{[t:T]})$
r5	$S_A^{[t:T]}$	$\rightarrow$	$have(S_A^{[t:T]})$
r6	$VP_A^{[t:T]}$	$\rightarrow$	$have(VP_A^{[t:T]})$
r7	$S_S^{[t:T,b:B]}$	$\rightarrow$	$have(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S)$
r8	$S_S^{[t:T,b:B]}$	$\rightarrow$	$eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A)$
r9	$S_S^{[t:T,b:B]}$	$\rightarrow$	$eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S)$
r10	$S_S^{[t:T,b:B]}$	$\rightarrow$	$eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:+]]}\ S_A\ NP_S\ VP_A)$
r11	$X_A^{[t:T,b:T]}$	$\rightarrow$	ϵ

Figure 3.3: An example FB-LTAG and the corresponding FB-RTG.

to the nodes $N_1, N_2, ...$ The controller initiates the generation process by sending a start message to the root node.

The algorithm proceeds in five major steps as follows.

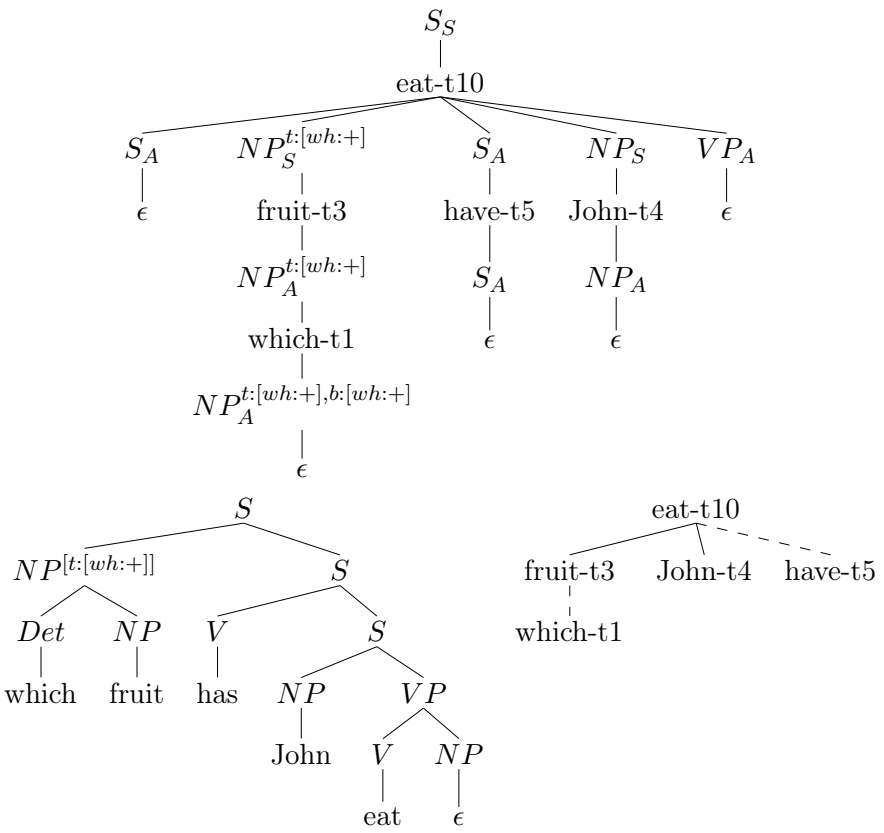


Figure 3.4: The FB-RTG derivation for "*Which fruit has John eaten*" and the corresponding FB-LTAG derived and derivation trees.

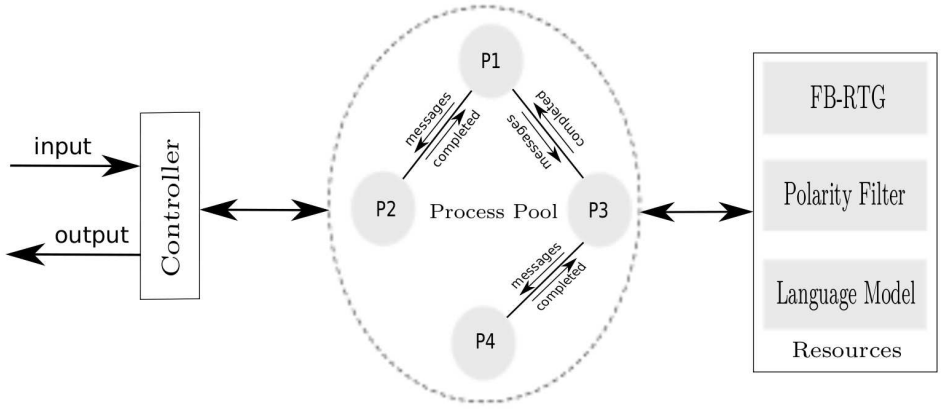


Figure 3.5: A parallel architecture for Surface Realisation.

Algorithm 1 Surface Realisation Algorithm: Steps taken at each Node

listen(parent-node)
Top-Down Rule Selection and Filtering
if leaf-node **then**
  Leaf Closure
else
  send(child-nodes)
  listen(child-nodes)
  Local Polarity Filtering
  Bottom-up Generation
  N-gram Filtering using a Language Model
end if
send(parent-node)

Top-Down Rule Selection and Filtering. Starting from the root node, the input dependency tree is traversed top-down to associate each node in the input tree with a set of FB-RTG grammar rules. This step corresponds to the lexical lookup phase of lexicalist approaches whereby each literal in the input selects the grammar rules whose semantics subsumes this literal. Our approach differs from existing lexicalist approaches however in that it uses the top-down information given by the structure of the input to filter out some possibilities that cannot possibly lead to a valid output. More precisely, for each input node n with lemma w, only those rules are selected which are associated with w in the lexicon. In addition, the left-hand side (LHS) category of each selected rule must occur at least once in the right-hand side (RHS) of the rules selected by the parent node.

For instance, given the input dependency tree shown in Figure 3.2 for the sentence "*Which fruit has John eaten?*", and the grammar given in Figure 3.3, all rules **r8**, **r9** and **r10** associated with the lemma '*eat*' will be selected[3] because all of them corresponds to the S^4 rooted initial trees[5].

$$\begin{array}{llll} \surd & \textbf{r8} & S_S^{[t:T,b:B]} & \rightarrow \quad eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A) \\ \surd & \textbf{r9} & S_S^{[t:T,b:B]} & \rightarrow \quad eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S) \\ \surd & \textbf{r10} & S_S^{[t:T,b:B]} & \rightarrow \quad eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:+]]}\ S_A\ NP_S\ VP_A) \end{array}$$

The parent process creates a new lexical selection constraint message consisting of its RHS requirements in selected RTG rules and passes it to its children processes. In

[3]The symbols $\surd$ and $\times$ before rules mark the successful and failed operations respectively.

[4]The controller triggers the root process "eat" with the initial lexical selection constraint (S_S, S rooted initial trees) to generate complete sentences.

[5]The grammar is lexicalised with lemmas rather than forms. The appropriate forms are generated at the end of the generation process based on the lemmas and on the feature structures decorating the yield of the trees output by the generator.

Figure 3.2, the process associated with the node '*eat*' will send a message consisting of S_A, NP_S and VP_A (RHS requirements of rules **r8**, **r9** and **r10**) to its children processes associated with '*fruit*', '*John*' and '*have*'.

Starting from the trigger initiated by the *controller*, the process of message spreading happens in a recursive and parallel manner throughout the process pool reflecting the input dependency tree in a top-down fashion. It eliminates all RTG rules which cannot possibly lead to a valid output well before carrying out any substitution and adjoining operation on the RTG rules.

For instance, the rule **r7** for '*have*' will not be selected because its left-hand side is S_S which does not satisfy the lexical selection constraints (S_A, NP_S and VP_A) sent by its parent '*eat*'.

$$\begin{array}{cccc} \surd & \mathbf{r5} & S_A^{[t:T]} & \rightarrow \quad have(S_A^{[t:T]}) \\ \surd & \mathbf{r6} & VP_A^{[t:T]} & \rightarrow \quad have(VP_A^{[t:T]}) \\ \times & \mathbf{r7} & S_S^{[t:T,b:B]} & \rightarrow \quad have(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S) \end{array}$$

Leaf Closure. When reaching the leaf nodes of the input tree, the top and bottom feature structures of the rules selected by these leaf nodes are unified. The completed rules of a leaf node are sent back to its parent.

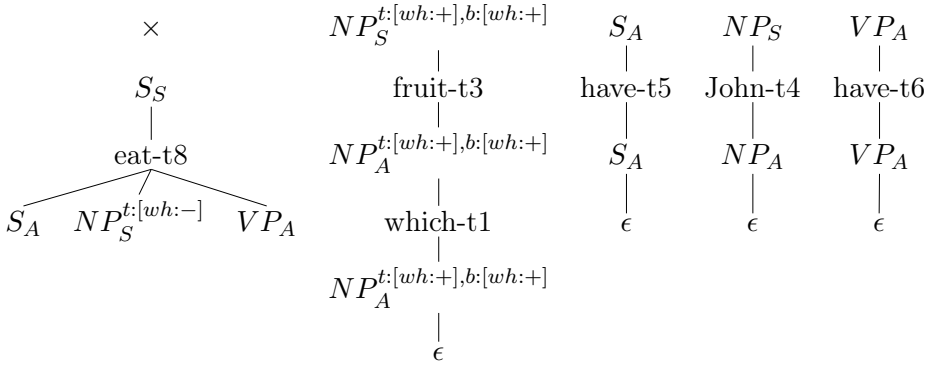


Figure 3.6: Local polarity filtering in FB-RTG derivation.

Local Polarity filtering. As mentioned in Section 3.2, polarity filtering [Gardent and Kow, 2005] eliminates from the search space those sets of rules which cover the input but cannot possibly lead to a valid derivation either because a substitution node cannot be filled or because a root node fails to have a matching substitution site[6]. While Gardent and Kow (2005) applies polarity filtering to the initial search

[6]Since it only eliminates combinations that cannot possibly lead to a valid parse, polarity filtering does not affect completeness. Nor does it place any particular constraint in the grammar. All that

space (the set of rules selected by all literals in the input), we apply polarity filtering to each local tree while going up the input tree . Thus, this filtering will weed out all combinations of parent rules and completed immediate children rules which cannot possibly yield a complete tree either because some child rule cannot be used or because some requirement of the parent rule cannot be satisfied. For instance, after processing the children of the '*eat*' node in the input dependency tree shown in Figure 3.2, all combinations of **r8** (intransitive '*eat*') with the children trees will be excluded. This is because at this stage of processing, the trees built bottom up for '*which fruit*', '*John*' and '*have*' includes two NPs with LHS category NP_S (Figure 3.6) while the **r8** rule only requires one such NP. That is, for this rule, the completed child rule for *which fruit* will show up as a superfluous syntactic resource.

LOCAL POLARITY FILTERING

$$\begin{array}{cccccl} \times & \mathbf{r8} & S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A) \\ \surd & \mathbf{r9} & S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S) \\ \surd & \mathbf{r10} & S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:+]]}\ S_A\ NP_S\ VP_A) \end{array}$$

By restricting polarity filtering[7] to local input trees, we avoid the computation of the very large automaton required when filtering the global initial search space as done in [Gardent and Kow, 2005]. In practice, automaton construction in this approach leads to very high runtimes on medium size input and to timeouts on larger (more than 10 literals) or more complex input (multiple adjunction cases).

The combined effect of top-down filtering and local polarity filtering avoids considering most of RTG rules which can never lead to valid output well before carrying out any substitution and adjoining operation on the RTG rules to try to complete them. The Earley, bottom-up algorithm described in [Gardent and Perez-Beltrachini, 2010] also achieves some amount of top-down filtering during its prediction stage but the lexical selection constraint is limited to the top of the RHS requirements of the RTG rule being processed, hence it may try completing the RTG rules which cannot possibly lead to a valid output whereas in our proposed approach all RHS requirements of the selected RTG rules are available as the lexical selection constraint information during both top-down filtering and local polarity filtering steps.

is required is that the grammar encodes a notion of resources and requirements i.e., of items that cancel each other out. Typically, grammar rules support this constraint in that e.g., the left-hand side of a rule and one category in the right-hand side of another rule can be viewed as canceling each other out if they match.

[7] As noted by one of our reviewers on the paper [Narayan and Gardent, 2012b], supertagging models can probably approximate local polarity filtering. For instance, a supertagger model might learn that an intransitive category is very unlikely whenever the input dependency tree contains more than one core arguments.

Bottom-up Generation. For each local tree in the input, the rule sets passing the local polarity filter are tried out for combination. The completed child RTG rules are combined with the rules selected by the parent node using substitution and adjoining operations. During the completion, a parent rule fails to complete if any feature conflicts are found.

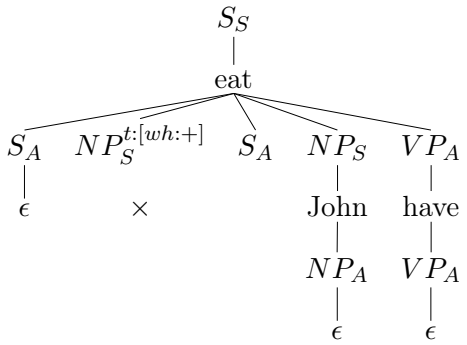


Figure 3.7: Local polarity filtering for the input tree `eat(john, has)`.

Note that for each rule set let through by polarity filtering, the category and the number of children trees exactly match the requirement of the associated parent rule. For instance, as explained above, the rule **r8** representing an intransitive use of the verb '*eat*' is ruled out by polarity filtering since it does not permit "consuming" the NP_S resource provided by one of NPs '*which fruit*' or '*John*'. Conversely, given an input tree of the form `eat(john, has)`, the rules **r9** and **r10** representing a transitive use of the verb '*eat*' would be filtered out by polarity filtering. As a result, the intermediate structure shown in Figure 3.7 will not be computed because it cannot be completed given the input. That is, while the global polarity filtering used in [Gardent and Kow, 2005] permits weeding out global combination of trees that are invalid, local polarity filtering additionally permits reducing the number of intermediate structures built first, because there is no need for prediction i.e., for active chart items and second, because intermediate structures that cannot possibly lead to a valid derivation are not built.

N-gram Filtering using a Language Model. To further prune the search space and to appropriately handle word order, the SR algorithm also integrates a language model and can be parametrized for the number of best scoring n-grams let through after each bottom-up generation step. In this way, not all possible orderings of intersective modifiers are produced, only those that are most probable according to the language model. In our experiments, we used a language model trained on the

Penn Treebank.

3.4 Empirical Evaluation

We now report on the results obtained when running the algorithm and the grammar described above on the shallow input data provided by the Generation Challenge Surface Realisation Task. Because we are presenting an algorithm for surface realisation rather than a surface realiser, the main focus of the evaluation is on speed (not coverage or accuracy). Nevertheless, we also report coverage and BLEU score as an indication of the capabilities of the surface realiser i.e., the algorithm combined with the grammar and the lexicon.

Note that the results reported in this chapter is with the lexicon and the grammar [Alahverdzhieva, 2008] which have not been tested before at a large scale. Hence, as we might expect coverage and accuracy of the surface realisation algorithm presented are low. The average BLEU score is reported only for the covered sentences by taking the average of the BLEU score for each covered sentence. For each covered sentence, the top ranked generated sentence is taken as the final output. We use the SR Task scripts for the computation of the BLEU score.

In Chapter 4, we propose to mine the shallow dependency trees for the errors in our symbolic generation system. We show that after improvements in our lexicon and grammar, we achieve a large boost in both coverage and accuracy.

3.4.1 Runtimes

The SR data on which we evaluate our surface realisation algorithm are the shallow dependency trees described in Section 3.3.1.

We use as a baseline the FB-RTG based lexicalist approach (**B**ASELINE) described in [Narayan, 2011]. In this approach, FB-RTG rules are selected top-down following the structure of the input dependency tree and all FB-RTG rules selected for a given local input tree are tried out for combination using a chart-based approach. This baseline thus permits observing the impact of the various optimisations described below. In future work, it would be interesting to obtain time information from the systems participating in the SR challenge and to compare them with those of our system.

TDBU-PAR (top-down, bottom-up and parallelised) is the algorithm presented here running on a 4 core system. To evaluate the impact of parallelism on runtimes, we also computed runtimes for a sequential version of the same algorithm (**T**DBU-SEQ). In **T**DBU-SEQ, children subtrees (processes) of the input dependency tree

Algorithm	Sentences (Length L)							
	$S(0-5)$		$S(6-10)$		$S(11-20)$		$S(All)$	
	Total	Succ	Total	Succ	Total	Succ	Total	Succ
	1084	985	2232	1477	5705	520	13661	2744
BASELINE	0.85	0.87	10.90	10.76	110.07	97.52	–	–
TDBU-SEQ	1.49	1.63	2.84	3.64	4.36	6.03	4.52	3.18
TDBU-PAR	1.53	1.66	2.56	3.28	2.66	4.14	2.57	2.78

Table 3.1: Comparison between generation times (seconds).

are processed sequentially.

				Algorithm	
				TDBU-SEQ	**T**DBU-PAR
Sentences (Arity)	S(1)	Total	190	0.89	0.97
		Succ	178	0.94	1.03
	S(2)	Total	1218	2.52	2.35
		Succ	964	2.63	2.50
	S(3)	Total	3619	3.65	2.63
		Succ	1039	3.39	3.10
	S(4)	Total	5320	5.07	2.91
		Succ	605	4.54	3.77
	S(5)	Total	2910	5.24	2.86
		Succ	137	4.62	3.88
	S(6)	Total	1093	8.20	3.09
		Succ	18	7.29	4.76

Table 3.2: Comparison between generation times (seconds) with varying arities.

Table 3.1 shows the runtimes for the three surface realisation algorithms **B**ASELINE, **T**DBU-SEQ and **T**DBU-PAR. For the TDBU algorithms, the n-gram filtering is set to 10 that is, for each local input tree, the 10 best n-grams are passed on. We split the data into 4 sets according to the input length where the input length is the number of nodes (or words) in the input dependency tree. The average number of words in a sentence in the first set $S(0-5)$ is 4, in the second set $S(6-10)$, 7, in the third set $S(11-20)$, 15, and in the final set $S(All)$ (all lengths), 17. The maximum length of a sentence in the final set $S(All)$ is 74. To make comparisons between **B**ASELINE, **T**DBU-SEQ and **T**DBU-PAR possible, the maximum arity of words present in the input dependency trees is set to 3 (because BASELINE mostly fails on input containing nodes with higher arity).

BASELINE turns out to be faster than **T**DBU-PAR and **T**DBU-SEQ for sentences of smaller length ($\leq$ 5). It can be explained because of the parallelism

and the multiprocessing overhead. But **T**DBU-PAR and **T**DBU-SEQ leaves behind **B**ASELINE for longer sentences. For input longer than 10, the simple **B**ASELINE algorithm times out whereas **T**DBU-PAR remains stable. For $S(All)$, **T**DBU-PAR achieves a reasonable average of 2.57 seconds for all sentences (Total) and 2.78 seconds for successful sentences (Succ).

Table 3.1 does not show a big difference in performance between **T**DBU-PAR and **T**DBU-SEQ because the maximum arity of the input dependency trees is kept low (maximum 3). In Table 3.2, we split the data by arity whereby the dataset $S(i)$ consists of input dependency trees with maximum arity i. As can be seen, the difference between the two algorithms steadily increases with the arity of the input thereby demonstrating the impact of parallelism.

3.4.2 Coverage and Accuracy

The grammar and lexicon used to test the surface realisation algorithm presented in this chapter had not been tested before and as the evaluation shows their coverage and accuracy are still low. Table 3.3 shows the coverage and accuracy (on the covered sentences) results obtained for sentences of size 6 (S-6), 8 (S-8) and all (S-All). The dataset S-All differs from the dataset $S(All)$ discussed in previous section. S-All considers all sentences without any restriction over the maximum arity in the input dependency trees. S-All consists of 26725 sentences with the average length of 22 and the maximum length of 134. The maximum arity in these sentences varies from 1 to 18 with an average of 4.

Data Type	Total	Coverage (#)		Coverage (%)	BLEU Score (Covered)
		Covered	Uncovered		
S-6	3877	3506	371	90.43	0.835
S-8	3583	3038	545	84.79	0.800
S-All	26725	10351	16374	38.73	0.675

Table 3.3: Coverage and Bleu Scores for covered sentences.

As can be seen coverage markedly decreases for longer sentences. Error mining (Chapter 4) on this data indicates that failure to generate is mostly due to mismatches between the dependency tree input and the structure expected by the grammar; to discrepancies in the lexicon; and to complex sentence coordinations (e.g., verb coordination, gapping phenomenon) which could be very common in sentences of average length 22 in S-All. Other failure causes are inadequate treatments of multiword expressions and foreign words. In the next chapter, we demonstrate how error mining permits rapidly addressing these issues and improving both coverage

and accuracy.

3.5 Conclusion

We presented a novel algorithm for surface realisation with lexicalised grammar which takes advantage of the input structure (a tree) to filter the initial search space both top-down and bottom up. To further improve efficiency, the algorithm integrates three additional optimisation techniques: the local polarity filtering, the use of a language model and the parallelised processes. We evaluated this algorithm on large scale SR Task data and showed that it drastically reduces runtimes for sentences longer that 6 words compared to a simple lexicalist approach which explores the whole search space.

As mentioned in Section 3.3.1, the input data provided by the SR task differs from the flat semantic representations assumed by most existing surface realisers in that it displays a clear tree structure. The algorithm presented here makes use of that structure to optimize performance. It would be interesting to see whether the hybrid top-down, bottom-up approach we developed to guide the SR search can be generalised to the graph structure of semantic representations. We put some more light on this area in the future work section in Chapter 9.

Chapter 4

Error Mining for Improving Symbolic Generation

Contents

This chapter is based on the following papers:

Claire Gardent and Shashi Narayan. *Error mining on dependency trees*, in Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL), pages 592–600, Jeju Island, Korea, July 2012.

Shashi Narayan and Claire Gardent. *Error mining with suspicion trees: Seeing the forest for the trees*, in Proceedings of the 24th International Conference on Computational Linguistics (COLING), pages 2027–2042, Mumbai, India, December 2012.

In this chapter, we introduce two novel error mining algorithms: first, for mining trees to detect the most likely sources of generation failure while generating from shallow dependency trees; and second, for structuring the output of error mining into a tree (called, suspicion tree) highlighting the relationships between suspicious forms. We show how these error mining algorithms permit us to address shortcomings of our generation system and to improve its coverage and accuracy.

4.1 Introduction

In recent years, error mining techniques have been developed to help identify the most likely sources (called, *Suspicious forms*) of parsing failure [van Noord, 2004; Sagot and de la Clergerie, 2006; de Kok *et al.*, 2009]. First, the input data (text) is separated into two subcorpora, a corpus of sentences that could be parsed (PASS) and a corpus of sentences that failed to be parsed (FAIL). For each n-gram of words (and/or part of speech tag) occurring in the corpus to be parsed, a suspicion rate is then computed which, in essence, captures the likelihood that this n-gram causes parsing to fail.

van Noord (2004) initiated error mining on parsing results with a very simple approach computing the parsability rate of each n-grams in a very large corpus. The parsability rate of an n-gram is the proportion of sentences in which this n-gram occurs and for which parsing succeeds. An n-gram then, is a suspicious form if it has a low parsability rate. van Noord's approach was extended and refined in [Sagot and de la Clergerie, 2006] and [de Kok *et al.*, 2009] as follows. Sagot and de la Clergerie (2006) defined a suspicion rate for n-grams which takes into account the number of occurrences of a given word form and iteratively defines the suspicion rate of each word form in a sentence based on the suspicion rate of this word form in the corpus. Further, de Kok *et al.* (2009) extended this iterative error mining to n-grams of words and POS tags of arbitrary length.

These error mining techniques have been applied with good results on parsing output and shown to help improve the large scale symbolic grammars and lexicons used by the parser. However these approaches have two major shortcomings.

First, the techniques they use (e.g., suffix arrays) to enumerate and count n-grams builds on the sequential nature of a text corpus and cannot easily extend to structured data. There are some NLP applications though where the processed data is structured data such as trees or graphs and which would benefit from error mining. For instance, when generating sentences from dependency trees, as was proposed recently in the Generation Challenge SR Task [Belz *et al.*, 2011], it would

be useful to be able to apply error mining on the input trees to find the most likely causes of generation failure.

Second, all these error mining approaches generates a flat list of suspicious forms ranked by decreasing order of suspicion. Such kind of list often presents related and distinct error cases in a interleaved fashion. There is no clear overview of how the various suspicious forms interact and as a result, the linguist must "hop" from one error case to another instead of focusing on improving sets of related error cases. In short, the output of these error mining approaches lacks structure thereby making it difficult to handle errors in a linguistically meaningful way.

In this chapter, we address these issues and propose two new algorithms: (i) we propose an approach that supports error mining on trees; we adapt an existing algorithm for tree mining which we then use to mine the SR Task dependency trees and identify the most likely causes of generation failure; and (ii) we propose another algorithm which structures the output of error mining into a *suspicion tree* making explicit both the ranking of the main distinct error cases and their subcases. The suspicion tree is a binary tree structure whose internal nodes are labelled with suspicious forms and whose leaf nodes represent the clusters of error mined data grouped according to the suspicious forms characterizing their elements. Like in a decision tree, each cluster in the suspicion tree is characterized by the set of attributes (suspicious forms) labelling its ancestors; and the tree itself represents a disjunction of mutually exclusive error cases.

We illustrate the impact of our error mining algorithms on error analysis by applying it to detect and analyse the most likely sources of failure in our surface realiser; and we show how this error mining algorithm permits improving the surface realiser. Moreover, we illustrate the impact of structuring the error mining output on how it helps handling errors in a linguistically meaningful way.

This chapter is structured as follows. Section 4.2 discusses related work [van Noord, 2004; Sagot and de la Clergerie, 2006; de Kok *et al.*, 2009]. Section 4.3 presents our algorithm to error mine dependency trees. In Section 4.3.1, we give a brief overview of the HybridTreeMiner algorithm, a complete and computationally efficient algorithm developed by Chi *et al.* (2004) for discovering frequently occurring subtrees in a database of labelled unordered trees. Section 4.3.2 shows how to adapt this algorithm to mine the SR dependency trees for subtrees with high suspicion rate. Section 4.3.3 shows how it permits mining the data for tree patterns of arbitrary size using different types of labelling information (POS tags, dependencies, word forms and any combination thereof). Section 4.4 presents our second algorithm which structures the output of error mining into a *suspicion tree*. In essence, this

algorithm adapts Quinlan (1986)'s ID3 algorithm to build a suspicion tree such that the clusters obtained group together sets of input data that share similar sources of failure (called *suspicious forms*); and the attributes labelling these clusters are the suspicious forms indicating which are these most likely causes of failure. We discuss in detail the error analysis using suspicion trees and their advantages over the ranked flat list of suspicious forms (Section 4.4.4). Finally, in Section 4.5, we show how these error mining algorithms help improving the coverage and the accuracy of the surface realiser described in Chapter 3. Section 4.6 concludes with pointers for further research.

4.2 Related Work

Various error mining techniques have been developed to help identify the most likely sources of parsing failure [van Noord, 2004; Sagot and de la Clergerie, 2006; de Kok *et al.*, 2009]. In this section we go through them in detail and also investigate if such techniques have been used to identify generation failures. We relate our proposal to the existing proposals.

4.2.1 Error Mining for Parsing

4.2.1.1 Global Parsability Metric for N-grams

van Noord (2004) initiated error mining on parsing results with a very simple approach computing the parsability rate of each n-gram in a very large corpus. The parsability rate of an n-gram $w_i \dots w_n$ is defined as the ratio:

$$R(w_i \dots w_n) = \frac{count(w_i \dots w_n \mid OK)}{count(w_i \dots w_n)}$$

where $count(w_i \dots w_n)$ is the number of sentences in which the n-gram $w_i \dots w_n$ occurs and $count(w_i \dots w_n \mid OK)$ the number of sentences containing $w_i \dots w_n$ which could be parsed. The corpus is stored in a *suffix array* and the sorted suffixes are used to compute the frequency of each n-grams in the total corpus and in the corpus of parsed sentences.

van Noord (2004) presents the result of the error mining as a flat list of suspicious n-grams by increasing order of the parsability rate. The n-gram with the lowest parsability rate, i.e., it seldom occurs with sentences that could be parsed, is the most suspicious form and tops the list.

4.2.1.2 Combining Global and Local Suspicion Rates

van Noord (2004) tries to find suspicious forms that are more frequent in sentences that could not be parsed. It does not take into account the fact that there is at least one cause of error in each unparsable sentence. Instead, Sagot and de la Clergerie (2006) defines a suspicion rate for n-grams which takes into account the number of occurrences of a given word form and iteratively defines the suspicion rate of each word form in a sentence based on the suspicion rate of this word form in the corpus.

Sagot and de la Clergerie (2006) defines the local and the global suspicion rates of a form as follows. Let $S_{f,t}$ be the suspicion rate of the form e in sentence t (local) and S_f, the mean suspicion rate of e in the overall corpus (global). S_f and $S_{f,t}$ are iteratively defined as follows:

$$
\begin{aligned}
S^0_{f,t} &= \frac{error(t)}{count(t)} \\
S^{n+1}_f &= smooth(count(f), \frac{\sum_{t \in T} count(f|t) S^n_{f,t}}{count(f)}) \\
S^{n+1}_{f,t} &= error(t) \frac{S^{n+1}_f}{\sum_{f' \in F} count(f'|t) S^{n+1}_{f'}}
\end{aligned}
$$

where T is the corpus of sentences; F is the set of suspicious forms; $error(t)$ is a function which return 1 if sentence t fails or 0 otherwise; $count(f|t)$ returns the count of the suspicious form f in the sentence t; $count(t)$ returns the count of all suspicious forms in the sentence t; $count(f)$ returns the count of the suspicious form f in the corpus; and $smooth(x, y)$ is a smoothing function defined as $smooth(x, y) = \lambda y + (1 - \lambda)S'$ where $\lambda = 1 - e^{-\beta x}$ and S' is the mean suspicion rate of the corpus and defined as $S' = \frac{\sum_{t \in T} error(t)}{count(F)}$ where $count(F)$ returns the count of all suspicious forms in the corpus.

Sagot and de la Clergerie (2006) aim to identify suspicious forms in the corpus along with suspicious forms for each sentence which fails. In our experiment, we are also more interested in identifying suspicious forms in the corpus, hence our algorithms build upon [van Noord, 2004] rather than [Sagot and de la Clergerie, 2006].

4.2.1.3 Generalized Suspicious Forms

Previous approaches has focused on n-grams of words as their suspicious forms. de Kok *et al.* (2009) combined the iterative error mining proposed by Sagot and

de la Clergerie (2006) with expansion of forms to n-grams of words and POS tags of arbitrary length. An inherent problem of incorporating longer n-grams is data sparseness. The proposed method takes sparseness into account, producing n-grams that are as long as necessary to identify problematic forms, but not longer. A longer n-gram is only considered if its suspicion rate if higher than each of its subgrams.

4.2.2 Error Mining for Generation

Not much work has been done on mining the results of surface realisers. Nonetheless, Gardent and Kow (2007a) describes an error mining approach which works on the output of surface realisation (the generated sentences), manually separates correct from incorrect output and looks for derivation items which systematically occur in incorrect output but not in correct ones. In contrast, our approach works on the input to surface realisation, automatically separates correct from incorrect items using surface realisation and targets the most likely sources of errors rather than the absolute ones.

4.2.3 Discussion

In this chapter, we propose two algorithms for error mining. In the best of our knowledge, they are first for mining a surface realiser for undergeneration. Indeed, apart from [Gardent and Kow, 2007a], most previous work on surface realisation evaluation has focused on evaluating the performance and the coverage of surface realisers. Approaches based on reversible grammars [Carroll *et al.*, 1999a] have used the semantic formulae output by parsing to evaluate the coverage and performance of their realiser; similarly, Gardent *et al.* (2010) developed a tool called GenSem which traverses the grammar to produce flat semantic representations and thereby provide a benchmark for performance and coverage evaluation. In both cases however, because it is produced using the grammar exploited by the surface realiser, the input produced can only be used to test for overgeneration (and performance) . Callaway (2003) avoids this shortcoming by converting the Penn Treebank to the format expected by his realiser. However, this involves identifying the mismatches between two formats much like symbolic systems did in the Generation Challenge SR Task [Belz *et al.*, 2011]. The error mining approaches we propose help identifying such mismatches automatically.

To summarise, our approaches differ from these previous approaches in several ways. First, our first proposal allows error mining to be performed on trees. Second, our second proposal extends the first proposal and structures the output of error min-

ing into a suspicion tree. Third, these approaches can be parameterised to use any combination of POS tag, dependency and/or word form information. And fourth, they are applied to generation input rather than parsing output. Typically, the input to surface realisation is a structured representation (i.e., a flat semantic representation, a first order logic formula or a dependency tree) rather than a string. Mining these structured representations thus permits identifying causes of undergeneration in surface realisation systems.

4.3 Error Mining on Dependency Trees

In this section, we propose an approach that supports error mining on trees. We adapt an existing algorithm [Chi *et al.*, 2004] for frequent subtree mining which we then use to mine the SR Task dependency trees and identify the most likely causes of generation failure. We start with a brief overview of the HybridTreeMiner algorithm [Chi *et al.*, 2004], a complete and computationally efficient algorithm for discovering frequently occurring subtrees in a database of labelled unordered trees (Section 4.3.1). In Section 4.3.2, we show how to adapt this algorithm to mine the SR dependency trees for subtrees with high suspicion rate. Finally, Section 4.3.3 shows how it permits mining the data for tree patterns of arbitrary size using different types of labelling information (POS tags, dependencies, word forms and any combination thereof).

4.3.1 Mining Trees using HybridTreeMiner

Mining for frequent subtrees is an important problem that has many applications such as XML data mining, web usage analysis and RNA classification. The HybridTreeMiner (HTM) algorithm [Chi *et al.*, 2004] provides a complete and computationally efficient method for discovering frequently occurring subtrees in a database of labelled unordered trees and counting them. We now sketch the intuition underlying this algorithm. We refer the reader to [Chi *et al.*, 2004] for a more complete definition. HybridTreeMiner

Given a set of trees T, the HybridTreeMiner algorithm proceeds in two steps. First, the unordered labelled trees contained in T are converted to a canonical form called BFCF (Breadth-First Canonical Form). In that way, distinct instantiations of the same unordered trees have a unique representation. Second, the subtrees of the BFCF trees are enumerated in increasing size order using two tree operations called join and extension and their support (the number of trees in the database that contains each subtree) is recorded. In effect, the algorithm builds *an enumeration*

tree whose nodes are the possible subtrees of T and such that, at depth d of this enumeration tree, all possible frequent subtrees consisting of d nodes are listed.

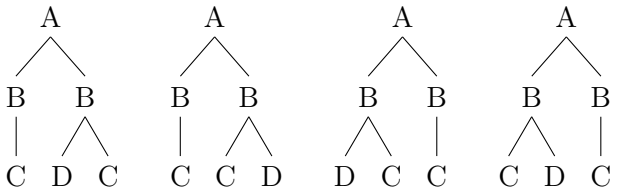


Figure 4.1: Unordered labelled trees with the rightmost in Breadth-First Canonical Form.

The BFCF canonical form of an unordered tree is an ordered tree t such that t has the smallest breath-first canonical string (BFCS) encoding according to lexicographic order. The BFCS encoding of a tree is obtained by breadth-first traversal of the tree, recording the string labelling each node, "$" to separate siblings with distinct parents and "#" to represent the end of the tree. We assume that "#" sorts greater than "$" and both sort greater than any other alphabetic symbols in node labels. For instance, the BFCS encodings of the four trees shown in Figure 4.1 are "ABBC$DC#", "A$BBCCD#", "ABBDC$C#" and "A$BBCDC#" respectively. Hence, the rightmost tree is the BFCF of all four trees.

The join and extension operations used to iteratively enumerate subtrees are depicted in Figure 4.2 and can be defined as follows.

- A *leg* is a leaf of maximal depth.
- **Extension:** Given a tree t of height h_t and a node n, extending t with n yields a tree t' (a child of t in the enumeration tree) with height $h_{t'}$ such that n is a child of one of t's legs and $h_{t'}$ is $h_t + 1$.
- **Join:** Given two trees t_1 and t_2 of same height h differing only in their rightmost leg and such that t_1 sorts lower than t_2, joining t_1 and t_2 yields a tree t' (a child of t_1 in the enumeration tree) of same height h by adding the rightmost leg of t_2 to t_1 at level $h - 1$.

For the set of trees T, the HybridTreeMiner algorithm initialises the enumeration tree with subtrees with one and two nodes. The join and extension operations are then applied on the BFCF canonical forms of these subtrees to produce subtrees with three nodes. This process is repeated until we produce all trees in T. The nodes (subtrees) in the enumeration trees are ordered with respect to their BFCF canonical forms. This allows us to apply the join and extension operations in an efficient manner and to discover all subtrees.

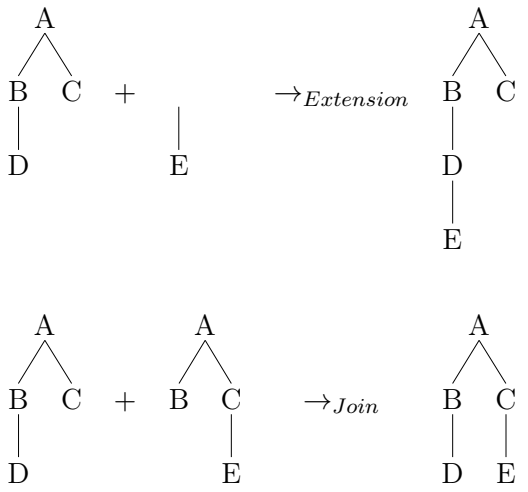


Figure 4.2: HybridTreeMiner: Join and Extension operations.

To support counting, the algorithm additionally records for each subtree a list (called *occurrence list*) of all trees in which this subtree occurs and of its position in the tree (represented by the list of tree nodes mapped onto by the subtree). Thus for a given subtree t, the *support* of t is the number of elements in that list. Occurrence lists are also used to check that trees that are combined occur in the data. For the join operation, the subtrees being combined must occur in the same tree at the same position (the intersection of their occurrence lists must be non-empty and the tree nodes must match except the last node). For the extension operation, the extension of a tree t is licensed for any given occurrence in the occurrence list only if the planned extension maps onto the tree identified by the occurrence.

4.3.2 Mining Dependency Trees for Errors

We develop an algorithm (called ErrorTreeMiner, ETM) which adapts the HybridTreeMiner algorithm to mine sources of generation errors in the Generation Challenge SR shallow input data . The main modification is that instead of simply counting trees, we want to compute their suspicion rate. Following de Kok *et al.* (2009), we take the suspicion rate of a given subtree t to be the proportion of cases where t occurs in an input tree for which generation fails:

ErrorTreeMiner

$$Sus(t) = \frac{count(t|\text{FAIL})}{count(t)}$$

where $count(t)$ is the number of occurrences of t in all input trees and $count(t|\text{FAIL})$

is the number of occurrences of t in input trees for which no output was produced.

Algorithm 2 ErrorTreeMiner($D, minsup$)

```
Note: D consists of D_fail and D_pass
F_1 ← {Frequent 1-trees}
F_2 ← ∅
for i ← 1, ..., |F_1| do
  for j ← 1, ..., |F_1| do
    q ← f_i plus leg f_j
    if Noord-Validation(q, minsup) then
      F_2 ← F_2 ∪ q
    end if
  end for
end for
F ← F_1 ∪ F_2
PUSH: sort(F_2) → L_Queue
Enum-Grow(L_Queue, F, minsup)
return F
```

Since we work with subtrees of arbitrary length, we also need to check whether constructing a longer subtree is useful that is, whether its suspicion rate is equal or higher than the suspicion rate of any of the subtrees it contains. In that way, we avoid computing all subtrees (thus saving time and space). As noted in ⌊de Kok *et al.*, 2009⌋, this also permits bypassing *suspicion sharing* that is the fact that, if t_1 is the cause of a generation failure, and if t_1 is contained in larger trees t_2 and t_3, then all three trees will have high suspicion rate making it difficult to identify the actual source of failure namely t_1. Because we use a milder condition however (we accept bigger trees whose suspicion rate is *equal* to the suspicion rate of any of their subtrees), some amount of suspicion sharing remains. As we shall see in Section 4.3.3.3, relaxing this check allows us to extract frequent larger tree patterns and thereby get a more precise picture of the *context* in which highly suspicious items occur.

Finally, we only keep subtrees whose support is above a given threshold where the support $Sup(t)$ of a tree t is defined as the ratio between the number of times it occurs in an input for which generation fails and the total number of generation failures:

$$Sup(t) = \frac{count(t|\text{FAIL})}{count(FAIL)}$$

The modified algorithm we use for error mining is given in Algorithms 2, 3 and 4. It can be summarised as follows.

Algorithm 3 Enum-Grow($L_{Queue}, F, minsup$)

while $L_{Queue} \neq empty$ **do**
  POP: $pop(L_{Queue}) \rightarrow C$
  for $i \leftarrow 1, ..., |C|$ **do**
    ◇The join operation
    $J \leftarrow \emptyset$
    for $j \leftarrow i, ..., |C|$ **do**
      $p \leftarrow join(c_i, c_j)$
      if Noord-Validation($p, minsup$) **then**
        $J \leftarrow J \cup p$
      end if
    end for
    $F \leftarrow F \cup J$
    PUSH: $sort(J) \rightarrow L_{Queue}$
    ◇The extension operation
    $E \leftarrow \emptyset$
    for possible leg l_m of c_i **do**
      for possible new leg $l_n (\in F_1)$ **do**
        $q \leftarrow$ extend c_i with l_n at position l_m
        if Noord-Validation($q, minsup$) **then**
          $E \leftarrow E \cup q$
        end if
      end for
    end for
    $F \leftarrow F \cup E$
    PUSH: $sort(E) \rightarrow L_{Queue}$
  end for
end while

Algorithm 4 Noord-Validation($t_n, minsup$)

Note: t_n, tree with n nodes
if $Sup(t_n) \geq minsup$ **then**
  if $Sus(t_n) \geq Sus(t_{n-1}), \forall t_{n-1}$ in t_n **then**
    return true
  end if
end if
return false

First, dependency trees are converted to Breadth-First Canonical Form whereby lexicographic order can apply to the word forms labelling tree nodes, to their part of speech, to their dependency relation or to any combination thereof[8].

[8]For convenience, the dependency relation labelling the edges of dependency trees is brought down to the daughter node of the edge.

Next, the algorithm iteratively enumerates the subtrees occurring in the input data in increasing size order and associating each subtree t with two occurrence lists namely, the list of input trees in which t occurs and for which generation was successful (PASS(t)); and the list of input trees in which t occurs and for which generation failed (FAIL(t)).

This process is initiated by building trees of size one (i.e., one-node tree) and extending them to trees of size two. It is then continued by extending the trees using the join and extension operations. As explained in Section 4.3.1 above, join and extension only apply provided the resulting trees occur in the data (this is checked by looking up occurrence lists). Each time an n-node tree t_n, is built, it is checked that (i) its support is above the set threshold and (ii) its suspicion rate is higher than or equal to the suspicion rate of all $(n-1)$-node subtrees of t_n.

In sum, the ETM algorithm differs from the HTM algorithm in two main ways. First, while HTM explores the enumeration tree depth-first, ETM proceeds breadth-first to ensure that the suspicion rate of (n-1)-node trees is always available when checking whether an n-node tree should be introduced. Second, while the HTM algorithm uses support to prune the search space (only trees with a minimum support bigger than the set threshold are stored), the ETM algorithm drastically prunes the search space by additionally checking that the suspicion rate of all subtrees contained in a new tree t is smaller or equal to the suspicion rate of t . As a result, while ETM looses the space advantage of HTM by a small margin[9], it benefits from a much stronger pruning of the search space than HTM through suspicion rate checking. In practice, the ETM algorithm allows us to process e.g., all NP chunks of size 4 and 6 present in the SR data (roughly 60,000 trees) in roughly 20 minutes on a single core PC.

4.3.3 Error Analysis

In this section, we show that our tree mining algorithm permits identifying not only errors in the grammar and the lexicon used by generation but also a few idiosyncrasies/errors (inconsistencies, missing information and questionable dependency structures) in the input data as well as mismatches between the structures contained in the SR input and the input structures expected by our generator. The latter is an important point since, for symbolic approaches, a major hurdle to participation in the SR challenge is known to be precisely these mismatches i.e., the fact that the input provided by the SR task fails to match the input expected by the

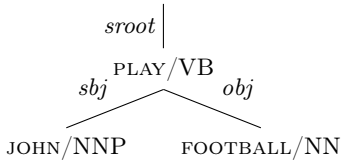
[9]ETM needs to store all (n-1)-node trees in queues before producing n-node trees.

symbolic generation systems [Belz *et al.*, 2011; Rajkumar *et al.*, 2011]. One feature of our approach is that it permits mining the data for tree patterns of arbitrary size using different types of labelling information (POS tags, dependencies, word forms and any combination thereof).

In what follows, in Section 4.3.3.1, we recap the input data and the generator. In the later sections, we show how our error mining algorithm permits mining the SR data for tree patterns of arbitrary size using different types of labelling information.

4.3.3.1 Experiment Setup

As described in the introduction, using the input data provided by the Generation Challenge SR Task, we applied the error mining algorithm described in the preceding section to debug and extend our symbolic surface realiser. We have discussed our input the shallow dependency trees from the Generation Challenge SR Task in Chapter 2 and the symbolic surface realiser in Chapter 3 in detail.



WORD: (PLAY, (JOHN), (FOOTBALL))
POS: (VB, (NNP), (NN))
dep: (*sroot*, (*sbj*), (*obj*))
WORD/**POS:** (PLAY/VB, (JOHN/NNP), (FOOTBALL/NN))
dep-**POS:** (*sroot*-VB, (*sbj*-NNP), (*obj*-NN))

Figure 4.3: An example shallow dependency tree from the SR Task and the corresponding representations used for error mining.

The dataset to be error mined are these shallow dependency trees (Figure 4.3) provided by the SR Task organisers and used as input for surface realisation. In this chapter, we represent a tree by an n-tuple with the root node of the tree as its first element followed by $(n-1)$ elements (tuples) representing its dependent subtrees. Dependency relations are lowered to the corresponding daughter node. Our error mining algorithms consider as suspicious forms, subtrees labelled with arbitrary conjunctions of lemmas (WORD), part-of-speech tags (POS), dependency relations (*dep*). For example, when the suspicious forms only consider lemma information, the tree shown in Figure 4.3 could be represented as (PLAY, (JOHN), (FOOTBALL)).

The data used for generation is preprocessed whereby named entities and hyphenated words are grouped into a single word and punctuation is removed so as to first focus on lexical and grammatical issues.

In the beginning, the coverage of the generator was very low on large sentences (Chapter 3). Hence, to facilitate error analysis, we first chunked the input data in

NPs, PPs and Clauses and performed error mining on the resulting sets of data. The chunking was performed by retrieving from the Penn Treebank (PTB), for each phrase type, the yields of the constituents of that type and by using the alignment between words and dependency tree nodes provided by the organisers of the SR Task. For instance, given the sentence *"The most troublesome report may be the August merchandise trade deficit due out tomorrow"*, the NPs *"The most troublesome report"* and *"the August merchandise trade deficit due out tomorrow"* will be extracted from the PTB and the corresponding dependency structures from the SR Task data.

Using this chunked data, we then ran the generator on the corresponding SR Task dependency trees and stored separately, the input dependency trees for which generation succeeded and the input dependency trees for which generation failed. Using information provided by the generator, we then removed from the failed data, those cases where generation failed either because a word was missing in the lexicon or because a TAG tree/family was missing in the grammar but required by the lexicon and the input data. These cases can easily be detected using the generation system and thus do not need to be handled by error mining.

Finally, we performed error mining on the data using different minimal support thresholds, different display modes (sorted first by size and second by suspicion rate *vs* sorted by suspicion rate) and different labels (part of speech, words and part of speech, dependency, dependency and part of speech).

4.3.3.2 Mining on single labels (Word Form, POS Tag or Dependency)

Mining on a single label permits (i) assessing the relative impact of each category in a given label category and (ii) identifying different sources of errors depending on the type of label considered (POS tag, dependency or word form).

Mining on POS tags Table 4.1 illustrates how mining on a single label (in this case, POS tags) gives a good overview of how the different categories in that label type impact generation: two POS tags (POSS and CC) have a suspicion rate of 0.99 indicating that these categories almost always lead generation to fail. Other POS tag with much lower suspicion rate indicate that there are unresolved issues with, in decreasing order of suspicion rate, cardinal numbers (CD), proper names (NNP), nouns (NN), prepositions (IN) and determiners (DT).

The highest ranking category (POSS[10]) points to a mismatch between the rep-

[10]In the Penn Treebank, the POSS tag is the category assigned to possessive *'s*. In fact, the actual representation is POS not POSS. We renamed it to avoid confusion with POS as an abbreviation for part-of-speech.

