Using a first-order theorem prover in the SemAnTE framework

Benno Kruit

Bachelor thesis Articial Intelligence major (7,5 ECTS)

3510255

Utrecht University

Supervisor: Yoad Winter

August 2013

Abstract

To allow proofs of textual entailments in the model described by Toledo et al.[10] using a first-order automated theorem prover, we need to rewrite higher-order expressions to first-order logic. This thesis describes a rewriting algorithm that captures a certain class of natural language inferences corresponding to modifiers. It is sound, but not complete.

1 Introduction

An important part of Natural Language Processing, a subfield of Artificial Intelligence, is open-domain Natural Language Inference. Given two natural language expressions, we want to determine whether the second, the hypothesis, can be inferred from the first, the premise.

While approaches exist that use minimal semantic information, the SemAnTE project aims to build a framework to show linguistic and informational processes that human readers employ when making an inference. It uses gold standard human annotations combined with a computational framework, proving the entailment relationship with full semantic information via logical deduction.

To prove the entailments, we use a first-order theorem prover, but this means the input needs to be first order. However, expressions generated in the SemAnTE framework are higher-order, so we need to convert these to first-order logic.

This thesis concerns the definition of supersets of first-order logic that capture some higher-order inference phenomena. It describes algorithms for converting expressions in these sets to first-order logic, which are proven to be sound, but not complete. It also concerns an approach for dealing with definite descriptions.

The most useful of these algorithms was implemented in Java as part of the SemAnTE framework. The component has been tested on a small set of examples and gave satisfying results. It made proving textual entailment from fully annotated sentences possible with minimal lexical information. However, our approach relies on non-logical axioms that cannot be generalized as much as we would want to.

The source code of the component of the SemAnTE framework specified in this thesis can be found at `http://phil.uu.nl/~kruit/predcalc.zip`.

2 Recognizing Textual Entailment

The Recognizing Textual Entailment (RTE) challenges [5] aim to determine automatically whether a an entailment relation holds between a naturally occurring text (T) sentence and hypothesis (H). In its definition, an entailment relation holds when, typically, a human reader would infer from T that H is most likely true. This makes textual entailment an informal notion, though it is one that covers a broad area of linguistic theory in an accessible way. The corpus used in this challenge [1] [6] [2] consists of pairs of sentences, together with whether there is an entailment relation between them. Some examples are given below.

Example 1 (TRUE) [1]
T: President Chirac has been advised that his resumption of nuclear testing would precipitate an international boycott of French products.
H: International pressure is exerted to end French nuclear tests.

[1]Pair 316 from the development set of RTE3

Example 2 (FALSE) [2]
T: Actual statistics about the deterrent value of capital punishment are not available because it is impossible to know who may have been deterred from committing a crime.
H: Capital punishment is a deterrent to crime.

The source of the data varies, and thus the type of inference varies wildly as well. Sometimes the inference can be made with very little work, but often a substantial amount of background knowledge is needed to correctly classify the entailment candidate. For example, the following inference needs no background knowledge at all:

Example 3 (TRUE) [3]
T: A Cuban American who is accused of espionage pleads innocent.
H:American accused of espionage

While we need to know that *charismatic* entails *articulate* in the following example:

Example 4 (TRUE) [4]
T: Clinton is a very charismatic person.
H: Clinton is articulate.

3 SemAnTE

All strategies for recognizing textual entailment must correspond somehow to semantic phenomena. Many of these are not yet fully described, and most approaches use various heuristic measures to solve the problem. The goal of the SemAnTE project is to elucidate the semantic phenomena of the inferential process underlying the entailments we see in the RTE corpus. It aims to advance the possibility of creating a benchmark for modeling entailment recognition. Initially, it aims to model a restrictive fragment of English, and then expand this to incrementally model increasingly complex

[2]Pair 735 from the development set of RTE2
[3]Pair 119 from the development set of RTE1
[4]Pair 48 from the test set of RTE1

semantic phenomena. Toledo et al.[10] show the predominance of restrictive, intersective and appositive modifiers in the RTE corpus and the possibility of annotating these with high consistency and the possibility to capture their various syntactic expressions by a limited set of concepts.

The SemAnTE platform [10] allows parsing and annotation of RTE text and hypothesis pairs, and generation of semantic representations that can be proven in an automatic theorem prover[8]. This way, the semantic model is guaranteed to be sound. However, the semantic model is higher-order. To be able to automatically prove the entailment relation, the generated representations need to be rewritten as first-order predicate logic. This thesis concerns a sound method for dealing with higher-order representations of modifiers and definite descriptions in the SemAnTE framework. It successfully allows proving several examples from the RTE corpus, but it is not complete.

