Natural Language to Predicate Calculus

CSC 173 – Logic Module

Julian E. Herwitz

The goal of this assignment was to create a program in Prolog to convert natural English language into first order predicate calculus. This was successfully accomplished in two stages - by first creating a parser to convert the input sentence into a parse tree and then creating a translator to convert the parse tree into first-order logic. The parser is used to check sentence grammatical, syntactic, and lexical accuracy for badly formed sentences. While doing this, a parse tree is created. This parse tree is then sent to a translator, which uses specified rules to infer semantics, and from these interpretations creates predicate calculus.

Goal

The overarching goal of this assignment was to create a program, written in Prolog, to convert natural English language into first-order predicate calculus. This program was created using two components, a parser and a translator.

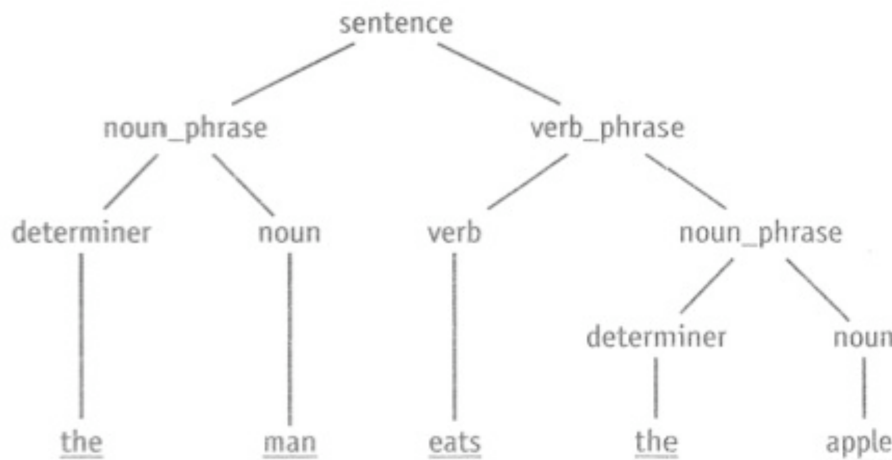
The parser can be seen abstractly as a black-box that takes an input sentence (in English) and returns a parse tree of the sentence. The goal of the parsing is to check the parameter sentence conforms to the language's grammatical rules and the program's lexicon. This parsing process doesn't necessarily factor into the functionality of the sentence-translation engine, but is implemented to simplify the translation process.

After parsing, the synthesized parse tree is sent to a translator. This translator recurses through the parse tree, using determiners, quantifiers, and other key words to quantify and connect statements, and returns the translated first order logic.

Using the parser and translator, the natural language to first-order predicate calculus converter was successfully created.

Methods

Before I describe the implementation of my program, it is first important to discuss the overarching abstractions the program is working towards.
When something is parsed, it is parsed according to a "tree" structure. This structure defines how to categorize words according to the defined grammar. The interior nodes of the tree represent different (and, as the nodes move lower, more specific) parts of speech, while the leaf nodes represent terminal symbols, or words in the language lexicon. An example of a parse tree is as follows:



Clocksin parse tree[1]

Although this is a simplified version of the parse tree used for the program (which will be depicted below), the structure remains the same. The sentence is broken down into different parts of speech, which can each in turn be broken

[1] Clocksin. Using Prolog Grammar Rules

further down. This continues until the actual word is reached, at which point the parser may recurse back up the tree and combine terms to complete the higher-order parts of speech.

The second fundamental concept to the program is the formation of first-order predicate calculus. First-order predicate calculus is a formalized logic system that uses quantifiers and propositional logic to represent and determine a wide range of logical situations. To convert something into predicate calculus, one must attempt to interpret the semantics of the sentence, and create relationships and quantifications using this interpretation. This is intuitive for humans, but is a challenge to automate computationally. It can be completed, however, by searching for certain words, which may be used to effectively determine semantics. For example, words such as "all" or "some," relate to existential and universal quantification. To-be verbs usually relate to implication, and adjectives create an "and" relationship between two predicates.