POS	Sus	Sup	Fail	Pass
POSS	0.99	0.38	3237	1
CC	0.99	0.21	1774	9
CD	0.39	0.16	1419	2148
NNP	0.35	0.32	2749	5014
NN	0.30	0.81	6798	15663
IN	0.30	0.16	1355	3128
DT	0.09	0.12	1079	10254

Table 4.1: Error Mining on POS tags with frequency cutoff 0.1 and displaying only trees of size 1 sorted by decreasing suspicion rate (Sus).

resentation of genitive NPs (e.g., *John's father*) in the SR Task data and in the grammar. While our generator expects the representation of 'John's father' to be (FATHER, ("S", (JOHN))), the structure provided by the SR Task is (FATHER, (JOHN, ("S"))). Hence whenever a possessive appears in the input data, generation fails. This is in line with [Rajkumar *et al.*, 2011]'s finding that the logical forms expected by their system for possessives differed from the shared task inputs.

The second highest ranked category is CC for coordinations. In this case, error mining unveils a bug in the grammar trees associated with conjunction which made all sentences containing a conjunction fail. Because the grammar is compiled out of a strongly factorised description, errors in this description can propagate to a large number of trees in the grammar. It turned out that an error occurred in a class inherited by all conjunction trees thereby blocking the generation of any sentence requiring the use of a conjunction.

Next but with a much lower suspicion rate come cardinal numbers (CD), proper names (NNP), nouns (NN), prepositions (IN) and determiners (DT). We will see below how the richer information provided by mining for larger tree patterns with mixed labelling information permits identifying the contexts in which these POS tags lead to generation failure.

Mining on Word Forms Because we remove from the failure set all cases of errors due to a missing word form in the lexicon, a high suspicion rate for a word form usually indicates a missing or incorrect lexical entry: the word is present in the lexicon but associated with either the wrong POS tag and/or the wrong TAG tree/family. To capture such cases, we therefore mine not on word forms alone but on pairs of word forms and POS tag. In this way, we found for instance, that cardinal numbers induced many generation failures whenever they were categorised

as determiners but not as nouns in our lexicon. As we will see below, larger tree patterns help identify the specific contexts inducing such failures.

One interesting case stood out which pointed to idiosyncrasies in the input data: The word form $ (Sus=1) was assigned the POS tag $ in the input data, a POS tag which is unknown to our system and not documented in the SR Task guidelines. The SR guidelines specify that the Penn Treebank tagset is used modulo the modifications which are explicitly listed. However for the $ symbol, the Penn treebank used SYM as a POS tag and the SR Task $, but the modification is not listed. Similarly, while in the Penn treebank, punctuations are assigned the SYM POS tag, in the SR data "," is used for the comma, "(" for an opening bracket and so on.

Mining on Dependencies When mining on dependencies, suspects can point to syntactic constructions (rather than words or word categories) that are not easily spotted when mining on words or parts of speech. Thus, while problems with coordination could easily be spotted through a high suspicion rate for the CC POS tag, some constructions are linked neither to a specific POS tag nor to a specific word. This is the case, for instance, for apposition (*appo*) with a suspicion rate of 0.19 (286F/1148P) identified as problematic. Similarly, a high suspicion rate (0.54, 183F/155P) on the *tmp* dependency indicates that temporal modifiers are not correctly handled either because of missing or erroneous information in the grammar or because of a mismatch between the input data and the fomat expected by the surface realiser.

Interestingly, the underspecified dependency relation *dep* which is typically used in cases for which no obvious syntactic dependency comes to mind shows a suspicion rate of 0.61 (595F/371P).

4.3.3.3 Mining on Trees of Arbitrary Sizes and Complex Labelling Patterns

While error mining with tree patterns of size one permits ranking and qualifying various sources of errors, larger patterns often provide more detailed contextual information about these errors. For instance, Table 4.1 shows that the CD POS tag has a suspicion rate of 0.39 (1419F/2148P). The larger tree patterns identified below permits a more specific characterization of the context in which this POS tag co-occurs with generation failure:

TP1 (CD, (IN),(RBR)) *more than 10*
TP2 (IN, (CD)) *of 1991*
TP3 (NNP, (CD)) *November 1*
TP4 (CD, (NNP, (CD))) *Nov. 1, 1997*

Two patterns clearly emerge: a pattern where cardinal numbers are parts of a date (tree patterns TP2-TP4) and a more specific pattern (TP1) involving the comparative construction (e.g., *more than 10*). All these patterns in fact point to a missing category for cardinals in the lexicon: they are only associated with determiner TAG trees, not nouns, and therefore fail to combine with prepositions (e.g., *of 1991, than 10*) and with proper names (e.g., *November 1*).

For proper names (NNP), dates also show up because months are tagged as proper names (TP3,TP4) as well as addresses (TP5):

TP5 (NNP, (",",(",")) *Brooklyn, n.y.,*

For prepositions (IN), we find, in addition to (TP1-TP2), the following two main patterns:

TP6 (DT, (IN)) *those with, some of*
TP7 (RB, (IN)) *just under, little more*

Pattern TP6 points to a missing entry for words such as *those* and *some* which are categorised in the lexicon as determiners but not as nouns. TP7 points to a mismatch between the SR data and the format expected by the generator: while the latter expects the structure (IN, (RB)), the input format provided by the SR Task is (RB, (IN)).

4.4 Error Mining with Suspicion Trees

In this section, we propose another error mining algorithm which structures the output of error mining into a *suspicion tree* making explicit both the ranking of the main distinct error cases and their subcases. The suspicion tree is a binary tree structure whose internal nodes are labelled with suspicious forms and whose leaf nodes represent the clusters of error mined data grouped according to the suspicious forms characterizing their elements. Like in a decision tree, each cluster in the suspicion tree is characterized by the set of attributes (suspicious forms) labelling its ancestors; and the tree itself represents a disjunction of mutually exclusive error cases.

We start with introducing the Suspicion Tree Algorithm in Section 4.4.1. In essence, this algorithm adapts Quinlan (1986)'s ID3 algorithm to build a suspicion tree such that the clusters obtained group together sets of input data that share

similar sources of failure (called *suspicious forms*); and the attributes labelling these clusters are the suspicious forms indicating which are these most likely causes of failure. Later in the section, we do a comparison with the ID3 algorithm (Section 4.4.2). In Section 4.4.3, we present the complexity analysis of our algorithm.

4.4.1 The Suspicion Tree Algorithm

As mentioned above, our error mining algorithm resembles Quinlan (1986)'s ID3 decision tree learning algorithm, in that it recursively partitions the data by first, selecting the attribute (here, a suspicious form) that best divides the data into more homogeneous subsets (*attribute selection*) and second, using this attribute to split the data into two subsets: a subset containing that attribute and a subset excluding that attribute (*dataset division*).

In what follows, we define the metric used to recursively select a suspicious form and partition the data, namely the *Suspicion Score* metric. We specify the termination conditions. We illustrate by means of examples how suspicion trees help structure the output of error mining. And we contrast the suspicion tree algorithm with Quinlan (1986)'s ID3 decision tree learning algorithm.

The Suspicion Score Metrics. Let $\mathbb{D}$ be the dataset to be error mined and $\mathbb{F}$ be the set of attributes used to partition the data. Here, $\mathbb{D}$ is a set of dependency trees provided for the Surface Realisation Task by the Generation Challenge; and $\mathbb{F}$ is the set of subtrees of $\mathbb{D}$ whose frequency is above a given threshold. This algorithm builds over the algorithm shown in Section 4.3 in that we use the Hybrid Tree Miner algorithm [Chi *et al.*, 2004] to compute the set of frequent subtrees that are present in $\mathbb{D}$.

Let $\mathbb{D}$ be divided into two disjoint sets: PASS is the set of input dependency trees $t^P \in \mathbb{D}$ for which the generation system succeeds; and FAIL is the set of input dependency trees $t^F \in \mathbb{D}$ for which the generation system fails. Given these two sets, we define two suspicion rates of a form[11] f: **F-Suspicion rate** and **P-Suspicion rate**.

The **F-Suspicion rate of** f is defined as the proportion of cases where f occurs in an input dependency tree t for which the generation system fails:

$$\text{Fail}(f) = \frac{count(f|\text{FAIL})}{count(f)}$$

[11] Following the previous Section 4.3, a suspicious form is a subtree present in the dataset FAIL.

count(f) is the number of input dependency trees containing f and *count*(f|FAIL) is the number of input dependency trees containing f for which generation failed.

Conversely, the **P-Suspicion rate of a form** f is defined as the proportion of cases not containing f and for which generation succeeds (*count*($\neg f$) is the number of input dependency trees where f is absent and *count*($\neg f$|PASS) is the number of input dependency trees not containing f for which generation succeeds):

$$\text{Pass}(\neg f) = \frac{count(\neg f|\text{PASS})}{count(\neg f)}$$

Having the F-Suspicion and P-Suspicion rates defined, the **suspicion score** $\text{S}_{score}(f)$ of a form $f \in \mathbb{F}$ is then defined as follows:

Suspicion Score Metric

$$\text{S}_{score}(f) = \frac{1}{2}(\text{Fail}(f) * \ln count(f) + \text{Pass}(\neg f) * \ln count(\neg f))$$

Intuitively, this metric captures the degree to which a form is associated with failure: it is high whenever a form f is often present in data associated with failure (high *F(ail)-Suspicion*, Fail(f)) and/or when it is often absent in data associated with success (high *P(ass)-Suspicion*, Pass($\neg f$)).

Attribute Selection, Dataset Division and Termination. The suspicion tree algorithm selects at each step of the tree building process, the form f with highest suspicion score i.e. the form such that, in the current dataset, most input dependency trees that contain f fail to be generated; and most input dependency trees that excludes f lead to successful generation.

Based on this selected f, the current dataset is divided into two subsets: the set of instances which contain f and the set of instances which exclude f.

The form selection and dataset division process are called recursively on the new subsets until (i) the obtained set of input dependency trees is fully homogeneous (all input dependency trees in that set lead to either successful or unsuccessful generation); (ii) all forms have been processed; or (iii) the depth upper bound is reached.

Example. Figure 4.4 shows an abstract suspicion tree which illustrates how suspicion trees help structuring the output of error mining. Internal nodes of the suspicion tree are labeled with suspicious forms and leaves indicate the number of input dependency trees in the current data set S_i for which generation succeeds (n_{p_i}); and for which generation fails (n_{f_i}). When the sources of errors are clearly identifiable, n_{p_i} will be low, n_{f_i} will be high and the rightmost leaf (f_4) will have a low n_{f_i}. The right frontier highlights the relative importance of the main distinct error cases while

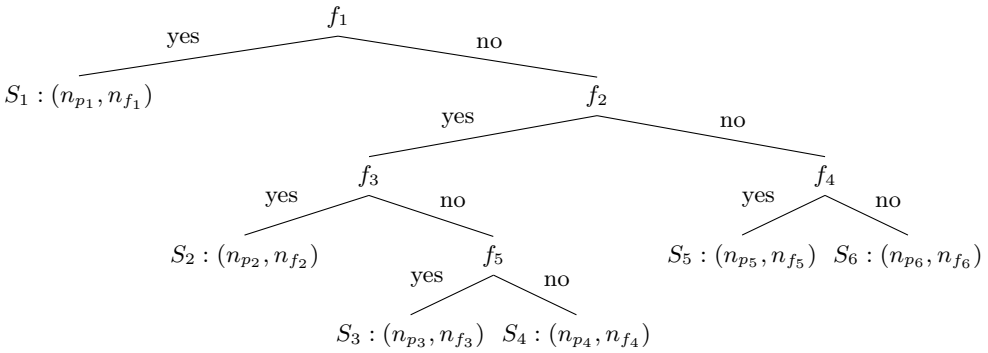


Figure 4.4: An example *Suspicion Tree*.

subtrees indicate how a given error case divides into smaller more specific cases. The branches of the tree also indicate the combinations of forms that frequently cooccur in failure cases.

More specifically, the root f_1 of this *suspicion tree* is the most suspicious form present in the corpus $\mathbb{D}$. Starting from the root, following the edges with label "no" (the *right-frontier* of the tree i.e., f_1, f_2 and f_4) yields the ranked list of suspicious forms present in $\mathbb{D}$ by decreasing order of suspicion. Following branches yields datasets labeled with sets (conjunctions) of suspicious forms. For example, the set S_2 with n_{p_2} of pass input dependency trees and n_{f_2} of failed input dependency trees has f_2 and f_3 as their top ranked suspicious forms. The *suspicion tree* also displays the relative ranking of the suspicious forms. For example, the set $(S_2 \cup S_3 \cup S_4)$ has f_2 as its most suspicious form, and f_3, f_5 as its next two most suspicious forms. Moreover, most of the instances in S_1, S_4 and S_5 fail because of a single form namely, f_1, f_2 and f_4 respectively.

4.4.2 Suspicion Tree Algorithm vs. ID3 Algorithm

There are two main differences between Quinlan (1986)'s ID3 decision tree learning algorithm and the suspicion tree construction algorithm.

First, the suspicion tree construction algorithm allows for stronger pruning and termination conditions (Section 4.4.3). In this way, only the most relevant suspicious forms are displayed thereby facilitating error analysis.

Second, attribute selection is determined not by the information gain (IG) but by the suspicion score (S_{score}) metrics. Recall that the information gain[12] metrics aims

[12]Information gain (IG) is defined as $IG = H(S) - ((|S_{f_i}|/|S|) * H(S_{f_i}) + (|S_{\neg f_i}|/|S|) * H(S_{\neg f_i}))$ where $H(X)$ is the entropy of set X, S is the set, S_{f_i} is the set with the form f_i and $S_{\neg f_i}$ is the set without the form f_i. [Quinlan, 1986]

to identify the attributes which lead to more homogeneous classes. In the present case, the classes are either PASS (the inputs for which generation succeeds) or FAIL (the inputs for which generation fails). Thus the IG metrics will indifferently seek to identify attributes which predominantly associate either with a FAIL or with a PASS. There is no preference for either the FAIL or the PASS class. For error mining however, what is needed is to identify attributes which predominantly associate with the FAIL class. That is, we need a metric which permits identifying attributes which leads to classes that are homogeneous in terms of FAIL instances rather than homogeneous in terms of either FAIL or PASS instances. The example shown in Figure 4.5 illustrates the difference.

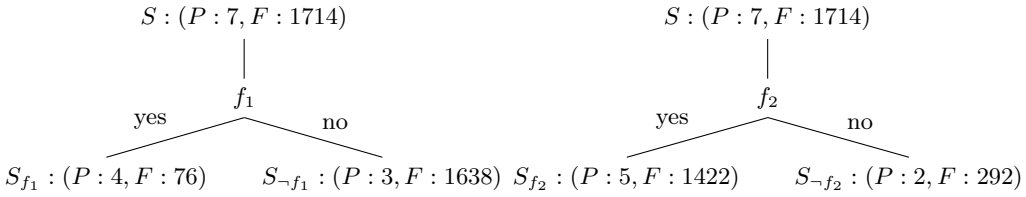


Figure 4.5: Attribute selection using Information Gain (left) and suspicion score (right).

In this example, we apply the IG and the S_{score} metrics to the same input data, a set containing 7 inputs associated with generation success and 1714 inputs associated with generation failure. While S_{score} selects f_2, an attribute which associates 1422 times with generation failure, IG selects f_1, an attribute which associate only 76 times with generation failure. In this case, the information gain metrics incorrectly select f_1 because its absence from the input, yields a numerically very homogeneous class in terms of generation failure. Indeed, the information gain of f_1 is close to but higher than the information gain of f_2 because the resultant subsets S_{f_i} and $S_{\neg f_i}$ are treated equally while computing the information gain.

4.4.3 Complexity Analysis and Extensions

Let n and m be the size of the dataset $\mathbb{D}$ and of the form set $\mathbb{F}$ respectively. Then, in the worst case, the suspicion tree will be of depth $O(log\ n)$ with $O(n)$ nodes. Each node chooses a suspicious form out of $O(m)$ forms. Thus the worst computational complexity for building the suspicion tree is $O(m\ n\ log\ n)$. But on average, the algorithm described in Section 4.4.1 performs much faster than this. The worst case happens when the forms used to classify the corpus into PASS and FAIL are not very discriminant i.e., when all suspicious forms are equally probable.

The algorithm for building the suspicion tree is directly proportional to the size of the set $\mathbb{F}$. Since $|\mathbb{F}|$ can be very large, this can be problematic. Indeed, in the error mining on sentences for parsing systems proposed in [Sagot and de la Clergerie, 2006], the authors indicate that, in order to remain computationally tractable, the approach must be restricted to n-grams of smaller size (unigrams and bigrams). The problem is accrued of course when considering tree shaped suspicious forms [Gardent and Narayan, 2012]. To abate this issue we propose two extensions to prune the suspicion tree.

First, we reduce the form space $\mathbb{F}$. Following a suggestion from [de Kok *et al.*, 2009], instead of considering all possible forms, we only consider those forms whose frequency is above a given threshold. We also account for *suspicion sharing* (i.e., the sharing of suspicion by several overlapping forms) by only considering a larger suspicious form if its suspicion rate is larger than the suspicion rate of all smaller forms it contains. These two extensions reduce the form space significantly and allow for an efficient building of the suspicion tree. To enumerate with these extensions, we use a complete and efficient algorithm described in Section 4.3 [Gardent and Narayan, 2012].

Second, we constrain the depth of the suspicion tree. Because error mining is a cyclic process, building the complete suspicion tree is usually unnecessary. The quantity of information processed in each cycle depends on the user but in general, the linguist will focus on the top suspicious forms, use these to improve the generator and rerun error mining on the improved results. The faster the error mining step is, the better this is for this development cycle. Considering this, we added an extra constraint over the depth of the suspicion tree. This depth limit permits pruning the suspicion tree and a faster improvement cycle. In our experiments, we used a depth limit of 10.

With these extensions, the enumeration process of suspicious forms takes 10-15 minutes for a dataset consisting of 123,523 trees on a 4-core PC. Building a suspicion tree for the same dataset takes about one minute.

4.4.4 Error Analysis

In this section, we report on the experiment we did using the suspicion tree algorithm described in the preceding section to detect and classify the most likely causes of failure when running a surface realiser on the Surface Realisation Task data. Moreover, we illustrate the impact of structuring the error mining output on how it helps handling errors in a linguistically meaningful way. We first describe the experimental setup (Section 4.4.4.1). We then illustrate by means of examples, how suspicion trees

better support error analysis than ranked lists proposed by previous error mining approaches (Section 4.4.4.2). In Section 4.4.4.3 we show how it helps handling errors in a linguistically meaningful way.

4.4.4.1 Experiment Setup

Dataset and Generation System The dataset to be error mined is the set of shallow dependency trees used as input for surface realisation. The data representation (n-tuples) and the generation system are identical to the one described in Section 4.3.3.1.

In Section 4.3.3.1, error mining was done on NP chunks only. In this experiment, however, we proceed in an incremental way and examine dependency trees in the SR data that correspond to NP and Sentences of increasing size. Here we report on error mining performed on NP-type dependency trees of sizes 4 (NP-4), 6 (NP-6) and all (NP-ALL), and S-type dependency trees of sizes 6 (S-6), 8 (S-8) and all (S-ALL) (where the size refer to the number of nodes/lemmas in the tree). As described in Section 4.3.3.1, the data used for generation is again preprocessed, generated and divided into two dataset, PASS and FAIL, for error mining.

Attributes The attributes used to partition the SR data are suspicious trees i.e., subtrees of the SR dependency trees whose frequency is above a given threshold. Following our previous error mining approach (described in Section 4.3), we allow for various views on errors by mining for forms labelled with lemmas only (WORD); with parts of speech (POS); with dependency relations (*dep*); with lemmas and parts of speech (WORD/POS); and with dependency relations and parts of speech (*dep*-POS) (cf. Figure 4.3).

Error Mining We iterate several times between error mining and performance improvement and applied the suspicion tree algorithm to both the NP and the S data. Iteration stops either when the results are perfect (perfect coverage and perfect BLEU score) or when the trees fail to be discriminative enough (low number of FAIL instances associated with the suspicion tree leaves).

4.4.4.2 Suspicion Trees vs. Ranked Lists

We now show how the overall structure of the suspicion tree (right frontier and subtrees) improves upon ranked lists when analysing the data. We then go on to show how subtrees in the suspicion tree permit differentiating between forms that are

suspicious in all contexts and require a single correction; forms that are suspicious in all contexts but require several corrections; and forms that are suspicious in some but not all contexts.

Figure 4.6 shows a top fragment of the suspicion tree obtained by error mining on NP-4. The node labels in this tree describe suspicious forms with part-of-speech information only. The leaves (p, f) represent the cluster with PASS (p) and FAIL (f) instances.

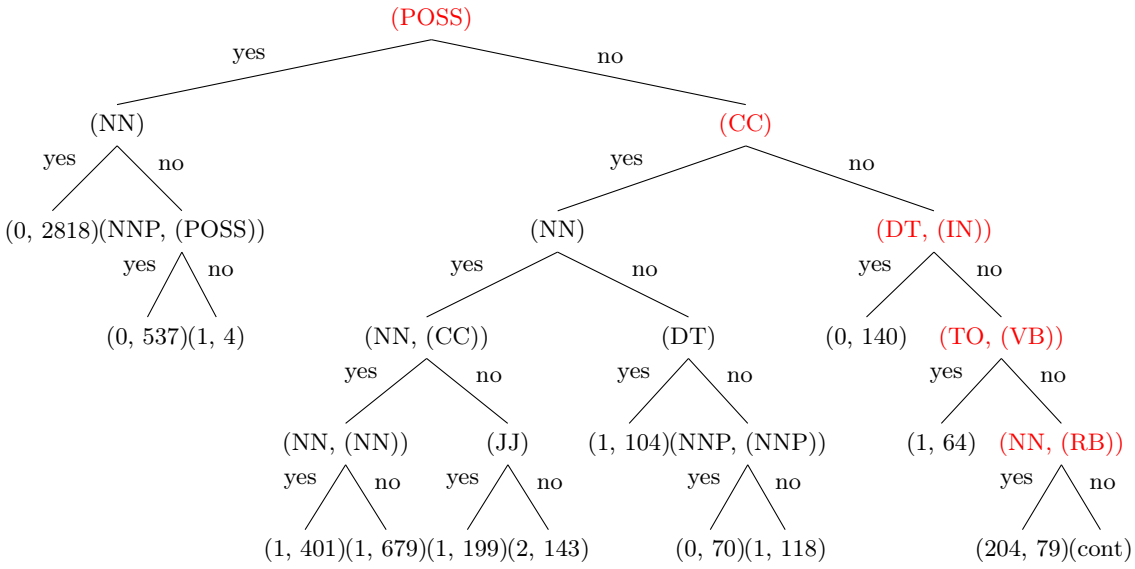


Figure 4.6: Suspicion tree for generation from the NP-4 data.

In that tree, the right frontier indicates that the main distinct suspicious forms are, in that order:

1. Possessive NPs (POSS is the part of speech tag assigned to possessive *'s*)
The suspicious form (POSS) points to a mismatch between the representation of genitive NPs ('*Oakland's thief*') in the SR Task data and in the grammar. We have discussed this before in the error analysis with tree mining algorithm in Section 4.3.3.2. To correct these cases, we implemented a rewrite rule that converts the SR representation of possessive NPs to conform with the format expected by our realiser.
2. NPs with coordination (CC with daughter node NN)
The second top right frontier node unveils a bug (conflicting feature values) in the grammar trees associated with NP conjunction (e.g., *Europe and the U.S.*) which made all sentences containing an NP conjunction fail. Again, we have discovered this before in the error analysis with tree mining algorithm in Section 4.3.3.2.

3. Determiners (DT) dominating a preposition (IN)
As we shall see below, this points to a discrepancy between the SR part of speech tag assigned to words like '*some*' in '*some of the audience*' and the part of speech tag expected by our generator. While in the SR data, such occurrences are labelled as determiners (DT), our generator expects these to be tagged as pronouns (PRP).
4. The complementizer *to* (TO) dominating a verb (VB)
As discussed below, this points to cases where the infinitival verb is a noun modifier and the input structure provided by the SR Task differs from that expected by our realiser.
5. Nouns (NN) dominating an adverb (RB)
This points to a discrepancy between the SR part of speech tag assigned to words like 'ALONE' in '*real estate alone*' and the part of speech tag expected by our generator. While in the SR data, such occurrences are labelled as adverbs (RB), our generator expects these to be tagged as adjectives (JJ).

In addition, for each node n on the right frontier, the subtree dominated by the yes-branch of n gives further information about the more specific forms that are subcases of the suspicious form labelling n.

1. (POSS)
2. (NNP, (POSS))
3. (CC)
4. (NN, (POSS))
5. (NN, (NNP, (POSS)))
6. (NN, (NN, (POSS)))
7. (NN, (CC))
8. (NNP, (NNP), (POSS))
9. (NN,(NNP,(NNP),(POSS)))
10. (NN, (NNP, (NNP)))
11. (CC, (JJ))
12. (JJ, (CC))
13. (NNP, (NNP, (POSS)))
14. (NN, (NN), (POSS))
15. (DT, (IN))
16. (JJ, (CC, (JJ)))
17. (NN, (CC), (NN))
18. (NN, (NNP), (POSS))
19. (TO, (VB))
20. (NN,(NNP,(POSS)),(NNP))
21. (NN, (NNP))
22. (NNP, (NNP))
23. (NN, (NN))
24. (NNP)
25. (NN)
26. (NN, (NNP), (NNP))
27. (VB)
28. (NN, (RB))
29. (PRP)
30. (DT)

Figure 4.7: Ranked list of suspicious forms for generation from the NP-4 data.

The suspicion tree gives a structured view of how the various suspicious forms relate. In comparison, the ranked lists produced by previous work are flat structures which may fail to adequately display this information. For instance, applying the error mining algorithm described in Section 4.3 [Gardent and Narayan, 2012] to the data used to produce the tree shown in Figure 4.6 yields the list shown in Figure 4.7. Contrary to the suspicion tree shown in Figure 4.6, this list fails to highlight the main culprits and the relations between the various suspicious forms. Thus the 5 main distinct suspects identified by the right frontier of the suspicion tree appears as 1st, 3rd, 15th, 19th and 28th in the ranked list. Furthermore, while subcases of the main

suspects are grouped in the yes-branch of these suspects in the suspicion tree, in the ranked list, they appear freely interspersed throughout. For example, suspicious forms involving the two main suspects in the suspicion tree approach (POSS and CC part-of-speech tags) are scattered throughout the list rather than grouped under the first two right frontier nodes respectively.

Also the stronger pruning conditions used for building the suspicion tree restrict the branch exploration as soon as homogeneous clusters are achieved. For a given dataset, it only explores those suspicious forms which are good enough to identify the problems causing the failure in that dataset. For example the data containing the suspicious form (POSS) is explored with 3 suspicious forms (POSS), (NN) and (NNP, (POSS)) in the suspicion tree shown in Figure 4.6 whereas in the ranked list shown in Figure 4.7, there are 11 suspicious forms associated with (POSS). In general, the number of forms displayed by the suspicion tree algorithm is much less than that of the ranked list ones thereby giving a clearer picture of the main culprits and of their subcases at each stage in the error mining/grammar debugging cycle.

4.4.4.3 Reading Error Types off the Tree Structure

For each node n labelled with suspicious form f_n in a suspicion tree, the subtree dominated by n gives detailed information about the possible contexts/causes for f_n. In what follows, we show how the suspicion tree algorithm permits distinguishing between three main types of suspicious forms namely, forms that are suspicious in all contexts and require a single correction; forms that are suspicious in all contexts but require several corrections; and forms that are suspicious in some but not all contexts.

Forms that are suspicious independently of context and require a single correction. When a suspicious form always leads to failure, the node labelled with that suspicious form has no subtree thereby indicating that all configurations including that suspicious form lead to generation failure independent of context.

Such cases are illustrated in Figure 4.8 which show two views (one with part of speech tag only, the other with words and parts of speech tags) of the suspicion tree obtained after addressing the two main causes of errors identified in the previous section. That is, a rewrite rule was applied to convert the SR representation of possessive NPs to conform with the format expected by our realiser ((POSS) suspicious form); and the grammar was corrected to generate for NP coordination ((CC) suspicious form).

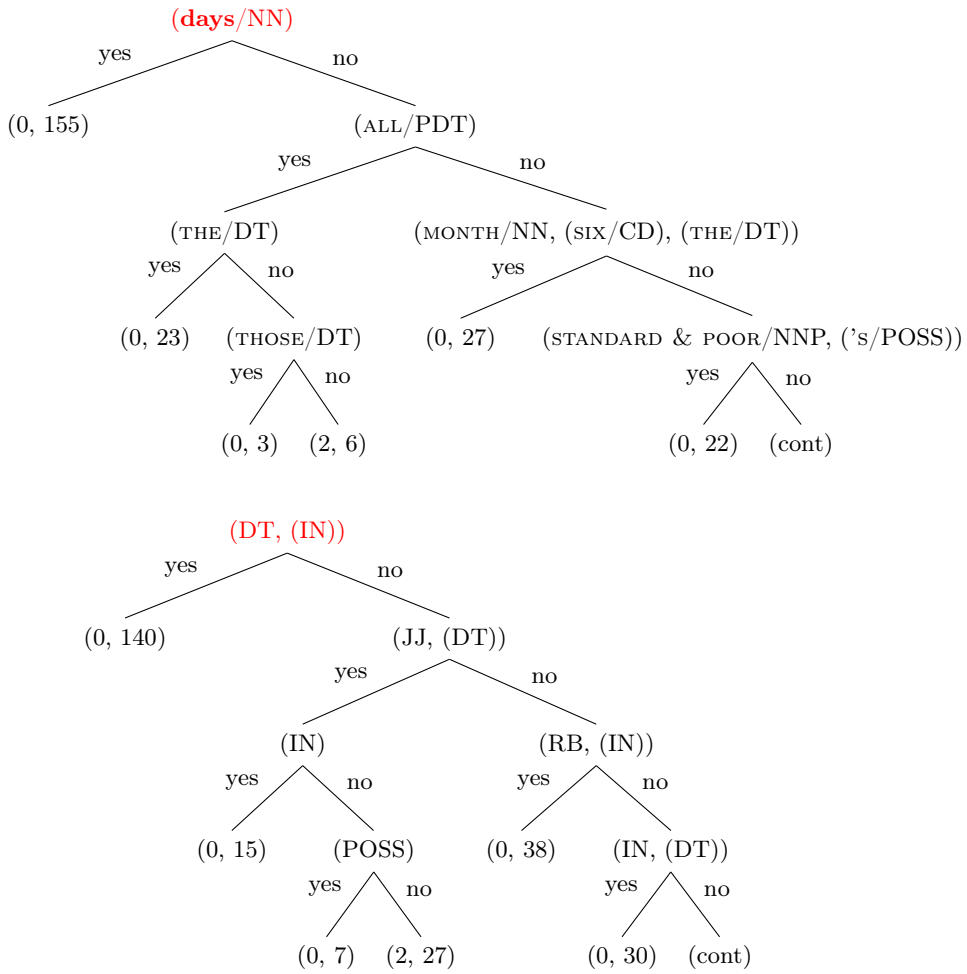


Figure 4.8: Suspicion Tree for (WORD/POS) (top) and (POS) (bottom) for generation from the NP-4 data (after fixing genitive and coordination cases).

In each of these two trees, the yes-branch of the root node has no subtree indicating that all input trees containing either the word form '*days*' with part of speech tag NN (DAYS/NN); or a determiner dominating a preposition (DT,(IN)) lead to generation failure.

The root node (DAYS/NN) of the suspicion tree shown on the top of Figure 4.8 points to a problem in the lexicon. Although DAYS/NN is present in the lexicon, it is not associated with the correct TAG family. We modified the entries corresponding to (DAYS/NN) in the lexicon to solve this problem.

As mentioned above, the root node (DT, (IN)) of the suspicion tree shown on the bottom in Figure 4.8 points to a part-of-speech tagging problem in the SR Data. Words like '*some*' or '*all*' followed by a preposition (e.g., *some of the audience*, *all of*

fiscal 1990, *those in other industries*) are assigned the determiner part of speech tag (DT) where our generator expects a pronoun (PRP) part-of-speech tag. To correct these cases, we implemented a rewrite rule that maps DT to PRP in the above specified context.

As these two examples illustrate, using different views (forms labelled with part of speech tags only *vs.* forms labelled with words and parts of speech) on the same data may help identifying problems at different levels. Both suspicion trees in Figure 4.8 are built for generation from same NP-4 dataset. The topmost tree (suspicious forms labelled with both lemma and part of speech information) helps identifying problems in the lexicon whereas the bottom tree (suspicious forms labelled with parts of speech only) points to problems in the input data.

We reran the generator on the modified input and the lexicon. The node (DAYS/NN) of Figure 4.8(top) and the node (DT, (IN)) of Figure 4.8(bottom) (also marked in red) disappeared in the newly generated suspicion trees.

Forms that are suspicious independent of context but require several corrections. It may be that a given form is almost always suspicious but that it occurs in different linguistic contexts requiring different corrections. In such cases, the suspicion tree will highlight these contexts. The root of the tree shown in Figure 4.9 is a case in point. The suspicious form (*im*-VB) describes subtrees whose head is related to a verb by the *im* dependency relation i.e., *infinitival verbs*. The subtrees (of the yes-branch) of that root further describe several syntactic configurations which are suspect and contain an infinitival verb. The node labelled with (*oprd*-TO) points to subcases where the infinitival verb is the complement of a control (7a[i]) or a raising verb (7a[ii]). The node labelled with (*im*-VB, (*prd*-JJ)) points to a subcase of that case namely that of an infinitival verb which is the complement of a control or a raising verb and subcategories for an adjectival complement e.g., (7b). Finally, the node labelled with (*nmod*-TO, (*im*-VB)) points to cases where the infinitival verb is a noun modifier (7c).

(7) a. (*oprd*-TO)

i *He will try to assuage the fears about finances.*
(TRY/VB, (*oprd*-TO/TO, (*im*-ASSUAGE/VB))

ii *Many of the morning session winners turned out to be losers.*
(TURN/VB, (*oprd*-TO/TO, (*im*-BE/VB, (*prd*-LOSER/NN)))

b. (*im*-VB, (*prd*-JJ))
Amex expects to be fully operational by tomorrow.
(EXPECT/VB, (*oprd*-TO/TO, (*im*-BE/VB, (*prd*-OPERATIONAL/JJ)))

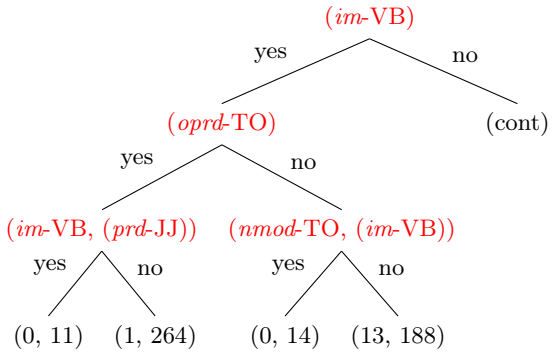


Figure 4.9: Suspicion tree (*dep*-POS) for generation from the S-6 data.

c. (*nmod*-TO, (*im*-VB))
The ability to trade without too much difficulty has steadily deteriorated.
(ABILITY/NN, (*nmod*-TO/TO, (*im*-TRADE/VB))

Although all of these cases are due to a mismatch between the SR Task dependency trees and the input expected by our realiser, they point to different input configurations requiring different modifications (rewritings) to ensure compatibility with the realiser. The structured information given by the suspicion tree provides a clear description of the main tree configurations that need to be rewritten to correct generation failures induced by infinitival verbs. We used this information to implement the rewrite rules required to resolve the identified mismatches.

Forms that are suspicious in some but not all contexts. The suspicion tree can also highlight forms that are suspicious in some but not all contexts. For instance, the right frontier of the suspicion tree in Figure 4.10 shows that the CD (cardinals) part of speech occurs in several suspicious forms namely, (IN, (CD)) (a preposition dominating a cardinal), (CD, (IN)) (a cardinal dominating a preposition), (CD, (CD)) (a cardinal dominating a cardinal), (CD, (DT)) (a cardinal dominating a determiner) and (CD, (RB)) (a cardinal dominating an adverb). Examples for these configurations and their subcases are given in (8).

(8) a. (IN, (CD))
the end of 1991
(END/NN, (THE/DT), (OF/IN, (1991/CD)))

b. (CD, (IN))
one of the authors
(ONE/CD, (OF/IN, (AUTHOR/NN, (THE/DT))))

c. (CD, (CD))
Nov. 1 , 1997
(1/CD, (1997/CD), (NOV./NNP), (,/SYS))

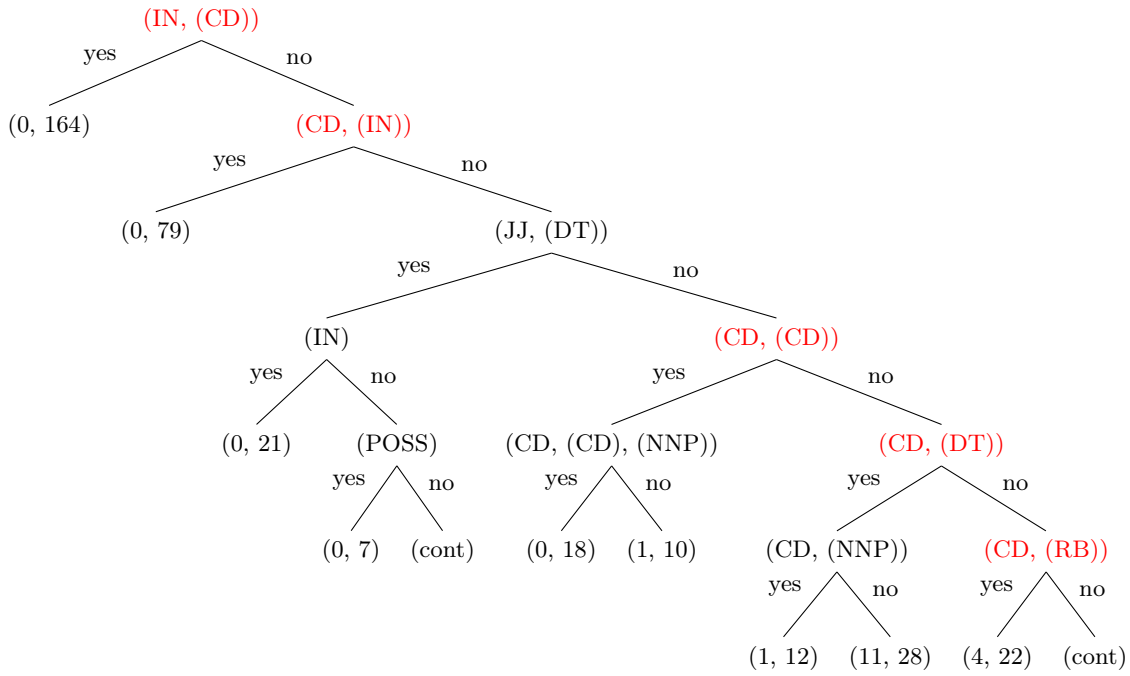


Figure 4.10: Suspicion tree (POS) for generation from the NP-6 data (after fixing genitive, coordination and determiner cases).

d. (CD, (DT))
A seasonally adjusted 332.000
(332.000/CD, (A/DT), (ADJUSTED/JJ, (SEASONALLY/RB)))

e. (CD, (RB))
1987 and early 1988
(1987/CD, (AND/CC, (1988/CD, (EARLY/RB))))

Noticeably, the suspicious form (CD) does not appear in the suspicion tree. In other words, the tree highlights the fact that cardinals lead to generation failure in the contexts shown but not in all contexts. Indeed, in this case, all suspicious forms points to a single cause of failure namely, a mismatch between grammar and lexicon. In the TAG grammar used, the constructions illustrated in (8) all expect cardinals to be categorised as nouns. In the lexicon however, cardinals are categorised as determiners. We modified the lexicon to categorise cardinals as both determiners and nouns; and rerun the generator on the input. In the newly built suspicion trees, cardinals no longer induce high generation failure rates. The fact that cardinals are not always associated with failure can be traced back to the fact that they are often used as determiners and that for this context, the lexicon contains the appropriate information.

4.5 Using Error Mining to Improve Generation Results

We now briefly report on how these error mining algorithms can help improve a generation system by showing the impact of corrections on undergeneration.

4.5.1 Fixing Grammar and Lexicon, and Rewriting Dependency Trees

Generating NP Chunks Table 4.2 summarises a run with 6 iterations between error mining and error correction on the NP data. The first row indicates the type of NP chunks to be processed, the second (*Input Data*) the number of NP chunks to be processed, the third (*Init Fail*) the number of input on which generation initially fails and the last 5 ones the decrease in errors (the number of failed cases with the *percentage failure*) after fixing error cases identified by the suspicion tree. R(X) indicates that the correction is obtained by rewriting the input for phenomena X, G(X) indicates corrections in the grammar and L(X) indicates corrections in the lexicon.

	NP-4	NP-6	NP-ALL
Input Data	23468	10520	123523
Init Fail	5939 (*25.3%*)	3560 (*33.8%*)	26769 (*21.7%*)
G(Coord)	4246 (*18.1%*)	2166 (*20.6%*)	21525 (*17.4%*)
R(Gen)	999 (*4.3%*)	956 (*9.1%*)	16702 (*13.5%*)
R(Dt)	833 (*3.6%*)	881 (*8.4%*)	16263 (*13.2%*)
L(days)	678 (*2.9%*)	876 (*8.3%*)	16094 (*13%*)
R(Adv)	649 (*2.7%*)	865 (*8.2%*)	16028 (*13%*)

Table 4.2: Diminishing the number of errors using information from error mining on NP data.

The corrections involve rewriting the SR data to the format expected by our realiser, grammar corrections and lexicon enrichment. Each time a correction is applied, the suspicion tree is recomputed thereby highlighting the next most likely sources of errors. G(Coord) indicates a fix in the grammar for coordination (discussed in Section 4.3.3.2 and Section 4.4.4.2). R(Gen) involves rewriting dependency trees for genitive NPs (e.g., *Oakland 's thief*) (Section 4.3.3.2 and Section 4.4.4.2) and R(Dt) rewriting dependency trees with determiners to map its part-of speech from determiner (DT) to pronoun (PRP) (Section 4.4.4.3) and to noun (NN) (nominal positions, e.g., *That's good*). L(days) involves updating the lexicon with correct DAYS/NN to TAG families mapping (Section 4.4.4.3). R(Adv) involves rewriting de-

pendency trees with adverbs to map its part-of speech from adverb (RB) to adjective (JJ) (e.g., *real estate alone*) (Section 4.4.4.2).

As the table shows, error mining permits decreasing undergeneration by 22.6, 25.6 and 8.7 points for NPs of size 4, 6 and ALL respectively. This suggests that simple NPs can be generated but that bigger NPs still cause undergeneration (8.2% and 13% of the cases respectively for NP-6 and NP-ALL) presumably because of more complex modifiers such as relative clauses, PPs and multiple determiners. Since in particular, relative clauses also appear in sentences, we proceeded to error mine the sentence data so as to provide more data for the error mining algorithm and therefore get a more global picture of the most important causes of failure.

Generating Sentences Tables 4.3 show the impact of corrections on generation for sentences. The first column indicates the type of sentences to be processed, the second (*Input Data*) the number of sentences to be processed, the third (*NP-Final*) the number of input (processed with all improvements from Table 4.2) on which generation fails and, the fourth and the fifth error rates after rewriting dependency trees for infinitival cases and auxiliary verb cases respectively.

	Input Data	NP-Final	R(Inf)	R(Aux)
S-6	3877	1707 (*44.0%*)	753 (*19.4%*)	398 (*10.3%*)
S-8	3583	1749 (*48.8%*)	936 (*26.1%*)	576 (*16.1%*)
S-ALL	26725	19280 (*72.1%*)	17862 (*66.8%*)	16445 (*61.5%*)

Table 4.3: Diminishing the number of errors using information from error mining on S data.

During error mining on the S data, *infinitival verbs* (discussed in Section 4.4.4.3) and *auxiliary verbs* appear as prominent mismatches between the SR dependency trees and the input expected by our generator. R(Inf) in Table 4.3 involves 3 different rewriting rules corresponding to dependency relations *im*, *oprd* and *prd* for rewriting dependency trees with infinitival verbs. R(Aux) indicates rewriting for dependency trees with Verb/Auxiliary nuclei (e.g., *the whole process might be reversed*).

Finally, Table 4.4 summarises results from Table 4.2 and Table 4.3. The first column indicates the type of dependency data to be processed and the second (*Input Data*) the number of data to be processed. The rows named (*Init Fail*), (*NP-Final*) and (*S-Final*) are initial error rates, errors after applying improvements from Table 4.2, and errors after applying improvements from Table 4.3. Table 4.4 adds an extra final improvement step (*Final-Coling*, final results reported in [Narayan and Gardent, 2012a]) consisting of minor grammar improvement (trees for pre-determiner

	Data	Init Fail	NP-Final	S-Final	Final-Coling
NP-4	23468	5939 (*25.3%*)	649 (*2.7%*)	-	503 (*2.1%*)
NP-6	10520	3560 (*33.8%*)	865 (*8.2%*)	-	584 (*5.5%*)
NP-ALL	123523	26769 (*21.7%*)	16028 (*13.0%*)	-	13967 (*11.3%*)
S-6	3877	-	1707 (*44.0%*)	398 (*10.3%*)	371 (*9.5%*)
S-8	3583	-	1749 (*48.8%*)	576 (*16.1%*)	545 (*15.2%*)
S-ALL	26725	-	19280 (*72.1%*)	16445 (*61.5%*)	16374 (*61.2%*)

Table 4.4: Overall impact of error mining on generation from different types of dependency trees.

PDT added, e.g., *all these millions*), lexicon enrichment (mapping to TAG families corrected) and rewriting rule (mapping part-of-speech from conjunction CC to determiner DT, e.g., *neither/CC the Bush administration nor arms-control experts*). The "Final-Coling" column in this Table shows the impact of S error reduction on NP error reduction. As can be seen reducing S-errors positively impact NP errors throughout.

In total we defined 11 rewrite rules (Gen-1, Dt-4, Adv-1, Inf-3, Aux-1 and Final-1), made 2 grammar corrections and performed a few lexicon updates.

Coverage and accuracy As the tables show, the corrections carried out after a few cycle of error mining and error correction helps achieve a large improvement in coverage for smaller dependency trees; we notice a large drop of 23.2 points (from 25.3% to 2.1%) in error rates for NP-4, 28.3 points for NP-6, 34.5 points for S-4 and 33.6 points for S-6. For larger dependency trees however, improvement is more limited. Thus, the failure rate is reduced by 10.4 points for NP-ALL (NPs from minimum size 1 to maximum size 91 with the average size 4); and by 10.9 points for S-ALL (sentences from minimum size 1 to maximum size 134 with the average size 22).

To assess the precision of the surface realiser after error mining, we computed the BLEU score for the covered sentence data and obtained a score of 0.835 for S-6, 0.80 for S-8 and 0.675 for S-ALL. The BLEU score before error mining and correction is not reported here since it has low coverage due to the mismatches between the structures provided by the SR task and those expected by the realiser.

4.5.2 Fixing Bug in Realization Algorithm

Symbolic generators are very sensitive to discrepancies in the generation system. Although, we improved significantly for all sorts of dependency trees (NPs and Ss)

with our vigorous error mining, the coverage of our generator was still very low (38.8%) for S-ALL. At this stage, our error mining techniques were not able to pin point any major problem in the grammar or the lexicon, or any mismatches in the input dependency trees. The reported suspicious forms were somehow uniformly distributed.

This scenario points us to possibility of bugs in our realization algorithm (described in Chapter 3). After thorough investigation of our generation system, we found that much of the remaining undergeneration was due to how multiple modifiers were represented.

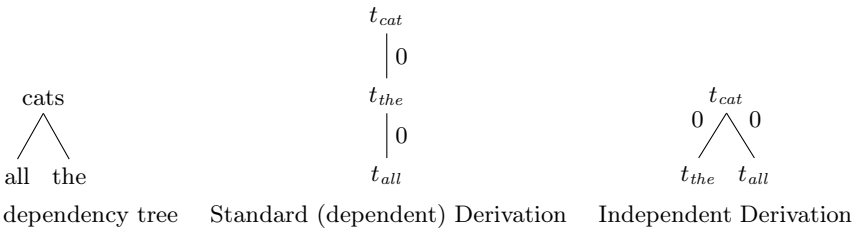


Figure 4.11: Shallow dependency tree for the phrase *All the cats* and its possible derivations.

Consider an example dependency tree shown in Figure 4.11 (leftmost) for the phrase *All the cats* where both *all* and *the* modifies *cats*. As shown in Figure 4.11, there are two possible derivations of this phrase in TAG. Figure 4.11 (middle) is an standard derivation [Vijay-Shanker and Joshi, 1988] where the tree for *all* adjoins to the root of the tree for *the* and the resulting tree then adjoins to the root of the tree for *cat*. Figure 4.11 (rightmost) is an independent derivation (multiple adjunction, [Schabes and Shieber, 1992; Schabes and Shieber, 1994]) which allows both trees for *the* and *all* to adjoin simultaneously at the root of the tree for *cat*. Schabes and Shieber (1994) have shown that multiple adjunction better captures dependencies between words. In fact, SR Task dependency trees also use such dependencies quite frequently for representing multiple auxiliary verbs, multiple intersective modifiers, sentential clauses etc.