3.1 Related work

An approach that uses very similar methods was built by Bos and Markert[4]. It produces first-order discourse representation structures (DRS, [7]) from the text and hypothesis using a CCG parser [3], which captures argument structure, polarity, and other semantic notions. It used background knowledge from WordNet to prove the DRS of the hypothesis from the text, but also matched up finite models generated by model builders to quantify the overlap between interpretations of the text and hypothesis. This corresponded to the likelihood of an inference, and a cutoff was used to classify the entailment pair.

The difference between the Montagovian approach of the semantic framework used by SemAnTE and the Discourse Representation approach used by Bos and Markert, is the amount of lexical information provided to the semantic framework. The semantic framework described below makes no assumptions about phenomena such as time, sentiment or anaphora, but is a purely structure-inspired compositional system based on minimal lexical annotations. This approach allows elegant testing of theories about inferential processes without having to deal with unrelated semantic phenomena. Furthermore, this simplicity will hopefully allow easier machine learning of the lexical information, which is now annotated manually.

4 Semantic Theory

The semantic framework we use [11] assigns types to denotations of natural language expressions. A sentence, for example, can be true or false, so its type is *truth value*, written as t. The domain of this type D_t is the set $\{0,1\}$ with the natural partial order $\leq$. Proper nouns are *entities*, written as e, whose domain D_e is the set of entities E. Using these basic types, we then construct complex types by defining functions over these sets in the standard way for functional types:

Definition 1 Let B be a finite non-empty set of basic types. The set of $\mathsf{functional\ types}$ over B is the smallest set $\mathcal{T}^{\mathsf{B}}$ that satisfies:
(i) $\mathsf{B} \subseteq \mathcal{T}^{\mathsf{B}}$
(ii) If τ and σ are types in $\mathcal{T}^{\mathsf{B}}$ then $(\tau \to \sigma)$ is also a type in $\mathcal{T}^{\mathsf{B}}$

Now we construct a domain for each of our complex types as well. For each non-basic type ts in $\mathcal{T}^{\mathsf{B}}$ we define the corresponding domain by:

$$D_{\tau\sigma} = D_{\sigma}^{D_\tau} = \text{the set of functions from } D_\tau \text{ to } D_\sigma$$

Our set of entities is not constant, so given a non-empty domain of entities $D_e = E$, we refer to the collection of domains $\mathcal{F}^E = \{D_\tau | \tau \in \mathcal{T}^{\{e,t\}}\}$ as the *frame over E*.

Every expression has a denotation in the domain of its type. For instance, the denotation of a proper noun like *Cambodia* is some entity $\mathsf{cambodia}$ in E, and the denotation of a transitive verb like *visit* is a function in D_{eet}. Because D_{eet} is the set $(\{0,1\}^E)^E$, we have a function f from entities to functions from entities to truth-values. By standard Currying, this function f characterizes a binary relationship over E. Applying these, *visits Cambodia* is denoted by a function in D_{et}, which corresponds informally to entities that visit Cambodia in a model M.

Starting out, individual words in the lexicon Σ are typed with the function TYP, and interpreted by the function I which is defined as follows:

Definition 2 An interpretation function I over a lexicon Σ with a typing function TYP and a set E, is a function from the words in Σ to the set $\mathcal{F}^E$, which sends every word $w \in \Sigma$ to an element in the corresponding typed domain: $I(w) \in D_{\text{TYP}(w)}$. In a pair with its entitiy set, it forms a model M.

Complex expressions can now be assigned denotations as following:

Definition 3 Let w be an binary parsed expression in the language L over Σ. Its denotation $[\![w]\!]^M$ is defined:

(i) If w is a word, $[\![w]\!]^M = I(w)$ of type $\textsc{typ}(w)$

(ii) If w consists of two expressions w_1 and w_2, $[\![w]\!]^M = [\![w_1]\!]^M([\![w_2]\!]^M)$ of type σ, where $[\![w_1]\!]^M \in D_{\tau\sigma}$ and $[\![w_2]\!]^M \in D_\tau$,

We now finally have a denotation of complete sentences. After supplying a model M, byparsing and annotating the sentences with lexicon functions, logical entailment between text and hypothesis sentencs T and H is model-theoretically described as the classical Tarskian property below.

Truth-conditionality criterion Let T and H be parsed expressions of type t, over an interpreted lexicon Σ. We say that the parsed sentence T logically entails H if and only if the relation $[\![T]\!]^M \leq [\![H]\!]^M$ holds between the truth-value denotations of T and H in all intended models M.