My program works by implementing these two concepts together to create an effective language conversion pipeline. Specifically, the parse tree for my program can be found in Appendix A.

The differences between this tree and the simplified tree are analyzed and interpreted in the subsequent discussion section. The translator uses the previously defined interpretation of certain English words to effectively synthesize a first-order logical statement from the above parse tree.

My program's parsing is done entirely recursively. The parse tree interior nodes are synthesized into rules and leaf nodes relate to hand-instrumented atoms. The parser recurses through the sentence, naturally creating a parse tree from the recursive stack. If a grammatical, syntactic, or lexical error is encountered, the parser returns with a false result and the program is terminated. This is important, as one of the major goals of the parser was to handle potential sentence errors. This aspect helps streamline the translator, as parse trees sent to the translator are free of errors so error handling does not need to be considered.

The translator's conversion is executed (as is everything in Prolog) recursively. The translator recurses through the parse tree, searching for certain words which may represent logical meaning. For example, the word "some" relates to an existential quantification, and according the program will implement an "exists" symbol. Adjectives elicit an "and" operator. Certain verbs relate to implication. Since the translation is executed recursively, it is simple to enter nested logical statements, and continue writing the original statement after the program has recursed back up to the original level. Although the assignment recommended implementing the parser as a subroutine of the translator, I decided to create them as separate entities, and call them simultaneously. The reasoning for this was based in modularity – I wanted to keep the different program components as separate and, in effect, modular as possible.

The following is the program's recognized lexicon. This is very easily extendable to a wide range of nouns, determiners, quantifiers, verbs, and adjectives.

Noun	**Verb**	**Determiner**	**Relative Clause**
apple	conscripts	the	that
apples	conscript	an	whom
boy	likes	a	who
boys	like		which
girl	runs		
girls	run	**Adjective**	
government	eats	evil	
governments	eat	divine	
watermelon	contains	pacifist	
watermelons	contain		
person	is		
people	are	**Quantifier**	
flavor		some	
flavors		all	

Program lexicon. Rows bear no relation. Words are case sensitive.

Results

The conversion program was successfully created and was broadly tested. Test results showed that the program worked as expected to arbitrary sentence complexity. Two of these tests are as follows:

Test 1:
Parameter sentence "All governments that conscript some pacifist people are evil."

```
?- sentence(_,Tree,[all,governments,that,conscript,some,pacifist,people,are,evil],[]), s(Tree,X).
```

Parse Tree:

```
Tree = sentence(noun_phrase(determiner(all), noun(governments), rel_clause(rel(that),
verb_phrase(verb(conscript), noun_phrase(determiner(some), adjective(pacifist), noun(people))))),
verb_phrase(be_verb(are), adjective(evil)))
```

First-Order logical representation:

```
X = all(x23, governments(x23)&conscript(x23, exists(x24, people(x24)&pacifist(x24)))=>evil(x23))
```

Test 2:
Parameter sentence "All people like all apples that contain some divine flavor."

```
?- sentence(_,Tree,[all,people,like,all,apples,that,contain,some,divine,flavor],[]), s(Tree,X).
```

Parse Tree:

```
Tree = sentence(noun_phrase(determiner(all), noun(people)), verb_phrase(verb(like),
noun_phrase(determiner(all), noun(apples), rel_clause(rel(that), verb_phrase(verb(contain),
noun_phrase(determiner(some), adjective(divine), noun(flavor)))))))
```

First-Order logical representation:

X = all(x36, people(x36)=>like(x36, all(x37, apples(x37)&contain(x37, exists(x38, flavor(x38)&divine(x38))))))

The only unsuccessful test was of the semantics of relative clauses. Certain sentences sent through the parser involving determination between the usage of which and that, which should be contradicted as a false sentence, are not differentiated. This error, however, has no significant bearing on translatability. Another important note is that the predicate first-order logic functions do not change in my implementation as they do in the assignment. I decided to keep the original names (e.g. people(x2) instead of person(x2)) to maintain understanding of where the logic was originating from.

For a complete trace of test one, see Appendix B.