Given the input shown in Figure 4.11 (leftmost), a generator producing standard derivations [Vijay-Shanker and Joshi, 1988] will fail to generate. The input dependency tree will be considered a mismatch between the input provided and the input expected by the grammar. Hence, the input dependency tree will require a rewrite to match it to the standard derivation (Figure 4.11, middle).

Instead, following Schabes and Shieber (1994), for the input shown in Figure 4.11 (leftmost), our generator tries to generate the independent derivation (Figure 4.11,

rightmost) capturing the exact dependencies.

Schabes and Shieber (1994) have provided an algorithm for TAG which supports multiple adjunction. The current version of our realisation algorithm extends Schabes and Shieber's approach to FB-LTAG. However, our recent research has shown that the direct extension of Schabes and Shieber's approach to FB-LTAG fails to derive the sentence under multiple adjunction because of inappropriate feature unifications. We give a much more detailed explanation of this issue in Appendix A. We also present a proper extension of multiple adjunction in TAG to FB-LTAG in Appendix A.

We incorporate these modifications to our realization algorithm and regenerate S-ALL. Note that these modifications fix a distinct undergeneration problem of the generator and will not alter the results discussed in Chapter 3 and the previous sections of this chapter.

Final coverage and accuracy As we discussed before, dependency trees reflecting the need for multiple adjunction are very frequent in the SR Task data. The discussed modifications improves our generation by an unprecedented margin of 42.5 points.

	Input Data	Final-Coling		Final	
		Fail (Failed %)	BLEU	Fail (Failed %)	BLEU
S-ALL	26725	16445 (*61.5%*)	0.675	5078 (*19%*)	0.72

Table 4.5: Diminishing the number of errors after fixing the realization algorithm.

Table 4.5 shows the final results achieved on S-ALL. *Final-Coling* are final results shown in Table 4.4. *Final* shows the final results after fixing the algorithm. As we can see, we drop the error rate from 61.5% to 19% and improve BLEU score from 0.675 to 0.72 for covered sentences.

The suspicion tree built after the *Final* step shows that coordination cases appear as most suspicious forms. Although the corrections made for coordination in the grammar G(Coord) permit generating simple coordinations (e.g., *John and Mary likes beans.*, *We played on the roof and in the garden.*, *I cooked beans and she ate it.*), the grammar still fails to generate for more complex coordination phenomena (e.g., verb coordination *I cooked and ate beans.*, gapping phenomenon *John eat fish and Harry chips.*, *I liked beans that Harry cooked and which Mary ate.*) [Sarkar and Joshi, 1996]. Other top suspicious forms are multiword expressions (e.g., *at least*, *so far*) and foreign words (part-of-speech FW) (e.g., *the naczelnik*, *perestroika*, *product de jour*).

4.6 Conclusion

Previous work on error mining has focused on applications (parsing) where the input data is sequential working mainly on words and part of speech tags. These techniques generate a flat list of suspicious forms ranked by decreasing order of suspicion.

In this chapter, we proposed two novel approaches to error mining: first, we proposed a novel approach to error mining which permits mining trees and second, we introduced another novel error mining algorithm to structure the output of error mining in a way that supports a linguistically meaningful error analysis.

We demonstrated its workings by applying it to the input data provided by the Generation Challenge SR Task to the analysis of undergeneration in a grammar based surface realisation algorithm. And we showed that this supports the identification of gaps and errors in the grammar and in the lexicon; and of mismatches between the input data format and the format expected by our realiser. We show that it permits quickly identifying the main sources of errors while providing a detailed description of the various subcases of these sources if any.

The approach is generic in that it permits mining trees and strings for suspicious forms of arbitrary size and arbitrary conjunctions of labelling. It could be used for instance to detect and structure the n-grams that frequently induce parsing errors; or to identify subtrees that frequently occur in agrammatical output produced by a generator.

One interesting direction would be to explore how these error mining approaches can be used to detect the most likely sources of *overgeneration* based on the output of this surface realiser on the SR Task data. Using the Penn Treebank sentences associated with each SR Task dependency tree, the two tree sets necessary to support error mining could be created by dividing the set of trees output by the surface realiser into a set of trees (FAIL) associated with overgeneration (the generated sentences do not match the original sentences) and a set of trees (SUCCESS) associated with success (the generated sentence matches the original sentences). Exactly which tree should populate the SUCCESS and FAIL set is an open question. The various evaluation metrics used by the SR Task (BLEU, NIST, METEOR and TER) could be used to determine a threshold under which an output is considered incorrect (and thus classificed as FAIL). Alternatively, a strict matching might be required. Similarly, since the surface realiser is non-deterministic, the number of output trees to be kept will need to be experimented with.

We could also investigate whether the approach can be useful in automatically detecting treebank and parse errors [Boyd *et al.*, 2008; Dickinson and Smith, 2011].

Chapter 5

Generating Elliptic Coordination

Contents

This chapter is based on the following paper:

Claire Gardent and Shashi Narayan. *Generating elliptic coordination*, in Proceedings of the 14th European Workshop on Natural Language Generation (ENLG), pages 40–50, Sofia Bulgaria, August 2013.

In this chapter, we focus on the task of generating elliptic sentences. We extract from the data provided by the Surface Realisation (SR) Task [Belz *et al.*, 2011] 2398 input whose corresponding output sentence contain an ellipsis. We show that 9% of the data contains an ellipsis and that both coverage and BLEU score markedly decrease for elliptic input (from 82.3% coverage for non-elliptic sentences to 65.3% for elliptic sentences and from 0.60 BLEU score to 0.47). We argue that elided material should be represented using phonetically empty nodes and we introduce a set of rewrite rules which permits adding these empty categories to the SR data. Finally, we evaluate an existing surface realiser on the resulting dataset. We show that, after rewriting, the generator achieves a coverage of 76% and a BLEU score of 0.74 on the elliptical data.

5.1 Introduction

To a large extent, previous work on generating ellipsis has assumed a semantically fully specified input [Shaw, 1998; Harbusch and Kempen, 2009; Theune *et al.*, 2006]. Given such input, elliptic sentences are then generated by first, producing full sentences and second, deleting from these sentences substrings that were identified to obey deletion constraints.

In contrast, recent work on generation often assumes input where repeated material has already been elided. This includes work on sentence compression which regenerates sentences from surface dependency trees derived from parsing the initial text [Filippova and Strube, 2008]; Surface realisation approaches which have produced results for regenerating from the Penn Treebank [Langkilde, 2002; Callaway, 2003; Zhong and Stent, 2005; Cahill and Van Genabith, 2006; White and Rajkumar, 2009]; and more recently, the SR Task [Belz *et al.*, 2011] which has proposed dependency trees and graphs derived from the Penn Treebank (PTB) as a common ground input representation for testing and comparing existing surface realisers. In all these approaches, repeated material is omitted from the representation that is input to surface realisation.

As shown in the literature, modelling the interface between the empty phonology and the syntactic structure of ellipses is a difficult task. For parsing, Sarkar and Joshi (1996), Banik (2004) and Seddah (2008) propose either to modify the derivation process of Tree Adjoining Grammar or to introduce elementary trees anchored with empty category in a synchronous TAG to accommodate elliptic coordinations. In HPSG (Head-Driven Phrase Structure Grammar), Levy and Pollard (2001) introduce a neutralisation mechanism to account for unlike constituent coordination ; in LFG

(Lexical Functional Grammar), Dalrymple and Kaplan (2000) employ set values to model coordination; in CCG (Combinatory Categorial Grammar, [Steedman, 1996]), it is the non-standard notion of constituency assumed by the approach which permits accounting for coordinated structures; finally, in TLCG (Type-Logical Categorial Grammar), gapping is treated as like-category constituent coordinations [Kubota and Levine, 2012].

In this chapter, we focus on how surface realisation handles elliptical sentences given an input where repeated material is omitted. We extract from the SR data 2398 input whose corresponding output sentence contain an ellipsis. Based on previous work on how to annotate and to represent ellipsis, we argue that elided material should be represented using phonetically empty nodes (Section 5.2) and we introduce a set of rewrite rules which permits adding these empty categories to the SR data (Section 5.3). We then evaluate our surface realiser (described in Chapter 3, [Narayan and Gardent, 2012b]) on the resulting dataset (Section 5.4) and we show that, on this data, the generator achieves a coverage of 76% and a BLEU score, for the generated sentences, of 0.74. Section 5.5 discusses related work on generating elliptic coordination. Finally, Section 5.6 concludes.

5.2 Representing and Annotating Elided Material

In elliptic sentences, there is meaning without sound. Thus the usual form/meaning mappings that in non-elliptic sentences allow us to map sounds onto their corresponding meanings, break down. We now describe various phenomena of elliptic coordination and then briefly review how elided material is represented in the literature and in the SR data.

5.2.1 Elliptic Sentences

Elliptic coordination involves a wide range of phenomena including in particular non-constituent coordination (9(a), NCC) i.e., cases where sequences of constituents are coordinated; gapping (9(b), Gapping) i.e., cases where the verb and possibly some additional material is elided; shared subjects (9(c), SS) and right node raising (9(d), RNR) i.e., cases where a rightmost constituent is shared by two or more clauses. Other types of elliptic coordination include sluicing[13] and Verb-Phrase ellipsis[14]. These will not be discussed here because they can be handled by the generator by

[13] *Phoebe wants to eat something, but she doesn't know what.*
[14] *You might do it, but I won't.*

having the appropriate categories in the grammar and the lexicon e.g., in a Tree Adjoining Grammar, an auxiliary anchoring a verb phrase for VP ellipsis and question words anchoring a sentence for sluicing.

(9) (a) [It rose]$_i$ 4.8 % in June 1998 and ϵ_i 4.7% in June 1999. *NCC*
(b) Sumitomo bank [donated]$_i$ \$500,000, Tokyo prefecture ϵ_i \$15,000 and the city of Osaka ϵ_i \$10,000. *Gapping*
(c) [the state agency 's figures]$_i$ ϵ_i confirm previous estimates and ϵ_i leave the index at 178.9. *Shared Subject*
(d) He commissions ϵ_i and splendidly interprets ϵ_i [fearsome contemporary scores]$_i$. *RNR*

We refer to the non-elliptic clause as the *source* and to the elliptic clause as the *target*. In the source, the brackets indicate the element shared with the target while in the target, the ϵ_i sign indicate the elided material with co-indexing indicating the antecedent/ellipsis relation. In gapping clauses, we refer to the constituents in the gapped clause, as *remnants*.

5.2.2 Representing and Annotating Elided Material

We now briefly review how elided material is represented in the literature and in the SR data.

5.2.2.1 Linguistic Approaches

While Sag (1976), Williams (1977), Kehler (2002), Merchant (2001) and van Craenenbroeck (2010) have argued for a structural approach i.e., one which posits syntactic structure for the elided material, Keenan (1971), Hardt (1993), Dalrymple *et al.* (1991), Ginzburg and Sag (2000) and Culicover and Jackendoff (2005) all defend a non structural approach. Although no consensus has yet been reached on these questions, many of these approaches do postulate an abstract syntax for ellipsis. That is they posit that elided material licenses the introduction of phonetically empty categories in the syntax or at some more abstract level (e.g., the logical form of generative linguistics).

5.2.2.2 Treebank Approaches

Similarly, in computational linguistics, the treebanks used to train and evaluate parsers propose different means of representing ellipsis.

Penn Treebank For phrase structure syntax, the Penn Treebank Bracketing Guidelines extensively describe how to annotate coordination and missing material in English [Bies *et al.*, 1995]. For shared complements (e.g., shared subject and right node raising constructions), these guidelines state that the elided material licenses the introduction of an empty *RNR* category co-indexed with the shared complement (cf. Figure 5.1) while gapping constructions are handled by labelling the gapping remnants (i.e., the constituents present in the gapping clause) with the index of their parallel element in the source (cf. Figure 5.2).

```
(S
 (VP (VB Do)(VP (VB avoid)
    (S (VP (VPG puncturing(NP *RNR*-5))
       (CC or)
          (VP (VBG cutting)(PP (IN into)
                 (NP *RNR*-5)))
                     (NP-5 meats))))
 )
```

Figure 5.1: Penn Treebank annotation for Right Node Raising "*Do avoid puncturing* ϵ_i *or cutting into* ϵ_i *[meats]*$_i$."

```
(S
  (S (NP-SBJ-10 Mary)
       (VP (VBZ likes) (NP-11 potatoes)))
         (CC and)
           (S (NP-SBJ=10 Bill)
                (, ,) (NP=11 ostriches))
 )
```

Figure 5.2: Penn Treebank annotation for gapping "*Mary [likes]*$_i$ *potatoes and Bill* ϵ_i *ostriches.*"

Dependency Treebanks In dependency treebanks, headless elliptic constructs such as gapping additionally raise the issue of how to represent the daughters of an empty head. Three main types of approaches have been proposed. In dependency treebanks for German [Daum *et al.*, 2004; Hajič *et al.*, 2009] and in the Czech treebank [Čmejrek *et al.*, 2004; Hajič *et al.*, 2009], one of the dependents of the headless phrase is declared to be the head. This is a rather undesirable solution because it hides the fact that there the clause lacks a head. In contrast, the Hungarian dependency treebank [Vincze *et al.*, 2010] explicitly represents the elided elements in the

trees by introducing phonetically empty elements that serve as attachment points to other tokens. This is the cleanest solution from a linguistic point of view. Similarly, Seeker and Kuhn (2012) present a conversion of the German Tiger treebank which introduces empty nodes for verb ellipses if a phrase normally headed by a verb is lacking a head. They compare the performance of two statistical dependency parsers on the canonical version and the CoNLL 2009 Shared Task data and show that the converted dependency treebank they propose yields better parsing results than the treebank not containing empty heads.

5.2.2.3 Shallow Dependency Trees from the SR Data

In the shallow dependency tree provided by the SR Task, the representation of ellipsis adopted in the Penn Treebank is preserved modulo some important differences regarding co-indexing. Figures 5.3, 5.4, 5.5, 5.6, 5.7 and 5.8 show (simplified) input trees from the SR data.

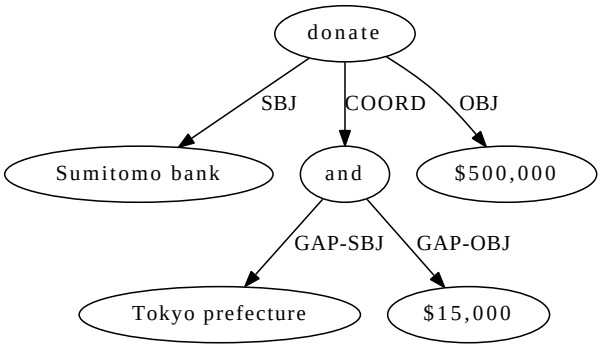


Figure 5.3: Gapping in the SR data. "*Sumitomo bank [donated]*$_i$ $\$500,000$ *and Tokyo prefecture* ϵ_i $\$15,000$."

Gapping is represented as in the PTB by labelling the remnants with a marker indicating the source element parallel to each remnant. However while in the PTB, this parallelism is made explicit by co-indexing (the source element is marked with an index i and its parallel target element with the marker $= i$), in the SR data this parallelism is approximated using functions. For instance, if the remnant is parallel to the source subject, it will be labelled GAP-SBJ (cf. Figure 5.3). GAPPING

This is problematic in cases with several gapped clauses such as *France voted the same as the US 76% of the time, West Germany 79%, Italy 81% and Britain 83%* whose SR input tree is shown in Figure 5.4. Since the input trees are unordered and the tree structure is flat, it is impossible to determine which remnants belong together i.e., whether the input tree represents the sentence "*France voted the same*

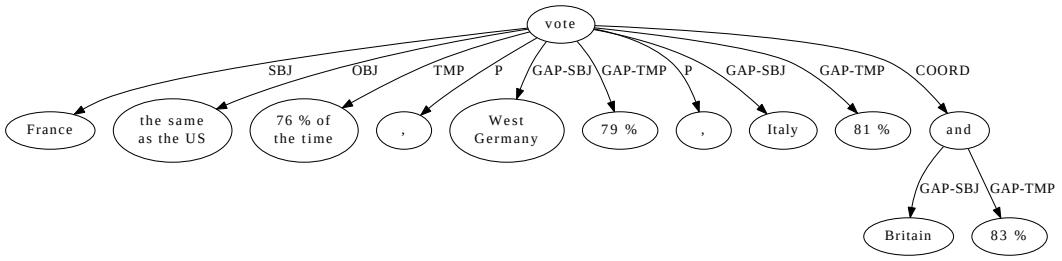


Figure 5.4: Problematic gapping case in the SR data. "*France [voted]$_i$ the same as the US* 76% *of the time, West Germany* ϵ_i 79%, *Italy* ϵ_i 81% *and Britain* ϵ_i 83%."

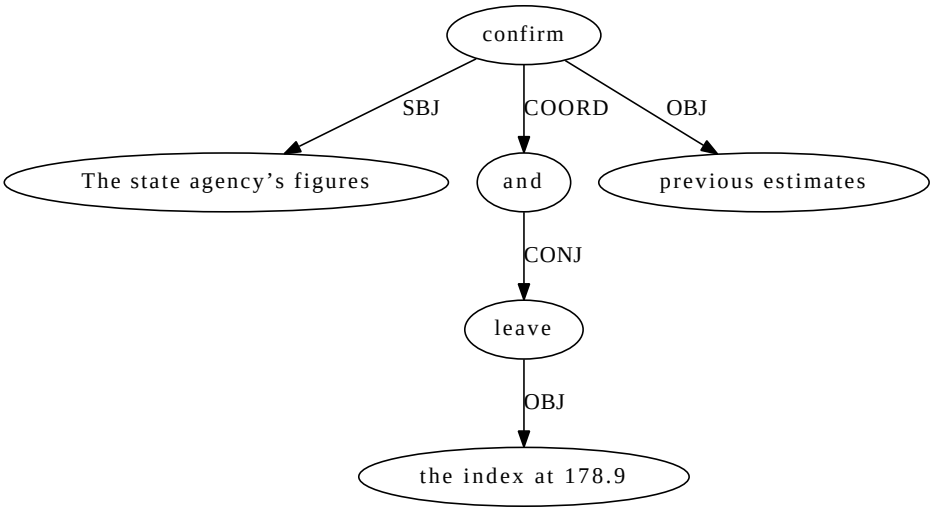


Figure 5.5: Subject sharing in the SR data. "*[The state agency 's figures]$_i$* ϵ_i *confirm previous estimates and* ϵ_i *leave the index at 178.9*"

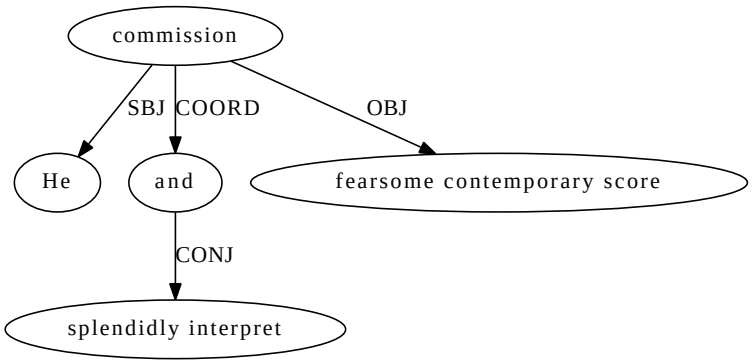


Figure 5.6: Subject Sharing and RNR in the SR data. "*[He]$_j$* ϵ_j *commissions* ϵ_i *and* ϵ_j *splendidly interprets* ϵ_i *[fearsome contemporary scores]$_i$*."

as the US 76% of the time, West Germany 79%, Italy 81% and Britain 83%" or "*France voted the same as the US 76% of the time, West Germany 81%, Italy 79% and Britain 83%*".

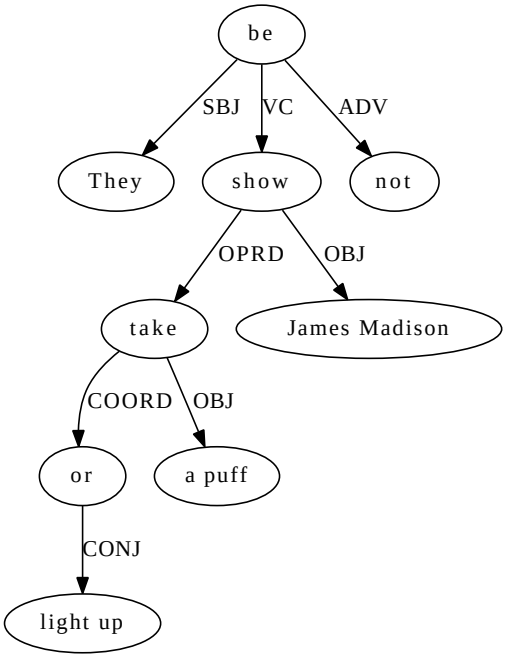


Figure 5.7: Non-shared Object "*They aren't showing James Madison taking a puff or lighting up*"

For right-node raising (Figure 5.6, RNR) and shared subjects (Figure 5.5 and 5.6, SS) , the coindexation present in the PTB is dropped in the SR data. That is, while in the PTB, the shared argument is represented by an empty category co-indexed with the realised complement, in the SR data this co-indexing disappear. As a result, the SR representation underspecifies the relation between the object and the coordinated verbs in RNR constructions: the object could be shared as in *He commissions* ϵ_i *and splendidly interprets* ϵ_i *[fearsome contemporary scores]*$_i$. (Figure 5.6) or not as in *They aren't showing James Madison taking a puff or lighting up* (Figure 5.7). In both cases, the representation is the same i.e., the shared object (*fearsome contemporary scores*) and the unshared object (*a puff*) are both attached to the first verb. RNR SS

Finally, non-constituent coordination (NCC) structures are handled in the same way as gapping by having the gapping remnants labelled with a GAP prefixed function (e.g., GAP-SBJ) indicating which element in the source the gapping remnant is parallel to (cf. Figure 5.8). NCC

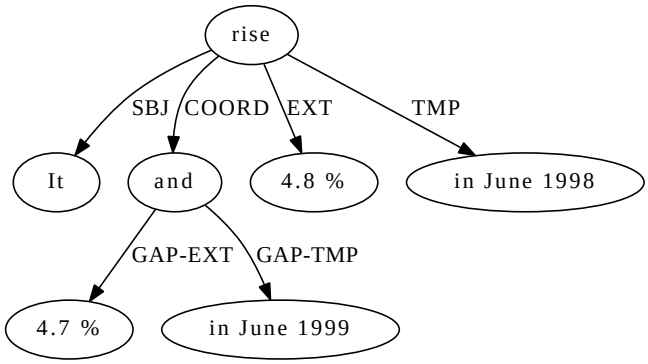


Figure 5.8: NCC in the SR data. "*It rose 4.8 % in June 1998 and 4.7% in June 1999.*"

In sum, while some linguists have argued for an approach where ellipsis has no syntactic representation, many have provided strong empirical evidence for positing empty nodes as place-holders for elliptic material (Section 5.2.2.1). Similarly, in devising treebanks, computational linguists have oscillated between representations with and without empty categories (Section 5.2.2.2).

In contrast, the SR representation schema underspecifies ellipsis in two ways. For gapping and non-constituent coordination, it describes parallelism between source and target elements rather than specifying the syntax of the elided material. For subject sharing and right node raising, it fails to explicitly specify argument sharing. In the following section, we propose to modify the SR representations by making the relationship between ellipsis and antecedent explicit using phonetically empty categories and co-indexing.

5.3 Rewriting the SR Data

In the previous section, we discussed how the SR data underspecifies the sentences to be generated. To resolve this underspecification, we rewrite the SR data using tree rewrite rules as follows.

Gapping and NCC structures rewriting For gapping and NCC structures, we copy the source material that has no (GAP- marked) counterpart in the target clause to the target clause marking it to indicate a phonetically empty category. Figure 5.9 shows the rule affecting such structures. V is a verb; *CONJ*, a conjunctive coordination; and X, Y and W, three sets of dependents. The antecedent verb (V) and the source material without counterpart in the gapping clause (W) are copied

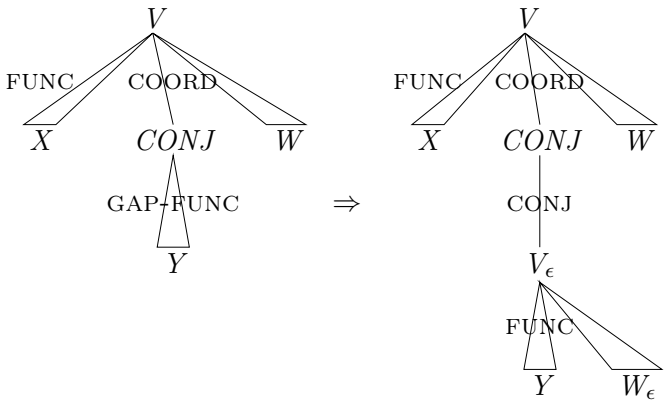


Figure 5.9: Gapping and Non-Constituent Coordination structures rewriting.

over to the gapping clause and marked as phonetically empty.

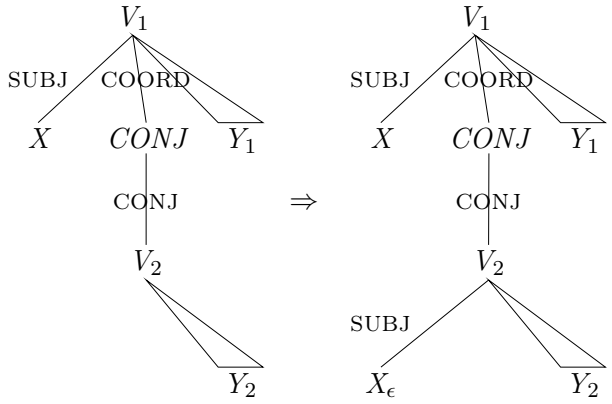


Figure 5.10: Subject sharing rewriting.

Subject sharing rewriting For Subject sharing, we copy the shared subject of the source clause in the target clause and mark it to be a phonetically empty category. Figure 5.10 shows the subject sharing rewrite rule. V_1 and V_2 are verbs; *CONJ*, a conjunctive coordination; and X, Y_1 and Y_2, three sets of dependents. The subject dependent X is shared between the verbs V_1 and V_2. The subject dependent X is copied over to the target clause and marked as phonetically empty.

Right-Node-Raising rewriting For Right-Node-Raising, we unfold the ambiguity producing structures where arguments present in the source but not in the target

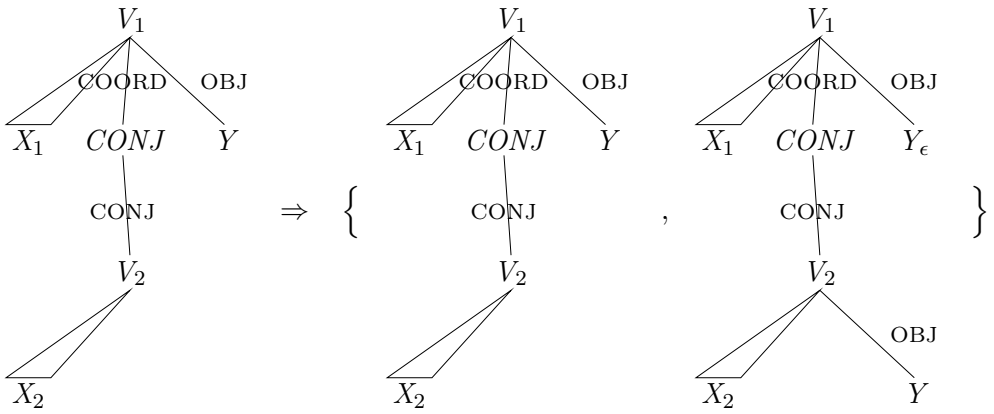


Figure 5.11: Right-Node-Raising rewriting.

are optionally copied over to the target. Figure 5.11 shows the right-node-raising rewrite rule. V_1 and V_2 are verbs; *CONJ*, a conjunctive coordination; and X_1, X_2 and Y, three sets of dependents. The object dependent Y of the verb V_1 could be shared with the verb V_2. Hence, the object dependent is optionally copied over to the target clause and marked as phonetically empty in the source clause. Generator tries to generate from both possible structures.

These rewrite rules are implemented efficiently using GrGen, an efficient graph rewriting system [Geiß*et al.*, 2006].

5.4 Generating Elliptic Coordination

5.4.1 The Surface Realiser

To generate sentences from the SR data, we use the surface realiser described in Chapter 3, [Narayan and Gardent, 2012b]. This generator first selects the elementary FB-LTAG trees associated in the lexicon with the lemmas and part of speech tags associated with each node in the input dependency tree. It then attempts to combine the selected trees bottom-up taking into account the structure of the input tree (only trees that are selected by nodes belonging to the same local input tree are tried for combination). A language model is used to implement a beam search letting through only the n most likely phrases at each bottom up combination step. In this experiment, we set n to 5. The generator thus outputs at most 5 sentences for each input.

As mentioned in the introduction, most computational grammars have difficulty

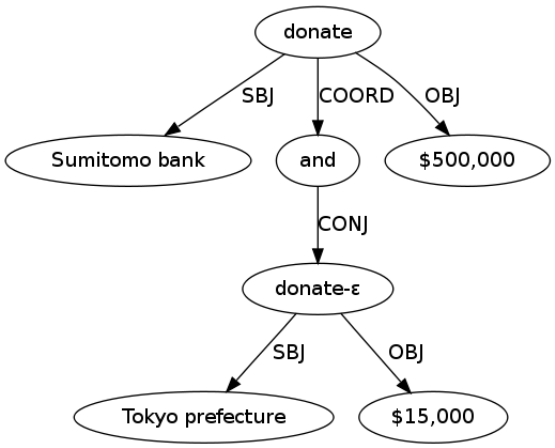


Figure 5.12: Gapping after rewriting "*Sumitomo bank [donated]*$_i$ $\$500,000$ *and Tokyo prefecture* ϵ_i $\$15,000$."

accounting for ellipses and FB-LTAG is no exception.

For parsing with TAG, two main methods have been proposed for processing elliptical sentences. Sarkar and Joshi (1996) introduces an additional operation for combining TAG trees which yields derivation graphs rather than trees. Seddah (2008) uses Multi-Component TAG and proposes to associate each elementary verb tree with an elliptic tree with different pairs representing different types of ellipses.

We could use either of these approaches for generation. The first approach [Sarkar and Joshi, 1996] however has the drawback that it leads to a non-standard notion of derivation (the derivation trees become derivation graphs). The second [Seddah, 2008] on the other hand, induces a proliferation of trees in the grammar and impacts efficiency.

Instead, we show that, given an input enriched with empty categories as proposed in the previous section, neither the grammar nor the tree combination operation need changing. Indeed, our FB-LTAG surface realiser directly supports the generation of elliptic sentences. It suffices to assume that an FB-LTAG elementary tree may be anchored by the empty string. Given an input node marked as phonetically empty, the generator will then select all FB-LTAG rules that are compatible with the lexical and the morpho-syntactic features labelling that node. Generation will then proceed as usual by composing the trees selected on the basis of the input using substitution and adjunction; and by retrieving from the generation forest those sentences whose phrase structure tree covers the input.

For instance, given the rewritten input shown in Figure 5.12, the TAG trees associated in the lexicon with *donate* will be selected; anchored with the empty

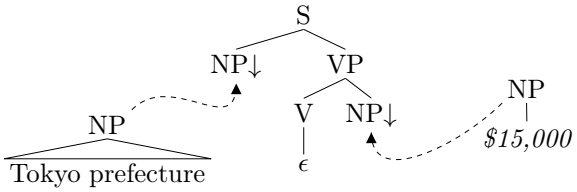


Figure 5.13: Derivation for "Tokyo prefecture ϵ $15,000".

string and combined with the TAG trees built for *Tokyo Prefecture* and *$15,000* thus yielding the derivation shown in Figure 5.13.

The generator treats coindexed nodes separately and so it may overgenerate by assigning different TAG trees to these nodes. Despite of overgeneration, it always generates grammatical sentences because coindexed nodes always share same grammatical features.

5.4.2 The Data

We use the shallow dependency trees (26604 sentences) provided by the SR Task to evaluate the performance of the surface realiser on elliptic coordination. This is comparatively less than the SR data (26725 sentences) discussed in Chapter 3 and 4. Sentences with problematic cases such as multiword expressions and foreign words (discussed at the end of Chapter 4) are dropped to focus on complex coordination cases.

To focus on ellipsis, we retrieve those sentences which were identified by our rewrite rules as potentially containing an elliptic coordination. In essence, these rewrite rules will identify all cases of non-constituent coordination and gapping (because these involve GAP-X dependencies with "X" a dependency relation and are therefore easily detected) and of shared-subjects (because the tree patterns used to detect are unambiguous i.e., only apply if there is indeed a shared subject). For RNR, as discussed in the previous section, the SR format is ambiguous and consequently, the rewrite rules might identify as object sharing cases where in fact the object is not shared. As noted by one of our anonymous reviewers at ENLG 2013, the false interpretation could be dropped out by consulting the Penn Treebank[15]. The approach would not generalise to other data however.

In total, we retrieve 2398 sentences potentially containing an elliptic coordination

[15]The Penn Treebank makes the RNR interpretations explicit (refer to Figure 5.1), but this will deter the purpose of the SR Task

Elliptic Coordination Data				
Elliptic Coordination		Pass	BLEU Scores	
			COV	ALL
RNR (384)	Before	66%	0.68	0.45
	After	81%	0.70	0.57
	Delta	+15	+0.02	+0.12
SS (1462)	Before	70%	0.74	0.52
	After	75%	0.75	0.56
	Delta	+5	+0.01	+0.04
SS + RNR (456)	Before	61%	0.71	0.43
	After	74%	0.73	0.54
	Delta	+13	+0.02	+0.11
Gapping (36)	Before	3%	0.53	0.01
	After	67%	0.74	0.49
	Delta	+64	+0.21	+0.48
NCC (60)	Before	5%	0.68	0.03
	After	73%	0.74	0.54
	Delta	+68	+0.06	+0.51
Total (2398)	Before	65%	0.72	0.47
	After	76%	0.74	0.56
	Delta	**+11**	+0.02	+0.09

Table 5.1: Generation results on elliptical data before and after input rewriting.

from the SR data. The number and distribution of these sentences in terms of ellipsis types are given in Table 5.1. The number in brackets in the first column is the number of cases. Pass stands for the coverage of the generator. COV and ALL in BLEU scores column stand for BLEU scores for the covered and the total input data. And, last but not least, SS stands for Shared Subject, NCC for Non-Constituent Coordination, and RNR for Right Node Raising.

5.4.3 Evaluation

We ran the surface realiser on the SR input data both before and after rewriting elliptic coordinations. The results are shown in Table 5.2. We present our results on three sets of sentences; on the sentences estimated to contain ellipsis (+E); on sentences devoid of ellipsis (-E); and on all sentences (T). They indicate coverage and BLEU score before and after rewriting. Average BLEU score is given both with respect to covered sentences (COV) i.e., the set of input for which generation succeeds; and for all sentences (ALL). While computing the average BLUE score for ALL, BLEU score for each uncovered sentence was taken to be 0.

SR Data		Pass	BLEU Scores	
			COV	ALL
+E (2398) (Table 5.1)	Before	65%	0.72	0.47
	After	76%	0.74	0.56
	Delta	**+11**	+0.02	+0.09
-E (24206)	Before	82%	0.73	0.60
	After	82%	0.73	0.60
	Delta	+0	+0.00	+0.00
T (26604)	Before	81%	0.72	0.58
	After	**82%**	**0.73**	0.60
	Delta	**+1**	+0.01	+0.02

Table 5.2: Generation results on the SR data before and after input rewriting.

The impact of ellipsis on coverage and precision. Previous work on parsing showed that coordination was a main source of parsing failure [Collins, 1999]. Similarly, ellipses is an important source of failure for the TAG generator. Ellipses are relatively frequent with 9% of the sentences in the SR data containing an elliptic structure and performance markedly decreases in the presence of ellipsis. Thus, before rewriting, coverage decreases from 82.3% for non-elliptic sentences to 80.75% on all sentences (elliptic and non-elliptic sentences) and to 65.3% on the set of elliptic sentences. Similarly, BLEU score decreases from 0.60 for non elliptical sentences to 0.58 for all sentences and to 0.47 for elliptic sentences. In sum, both coverage and BLEU score decrease as the number of elliptic input increases.

The impact of the input representation on coverage and precision. Recent work on treebank annotation has shown that the annotation schema adopted for coordination impacts parsing. In particular, Maier *et al.* (2012) propose revised annotation guidelines for coordinations in the Penn Treebank whose aim is to facilitate the detection of coordinations. And Dukes and Habash (2011) show that treebank annotations which include phonetically empty material for representing elided material allows for better parsing results.

Similarly, Table 5.2 shows that the way in which ellipsis is represented in the input data has a strong impact on generation. Thus rewriting the input data markedly extends coverage with an overall improvement of 11 points (from 65% to 76%) for elliptic sentences and of almost 1 point for all sentences.

As detailed in Table 5.1 though, there are important differences between the different types of elliptic constructs: coverage increases by 68 points for NCC and 64 points for gapping against only 15, 13 and 5 points for RNR, mixed RNR-Shared

Subject and Shared Subject respectively. The reason for this is that sentences are generated for many input containing the latter types of constructions (RNR and Shared Subject) *even without rewriting*. In fact, generation succeeds on the non-rewritten input for a majority of RNR (66% PASS), Shared Subject (70% PASS) and mixed RNR-Shared Subject (61% PASS) constructions whereas it fails for almost all cases of gapping (3% PASS) and of NCC (5% PASS). The reason for this difference is that, while the grammar cannot cope with headless constructions such as gapping and NCC constructions, it can often provide a derivation for shared subject sentences by using the finite verb form in the source sentence and the corresponding infinitival form in the target. Since the infinitival does not require a subject, the target sentence is generated. Similarly, RNR constructions can be generated when the verb in the source clause has both a transitive and an intransitive form: the transitive form is used to generate the source clause and the intransitive for the target clause. In short, many sentences containing a RNR or a shared subject construction can be generated without rewriting because the grammar overgenerates i.e., it produces sentences which are valid sentences of English but whose phrase structure tree is incorrect.

Nevertheless, as the results show, rewriting consistently helps increasing coverage even for RNR (+15 points), Shared Subject (+5 points) and mixed RNR-Shared Subject (+13 points) constructions because (i) not all verbs have both a transitive and an intransitive verb form and (ii) the input for the elliptic clause may require a finite form for the target verb (e.g., in sentences such as "*[they]$_i$ weren't fired but instead ϵ_i were neglected*" where the target clause includes an auxiliary requiring a past participial which in this context requires a subject).

Precision is measured using the BLEU score. For each input, we take the best score obtained within the 5 derivations[16] produced by the generator. Since the BLEU score reflects the degree to which a sentence generated by the system matches the corresponding Penn Treebank sentence, it is impacted not just by elliptic coordination but also by all linguistic constructions present in the sentence. Nonetheless, the results show that rewriting consistently improves the BLEU score with an overall increase of 0.09 points on the set of elliptic sentences. Moreover, the consistent improvement in terms of BLEU score for generated sentences (COV column) shows that rewriting simultaneously improves both coverage and precision that is, that for those sentences that are generated, rewriting consistently improves precision.

[16]The language model used in the generator allows only 5 likely derivations (refer to section 5.4.1).

Analysing the remaining failure cases. To better assess the extent to which rewriting and the FB-LTAG generation system succeed in generating elliptic coordinations, we performed error mining on the elliptic data using our error miner described in Chapter 4 [Gardent and Narayan, 2012; Narayan and Gardent, 2012a]. This method permits highlighting the most likely sources of error given two datasets: a set of successful cases and a set of failure cases. In this case, the successful cases is the subset of rewritten input data for elliptic coordination cases for which generation succeeds. The failure cases is the subset for which generation fails. If elliptic coordination was still a major source of errors, input nodes or edges labelled with labels related to elliptic coordination (e.g., the COORD and the GAP-X dependency relations or the CONJ part of speech tag) would surface as most suspicious forms. In practice however, we found that the 5 top sources of errors highlighted by error mining all include the DEP relation, an unknown dependency relation used by the Penncoverter when it fails to assign a label to a dependency edge. In other words, most of the remaining elliptic cases for which generation fails, fails for reasons unrelated to ellipsis.

Comparison with other surface realisers There is no data available on the performance of surface realisers on elliptic input. However, the performance of the surface realiser can be compared with those participating in the shallow track of the SR challenge. Since the realiser we are using is not trained on the training data (the grammar was written manually), we try to generate the complete training data (Chapter 2). Using the training data also allows us to gather a larger set of elliptic sentences for evaluation while evaluating also on the test data (100 sentences) allows comparison with other realisers.

On the SR training data, the TAG surface realiser has an average run time of 2.78 seconds per sentence (with an average of 20 words per sentence), a coverage of 82% and BLEU scores of 0.73 for covered and 0.60 for all. On the SR test data, the realiser achieves a coverage of 79% and BLEU scores of 0.59 for covered and 0.47 for all. In comparison, the statistical systems in the SR Tasks achieved 0.88, 0.85 and 0.67 BLEU score on the SR test set and the best symbolic system 0.25 [Belz *et al.*, 2011].

5.5 Related Work

Previous work on generating elliptic sentences has mostly focused on identifying material that could be elided and on defining procedures capable of producing input

structures for surface realisation that support the generation of elliptic sentences.

Dalianis and Hovy (1996) carried out a study to see how and when people use aggregation i.e., operations for reducing redundancy. Using the result of this study, they proposed 8 aggregation rules including some rules for mapping repetitive syntactic structures to structures representing shared subjects and gapping.

Shaw (1998) developed a sentence planner which generates elliptic sentences in 3 steps. First, input data are grouped according to their similarities. Second, repeated elements are marked. Third, constraints are used to determine which occurrences of a marked element should be deleted. The approach is integrated in the PLAN-Doc system [McKeown *et al.*, 1994] and shown to generate a wide range of elliptic constructs including RNR, VPE and NCC using FUF/SURGE [Elhadad, 1993b], a realisation component based on Functional Unification Grammar.

Theune *et al.* (2006) describe how elliptic sentences are generated in a story generation system. The approach covers conjunction reduction, right node raising, gapping and stripping and uses dependency trees connected by rhetorical relations as input. Before these trees are mapped to sentences, repeated elements are deleted and their antecedent (the*source element*) is related by a SUBORROWED relation to their governor in the elliptic clause and a SUIDENTICAL relation to their governor in the antecedent clause. This is then interpreted by the surface realiser to mean that the repeated element should be realised in the source clause, elided in the target clause and that it licenses the same syntactic structure in both clauses.

Harbusch and Kempen (2009) have proposed a module called Elleipo which takes as input unreduced, non-elliptic, syntactic structures annotated with lexical identity and coreference relationships between words and word groups in the conjuncts; and returns as output structures annotated with elision marks indicating which elements can be elided and how (i.e., using which type of ellipsis). The focus is on developing a language independent module which can mediate between the unreduced input syntactic structures produced by a generator and syntactic structures that are enriched with elision marks rich enough to determine the range of possible elliptic and non-elliptic output sentences.

In CCG, grammar rules (type-raising and composition) permit combining non-constituents into a functor category which takes the shared element as argument; and gapping remnants into a clause taking as argument its left-hand coordinated source clause. White (2006) describes a chart based algorithm for generating with CCG and shows that it can efficiently realise NCC and gapping constructions.

Our proposal differs from these approaches in that it focuses on the surface realisation stage (assuming that the repeated elements have already been identified)

and is tested on a large corpus of newspaper sentences rather than on hand-made document plans and relatively short sentences.

5.6 Conclusion

In this chapter, we showed that elliptic structures are frequent and can impact the performance of a surface realiser. In line with linguistic theory and with some recent results on treebank annotation, we argued that the representation of ellipsis should involve empty categories and we provided a set of tree rewrite rules to modify the SR data accordingly. We then evaluated the performance of a TAG based surface realiser on 2398 elliptic input derived by the SR task from the Penn Treebank and showed that it achieved a coverage of 76% and a BLEU score of 0.74 on generated sentences. Our approach relies both on the fact that the grammar is lexicalised (each rule is associated with a word from the input) and on TAG extended domain of locality (which permits using a rule anchored with the empty string to reconstruct the missing syntax in the elided clause thereby making it grammatical).

We have released the 2398 input representations[17] we gathered for evaluating the generation of elliptic coordination so as to make it possible for other surface realisers to be evaluated on their ability to generate ellipsis. In particular, its would be interesting to examine how other grammar based generators perform on this dataset such as White's CCG based generator [2006] (which eschews empty categories by adopting a more flexible notion of constituency) and Carroll and Oepen's HPSG based generator [2005] (whose domain of locality differs from that of TAG).

[17] `http://www.loria.fr/~narayans/data/ellipsis-enlg2013.tar.gz`

Part II

Simplifying Sentences

Chapter 6

Background and Related Work

Contents

6.1 Sentence Simplification

This part of the dissertation focuses on text-to-text generation in particular sentence simplification. Sentence simplification maps a sentence to a simpler, more readable one approximating its content. For example, it takes a complex sentence shown in (4**Complex**) as input and generates its simplified version shown in (4**Simple**). Typically, a simplified sentence differs from a complex one in that it involves simpler, more usual and often shorter, words (e.g., *wrote* instead of *published*); simpler syntactic constructions (e.g., no relative clauses or apposition or coordination); and

fewer modifiers (e.g., *his paper* vs. *his second paper*). In practice, simplification is thus often modeled using four main operations: *splitting* a complex sentence into several simpler sentences; *dropping* and *reordering* phrases or constituents; *substituting* words/phrases with simpler ones.

(10) **Complex:** In 1964 Peter Higgs published his second paper in Physical Review Letters describing Higgs mechanism which predicted a new massive spin-zero boson for the first time.
Simple: Peter Higgs wrote his paper explaining Higgs mechanism in 1964. Higgs mechanism predicted a new elementary particle.

6.1.1 Potential Applications

The popularity of sentence simplification comes from its potential relevance to various applications.

Societal application Since sentence simplification helps users read sentences easily and faster, it helps in the development of many societal applications. For example, sentence simplification can be a useful asset for people with reading disabilities [Inui *et al.*, 2003] such as aphasia [Carroll *et al.*, 1999b], individuals with low-literacy [Watanabe *et al.*, 2009], or children and non-native speakers learning English [Siddharthan, 2002].

Preprocessing step for NLP Applications Not only humans but also NLP applications can benefit from sentence simplification. Complex sentences are often problematic for NLP tasks. Sentence simplification has been used to preprocess the input of those tasks in order to improve their performance. Chandrasekar *et al.* (1996) used sentence simplification to facilitate parsers and machine translation systems; Knight and Marcu (2000) and Beigman Klebanov *et al.* (2004) used it for summarisation; Filippova and Strube (2008) for sentence fusion; Vickrey and Koller (2008) for semantic role labelling; Heilman and Smith (2009) for question generation; Zhao *et al.* (2009) for paraphrase generation and Jonnalagadda and Gonzalez (2009) for biomedical information extraction.

Instead of assisting by preprocessing the complex input, sentence simplification has also been used directly to generate output in a specific limited format as subtitles [Daelemans *et al.*, 2004].

6.2 Related Work

Shardlow (2014) explores the task of simplification from its very beginning. In this

section however, we discuss few related works surrounding our research.

6.2.1 Handcrafted rules for Syntactic Simplification

Earlier work on sentence simplification relied on handcrafted rules to capture syntactic simplification e.g., to split coordinated and subordinated sentences into several, simpler clauses or to model active/passive transformations [Chandrasekar and Srinivas, 1997; Siddharthan, 2002; Canning, 2002; Siddharthan, 2006; Siddharthan, 2010; Siddharthan, 2011; Bott *et al.*, 2012]. Systems based on these approaches argue that rules for syntactic simplification are difficult to learn from corpora, as difficult morphology and tense variations have to be learned from specific instances seen in the corpus. They propose to code these rules directly into the system.

Some systems worked on flat representations e.g., chunked text [Siddharthan, 2002; Chandrasekar and Srinivas, 1997] while others define rules applying to hierarchical representations such as dependency or phrase-based parses [Bott *et al.*, 2012; Canning, 2002; Siddharthan, 2011] and semantic representations [Siddharthan, 2010; Siddharthan and Mandya, 2014]. Siddharthan (2010) compared dependency tree, phrase-based parse trees and semantic representations and concluded that phrasal parse trees were inadequate for learning complex lexico-syntactic transformation rules and that dependency structures were better suited to the task.

```
Rule: Passive2Active
1) Delete
   a) nsubjpass(?X0, ?X1)
   b) auxpass(?x0, ?X2)
   c) agent(?X0, ?X3)

2) Insert
   a) nsubj(?X0, ?X3)
   b) dobj(?X0, ?X1)

3) Node Operations
   a) AGR-TENSE: ?X0 <-- ?X2
   b) AGR-NUMBER: ?X0 <-- ?X3
```

```
Rule: RelativeClause
1) Delete
   a) rcmod(?X0, ?X1)
   b) nsubj(?X1, ?X0)

2) Insert
   a) nsubj(?X1, ?X0)
```

Figure 6.1: Handwritten rules simplifying syntactic cases: Passive to Active and Relative Clause.