5 Creating lambda-expressions

In the SemAnTE user interface, annotators create a valid binary parse for the sentence. Sometimes it is neccesary to add anonymous nodes to the tree for a correct parse. They then attach an entry from a set of lexicon functions to each terminal node in the parse tree.

The annotated, parsed tree is now converted to an unsimplified expression by transforming each terminal node into a lambda term using the lexicon function attached to it. For example, the annotation V_1 creates a constant of type et, and the annotation MOD_R creates a more complex expression with a constant of type $(et)et$ embedded in it. This constitutes an interpreted lexicon, by creating a denotation for each word. In this way, the lexicon functions determine the phenomena we need to account for when proving the entailment relation.

Next, the binary tree of word denotations is flattened. Terms are applied when their type allows it, which finds annotation errors when neither denotation accepts the other. The generated semantic representation is fully simplified by beta-reducing, and can now be rewritten to first-order logic.

Instead of introducing the quantifiers and operators from predicate calculus into our lambda-expression language, we use a number of typed constants. For instance, quantifier constants like $\mathsf{EXISTS}_{(et)t}$ can be applied to a lambda-abstraction over a variable of type e, which is converted to a predicate logic expression with a quantifier before being sent to the theorem prover. This approach keeps the lambda calculus simple and elegant, using only a few base symbols and a mapping of constants for the associated predicate calculus symbols.

Example "British servicemen detained" [5]

Parse:

((SOME (british:$\mathrm{MOD}_{\mathrm{R}}$ servicemen:N_1)) ($\mathrm{V}_{\mathrm{AUX}}$ detained:V_1))

Unsimplified expression:

```
(
    (
        (λA0_et.(λA1_et.(EXISTS_(et)t (λx0_e.((AND_ttt (A0_et x0_e)) (A1_et x0_e))))))
            (
                (λA2_et.(λx1_e.((AND_ttt (A2_et x1_e)) ((british_(et)et A2_et) x1_e))))
                servicemen_et
            )
    )
    (
        (λA3_et.A3_et)
        detained_et
    )
)
```

Simplified expression:

$\mathsf{EXISTS}_{(et)t}\ (\lambda x_e.((\mathsf{AND}_{ttt}\ ((\mathsf{AND}_{ttt}\ (\mathsf{servicemen}_{et}\ x_e))\ ((\mathsf{british}_{(et)et}\ \mathsf{servicemen}_{et})\ x_e)))\ (\mathsf{detained}_{et}\ x_e)))$

6 Converting to first-order expressions

The lexicon functions we have for modifiers need to be defined using higher-order constants. This section concerns several extensions to a first-order lambda calculus that can be converted soundly to predicate logic. We will

[5]Hypothesis from pair 63, RTE1 development set

first inductively define the set of expressions that correspond to First-Order Logic (FOL). Then we'll define some supersets of FOL, which we call MOD, that contain modifiers. The modifiers are of type $(et)et$, so they're higher order, but the expressions in the sets we define can be converted to expressions in FOL. The conversion algorithms will be proven to be sound, but not complete. Every definition we'll give is an extension of the previous one, so MOD_2 will be a superset of MOD_1, etc. Finally, we arrive at a set of expressions that is useful to define our lexicon in.

In predicate logic, when we check whether there is an entailment between a text t and hypothesis h, we try to prove that the expression $t \rightarrow h$ is a tautology. If it is, the entailment holds.

Soundness An algorithm is *sound* if we do not generate expressions that can be proven to be a tautology while the original cannot. That is, if the generated expression φ' is a tautology, the original φ should be as well. Equivalently, if $\neg\varphi'$ is not satisfiable, $\neg\varphi$ should not be satisfiable either.

Non-completeness An algorithm is *complete* if it can generate a tautology from any expression that can be proven to be a tautology. That is, $\vDash \varphi \Rightarrow \vDash \varphi'$. To prove an algorithm is not complete, we derive for some expression φ that φ is a tautology and φ' is not satisfiable.

6.1 First-order logic

Fundamental to our definition of first-order expressions is the set of first-order constants $\mathbf{C}$. This includes arbitrary terms, of type e, and arbitrary n-ary predicates, of type $e...et$ with n es. We define $\mathbf{C}$ as the set of terms and predicates unified with the quantifier and connective constants $\{$ $\mathsf{EXISTS}_{(et)t}$, $\mathsf{FORALL}_{(et)t}$, AND_{ttt}, OR_{ttt}, $\mathsf{IMPLIES}_{ttt}$, $\mathsf{EQUIVALENCES}_{ttt}$, NOT_{tt}, TRUE_{t}, FALSE_{t} $\}$, as explained above.