Discussion

This project contained multiple difficult technical problems relevant for discussion. For the parser, the most significant of these involved differentiating types of phrases from each other. In the Clocksin parse tree, only one type of noun phrase exists, containing a single determiner followed by a single noun. To make a more flexible and powerful parser, a more diverse interpretation of phrases is required. I extended this narrow interpretation by creating multiple definitions for different parts of speech, allowing the program greater flexibility in searching for a relevant rule to apply. This was implemented by duplicating rules and modifying each to represent another permutation of the phrase. For example, I implemented five separate noun phrases. As can be seen in the parse tree above, each individual phrase contains different component parts of speech. One contains only a determiner followed by a noun, while another contains a determiner, adjective, noun, and relative clause (which itself has multiple implementations). In this way, the parser can effectively parse sentences with both types of noun phrases.

Another difficulty accounted for while parsing was the continuity and consistency of number throughout the sentences. An important consideration, not only do syntactic and lexical accuracy factor into a sentence's correctness, but also the consistency of number throughout. For example, "the boys run" is a correct sentence while "the boys runs" is not. This problem is handled by implementation of Clocksin's analogous solution – the passing through recursion of a plurality variable. Each rule takes and returns a parameter that displays the current object number. If the parameter number does not match the number located in the word atom, the parsing will fail because of a badly formed sentence.

There are certain quantifications, however, which are impossible to parse or translate – they are inherently ambiguous. "Every boy loves a girl" is an example of this. This sentence is ambiguous, does every boy love the same girl or does each boy love a different girl? There is no definitive way to computationally determine the answer.

A major problem that had to be overcome with the Prolog implementation involved the way Prolog deals with the remaining sentence after recursing through the phrase rule. By default, if not explicitly handled, Prolog will not return the remainder of the sentence to above functions. This effectively makes the rest of the

sentence inaccessible garbage. In order to maintain this remainder, the implementation of the “-->” operator was required. This operator makes sure the remaining sentence isn’t consumed; the function returns the remainder of the sentence for further use.