(11) (a) The cat was chased by a dog that was barking.
(b) A dog chased the cat. A dog was barking.

```
det(cat-2, The-1)
nsubjpass(chased-4, cat-2)
auxpass(chased-4, was-3)
det(dog-7, a-6)
agent(chased-4, dog-7)
nsubj(barking-10, dog-7)
aux(barking-10, was-9)
rcmod(dog-7, barking-10)
```

```
det(cat-2, The-1)
dobj(chased-4, cat-2)
det(dog-7, a-6)
nsubj(chased-4, dog-7)
aux(barking-10, was-9)
nsubj(barking-10, dog-7)
```

Figure 6.2: Typed dependency structures of Sentences 11(a) and 11(b).

In Figure 6.1 and 6.2, we show an example of handcrafted rules based on typed dependency structures for syntactic simplification. We have borrowed this example from Siddharthan and Mandya (2014). Figure 6.1 shows rules for passive to active conversion and relative clause simplification. Figure 6.2(left) shows the dependency structure of the complex sentence 11(a). The rule for relative clause removes the embedding "rcmod" relation, when there is a subject available for the verb in the relative clause. The rule for passive to active deletes relations "nsubjpass", "auxpass" and "agent" and inserts two new relations "nsubj" and dobj" making the sentence active. Node operations changes the tense and number information of words accordingly. After the application of these two rules on 6.2(left) we are left with the dependency structure shown in 6.2(right) which generates the simplified sentence 11(b). We refer the reader to [de Marneffe *et al.*, 2006] for more detailed explanations of the typed dependency relations.

While these hand-crafted approaches can encode precise and linguistically well-informed syntactic transformation (using e.g., detailed morphological and syntactic information), they are limited in scope to purely syntactic rules and do not account for lexical simplifications and their interaction with the sentential context. Siddharthan and Mandya (2014) therefore proposes an approach where hand-crafted syntactic simplification rules are combined with lexical simplification rules extracted from aligned English and simple English sentences, and revision histories of Simple Wikipedia. They show that such a hybrid symbolic/statistical approach outperforms state of the art systems in a human evaluation based on 25 sentences.

6.2.2 Statistical framework for Sentence Simplification

Using the parallel dataset formed by Simple English Wikipedia (SWKP)[18] and tra- SWKP

[18] http://simple.wikipedia.org

EWKP

ditional English Wikipedia (EWKP)[19] , more recent work has focused on developing machine learning approaches to sentence simplification. Yatskar *et al.* (2010) explore data-driven methods to learn lexical simplifications from Wikipedia revision histories. Zhu *et al.* (2010) learns a simplification model inspired by *syntax-based* machine translation [Yamada and Knight, 2001] which encodes the probabilities for four rewriting operations on the parse tree of an input sentences namely, substitution, reordering, splitting and deletion. Woodsend and Lapata (2011) learn a quasi-synchronous grammar [Smith and Eisner, 2006] describing a loose alignment between parse trees of complex and of simple sentences. Following Dras (1999), they then generate all possible rewrites for a source tree and use integer linear programming to select the most appropriate simplification. Coster and Kauchak (2011) and Wubben *et al.* (2012) view simplification as a monolingual translation task where the complex sentence is the source and the simpler one is the target.

Zhu *et al.* (2010) constructed a parallel corpus (PWKP) of complex/simple sentences by aligning sentences from EWKP and SWKP. PWKP has later been used for training various state-of-the-art supervised systems [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Wubben *et al.*, 2012] including ours (described in Chapter 7). In what follows, we start with describing PWKP and then we discuss supervised systems [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Coster and Kauchak, 2011; Wubben *et al.*, 2012] one by one in more detail.

6.2.2.1 Parallel Complex-Simple Sentence-Aligned Wikipedia Corpus

Wikipedia defines SWKP as a resource for students, children, people who are learning English and anyone who are unable to understand hard ideas or topics. The authors of SWKP are advised to use basic English, i.e., use easy words and short sentences. This make SWKP a very useful resource for sentence simplification and has been used for constructing parallel corpus of comple-simple sentence pairs by aligning it with EWKP. The sentences from EWKP and SWKP are considered as "complex" and "simple" respectively. Here, we describe Zhu *et al.* (2010)'s parallel Wikipedia corpus.

Zhu *et al.* (2010) extracted 65,133 paired articles from EWKP and SWKP using the Wikipedia dump files[20]. As a preprocessing step, both SWKP and EWKP were processed for removing Wiki tags, sentence boundary detection, tokenisation and lemmatisation. Zhu *et al.* then aligned sentences from EWKP and SWKP using TF*IDF [Nelken and Shieber, 2006]. In order to account for sentence splitting, they

[19] `http://en.wikipedia.org`
[20] `http://download.wikimedia.org`

Number of Sentence Pairs	Average Sentence Length		Average Token Length	
	complex	simple	complex	simple
108,016	25.01	20.87	5.06	4.89

Table 6.1: Statistics for the PWKP dataset.

	Number of Simple Sentences Aligned to a Complex Sentence						
	1	2	3	4	5	6	11
Number of Pairs	101,420	6,326	246	16	3	4	1

Table 6.2: Split distribution in the PWKP dataset.

allowed mapping of 1 to n adjacent simple sentences to one complex sentence. As a result, they constructed a parallel Wikipedia corpus (PWKP) of 108,016/114,924 complex/simple sentences. PWKP

Table 6.1 and 6.2 show the statistics of PWKP. Both the average sentence length and average token length in simple sentences are shorter than those in complex sentences which is in line with the assumption of Simple Wikipedia of using smaller words and sentences. Table 6.2 shows the split distribution of PWKP. As we can see, PWKP has 101,420 pairs which does not have any split and only 6,596 pairs have one or more splits.

Zhu *et al.* (2010) randomly selected 100 pairs (100/131 complex-simple sentences) from the PWKP corpus for evaluation. Most of the state-of-the-art systems have used this corpus for their evaluations. Henceforth, we refer this evaluation corpus by the PWKP Evaluation Corpus . PWKP Evaluation Corpus

6.2.2.2 Monolingual tree-based translation model

Zhu *et al.* (2010) proposed a simplification model inspired by *syntax-based* machine translation [Yamada and Knight, 2001]. Given a complex sentence to simplify, Zhu *et al.* (2010) parse the complex sentence and applies four operations: splitting, dropping, reordering and substitution; on the parse tree to simplify. We show an example from [Zhu *et al.*, 2010] to explain how the simplification proceeds.

(12) (a) August was the sixth month in the ancient Roman calender which started in 735BC.
(b) August was the sixth month in the old calender. The old calender started in 735BC.

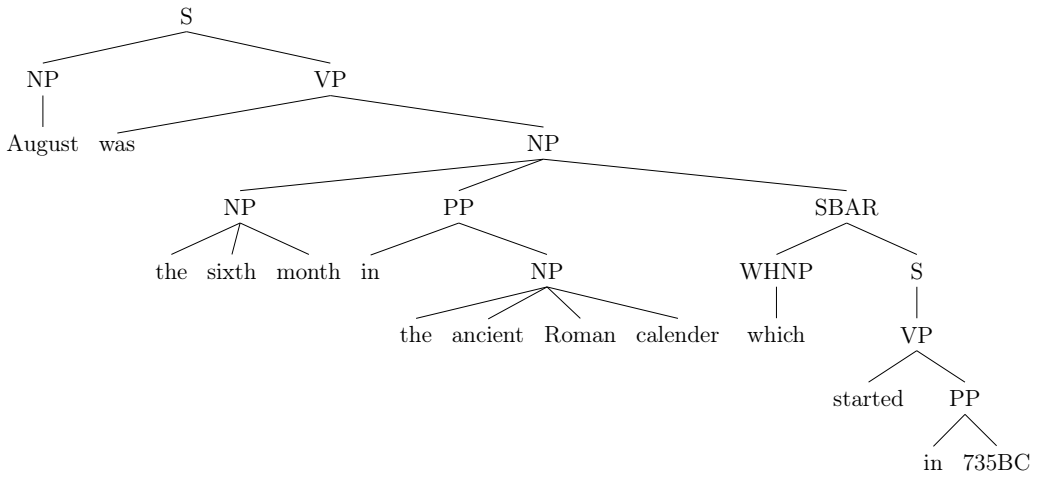


Figure 6.3: Parse tree of the complex sentence 12(a).

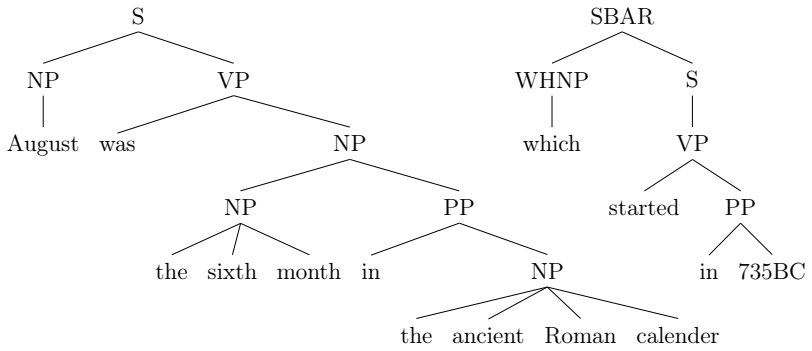


Figure 6.4: Splitting: Segmentation.

Splitting proceeds in two steps: segmentation and completion. The segmentation step decides if we can split the parse tree by judging the syntactic constituent of the split boundary word and the length of the complex sentence. Given the parse tree of the complex sentence 12(a) shown in Figure 6.3, it decides if we could split at the boundary word "which" with the syntactic constituent "SBAR". Figure 6.4 shows the segmented parse tree in the case of a successful split. Splitting operation does not end here and undergoes through a completion step. The completion step reconstructs the sentences after split; it decides weather to drop or retained the boundary word using two features: the boundary word and its direct constituents; and it copies the necessary parts to complete the new sentences. Thy last part is judged by two features: the dependency relations and the constituent. Figure 6.5 shows the most likely output of the completion step where the boundary word "which" has been dropped and the whole NP phrase "the ancient Roman calender" has been copied to

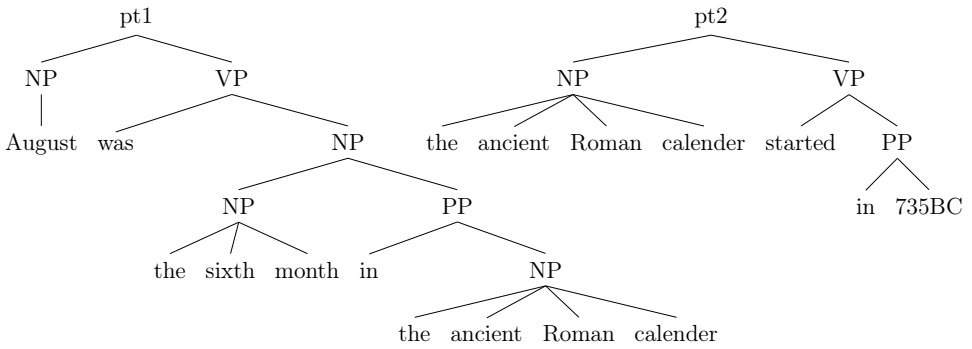


Figure 6.5: Splitting: Completion.

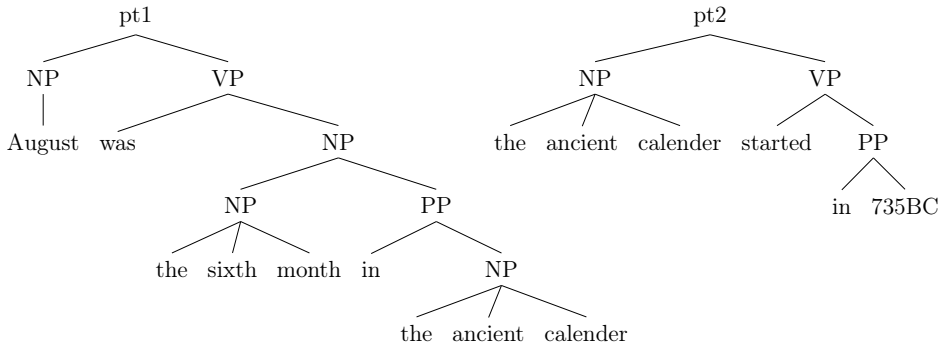


Figure 6.6: Dropping and Reordering.

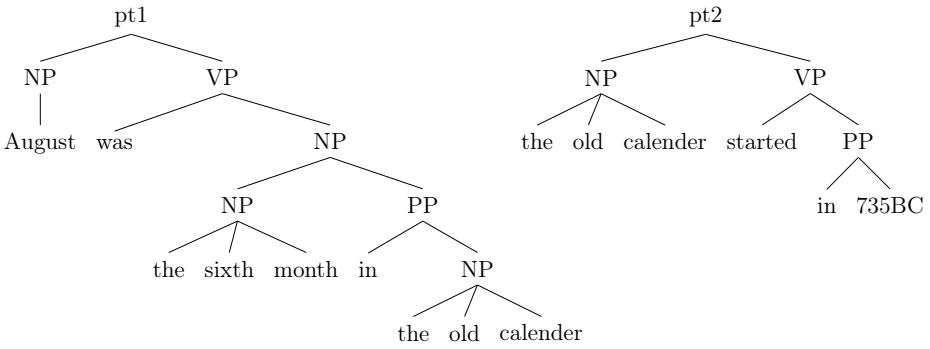


Figure 6.7: Substitution.

the sentence on the right.

After the splitting step, we then try to drop and reorder each non terminal node in the parse tree from top to bottom. We use the same features for both operations: the node's direct constituent and its children's constituent pattern. Dropping decided if a non-terminal being processed should be dropped or not, whereas reordering decided the order among the children of the non-terminal. Figure 6.6 shows the most likely

result of dropping and reordering on the parse trees shown in Figure 6.5.

Substitution step considers the parse tree for word and phrase substitutions. Word substitution happens on the terminal nodes of the parse tree, whereas phrase substitution happens on the non-terminal nodes. Figure 6.7 shows the final results after the substitution step (replacing "ancient" by "old") producing the final simplified sentences 12(b).

Zhu *et al.* (2010) use the PWKP corpus to learn their simplification model which encodes the probabilities for four rewriting operations on the parse tree of an input sentences. Following Yamada and Knight (2001), EM algorithm has been used for the estimation of the model parameters. In addition, the simplification model is combined with a language model to improve grammaticality. The decoder translates sentences into simpler ones by greedily selecting the output sentence with highest probability.

They evaluate their system on the PWKP evaluation corpus using BLEU and NIST scores, as well as various readability scores such as Flesch Reading Ease test and n-gram language model perplexity. Although their system outperforms several baselines at the readability scores, they do not perform better at BLEU or NIST scores.

6.2.2.3 Quasi-synchronous grammar and Integer linear programming

Using both the PWKP corpus and the edit history of Simple Wikipedia, Woodsend and Lapata (2011) learn a quasi synchronous grammar [Smith and Eisner, 2006] describing a loose alignment between parse trees of complex and of simple sentences. In comparison to Yatskar *et al.* (2010) who only learn learn lexical simplification rules, Woodsend and Lapata (2011) learn syntactic simplification rules, lexical simplification rules and sentence splitting rules.

Figure 6.8 shows the word alignment (marked by the dotted lines) between the parse tree of the complex sentence 13(a) and its simplified version 13(b). Based on these alignments, two quasi-synchronous grammar rules (Table 6.3) have been learned: one for lexical simplification and another for sentence splitting.

(13) (a) John Smith walked his dog and afterwards met Mary.
(b) John Smith walked his dog. He met Mary later.

Following Dras (1999), they then generate all possible rewrites for a source tree using the learned quasi synchronous grammar and use integer linear programming to select the most appropriate simplification.

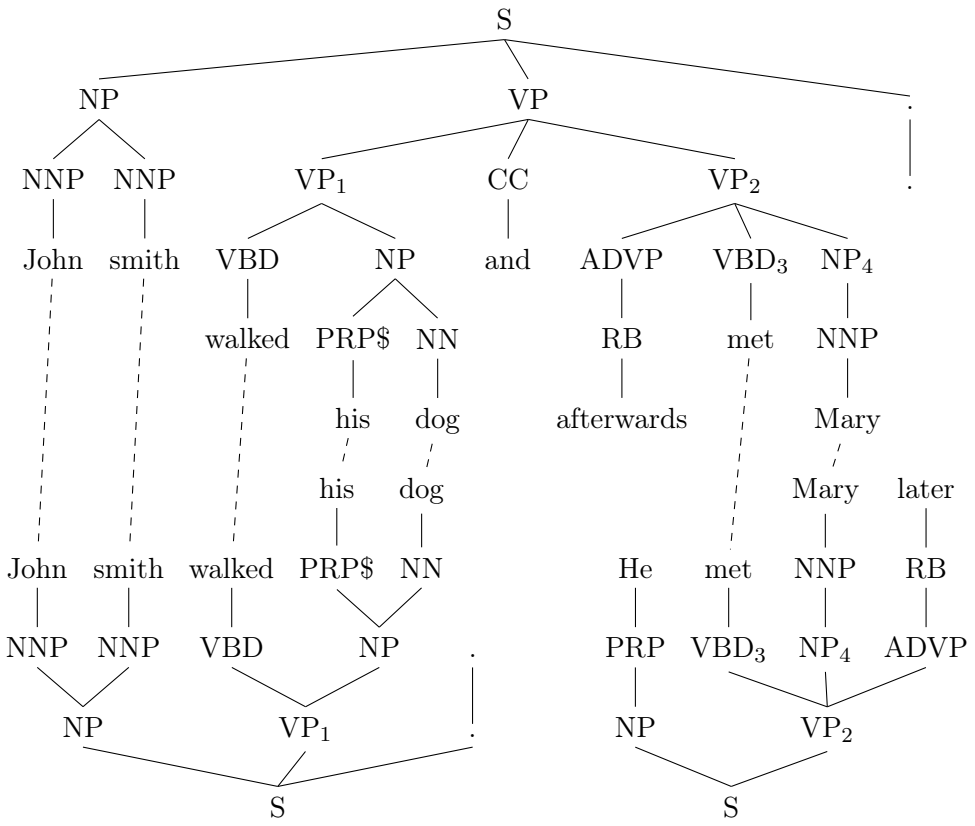


Figure 6.8: Word alignment in the parse trees of complex 13(a) and simple 13(b) sentences.

Lexical substitution:
$\langle$VP, VP$\rangle \rightarrow \langle$ [ADVP [RB afterwards] VBD_3 NP_4], [VBD_3 NP_4 ADVP [RB later]]$\rangle$
Splitting:
$\langle$VP, VP, S$\rangle \rightarrow \langle$ [VP_1 and VP_2], [VP_1], [NP [PRP He] VP_2] $\rangle$

Table 6.3: Quasi synchronous grammar learned from the alignment.

They also evaluate their model on the PWKP evaluation corpus and show that they achieve better BLEU score than Zhu *et al.* (2010). They also conducted a human evaluation on randomly selected 64 of the 100 test sentences and showed again a better performance in terms of simplicity, grammaticality and meaning preservation.

6.2.2.4 Simplification as monolingual machine translation

Coster and Kauchak (2011) and Wubben *et al.* (2012) view simplification as a monolingual translation task where the complex sentence is the source and the simpler

one is the target.

Tuning phrase alignment table. To account for deletions, reordering and substitution, Coster and Kauchak (2011) trained a phrase based machine translation system on the PWKP corpus while manually modifying the word alignment output by GIZA++ in Moses to allow for null phrasal alignments. In this way, they allow for phrases to be deleted during translation. No human evaluation is provided but the approach is shown to result in significant improvements over a traditional phrase based approach.

Dissimilarity metric Wubben *et al.* (2012) use Moses and the PWKP data to train a phrase based machine translation system augmented with a post-hoc reranking procedure designed to rank the output based on their dissimilarity from the source. A human evaluation on 20 sentences randomly selected from the test data indicates that, in terms of fluency and adequacy, their system is judged to outperform both Zhu *et al.* (2010) and Woodsend and Lapata (2011) systems.

6.3 Summary

To summarise, on one hand Zhu *et al.* (2010) and Woodsend and Lapata (2011) operate at the syntax level of the input sentence, and on the other hand, Coster and Kauchak (2011) and Wubben *et al.* (2012) operate directly on the input sentence. In contrast, we debate on using rich linguistic information in the form of deep semantic representation to improve the sentence simplification task. In our experiments, we use the Discourse Representation Structures [Kamp, 1981] assigned by Boxer [Curran *et al.*, 2007] for the deep semantic representations. We propose two methods for sentence simplification: a supervised approach to hybrid simplification using deep semantics and machine translation (Chapter 7); and an unsupervised approach to sentence simplification using the comparable Wikipedia corpora (Chapter 8). Our supervised approach is trained on the PWKP corpus and our unsupervised approach is trained on EWKP and SWKP without any sort of alignment. Both approaches are evaluated against the PWKP evaluation corpus.

Chapter 7

Hybrid Simplification using Deep Semantics and Machine Translation

Contents

This chapter is based on the following paper:

Shashi Narayan and Claire Gardent. *Hybrid Simplification using Deep Semantics and Machine Translation*, in Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL), Baltimore, June 2014.

In this chapter, we present a hybrid approach to sentence simplification which combines deep semantics and monolingual machine translation to derive simple sentences from complex ones. The approach differs from previous work in two main ways. First, it is semantically based in that it takes as input a deep semantic representation rather than e.g., a sentence or a parse tree. Second, it combines a simplification model for splitting and deletion with a monolingual translation model for phrase substitution and reordering. When compared against current state of the art methods, our model yields significantly simpler output that is both grammatical and meaning preserving.

7.1 Introduction

On one hand, the machine-learning approaches based on parse trees [Zhu *et al.*, 2010; Woodsend and Lapata, 2011] face the problems of identifying the split boundary, the re-creating of the shared element in the split sentences and avoiding the deletion of obligatory arguments, and on the other hand, the machine-learning approaches based on monolingual machine translation with complex sentences as source and simple sentences as target [Coster and Kauchak, 2011; Wubben *et al.*, 2012] do not achieve significant amount of deletion or splitting at all.

In this chapter, we present a hybrid approach to sentence simplification which departs from this previous work in two main ways.

First, it combines a model encoding probabilities for splitting and deletion with a monolingual machine translation module which handles reordering and substitution. In this way, we exploit the ability of statistical machine translation (SMT) systems to capture phrasal/lexical substitution and reordering while relying on a dedicated probabilistic module to capture the splitting and deletion operations which are less well (deletion) or not at all (splitting) captured by SMT approaches.

Second, our approach is semantically based. While previous simplification approaches starts from either the input sentence or its parse tree, our model takes as input a deep semantic representation namely, the Discourse Representation Structure (DRS, [Kamp, 1981]) assigned by Boxer [Curran *et al.*, 2007] to the input complex sentence. As we shall see in Section 7.3, this permits a linguistically principled account of the splitting operation in that semantically shared elements are taken to be

the basis for splitting a complex sentence into several simpler ones; this facilitates completion (the re-creation of the shared element in the split sentences); and this provide a natural means to avoid deleting obligatory arguments.

Our approach is supervised in that it is trained on the PWKP corpus. We evaluate our approach on the PWKP evaluation corpus. When compared against current state of the art methods [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Wubben *et al.*, 2012], our model yields significantly simpler output that is both grammatical and meaning preserving.

7.2 Simplification Framework

We start by motivating our approach and explaining how it relates to previous proposals w.r.t., the four main operations involved in simplification namely, splitting, deletion, substitution and reordering. We then introduce our framework.

Sentence Splitting. Sentence splitting is arguably semantically based in that in many cases, splitting occurs when the same semantic entity participates in two distinct events. For instance, in example (14) below, the split is on the noun *bricks* which is involved in two eventualities namely, *"being resistant to cold"* and *"enabling the construction of permanent buildings"*.

(14) **C.** Being more resistant to cold, bricks enabled the construction of permanent buildings.
S. Bricks were more resistant to cold. Bricks enabled the construction of permanent buildings.

While splitting opportunities have a clear counterpart in syntax (i.e., splitting often occurs whenever a relative, a subordinate or an appositive clause occurs in the complex sentence), completion i.e., the reconstruction of the shared element in the second simpler clause, is arguably semantically governed in that the reconstructed element corefers with its matching phrase in the first simpler clause. While our semantically based approach naturally accounts for this by copying the phrase corresponding to the shared entity in both phrases, syntax based approach such as Zhu *et al.* (2010) and Woodsend and Lapata (2011) will often fail to appropriately reconstruct the shared phrase and introduce agreement mismatches because the alignment or rules they learn are based on syntax alone. For instance, in example (15), Zhu *et al.* (2010) fail to copy the shared argument *"The judge"* to the second clause whereas Woodsend and Lapata (2011) learn a synchronous rule matching (VP and VP) to

(VP. NP(It) VP) thereby failing to produce the correct subject pronoun (*"he"* or *"she"*) for the antecedent *"The judge"*.

(15) **C.** The judge ordered that Chapman should receive psychiatric treatment in prison and sentenced him to twenty years to life.
S_1. The judge ordered that Chapman should get psychiatric treatment. In prison and sentenced him to twenty years to life. [Zhu *et al.*, 2010]
S_2. The judge ordered that Chapman should receive psychiatric treatment in prison. It sentenced him to twenty years to life. [Woodsend and Lapata, 2011]

Deletion. By handling deletion using a probabilistic model trained on semantic representations, we avoid deleting obligatory arguments. Thus in our approach, semantic subformulae which are related to a predicate by a core thematic roles (e.g., *agent* and *patient*) are never considered for deletion. By contrast, syntax based approaches [Zhu *et al.*, 2010; Woodsend and Lapata, 2011] do not distinguish between optional and obligatory arguments. For instance Zhu *et al.* (2010) simplifies (16C) to (16S) thereby incorrectly deleting the obligatory theme (*gifts*) of the complex sentence and modifying its meaning to *giving knights and warriors* (instead of *giving gifts to knights and warriors*).

(16) **C.** Women would also often give knights and warriors gifts that included thyme leaves as it was believed to bring courage to the bearer.
S. Women also often give knights and warriors. Gifts included thyme leaves as it was thought to bring courage to the saint. [Zhu *et al.*, 2010]

We also depart from Coster and Kauchak (2011) who rely on null phrasal alignments for deletion during phrase based machine translation. In their approach, deletion is constrained by the training data and the possible alignments, independent of any linguistic knowledge.

Substitution and Reordering SMT based approaches to paraphrasing [Barzilay and Elhadad, 2003; Bannard and Callison-Burch, 2005] and to sentence simplification [Wubben *et al.*, 2012] have shown that by utilising knowledge about alignment and translation probabilities, SMT systems can account for the substitutions and the reorderings occurring in sentence simplification. Following on these approaches, we therefore rely on phrase based SMT to learn substitutions and reordering. In addition, the language model we integrate in the SMT module helps ensuring better fluency and grammaticality.

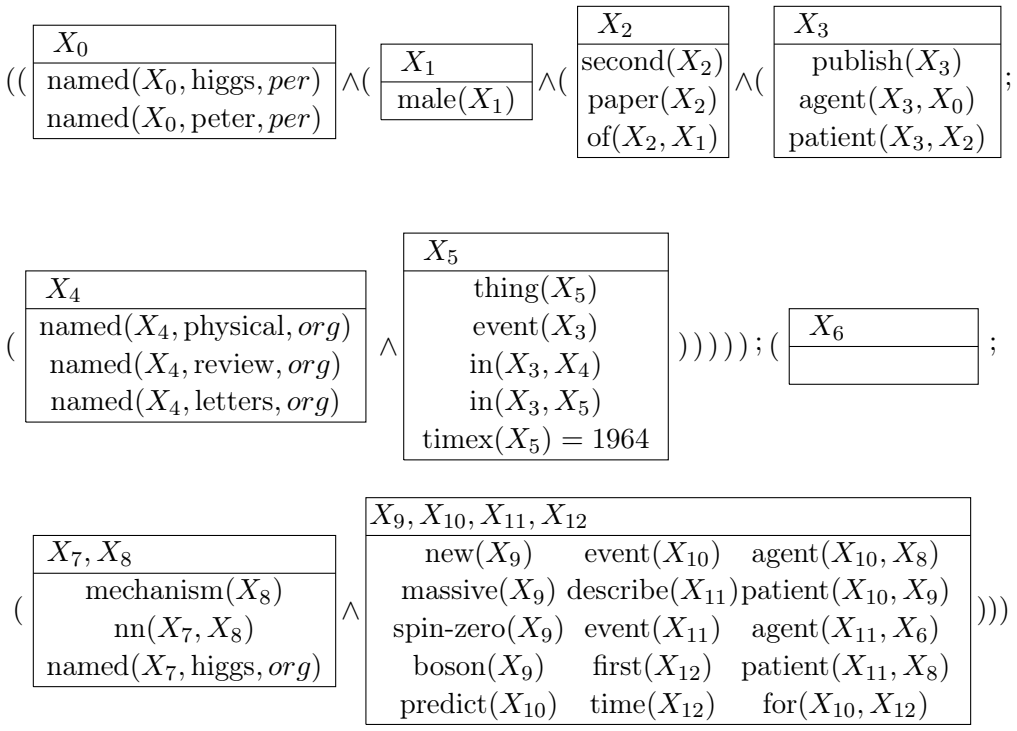


Figure 7.1: Discourse Representation Structure of the complex sentence 17(C) produced by BOXER.

node	pos. in S	predicate/type
X_0	3, 4	higgs/per, peter/per
X_1	6	male/a
X_2	6, 7, 8	second/a, paper/a
X_3	5	publish/v, **event**
X_4	10, 11, 12	physical/org review/org, letters/org
X_5	2	thing/n, 1964
X_6	6, 7, 8	−−
X_7	15	higgs/org
X_8	14, 15, 16	mechanism/n
X_9	19, 20, 21 22, 23	new/a, spin-zero/a massive/a, boson/n
X_{10}	18	predict/v, **event**
X_{11}	13	describe/v, **event**
X_{12}	25, 26, 27	first/a, time/n
O_1	17	which/WDT

rel	pos. in S	predicate
R_1	5	$agent, X_3 \rightarrow X_0$
R_2	5	$patient, X_3 \rightarrow X_2$
R_3	6	$of, X_2 \rightarrow X_1$
R_4	9	$in, X_3 \rightarrow X_4$
R_5	1	$in, X_3 \rightarrow X_5$
R_6	13	$agent, X_{11} \rightarrow X_6$
R_7	13	$patient, X_{11} \rightarrow X_8$
R_8	−−	$nn, X_8 \rightarrow X_7$
R_9	18	$agent, X_{10} \rightarrow X_8$
R_{10}	18	$patient, X_{10} \rightarrow X_9$
R_{11}	24	$for, X_{10} \rightarrow X_{12}$

Figure 7.2: Node table and Relation table for DRS graph.

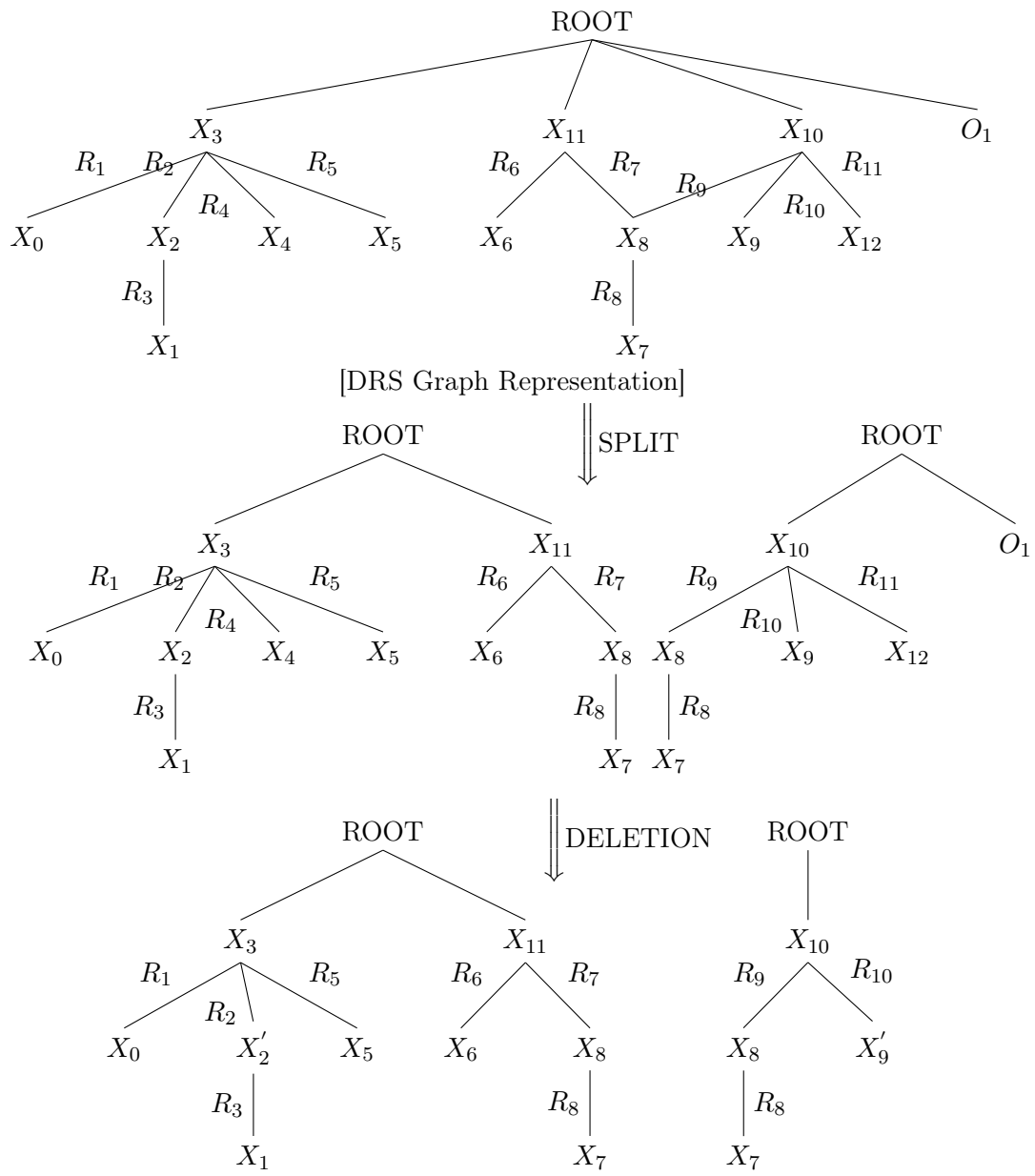


Figure 7.3: Simplification of the complex sentence 17(C) to the simple sentences 17(S).

7.2.1 An Example

Figure 7.1, 7.2 and 7.3 shows how our approach simplifies (17C) into (17S). The simplification model is described in the section below.

(17) **C.** In 1964 Peter Higgs published his second paper in Physical Review Letters describing the Higgs mechanism which predicted a new massive spin-zero boson for the first time.
S. Peter Higgs wrote his paper explaining the Higgs mechanism in 1964. The Higgs mechanism predicted a new elementary particle.

The DRS for (17C) produced using Boxer [Curran *et al.*, 2007] is shown in Figure 7.1. To facilitate simplification, we extract a graph (Figure 7.3, topmost) capturing the dependencies between the variables of the DRS.

The DRS to graph conversion goes through several preprocessing steps: the relation *nn* is inverted making modifier noun (*higgs*) dependent of modified noun (*mechanism*) to facilitate deletion of modifier noun, *named* and *timex* are converted to unary predicates, e.g., $named(x, peter)$ is mapped to $peter(x)$ and $timex(x) = 1964$ is mapped to $1964(x)$; and nodes are introduced for orphan words (OW) i.e., words which have no corresponding material in the DRS (e.g., *which* at position 16). After the preprocessing step, a graph, shown in Figure 7.3(topmost), is extracted such that each DRS variable labels a node in the graph and each edge is labelled with the relation holding between the variables labelling its end vertices. The two tables in Figure 7.2 show the predicates (left table) associated with each variable and the relation label (right table) associated with each edge. Boxer also outputs the associated positions in the complex sentence for each predicate (not shown in the DRS but in the graph tables). Orphan words are added to the graph (node O_1 for *which* at position 16) thus ensuring that the position set associated with the graph exactly matches the positions in the input sentence and thus deriving the input sentence.

Split Candidate	isSplit	prob.
(*agent*, *for*, *patient*) - (*agent*, *in*, *in*, *patient*)	true	0.63
	false	0.37

Table 7.1: Simplification: SPLIT.

Given the input DRS shown in Figure 7.3, simplification proceeds as follows.

Splitting. The splitting candidates of a DRS are event pairs contained in that DRS. More precisely, the splitting candidates are pairs[21] of event variables associated

[21]The splitting candidates could be sets of event variables depending on the number of splits required. Here, we consider pairs for 2 splits.

with at least one of the core thematic roles (e.g., *agent* and *patient*). The features conditioning a split are the set of thematic roles associated with each event variable. The DRS shown in Figure 7.3 contains three such event variables X_3, X_{11} and X_{10} with associated thematic role sets *{agent, in, in, patient}*, *{agent, patient}* and *{agent, for, patient}* respectively. Hence, there are 3 splitting candidates (X_3-X_{11}, X_3-X_{10} and X_{10}-X_{11}) and 4 split options: no split or split at one of the splitting candidates. Here the split with highest probability (cf. Table 7.1) is chosen and the DRS is split into two sub-DRS, one containing X_3, and the other containing X_{10}. After splitting, dangling subgraphs are attached to the root of the new subgraph maximizing either proximity or position overlap. Here the graph rooted in X_{11} is attached to the root dominating X_3 and the orphan word O_1 to the root dominating X_{10}.

Deletion. The deletion model (cf. Table 7.2) regulates the deletion of relations and their associated subgraph; of adjectives and adverbs; and of orphan words. Here, the relations *in* between X_3 and X_4 and *for* between X_{10} and X_{12} are deleted resulting in the deletion of the phrases *"in Physical Review Letters"* and *"for the first time"* as well as the adjectives *second, massive, spin-zero* and the orphan word *which*.

Substitution and Reordering. Finally the translation and language model ensures that *published, describing* and *boson* are simplified to *wrote, explaining* and *elementary particle* respectively; and that the phrase *"In 1964"* is moved from the beginning of the sentence to its end.

7.2.2 The Simplification Model

DRS Simplification Model

Our simplification framework consists of a probabilistic model for splitting and dropping which we call DRS simplification model (DRS-SM), a phrase based translation model for substitution and reordering (PBMT), and a language model learned on Simple English Wikipedia (LM) for fluency and grammaticality. Given a complex sentence c, we split the simplification process into two steps. First, DRS-SM is applied to D_c (the DRS representation of the complex sentence c) to produce one or more (in case of splitting) intermediate simplified sentence(s) s'. Second, the simplified sentence(s) s' is further simplified to s using a phrase based machine translation system (PBMT+LM). Hence, our model can be formally defined as:

relation candidate		isDrop	probability
relation word	length range		
in	0-2	true	0.22
		false	0.72
in	2-5	true	0.833
		false	0.167

modifier candidate	isDrop	probability
modifier word		
new	true	0.22
	false	0.72
massive	true	0.833
	false	0.167

OW candidate		isDrop	probability
orphan word	isBoundary		
and	true	true	0.82
		false	0.18
which	false	true	0.833
		false	0.167

Table 7.2: Simplification: DELETION (Relations, modifiers and OW respectively).

$$
\begin{aligned}
\hat{s} &= \operatorname*{argmax}_s p(s|c) \\
&= \operatorname*{argmax}_s \sum_{s'} p(s, s'|c) \\
&= \operatorname*{argmax}_s \sum_{s'} p(s|s', c)p(s'|c) \\
&= \operatorname*{argmax}_s \sum_{s'} p(s'|c)p(s|s') \\
&= \operatorname*{argmax}_s \sum_{s'} p(s'|\mathrm{D}_c)p(s'|s)p(s)
\end{aligned}
$$

where the probabilities $p(s'|\mathrm{D}_c)$, $p(s'|s)$ and $p(s)$ are given by the DRS simplification model, the phrase based machine translation model and the language model respectively.

To get the DRS simplification model, we combine the probability of splitting with the probability of deletion:

$$p(s'|\mathrm{D}_c) = \sum_{\theta:str(\theta(\mathrm{D}_c))=s'} p(\mathrm{D}_{split}|\mathrm{D}_c)p(\mathrm{D}_{del}|\mathrm{D}_{split})$$

where θ is a sequence of simplification operations and $str(\theta(\mathrm{D}_c))$ is the sequence of words associated with a DRS resulting from simplifying D_c using θ.

The probability of a splitting operation for a given DRS D_c is:

$$p(\mathrm{D}_{split}|\mathrm{D}_c) = \begin{cases} \textsc{split}(sp^{\text{true}}_{\text{cand}}), & \text{split at } sp_{\text{cand}} \\ \prod_{sp_{\text{cand}}} \textsc{split}(sp^{\text{false}}_{\text{cand}}), & \text{otherwise} \end{cases}$$

That is, if the DRS is split on the splitting candidate sp_{cand}, the probability of the split is then given by the *SPLIT* table (Table 7.1) for the *isSplit* value "true" and the split candidate sp_{cand}; else it is the product of the probability given by the *SPLIT* table for the *isSplit* value "false" for all split candidate considered for D_c. As mentioned above, the features used for determining the split operation are the role sets associated with pairs of event variables (cf. Table 7.1).

The deletion probability is given by three models: a model for relations determining the deletion of prepositional phrases; a model for modifiers (adjectives and adverbs) and a model for orphan words (Table 7.2). All three deletion models use the associated word itself as a feature. In addition, the model for relations uses the PP length-range as a feature while the model for orphan words relies on boundary information i.e., whether or not, the OW occurs at the associated sentence boundary.

$$p(\mathrm{D}_{del}|\mathrm{D}_{split}) = \prod_{\text{rel}_{\text{cand}}} \textsc{del}_{\text{rel}}(\text{rel}_{\text{cand}}) \prod_{\text{mod}_{\text{cand}}} \textsc{del}_{\text{mod}}(\text{mod}_{\text{cand}}) \prod_{\text{ow}_{\text{cand}}} \textsc{del}_{\text{ow}}(\text{ow}_{\text{cand}})$$

7.2.3 Estimating the Parameters

7.2.3.1 DRS Simplification Model

We use the EM algorithm [Dempster *et al.*, 1977] to estimate our split and deletion model parameters. For an efficient implementation of the EM algorithm, we follow the work of Yamada and Knight (2001) and Zhu *et al.* (2010); and build training graphs (Figure 7.4) from the pair of complex and simple sentence pairs in the training data. Yamada and Knight (2001) used it for a syntax based translation model whereas Zhu *et al.* (2010), later, used it to learn a tree based translation model for

sentence simplification.

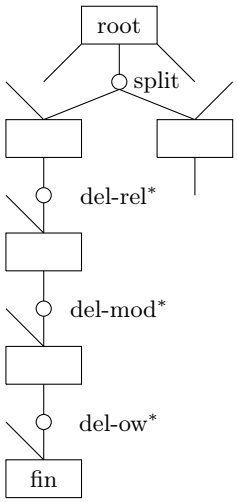


Figure 7.4: An example training graph. M-nodes are represented by triangles and O-nodes are represented by circles.

Algorithm 5 EM Algorithm

Construct training graph for each (complex, simple) sentence(s) pairs in the training data.
Initialize all probability tables using the uniform distribution.
for multiple iterations **do**
  Reset all count tables to zero
  for each training graph **do**
    Calculate Inside (β) Probabilities
    Calculate Outside (α) Probabilities
    Update count for each operation features in each O-node n of the training graph: $count = count + (\alpha_n * \beta_n / \beta_{root})$
  end for
end for

Our EM algorithm (as depicted in Algorithm 5, 6 and 7) starts with building training graphs from the training data. Each training graph represents a "complex-simple" sentence pair and consists of two types of nodes: major nodes (M-nodes) and operation nodes (O-nodes). An M-node contains the DRS representation D_c of a complex sentence c and the associated simple sentence(s) s_i while O-nodes determine split and deletion operations on their parent M-node. Only the root M-node is considered for the split operations. For example, given the root M-node $(\text{D}_c, (s_1, s_2))$, multiple successful split O-nodes will be created, each one further creating two M-nodes (D_{c1}, s_1) and (D_{c2}, s_2). For the training pair (c, s), the root M-node (D_c, s)

is followed by a single split O-node producing an M-node (D_c, s) and counting all split candidates in D_c for failed split. The M-nodes created after split operations are then tried for multiple deletion operations of relations, modifiers and OW respectively. Each deletion candidate creates a deletion O-node marking successful or failed deletion of the candidate and a result M-node. The deletion process continues on the result M-node until there is no deletion candidate left to process. The governing criteria for the construction of the training graph is that, at each step, it tries to minimize the Levenshtein edit distance between the complex and the simple sentences. Moreover, for the splitting operation, we introduce a split only if the reference sentence consists of several sentences (i.e., there is a split in the training data); and only consider splits which maximises the overlap between split and simple reference sentences.

Algorithm 6 Calculate Inside (β) Probability

for each node n from the bottom to the root of training graph **do**
  if node n is a final M-node **then**
    $\beta_n = 1$
  else if node n is an O-node **then**
    $\beta_n = \prod_{n' \in child(n)} \beta_{n'}$
  else
    $\beta_n = \sum_{n' \in child(n)} p(oper|n) * \beta_{n'}$
  end if
end for

Algorithm 7 Calculate Outside (α) Probability

for each node n from the root to the bottom of training graph **do**
  if node n is the root M-node **then**
    $\alpha_n = 1$
  else if node n is an O-node **then**
    $\alpha_n = p(oper|parent_n) * \alpha_{parent_n}$
  else
    $\alpha_n = \sum_{n' \in parent(n)} \alpha_{n'} \prod_{\substack{n'' \in child(n') \\ n'' \neq n}} \beta_{n''}$
  end if
end for

We initialize our probability tables Table 7.1 and Table 7.2 with the uniform distribution, i.e., 0.5 because all our features are binary. The EM algorithms iterates over training graphs; estimating inside and outside probabilities and counting model

features from O-nodes and updating our probability tables. The outside probability of the root M-node is 1 as it is the starting point. We assume that our supervised construction of the training graph is perfect and hence all final M-nodes have been assigned the inside probability of 1. EM algorithm (as depicted in Algorithm 5, 6 and 7) iterates over training graphs and exercises all possible paths. We refer the reader to [Yamada and Knight, 2001] for more detail.

7.2.3.2 Phrase based translation model

Our phrase based translation model is trained using the Moses toolkit[22] [Koehn *et al.*, 2007] with its default command line options on the PWKP corpus (except the sentences from the test set) considering the complex sentence as the source and the simpler one as the target. Our trigram language model is trained using the SRILM toolkit[23] [Stolcke, 2002] on the SWKP corpus[24].

7.2.4 Decoding

We explore the decoding graph similar to the training graph but in a greedy approach always picking the choice with maximal probability. Given a complex input sentence c, a split O-node will be selected corresponding to the decision of whether to split and where to split. Next, deletion O-nodes are selected indicating whether or not to drop each of the deletion candidate. The DRS associated with the final M-node D_{fin} is then mapped to a simplified sentence s'_{fin} which is further simplified using the phrase-based machine translation system to produce the final simplified sentence s_{simple}. Also, if Boxer fails to produce the DRS D_c for the complex input sentence c, the DRS simplification model is bypassed and c is directly sent to the phrase-based machine translation system to produce the final simplified sentence s_{simple}.

7.3 Experiments and Evaluations

We trained our simplification and translation models on the PWKP corpus. To evaluate performance, we compare our approach with three other state of the art systems using the PWKP evaluation corpus and relying both on automatic metrics and on human judgments.

[22] http://www.statmt.org/moses/

[23] http://www.speech.sri.com/projects/srilm/

[24] We downloaded the snapshots of Simple Wikipedia dated 2013-10-30 available at http://dumps.wikimedia.org/.

7.3.1 Training and Test Data

The DRS-Based simplification model is trained on PWKP, a bi-text of complex and simple sentences ([Zhu *et al.*, 2010], described in Chapter 6). PWKP contains 108016/114924 complex/simple sentence pairs. We tokenize PWKP using Stanford CoreNLP toolkit[25] [Manning *et al.*, 2014]. We then parse all complex sentences in PWKP using Boxer[26] to produce their DRSs. Finally, our DRS-Based simplification model is trained on 97.75% of PWKP; we drop out 2.25% of the complex sentences in PWKP which are repeated in the test set or for which Boxer fails to produce DRSs.

We evaluate our model on the PWKP evaluation corpus ([Zhu *et al.*, 2010], described in Chapter 6). Boxer produces a DRS for 96 of the 100 input sentences. These input are simplified using our simplification system namely, the DRS-SM model and the phrase-based machine translation system (Section 7.2.2). For the remaining four complex sentences, Boxer fails to produce DRSs. These four sentences are directly sent to the phrase-based machine translation system to produce simplified sentences.