Definition **FOL**, the set of first-order expressions, is defined:
- if $\mathbf{c} \in \mathbf{C}$, then $\mathbf{c} \in \mathbf{FOL}$
- if $\mathbf{v}$ is a variable of type e, then $\mathbf{v} \in \mathbf{FOL}$
- if some expressions $a_{\sigma\tau}, b_{\sigma} \in \mathbf{FOL}$ then $(a\ b) \in \mathbf{FOL}$
- if some expressions $a_{(et)t}, b_t \in \mathbf{FOL}$ then $(a_{(et)t}\ (\lambda x_e\ b_t)) \in \mathbf{FOL}$

6.2 Only constants

$\mathbf{MOD_1}$ is the smallest superset of **FOL** that includes modifiers, constants of type $(et)et$, with constants in the modifier argument position. We add a clause in the definition for the constants and limit the inductive definition of applications. Formally, it is the smallest set that satisfies:

- if $\mathbf{c} \in \mathbf{C}$, then $\mathbf{c} \in \mathbf{MOD_1}$
- if $\mathbf{m}$ is a constant of type $(et)et$, then $\mathbf{m} \in \mathbf{MOD_1}$
- if $\mathbf{v}$ is a variable of type e, then $\mathbf{v} \in \mathbf{MOD_1}$
- if some expressions $a_{\sigma\tau}, b_\sigma \in \mathbf{MOD_1}$, and when $\sigma\tau = (et)et$, b consists of only constants, then $(a\ b) \in \mathbf{MOD_1}$
- if some expressions $a_{(et)t}, b_t \in \mathbf{MOD_1}$ then $(a_{(et)t}\ (\lambda x_e\ b_t)) \in \mathbf{MOD_1}$

Algorithm To convert an expression φ from $\mathbf{MOD_1}$ to **FOL**:

- For each pair of constant $\mathbf{a}_{(et)et}$ and expression b_{et}:
  - Create a fresh constant $\mathbf{c}_{et}$
  - Let φ_{i+1} be φ_i where each occurrence of $(\mathbf{a}_{(et)et}\ b_{et})$ is replaced by $\mathbf{c}_{et}$.

In the implementation, the fresh constants that are created correspond to the expression they replace in name for readability. For example, $(\mathsf{british}_{(et)et}\ \mathsf{servicemen}_{et})$ is replaced by $\mathsf{british\_servicemen}_{et}$.

Output We start with the expression $\varphi_0 \in \mathbf{MOD_1}$, and eliminate all n constants of type $(et)et$ to end with expression φ_n . No expressions in $\mathbf{MOD_1} \setminus \mathbf{FOL}$ remain, thus $\varphi_n \in \mathbf{FOL}$.

Soundness We assume $\neg\varphi_{i+1}$ is not satisfiable. If there was a model M_i that satified $\neg\varphi_i$, we could construct a model M_{i+1} without loss of generality such that $M_{i+1}(\mathbf{c}) = M_i(\mathbf{a})(M_i(b))$. But this would make M_{i+1} satisfy $\neg\varphi_{i+1}$, leading to a contradiction, thus $\neg\varphi_i$ is not satisfiable.
All steps are sound, thus the entire algorithm is sound: $\vDash \varphi_n \Rightarrow \vDash \varphi_0$

Non-completeness Let φ_i contain two equivalent constants $\mathbf{p}_{et}, \mathbf{q}_{et}$. Because we always assume $\forall x\ \mathbf{p}(x) \leftrightarrow \mathbf{q}(x)$, we know $\forall M.M(\mathbf{p}) = M(\mathbf{q})$ holds. Additionally, φ_i contains the expressions $(\mathbf{a}_{(et)et}\ \mathbf{p}_{et})$ and $(\mathbf{a}_{(et)et}\ \mathbf{q}_{et})$, which implies $\forall M.M(\mathbf{a})(M(\mathbf{p})) = M(\mathbf{a})(M(\mathbf{q}))$. These expressions are replaced by the constants $\mathbf{c}$ and $\mathbf{d}$, respectively, but $\neg\forall M.M(\mathbf{c}) = M(\mathbf{d})$. Thus $\vDash \varphi_i \not\Rightarrow \vDash \varphi_{i+1}$, the algorithm is not complete.