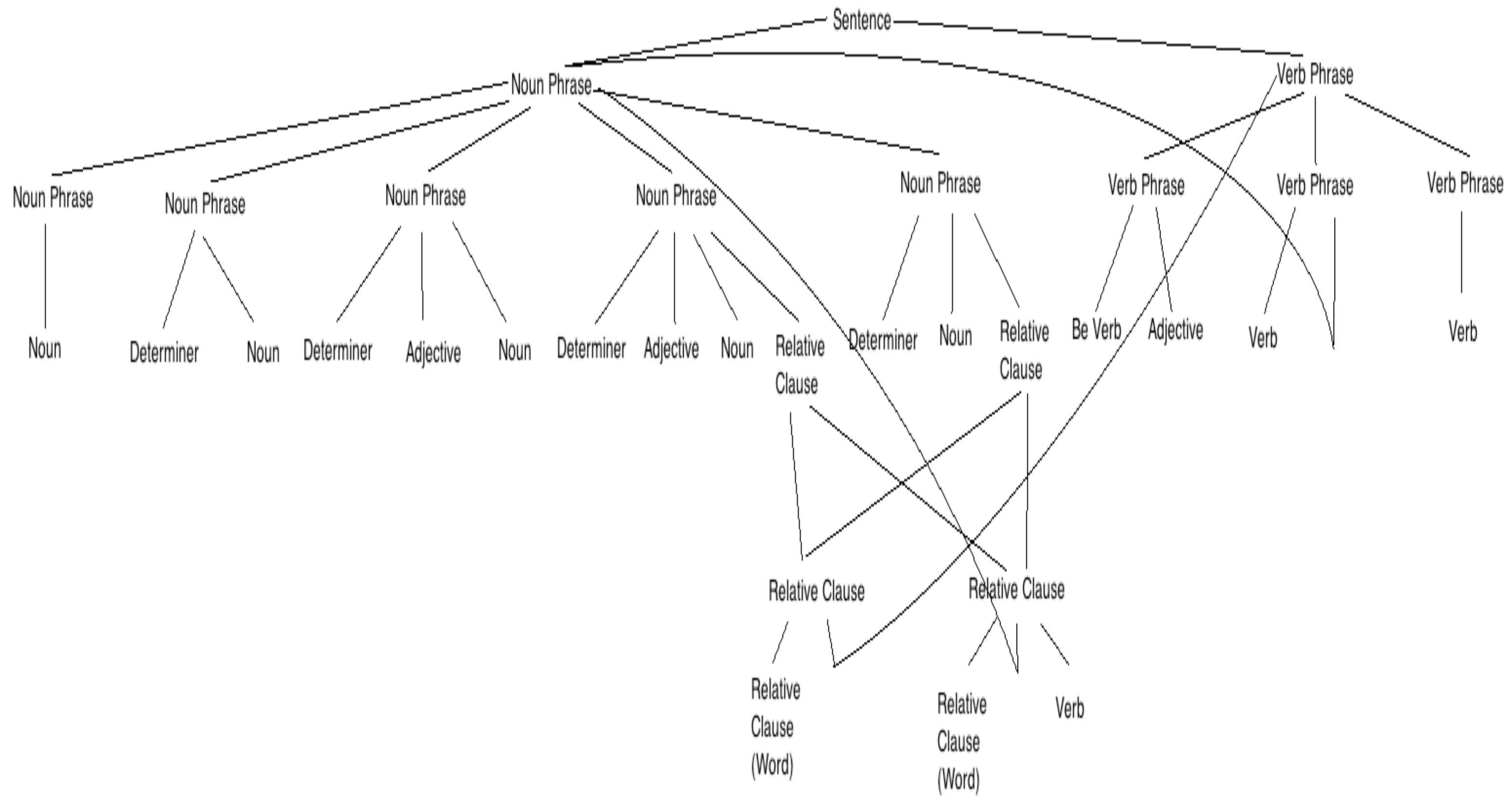
References

Karl L. Stratos (a.k.a. Jang Sun Lee), October 2010.
http://www.cs.rochester.edu/u/brown/173/Exercises/08Exer/NLtoFOL/LogicTrans.html,
http://www.cs.rochester.edu/u/brown/173/Exercises/08Exer/NLtoFOL/comment2.html

Clocksin. Using Prolog Grammar Rules. 2003.

Appendix A

Parse tree for parser.



Sentence
Noun Phrase
Verb Phrase
Noun Phrase
Noun Phrase
Noun Phrase
Noun Phrase
Noun Phrase
Verb Phrase
Verb Phrase
Verb Phrase
Noun
Determiner
Noun
Determiner
Adjective
Noun
Determiner
Adjective
Noun
Relative Clause
Determiner
Noun
Relative Clause
Be Verb
Adjective
Verb
Verb
Relative Clause
Relative Clause
Relative Clause (Word)
Relative Clause (Word)
Verb