7.3.2 Automatic Evaluation Metrics

To assess and compare simplification systems, two main automatic metrics have been used in previous work namely, BLEU and the Flesch-Kincaid Grade Level Index (FKG).

The FKG index is a readability metric taking into account the average sentence length in words and the average word length in syllables. In its original context (language learning), it was applied to well formed text and thus measured the simplicity of a well formed sentence. In the context of the simplification task however, the automatically generated sentences are not necessarily well formed so that the FKG index reduces to a measure of the sentence length (in terms of words and syllables) approximating the simplicity level of an output sentence irrespective of the length of the corresponding input. To assess simplification, we instead use metrics that are directly related to the simplification task namely, the number of splits in the overall (test and training) data and in average per sentences; the number of generated sentences with no edits i.e., which are identical to the original, complex one; and the average Levenshtein distance between the system's output and both the complex and the simple reference sentences.

BLEU gives a measure of how close a system's output is to the gold standard

[25] http://nlp.stanford.edu/software/corenlp.shtml
[26] http://svn.ask.it.usyd.edu.au/trac/candc, Version 1.00

simple sentence. Because there are many possible ways of simplifying a sentence, BLEU alone fails to correctly assess the appropriateness of a simplification. Moreover BLEU does not capture the degree to which the system's output differs from the complex sentence input. We therefore use BLEU as a means to evaluate how close the systems output are to the reference corpus but complement it with further manual metrics capturing other important factors when evaluating simplifications such as the fluency and the adequacy of the output sentences and the degree to which the output sentence simplifies the input.

7.3.2.1 Results and Discussion

Number of Splits Table 7.3 shows the proportion of input whose simplification involved a splitting operation. We measure percentage of split (% split) in the training and the test data from all systems; and average split per sentence (average split / sentence) per system. GOLD is the test data with the gold standard SWKP sentences; Zhu, Woodsend, Wubben are the best output of the models of Zhu *et al.* (2010), Woodsend and Lapata (2011) and Wubben *et al.* (2012) respectively; Hybrid is our model.

While our system splits in proportion similar to that observed in the training data, the other systems either split very often (80% of the time for Zhu and 63% of the time for Woodsend) or not at all (Wubben). In other words, when compared to the other systems, our system performs splits in proportion closest to the reference both in terms of total number of splits and of average number of splits per sentence.

Data	Total number of sentences	% split	average split / sentence
PWKP	108,016	6.1	1.06
GOLD	100	28	1.30
Zhu	100	80	1.80
Woodsend	100	63	2.05
Wubben	100	1	1.01
Hybrid	100	10	1.10

Table 7.3: Proportion of Split Sentences.

Number of Edits Table 7.4 indicates the average Levenshtein edit distance (LD) of the output sentences w.r.t. both the complex and the simple reference sentences as well as the number of input for which no simplification occur (No edit) per system. The right part of the table shows that our system generate simplifications which are closest to the reference sentence (in terms of edits) compared to those output

by the other systems. It also produces the highest number of simplifications which are identical to the reference. Conversely our system only ranks third in terms of dissimilarity with the input complex sentences (6.32 edits away from the input sentence) behind the Woodsend (8.63 edits) and the Zhu (7.87 edits) system. This is in part due to the difference in splitting strategies noted above : the many splits applied by these latter two systems correlate with a high number of edits.

System	BLEU	Edits (Complex to System)		Edits (System to Simple)	
		LD	No edit	LD	No edit
GOLD	100	12.24	3	0	100
Zhu	37.4	7.87	2	14.64	0
Woodsend	42	8.63	24	16.03	2
Wubben	41.4	3.33	6	13.57	2
Hybrid	53.6	6.32	4	11.53	3

Table 7.4: Automated Metrics for Simplification.

BLEU score We used Moses support tools: multi-bleu[27] to calculate BLEU scores. The BLEU scores shown in Table 7.4 show that our system produces simplifications that are closest to the reference.

In sum, the automatic metrics indicate that our system produces simplification that are consistently closest to the reference in terms of edit distance, number of splits and BLEU score.

7.3.3 Human Evaluation

The human evaluation was done online using the LG-Eval toolkit [Kow and Belz, 2012][28]. The evaluators were allocated a trial set using a Latin Square Experimental Design (LSED) such that each evaluator sees the same number of output from each system and for each test set item. During the experiment, the evaluators were presented with a pair of a complex and a simple sentence(s) and asked to rate this pair w.r.t. to *adequacy* (Does the simplified sentence(s) preserve the meaning of the input?) and *simplification* (Does the generated sentence(s) simplify the complex input?). They were also asked to rate the second (simplified) sentence(s) of the pair w.r.t. to *fluency* (Is the simplified output fluent and grammatical?). Similar to the Wubben *et al.* (2012)'s human evaluation setup, we randomly selected 20 complex sentences from the PWKP evaluation corpus and included in the human evaluation corpus: the corresponding simple (Gold) sentence from the PWKP evaluation corpus,

[27] http://www.statmt.org/moses/?n=Moses.SupportTools
[28] http://www.nltg.brighton.ac.uk/research/lg-eval/

the output of our system (Hybrid) and the output of the other three systems (Zhu, Woodsend and Wubben) which were provided to us by the system authors. The evaluation data thus consisted of 100 complex/simple pairs[29]. We collected ratings from 25 participants. All were either native speakers or proficient in English, having taken part in a Master taught in English or lived in an English speaking country for an extended period of time.

7.3.3.1 Results and Discussion

Table 7.5 shows the average ratings of the human evaluation on a slider scale from 0 to 5. Pairwise comparisons between all models and their statistical significance were carried out using a one-way ANOVA with post-hoc Tukey HSD tests and are shown in Table 7.6. With a significance level of $p < 0.05$, ◊/⧫ shows if two models are/not significantly different w.r.t. simplicity. □/■ shows if two models are/not significantly different w.r.t. fluency. △/▲ shows if two models are/not significantly different w.r.t. adequacy.

Systems	Simplification	Fluency	Adequacy
GOLD	3.52	3.92	3.54
Zhu	2.83	2.42	2.43
Woodsend	1.78	2.97	3.02
Wubben	1.97	3.82	3.91
Hybrid	3.27	3.58	3.46

Table 7.5: Average Human Ratings for simplicity, fluency and adequacy.

Systems	GOLD	Zhu	Woodsend	Wubben
Zhu	◊□△			
Woodsend	◊□▲	◊■▲		
Wubben	◊■▲	◊□△	⧫□△	
Hybrid	⧫■▲	⧫□△	◊■▲	◊■▲

Table 7.6: Pairwise comparisons between all models and their statistical significance.

With regard to simplification, our system ranks first and is very close to the manually simplified input (the difference is not statistically significant). The low rating for Woodsend reflects the high number of unsimplified sentences (24/100 in

[29]The casing of words in output sentences varied depending on the systems (e.g., output sentences were lower cased in Wubben while output sentences in Zhu and Woodsend were not). To make the human evaluation independent of casing, all sentences in the evaluation data were lower cased.

the test data used for the automatic evaluation and 6/20 in the evaluation data used for human judgments). Our system data is not significantly different from the manually simplified data for simplicity whereas all other systems are.

For fluency, our system rates second behind Wubben and before Woodsend and Zhu. The difference between our system and both Zhu and Woodsend system is significant. In particular, Zhu's output is judged less fluent probably because of the many incorrect splits it licenses. Manual examination of the data shows that Woodsend's system also produces incorrect splits. For this system however, the high proportion of non-simplified sentences probably counterbalances these incorrect splits, allowing for a good fluency score overall.

Regarding adequacy, our system is against closest to the reference (3.50 for our system vs. 3.66 for manual simplification). Our system, the Wubben system and the manual simplifications are in the same group (the differences between these systems are not significant). The Woodsend system comes second and the Zhu system third (the difference between the two is significant). Wubben's high fluency, high adequacy but low simplicity could be explained with their minimal number of edit (3.33 edits) from the source sentence.

In sum, if we group together systems for which there is no significant difference, our system ranks first (together with GOLD) for simplicity; first for fluency (together with GOLD and Wubben); and first for adequacy (together with GOLD, Wubben and Woodsend).

7.4 Conclusion

A key feature of our approach is that it is semantically based. Typically, discourse level simplification operations such as sentence splitting, sentence reordering, cue word selection, referring expression generation and determiner choice are semantically constrained. As argued by Siddharthan (2006), correctly capturing the interactions between these phenomena is essential to ensuring text cohesion. It would be interesting to see how our framework deals with such discourse level simplifications i.e., simplifications which involves manipulation of the coreference and of the discourse structure. In the PWKP data, the proportion of split sentences is rather low (6.1 %) and many of the split sentences are simple sentence coordination splits. A more adequate but small corpus is that used in [Siddharthan, 2006] which consists of 95 cases of discourse simplification. Using data from the language learning or the children reading community, it would be interesting to first construct a similar, larger scale corpus; and to then train and test our approach on more complex cases

of sentence splitting.

Chapter 8

Unsupervised Sentence Simplification using Wikipedia

Contents

In this chapter, we present a novel approach to sentence simplification which departs from previous work in two main ways. First, it requires neither hand written rules nor a training corpus of aligned standard and simplified sentences. Second, sentence splitting operates on deep semantic structure. We show (i) that the unsupervised framework we propose is competitive with four state-of-the-art supervised systems and (ii) that our semantically based approach allows for a principled and effective handling of sentence splitting.

8.1 Introduction

Constructing a good quality aligned corpora of complex and simple sentences for supervised simplification methods is not an easy task but it easily affects the performance of systems trained on them. In fact, Woodsend and Lapata (2011) debate on the issue and try to improve the performance by using the edit history of Simple Wikipedia. We also show that because of the corpus used for training, our supervised approach ([Narayan and Gardent, 2014], described in Chapter 8) lags behind the gold standard in the number of splitting.

In this chapter, we present a novel approach to sentence simplification which departs from previous work in two main ways. First, it requires neither hand written rules nor a training corpus of aligned standard and simplified sentences. Instead, we exploit non-aligned Simple and English Wikipedia to learn the probability of lexical simplifications, of the semantic of simple sentences and of optional phrases i.e., phrase which may be deleted when simplifying. Second, sentence splitting is semantic based in that it operates on deep semantic structure.

Our approach takes as input a deep semantic representation (rather than e.g., a sentence or a parse tree) and pipelines three dedicated modules for lexical simplification, sentence splitting and sentence compression. Lexical simplification is handled using simplification rules acquired from non-aligned Simple and English Wikipedia; sentence splitting is guided by a probabilistic model learned from Simple wikipedia: and deletions depend on an objective function encoding the probability of semantic dependencies and the importance of semantic arguments.

We show (i) that the unsupervised framework we propose is competitive with four state-of-the-art systems [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Wubben *et al.*, 2012; Narayan and Gardent, 2014] in producing simpler, fluent and meaning preserving sentences and (ii) that our semantically based approach allows for a principled and effective handling of sentence splitting.

8.2 Simplification Framework

Our simplification framework pipelines three dedicated modules inspired from previous work on lexical simplification, syntactic simplification and sentence compression.

The first module replaces standard words with simpler ones using a lexical simplification method [Biran *et al.*, 2011] shown to be effective for words that occur frequently in a domain. As Biran *et al.* (2011) remark, "In many scenarios, these are precisely the cases where simplification is most desirable. For rare words, it may be advisable to maintain the more complex form, to ensure that the meaning is preserved".

The second module decides whether and, if so, how to split the resulting lexically simplified sentence into several smaller sentences. Following Narayan and Gardent (2014), this module is semantically based. We use Boxer [Curran *et al.*, 2007] to map sentences to DRSs (Discourse Representation Structures [Kamp, 1981]) and input this DRS to the sentence splitting module. This module then exploits the fact that semantic representations give a clear handle on shared elements thereby facilitating the reconstruction of these elements in the sentences resulting from a split. For instance, as is illustrated by Figure 8.1, the Discourse Representation Structure of the sentence "*In 1964 Peter Higgs published his second paper in Physical Review Letters describing the Higgs mechanism which predicted a new massive spin-zero boson for the first time.*" makes clear that *the Higgs mechanism* is shared between two propositions thereby facilitating a reconstruction after splitting, where the shared element is repeated in each sentence resulting from the split. In comparison, as shown in (18) given the sentences "*Since then they have changed their name to Palladium and played alongside Amy Winehouse.*", Zhu *et al.* (2010) and Woodsend and Lapata (2011) fail to identify the shared element and/or to reconstruct the sentences resulting from the split.

(18) **C.** Since then they have changed their name to Palladium and played alongside Amy Winehouse.
S_1. Since then they have changed their name. To Palladium and played alongside Amy Winehouse. [Zhu *et al.*, 2010]
S_2. Since then they have changed their name to Palladium. It played alongside Amy Winehouse. [Woodsend and Lapata, 2011]

Finally, the third module draws on Filippova and Strube (2008)'s unsupervised sentence compression proposal to determine which phrases will be deleted, however adapts it for the sentence simplification task. As a result, deletion is unsupervised

requiring only the existence of a large corpus and of the Boxer deep parser; it requires neither a subcategorisation lexicon nor hand-crafted rules to decide which arguments are obligatory; and it finds a globally optimal compression by taking into semantic dependency and word simplicity into account.

8.2.1 An Example Simplification

Figure 8.1 illustrates the successive simplification of the complex sentence (19C) yielding the intermediate and final simplified sentences shown in (19S_1-S).

(19) **C.** In 1964 Peter Higgs published his second paper in Physical Review Letters describing the Higgs mechanism which predicted a new massive spin-zero boson for the first time.
S_1 (Lex Simp). In 1964 Peter Higgs wrote his second paper in Physical Review Letters explaining the Higgs mechanism which predicted a new massive elementary particle for the first time.
S_2 (Split). In 1964 Peter Higgs wrote his second paper in Physical Review Letters explaining the Higgs mechanism. The Higgs mechanism predicted a new massive elementary particle for the first time.
S (Deletion). In 1964 Peter Higgs wrote his paper explaining the Higgs mechanism. The Higgs mechanism predicted a new elementary particle.

First (lexical simplification), (19C) is rewritten as (19S_1) and input to Boxer. The resulting DRS is shown in Figure 8.2 and a graph representation of the dependencies between its variables is shown immediately below the sentence (19S_1) in Figure 8.1.

As also explained in Chapter 7, the DRS to graph conversion goes through several preprocessing steps: the relation *nn* is inverted making modifier noun (*higgs*) dependent of modified noun (*mechanism*), *named* and *timex* are converted to unary predicates, e.g., $named(x, peter)$ is mapped to $peter(x)$ and $timex(x) = 1964$ is mapped to $1964(x)$; and nodes are introduced for orphan words (OW) i.e., words which have no corresponding material in the DRS (e.g., *which* at position 16). Each DRS variable labels a node in the graph and each edge is labelled with the relation holding between the variables labelling its end vertices. The two tables in Figure 8.3 show the predicates (left table) associated with each variable and the relation label (right table) associated with each edge. Boxer also outputs the associated positions in the complex sentence for each predicate (not shown in the DRS but in the graph tables). Orphan words are added to the graph (e.g., node O_1 for *which* at position 16) thus ensuring that the position set associated with the graph exactly generates the input sentence.

The split and the deletion steps operate on boxer graphs simplifying it to sentences (19S_2) and finally (19S) respectively.

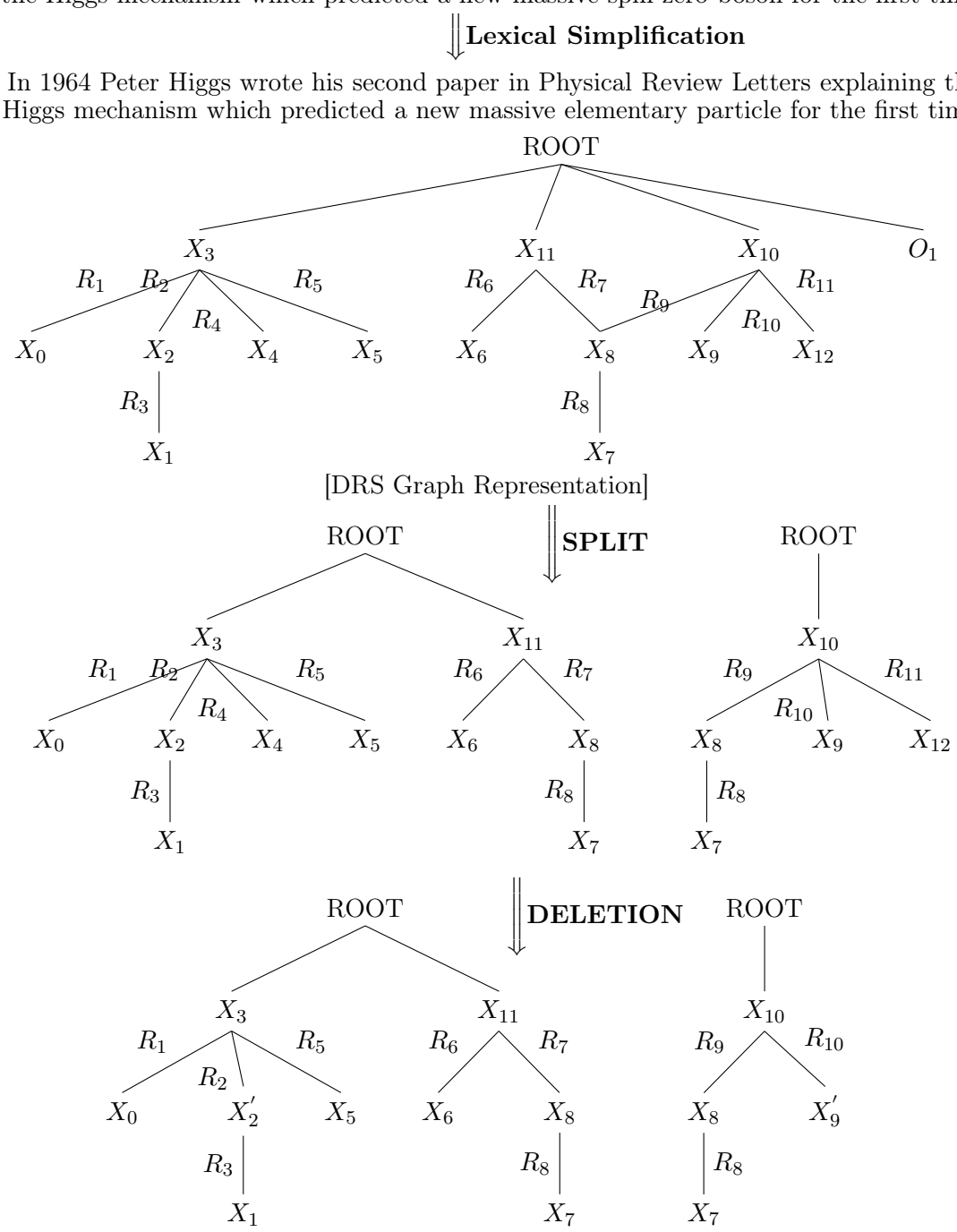


Figure 8.1: Simplification of the complex sentence 19(C) to the simple sentences 19(S).

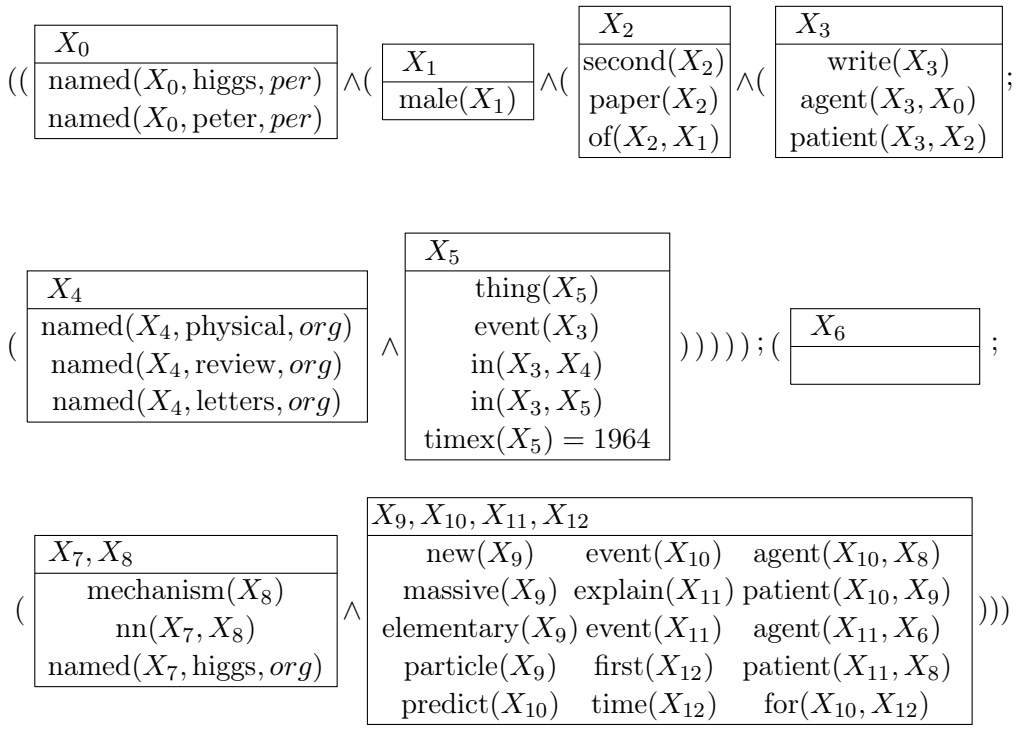


Figure 8.2: Discourse Representation Structure of the complex sentence 19(S_1) produced by BOXER.

node	pos. in S	predicate/type
X_0	3, 4	higgs/per, peter/per
X_1	6	male/a
X_2	6, 7, 8	second/a, paper/a
X_3	5	write/v, **event**
X_4	10, 11, 12	physical/org review/org, letters/org
X_5	2	thing/n, 1964
X_6	6, 7, 8	−−
X_7	15	higgs/org
X_8	14, 15, 16	mechanism/n
X_9	19, 20, 21 22, 23	new/a, elementary/a massive/a, particle/n
X_{10}	18	predict/v, **event**
X_{11}	13	explain/v, **event**
X_{12}	25, 26, 27	first/a, time/n
O_1	17	which/WDT

rel	pos. in S	predicate
R_1	5	$agent, X_3 \rightarrow X_0$
R_2	5	$patient, X_3 \rightarrow X_2$
R_3	6	$of, X_2 \rightarrow X_1$
R_4	9	$in, X_3 \rightarrow X_4$
R_5	1	$in, X_3 \rightarrow X_5$
R_6	13	$agent, X_{11} \rightarrow X_6$
R_7	13	$patient, X_{11} \rightarrow X_8$
R_8	−−	$nn, X_8 \rightarrow X_7$
R_9	18	$agent, X_{10} \rightarrow X_8$
R_{10}	18	$patient, X_{10} \rightarrow X_9$
R_{11}	24	$for, X_{10} \rightarrow X_{12}$

Figure 8.3: Node table and relation table for DRS Graph.

8.2.2 Lexical Simplification

To perform lexical simplification, we first learn lexical simplification rules from EWKP and SWKP[30] using a context-aware approach described by Biran *et al.* (2011). We then (i) select all rules that are applicable to the content words occurring in the input sentence and (ii) identify the best combination of lexical simplifications using a score designed to jointly capture grammaticality (using a language model trained on SWKP) and contextually appropriate lexical simplifications (using cosine similarity between the contextual vector of the word in the input sentence and the contextual vector of the simplification rule).

In what follows we give a brief description of how simplification rules are extracted and applied. For more details, the reader is referred to Biran *et al.* (2011).

8.2.2.1 Learning Lexical Simplification Rules

Following Biran *et al.* (2011), lexical simplification rules are learned from two comparable corpora namely, English Wikipedia and Simple English Wikipedia which are tokenized, parts-of-speech tagged and lemmatized using Stanford CoreNLP toolkit[31] [Manning *et al.*, 2014]. Lexical simplification rules are of the form:

$$r : l_c \rightarrow (l_s, F_{l_s}) \quad w_r \quad CCV_r$$

They map a complex lemma l_c to a simpler lemma l_s which is associated with the set F_{l_s} of its corresponding form/part of speech pairs (e.g., (sleeps,VBZ), (slept,VBD), (sleep, NN) etc.). To extract such rules, we consider all lemmas ($l_c \in$ EWKP and $l_s \in$ SWKP) except punctuations, stop words and numbers. First, Context vectors CV_{l_c} for each lemma l_c are built over EWKP and similarly, CV_{l_s} for each lemma l_s over SWKP. The dimension of the context vectors are the words occurring in both input corpora (EWKP and SWKP) and the value of each dimension $CV_l[i]$ is the number of occurrences of the word w_i within a ten token window surrounding an instance of the lemma l. Punctuation is disregarded also for the vector dimensions and numbers are merged into a single dimension.

We consider all pairs (l_c, l_s)s as valid lexical simplification rules if the lemma l_s is simpler than the lemma l_c. Following Biran *et al.* (2011), a lemma l_s is deemed simpler than a lemma l_c if l_s is a WordNet synonym or immediate hypernym of l_c; l_c is more complex than l_s; and l_c and l_s have a common POS tag. We use the

[30] We downloaded the snapshots of English Wikipedia dated 2013-12-31 and of Simple English Wikipedia dated 2014-01-01 available at `http://dumps.wikimedia.org`.

[31] `http://nlp.stanford.edu/software/corenlp.shtml`

same formula for lexical complexity introduced by Biran *et al.* (2011) as the product $L_l \times C_l$ where L_l is the length of the lemma l and $C_l = \frac{f_{l,EWKP}}{f_{l,SWKP}}$ with $f_{l,W}$, the frequency of l in the Wikipedia corpus W.

Each rule is furthermore associated with a weight (w_r) reflecting the cosine similarity between the context vectors of the complex (CV_{l_c}) and of the simplified word (CV_{l_s}); and with a common context vector (CCV_r) representing this similarity i.e., a vector containing the lemmas that cooccur both with l_c and with l_s ($CCV_r[i] = min(CV_{l_c}[i], CV_{l_s}[i])$).

8.2.2.2 Applying Simplification Rules

Given an input sentence and a set of simplification rules, this stage determines which word to simplify and which simplification rule to apply. For each content word w with the lemma l_c and the POS p, in the input sentence S and each simplification rule r of the form $\{r : l_c \rightarrow (l_s, F_{l_s}) \quad w_r \quad CCV_r\}$, we consider (i) the language model weight lm_{S_w} of S in which w is replaced with a form of l_s with POS p, (ii) the cosine similarity between the common context vector CCV_r of r and the context vector CV_{l_c} of l_c in S and (iii) the weight w_r of r. Using the product of these weights, we then identify the best combination of lexical simplification using the Viterbi algorithm. Here, we differ from Biran *et al.* (2011) by including the language model to boost grammaticality of the simplified sentences and using the Viterbi algorithm to explore the best solution at the sentence level. The simplification decisions are made locally or on the word by word level by Biran *et al.* (2011).

More generally, our lexical simplification process searches for simplifications (i) which maximize sentence probability (language model weight of the words in the simplified sentence); (ii) that are context aware (cosine similarity between the rule common context vector and the word vector in the input sentence) and (iii) which use highly weighted rules.

8.2.3 Sentence Splitting

Once lexical simplification has applied, we input the resulting lexically simplified sentence to Boxer[32] [Curran *et al.*, 2007] to derive its deep semantic representation in this case, a Discourse Representation Structure (DRS, [Kamp, 1981]). To determine whether and where to split the input sentence, we use a probabilistic model trained on the DRSs of the Simple Wikipedia sentences and a language model also trained on Simple Wikipedia. Intuitively, this model helps determine the most probable

[32] `http://svn.ask.it.usyd.edu.au/trac/candc`, Version 1.00

sentence combination patterns given a set of events and their thematic roles. More specifically, given the event variables contained in the DRS of the input sentence, we consider all possible splits between subsets of events and choose the split(s) with maximum split score. For instance, in the sentence shown in Figure 8.1, there are three event variables X_3, X_{10} and X_{11} in the DRS. so we will consider 5 split possibilities namely, no split ($\{X_3, X_{10}, X_{11}\}$), 2 splits resulting in three sentences describing an event each ($\{X_3\}$, $\{X_{10}\}$, $\{X_{11}\}$) and 1 split resulting in 2 sentences describing a single and two events respectively (i.e., ($\{X_3\}, \{X_{10}, X_{11}\}$) , ($\{X_3, X_{10}\}, \{X_{11}\}$) and $\{X_{10}\}, \{X_3, X_{11}\}$). The split $\{X_{10}\}, \{X_3, X_{11}\}$ gets the maximum split score and is chosen to split the sentence ($19S_1$) producing the sentences ($19S_2$). The orphan nodes (e.g., *which*) are dropped out[33] if they fall at the boundary after split.

Semantic Pattern	probability
(*agent*, *patient*)	0.059
(*agent*, *in*, *in*, *patient*)	0.002
(*agent*, *for*, *patient*)	0.002
(*agent*, *patient*), (*agent*, *for*, *patient*)	0.020
(*agent*, *patient*), (*agent*, *in*, *in*, *patient*)	0.023
(*agent*, *for*, *patient*), (*agent*, *in*, *in*, *patient*)	0.009
(*agent*, *patient*), (*agent*, *for*, *patient*), (*agent*, *in*, *in*, *patient*)	0.001

Table 8.1: Split Feature Table (SFT) showing all of the semantic patterns from Figure 8.1.

Formally, the split score P_{split} associated with the splitting of a sentence S into a sequence of sentences $s_1 ... s_n$ is defined as:

$$P_{split} = \sum_{s_i} \frac{1}{n} \times \frac{L_{split}}{L_{split} + \mid L_{split} - L_{s_i} \mid} \times lm_{s_i} \times SFT_{s_i}$$

where n is the number of sentences produced after split; L_{split} is the average length of the split sentences ($L_{split} = \frac{L_S}{n}$ where L_S is length of the sentence S); lm_{s_i} is the probability of s_i given by the language model and SFT_{s_i} is the probability of the semantic pattern associated with s_i. The Split Feature Table (SFT, Table 8.1) is derived from the corpus of DRSs associated with the SWKP sentences and counts semantic patterns consisting of events with their thematic roles in each sentence. For example the sentence after split $\{X_3, X_{11}\}$ in Figure 8.1 has a semantic pattern of $\{(agent, in, in, patient), (agent, patient)\}$. Intuitively, P_{split} favors splits involving frequent semantic patterns and sub-sentences of roughly equal length.

[33]Note that the treatment of orphan words is different here than one described in Chapter 7. In the approach described in Chapter 7, they were managed during the deletion step.

8.2.4 Phrasal Deletion

Following Filippova and Strube (2008), we formulate phrase deletion as an optimisation problem which is solved using integer linear programming[34]. Given the DRS K associated with a sentence to be simplified, for each relation $r \in K$, the deletion module determines whether r and its associated DRS subgraphs should be deleted by maximising the following objective function:

$$\sum_x x^r_{h,w} \times P(r|h) \times P(w) \quad r \notin \{agent, patient, theme, eq\}$$

where for each relation $r \in K$, $x^r_{h,w} = 1$ if r is preserved and $x^r_{h,w} = 0$ otherwise; $P(r|h)$ is the conditional probability (estimated on the DRS corpus derived from SWKP) of r given the head label h; and $P(w)$ is the relative frequency of w in SWKP.

To account for modifiers which are represented as predicates on nodes rather than relations, we preprocess the DRSs and transform each of these predicates into a single node subtree of the node it modifies. For example in Figure 8.1, the node X_2 labeled with the modifier predicate *second* is updated to a new node X'_2 dominating a child labeled with that predicate and related to X'_2 by a modifier relation.

Intuitively, the objective function will favor obligatory dependencies over optional ones and simple (i.e., frequent in SWKP) words over more minor ones. Here, we differ from Filippova and Strube (2008) by using the simplicity and not the importance of the word. We argue that the importance of the word is more relevant for sentence compression than sentence simplification. To ensure that some deletion takes place, it is subject to the following length constraint:

$$\sum_x C_r(1 - x^r_{h,w}) \geq DL \quad r \notin \{agent, patient, theme, eq\}$$

This constraint ensures that some deletion takes place (i.e., that not all nodes in the original DRS are kept). C_r is the count of words deleted when deleting the relation r. DL is the minimum length deletion. We approximate this value from the decrease in average sentence length between sentences from EWKP and SWKP and adapt it to the length of the input sentence to allow more or less deletion. We use the same connectivity constraint as Filippova and Strube (2008) to ensure that the resulting DRS is a well-formed graph.

[34]Like Filippova and Strube (2008) , in our implementation we use *lp_solve*, `http://sourceforge.net/projects/lpsolve`. *lp_solve* is a Mixed Integer Linear Programming (MILP) solver.

8.3 Experiments and Evaluations

We compare our unsupervised approach with four supervised approaches trained on the PWKP corpus using the PWKP evaluation corpus ([Zhu *et al.*, 2010], described in Chapter 6). To compare these systems, we use metrics that are directly related to the simplification task namely, the number of splits in the overall data, the number of output sentences with no edits (i.e., sentences which have not been simplified) and the average Levenshtein distance (LD) between the system output and both the complex and the simple reference sentences. We use BLEU as a means to evaluate how close the systems output are to the reference corpus. We carry out a human evaluation geared at assessing further important aspects when evaluating simplifications namely fluency, grammaticality and adequacy. And, we conducted a qualitative analysis of splits generated by various systems.

8.3.1 Automatic Evaluation

We divide our automatic evaluation in two parts. First, we evaluate the impact of various modules (lexical simplification, deletion and sentence splitting) separately and in combinations, on simplification. Second, we compare our best results with four state-of-the-art supervised systems.

8.3.1.1 Modular Evaluation

System	Levenshtein Edit distance (LD)			
	Complex to System		System to Simple	
	LD	No edit	LD	No edit
complex	0	100	12.24	3
LexSimpl	2.07	22	13.00	1
Split	2.27	51	13.62	1
Deletion	2.39	4	12.34	0
LexSimpl-Split	4.43	11	14.39	0
LexSimpl-Deletion	4.29	3	13.09	0
Split-Deletion	4.63	4	13.42	0
LexSimpl-Split-Deletion (Unsup)	6.75	3	14.29	0
GOLD (simple)	12.24	3	0	100

Table 8.2: Automated metrics for simplification: modular evaluation (A)

To assess the relative impact of each module, i.e., lexical simplification (LexSimpl), deletion (Deletion) and sentence splitting (Split) on simplification, we also conduct

an automated evaluation on each module separately and in groups. The results are shown in Tables 8.2 and 8.3. We also show an example in Figure 8.4 showing outputs from various modules. Complex represents complex sentences from the PWKP evaluation corpus, whereas, Gold (simple) represents simple (reference) sentences from the PWKP evaluation corpus. Others represents outputs from various modules or their combinations. In particular, LexSimpl-Split-Deletion (Unsup) represents our unsupervised simplification framework combining all three modules.

System	BLEU Scores with respect to		Average sentence length	Average token length
	complex	simple		
complex	100	49.85	27.80	4.62
LexSimpl	82.05	44.29	27.80	4.46
Split	89.70	46.15	29.10	4.63
Deletion	85.15	47.33	25.41	4.54
LexSimpl-Split	73.20	41.18	29.15	4.48
LexSimpl-Deletion	69.84	41.91	25.42	4.38
Split-Deletion	77.82	43.44	26.19	4.55
LexSimpl-Split-Deletion (Unsup)	63.41	38.47	26.22	4.40
GOLD (simple)	49.85	100	23.38	4.40

Table 8.3: Automated metrics for simplification: modular evaluation (B)

One first observation is that each module has an impact on simplification. Thus the average Levenshtein Edit distance (LD) to the source clause (complex) is never null for any module while the number of "No edit" indicates that lexical simplification modifies the input sentence in 78%, sentence splitting 49% and deletion 96% of the cases.

In terms of output quality and in particular, similarity with respect to the target clause, deletion is the most effective (smallest LD, best BLEU score w.r.t. target). Further, the results for average token length indicate that lexical simplification is effective in producing shorter words (smaller average length for this module compared to the other two modules).

Predictably, combining modules yields systems that have stronger impact on the source clause (higher LD to complex, lower number of No Edits) whereby the full system (i.e., the system combining the 3 modules) showing the largest LD to the sources (LD to complex) and the smallest number of source sentences without simplification (3 No Edits).

However, it is interesting that the original complex sentences are more similar to the GOLD simple sentences (both in terms of LD to Simple and BLEU score) than

Complex. By 1928, the regional government was moved from the old Cossack capital Novocherkassk to Rostov, which also engulfed the nearby Armenian town of Nor Nakhijevan.
LexSimpl. By 1928, the regional government was moved from the old Cossack capital Novocherkassk to Rostov, which also absorbed the near by Armenian town of Nor Nakhijevan. [1]
Split. By 1928, the regional government was moved from the old Cossack capital Novocherkassk to Rostov. Rostov also engulfed the nearby Armenian town of Nor Nakhijevan. [2]
Deletion. The regional government was moved from the old Cossack capital Novocherkassk to Rostov, which also engulfed the nearby Armenian town of Nor Nakhijevan. [3]
LexSimpl-Split. By 1928, the regional government was moved from the old Cossack capital Novocherkassk to Rostov. Rostov also absorbed the nearby Armenian town of Nor Nakhijevan. [3]
LexSimpl-Deletion. The regional government was moved from the old Cossack capital Novocherkassk to Rostov, which also absorbed the nearby Armenian town of Nor Nakhijevan. [4]
Split-Deletion. The regional government was moved from the old Cossack capital Novocherkassk to Rostov. Rostov engulfed the nearby Armenian town of Nor Nakhijevan. [6]
LexSimpl-Split-Deletion (Unsup). The regional government was moved from the old Cossack capital Novocherkassk to Rostov. Rostov also absorbed the nearby town of Nor Nakhijevan. [7]
Simple. By 1928, the regional government was moved from the old Cossack capital Novocherkassk to Rostov. The nearby Armenian town of Nor Nakhijevan became a part of the city. [10]

Figure 8.4: Example Outputs for modular evaluation with their Levenshtein edit distance with respect to the complex reference sentence.

any of the outputs from (combinations of) modules.

8.3.1.2 Comparison with State-of-the-art Methods

Tables 8.4 and 8.5 shows the results of the automatic evaluation. Zhu, Woodsend, Wubben, Narayan are the best output of the models of Zhu *et al.* (2010), Woodsend and Lapata (2011), Wubben *et al.* (2012) and Narayan and Gardent (2014) (Chapter 7) respectively; Unsup is the model described in this chapter.

The most noticeable result is that our unsupervised system yields results that are comparable with those of the supervised approaches. In terms of BLEU score[35] wrt to the GOLD simplified sentences, Unsup ranks 4th while in terms of edit distance

[35]Moses support tools: multi-bleu `http://www.statmt.org/moses/?n=Moses.SupportTools`.

from the original, and to the simplified sentence, it ranks 3rd.

System	Levenshtein Edit distance			
	Complex to System		System to Simple	
	LD	No edit	LD	No edit
GOLD	12.24	3	0	100
Zhu	7.87	2	14.64	0
Woodsend	8.63	24	16.03	2
Wubben	3.33	6	13.57	2
Narayan	6.32	4	11.53	3
Unsup	6.75	3	14.29	0

Table 8.4: Automatic evaluation results (A).

System	BLEU w.r.t simple	Sentences with splits	Average sentence length	Average token length
GOLD	100	28	27.80	4.40
Zhu	37.4	80	24.21	4.38
Woodsend	42	63	28.10	4.50
Wubben	41.4	1	28.25	4.41
Narayan	53.6	10	26.24	4.36
Unsup	38.47	49	26.22	4.40

Table 8.5: Automatic evaluation results (B).

The results also show that, in contrast to Woodsend system which often leaves the input unsimplified (24% of the input), our system almost always modifies the input sentence (only 3% of the input are not simplified); and that the number of simplifications including a split is relatively high (49% of the cases) suggesting a good ability to split complex sentences into simpler ones.

8.3.2 Human Evaluation for Simplicity, Fluency and Adequacy

To better compare the 5 simplification systems, we conducted a human evaluation geared at comparing the systems w.r.t. simplicity, fluency and adequacy. The evaluation was done online using the LG-Eval toolkit [Kow and Belz, 2012][36] and a Latin Square Experimental Design (LSED) was used to ensure a fair distribution of the systems and the data across raters.

[36] http://www.nltg.brighton.ac.uk/research/lg-eval/

Similar to the experiment in the previous chapter (Table 7.5), the raters were asked to rate input/output pairs w.r.t. to *adequacy* (How much does the simplified sentence(s) preserve the meaning of the input?) and to *simplification* (How much does the generated sentence(s) simplify the complex input?). They were also asked to rate simplified sentences w.r.t. to *fluency* (how grammatical and fluent the sentences are?). We randomly selected 18 complex sentences from the PWKP evaluation corpus and included in the human evaluation corpus: the corresponding simple (Gold) sentence from the PWKP evaluation corpus, the output of our system (Unsup) and the output of the other four systems (Zhu, Woodsend, Narayan and Wubben) which were provided to us by the system authors[37]. We collected ratings from 18 participants. All were either native speakers or proficient in English, having taken part in a Master taught in English or lived in an English speaking country for an extended period of time.

Systems	Simplicity	Fluency	Adequacy
GOLD	3.62	4.69	3.80
Zhu	2.62	2.56	2.47
Woodsend	1.69	3.15	3.15
Wubben	1.52	3.05	3.38
Narayan	2.30	3.03	3.35
Unsup	2.83	3.56	2.83

Table 8.6: Average human ratings for simplicity, fluency and adequacy.

Systems	GOLD	Zhu	Woodsend	Wubben	Narayan
Zhu	◊□△				
Woodsend	◊□▲	◊■▲			
Wubben	◊□▲	◊■△	♦■▲		
Narayan	◊□▲	♦■▲	♦■▲	♦■▲	
Unsup	♦□△	♦□▲	◊■▲	◊■▲	♦■▲

Table 8.7: Pairwise comparisons between all models and their statistical significance.

Table 8.6 shows the average ratings of the human evaluation on a scale from 0 to 5. Pairwise comparisons between all models and their statistical significance were carried out using a one-way ANOVA with post-hoc Tukey HSD tests (Table 8.7). If we group together systems for which there is no significant difference (significance

[37]The casing of words in output sentences varied depending on the systems (e.g., output sentences were lower cased in Wubben while output sentences in Zhu and Woodsend were not). Like in the previous chapter, to make the human evaluation independent of casing all sentences in the evaluation data were lower cased.

level: $p < 0.05$), our system ranks first together with Narayan and Zhu for simplicity; first for fluency; and second for adequacy (together with Woodsend and Zhu). A manual examination of the results indicates that Unsup achieves good simplicity rates through both deletion and sentence splitting. In particular, the average word length of simplified sentences is smaller for W-SUP (26.22) than for Wubben (28.25) and Woodsend (28.10); comparable with Narayan (26.19) and higher only than Zhu (24.21).

In this experiment, number of systems (i.e., 6) was different than that (i.e., 5) of the experiment in the previous chapter (Table 7.5). Hence, to setup the human evaluation experiment with the LG-Eval toolkit, we could not work on the same 20 sentences (Chapter 7, Table 7.5). Instead, we worked on 18 randomly selected sentences. While comparing tables from Chapter 7 (Table 7.5) and this chapter (Table 8.6), we register the same pattern of scores for various systems both in Table 7.5 and Table 8.6. However, we notice a pattern in Table 8.6 of scoring GOLD system very high than that of in Table 7.5, compared to all other systems. As a result, none of the systems get to rank with GOLD (including our supervised system, Table 8.7). This bias could be reduced by conducting the human evaluation on a larger set of sentences.

8.3.3 Analysis of the Split Quality

The examples in Figure 8.5 show some example splits achieved by systems and their human annotation scores. It illustrates errors commonly made by each system, e.g., Zhu often fails to detect the sentence boundary (S1) or to properly reconstruct the new sentence (S3) whereas Woodsend often over splits (S2) or does mistakes with reconstruction (S3). Compared to other systems, Unsup consistently produces better splits.

8.4 Conclusion

A major limitation for supervised simplification systems is the limited amount of available parallel standard/simplified data. Even for English where the parallel EWKP/SWKP corpus is available, learning is limited by the content of the corpus. For instance, Narayan and Gardent (2014) (Chapter 8) note that in that corpus only 6.1% of the simplification involves splitting the input sentence into smaller sentences. However, a quick look at the data clearly indicates that many more simplifications could be improved through sentence splitting. For languages other than English, the lack of parallel data makes supervised approaches redundant.

S1	**Complex.** This array distributes data across multiple disks, but the array is seen by the computer user and operating system as one single disk. **Zhu.** This array sells data across multiple disks but the array is seen. The computer user and operating as one disk. [0] **Woodsend.** This array distributes data across several disks. The array is seen by the computer user and operating system as one single disk. [4.85] **Unsup.** This array moves data across disks. The array is seen by the computer user and operating system as one disk. [4.83]
S2	**Complex.** Here, simulations of evolution using evolutionary algorithms and artificial life started with the work of Nils Aall Barricelli in the 1960s, and was extended by Alex Fraser, who published a series of papers on simulation of artificial selection. **Zhu.** Here, simulations of evolution using evolutionary algorithms and artificial life started with the work of nils aall barricelli in the 1960s, and was extended by alex fraser. Alex fraser published a series of papers on feeling of artificial selection. [3.6] **Woodsend.** It has Genetic algorithms. Evolution is using artificial life. This is Nils Aall Barricelli in the 1960s. Here, simulations started with the work. He was extended by Alex Fraser. He published a group of papers on simulation of artificial selection. [0.33] **Unsup.** Here, models of evolution using evolutionary algorithms and artificial life started with the work in the 1960s, and was led by Alex Fraser. Alex Fraser wrote a series of papers on model of selection. [5]
S3	**Complex.** By 1928, the regional government was moved from the old Cossack capital Novocherkassk to Rostov, which also engulfed the nearby Armenian town of Nor Nakhijevan. **Zhu.** By 1928, the government was moved from the old cossack capital novocherkassk to rostov. Rostov also of the city the nearby armenian town of nor nakhijevan. [2.8] **Woodsend.** By 1928, the regional government was moved from the old Cossack capital Novocherkassk to Rostov. Both also engulfed the nearby Armenian town of Nor Nakhijevan. [3] **Wubben.** by 1928 , the regional government was moved from the old cossack capital novocherkassk to rostov. the nearby armenian town of nor nakhichevan. [2.7] **Narayan.** by 1928, the regional government was moved from the old cossack capital novocherkassk to rostov. rostov that engulfed the nearby armenian town of nor nakhichevan. [2.7] **Unsup.** The regional government was moved from the old Cossack capital Novocherkassk to Rostov. Rostov also absorbed the nearby town of Nor Nakhijevan. [4.75]

Figure 8.5: Example Outputs for sentence splitting with their average human annotation scores.

In this chapter, we have shown that it is possible to take an unsupervised approach to sentence simplification which requires a large corpus of standard and simplified language but no alignment between the two. Although the approach uses a deep semantic analyzer to produce the representations being processed, we believe that a similar approach could be developed using either dependency trees or joint syntactic/semantic role labellers whereby split probabilities would be learned relative to dependency trees/ SRL output instead of DRSs. We plan to explore this issue in the future and to thereby extend the coverage of the approach to all languages for which either a dependency parser or an SRL system exists.

Chapter 9

Conclusions

9.1 Summary and Conclusions

This thesis circles around two main questions: one, how to make symbolic grammar based surface realisation robust and efficient, and two, how to exploit rich linguistic information in the form of deep semantic representation to improve sentence simplification. In what follows, we summarize the main contributions of this thesis.

Improving the Efficiency of Symbolic SR. We present a novel algorithm for surface realisation with a Feature-based Tree-Adjoining grammar. It takes as input shallow structures provided in the format of dependency trees and combines techniques and ideas from the head-driven and lexicalist approaches. The input structure (a tree) is used to filter the initial search space both top-down and bottom up; and to parallelise processes. We evaluated our algorithm on a large scale data provided by the SR shared task and showed that our algorithm drastically reduces generation times compared to a baseline lexicalist approach which explores the whole search space.

Improving the Coverage of Symbolic SR. Symbolic surface realisers are very prone to errors in grammars and lexicon and to mismatches in the input structure and the structure expected by the grammar. We propose two novel error mining algorithms to identify these fallacies and consequently improve the robustness of our symbolic surface realiser. Our first error mining algorithm allows us to mine for suspicious trees efficiently. It improves over previous error mining algorithms which are limited for mining sequential nature of suspicious forms such as n-grams of words or parts of speech tags. Our second algorithm structures the output of error mining into a suspicion tree that supports a linguistically meaningful error analysis. Previous error mining algorithms are used to produce a flat list of suspicious forms.

In addition, both of our approaches are generic in that they permit mining trees and strings for suspicious forms of arbitrary size and arbitrary conjunctions of labelling.

We apply these error mining techniques on the SR data to identify the causes of undergeneration by our symbolic surface realiser. We showed that our algorithms help us to easily identify gaps and errors in the grammar and in the lexicon; and mismatches between the input data format and the format expected by our realiser. With the help of these error mining algorithms, we improve the coverage of our realiser by a wide margin. On the SR data, we achieved a coverage of 81% with a BLEU score of 0.72 on generated sentences.