6.3 Introducing abstractions

Here, we extend $\mathbf{MOD_1}$ to $\mathbf{MOD_2}$ by introducing abstractions in the modifier argument position as the last clause of the definition. It is the smallest superset of **FOL** that includes modifier constants applied to abstractions or constants. Chaining this algorithm with the previous one results in first-order expressions. Formally, $\mathbf{MOD_2}$ is defined as the smallest set that satisfies:

- if $\mathbf{c} \in \mathbf{C}$, then $\mathbf{c} \in \mathbf{MOD_2}$
- if $\mathbf{m}$ is a constant of type $(et)et$, then $\mathbf{m} \in \mathbf{MOD_2}$
- if $\mathbf{v}$ is a variable of type e, then $\mathbf{v} \in \mathbf{MOD_2}$
- if some expressions $a_{\sigma\tau}, b_\sigma \in \mathbf{MOD_2}$, and when $\sigma\tau = (et)et$, b consists of only constants, then $(a\ b) \in \mathbf{MOD_2}$
- if some expressions $a_{(et)t}, b_t \in \mathbf{MOD_2}$ then $(a_{(et)t}\ (\lambda x_e\ b_t)) \in \mathbf{MOD_2}$
- if $\mathbf{m}$ is a constant of type $(et)et$ and some expression b_t is in $\mathbf{MOD_2}$ and x is the only free variable in b_t, then $(\mathbf{m}_{(et)et}\ (\lambda x_e\ b_t)) \in \mathbf{MOD_2}$

Algorithm To convert an expression φ from $\mathbf{MOD_2}$ to $\mathbf{MOD_1}$:

- For each constant $\mathbf{m}_{(et)et}$ and expression b_t :
  - Create a new constant $\mathbf{c}_{et}$
  - Let φ_{i+1} be φ_i where each occurrence of $(\mathbf{m}_{(et)et}\ (\lambda x_e\ b_t))$ is replaced by $(\mathbf{m}_{(et)et}\ \mathbf{c}_{et})$.

Output We start with the expression $\varphi_0 \in \mathbf{MOD_2}$, and eliminate all n λ-abstractions of type et that a constant of type $(et)et$ applies to, ending with expression φ_n . No expressions in $\mathbf{MOD_2} \setminus \mathbf{MOD_1}$ remain, thus $\varphi_n \in \mathbf{MOD_1}$.

Soundness We assume $\neg\varphi_{i+1}$ is not satisfiable. If there was a model M_i that satified $\neg\varphi_i$, we could construct a model M_{i+1} without loss of generality such that for all x, $M_{i+1}(\mathbf{c})(x) = M_i(b)$. But this would make M_{i+1} satisfy $\neg\varphi_{i+1}$, leading to a contradiction, thus $\neg\varphi_i$ is not satisfiable.
All steps are sound, thus the entire algorithm is sound: $\vDash \varphi_n \Rightarrow \vDash \varphi_0$

Non-completeness Non-completeness is derived in the same manner as with Algorithm 1.

6.4 Free variables

Now we extend $\mathbf{MOD_2}$ to $\mathbf{MOD_3}$ and allow free variables in the modifier argument position. This means we can remove the restrictions in the definition on both clauses. This algorithm can be seen as a generalization of the two previous ones. $\mathbf{MOD_3}$ is the smallest set that satisfies:

- if $\mathbf{c} \in \mathbf{C}$, then $\mathbf{c} \in \mathbf{MOD_3}$
- if $\mathbf{m}$ is a constant of type $(et)et$, then $\mathbf{m} \in \mathbf{MOD_3}$
- if $\mathbf{v}$ is a variable of type e, then $\mathbf{v} \in \mathbf{MOD_3}$
- if some expressions $a_{\sigma\tau}, b_\sigma \in \mathbf{MOD_3}$, then $(a\ b) \in \mathbf{MOD_3}$
- if some expressions $a_{(et)t}, b_t \in \mathbf{MOD_3}$ then $(a_{(et)t}\ (\lambda x_e\ b_t)) \in \mathbf{MOD_3}$
- if $\mathbf{m}$ is a constant of type $(et)et$ and some expression $b_t \in \mathbf{MOD_3}$, then $(\mathbf{m}_{(et)et}\ (\lambda x_e\ b_t)) \in \mathbf{MOD_3}$

Algorithm To convert an expression from $\mathbf{MOD_3}$ to $\mathbf{FOL}$:

- For each constant $\mathbf{m}_{(et)et}$ and expression p_{et}:
  - Let $v_0 ... v_n$ be the n free variables in p (which are type e).
  - Let $\mathbf{c}$ be a fresh constant with a predicate type, where its type is $e...et$ with $n+1$ es.
  - Let ψ_{et} be the constant $\mathbf{c}_{e..et}$ applied to all variables v.
  - Let φ_{i+1} be φ_i where each occurrence of $(\mathbf{m}_{(et)et}\ p_{et})$ is replaced by ψ_{et}.

For example, when x_e is a free variable, $((\mathsf{quickly}_{(et)et}\ (\mathsf{ate}_{eet}\ x_e))\ \mathsf{john}_e)$ becomes $((\mathsf{quickly\_ate}_{eet}\ x_e)\ \mathsf{john}_e)$.