Appendix B

Trace of conversion of "All governments that conscript some pacifist people are evil."

```
?- trace,
sentence(_,Tree,[all,governments,that,conscript,some,pacifist,people,are,evil],[])
, s(Tree,X).
  Call: (8) sentence(_G257, _G258, [all, governments, that, conscript, some,
pacifist, people, are|...], []) ? creep
  Call: (9) noun_phrase(_G257, _G445, [all, governments, that, conscript, some,
pacifist, people, are|...], _L211) ? creep
  Call: (10) noun(_G257, _G448, [all, governments, that, conscript, some, pacifist,
people, are|...], _L211) ? creep
  Call: (11) is_noun(all, _G257) ? creep
  Fail: (11) is_noun(all, _G257) ? creep
  Fail: (10) noun(_G257, _G448, [all, governments, that, conscript, some, pacifist,
people, are|...], _L211) ? creep
  Redo: (9) noun_phrase(_G257, _G445, [all, governments, that, conscript, some,
pacifist, people, are|...], _L211) ? creep
  Call: (10) determiner(_G257, _G448, [all, governments, that, conscript, some,
pacifist, people, are|...], _L233) ? creep
  Call: (11) is_determiner(all, _G257) ? creep
  Exit: (11) is_determiner(all, plural) ? creep
  Call: (11) _L233=[governments, that, conscript, some, pacifist, people, are, evil]
? creep
  Exit: (11) [governments, that, conscript, some, pacifist, people, are,
evil]=[governments, that, conscript, some, pacifist, people, are, evil] ? creep
  Exit: (10) determiner(plural, determiner(all), [all, governments, that,
conscript, some, pacifist, people, are|...], [governments, that, conscript, some,
pacifist, people, are, evil]) ? creep
  Call: (10) noun(plural, _G449, [governments, that, conscript, some, pacifist,
people, are, evil], _L211) ? creep
  Call: (11) is_noun(governments, plural) ? creep
  Exit: (11) is_noun(governments, plural) ? creep
  Call: (11) _L211=[that, conscript, some, pacifist, people, are, evil] ? creep
  Exit: (11) [that, conscript, some, pacifist, people, are, evil]=[that, conscript,
some, pacifist, people, are, evil] ? creep
  Exit: (10) noun(plural, noun(governments), [governments, that, conscript,
some, pacifist, people, are, evil], [that, conscript, some, pacifist, people, are, evil])
? creep
  Exit: (9) noun_phrase(plural, noun_phrase(determiner(all),
noun(governments)), [all, governments, that, conscript, some, pacifist, people,
are|...], [that, conscript, some, pacifist, people, are, evil]) ? creep
  Call: (9) verb_phrase(plural, _G446, [that, conscript, some, pacifist, people, are,
evil], []) ? creep
  Call: (10) verb(plural, _G455, [that, conscript, some, pacifist, people, are, evil],
[]) ? creep
  Call: (11) is_verb(that, plural) ? creep
  Fail: (11) is_verb(that, plural) ? creep
```