Generating Elliptic Coordination. With our efficient surface realiser coupled with our capable error mining algorithms, we were able to narrow down our focus towards generating more complex linguistic phenomena such as elliptic coordination. We showed that, in the SR data, elliptic structures are frequent and can impact the performance of a surface realiser. We argued about the representation of ellipsis in sentences and provided a set of tree rewrite rules to introduce phonetically empty nodes in the dependency trees with ellipsis. We showed that the introduction of empty nodes for elliptical sentences improves both coverage and accuracy of our surface realiser. On the dataset of 2398 elliptical sentences, our surface realiser achieved a coverage of 76% and a BLEU score of 0.74 on generated sentences.

Resource: Elliptic Coordination Corpus. To evaluate our surface realiser on the ability to generate elliptic coordination, we collected 2398 potential input representations from the SR data. It contains 384 potential cases of right-node raising (RNR), 1462 potential cases of subject sharing (SS), 456 potential cases of SS+RNR, 36 cases to gapping and 60 cases of non-constituent coordination (NCC). We believe that this dataset could be very useful for other generators to test their abilities to generate ellipsis.

Deep Semantics for Sentence Simplification. While simplifying sentences using hand-written rules, Siddharthan (2010) have found that phrasal parse trees were inadequate for learning complex lexico-syntactic transformation rules and that dependency structures were better suited to the task. In comparison, all existing machine-learning approaches [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Coster and Kauchak, 2011; Wubben *et al.*, 2012] to sentence simplification still either starts from the input sentence or its phrasal parse tree.

We went one step further than Siddharthan (2010) and proposed using rich linguistic information in the form of deep semantic representations to improve the sentence simplification task. We use the DRS representation [Kamp, 1981] assigned by Boxer [Curran *et al.*, 2007] for the deep semantic representations and we proposed

two novel algorithms for sentence simplification: supervised and unsupervised. Both algorithms use the deep semantic representation to learn how to split a complex sentence into multiple simpler sentences and how to delete unnecessary modifiers in a sentence without significantly altering its meaning. We show that models learned on the deep semantic representation facilitate completion (the re-creation of the shared element in the split sentences) and provide a natural means to avoid deleting obligatory arguments.

Hybrid Supervised Approach to Simplification. We proposed a hybrid approach to sentence simplification which combines DRS simplification model for splitting and deletion with a monolingual machine translation system for substitution and reordering. We trained our system on the PWKP corpus and evaluated on the PWKP evaluation corpus. We compared our system against three state-of-the-art methods [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Wubben *et al.*, 2012] and we showed that our approach produces significantly simpler output that is both grammatical and meaning preserving.

Comparable Corpora for Sentence Simplification. In addition to our hybrid approach, we proposed another novel approach to sentence simplification which does not need a sentence-aligned corpus of complex and simple sentences. It has been shown in [Woodsend and Lapata, 2011; Narayan and Gardent, 2014] that supervised approaches are easily effected by the quality of their training corpora. Our proposed approach is unsupervised in that it requires only a large corpus of standard and simplified language but no alignment between the two. Lexical simplification probabilities are learned by analysing words with their context in both traditional English Wikipedia and Simple English Wikipedia, while splitting and deletion probabilities are learned by analysing frequent structures in the DRS representation of Simple English Wikipedia only. We evaluated our system on the PWKP evaluation corpus and found that our method is competitive with four state-of-the-art supervised systems [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Wubben *et al.*, 2012; Narayan and Gardent, 2014].

9.2 Pointers for Future Research

In this section, we discuss potential directions for future research.

Generating from Graph Structure of Semantic Representations. The surface realisation algorithm described in Chapter 3 clearly takes advantage of the input structure (a dependency tree) to optimize performance by filtering the initial search space both top-down and bottom-up. The tree structure also helps in parallelising

generation processes. We have shown that the proposed algorithm is very efficient in generating longer sentences.

One interesting direction would be to explore how our algorithm could benefit the generation process from more complex input representations such as flat semantics [Copestake *et al.*, 2001] and deep dependency graphs provided by the SR shared task [Belz *et al.*, 2011]. Note that, surface realization from flat semantics has been shown to be an NP-Complete problem [Koller and Striegnitz, 2002]. It would be interesting to investigate if our proposed approach could provide an efficient solution to the problem.

Because of the graph structures of flat semantics or deep dependency graphs, the direct adaptation of our algorithm to these complex representation is not straightforward. Instead, we think that the problem of generation from these representations could be divided into subproblems of generating from tree structures using our algorithm and then combining the intermediate results using a chart generator. It is in line with White (2004) who has proposed to chunk input logical forms into subproblems to be solved prior to further combination, to avoid a proliferation of semantically incomplete edges. However we propose to extract all subtrees from the input graph representation and to apply our realiser (instead of a normal chart-based realiser) to generate from each of them separately. This way, we take advantages of local polarity filtering and parallelisation while generating from these subtrees. At the end, resulting derivations for the subtrees could be combined using an efficient chart generator [Gardent and Perez-Beltrachini, 2010].

Error Mining for Overgeneration Suspects. We have evaluated our error mining techniques on their abilities to identify undergeneration suspects of our generation system. One interesting research direction would be to explore how these techniques could be used to identify overgeneration suspects of our grammar.

Our final BLEU score (0.73, Table 5.2 in Chapter 5) on successfully generated sentences indicates that not all of them are accurate. This could be an indication of overgeneration by our grammar. To identify suspicious TAG elementary tree which cause overgeneration, we could error mine the output derivation trees[38] of our surface realiser on the SR Task data. However, to support error mining, we need to classify each output derivation tree as a success (SUCCESS) or a failure (FAIL). Using the reference sentence (takes from the Penn Treebank) associated with each SR Task dependency tree, one could use various evaluation metrics (e.g., Levenshtein edit distance, BLEU, NIST, METEOR and TER) with a threshold to classify the

[38]Note that the nodes in the TAG derivation trees are decorated with the elementary trees used to produce derivations.

produced derivation tree as SUCCESS or FAIL. The resulting datasets (SUCCESS and FAIL) could be later error mined for overgeneration suspects in our grammar.

Another open question to investigate is that how many output derivation trees for each dependency tree should be kept for error mining. Since the surface realiser is non-deterministic, it generally generates more than one derivation trees for a given input dependency tree.

Use Elliptic Coordination Corpus. We have released the 2398 input representations we collected for the evaluation of our realiser on its generation ability of elliptic coordination. After rewriting this dataset for introducing empty categories for elliptic arguments, we have shown that our generator improved on both accuracy and coverage. However, because of the lack of such evaluations, we could not compare our approach with another approaches of representing and generating ellipses.

Hence, one interesting direction would be to investigate how our approach performs compared to the performances of White (2006)'s CCG based generator and Carroll and Oepen (2005)'s HPSG based generator. White (2006) avoids empty categories by adopting a more flexible notion of constituency, whereas the domain of locality in the HPSG grammar (used in [Carroll and Oepen, 2005]) differs from that of TAG.

Evaluating Sentence Simplification for Discourse Level Simplifications. One of our main contribution to sentence simplification is that we argued to use deep semantic representation in the form of Discourse Representation Structures to improve sentence simplification task. Our simplification framework uses DRS simplification model for splitting and deletion probabilities. In fact, we have shown the advantages of discourse level simplification on the PWKP evaluation corpus (100 sentences) by comparing our both supervised and unsupervised system against other state-of-the-art simplification methods. We have shown that DRS-based simplification eases the reconstruction of the shared element in the split sentences and eschews deleting obligatory arguments.

In general however, discourse level simplification operations such as sentence splitting, sentence reordering, cue word selection, referring expression generation and determiner choice are semantically constrained. In that regard, our evaluation on the PWKP evaluation corpus is very limited. In the PWKP data, the proportion of split sentences is rather low (only 6.1%) and many of the split sentences are simple sentence coordination splits. Our supervised system trained on the PWKP corpus fails to split adequately (only 10%) on the PWKP evaluation corpus. Our unsupervised system trained on SWKP performs better and splits 49% on the PWKP evaluation corpus but again splits are limited to coordination splits.

[Siddharthan, 2006] has created a corpus of 95 cases of discourse simplification. This corpus could be a good starting point for a proper evaluation of our discourse level simplification. This evaluation will be limited to our unsupervised approach, however. Our supervised approach needs a parallel corpus to train upon and training on the PWKP corpus does not help learning various types of discourse level simplification.

Using data from the language learning or the children reading community, it would be interesting to first construct a similar, larger scale corpus; and to then train and test our supervised approach on more complex cases of sentence splitting.

Exploiting Paraphrase Database for Simplification. As we have discussed before, using DRS-based simplification our supervised and unsupervised approaches perform very good in semantically constrained splits and deletion. However they fail to capture active-passive transformations and complex multi-word expression simplification.

Handcrafted rule based simplification systems [Chandrasekar and Srinivas, 1997; Siddharthan, 2002; Canning, 2002; Siddharthan, 2006; Siddharthan, 2010; Siddharthan, 2011; Bott *et al.*, 2012] are shown to be best in capturing active-passive transformations. For machine learning approaches [Zhu *et al.*, 2010; Woodsend and Lapata, 2011; Coster and Kauchak, 2011; Wubben *et al.*, 2012], either they fail to capture such transformations because of the poor quality of the training data or they are not rigged for capturing such transformations.

Similarly, the sentence simplification literature either overlooks multi-word expressions or does not consider them at all. The supervised systems claims to achieve multi-word simplification with phrase substitution but their strength is limited with the poor quality of the training data.

Recently, the Paraphrase Database (PPDB, [Ganitkevitch *et al.*, 2013]) has been released consisting of 169 millions lexical, phrasal and syntactic rules. Because of its broad coverage, it would be interesting to investigate PPDB for syntactic rules which corresponds to active-passive transformations and for lexical and phrasal rules which corresponds to multi-word expression simplification. In fact, it would be interesting to build a sentence simplification system which uses DRS-based simplification for splitting and deletion, and the PPDB corpus for lexical and phrasal substitutions (including multi-word expression simplification) and reordering (including active-passive transformations).

The PPDB copus, however, does not only have rules for simplification. Hence, the extraction of simplification rules from the PPDB corpus is an open question. One trend could be to use filters using the complexities of both terms (left and right

terms) of a rules estimated using Simple English Wikipedia and traditional English Wikipedia [Biran *et al.*, 2011]. Another trend could be to tune the rephrasing rules from the PPDB corpus for simplification [Ganitkevitch *et al.*, 2011].

It would be interesting to compare the results from this research with the results from handwritten syntactic simplification rule based systems [Siddharthan, 2010; Siddharthan, 2011; Siddharthan and Mandya, 2014]. These systems claim to achieve active-passive transformation using handcrafted rules based on typed-dependency structures.

Beyond simplification. One interesting direction of research would be adapting DRS-based simplification to another domain such as sentence compression, text simplification or summarisation, and machine translation.

Sentence compression. Filippova and Strube (2008) have used dependency tree representation to shorten sentences by removing subtrees. Their approach tries to preserve dependencies which are either required for the output to be grammatical or have an important word as the dependent. Among the many steps of dependency tree transformation, one of the step marks every inflected verb in the tree by adding a dependency originating from the root node. In a way, this step tries to identify events in the sentence before dropping our subtrees. In contrast, the DRS representation generated by Boxer [Curran *et al.*, 2007] automatically captures all event nodes. Also, the obligatory dependencies such as agent and patient are marked in the DRS representation. To conclude, it would be interesting to compare DRS based compression with the Filippova and Strube (2008)'s approach.

Machine Translation. Chandrasekar *et al.* (1996) have argued that the long and complicated sentences prove to be a stumbling block for current systems relying on natural language input. They have shown how sentence simplification facilitates statistical machine translation (SMT).

Syntax-based models [Charniak, 2001; Yamada and Knight, 2001; Charniak *et al.*, 2003; Eisner, 2003] are argued to be best for statistical machine translations (SMT). However, recent trends [Aue *et al.*, 2004; Banchs and Costa-jussà, 2011; Jones *et al.*, 2012; Li *et al.*, 2014] have tried to fuse semantic features to improve SMT.

It would be very interesting to combine these two aspects: sentence simplification and semantics, using our discourse level simplification model to machine translation[39]. Our sentence simplification exploits deep semantics and produces simple, grammatical and meaning preserving sentences. It would be interesting to see how

[39]This idea is a result of a personal communication with Alexandra Birch University of Edinburgh in May 2014.

SMT systems perform on these simple sentences.

Text Simplification and Summarization. The Boxer toolkit [Curran *et al.*, 2007] could be used to generate discourse representation structures of a text instead of a single sentence. In fact, it has been used in the construction of the Groningen Meaning Bank (GMB, [Basile *et al.*, 2012]), a large scale semantically annotated corpus of texts.

It would be quite interesting to investigate how Boxer represents deep semantic representation of a text, does it allow anaphora resolution, and how we could exploit our DRS simplification model to improve text simplification and summarisation.

Appendices

Appendix A

Multiple Adjunction in Feature-based Tree Adjoining Grammar

This chapter is based on the following paper:

> Claire Gardent and Shashi Narayan. *Multiple Adjunction in Feature-based Tree Adjoining Grammar*, Accepted for publication in Computational Linguistics, 2014.

In parsing with Tree Adjoining Grammar (TAG), independent derivations have been shown by Schabes and Shieber (1994) to be essential for correctly supporting syntactic analysis, semantic interpretation and statistical language modelling. However, the parsing algorithm they propose is not directly applicable to Feature-Based TAGs (FB-TAG). We provide a recognition algorithm for FB-TAG which supports both dependent and independent derivations. The resulting algorithm combines the benefits of independent derivations with those of Feature-Based grammars. In particular, we show that it accounts for a range of interactions between dependent vs. independent derivation on the one hand, and syntactic constraints, linear ordering, and scopal vs. nonscopal semantic dependencies on the other hand.

A.1 Introduction

A Tree Adjoining Grammar (TAG, [Joshi and Schabes, 1997]) consists of a set of elementary trees and two combining operations, substitution and adjunction. Consequently, a TAG derivation can be described by a tree (called a *derivation tree*)

specifying which elementary TAG trees were combined using which operations to yield that derivation. In this tree, each vertex is labelled with a tree name and each edge with a description of the operation (node address and operation type) used to combine the trees labelling its end vertices. As we shall see in Section A.3.2, in TAG, each derivation tree specifies a unique parse tree also called *derived tree*.

In previous work, it has been argued that TAG derivation trees provide a good approximation of semantic dependencies between the words of a sentence [Kroch, 1989; Rambow *et al.*, 1995; Candito and Kahane, 1998; Kallmeyer and Kuhlmann, 2012]. As shown by Schabes and Shieber (1994) however, there are several possible ways of defining TAG derivation trees depending on how multiple adjunction node is handled. The *standard notion of derivation* proposed by Vijay-Shanker (1987) forbids multiple adjunction thus enforcing *dependent* derivations. In contrast, the *extended notion of derivation* proposed by Schabes and Shieber (1992) and Schabes and Shieber (1994) allows multiple adjunction at a single node thereby yielding so-called *independent* derivations i.e., derivations where the relation between the adjoining trees is left unspecified. The difference between the two types of derivations is illustrated in Figure A.1. Figure A.1 presents an example TAG with the alternative TAG derivations for the phrase *roasted red pepper*. α_{pepper}, β_{red} and $\beta_{roasted}$ are the elementary trees for *pepper* (initial tree), *red* (auxiliary tree) and *roasted* (auxiliary tree) respectively. While in the standard (dependent) derivation, one adjective tree is adjoined to the other adjective tree which itself is adjoined to the noun tree for *pepper*, in the extended (independent) derivation, both adjective trees adjoin to the noun tree.

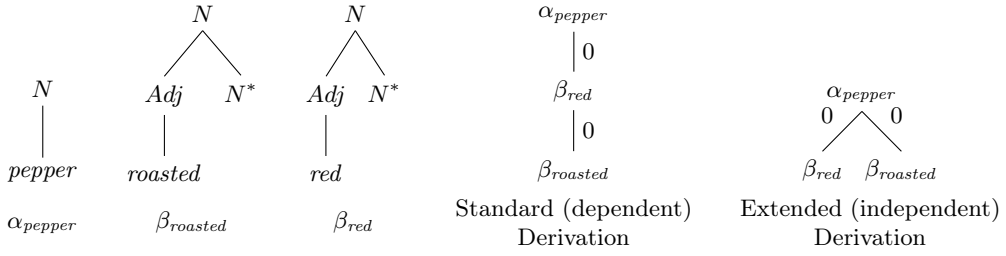


Figure A.1: An example TAG with the alternative TAG derivations for the phrase *roasted red pepper*.

Schabes and Shieber (1994) argue that allowing both for dependent and independent derivations better reflects linguistic dependencies. Making use of the distinction introduced in TAG between predicative and modifier auxiliary trees (Schabes and Shieber (1994), Section A.3.1), they define a parsing algorithm which assigns dependent derivations to predicative auxiliary trees but independent derivations to

multiple modifier auxiliary trees adjoining to the same node. In case both predicative and modifier auxiliary trees adjoin to the same node, their parsing algorithm ensures that predicative trees appear above the modifier trees in the derived tree.

This parsing algorithm is defined for featureless variants of TAG. In contrast, in implemented TAGs (e.g., XTAG [The XTAG Research Group, 2001], SemXTAG [Gardent, 2008] or XXTAG[40] [Alahverdzhieva, 2008]) feature structures and feature unification are central. They are used to minimize the size of the grammar; to model linguistic phenomena such as verb/subject agreement; and to encode a unification-based syntax/semantics interface (cf., e.g., [Gardent and Kallmeyer, 2003]).

In this chapter, we extend Schabes and Shieber's proposal to Feature-Based TAG (FB-TAG) and we show that the resulting parsing algorithm naturally accounts for the interplay of dependent vs. independent derivation structures with syntactic constraints, linear ordering, and scopal vs. nonscopal semantic dependencies.

The chapter is organized as follows. In Section A.2, we recap the motivations for independent derivations put forward by Schabes and Shieber (1994) and we briefly discuss the interactions that may arise between dependent and independent derivations. Section A.3 summarises their approach. In Section A.4, we present the intuitions and motivations underlying our proposal and we highlight the differences with Schabes and Shieber's approach. Section A.5 presents our proposal. Section A.7 concludes.

A.2 Why are Independent Derivations Desirable?

We start by summarizing Schabes and Shieber's motivations for independent derivations. We then discuss the interactions between dependent and independent derivations.

A.2.1 Motivations for Independent Derivations

Schabes and Shieber (1994) give three main motivations for independent derivations. The first motivation concerns the interaction of verbs with multiple modifiers. Consider sentences[41] in (20) and (21).

(20) a. Richard Parker and Pi wandered the Algae Island yesterday through the meerkats.

b. Richard Parker and Pi wandered the Algae Island yesterday.

c. Richard Parker and Pi wandered the Algae Island through the meerkats.

[40] XXTAG stands for XMG [Crabbé *et al.*, 2013] based XTAG.

[41] The characters in these sentences are borrowed from Yann Martel's book *Life of Pi*.

(21) a. ⊗ The Orangutan reminded Pi of his mother yesterday through the meerkats.
b. The Orangutan reminded Pi of his mother yesterday.
c. ⊗ The Orangutan reminded Pi of his mother through the meerkats.

Movement verbs such as *to wander* allow for directional modifiers such as *through the meerkats* whereas verbs such as *to remind* do not. In TAG, such restrictions can be modeled using selective adjoining constraints to specify which modifier tree may or may not be adjoined at a particular node in a given tree. Therefore it is possible to license (20) and to rule out (21c). In (21a) however, under the dependent notion of adjunction, the tree for the directional adverbial *through the meerkats* will adjoin to the modifier tree for *yesterday* which itself will adjoin to the tree selected by *reminded*. Thus constraints placed by the verb on its modifiers must be passed through by modifier trees (here the tree for *yesterday*) to also rule out sentences such as (21a). Propagating selective adjunction constraints in TAG would lead to a formalism for which derivation trees are no longer context-free [Schabes and Shieber, 1994].

The second motivation for independent adjunction stems from probabilistic approaches. Stochastic lexicalized TAG specifies the probability of an adjunction of a given auxiliary tree at a given node in another elementary tree [Schabes, 1992; Resnik, 1992]. Thus under the standard notion of derivation, the overall probability of the string *roasted red pepper* would be determined by the probability of *red* adjoining to *pepper* and the probability of *roasted* adjoining to *red*. In contrast, independent adjunction would result in a derivation such that the overall probability of the string *roasted red pepper* would be determined by the probability of both *red* and *roasted* adjoining to *pepper*. Schabes and Shieber (1994) argue that it is plausible that "the most important relationships to characterize statistically are those between modifier and modified, rather than between two modifiers".

A third motivation comes from semantics and more particularly, from scope ambiguities involving modifiers. Given a sentence such as (22) where the relative scope of the modifiers *twice* and *intentionally* is ambiguous[42], Shieber (1994) shows that, under the extended definition of adjunction, a synchronous TAG modelling the relation between syntactic trees and logical formulae can account for both readings.

(22) John blinked twice intentionally.

The account crucially relies on multiple independent adjunction of the two modifier trees to the tree for *blink*: depending on which order the auxiliary trees for *twice*

[42]The sentence can describe either *a single intentional act of blinking twice* or *two intentional acts each of single blinking* [Shieber, 1994].

and *intentionally* adjoins to *blink*, the logical formula built will be either *intentionally(twice(blink))* or *twice(intentionally(blink))* thus capturing the ambiguity.

Another motivation for multiple adjunction accrues from the task of generation and more specifically surface realisation. The underspecified input to the surface realization task does not determine the ordering of intersective modifiers. For example, the flat semantics representation such as *Minimal Recursion Semantics* [Copestake *et al.*, 2001] for "*roasted red pepper*" is as follows:

$$(roasted(x), red(x), pepper(x))$$

Recently, the Generation Challenge has promoted a Surface Realisation (SR) task [Belz *et al.*, 2011] where the input provided are unordered dependency structures. The corresponding unordered dependency structure for "*roasted red pepper*" is shown in Figure A.2.

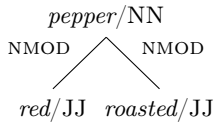


Figure A.2: Dependency structure for "roasted red pepper".

Therefore in practice, the input to generation models the semantic relationships between modifiers and modified in its underspecified representation. Under Vijay-Shanker's standard derivation, the derivation tree fails to capture these relationships among intersective modifiers and makes the task of generation computationally expensive [Bangalore and Rambow, 2000b]. Schabes and Shieber's extended derivation with multiple adjunction formulates the appropriate relationships straightforwardly [Shieber, 1994].

A.2.2 Dependent, Independent and Mixed Derivations

To capture the different types of semantic dependencies and morpho-syntactic constraints which may hold between multiple auxiliary trees adjoining to the same entity, both dependent and independent derivations are needed.

As argued above, because there are no constraints or semantic relation holding between each of them, multiple intersective modifiers applying to the same entity (e.g., 23) are best modeled using an independent derivation.

(23) a. The tall black meerkat slept. (Independent derivation)

In contrast, because they may involve strong scopal and morpho-syntactic constraints, stacked predicative verbs (i.e., verbs taking a sentential complement, 24a) and non-intersective modifiers (e.g., 24c) require dependent derivations. Consider sentences (24a-b) for instance. If predicative trees were assigned an independent derivation, sentence (24a) would be judged ungrammatical (because *want* requires an infinitival complement but would adjoin to the finite verb *slept*) and conversely, sentence (24b) would incorrectly be judged grammatical (because both *want* and *try* require an infinitival complement). Similarly, in example (24c), the church is Syrian Orthodox, not Syrian and Orthodox. Assigning a dependent rather than an independent derivation to such cases straightforwardly capture the distinction between intersective and non intersective modifiers.

(24) a. ✓John wanted to assume that Peter slept. (Dependent derivation)

b. ⊗John wanted Peter tries to walk.

c. The meerkat admired the Syrian Orthodox church. (Dependent derivation)

Finally, some multiple adjunctions may involve both dependent and independent derivations, e.g., when multiple modifiers and predicative verbs adjoin to the same verb (e.g., 25a) or in the case of a derivation (e.g., 25b) involving both intersective (old) and non-intersective (i.e., Syrian in Syrian Orthodox) modifiers.

(25) a. Yann said that John knows that Richard Parker and Pi wandered the Algae Island yesterday through the meerkats. (Mixed derivation)

b. The meerkat admired the old Syrian Orthodox church. (Mixed derivation)

As we shall see in Section A.5.3, the parsing algorithm we propose licenses dependent, independent and mixed derivations but is restricted to appropriately distinguish between various types of modifiers. Moreover, the feature information encoded in the grammar further restricts the derivation structures produced thereby accounting for the interactions between adjunction, linear ordering and morpho-syntactic constraints.

A.3 Multiple Adjunction in Tree Adjoining Grammars

Vijay-Shanker and Weir (1991) introduce a compilation of TAG to Linear Indexed Grammars (LIG, [Gazdar, 1988]) which makes the derivation process explicit. Schabes and Shieber (1994) modify this compilation to allow both for dependent and for

independent derivations. The resulting LIG is further exploited to specify a parsing algorithm which recovers those derivations.

In this section, we summarize Schabes and Shieber's proposal. We start (Section A.3.1) with an informal description of their approach. In Section A.3.2, we introduce ordered derivation trees. Section A.3.3 gives a brief introduction to LIG. Section A.3.4 summarizes the TAG-to-LIG compilation proposed by Vijay-Shanker and Weir (1991). Finally, Section A.3.5 describes the modifications introduced by Schabes and Shieber (1994) to allow both for dependent and for independent derivations.

A.3.1 Schabes and Shieber's Proposal: Motivations and Intuitions

Tree Adjoining Grammar distinguishes between two types of auxiliary trees namely, modifier vs. predicative auxiliary trees [Joshi and Vijay-Shanker, 2001]. While predicative trees are assigned to verbs taking a sentential argument, modifier trees are assigned to all other auxiliary trees, e.g., verbal auxiliaries, adjectives, adverbs, prepositions and determiners. More generally, the difference between a predicative and a modifier tree is that in a predicative tree, the foot node, like the substitution nodes, corresponds to an argument node selected by its lexical anchor (i.e., the word that selects that tree) while in a modifier auxiliary tree, the foot node is an open slot corresponding to the phrase being modified. When associating semantic entities with tree nodes (as proposed, e.g., by Joshi and Vijay-Shanker (2001) and Gardent and Kallmeyer (2003)), this difference can be seen by noting the entities associated with root and foot nodes: these are distinct in a predicative tree but identical in modifier trees.

In their approach, Schabes and Shieber specify a TAG to LIG conversion which systematically associates dependent derivations with predicative auxiliary trees and independent derivations with modifier auxiliary trees. In addition, they introduce two mechanisms to ensure that each derivation tree unambiguously specifies a linguistically plausible derived tree.

First, they enforce ordering constraints between modifier trees adjoining at the same node (which are thus ambiguous with respect to the derived tree they describe) by assuming that derivation trees are ordered and that linear precedence (LP) statements can be used to constrain the order of siblings in a derivation tree. For instance, given the independent derivation shown in Figure A.1, an LP statement stating that β_{red} must occur before $\beta_{roasted}$ in the derivation tree will ensure that $\beta_{roasted}$ appears above β_{red} in the derived tree and therefore that the resulting derived tree yields the phrase *roasted red pepper* rather than *red roasted pepper*.

Second, when both predicative and modifier trees adjoin at the same address, predicative trees always occur above all modifier trees in the derived tree ("outermost predication"). This ensures for instance, that under the reading where *yesterday* refers to the *arriving* rather than the *saying* i.e., when both *say* and *yesterday* adjoin to *arrive*, (26a) is derived but not (26b).

(26) a. ✓Peter says that yesterday John arrived late.

b. ⊗Yesterday Peter says that John arrived late.

A.3.2 Ordered Derivation Trees

In the standard version of TAG, each derivation tree describes a unique derived tree. In the case of a dependent derivation, unicity follows from the fact that dependent derivations specify the order in which adjunction takes place (e.g., β_2 adjoins to β_1 and the result to α). As a result, if β_2 adjoins to β_1, there is only one possible derived tree namely, a tree where β_2 appears above β_1.

When allowing for independent derivations however, several derived trees are possible depending on the order in which the auxiliary trees are adjoined. To ensure a unique mapping from derivation to derived tree, Schabes and Shieber (1994) therefore introduce the notion of ordered derivation trees. Ordered derivation trees differ from standard TAG derivation trees in that (i) they may contain sibling edges labelled with the same address and (ii) they specify a total order on such siblings.

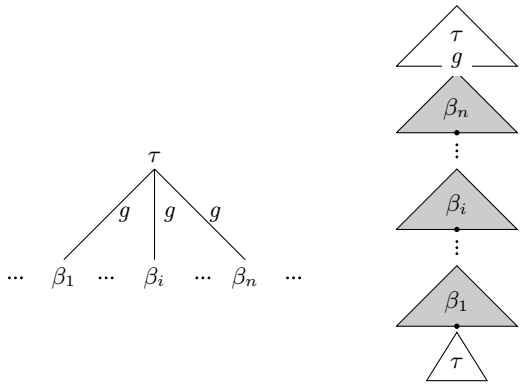


Figure A.3: Ordered derivation tree and corresponding derived tree.

Figure A.3 shows an example ordered derivation tree and associated derived tree. As indicated by the shared g address on their parent edge, auxiliary trees $\beta_1, \ldots, \beta_n$ adjoin to the same node namely the node with address g in the elementary tree τ. Because the derivation tree is ordered, β_1 will appear below β_2 in the derived tree

which in turn will be below β_3, and so on. In short, given a set of auxiliary trees all adjoining to the same tree node, the derived tree produced from an ordered derivation tree following an independent derivation will be identical to the derived tree produced with the corresponding dependent derivation i.e., the dependent derivation where β_1, ..., β_n appear in increasing index order from top to bottom.

A.3.3 Linear Indexed Grammar

Like Context-Free Grammars (CFG), Linear Indexed Grammars (LIG, [Gazdar, 1988]) are string rewriting systems where strings are composed of terminals and nonterminals. In a LIG however, nonterminal symbols may be associated with a stack of symbols, called indices. A LIG rule can thus be represented as follows:

$$N[..\mu] \rightarrow N_1[\mu_1] \ldots N_{i-1}[\mu_{i-1}] N_i[..\mu_i] N_{i+1}[\mu_{i+1}] \ldots N_n[\mu_n] \tag{A.1}$$

N and N_i are nonterminals while μ and μ_i are strings of stack symbols. The symbol .. stands for the remainder of the stack symbols. Note that the remainder of the stack symbols associated with the LHS is associated with only one of the nonterminal (namely, N_i) on the RHS.

Linear Indexed Grammars (LIG) have been used in the literature [Weir and Joshi, 1988; Vijay-Shanker and Weir, 1991] to provide a common framework for the extensions of context-free grammars. In particular, Vijay-Shanker and Weir (1991) and Vijay-Shanker and Weir (1993) showed a weak equivalence between LIGs, TAGs and CCGs (Combinatory Categorial Grammars, [Steedman, 2000]) and proposed a LIG based polynomial-time CYK recognition algorithm for TAGs and CCGs. In what follows, we show how Schabes and Shieber (1994) use a LIG variant of TAGs to license both dependent and independent derivations.

A.3.4 TAG to LIG Compilation

The TAG to LIG compilation proposed by Vijay-Shanker and Weir (1991) produces LIG rules which simulate a traversal of the derived tree produced by the original TAG grammar. Figure A.4 illustrates the traversal of the TAG derived trees specified by the LIG resulting fromVijay-Shanker and Weir (1991) TAG to LIG compilation. Each of the tree nodes in the grammar is assigned a unique address. For example, here η, η_1, η_i and η_n point to the distinct nodes in the left elementary tree whereas η_r and η_f point to the root and the foot nodes of the shown auxiliary tree β in the grammar. In these LIG rules, each node η of a TAG elementary tree is viewed as

having both a top $t[..\eta]$ and a bottom $b[..\eta]$ component to account for the possibility of an adjunction.

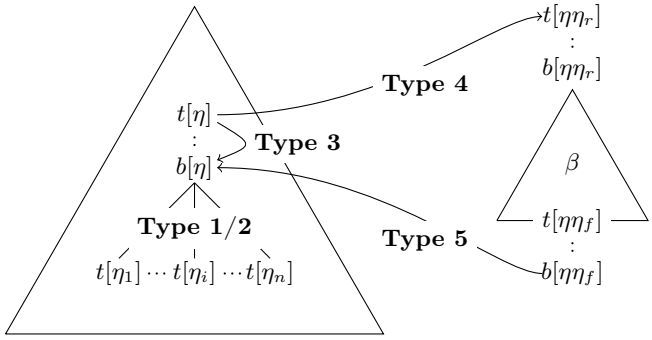


Figure A.4: LIG variant of TAG for the standard derivation.

Type 1: *Immediate domination dominating foot.* For each node η in the auxiliary trees which dominates the foot node and with children $\eta_1, \ldots, \eta_i, \ldots, \eta_n$ where the child η_i also dominates the foot node, the following LIG production rule is generated.

$$b[..\eta] \rightarrow t[\eta_1] \ldots t[\eta_{i-1}] t[..\eta_i] t[\eta_{i+1}] \ldots t[\eta_n]$$

Type 2: *Immediate domination not dominating foot.* For each elementary tree node η which does not dominate the foot node and with children $\eta_1, \ldots, \eta_n$, the following LIG production rule is generated.

$$b[\eta] \rightarrow t[\eta_1] \ldots t[\eta_n]$$

Type 3: *No adjunction.* For each elementary tree node η that is not marked for substitution or obligatory adjunction, the following LIG production rule is generated.

$$t[..\eta] \rightarrow b[..\eta]$$

Type 4: *Start root of adjunction.* For each elementary tree node η which allows the adjunction of the auxiliary tree with the root node η_r, the following LIG production rule is generated.

$$t[..\eta] \rightarrow t[..\eta\eta_r]$$

Type 5: *Start foot of adjunction.* For each elementary tree node η which allows the adjunction of the auxiliary tree with the foot node η_f, the following LIG production rule is generated.

$$b[..\eta\eta_f] \rightarrow b[..\eta]$$

Type 6: *Start substitution.* For each elementary tree node η which allows the substitution of the initial tree with the root node η_r, the following LIG production rule is generated (not shown in Figure A.4).

$$t[\eta] \rightarrow t[\eta_r]$$

Figure A.5: LIG production rules for the standard derivation.

Figure A.5 lists the LIG rules resulting from the TAG to LIG compilation process. Each nonterminal ($t[..\eta]$ or $b[..\eta]$) with the top of the stack symbol in a LIG rule

corresponds to a unique node in some elementary tree of the grammar. The inner stack symbols are used to keep track of the nodes higher in the derived tree where an auxiliary tree has been adjoined.

Rules of Type 1 and 2 capture immediate dominance between the bottom of a node η and the top of its immediate daughters in two configurations depending on whether η dominates the foot node (Type 1) or not (Type 2). Rules of Type 3 handle nodes which require neither substitution nor adjunction. This rule handles cases where no adjunction occurs at a node by rewriting the top of this node to its bottom. Rules of Type 6 model substitution. Finally, rules of Type 4 and 5 handle adjunction. They specify that, for any given node η and any auxiliary tree β which may adjoin to η, the top of η rewrites to the top of the root node of β; and the bottom of the foot of β to the bottom of η. It follows that there can be no multiple adjunction in this LIG version of TAG.

A.3.5 Modifying the TAG to LIG Compilation to Allow for Multiple Adjunctions

To associate predicative tree adjunctions with dependent and multiple modifier adjunctions with independent derivations, Schabes and Shieber (1994) modify the compilation of TAG to LIG proposed by Vijay-Shanker and Weir (1991) as sketched in Figure A.6 (top and bottom components of the nodes are presented by •). Type 4(a) rules apply to adjunctions involving predicative trees. They are identical to Type 4 rules in the Vijay-Shanker and Weir's approach and therefore enforce a standard (dependent) derivation for predicative trees. In contrast, Type 4(b) rules apply to adjunctions involving modifiers and result in an independent derivation.

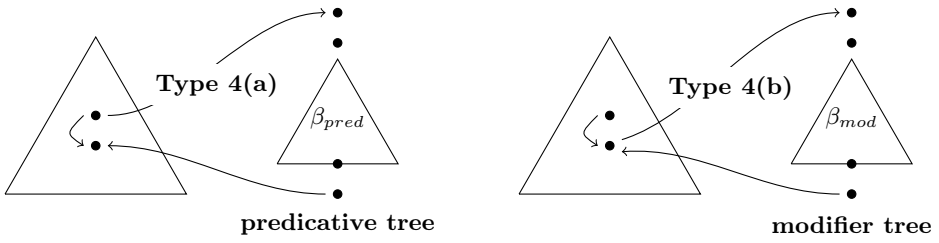


Figure A.6: LIG variant of TAG for Schabes and Shieber's extended derivation.

Type 4(a): *Start root of adjunction for predicative trees.* For each elementary tree node η which allows the adjunction of the predicative auxiliary tree with the root node η_r, the following LIG production rule is generated.

$$t[..\eta] \rightarrow t[..\eta\eta_r] \tag{A.2}$$

Type 4(b): *Start root of adjunction for modifier trees.* For each elementary tree node η which allows the adjunction of the modifier auxiliary tree with the root node η_r, the following LIG production rule is generated.

$$b[..\eta] \rightarrow t[..\eta\eta_r] \tag{A.3}$$

Note also that the "outermost predication" constraint i.e., predicative trees always occur above modifier trees adjoined at the same node, alluded to in Section A.3.1 follows from the interactions between the Type 4(a) and Type 4(b) LIG rules.

Schabes and Shieber prove the weak-generative equivalence of TAGs under both standard and extended derivation using the LIG compilation. They also propose a recognition and a parsing algorithm with complexity of $O(n^6)$ in the length of the string.

A.4 Multiple Adjunction in Feature-Based TAG

In this section, we explain why a straightforward extension of Schabes and Shieber's proposal to FB-TAG would not work and we outline the intuitions and motivations underlying our approach. Section A.5 will then introduce the details of our proposal.

A.4.1 Feature-Based Tree Adjoining Grammar

We start by a brief description of FB-TAG and of the unifications performed during derivation. FB-TAG was introduced by Vijay-Shanker (1987), Vijay-Shanker and Joshi (1988) and Vijay-Shanker and Joshi (1991) to support the use of feature structures in TAG. Figure A.7 shows a toy FB-TAG for illustration.

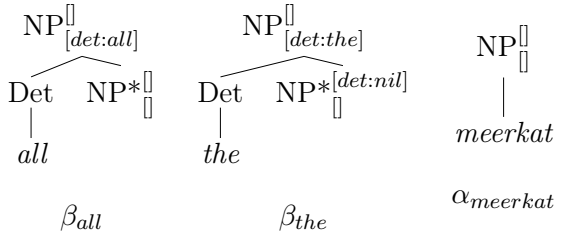


Figure A.7: A toy FB-TAG. For the sake of clarity, feature structures are abbreviated.

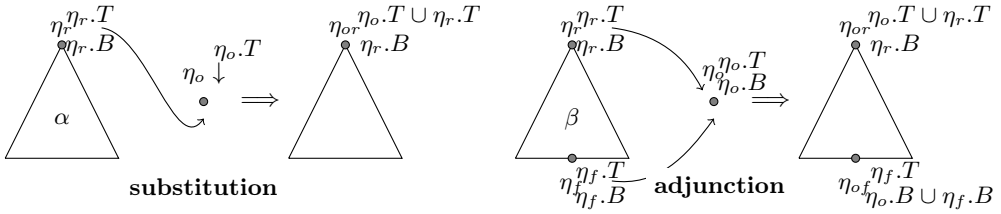


Figure A.8: Feature unifications along substitution and adjunction in FB-TAG.

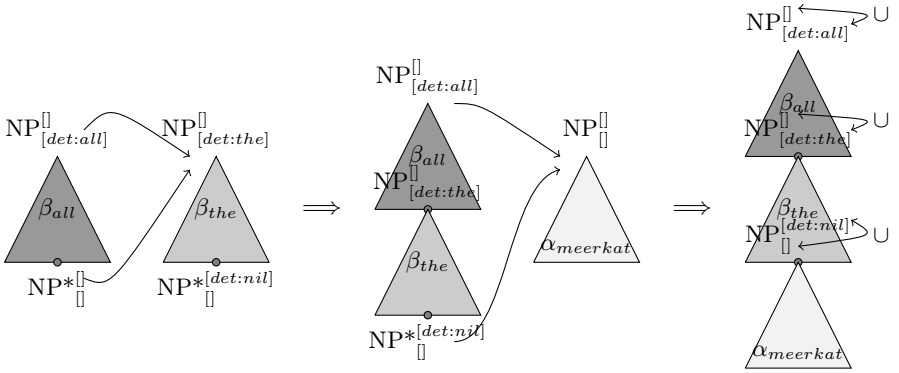


Figure A.9: Standard derivation for "all the meerkats".

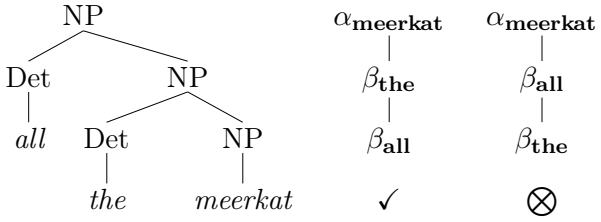


Figure A.10: The derived tree (left), the successful standard derivation tree (middle) and the failed dependent derivation tree (right) for "all the meerkats".

We have introduced FB-TAG in Chapter 2 but because of the requirement of this chapter, we repeat them here with relevant examples. An FB-TAG differs from a TAG in that tree nodes are decorated with feature structures. While nonterminal and foot nodes are decorated with two feature structures called top (T) and bottom (B), substitution nodes are decorated with a single top feature structure. During derivation, feature structure unification constrains tree combination as illustrated in Figure A.8. The node η_o (Figure A.8) in some elementary tree τ is the operation site for a substitution or an adjunction. For the sake of clarity, we only show the operation node η_o in Figure A.8. Substitution unifies the top feature structure of a substitution node with the top feature structure of the root node of the tree being

substituted in. The adjunction of an auxiliary tree β to a tree node η_o unifies the top and bottom feature structures of η_o with the top feature structure of the root node of β and the bottom feature structure of its foot node respectively. Finally, at the end of the derivation, the top and bottom feature structures of all nodes in the derived tree are unified.

Figure A.9 shows the standard derivation for the phrase *all the meerkats* using the grammar shown in Figure A.7 while Figure A.10 shows the corresponding derived and derivation trees. As can be seen, the feature constraints encoded in the grammar correctly ensure that *all the meerkats* can be derived (leftmost derivation tree in Figure A.10) but not *the all meerkats* (rightmost in Figure A.10). The incorrect derivation is blocked by the feature structure $[det : nil]$ on the foot of the auxiliary tree β_{the} which leads to a unification failure if β_{the} is adjoined at the root of β_{all} with bottom feature structure $[det : the]$.

A.4.2 Why a Simple Extension of the LIG Framework to FB-TAG will not Work?

To motivate our approach, we start by considering a simple extension of Schabes and Shieber's LIG framework to FB-TAG where each LIG rule enforces unifications mimicking those applied in FB-TAG. In particular, let us assume that Type 3 rules ("No adjunction") unify the top and the bottom feature structures of nodes where no adjunction occurs while Type 4(b) rules ("Start root of adjunction") unify the top feature ($\eta.T$) of the node (η) being adjoined to with the top feature structure ($\eta_r.T$) of the root node (η_r) of the auxiliary tree being adjoined[43]:

$$b[..\eta] \rightarrow t[..\eta\eta_r] \qquad \eta.T \cup \eta_r.T \qquad \text{(Type 4b)} \qquad \text{(A.4)}$$

$$t[..\eta] \rightarrow b[..\eta] \qquad \eta.T \cup \eta.B \qquad \text{(Type 3)} \qquad \text{(A.5)}$$

As shown in Figure A.11, this approach can incorrectly lead to derivation failures in the case of an independent multiple adjunction. Type 4(b) rules unify $\text{NP}_m.T$ with both $\text{NP}_{the}.T$ and $\text{NP}_{all}.T$ while Type 3 rules unify $\text{NP}_{the}.T$ with $\text{NP}_{the}.B$ and $\text{NP}_{all}.T$ with $\text{NP}_{all}.B$. Hence $\text{NP}_{the}.B$ and $\text{NP}_{all}.B$ should unify. However since their *det* values differ, derivation fails.

Intuitively, the reason for this is that, in the Schabes and Shieber's approach, multiple adjunction starts and ends from the bottom component of the node being

[43] We associate $\eta.T \cup \eta_r.T$ with the Type 4(b) rules to mimic the adjunction in FB-TAG as shown in Figure A.8.

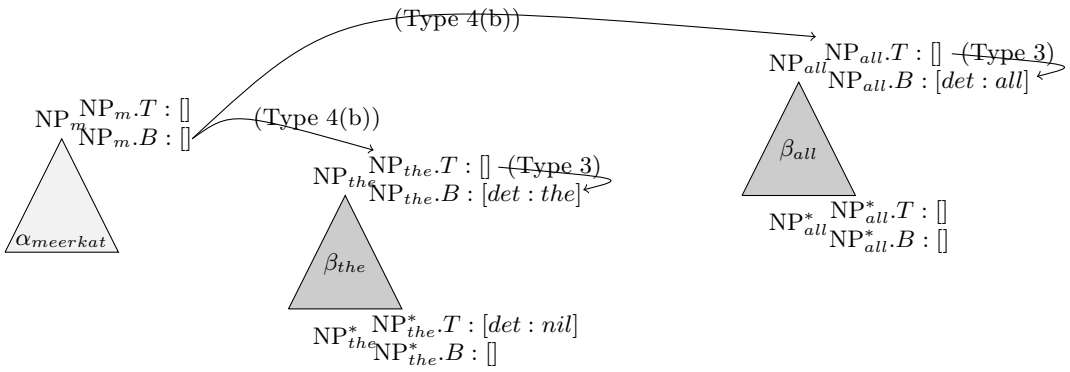


Figure A.11: Failed derivation under a simple extension of the Schabes and Shieber's LIG framework to FB-TAG.

adjoined to. This is fine when no features are involved because the category of the node being adjoined to is always identical to the root and foot node of the auxiliary trees being adjoined. When nodes carry feature structures however, a unification clash can occur which makes derivation fail. Thus in our example, derivation incorrectly fails because the bottom feature structures of the root node of the auxiliary tree for *all* and the bottom feature structure of the root node of the auxiliary tree for *the* should unify but have conflicting value. As shown by the dependent derivation for *all the meerkats* depicted in Figure A.9, this is incorrect.

A.4.3 Proposal: Intuition and Motivations

As we just saw, in the case of multiple independent adjunctions, a straightforward extension of Schabes and Shieber's LIG framework to FB-TAG fails to correctly capture the unification constraints encoded in the grammar. More generally, when extending multiple independent adjunction to FB-TAG, it is crucial that the feature constraints encoded by the linguist describe the same set of derived trees no matter which derivation tree is produced. We therefore propose a parsing algorithm which, given several auxiliary trees $\beta_1, \ldots, \beta_n$ adjoining at the same node η_o, performs the same unifications independently of whether the derivation is dependent, independent or mixed dependent/independent.

Figure A.12 shows the unifications resulting from the multiple adjunction of β_1 and β_2 to a single node η_o. While it depicts the unifications enforced by our parsing algorithm for the derivation tree shown on the right hand side namely for the independent adjunction of β_1 and β_2 to η_o, these unifications are in fact exactly the same as those that would be enforced by a dependent adjunction of β_2 into β_1 into

η_o.

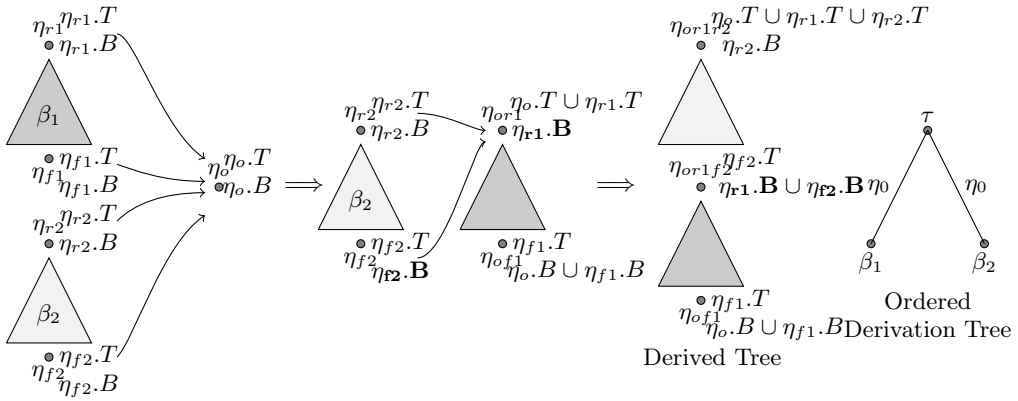


Figure A.12: Independent derivation and feature unification.