Output We start with the expression $\varphi_0 \in \mathbf{MOD_3}$, and eliminate all expressions of the form $(\mathbf{m}_{(et)et}\ p_{et})$, ending with expression φ_n . No expressions in $\mathbf{MOD_3} \setminus \mathbf{FOL}$ remain, thus $\varphi_n \in \mathbf{FOL}$.

Soundness We assume $\neg\varphi_{i+1}$ is not satisfiable. If there was a model M_i that satified $\neg\varphi_i$, there would be a function $f_{et} = M_i(p)$. We could construct a function g such that g applied to the values of all free variables v in p would give us this f. Now we can construct a model M_{i+1} without loss of generality where g is the denotation of $\mathbf{c}$. So now $M_i(p) = f = g(v_0 ... v_n)$ and $g = M_{i+1}(\mathbf{c})$. But this would make M_{i+1} satisfy $\neg\varphi_{i+1}$, leading to a contradiction, thus $\neg\varphi_i$ is not satisfiable.
All steps are sound, thus the entire algorithm is sound: $\vDash \varphi_n \Rightarrow \vDash \varphi_0$

Non-completeness Non-completeness is derived in the same manner as with Algorithm 1.

6.5 Complex modifiers

We generalize the previous algorithm further to show conversion of any expression built with modifiers. $\mathbf{MOD_4}$ extends $\mathbf{MOD_3}$ by allowing complex expressions of type $(et)et$, introducing two more constants in the second clause. Formally, $\mathbf{MOD_4}$ is the smallest set that satisfies:

- if $\mathbf{c} \in \mathbf{C}$, then $\mathbf{c} \in \mathbf{MOD_4}$
- if $\mathbf{m}$ is a constant of type $(et)et$, $e(et)et$, or $((et)et)(et)et$, then $\mathbf{m} \in \mathbf{MOD_4}$
- if $\mathbf{v}$ is a variable of type e, then $\mathbf{v} \in \mathbf{MOD_4}$
- if some expressions $a_{\sigma\tau}, b_\sigma \in \mathbf{MOD_4}$, then $(a\ b) \in \mathbf{MOD_4}$
- if some expressions $a_{(et)t}, b_t \in \mathbf{MOD_4}$ then $(a_{(et)t}\ (\lambda x_e\ b_t)) \in \mathbf{MOD_4}$
- if $\mathbf{m}$ is an expression of type $(et)et$ and some expression $b_t \in \mathbf{MOD_4}$, then $(\mathbf{m}_{(et)et}\ (\lambda x_e\ b_t)) \in \mathbf{MOD_4}$

Algorithm To convert an expression from $\mathbf{MOD_4}$ to $\mathbf{FOL}$:

- For each application χ_{et} of expressions $m_{(et)et}$ and p_{et}:
  - Let $v_0 ... v_n$ be the n free variables in χ_{et} (which are type e).
  - Let $\mathbf{c}$ be a fresh constant with a type $e...et$ with $n+1$ es.
  - Let ψ_{et} be the constant $\mathbf{c}_{e..et}$ applied to all variables v.
  - Let φ_{i+1} be φ_i where each occurrence of $(m_{(et)et}\ p_{et})$ is replaced by ψ_{et}.

Output We start with the expression $\varphi_0 \in \mathbf{MOD_4}$. The only expressions in $\mathbf{MOD_4} \setminus \mathbf{FOL}$ are of type et, and we replace each one with an expression of type et in $\mathbf{FOL}$, ending with expression $\varphi_n \in \mathbf{FOL}$.

Soundness Soundness is derived in the same manner as with Algorithm 3.

Non-completeness Non-completeness is derived in the same manner as with Algorithm 1.

$\mathbf{MOD_4}$ is the largest superset of $\mathbf{FOL}$ we explored, and it is useful to define the lexicon in it.

7 Handling definite descriptions

Definite articles are defined in the lexicon using the iota function. This is a function from one-place predicates, which are of type *et*, to entities, which are of type *e*. Use of a definite article means that the predicate holds on only one entity, whose uniqueness is presupposed. The iota function returns this entity, which means that we assume a simple context where all anaphora are uniquely described.

The iota function is not first-order and the presupposition is not part of the entailment relation. Thus, we separate these parts of the expressions and presuppose them as assumptions in the proof.