Fail: (10) verb(plural, _G455, [that, conscript, some, pacifist, people, are, evil], []) ? creep
Redo: (9) verb_phrase(plural, _G446, [that, conscript, some, pacifist, people, are, evil], []) ? creep
Call: (10) verb(plural, _G455, [that, conscript, some, pacifist, people, are, evil], _L255) ? creep
Call: (11) is_verb(that, plural) ? creep
Fail: (11) is_verb(that, plural) ? creep
Fail: (10) verb(plural, _G455, [that, conscript, some, pacifist, people, are, evil], _L255) ? creep
Redo: (9) verb_phrase(plural, _G446, [that, conscript, some, pacifist, people, are, evil], []) ? creep
Call: (10) be_verb(plural, _G455, [that, conscript, some, pacifist, people, are, evil], _L255) ? creep
Call: (11) is_be_verb(that, plural) ? creep
Fail: (11) is_be_verb(that, plural) ? creep
Fail: (10) be_verb(plural, _G455, [that, conscript, some, pacifist, people, are, evil], _L255) ? creep
Fail: (9) verb_phrase(plural, _G446, [that, conscript, some, pacifist, people, are, evil], []) ? creep
Redo: (9) noun_phrase(_G257, _G445, [all, governments, that, conscript, some, pacifist, people, are|...], _L211) ? creep
Call: (10) determiner(_G257, _G448, [all, governments, that, conscript, some, pacifist, people, are|...], _L234) ? creep
Call: (11) is_determiner(all, _G257) ? creep
Exit: (11) is_determiner(all, plural) ? creep
Call: (11) _L234=[governments, that, conscript, some, pacifist, people, are, evil] ? creep
Exit: (11) [governments, that, conscript, some, pacifist, people, are, evil]=[governments, that, conscript, some, pacifist, people, are, evil] ? creep
Exit: (10) determiner(plural, determiner(all), [all, governments, that, conscript, some, pacifist, people, are|...], [governments, that, conscript, some, pacifist, people, are, evil]) ? creep
Call: (10) adjective(plural, _G449, [governments, that, conscript, some, pacifist, people, are, evil], _L235) ? creep
Call: (11) is_adjective(governments, plural) ? creep
Fail: (11) is_adjective(governments, plural) ? creep
Fail: (10) adjective(plural, _G449, [governments, that, conscript, some, pacifist, people, are, evil], _L235) ? creep
Redo: (9) noun_phrase(_G257, _G445, [all, governments, that, conscript, some, pacifist, people, are|...], _L211) ? creep
Call: (10) determiner(_G257, _G448, [all, governments, that, conscript, some, pacifist, people, are|...], _L234) ? creep
Call: (11) is_determiner(all, _G257) ? creep
Exit: (11) is_determiner(all, plural) ? creep
Call: (11) _L234=[governments, that, conscript, some, pacifist, people, are, evil] ? creep
Exit: (11) [governments, that, conscript, some, pacifist, people, are, evil]=[governments, that, conscript, some, pacifist, people, are, evil] ? creep

```
  Exit: (10) determiner(plural, determiner(all), [all, governments, that, conscript, some, pacifist, people, are|...], [governments, that, conscript, some, pacifist, people, are, evil]) ? creep
  Call: (10) noun(plural, _G449, [governments, that, conscript, some, pacifist, people, are, evil], _L235) ? creep
  Call: (11) is_noun(governments, plural) ? creep
  Exit: (11) is_noun(governments, plural) ? creep
  Call: (11) _L235=[that, conscript, some, pacifist, people, are, evil] ? creep
  Exit: (11) [that, conscript, some, pacifist, people, are, evil]=[that, conscript, some, pacifist, people, are, evil] ? creep
  Exit: (10) noun(plural, noun(governments), [governments, that, conscript, some, pacifist, people, are, evil], [that, conscript, some, pacifist, people, are, evil]) ? creep
  Call: (10) rel_clause(plural, _G450, [that, conscript, some, pacifist, people, are, evil], _L211) ? creep
  Call: (11) rel(plural, _G456, [that, conscript, some, pacifist, people, are, evil], _L257) ? creep
  Call: (12) is_rel(that, plural) ? creep
  Exit: (12) is_rel(that, plural) ? creep
  Call: (12) _L257=[conscript, some, pacifist, people, are, evil] ? creep
  Exit: (12) [conscript, some, pacifist, people, are, evil]=[conscript, some, pacifist, people, are, evil] ? creep
  Exit: (11) rel(plural, rel(that), [that, conscript, some, pacifist, people, are, evil], [conscript, some, pacifist, people, are, evil]) ? creep
  Call: (11) verb_phrase(plural, _G457, [conscript, some, pacifist, people, are, evil], _L211) ? creep
  Call: (12) verb(plural, _G461, [conscript, some, pacifist, people, are, evil], _L211) ? creep
  Call: (13) is_verb(conscript, plural) ? creep
  Exit: (13) is_verb(conscript, plural) ? creep
  Call: (13) _L211=[some, pacifist, people, are, evil] ? creep
  Exit: (13) [some, pacifist, people, are, evil]=[some, pacifist, people, are, evil] ? creep
  Exit: (12) verb(plural, verb(conscript), [conscript, some, pacifist, people, are, evil], [some, pacifist, people, are, evil]) ? creep
  Exit: (11) verb_phrase(plural, verb_phrase(verb(conscript)), [conscript, some, pacifist, people, are, evil], [some, pacifist, people, are, evil]) ? creep
  Exit: (10) rel_clause(plural, rel_clause(rel(that), verb_phrase(verb(conscript))), [that, conscript, some, pacifist, people, are, evil], [some, pacifist, people, are, evil]) ? creep
  Exit: (9) noun_phrase(plural, noun_phrase(determiner(all), noun(governments), rel_clause(rel(that), verb_phrase(verb(conscript)))), [all, governments, that, conscript, some, pacifist, people, are|...], [some, pacifist, people, are, evil]) ? creep
  Call: (9) verb_phrase(plural, _G446, [some, pacifist, people, are, evil], []) ? creep
  Call: (10) verb(plural, _G465, [some, pacifist, people, are, evil], []) ? creep
  Call: (11) is_verb(some, plural) ? creep
  Fail: (11) is_verb(some, plural) ? creep
  Fail: (10) verb(plural, _G465, [some, pacifist, people, are, evil], []) ? creep
```