One key point illustrated by Figure A.12 is that while multiple adjunction operates on a single node (here η_o), the unification constraints of FB-TAG require that the bottom feature structure of the foot of an auxiliary tree which appears higher in the derived tree (here, β_2) unifies with the bottom feature structure of the root of the auxiliary tree appearing immediately below it in the derived tree (here β_1) – not with that of the root of the node to which it adjoins (here η_o). In other words, while a multiple adjunction on η_o operates on η_o only, a correct implementation of FB-TAG unification constraints requires keeping track of the feature structures associated with the auxiliary trees successively adjoining to η_o, i.e., although both β_1 and β_2 adjoin to η_o, the adjunction of β_2 requires access to the feature structure of the root of β_1 ($\eta_{\mathbf{f2}}.\mathbf{B} \cup \eta_{\mathbf{r1}}.\mathbf{B}$).

In our proposal, we capture this bookkeeping requirement by associating tree nodes not with feature structures but with reference variables pointing to feature structures. The parsing algorithm is then specified so as to support dependent, independent and mixed derivations while enforcing the same unifications as would be performed under a dependent adjunction.

A.4.4 Comparison with Schabes and Shieber's Approach

Before giving the technical details of our parsing algorithm (cf. Section A.5), we first highlight some differences between our and Schabes and Shieber's approach. In particular, we show (i) that whereas Schabes and Shieber resort to three distinct mechanisms to account for word order constraints (namely, selective adjoining constraints, linear precedence statements on derivation trees and a constraint on parsing), the FB-TAG approach supports a uniform treatment of word order and

(ii) that our approach straightforwardly accounts for mixed dependent/independent derivations which would require some additional stipulation in Schabes and Shieber's approach.

A.4.4.1 Ordering constraints among modifier auxiliary trees

In TAG, determiners and verbal auxiliaries are modifier rather than predicative auxiliary trees (cf. Section A.3.1). Because the Schabes and Shieber's definitions systematically associates modifiers with independent derivations, all examples in (27a-d) undergo an independent derivation and constraints must therefore be provided to determine the order of the sibling nodes in the resulting derivation tree.

(27) a. ✓The sonatas should have been being played by Sarah.

b. ⊗The sonatas have should been being played by Sarah.

c. ✓All the meerkats

d. ⊗The all meerkats

To specify these constraints on similar cases (soft ordering constraints on adjectives and strict ordering constraints on temporal and spatial adverbial phrases in German), Schabes and Shieber (1994) suggest the use of linear precedence constraints (LP) on derivation tree siblings. As illustrated by the derivation of example (27c-d) in Figure A.9 and A.10, in the FB-TAG approach, such additional constraints are unnecessary: they simply fall out of the feature constraints encoded in the grammar.

Note that even if determiners and auxiliary verbs were to be handled using dependent adjunction, the word ordering constraints used by Schabes and Shieber would fail to account for cases such as (28) where auxiliary verbs are interleaved with adverbs.

(28) John has often been selected for nomination.

In this case, if the auxiliary verbs *has* and *been* were treated as predicative trees, Schabes and Shieber's constraint that predicative trees adjoin above modifier trees would preclude the derivation of (28) and incorrectly predict the derived sentence to be *John has been often selected for nomination.*

A.4.4.2 Ordering constraints among predicative trees

As discussed by Schabes and Shieber (1994), auxiliary predicative trees may impose different constraints on the type of sentential complement they accept. Thus example (29a) is correct but not example (29b) because *want* expects an infinitival complement (previously shown in example (24)).

(29) a. ✓John wanted to assume that Peter slept.

b. ⊗John wanted Peter tries to walk.

While in the Schabes and Shieber's approach, selective adjoining constraints are used to license (29a) and rule out (29b), in the FB-TAG approach, this can be achieved using feature constraints.

A.4.4.3 Ordering Constraints between Predicative and Modifier Auxiliary Trees

In sentences such as (30a) where both modifier and predicative auxiliary trees adjoin to the same address, the predicative trees should generally adjoin above any modifier trees so that the predicative verb precedes the modifier in the derived string.

(30) a. ✓John promised that Peter will leave tomorrow.

b. ⊗Tomorrow John promised that Peter will leave.

To ensure the appropriate linearisation, the Schabes and Shieber's approach introduces the *outermost-predication rule* which stipulates that predicative trees adjoin above modifier auxiliary trees. In contrast, the FB-TAG approach allows both orders and lets feature constraints rule out ungrammatical sentences such as (30b). This allows the approach to directly extend to a counter-example discussed by Schabes and Shieber (1994) where a modifier (here *At what time*) must in fact adjoin above a predicative tree.

(31) At what time did Brockway say Harrison arrived?

Figure A.13 shows two possible derivation trees for the sentence (31) under the interpretation where it is the time of arriving (rather than the time of saying) which is questioned. These derivation trees show the two possible relative orderings of the (predicative) auxiliary tree for *say* and the (modifier) auxiliary tree *at what time*. Because the outermost-predication rule requires that predicative trees adjoin

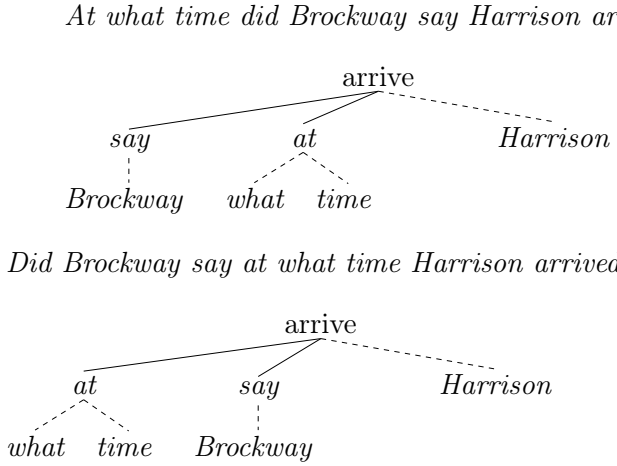


Figure A.13: Ordered derivation trees for the sentence (31) (Dotted lines indicate substitutions and plain lines adjunctions).

above modifier trees (and thus occur outermost in the derivation tree), in Schabes and Shieber's approach, only the bottom derivation is possible thus failing to derive sentence (31). In contrast, since our approach does not explicitly constrain the relative ordering of predicative and modifier auxiliary trees adjoining to the same node, both derivations are possible thereby licensing both the sentence (31) and the sentence *Did Brockway say at what time Harrison arrived?*

A.4.4.4 Mixed Dependent and Independent Multiple Adjunctions

In Schabes and Shieber's approach, all modifier auxiliary trees undergo independent derivation. As shown in Section A.2.2 however, non-intersective modifiers arguably license a dependent derivation while some cases of multiple adjunction may involve both a dependent and an independent derivation. As we shall see in Section A.5, our FB-TAG approach accounts for such cases by allowing both for independent and dependent derivations, by ruling out dependent derivations for intersective modifiers and by using feature constraints to regulate the interactions between multiply adjoining auxiliary trees.

A.5 Extending Schabes and Shieber's LIG Framework for FB-TAGs

We now propose a compilation of FB-TAG to LIG which makes both dependent and independent derivations in FB-TAG explicit. We use this resulting LIG to specify an

Earley algorithm for recovering multiple adjunctions in FB-TAG. This compilation differs in two main ways from that proposed by Schabes and Shieber (1994). First, tree nodes are associated with reference variables pointing to feature structures. Second, the LIG rules are modified and extended with unification operations.

A.5.1 Feature Structures and Reference Variables

To account for FB-TAG unifications while allowing for independent derivations, we replace the feature structures of FB-TAG with reference variables pointing to those feature structures. Each node in the elementary trees is decorated with two reference variables: the top reference variable P_T contains the reference to the top feature structure T and the bottom reference variable P_B contains the reference to the bottom feature structure B. The top and the bottom feature structures of a node η can be traced by $val(\eta.P_T)$ and $val(\eta.P_B)$ respectively where P_T and P_B are the top and the bottom reference variables decorating the node η and the function $val(P)$ returns the feature structures referred to by the reference variable P.

When specifying the parsing algorithm, we use reference variables to ensure the appropriate unifications as follows. In an independent derivation where the node η_o is adjoined to, first by β_1 and second by β_2, the bottom feature structure $\eta_o.B$ of η_o (i) unifies with the bottom feature structure $\eta_{f1}.B$ of the foot of β_1 and (ii) is reassigned (:=) to the bottom reference variable $\eta_{r1}.P_B$ of the root of β_1. When β_2 is adjoined, its foot node will therefore correctly be unified, not with the bottom feature structure of η_o but with that of η_{r1}.

A.5.2 LIG Rules with Unification Operations

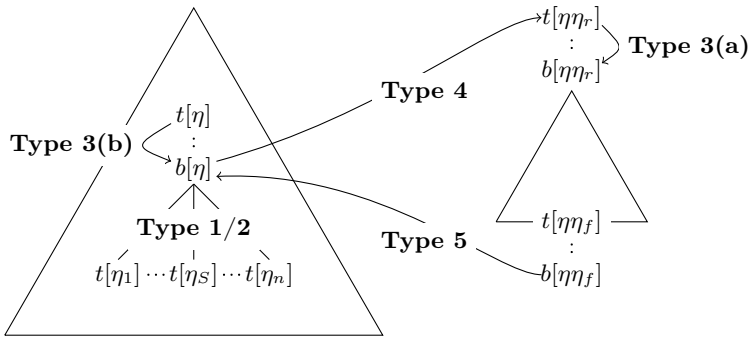


Figure A.14: LIG variant of TAG for the extended derivation in FB-TAG.

To support both dependent and independent derivations while enforcing the correct unifications, we modify the TAG to LIG compilation in such a way that the

resulting LIG rules capture the tree traversal depicted in Figure A.14. Independent derivations are accounted for by the fact that adjunction starts and ends at the bottom component of the node being adjoined to (Type 4 and 5 rules). Our LIG compilation automatically supports dependent derivations by allowing sequential adjunctions at the roots of auxiliary trees.

Type 4: *Start root of adjunction.* For each elementary tree node η which allows the adjunction of the auxiliary tree with the root node η_r, the following LIG production rule is generated.

$$b[..\eta] \rightarrow t[..\eta\eta_r] \qquad val(\eta.P_T) \cup val(\eta_r.P_T), \eta.P_B := \eta_r.P_B$$

Type 5: *Start foot of adjunction.* For each elementary tree node η which allows the adjunction of the auxiliary tree with the foot node η_f, the following LIG production rule is generated.

$$b[..\eta\eta_f] \rightarrow b[..\eta] \qquad val(\eta.P_B) \cup val(\eta_f.P_B)$$

Type 6: *Start substitution.* For each elementary tree node η which allows the substitution of the initial tree with the root node η_r, the following LIG production rule is generated (not shown in Figure A.14).

$$t[\eta] \rightarrow t[\eta_r] \qquad val(\eta.P_T) \cup val(\eta_r.P_T)$$

To perform multiple adjunction while enforcing the appropriate feature unifications (as depicted in Figure A.12), we split Type 3 rules into two subtypes. Type 3(a) rules apply to the root of auxiliary trees and perform no unification. By no unification, they ensure that feature structures are not blocked for the possibility of the adjunction of the following auxiliary tree and allow for the correct unifications to be carried out for independent derivations. Type 3(b) rules function as termination of multiple adjunction by unifying the top and bottom feature structures of the node. It is applicable to all tree nodes except roots of auxiliary trees.

Type 3(a): *terminating adjunction at the root of the auxiliary tree.* For each root node η of the auxiliary trees, the following LIG production rule is generated.

$$t[..\eta] \rightarrow b[..\eta] \qquad \emptyset$$

Type 3(b): *terminating adjunction at any other node.* For each node η that is not a root node of some auxiliary tree and is not marked for substitution, the following LIG production rule is generated.

$$t[..\eta] \rightarrow b[..\eta] \qquad val(\eta.P_T) \cup val(\eta.P_B)$$

Given this set of rules, both dependent and independent derivations are possible. For example, given two auxiliary trees β_1 and β_2 adjoining at the node η in an elementary tree τ, a dependent derivation will occur whenever the Type 4 rule applies to predict the adjunction of, e.g., β_2 at the root of β_1. Conversely, if the Type 3(a) rule applies at the root of β_1, recognition will move from the top of the root of β_1 to its bottom allowing for Type 5 rule to complete the adjunction of β_1 at the node η and the Type 4 rule applies to predict the adjunction of β_2 at the node η of τ, registering an independent derivation.

A.5.3 Parsing Algorithm

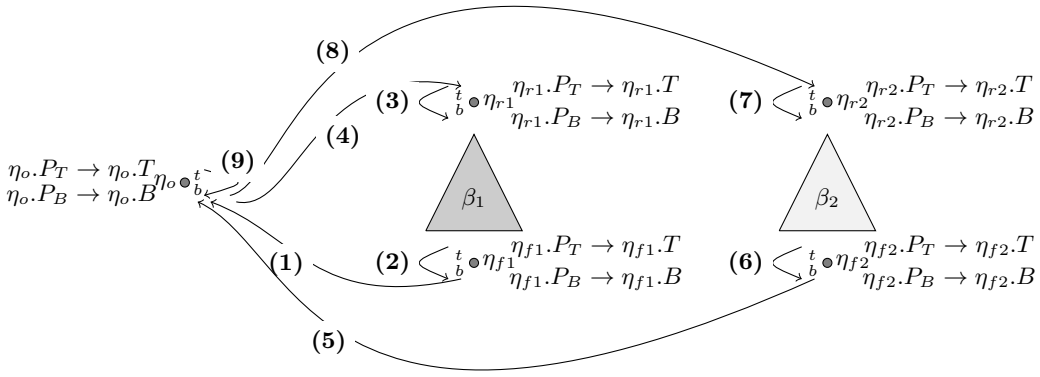
In this section, we present our parsing algorithm for FB-TAGs with dependent and independent derivations. We start with an informal description of how the algorithm handles the interactions between unification and independent derivations. We then go on to specify the inference rules making up the algorithm. We do this in two steps. First, we present a basic set of rules allowing for both dependent and independent derivations. Second, we show how to constrain this algorithm to minimize spurious ambiguity.

A.5.3.1 Independent Derivations and Feature-Structure Unification

Before specifying the parsing algorithm, we illustrate by means of an example the interplay between multiple independent adjunction and feature structure unifications.

Figure A.15 displays the feature unifications and reassignment performed during the recognition process of a multiple independent adjunction. The linear ordering of the equations capturing the unifications and reassignments reflects the order of the parsing completion operations.

Given the auxiliary tree β_1 and the adjunction site η_o, the picture shows that unifying the bottom feature structure of the foot node of β_1 with the bottom feature structure of η_o (Step 1: Type 5, $\eta_o.B \cup \eta_{f1}.B$) occurs before the bottom reference variable of η_o is reassigned to the bottom feature structure of the root of β_1 (Step 4: Type 4, $\eta_o.P_B \rightarrow \eta_{r1}.B$). Also the reassignment ensures that the follow up adjunction



(1). Type 5, $\eta_o.P_B \to F_1, \eta_{f1}.P_B \to F_1, F_1 = \eta_{\mathbf{o}}.\mathbf{B} \cup \eta_{\mathbf{f1}}.\mathbf{B}$
(2). Type 3(b), $\eta_{f1}.P_T \to F_2, \eta_{f1}.P_B \to F_2, F_2 = \eta_o.B \cup \eta_{f1}.T \cup \eta_{f1}.B$
(3). Type 3(a), $\eta_{r1}.P_T \to \eta_{r1}.T, \eta_{r1}.P_B \to \eta_{r1}.B$
(4). Type 4, $\eta_o.P_T \to F_3, \eta_{r1}.P_T \to F_3, F_3 = \eta_o.T \cup \eta_{r1}.T, \eta_{\mathbf{o}}.\mathbf{P_B} \to \eta_{\mathbf{r1}}.\mathbf{B}$
(5). Type 5, $\eta_o.P_B \to F_4, \eta_{f2}.P_B \to F_4, F_4 = \eta_{r1}.B \cup \eta_{f2}.B$
(6). Type 3(b), $\eta_{f2}.P_T \to F_5, \eta_{f2}.P_B \to F_5, F_5 = \eta_{r1}.B \cup \eta_{f2}.T \cup \eta_{f2}.B$
(7). Type 3(a), $\eta_{r2}.P_T \to \eta_{r2}.T, \eta_{r2}.P_B \to \eta_{r2}.B$
(8). Type 4, $\eta_o.P_T \to F_6, \eta_{r2}.P_T \to F_6, F_6 = \eta_o.T \cup \eta_{r1}.T \cup \eta_{r2}.T, \eta_o.P_B \to \eta_{r2}.B$
(9). Type 3(b), $\eta_o.P_T \to F_7, \eta_o.P_B \to F_7, F_7 = \eta_o.T \cup \eta_{r1}.T \cup \eta_{r2}.T \cup \eta_{r2}.B$

Figure A.15: Multiple independent adjunction of β_1 and β_2 to η_o.

of β_2 at the node η_o has access to the bottom feature of the root of the previous auxiliary tree β_1 (Step 5: Type 5, $\eta_{r1}.B \cup \eta_{f2}.B$). At the end of the adjunction (Step 9), the Type 3(b) rule ensures that the top and the bottom features of the root of the last auxiliary tree (here, β_2) adjoined are unified ($\eta_{r2}.T \cup \eta_{r2}.B$).

As we shall see below, this correct ordering between unification and reassignment follows from the proposed Earley algorithm. Type 4 completor rules complete the prediction triggered at the root of an auxiliary tree ("Start root of adjunction") while Type 5 completor rules complete the prediction triggered at the foot node of an auxiliary tree ("Start foot of adjunction"). Since completion operates bottom-up, it follows that Type 5 Rules apply before Type 4 rules. Thus, when adjoining an auxiliary tree β_1 to a node η_o, the Type 5 completor rules, unifying the bottom feature structure of the foot node of β_1 with the bottom feature structure of the node η_o, occurs before the Type 4 completor rules which reassign the bottom reference variable of η_o to the bottom feature structure of the root of β_1.

A.5.3.2 Inference Rules

The parsing algorithm for FB-TAG is a modification of the algorithm presented by Schabes and Shieber (1994). It is a chart-based parsing method based on the Earley type deduction system. Each item in the chart is of the format $\langle N[..\eta] \rightarrow \Gamma \bullet \Delta, i, j, k, l \rangle$ where N is some LIG nonterminal (i.e., t or b) and, Γ and Δ present the sequences of LIG nonterminals associated with stacks of node indices. The indices i, j, k, and l are markers in the input string showing the recognized portion[44]: the recognized item starts in position i, ends in position l and if η dominates a foot node, the tree dominated by the foot node starts in j and ends in k. If the foot node is not dominated by the recognized nonterminal sequence Γ, the values for j and k are taken to be the dummy value '$-$'. As in Earley algorithms, the $\bullet$ separates the nonterminal sequence Γ which was parsed from the nonterminal sequence Δ yet to be parsed.

The first three types of rules (Scanner, Predictor and Type 1/2 Completor) are identical to those introduced by Schabes and Shieber (1994) and do not involve any unification operations.

The Type 3(b) completor rule enforces top and bottom unification on all nodes which are not the root of an auxiliary tree while the Type 3(a) completor rule prevents top and bottom unification at the root of auxiliary trees.

The Type 4 completor rule unifies the top feature structure of the root of the auxiliary tree with the top feature structure of the adjunction site. In addition, it ensures that on completion of an adjunction at node η, the bottom feature structure of η is reassigned to the bottom feature structure labelling the root of the auxiliary tree. In this way, the unifications occurring in an independent derivation will mirror those occurring in a dependent one in that any following adjunction will induce unifications as if it were happening at the root node η_r of the preceding auxiliary tree (not at η).

On completion of a foot node prediction (the tree dominated by the foot of the auxiliary tree has been recognized), the Type 5 completor rule unifies the bottom feature structure of the foot of the auxiliary tree with the bottom feature structure of the adjunction site.

Finally, the Type 6 completor unifies the top feature structure of a substitution node with the top feature structure of the root of the tree being substituted in.

[44]The indices $\langle i, j, k, l \rangle$ have been used in previous parsing algorithms for tree-adjoining grammars [Vijay-Shankar and Joshi, 1985; Schabes and Joshi, 1988; Schabes, 1991]. They deliver the same functionality here.

- *Scanner:*

$$\frac{\langle b[..\eta] \rightarrow \Gamma \bullet w\Delta, i, j, k, l\rangle}{\langle b[..\eta] \rightarrow \Gamma w \bullet \Delta, i, j, k, l+1\rangle}, \qquad w = w_{l+1}, \qquad \emptyset$$

  If w (a terminal symbol) occurs at position $l{+}1$, the scanner rule creates a new item whose span extends to $l{+}1$.

- *Predictor:*

$$\frac{\langle N[..\eta] \rightarrow \Gamma \bullet N'[\mu]\Delta, i, j, k, l\rangle}{\langle N'[\mu] \rightarrow \bullet\Theta, l, -, -, l\rangle}, \qquad \emptyset$$

  Predictor rules are produced for all types of production rules. N and N' are LIG variables taking the value t or b. Γ, Δ and Θ are the sequences of LIG nonterminals associated with stacks of node indices. μ is a sequence of node indices.

- *Type 1 and 2 Completor:*

$$\frac{\langle b[..\eta] \rightarrow \Gamma \bullet t[\eta_1]\Delta, m, j', k', i\rangle \quad \langle t[\eta_1] \rightarrow \Theta\bullet, i, j, k, l\rangle}{\langle b[..\eta] \rightarrow \Gamma t[\eta_1] \bullet \Delta, m, j \oplus j', k \oplus k', l\rangle}, \qquad \begin{array}{l}\emptyset,\\ \eta_1 \text{ not a root node}\end{array}$$

  Type 1 and 2 Completor rules permit completing Rules 1 and 2 whenever the top of a child node is fully recognized. Here, $t[\eta_1]$ has been fully recognized as the substring between i and l (i.e., $w_{i+1} \ldots w_l$). Therefore, $t[\eta_1]$ can be completed in $b[..\eta]$. If one of $t[\eta_1]$ or $b[..\eta]$ dominates the foot node of the tree, the final $b[..\eta]$ will have indices associated with the substring recognized by the foot subtree. The operation $\oplus$ is defined as follows:

$$x \oplus y = \begin{cases} x, & \text{if } y = -. \\ y, & \text{if } x = -. \\ x, & \text{if } x = y. \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

- *Type 3(a) Completor:*

$$\frac{\langle t[..\eta] \rightarrow \bullet b[..\eta], i, -, -, i\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l\rangle}{\langle t[..\eta] \rightarrow b[..\eta]\bullet, i, j, k, l\rangle}, \qquad \begin{array}{l}\emptyset,\\ \eta \text{ an auxiliary tree root node}\end{array}$$

  This rule is used to complete the prediction of an auxiliary tree rooted in η. Once the auxiliary tree dominated by $b[..\eta]$ has been recognized, the auxiliary

tree itself is completely recognized. As explained above, there is in this case no feature unification between the top and the bottom of the root of the auxiliary tree.

- *Type 3(b) Completor:*

$$\frac{\langle t[..\eta] \rightarrow \bullet b[..\eta], i, -, -, i\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l\rangle}{\langle t[..\eta] \rightarrow b[..\eta]\bullet, i, j, k, l\rangle}, \qquad \begin{array}{l} val(\eta.P_T) \cup val(\eta.P_B), \\ \eta \text{ not an auxiliary tree root node} \end{array}$$

This completion rule ensures the unification of the top and bottom feature structures for all nodes that are not the root node of an auxiliary tree.

- *Type 4 Completor:*

$$\frac{\begin{array}{r} \langle b[..\eta] \rightarrow \bullet t[..\eta\eta_r], i, -, -, i\rangle \\ \langle t[..\eta\eta_r] \rightarrow \Theta\bullet, i, j, k, l\rangle \\ \langle b[..\eta] \rightarrow \Delta\bullet, j, p, q, k\rangle \end{array}}{\langle b[..\eta] \rightarrow t[..\eta\eta_r]\bullet, i, p, q, l\rangle}, \qquad \begin{array}{c} val(\eta.P_T) \cup val(\eta_r.P_T) \\ \eta.P_B := \eta_r.P_B \end{array}$$

The auxiliary tree associated with the predicted adjunction ($t[..\eta\eta_r]$) at the node η and the subtree dominated by the node η (below $b[..\eta]$) are completed, hence, $b[..\eta]$ can be completely recognized with this adjunction. The associated feature unification unifies the content of the top reference variable of the adjoining node site η with the content of the top reference variable of the root node η_r of the adjoined auxiliary tree. After the successful adjunction of this adjoining tree, the bottom reference variable of the adjoining node site η is reassigned to the content of the bottom reference variable of the root node η_r of the adjoined auxiliary tree.

- *Type 5 Completor:*

$$\frac{\langle b[..\eta\eta_f] \rightarrow \bullet b[..\eta], i, -, -, i\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l\rangle}{\langle b[..\eta\eta_f] \rightarrow b[..\eta]\bullet, i, i, l, l\rangle}, \qquad val(\eta.P_B) \cup val(\eta_f.P_B)$$

The foot node prediction can be completed when the adjunction has been performed and the bottom part of the adjoining node site η has been recognized. The associated feature unification unifies the content of the bottom reference variable of the adjoining node site η with the content of the bottom reference variable of the foot node η_f of the auxiliary tree being adjoined.

- *Type 6 Completor:*

$$\frac{\langle t[\eta] \rightarrow \bullet t[\eta_r], i, -, -, i\rangle \quad \langle t[\eta_r] \rightarrow \Theta\bullet, i, -, -, l\rangle}{\langle t[\eta] \rightarrow t[\eta_r]\bullet, i, -, -, l\rangle}, \qquad val(\eta.P_T) \cup val(\eta_r.P_T)$$

  This rule completes the substitution at the node η. The associated feature unification unifies the content of the top reference variable of the node η with the content of the top reference variable of the root node η_r of the initial tree.

Given these inference rules, the recognition process is initialized using axioms of the form $\langle t[\eta_s] \rightarrow \bullet\Gamma, 0, -, -, 0\rangle$ for each rule $t[\eta_s] \rightarrow \Gamma$ where η_s is the root node of an initial tree labeled with the start symbol. Given an input string $w_1 \ldots w_n$ to be recognized, the goal items in the chart are of the form $\langle S \rightarrow t[\eta_s]\bullet, 0, -, -, n\rangle$. Once at least one goal item is found in the chart, the recognition process succeeds and the string is successfully accepted by the grammar, otherwise it is rejected. We refer the reader to Section A.6 (cf. Figure A.18) at the end of the chapter for a detailed example of the recognition of the sentence "all the meerkats" using the proposed inference system.

Note also that while the recognition algorithm we described uses unreduced rules i.e., generated grammar rules maintaining the full information of nonterminals and the associated index stacks, it is possible to define a more efficient algorithm by having reduced LIG rules and chart items listing only the single top stack element for each constituent [Vijay-Shanker and Weir, 1991; Vijay-Shanker and Weir, 1993]. The resulting recognition algorithm is still complete because the proposed TAG to LIG compilation maintains a one-to-one correspondence between the generated rules and their reduced forms [Schabes and Shieber, 1994].

As mentioned by Schabes and Shieber (1994), this recognition algorithm can be turned into a parsing algorithm by associating a set of operations with each chart item to build up associated derived trees.

Note also that the derivation trees built as a side effect of the parsing process are the (dependent and/or independent) derivation trees of an FB-LTAG and are therefore context-free.

A.5.3.3 Handling Spurious Parses

As explained at the end of Section A.5.2, the parsing algorithm presented in the previous section systematically allows for dependent and independent adjunction. For example, the recognition of the sentence "all the meerkats" (Section A.6, Figure A.18) produces both dependent and independent derivations which are not rejected by the

unification constraints. In Section A.2.2 however, we argued that different types of auxiliary trees license different types of derivations. To capture these distinctions, we modify the recognition algorithm so that it associates scopal auxiliary trees (e.g., 32a-b) with dependent derivations only and multiple intersective modifier auxiliary trees (32c) with only an independent derivation.

(32) a. John thinks that Peter said that the meerkat left.

b. The meerkat admired the Syrian orthodox church.

c. The tall black meerkat slept.

To block dependent adjunctions between intersective modifiers, we modify the TAG to LIG transformation so that given two intersective modifier trees β_1 and β_2, no Type 4 or Type 5 rule is produced.

Type 4: *Start root of adjunction.* For each elementary tree node η in tree β_1 which allows the adjunction of the auxiliary tree β_2 with root node η_r, the following LIG production rule is generated *if and only if* β_1 *and* β_2 *are not intersective modifier auxiliary trees.*

$$b[..\eta] \rightarrow t[..\eta\eta_r] \qquad val(\eta.P_T) \cup val(\eta_r.P_T)$$

Type 5: *Start foot of adjunction.* For each elementary tree node η which allows the adjunction of the auxiliary tree β_2 with the foot node η_f, the following LIG production rule is generated *if and only if* β_1 *and* β_2 *are not intersective modifier auxiliary trees.*

$$b[..\eta\eta_f] \rightarrow b[..\eta] \qquad val(\eta.P_B) \cup val(\eta_f.P_B)$$

Thus for instance, in the derivation of *All the meerkats* depicted in Figure A.18 (Section A.6), the following rules will not be produced thereby blocking the production of the dependent derivation.

$$\left.\begin{array}{l} b[..\ \mathrm{NP}_r^{the}] \rightarrow t[..\ \mathrm{NP}_r^{the}\ \mathrm{NP}_r^{all}] \\ b[..\ \mathrm{NP}_r^{all}] \rightarrow t[..\ \mathrm{NP}_r^{all}\ \mathrm{NP}_r^{the}] \end{array}\right\} Type4$$

$$\left.\begin{array}{l} b[..\ \mathrm{NP}_r^{all}\ \mathrm{NP}_f^{the}] \rightarrow b[..\ \mathrm{NP}_r^{all}] \\ b[..\ \mathrm{NP}_r^{the}\ \mathrm{NP}_f^{all}] \rightarrow b[..\ \mathrm{NP}_r^{the}] \end{array}\right\} Type5$$

Similarly, to block independent adjunctions between scopal auxiliary trees, we add a flag *scopal?* to states in the parsing algorithm. The Type 4 completor rules associated with scopal modifiers are modified to mark the progress of a scopal adjunction and to block the independent adjunction of another scopal modifier at the same node.

Type 4 Completor:

$$\frac{\begin{array}{r}\langle b[..\eta] \rightarrow \bullet t[..\eta\eta_r], i, -, -, i, \mathit{scopal?}\rangle \\ \langle t[..\eta\eta_r] \rightarrow \Theta\bullet, i, j, k, l, \mathit{scopal?}\rangle \\ \langle b[..\eta] \rightarrow \Delta\bullet, j, p, q, k, \mathit{scopal?}\rangle\end{array}}{\langle b[..\eta] \rightarrow t[..\eta\eta_r]\bullet, i, p, q, l, \mathit{True}\rangle}, \quad \begin{array}{c} val(\eta.P_T) \cup val(\eta_r.P_T) \\ \eta.P_B := \eta_r.P_B \end{array}$$

Once a scopal auxiliary tree β with root node η_r adjoins at some node η, the bottom component of the node η is marked with *True* recording that a scopal adjunction has occurred at node η and that it therefore should not accept any further scopal adjunction.

Thus for instance, the derivation of *Syrian orthodox churches* will proceed in a similar manner as the derivation of *All the meerkats* depicted in Figure A.18 (Section A.6) but it will fail to produce the chart items $(40, 42, ..., 52)$ associated with the independent adjunction. Therefore, only the dependent derivation will be produced.

Note that the above modification does not block modifier adjunction above a predicative adjunction. Therefore, it successfully recognizes the sentence *At what time did Brockway say Harrison arrived?*, shown in (31a), where a wh-modifier needs to be adjoined above a predicative adjunction. Figure A.16 shows the complete recognition algorithm modified to rule out spurious parses in the case of multiple scopal auxiliary trees and intersective modifier auxiliary trees.

A.5.3.4 Weak Generative Equivalence

The weak-generative equivalence refers to the set of strings characterized by the formal system. In contrast, the strong-generative equivalence relates to the set of structural descriptions (such as derivation trees, dags, proof trees, etc.) assigned by a formal system to the strings that it specifies [Vijay-Shankar and Joshi, 1985; Joshi, 2000].

Using an argument similar to that put forward by Schabes and Shieber (1994), we can prove the weak-generative equivalence of TAGs under the dependent and

- *Scanner:*
$$\frac{\langle b[..\eta] \rightarrow \Gamma \bullet w\Delta, i, j, k, l, S?\rangle}{\langle b[..\eta] \rightarrow \Gamma w \bullet \Delta, i, j, k, l+1, S?\rangle}, \qquad w = w_{l+1}, \qquad \emptyset$$

- *Predictor:*
$$\frac{\langle P[..\eta] \rightarrow \Gamma \bullet P'[\mu]\Delta, i, j, k, l, -\rangle}{\langle P'[\mu] \rightarrow \bullet\Theta, l, -, -, l, S?\rangle}, \qquad \emptyset$$

- *Type 1 and 2 Completor:*
$$\frac{\langle b[..\eta] \rightarrow \Gamma \bullet t[\eta_1]\Delta, m, j', k', i, S?\rangle \quad \langle t[\eta_1] \rightarrow \Theta\bullet, i, j, k, l, -\rangle}{\langle b[..\eta] \rightarrow \Gamma t[\eta_1] \bullet \Delta, m, j \oplus j', k \oplus k', l, S?\rangle}, \qquad \begin{array}{l}\emptyset,\\ \eta_1 \text{ not a root node}\end{array}$$

- *Type 3(a) Completor:*
$$\frac{\langle t[..\eta] \rightarrow \bullet b[..\eta], i, -, -, i, -\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l, S?\rangle}{\langle t[..\eta] \rightarrow b[..\eta]\bullet, i, j, k, l, S?\rangle}, \qquad \begin{array}{l}\emptyset,\\ \eta \text{ an auxiliary tree root node}\end{array}$$

- *Type 3(b) Completor:*
$$\frac{\langle t[..\eta] \rightarrow \bullet b[..\eta], i, -, -, i, -\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l, S?\rangle}{\langle t[..\eta] \rightarrow b[..\eta]\bullet, i, j, k, l, S?\rangle}, \qquad \begin{array}{l}val(\eta.P_T) \cup val(\eta.P_B),\\ \eta \text{ not an auxiliary tree root node}\end{array}$$

- *Type 4 Completor:*
$$\frac{\begin{array}{r}\langle b[..\eta] \rightarrow \bullet t[..\eta\eta_r^{scopal}], i, -, -, i, -\rangle\\ \langle t[..\eta\eta_r^{scopal}] \rightarrow \Theta\bullet, i, j, k, l, -\rangle\\ \langle b[..\eta] \rightarrow \Delta\bullet, j, p, q, k, S?\rangle\end{array}}{\langle b[..\eta] \rightarrow t[..\eta\eta_r^{scopal}]\bullet, i, p, q, l, \mathit{True}\rangle}, \qquad \begin{array}{c}val(\eta.P_T) \cup val(\eta_r^{scopal}.P_T)\\ \eta.P_B := \eta_r^{scopal}.P_{B_r}\end{array}$$

$$\frac{\begin{array}{r}\langle b[..\eta] \rightarrow \bullet t[..\eta\eta_r^{others}], i, -, -, i, -\rangle\\ \langle t[..\eta\eta_r^{others}] \rightarrow \Theta\bullet, i, j, k, l, -\rangle\\ \langle b[..\eta] \rightarrow \Delta\bullet, j, p, q, k, S?\rangle\end{array}}{\langle b[..\eta] \rightarrow t[..\eta\eta_r^{others}]\bullet, i, p, q, l, S?\rangle}, \qquad \begin{array}{c}val(\eta.P_T) \cup val(\eta_r^{others}.P_T)\\ \eta.P_B := \eta_r^{others}.P_{B_r}\end{array}$$

- *Type 5 Completor:*
$$\frac{\langle b[..\eta\eta_f^{scopal}] \rightarrow \bullet b[..\eta], i, -, -, i, S?\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l, S1?\rangle}{\langle b[..\eta\eta_f^{scopal}] \rightarrow b[..\eta]\bullet, i, i, l, l, S?\rangle}, \qquad \begin{array}{c}val(\eta.P_B) \cup val(\eta_f^{scopal}.P_{B_f})\\ S1? \neq \mathit{True}\end{array}$$

$$\frac{\langle b[..\eta\eta_f^{others}] \rightarrow \bullet b[..\eta], i, -, -, i, S?\rangle \quad \langle b[..\eta] \rightarrow \Theta\bullet, i, j, k, l, -\rangle}{\langle b[..\eta\eta_f^{others}] \rightarrow b[..\eta]\bullet, i, i, l, l, S?\rangle}, \qquad val(\eta.P_B) \cup val(\eta_f^{others}.P_{B_f})$$

- *Type 6 Completor:*
$$\frac{\langle t[\eta] \rightarrow \bullet t[\eta_r], i, -, -, i, S?\rangle \quad \langle t[\eta_r] \rightarrow \Theta\bullet, i, -, -, l, -\rangle}{\langle t[\eta] \rightarrow t[\eta_r]\bullet, i, -, -, l, S?\rangle}, \qquad val(\eta.P_T) \cup val(\eta_r.P_{T_r})$$

Figure A.16: Recognition algorithm taking into account spurious parses.

our independent derivations. We call the set of languages generated by the standard derivation in TAG, TAL_{std}; the set of languages generated by Schabes and Shieber's extended derivation in TAG, $TAL_{ext_{ss}}$; the set of languages generated with our modifications for FB-TAG, TAL_{ext} and the set of languages generated by the LIG, LIL. Our derivation allows both dependent and independent derivations, therefore, our derivation will recognize all the strings recognized by the standard (dependent) derivation. More precisely, our derivation can mimic the standard derivation by not allowing more than one adjunction on a tree node by treating all auxiliary trees as scopal auxiliary trees, cf. Section A.5.3.3, henceforth, $TAL_{std} \subseteq TAL_{ext}$. The proposed compilation from TAGs to LIGs for the independent derivation concluded $TAL_{ext} \subseteq LIL$. Finally, $LIL \subseteq TAL_{std}$ has been proven by Vijay-Shanker (1987). Combining these three inclusions, we can conclude that $TAL_{std} = TAL_{ext}$. In addition, Schabes and Shieber (1994) have shown that $TAL_{std} = TAL_{ext_{ss}}$. Hence, we can conclude the weak generative equivalence of all three derivations in TAGs, $TAL_{std} = TAL_{ext_{ss}} = TAL_{ext}$. Feature structures enhance TAGs descriptive ability without affecting their generative capacity [Vijay-Shanker and Joshi, 1988]. The proposed algorithm simulates the established unification mechanism in FB-TAG without affecting the representation and the stipulations (e.g., null adjunction at the foot node and the bounded feature structures) of the grammar itself. Therefore, the association with feature structures will not affect this equivalence.

A.6 Recognititon of the string "all the meerkats"

We show the recognition of the "all the meerkats" as per the inference rules described in Section A.5.3.2.

Figure A.17 shows the grammar used for the derivation. To support multiple adjunction in FB-TAG, it implements two main modifications. First, to facilitate the TAG to LIG compilation, each tree node in the grammar is marked with a unique identifier. For example, in Figure A.17, NP^{mk}, NP_r^{the}, Det^{the}, $\text{NP}_f^{the}*$, NP_r^{all}, Det^{all} and $\text{NP}_f^{the}*$ are unique node identifiers in the grammar. Second, to implement the reassignment mechanism in the parsing algorithm, the top (T) and the bottom (B) feature structures of each node are assigned reference variables P_T and P_B respectively.

Figure A.18 shows the LIG production rules for the FB-TAG shown in Figure A.17.

Table A.1 shows the step-wise recognition of the string "all the meerkats". Recall Section A.5.3.2, the LIG rules shown in Figure A.17 does not deal with spurious

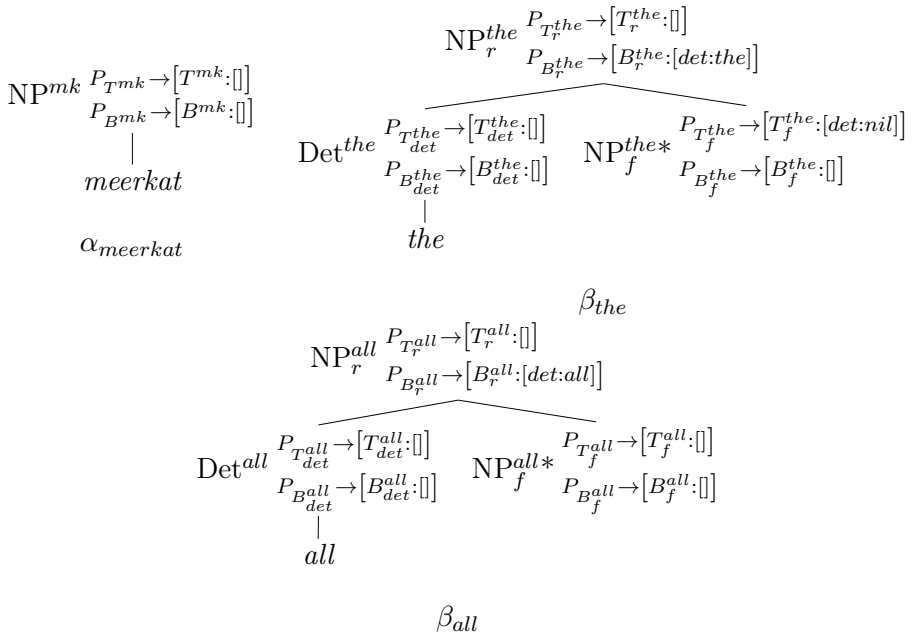


Figure A.17: A feature-based Tree-adjoining grammar.

parses and produces all valid derivations dependent or independent which are not blocked by feature unification constraints. Hence, for the string "all the meerkats", it generates both independent (step 52) and dependent (step 53) derivations. As explained in Figure A.19 and Figure A.20, both derivations undergo the identical set of feature unifications. In both figures, prediction rules are abbreviated because they do not enforce any feature unification.

Table A.1: Recognition of the string "$_0$ all $_1$ the $_2$ meerkats $_3$" in FB-TAG.

#	Chart Items	Description
1.	$\langle S \rightarrow \bullet t[\mathrm{NP}^{mk}], 0, -, -, 0\rangle$	axiom
2.	$\langle t[\mathrm{NP}^{mk}] \rightarrow \bullet b[\mathrm{NP}^{mk}], 0, -, -, 0\rangle$	3(b)-pred, 1
3.	$\langle b[\mathrm{NP}^{mk}] \rightarrow \bullet t[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{the}], 0, -, -, 0\rangle$	4-pred, 2
4.	$\langle b[\mathrm{NP}^{mk}] \rightarrow \bullet t[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{all}], 0, -, -, 0\rangle$	4-pred, 2
5.	$\langle t[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{the}] \rightarrow \bullet b[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{the}], 0, -, -, 0\rangle$	3(a)-pred, 3
6.	$\langle t[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{all}] \rightarrow \bullet b[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{all}], 0, -, -, 0\rangle$	3(a)-pred, 4
7.	$\langle b[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{the}] \rightarrow \bullet t[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{the}\ \mathrm{NP}_r^{all}], 0, -, -, 0\rangle$	4-pred, 5
8.	$\langle b[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{all}] \rightarrow \bullet t[\mathrm{Det}^{all}]\ t[\mathrm{NP}^{mk}\ \mathrm{NP}_f^{all}], 0, -, -, 0\rangle$	1/2-pred, 6
9.	$\langle t[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{the}\ \mathrm{NP}_r^{all}] \rightarrow \bullet b[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{the}\ \mathrm{NP}_r^{all}], 0, -, -, 0\rangle$	3(a)-pred, 7
10.	$\langle t[\mathrm{Det}^{all}] \rightarrow \bullet b[\mathrm{Det}^{all}], 0, -, -, 0\rangle$	3(b)-pred, 8
11.	$\langle b[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{the}\ \mathrm{NP}_r^{all}] \rightarrow \bullet t[\mathrm{Det}^{all}]\ t[\mathrm{NP}^{mk}\ \mathrm{NP}_r^{the}\ \mathrm{NP}_f^{all}], 0, -, -, 0\rangle$	1/2-pred, 9
12.	$\langle b[\mathrm{Det}^{all}] \rightarrow \bullet \mathrm{all}, 0, -, -, 0\rangle$	1/2-pred, 10

Continued on next page

Table A.1 – continued from previous page

#	Chart Items	Description
13.	$\langle b[\text{Det}^{all}] \rightarrow \text{all}\bullet, 0, -, -, 1\rangle$	scan, 12
14.	$\langle t[\text{Det}^{all}] \rightarrow b[\text{Det}^{all}]\bullet, 0, -, -, 1\rangle$	3(b)-comp, (10, 13)
15.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{all}] \rightarrow t[\text{Det}^{all}]\ \bullet\ t[\text{NP}^{mk}\ \text{NP}_f^{all}], 0, -, -, 1\rangle$	1/2-comp, (8, 14)
16.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}] \rightarrow t[\text{Det}^{all}]\ \bullet\ t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}], 0, -, -, 1\rangle$	1/2-comp, (11, 14)
17.	$\langle t[\text{NP}^{mk}\ \text{NP}_f^{all}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_f^{all}], 1, -, -, 1\rangle$	3(b)-pred, 15
18.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}], 1, -, -, 1\rangle$	3(b)-pred, 16
19.	$\langle b[\text{NP}^{mk}\ \text{NP}_f^{all}] \rightarrow \bullet b[\text{NP}^{mk}], 1, -, -, 1\rangle$	5-pred, 17
20.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_r^{the}], 1, -, -, 1\rangle$	5-pred, 18
21.	$\langle b[\text{NP}^{mk}] \rightarrow \bullet t[\text{NP}^{mk}\ \text{NP}_r^{the}], 1, -, -, 1\rangle$	4-pred, 19
22.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow \bullet t[\text{Det}^{the}]\ t[\text{NP}^{mk}\ \text{NP}_f^{the}], 1, -, -, 1\rangle$	1/2-pred, 20
23.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_r^{the}], 1, -, -, 1\rangle$	3(a)-pred, 21
24.	$\langle t[\text{Det}^{the}] \rightarrow \bullet b[\text{Det}^{the}], 1, -, -, 1\rangle$	3(b)-pred, 22
25.	$\langle b[\text{Det}^{all}] \rightarrow \bullet\text{the}, 1, -, -, 1\rangle$	1/2-pred, 24
26.	$\langle b[\text{Det}^{the}] \rightarrow \text{the}\bullet, 1, -, -, 2\rangle$	scan, 25
27.	$\langle t[\text{Det}^{the}] \rightarrow b[\text{Det}^{the}]\bullet, 1, -, -, 2\rangle$	3(b)-comp, (24, 26)
28.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow t[\text{Det}^{the}]\ \bullet\ t[\text{NP}^{mk}\ \text{NP}_f^{the}], 1, -, -, 2\rangle$	1/2-comp, (22, 27)
29.	$\langle t[\text{NP}^{mk}\ \text{NP}_f^{the}] \rightarrow \bullet b[\text{NP}^{mk}\ \text{NP}_f^{the}], 2, -, -, 2\rangle$	3(b)-pred, 28
30.	$\langle b[\text{NP}^{mk}\ \text{NP}_f^{the}] \rightarrow \bullet b[\text{NP}^{mk}], 2, -, -, 2\rangle$	5-pred, 29
31.	$\langle b[\text{NP}^{mk}] \rightarrow \bullet\text{meerkat}, 2, -, -, 2\rangle$	1/2-pred, 30
32.	$\langle b[\text{NP}^{mk}] \rightarrow \text{meerkat}\bullet, 2, -, -, 3\rangle$	scan, 31
33.	$\langle b[\text{NP}^{mk}\ \text{NP}_f^{the}] \rightarrow b[\text{NP}^{mk}]\bullet, 2, 2, 3, 3\rangle$	5-comp, (30, 32)
34.	$\langle t[\text{NP}^{mk}\ \text{NP}_f^{the}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_f^{the}]\bullet, 2, 2, 3, 3\rangle$	3(b)-comp, (29, 33)
35.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow t[\text{Det}^{the}]\ t[\text{NP}^{mk}\ \text{NP}_f^{the}]\bullet, 1, 2, 3, 3\rangle$	1/2-comp, (28, 34)
36.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{the}]\bullet, 1, 2, 3, 3\rangle$	3(a)-comp, (23, 35)
37.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{the}]\bullet, 1, 1, 3, 3\rangle$	5-comp, (20, 35)
38.	$\langle b[\text{NP}^{mk}] \rightarrow t[\text{NP}^{mk}\ \text{NP}_r^{the}]\bullet, 1, -, -, 3\rangle$	4-comp, (21, 36, 32)
39.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}]\bullet, 1, 1, 3, 3\rangle$	3(b)-comp, (18, 37)
40.	$\langle b[\text{NP}^{mk}\ \text{NP}_f^{all}] \rightarrow b[\text{NP}^{mk}]\bullet, 1, 1, 3, 3\rangle$	5-comp, (19, 38)
41.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}] \rightarrow t[\text{Det}^{all}]\ t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_f^{all}]\bullet, 0, 1, 3, 3\rangle$	1/2-comp, (16, 39)
42.	$\langle t[\text{NP}^{mk}\ \text{NP}_f^{all}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_f^{all}]\bullet, 1, 1, 3, 3\rangle$	3(b)-comp, (17, 40)
43.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}]\bullet, 0, 1, 3, 3\rangle$	3(a)-comp, (9, 41)
44.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{all}] \rightarrow t[\text{Det}^{all}]\ t[\text{NP}^{mk}\ \text{NP}_f^{all}]\bullet, 0, 1, 3, 3\rangle$	1/2-comp, (15, 42)
45.	$\langle b[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow t[\text{NP}^{mk}\ \text{NP}_r^{the}\ \text{NP}_r^{all}]\bullet, 0, 2, 3, 3\rangle$	4-comp, (7, 43, 35)
46.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{all}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{all}]\bullet, 0, 1, 3, 3\rangle$	3(a)-comp, (6, 44)
47.	$\langle t[\text{NP}^{mk}\ \text{NP}_r^{the}] \rightarrow b[\text{NP}^{mk}\ \text{NP}_r^{the}]\bullet, 0, 2, 3, 3\rangle$	3(a)-comp, (5, 45)
48.	$\langle b[\text{NP}^{mk}] \rightarrow t[\text{NP}^{mk}\ \text{NP}_r^{all}]\bullet, 0, -, -, 3\rangle$	4-comp, (4, 46, 38)
49.	$\langle b[\text{NP}^{mk}] \rightarrow t[\text{NP}^{mk}\ \text{NP}_r^{the}]\bullet, 0, -, -, 3\rangle$	4-comp, (3, 47, 32)
50.	$\langle t[\text{NP}^{mk}] \rightarrow b[\text{NP}^{mk}]\bullet, 0, -, -, 3\rangle$	3(b)-comp, (2, 48)
51.	$\langle t[\text{NP}^{mk}] \rightarrow b[\text{NP}^{mk}]\bullet, 0, -, -, 3\rangle$	3(b)-comp, (2, 49)
52.	$\langle S \rightarrow t[\text{NP}^{mk}]\bullet, 0, -, -, 3\rangle$	axiom-comp, (1, 50)
53.	$\langle S \rightarrow t[\text{NP}^{mk}]\bullet, 0, -, -, 3\rangle$	axiom-comp, (1, 51)

Type 1 and Type 2 production rules

$$b[\mathrm{NP}^{mk}] \rightarrow \text{meerkat}$$
$$b[..\ \mathrm{NP}_r^{the}] \rightarrow t[\mathrm{Det}^{the}] \quad t[..\ \mathrm{NP}_f^{the}]$$
$$b[\mathrm{Det}^{the}] \rightarrow \text{the}$$
$$b[..\ \mathrm{NP}_r^{all}] \rightarrow t[\mathrm{Det}^{all}] \quad t[..\ \mathrm{NP}_f^{all}]$$
$$b[\mathrm{Det}^{all}] \rightarrow \text{all}$$

Type 3(a) production rules

$$t[..\ \mathrm{NP}_r^{the}] \rightarrow b[..\ \mathrm{NP}_r^{the}]$$
$$t[..\ \mathrm{NP}_r^{all}] \rightarrow b[..\ \mathrm{NP}_r^{all}]$$

Type 3(b) production rules

$$t[..\ \mathrm{NP}^{mk}] \rightarrow b[..\ \mathrm{NP}^{mk}]$$
$$t[..\ \mathrm{Det}^{the}] \rightarrow b[..\ \mathrm{Det}^{the}]$$
$$t[..\ \mathrm{NP}_f^{the}] \rightarrow b[..\ \mathrm{NP}_f^{the}]$$

Type 3(b) production rules (continued)

$$t[..\ \mathrm{Det}^{all}] \rightarrow b[..\ \mathrm{Det}^{all}]$$
$$t[..\ \mathrm{NP}_f^{all}] \rightarrow b[..\ \mathrm{NP}_f^{all}]$$

Type 4 production rules

$$b[..\ \mathrm{NP}^{mk}] \rightarrow t[..\ \mathrm{NP}^{mk}\ \mathrm{NP}_r^{the}]$$
$$b[..\ \mathrm{NP}^{mk}] \rightarrow t[..\ \mathrm{NP}^{mk}\ \mathrm{NP}_r^{all}]$$
$$b[..\ \mathrm{NP}_r^{the}] \rightarrow t[..\ \mathrm{NP}_r^{the}\ \mathrm{NP}_r^{all}]$$
$$b[..\ \mathrm{NP}_r^{all}] \rightarrow t[..\ \mathrm{NP}_r^{all}\ \mathrm{NP}_r^{the}]$$

Type 5 production rules

$$b[..\ \mathrm{NP}^{mk}\ \mathrm{NP}_f^{the}] \rightarrow b[..\ \mathrm{NP}^{mk}]$$
$$b[..\ \mathrm{NP}_r^{all}\ \mathrm{NP}_f^{the}] \rightarrow b[..\ \mathrm{NP}_r^{all}]$$
$$b[..\ \mathrm{NP}^{mk}\ \mathrm{NP}_f^{all}] \rightarrow b[..\ \mathrm{NP}^{mk}]$$
$$b[..\ \mathrm{NP}_r^{the}\ \mathrm{NP}_f^{all}] \rightarrow b[..\ \mathrm{NP}_r^{the}]$$

Figure A.18: LIG production rules for the TAG shown in Figure A.17.