Before converting the expressions with modifiers to first-order expressions, we convert an expression to a set of presuppositions and semantic content. Every iota function and its argument are replaced by a fresh constant, and the uniqueness of the entity that satisfies the predicate is added to the set of presuppositions:

$$Q(\iota(P)) \rightsquigarrow \begin{cases} \text{presupposition:} & \forall x.(P(x) \leftrightarrow x = c) \\ \text{semantic content:} & Q(c) \end{cases}$$

In the implementation, the fresh constants are numbered. When running the theorem prover, it tries to prove the semantic content of the hypothesis from both sets of presuppositions, and the semantic content of the text.

Example

Entailment pair:
T: john is the man
H: john is a man

Analysis:
$(\mathsf{john} = \iota(\mathsf{man})) \leq \mathsf{man}(\mathsf{john})$

Theorem prover input:
Assumptions:

```
 all x (man(x) <-> x = c0)     presupposition
 john = c0                     semantic content
```

Goals:

```
man(john)
```

However, when we switch the text and hypothesis, the entailment still holds. This is a side-effect of this implementation, but is not unwanted behaviour.

Example (inverse)

Entailment pair:
T: john is a man
H: john is the man

Analysis:
$\mathsf{man}(\mathsf{john}) \leq (\mathsf{john} = \iota(\mathsf{man}))$

Theorem prover input:
Assumptions:
```
 all x (man(x) <-> x = c0)      presupposition
 man(john)                      semantic content
```
Goals:
```
john = c0
```

8 Implementation results

The algorithms were implemented in Java as a part of the SemAnTE annotation framework. They were run on a small set of test cases and gave satisfying results. Below are two examples.

Example 1 In this example[6], no non-logical axioms are needed for the proof. The algorithm creates two new constants from the definite descriptions, and uses one for a correct inference. The sentences are annotated with quantifiers, an anonymous node, a restrictive modifier and restrictive prepositions.
T: [A:SOME [[Cuban:MOD$_R$ American:N] [who:WHO [is:V$_{AUX}$ [accused:V$_1$ [of:P$_R$ espionage:N]]]]] [pleads‿innocent:V$_1$]].
H: [SOME [American:N [V$_{AUX}$ [accused:V$_1$ [of:P$_R$ [espionage:NP]]]]]]

[6]Pair 119 from the development set of RTE1

Lambda-expression:
T: $\mathsf{EXISTS}_{(et)t}$ $(\lambda \mathsf{x0}_e.((\mathsf{AND}_{ttt}$ $((\mathsf{AND}_{ttt}$ $((\mathsf{AND}_{ttt}$ $(\mathsf{american}_{et}$ $\mathsf{x0}_e))$ $((\mathsf{cuban}_(\mathrm{et})\mathrm{et}$ $\mathsf{american}_{et})$ $\mathsf{x0}_e)))$ $((\mathsf{AND}_{ttt}$ $(\mathsf{accused}_{et}$ $\mathsf{x0}_e))$ $(((\mathsf{of}_{e(et)et}$ $\mathsf{x0}_e)$ $\mathsf{accused}_{et})$ $\mathsf{espionage}_e))))$ $(\mathsf{pleads\_innocent}_{et}$ $\mathsf{x0}_e)))$
H: $\mathsf{EXISTS}_{(et)t}$ $(\lambda \mathsf{x0}_e.((\mathsf{AND}_{ttt}$ $(\mathsf{american}_{et}$ $\mathsf{x0}_e))$ $((\mathsf{AND}_{ttt}$ $(\mathsf{accused}_{et}$ $\mathsf{x0}_e))$ $(((\mathsf{of}_{e(et)et}$ $\mathsf{x0}_e)$ $\mathsf{accused}_{et})$ $\mathsf{espionage}_e))))$

Here, the predicate $\mathsf{accused}_{et}$ corresponds to the passive meaning of the word *accused*. Below, we see the algorithm creates a new predicate `cuban_american` from the expression $(\mathsf{cuban}_(\mathrm{et})\mathrm{et}$ $\mathsf{american}_{et})$.

Theorem prover input:
Assumption:
```
   exists x0 (((american(x0) & cuban_american(x0)) & (accused(x0)
& accused_of(x0, espionage))) & pleads_innocent(x0)).
```
Goal:
```
   exists x0 (american(x0) & (accused(x0) & accused_of(x0, espionage))).
```

Example 2 In the next example[7], four definite descriptions lead to four constants defined as presuppositions. We also see that it is not possible to prove this entailment without a non-logical axiom: `all x all y (last_president_of(x, y) -> former_president_of(x, y)).` The lexical annotations are not shown here.
T: The head of the Italian opposition, Romano Prodi, was the last president of the European Commission.
H: Romano Prodi is a former president of the European Commission.

The lambda-expression left out for readability. Theorem prover input:
Assumption:
```
   all x ((commission(x) & european_commission(x)) <-> x=c1).
   all x (((president(x) & president_of(x, c1)) & last_president_of(x,
c1)) <-> x=c2).
   all x ((opposition(x) & italian_opposition(x)) <-> x=c3).
   all x ((head(x) & head_of(x, c3)) <-> x=c4).
   exists x (c2=c4 & (x=c2 & x=Romano_Prodi)).
```
Goal:

[7]Pair 401 from the test set of RTE2

```
   exists x (((president(x0) & president_of(x0, c1)) & former_president_of(x,
c1)) & x=Romano_Prodi).
```

We see that the definite descriptions have created four numbered constants.

9 Discussion

Although the algorithm works adequately in the minimalist SemAnTE environment, there is a problem we have to keep an eye on in the future. The non-logical axiom we needed to add in the second example above, `all x all y (last_president_of(x, y) -> former_president_of(x, y)).`, does not capture very general lexical information. Ideally, we would like to add a higher-order axiom such as $\forall P \forall x \forall y.\mathsf{last}(P)(x)(y) \rightarrow \mathsf{former}(P)(x)(y)$.

One option suggested by Stephen Pulman [9] basically concerns the reification of predicates within modifiers. Modifiers with constants would create an entity that corresponds to its argument. For example, half a restrictive modifier $((\mathsf{short}_{(et)et}\ \mathsf{man}_{et})\ x_e)$ would become `short(Man, x)`.

This approach has not been implemented, because it was unclear how to deal with complex nested modifiers. The expression $((\mathsf{former}_{(et)et}\ ((\mathsf{of}_{e(et)et}\ \mathsf{x}_e)\ \mathsf{president}_{et}))\ y_e)$ would be converted to something of the form `former(H, y)`, but it is not specified how `H` could contain the free variable x_e. One option for `H` is a function `Of(President, x)`, but this has not been explored due to the scope of this work, and neither has a hybrid approach of only applying this method to the simple cases.

10 Conclusion

This work has shown an algorithm for converting higher-order expressions from the SemAnTE framework to first-order expressions. It is shown to be sound, but not complete. In the minimalist semantic representations, the only phenomena we had to deal with were definite descriptions and modifiers. The approach we took made proving textual entailment from fully annotated sentences possible with minimal lexical information.

Future work is needed to deal with completeness and more general non-logical axioms. In the mean time, the current implementation facilitates soundly annotating the RTE dataset in the SemAnTE platform.

Acknowledgements

I would like to thank Assaf Toledo and Pepijn Kokke for their help on implementing my component of the SemAnTE platform. It was impressive to see the result of my work integrated into the annotation tool. I would also like to thank Assaf for his helpful comments on this thesis.

Finally, I would like to thank my advisor and head of the SemAnTE project Yoad Winter for all his help and getting me to finally start working bottom-up. He has truly taught me the virtues of starting simple and building outwards, instead of implementing first and thinking later.

References

[1] R. Bar Haim, I. Dagan, B. Dolan, L. Ferro, D. Giampiccolo, B. Magnini, and I. Szpektor. The second pascal recognising textual entailment challenge. 2006.

[2] L. Bentivogli, I. Dagan, H. T. Dang, D. Giampiccolo, and B. Magnini. The fifth pascal recognizing textual entailment challenge. *Proceedings of TAC*, 9:14–24, 2009.

[3] J. Bos, S. Clark, M. Steedman, J. R. Curran, and J. Hockenmaier. Wide-coverage semantic representations from a ccg parser. In *Proceedings of the 20th international conference on Computational Linguistics*, page 1240. Association for Computational Linguistics, 2004.

[4] J. Bos and K. Markert. Recognising textual entailment with logical inference. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 628–635. Association for Computational Linguistics, 2005.

[5] I. Dagan, O. Glickman, and B. Magnini. The pascal recognising textual entailment challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190. Springer, 2006.

[6] D. Giampiccolo, B. Magnini, I. Dagan, and B. Dolan. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics, 2007.

[7] H. Kamp and U. Reyle. *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*. Number 42. Springer, 1993.

[8] W. McCune. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[9] S. Pulman. Second order inference in nl semantics, invited talk at the third workshop on controlled natural language (cnl 2012), 2012. `http://attempto.ifi.uzh.ch/site/cnl2012/slides/pulman_secondorder.pdf`.

[10] A. Toledo, S. Katrenko, S. Alexandropoulou, H. Klockmann, A. Stern, I. Dagan, and Y. Winter. Semantic annotation for textual entailment recognition. In *Advances in Computational Intelligence*, pages 12–25. Springer, 2013.

[11] Y. Winter. *Elements of Formal Semantics*. Unpublished ms., to appear with Edinburgh University Press, 2010.