A.7 Conclusion

While independent derivations have been shown by Schabes and Shieber (1994) to be essential for correctly supporting syntactic analysis, semantic interpretation and statistical language modelling, the parsing algorithm they propose is restricted to TAG and is therefore not directly applicable to large scale implemented Feature-Based TAGs. We have provided a recognition algorithm for FB-TAG which supports both dependent and independent derivations under certain restrictions enforced jointly by feature constraints and by side conditions in the parsing algorithm. The resulting algorithm combines the benefits of independent derivations with those of Feature-Based grammars. In particular, we showed that it accounts for a range of interactions between dependent vs. independent derivation on the one hand, and syntactic constraints, linear ordering, and scopal vs. nonscopal semantic dependencies on the other hand.

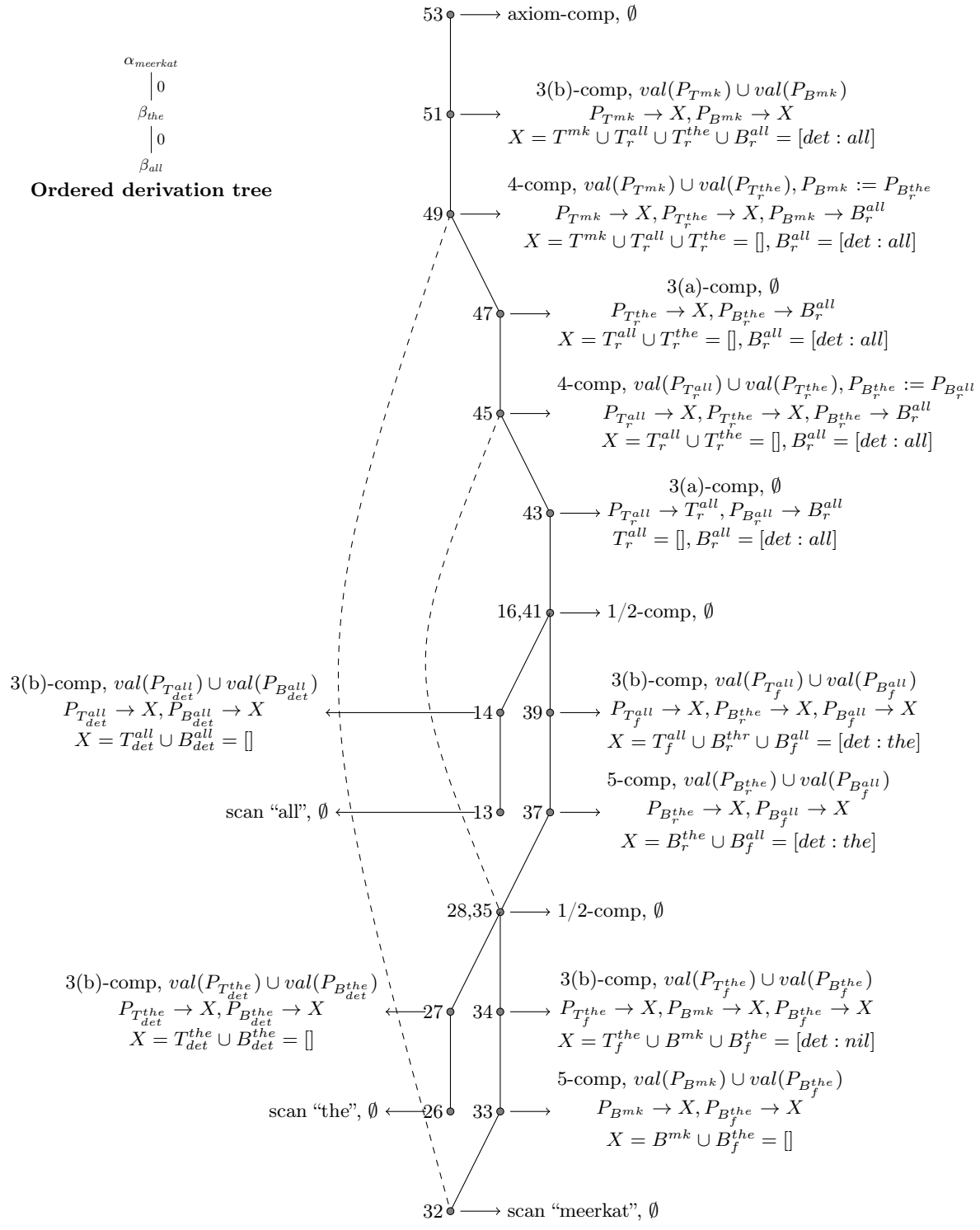


Figure A.19: Feature unifications in dependent derivation of “all the meerkats” (prediction rules are abbreviated).

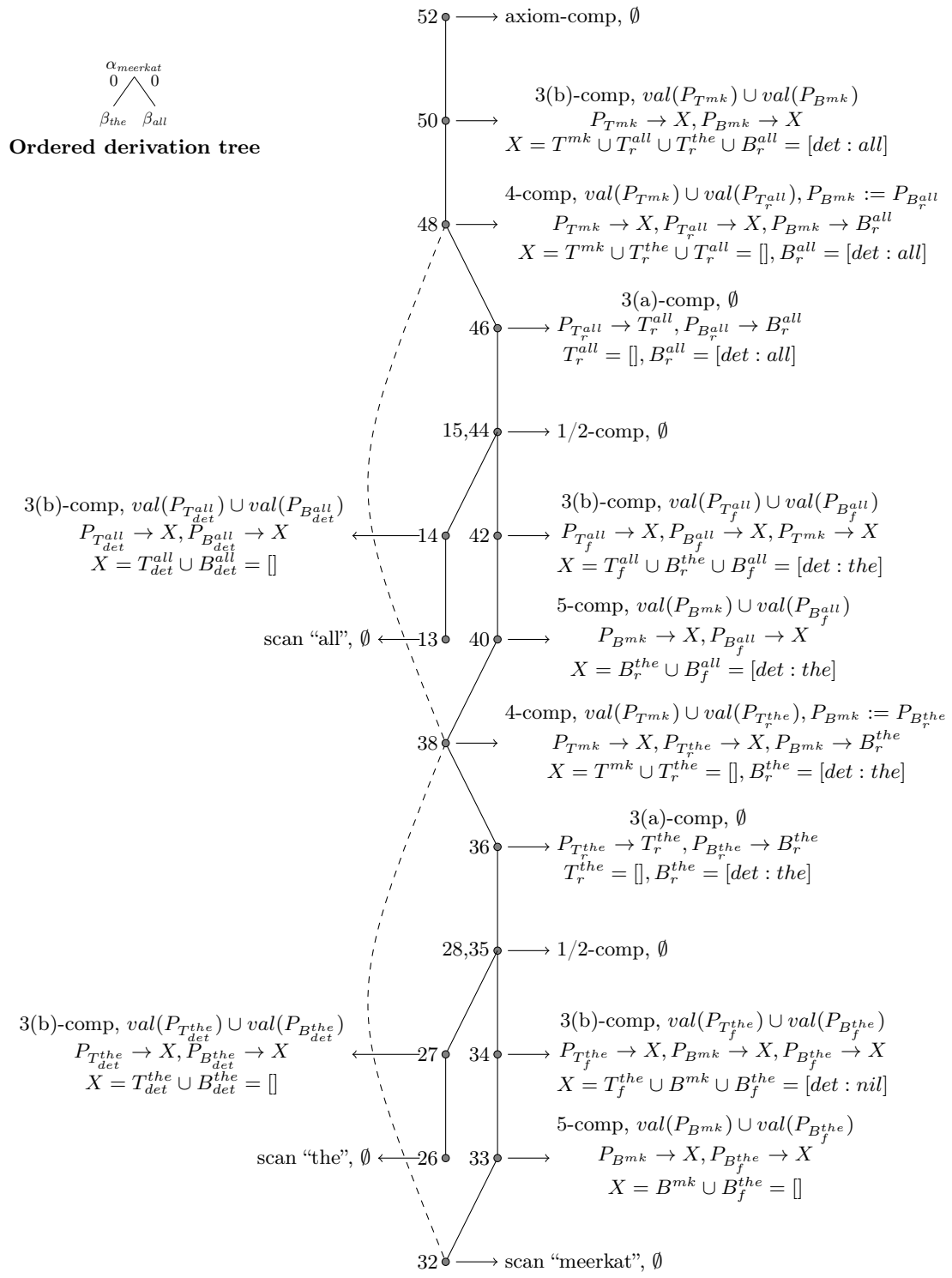


Figure A.20: Feature unifications in independent derivation of “all the meerkats” (prediction rules are abbreviated).

Bibliography

[Alahverdzhieva, 2008] Katya Alahverdzhieva. XTAG using XMG. Master's thesis, Université de Nancy, 2008.

[Aue *et al.*, 2004] Anthony Aue, Arul Menezes, Robert Moore, Chris Quirk, and Eric Ringger. Statistical machine translation using labeled semantic dependency graphs. ACL/SIGPARSE, October 2004.

[Banchs and Costa-jussà, 2011] Rafael E. Banchs and Marta R. Costa-jussà. A semantic feature for statistical machine translation. In *Proceedings of the Fifth Workshop on Syntax, Semantics and Structure in Statistical Translation*, SSST-5, pages 126–134, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[Bangalore and Joshi, 1999] Srinivas Bangalore and Aravind K. Joshi. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265, 1999.

[Bangalore and Rambow, 2000a] Srinivas Bangalore and Owen Rambow. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th Conference on Computational Linguistics - Volume 1*, COLING '00, pages 42–48, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[Bangalore and Rambow, 2000b] Srinivas Bangalore and Owen Rambow. Using TAGs, a tree model and a language model for generation. In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, Paris, France, 2000.

[Banik, 2004] Eva Banik. Semantics of VP coordination in LTAG. In *Proceedings of the 7th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+)*, volume 7, pages 118–125, Vancouver, Canada, 2004.

[Bannard and Callison-Burch, 2005] Colin Bannard and Chris Callison-Burch. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting*

on Association for Computational Linguistics (ACL), pages 597–604. Association for Computational Linguistics, 2005.

[Barzilay and Elhadad, 2003] Regina Barzilay and Noemie Elhadad. Sentence alignment for monolingual comparable corpora. In *Proceedings of the 2003 conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 25–32. Association for Computational Linguistics, 2003.

[Basile and Bos, 2013] Valerio Basile and Johan Bos. *Proceedings of the 14th European Workshop on Natural Language Generation*, chapter Aligning Formal Meaning Representations with Surface Strings for Wide-Coverage Text Generation, pages 1–9. Association for Computational Linguistics, 2013.

[Basile *et al.*, 2012] Valerio Basile, Johan Bos, Kilian Evang, and Noortje Venhuizen. Developing a large semantically annotated corpus. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, pages 3196–3200, Istanbul, Turkey, 2012.

[Bateman, 1997] John A. Bateman. Enabling technology for multilingual natural language generation: the KPML development environment. *Natural Language Engineering*, 3:15–55, 3 1997.

[Beigman Klebanov *et al.*, 2004] Beata Beigman Klebanov, Kevin Knight, and Daniel Marcu. Text simplification for information-seeking applications. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3290 of *Lecture Notes in Computer Science*, pages 735–747. Springer Berlin Heidelberg, 2004.

[Belz *et al.*, 2011] Anja Belz, Michael White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. The first surface realisation shared task: Overview and evaluation results. In *Proceedings of the 13th European Workshop on Natural Language Generation (ENLG)*, Nancy, France, 2011.

[Bies *et al.*, 1995] Ann Bies, Mark Ferguson, Katz Katz, Robert MacIntyre, Victoria Tredinnick, Grace Kim, Marry Ann Marcinkiewicz, and Britta Schasberger. Bracketing guidelines for treebank II style penn treebank project. *University of Pennsylvania*, 1995.

[Biran *et al.*, 2011] Or Biran, Samuel Brody, and Noémie Elhadad. Putting it simply: a context-aware approach to lexical simplification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human*

Language Technologies: short papers-Volume 2, pages 496–501. Association for Computational Linguistics, 2011.

[Bonfante *et al.*, 2004] Guillaume Bonfante, Bruno Guillaume, and Guy Perrier. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, Geneva, Switzerland, 2004.

[Bott *et al.*, 2012] Stefan Bott, Horacio Saggion, and Simon Mille. Text simplification tools for spanish. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC)*, pages 1665–1671, 2012.

[Boyd *et al.*, 2008] Adriane Boyd, Markus Dickinson, and Detmar Meurers. On detecting errors in dependency treebanks. *Research on Language and Computation*, 6(2):113–137, 2008.

[Butt *et al.*, 2002] Miriam Butt, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. The parallel grammar project. In *Proceedings of the Workshop on Grammar Engineering and Evaluation - Volume 15, International Conference on Computational Linguistics (COLING)*, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[Cahill and Van Genabith, 2006] Aoife Cahill and Josef Van Genabith. Robust pcfg-based generation using automatically acquired lfg approximations. In *Proceedings of the 21st International Conference on Computational Linguistics (COLING) and the 44th annual meeting of the Association for Computational Linguistics (ACL)*, pages 1033–1040, Sydney, Australia, 2006.

[Callaway, 2003] Charles B. Callaway. Evaluating coverage for large symbolic NLG grammars. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 811–817, Acapulco, Mexico, 2003.

[Candito and Kahane, 1998] Marie-Helene Candito and Sylvain Kahane. Can the tag derivation tree represent a semantic graph? an answer in the light of meaning-text theory. In *Proceedings of the Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks*. Citeseer, 1998.

[Canning, 2002] Yvonne Margaret Canning. *Syntactic simplification of Text*. PhD thesis, University of Sunderland, 2002.

[Carroll and Oepen, 2005] John Carroll and Stephan Oepen. High efficiency realization for a wide-coverage unification grammar. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP)*, Jeju Island, Korea, 2005.

[Carroll *et al.*, 1999a] John Carroll, Ann Copestake, Dan Flickinger, and Viktor Paznański. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation*, pages 86–95, Toulouse, France, 1999.

[Carroll *et al.*, 1999b] John Carroll, Guido Minnen, Darren Pearce, Yvonne Canning, Siobhan Devlin, and John Tait. Simplifying text for language-impaired readers. In *Proceedings of 9th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, volume 99, pages 269–270. Citeseer, 1999.

[Chandrasekar and Srinivas, 1997] Raman Chandrasekar and Bangalore Srinivas. Automatic induction of rules for text simplification. *Knowledge-Based Systems*, 10(3):183–190, 1997.

[Chandrasekar *et al.*, 1996] Raman Chandrasekar, Christine Doran, and Bangalore Srinivas. Motivations and methods for text simplification. In *Proceedings of the 16th International conference on Computational linguistics (COLING)*, pages 1041–1044. Association for Computational Linguistics, 1996.

[Charniak *et al.*, 2003] Eugene Charniak, Kevin Knight, and Kenji Yamada. Syntax-based language models for statistical machine translation. In *MT Summit IX. International Association for Machine Translation*, 2003.

[Charniak, 2001] Eugene Charniak. Immediate-head parsing for language models. In *In Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 116–123, 2001.

[Chi *et al.*, 2004] Yun Chi, Yirong Yang, and Richard R. Muntz. Hybridtreeminer: An efficient algorithm for mining frequent rooted trees and free trees using canonical form. In *Proceedings of the 16th International Conference on and Statistical Database Management (SSDBM)*, pages 11–20, Santorini Island, Greece, 2004. IEEE Computer Society.

[Čmejrek *et al.*, 2004] Martin Čmejrek, Jan Cuřín, Jirí Havelka, Jan Hajič, and Vladislav Kuboň. Prague czech-english dependency treebank. syntactically annotated resources for machine translation. In *Proceedings of the 4th International*

Conference on Language Resources and Evaluation (LREC), Lisbon, Portugal, 2004.

[Collins, 1999] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[Copestake *et al.*, 2001] Ann Copestake, Alex Lascarides, and Dan Flickinger. An algebra for semantic construction in constraint-based grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, Toulouse, France, 2001.

[Coster and Kauchak, 2011] William Coster and David Kauchak. Learning to simplify sentences using wikipedia. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pages 1–9. Association for Computational Linguistics, 2011.

[Crabbé *et al.*, 2013] Benoît Crabbé, Denys Duchier, Claire Gardent, Joseph Le Roux, and Yannick Parmentier. XMG : eXtensible MetaGrammar. *Computational Linguistics*, 39(3):1–66, September 2013.

[Culicover and Jackendoff, 2005] Peter W. Culicover and Ray Jackendoff. *Simpler Syntax*. Oxford University Press, 2005.

[Curran *et al.*, 2007] James R Curran, Stephen Clark, and Johan Bos. Linguistically motivated large-scale NLP with C&C and Boxer. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL) on Interactive Poster and Demonstration Sessions*, pages 33–36. Association for Computational Linguistics, 2007.

[Daelemans *et al.*, 2004] Walter Daelemans, A. Hoethker, and E. Tjong Kim Sang. *Automatic sentence simplification for subtitling in Dutch and English*, pages 1045–1048. Lisbon, 2004.

[Dalianis and Hovy, 1996] Hercules Dalianis and Eduard Hovy. Aggregation in natural language generation. *Trends in Natural Language Generation: An Artificial Intelligence Perspective*, pages 88–105, 1996.

[Dalrymple and Kaplan, 2000] Mary Dalrymple and Ronald M. Kaplan. Feature indeterminacy and feature resolution. *Language*, pages 759–798, 2000.

[Dalrymple *et al.*, 1991] Mary Dalrymple, Stuart M. Sheiber, and Fernando C. N. Pereira. Ellipsis and higher-order unification. *Linguistics and Philosophy*, 1991.

[Daum *et al.*, 2004] Michael Daum, Kilian Foth, and Wolfgang Menzel. Automatic transformation of phrase treebanks to dependency trees. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC)*, Lisbon, Portugal, 2004.

[de Kok *et al.*, 2009] Daniël de Kok, Jianqiang Ma, and Gertjan van Noord. A generalized method for iterative error mining in parsing results. In *Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks (GEAF 2009)*, pages 71–79, Suntec, Singapore, 2009. Association for Computational Linguistics.

[de Marneffe *et al.*, 2006] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *Proceedings of the internation conference on Language Resources and Evaluation (LREC*, pages 449–454, 2006.

[Dempster *et al.*, 1977] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[Dickinson and Smith, 2011] Markus Dickinson and Amber Smith. Detecting dependency parse errors with minimal resources. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT 2011)*, Dublin, Ireland, 2011.

[Dras, 1999] Mark Dras. *Tree adjoining grammar and the reluctant paraphrasing of text*. PhD thesis, Macquarie University NSW 2109 Australia, 1999.

[Dukes and Habash, 2011] Kais Dukes and Nizar Habash. One-step statistical parsing of hybrid dependency-constituency syntactic representations. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 92–103, Dublin, Ireland, 2011. Association for Computational Linguistics.

[Eisner, 2003] Jason Eisner. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 2*, ACL '03, pages 205–208, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[Elhadad and Robin, 1996] Michael Elhadad and Jacques Robin. An overview of SURGE: a reusable comprehensive syntactic realization component. In *Proceedings of the eighth International Natural Language Generation Workshop (INLG)*, pages 1–4, 1996.

[Elhadad, 1993a] Michael Elhadad. FUF: The universal unifier — User Manual, version 5.2. Technical report, Ben Gurion University of the Negev, 1993.

[Elhadad, 1993b] Michael Elhadad. *Using argumentation to control lexical choice: a functional unification implementation.* PhD thesis, Columbia University, 1993.

[Espinosa *et al.*, 2008] Dominic Espinosa, Michael White, and Dennis Mehay. Hypertagging: Supertagging for surface realization with CCG. In *Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL): Human Language Technologies (HLT)*, pages 183–191, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[Espinosa *et al.*, 2010] Dominic Espinosa, Rajakrishnan Rajkumar, Michael White, and Shoshana Berleant. Further meta-evaluation of broad-coverage surface realization. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 564–574, Cambridge, MA, October 2010. Association for Computational Linguistics.

[Filippova and Strube, 2008] Katja Filippova and Michael Strube. Dependency tree based sentence compression. In *Proceedings of the Fifth International Natural Language Generation Conference (INLG)*, pages 25–32, Salt Fork, Ohio, USA, 2008. Association for Computational Linguistics.

[Ganitkevitch *et al.*, 2011] Juri Ganitkevitch, Chris Callison-Burch, Courtney Napoles, and Benjamin Van Durme. Learning sentential paraphrases from bilingual parallel corpora for text-to-text generation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1168–1179, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

[Ganitkevitch *et al.*, 2013] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. Ppdb: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[Gardent and Kallmeyer, 2003] Claire Gardent and Laura Kallmeyer. Semantic construction in ftag. In *Proceedings of the 10th meeting of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary, 2003.

[Gardent and Kow, 2005] Claire Gardent and Eric Kow. Generating and selecting grammatical paraphrases. In *Proceedings of the 10th European Workshop on Natural Language Generation (ENLG)*, Aberdeen, Scotland, Aug 2005.

[Gardent and Kow, 2007a] Claire Gardent and Eric Kow. Spotting overgeneration suspect. In *Proceedings of the 11th European Workshop on Natural Language Generation (ENLG)*, pages 41–48, Schloss Dagstuhl, Germany, 2007.

[Gardent and Kow, 2007b] Claire Gardent and Eric Kow. A symbolic approach to near-deterministic surface realisation using tree adjoining grammar. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 328–335, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[Gardent and Narayan, 2012] Claire Gardent and Shashi Narayan. Error mining on dependency trees. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 592–600, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[Gardent and Narayan, 2013] Claire Gardent and Shashi Narayan. Generating elliptic coordination. In *Proceedings of the 14th European Workshop on Natural Language Generation (ENLG)*, pages 40–50, Sofia Bulgaria, August 2013. Association for Computational Linguistics.

[Gardent and Perez-Beltrachini, 2010] Claire Gardent and Laura Perez-Beltrachini. RTG based surface realisation for TAG. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*, pages 367–375, Beijing, China, 2010.

[Gardent and Perez-Beltrachini, 2012] Claire Gardent and Laura Perez-Beltrachini. Using FB-LTAG Derivation Trees to Generate Transformation-Based Grammar Exercises. In *TAG+11: The 11th International Workshop on Tree Adjoining Grammars and Related Formalisms*, Paris, France, September 2012.

[Gardent *et al.*, 2010] Claire Gardent, Benjamin Gottesman, and Laura Perez-Beltrachini. Comparing the performance of two TAG-based Surface Realisers using controlled Grammar Traversal. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING - Poster session)*, pages 338–346, Beijing, China, 2010.

[Gardent *et al.*, 2011] Claire Gardent, Benjamin Gottesman, and Laura Perez-Beltrachini. Using regular tree grammars to enhance sentence realisation. *Natural Language Engineering (NLE), Issue 2*, 17, 2011.

[Gardent, 2008] Claire Gardent. Integrating a unification-based semantics in a large scale lexicalised tree adjoining grammar for french. In *Proceedings of the 22Nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 249–256, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[Gazdar, 1988] Gerald Gazdar. Applicability of indexed grammars to natural languages. In Uwe Reyle and Christian Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, volume 35 of *Studies in Linguistics and Philosophy*, pages 69–94. Springer Netherlands, 1988.

[Geiß*et al.*, 2006] Rubino Geiß, Gernot Veit Batz, Daniel Grund, Sebastian Hack, and Adam M. Szalkowski. GrGen: A Fast SPO-Based Graph Rewriting Tool. In *Proceedings of the 3rd International Conference on Graph Transformation*, pages 383–397. Springer, 2006. Natal, Brasil.

[Ginzburg and Sag, 2000] Jonathan Ginzburg and Ivan Sag. *Interrogative investigations*. CSLI Publications, 2000.

[Hajič *et al.*, 2009] Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria A. Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the 13th Conference on Computational Natural Language Learning: Shared Task*, pages 1–18, 2009.

[Harbusch and Kempen, 2009] Karin Harbusch and Gerard Kempen. Generating clausal coordinate ellipsis multilingually: A uniform approach based on postediting. In *Proceedings of the 12th European Workshop on Natural Language Generation*, pages 138–145, Athens, Greece, 2009. Association for Computational Linguistics.

[Hardt, 1993] Daniel Hardt. *Verb phrase ellipsis: Form, meaning and processing*. PhD thesis, University of Pennsylvania, 1993.

[Heilman and Smith, 2009] Michael Heilman and Noah A. Smith. Question generation via overgenerating transformations and ranking, technical report cmu-lti-09-

013. Technical report, Language Technologies Institute, Carnegie Mellon University, 2009.

[Inui *et al.*, 2003] Kentaro Inui, Atsushi Fujita, Tetsuro Takahashi, Ryu Iida, and Tomoya Iwakura. Text simplification for reading assistance: A project note. In *Proceedings of the Second International Workshop on Paraphrasing*, pages 9–16, Sapporo, Japan, July 2003. Association for Computational Linguistics.

[Johansson and Nugues, 2007] Richert Johansson and Pierre Nugues. Extended constituent-to-dependency conversion for english. In *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA)*, pages 105–112, Tartu, Estonia, 2007.

[Jones *et al.*, 2012] Bevan Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, and Kevin Knight. Semantics-based machine translation with hyperedge replacement grammars. In *Proceedings of COLING 2012*, pages 1359–1376, Mumbai, India, December 2012. The COLING 2012 Organizing Committee.

[Jonnalagadda and Gonzalez, 2009] Siddhartha Jonnalagadda and Graciela Gonzalez. Sentence simplification aids protein-protein interaction extraction. In *Proceedings of the 3rd International Symposium on Languages in Biology and Medicine (LBM'09)*, pages 8–10, November 2009. informal publication.

[Joshi and Schabes, 1997] Aravind K. Joshi and Yves Schabes. Handbook of formal languages, vol. 3. chapter Tree-adjoining Grammars, pages 69–123. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[Joshi and Vijay-Shanker, 2001] Aravind K Joshi and K Vijay-Shanker. Compositional semantics with lexicalized tree-adjoining grammar (LTAG): How much underspecification is necessary? In *Computing Meaning*, pages 147–163. Springer, 2001.

[Joshi *et al.*, 1975] Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136 – 163, 1975.

[Joshi, 1985] Aravind K. Joshi. Natural language parsing. chapter Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions?, pages 206–250. Cambridge University Press., 1985.

[Joshi, 2000] Aravind K. Joshi. Relationship between strong and weak generative power of formal systems. In *In Proceedings of the Fifth International Workshop*

on Tree Adjoining Grammars and Related Formalisms (TAG+5, pages 107–114, 2000.

[Kallmeyer and Kuhlmann, 2012] Laura Kallmeyer and Marco Kuhlmann. A formal model for plausible dependencies in lexicalized tree adjoining grammar. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+)*, pages 108–116, Paris, France, 2012.

[Kamp, 1981] Hans Kamp. A theory of truth and semantic representation. In J.A.G. Groenendijk, T.M.V. Janssen, B.J. Stokhof, and M.J.B. Stokhof, editors, *Formal methods in the study of language*, number pt. 1 in Mathematical Centre tracts. Mathematisch Centrum, 1981.

[Kay, 1996] Martin Kay. Chart generation. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 200–204, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.

[Keenan, 1971] Edward Keenan. Names, quantifiers, and the sloppy identity problem. *Papers in Linguistics*, 4:211–232, 1971.

[Kehler, 2002] Andrew Kehler. *Coherence in discourse*. CSLI Publications, 2002.

[Knight and Marcu, 2000] Kevin Knight and Daniel Marcu. Statistics-based summarization-step one: Sentence compression. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI) and Twelfth Conference on Innovative Applications of Artificial Intelligence (IAAI)*, pages 703–710. AAAI Press, 2000.

[Koehn *et al.*, 2007] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[Koller and Striegnitz, 2002] Alexander Koller and Kristina Striegnitz. Generation as dependency parsing. In *Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL)*, Philadelphia, 2002.

[Konstas and Lapata, 2012a] Ioannis Konstas and Mirella Lapata. Concept-to-text generation via discriminative reranking. In *Proceedings of the 50th Annual Meeting*

of the Association for Computational Linguistics: Long Papers - Volume 1, ACL '12, pages 369–378, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[Konstas and Lapata, 2012b] Ioannis Konstas and Mirella Lapata. Unsupervised concept-to-text generation with hypergraphs. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pages 752–761, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[Kow and Belz, 2012] Eric Kow and Anja Belz. LG-Eval: A Toolkit for Creating Online Language Evaluation Experiments. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC)*, pages 4033–4037, 2012.

[Kroch, 1989] Anthony Kroch. Asymmetries in long distance extraction in a tree adjoining grammar. *Alternative conceptions of phrase structure*, pages 66–98, 1989.

[Kubota and Levine, 2012] Yusuke Kubota and Robert Levine. Gapping as like-category coordination. In *Proceedings of the 7th international conference on Logical Aspects of Computational Linguistics (LACL)*, pages 135–150, Nantes, France, 2012. Springer-Verlag.

[Langkilde and Knight, 1998] Irene Langkilde and Kevin Knight. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL '98, pages 704–710, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics.

[Langkilde, 2002] Irene Langkilde. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the 12th International Natural Language Generation Workshop*, pages 17–24, 2002.

[Lavoie and Rambow, 1997] Benoit Lavoie and Owen Rambow. A fast and portable realizer for text generation systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)*, pages 265–268, Washington, 1997.

[Levy and Pollard, 2001] Roger Levy and Carl Pollard. Coordination and neutralization in HPSG. *Technology*, 3:5, 2001.

[Li *et al.*, 2014] Junhui Li, Yuval Marton, Philip Resnik, and Hal Daumé III. A unified model for soft linguistic reordering constraints in statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1123–1133, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[Maier *et al.*, 2012] Wolfgang Maier, Erhard Hinrichs, Julia Krivanek, and Sandra Kübler. Annotating coordination in the Penn Treebank. In *Proceedings of the 6th Linguistic Annotation Workshop (LAW)*, pages 166–174, Jeju, Republic of Korea, 2012. Association for Computational Linguistics.

[Manning *et al.*, 2014] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[Marcus *et al.*, 1993] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, June 1993.

[McKeown *et al.*, 1994] Kathleen McKeown, Karen Kukich, and James Shaw. Practical issues in automatic documentation generation. In *Proceedings of the fourth conference on Applied natural language processing (ANLC)*, pages 7–14, Stuttgart, Germany, 1994. Association for Computational Linguistics.

[Merchant, 2001] Jason Merchant. *The syntax of silence: Sluicing, islands, and the theory of ellipsis*. Oxford University Press, 2001.

[Meyers *et al.*, 2004] Adam Meyers, Ruth Reeves, Catherine Macleod, Rachel Szekely, Veronika Zielinska, Brian Young, and Ralph Grishman. The nombank project: An interim report. In *Proceedings of the NAACL/HLT Workshop on Frontiers in Corpus Annotation*, 2004.

[Moortgat, 1997] Michael Moortgat. Categorial type logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 93–177. Elsevier, 1997.

[Nakanishi and Miyao, 2005] Hiroko Nakanishi and Yusuke Miyao. Probabilistic models for disambiguation of an hpsg-based chart generator. In *In Proceedings*

of the 9th International Workshop on Parsing Technologies (pp. 93 – 102, pages 93–102, 2005.

[Narayan and Gardent, 2012a] Shashi Narayan and Claire Gardent. Error mining with suspicion trees: Seeing the forest for the trees. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING)*, pages 2011–2026, Mumbai, India, December 2012. The COLING 2012 Organizing Committee.

[Narayan and Gardent, 2012b] Shashi Narayan and Claire Gardent. Structure-driven lexicalist generation. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING)*, pages 2027–2042, Mumbai, India, December 2012. The COLING 2012 Organizing Committee.

[Narayan and Gardent, 2014] Shashi Narayan and Claire Gardent. Hybrid simplification using deep semantics and machine translation. In *Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Baltimore, June 2014. Association for Computational Linguistics.

[Narayan, 2011] Shashi Narayan. RTG based surface realisation from dependency representations. Master's thesis, University of Nancy 2 and University of Malta, 2011. Erasmus Mundus Master "Language and Communication Technology".

[Nelken and Shieber, 2006] Rani Nelken and Stuart M. Shieber. Towards robust context-sensitive sentence alignment for monolingual corpora. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 161–166, 2006.

[Palmer *et al.*, 2005] Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, March 2005.

[Perez-Beltrachini *et al.*, 2012] Laura Perez-Beltrachini, Claire Gardent, and German Kruszewski. Generating grammar exercises. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*, pages 147–156, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[Perez-Beltrachini *et al.*, 2014] Laura Perez-Beltrachini, Claire Gardent, and Enrico Franconi. Incremental query generation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Gothenburg, Sweden, April 2014.

[Perez-Beltrachini, 2013] Laura Perez-Beltrachini. *Surface Realisation and Computer Aided Language Learning*. PhD thesis, Université de Lorraine, Nancy, France, 2013.

[Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, pages 81–106, March 1986.

[Rajkumar *et al.*, 2011] Rajakrishnan Rajkumar, Dominic Espinosa, and Michael White. The osu system for surface realization at generation challenges 2011. In *Proceedings of the 13th European Workshop on Natural Language Generation (ENLG)*, pages 236–238, Nancy, France, 2011.

[Rambow *et al.*, 1995] Owen Rambow, K. Vijay-Shanker, and David Weir. D-tree grammars. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 151–158, Cambridge, Massachusetts, USA, June 1995. Association for Computational Linguistics.

[Ratnaparkhi, 2000] Adwait Ratnaparkhi. Trainable methods for surface natural language generation. In *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, NAACL 2000, pages 194–201, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[Reiter and Dale, 2000] Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press, 2000.

[Resnik, 1992] Philip Resnik. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the 14th conference on Computational linguistics - Volume 2*, COLING '92, pages 418–424, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[Sag, 1976] Ivan Sag. *Deletion and logical form*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1976.

[Sagot and de la Clergerie, 2006] Benoît Sagot and Éric de la Clergerie. Error mining in parsing results. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 329–336, Sydney, Australia, 2006.

[Sarkar and Joshi, 1996] Anoop Sarkar and Aravind K. Joshi. Coordination in tree adjoining grammars: Formalization and implementation. In *Proceedings of the*

16th International Conference on Computational Linguistics (COLING), pages 610–615, 1996.

[Schabes and Joshi, 1988] Yves Schabes and Aravind K. Joshi. An Earley-type parsing algorithm for tree adjoining grammars. In *Proceedings of the 26th annual meeting on Association for Computational Linguistics*, ACL '88, pages 258–269, Stroudsburg, PA, USA, 1988. Association for Computational Linguistics.

[Schabes and Shieber, 1992] Yves Schabes and Stuart M. Shieber. An alternative conception of tree-adjoining derivation. In *Proceedings of the 30th annual meeting on Association for Computational Linguistics*, ACL '92, pages 167–176, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[Schabes and Shieber, 1994] Yves Schabes and Stuart M. Shieber. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124, mar 1994.

[Schabes, 1991] Yves Schabes. The valid prefix property and left to right parsing of tree-adjoining grammar. In *Proceedings of the 2nd International Workshop on Parsing Tenchnologies (IWPT)*, pages 21–30, Cancun, Mexico, 1991.

[Schabes, 1992] Yves Schabes. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th conference on Computational linguistics - Volume 2*, COLING '92, pages 425–432, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[Schmitz and Roux, 2008] Sylvain Schmitz and Joseph Le Roux. Feature unification in TAG derivation trees. In *Proceedings of the 9th International Workshop on Tree Adjoining Grammars and Related Formalisms*, Tubingen, Germany, 2008.

[Seddah, 2008] Djamé Seddah. The use of mctag to process elliptic coordination. In *Proceedings of The Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 9)*, volume 1, page 2, Tübingen, Germany, 2008.

[Seeker and Kuhn, 2012] Wolfgang Seeker and Jonas Kuhn. Making ellipses explicit in dependency conversion for a german treebank. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC)*, Istanbul, Turkey, 2012.

[Shardlow, 2014] Matthew Shardlow. A survey of automated text simplification. *International Journal of Advanced Computer Science and Applications (IJACSA), Special Issue on Natural Language Processing*, 2014.

[Shaw, 1998] James Shaw. Segregatory coordination and ellipsis in text generation. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 1220–1226, Montreal, Quebec, Canada, 1998.

[Shieber *et al.*, 1989] Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C. N. Pereira. A semantic-head-driven generation algorithm for unification-based formalisms. In *Proceedings of the 27th Annual Meeting on Association for Computational Linguistics*, ACL '89, pages 7–17, Stroudsburg, PA, USA, 1989. Association for Computational Linguistics.

[Shieber *et al.*, 1990] Stuart M. Shieber, Gertjan Van Noord, Fernando C. N. Pereira, and Robert C. Moore. Semantic-head-driven generation. *Computational Linguistics*, 16, 1990.

[Shieber, 1988] Stuart M. Shieber. A uniform architecture for parsing and generation. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 2*, COLING '88, pages 614–619, Stroudsburg, PA, USA, 1988. Association for Computational Linguistics.

[Shieber, 1993] Stuart M. Shieber. The problem of logical form equivalence. *Computational Linguistics*, 19(1):179–190, 1993.

[Shieber, 1994] Stuart M. Shieber. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385, 1994.

[Siddharthan and Mandya, 2014] Advaith Siddharthan and Angrosh Mandya. Hybrid text simplification using synchronous dependency grammars with hand-written and automatically harvested rules. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 722–731, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.

[Siddharthan, 2002] Advaith Siddharthan. An architecture for a text simplification system. In *Proceedings of the Language Engineering Conference (LEC)*, pages 64–71. IEEE Computer Society, 2002.

[Siddharthan, 2006] Advaith Siddharthan. Syntactic simplification and text cohesion. *Research on Language and Computation*, 4(1):77–109, 2006.

[Siddharthan, 2010] Advaith Siddharthan. Complex lexico-syntactic reformulation of sentences using typed dependency representations. In *Proceedings of the 6th International Natural Language Generation Conference (INLG)*, pages 125–133. Association for Computational Linguistics, 2010.

[Siddharthan, 2011] Advaith Siddharthan. Text simplification using typed dependencies: a comparison of the robustness of different generation strategies. In *Proceedings of the 13th European Workshop on Natural Language Generation (ENLG)*, pages 2–11. Association for Computational Linguistics, 2011.

[Smith and Eisner, 2006] David A Smith and Jason Eisner. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings of the HLT-NAACL Workshop on Statistical Machine Translation*, pages 23–30. Association for Computational Linguistics, 2006.

[Steedman, 1996] Mark Steedman. *Surface Structure and Interpretation*, volume 30. MIT press Cambridge, MA, 1996.

[Steedman, 2000] Mark Steedman. *The syntactic process*. MIT Press, Cambridge, MA, USA, 2000.

[Stolcke, 2002] Andreas Stolcke. SRILM-an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, pages 257–286, November 2002.

[Surdeanu *et al.*, 2008] Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. The conll 2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177, Manchester, England, August 2008. Coling 2008 Organizing Committee.

[The XTAG Research Group, 2001] The XTAG Research Group. A lexicalised tree adjoining grammar for english. Technical report, Institute for Research in Cognitive Science, University of Pennsylvannia, 2001.

[Theune *et al.*, 2006] Mariët Theune, Feikje Hielkema, and Petra Hendriks. Performing aggregation and ellipsis using discourse structures. *Research on Language & Computation*, 4(4):353–375, 2006.

[van Craenenbroeck, 2010] Jeoren van Craenenbroeck. *The syntax of ellipsis: Evidence from Dutch dialects*. Oxford University Press, 2010.

[van Noord, 2004] Gertjan van Noord. Error mining for wide-coverage grammar engineering. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL)*, pages 446–453, Barcelona, Spain, 2004.

[Vickrey and Koller, 2008] David Vickrey and Daphne Koller. Sentence simplification for semantic role labeling. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL) and the Human Language Technology Conference (HLT)*, pages 344–352, 2008.

[Vijay-Shankar and Joshi, 1985] K. Vijay-Shankar and Aravind K. Joshi. Some computational properties of tree adjoining grammars. In *Proceedings of the 23rd annual meeting on Association for Computational Linguistics*, ACL '85, pages 82–93, Stroudsburg, PA, USA, 1985. Association for Computational Linguistics.

[Vijay-Shanker and Joshi, 1988] K. Vijay-Shanker and Arvind K. Joshi. Feature structures based tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, Hungary, 1988.

[Vijay-Shanker and Joshi, 1991] K. Vijay-Shanker and Arvind K. Joshi. *Unification-based Tree Adjoining Grammars.* Technical report. University of Pennsylvania, School of Engineering and Applied Science, Department of Computer and Information Science, 1991.

[Vijay-Shanker and Weir, 1991] K. Vijay-Shanker and David J. Weir. Polynomial parsing of extensions of context-free grammars. In Masaru Tomita, editor, *Current Issues in Parsing Technology*, volume 126 of *The Springer International Series in Engineering and Computer Science*, pages 191–206. Springer US, 1991.

[Vijay-Shanker and Weir, 1993] K. Vijay-Shanker and David J. Weir. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636, December 1993.

[Vijay-Shanker, 1987] K. Vijay-Shanker. *A Study of Tree Adjoining Grammars.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, 1987.

[Vincze *et al.*, 2010] Veronika Vincze, Dóra Szauter, Attila Almási, György Móra, Zoltán Alexin, and János Csirik. Hungarian dependency treebank. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC)*, pages 1855–1862, Valletta, Malta, 2010.

[Watanabe *et al.*, 2009] Willian Massami Watanabe, Arnaldo Candido Junior, Vinícius Rodriguez Uzêda, Renata Pontin de Mattos Fortes, Thiago Alexandre Salgueiro Pardo, and Sandra Maria Aluísio. Facilita: reading assistance for low-literacy readers. In *Proceedings of the 27th ACM international conference on Design of communication*, pages 29–36. ACM, 2009.

[Weir and Joshi, 1988] David J. Weir and Aravind K. Joshi. Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 278–285, Buffalo, New York, USA, June 1988. Association for Computational Linguistics.

[White and Rajkumar, 2009] Michael White and Rajakrishnan Rajkumar. Perceptron reranking for ccg realization. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 410–419, Singapore, 2009. Association for Computational Linguistics.

[White, 2004] Michael White. Reining in CCG chart realization. In *Proceedings of the third International Conference on Natural Language Generation (INLG)*, pages 182–191, Brighton, UK, 2004.

[White, 2006] Michael White. Efficient realization of coordinate structures in combinatory categorial grammar. *Research on Language and Computation*, 4(1):39–75, 2006.

[Williams, 1977] Edwin Williams. Discourse and logical form. *Linguistic Inquiry*, 1977.

[Woodsend and Lapata, 2011] Kristian Woodsend and Mirella Lapata. Learning to simplify sentences with quasi-synchronous grammar and integer programming. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 409–420. Association for Computational Linguistics, 2011.

[Wubben *et al.*, 2012] Sander Wubben, Antal van den Bosch, and Emiel Krahmer. Sentence simplification by monolingual machine translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL): Long Papers-Volume 1*, pages 1015–1024. Association for Computational Linguistics, 2012.

[Yamada and Knight, 2001] Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting on As-*

sociation for Computational Linguistics (ACL), pages 523–530. Association for Computational Linguistics, 2001.

[Yatskar *et al.*, 2010] Mark Yatskar, Bo Pang, Cristian Danescu-Niculescu-Mizil, and Lillian Lee. For the sake of simplicity: Unsupervised extraction of lexical simplifications from wikipedia. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 365–368, Los Angeles, California, June 2010. Association for Computational Linguistics.

[Zhao *et al.*, 2009] Shiqi Zhao, Xiang Lan, Ting Liu, and Sheng Li. Application-driven statistical paraphrase generation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 834–842, Suntec, Singapore, August 2009. Association for Computational Linguistics.

[Zhong and Stent, 2005] Huayan Zhong and Amanda Stent. Building surface realizers automatically from corpora. In *Proceedings of the Workshop on Using Corpora for Natural Language Generation (UCNLG)*, volume 5, pages 49–54, 2005.

[Zhu *et al.*, 2010] Zhemin Zhu, Delphine Bernhard, and Iryna Gurevych. A monolingual tree-based translation model for sentence simplification. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*, pages 1353–1361, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.