Redo: (9) verb_phrase(plural, _G446, [some, pacifist, people, are, evil], []) ?
creep
Call: (10) verb(plural, _G465, [some, pacifist, people, are, evil], _L299) ? creep
Call: (11) is_verb(some, plural) ? creep
Fail: (11) is_verb(some, plural) ? creep
Fail: (10) verb(plural, _G465, [some, pacifist, people, are, evil], _L299) ? creep
Redo: (9) verb_phrase(plural, _G446, [some, pacifist, people, are, evil], []) ?
creep
Call: (10) be_verb(plural, _G465, [some, pacifist, people, are, evil], _L299) ?
creep
Call: (11) is_be_verb(some, plural) ? creep
Fail: (11) is_be_verb(some, plural) ? creep
Fail: (10) be_verb(plural, _G465, [some, pacifist, people, are, evil], _L299) ?
creep
Fail: (9) verb_phrase(plural, _G446, [some, pacifist, people, are, evil], []) ?
creep
Redo: (11) verb_phrase(plural, _G457, [conscript, some, pacifist, people, are,
evil], _L211) ? creep
Call: (12) verb(plural, _G461, [conscript, some, pacifist, people, are, evil],
_L279) ? creep
Call: (13) is_verb(conscript, plural) ? creep
Exit: (13) is_verb(conscript, plural) ? creep
Call: (13) _L279=[some, pacifist, people, are, evil] ? creep
Exit: (13) [some, pacifist, people, are, evil]=[some, pacifist, people, are, evil] ?
creep
Exit: (12) verb(plural, verb(conscript), [conscript, some, pacifist, people, are,
evil], [some, pacifist, people, are, evil]) ? creep
Call: (12) noun_phrase(_L295, _G462, [some, pacifist, people, are, evil], _L211)
? creep
Call: (13) noun(_L295, _G466, [some, pacifist, people, are, evil], _L211) ? creep
Call: (14) is_noun(some, _L295) ? creep
Fail: (14) is_noun(some, _L295) ? creep
Fail: (13) noun(_L295, _G466, [some, pacifist, people, are, evil], _L211) ? creep
Redo: (12) noun_phrase(_L295, _G462, [some, pacifist, people, are, evil],
_L211) ? creep
Call: (13) determiner(_L295, _G466, [some, pacifist, people, are, evil], _L301) ?
creep
Call: (14) is_determiner(some, _L295) ? creep
Exit: (14) is_determiner(some, singular) ? creep
Call: (14) _L301=[pacifist, people, are, evil] ? creep
Exit: (14) [pacifist, people, are, evil]=[pacifist, people, are, evil] ? creep
Exit: (13) determiner(singular, determiner(some), [some, pacifist, people, are,
evil], [pacifist, people, are, evil]) ? creep
Call: (13) noun(singular, _G467, [pacifist, people, are, evil], _L211) ? creep
Call: (14) is_noun(pacifist, singular) ? creep
Fail: (14) is_noun(pacifist, singular) ? creep
Fail: (13) noun(singular, _G467, [pacifist, people, are, evil], _L211) ? creep
Redo: (14) is_determiner(some, _L295) ? creep
Exit: (14) is_determiner(some, plural) ? creep
Call: (14) _L301=[pacifist, people, are, evil] ? creep

```
  Exit: (14) [pacifist, people, are, evil]=[pacifist, people, are, evil] ? creep
  Exit: (13) determiner(plural, determiner(some), [some, pacifist, people, are,
evil], [pacifist, people, are, evil]) ? creep
  Call: (13) noun(plural, _G467, [pacifist, people, are, evil], _L211) ? creep
  Call: (14) is_noun(pacifist, plural) ? creep
  Fail: (14) is_noun(pacifist, plural) ? creep
  Fail: (13) noun(plural, _G467, [pacifist, people, are, evil], _L211) ? creep
  Redo: (12) noun_phrase(_L295, _G462, [some, pacifist, people, are, evil],
_L211) ? creep
  Call: (13) determiner(_L295, _G466, [some, pacifist, people, are, evil], _L302) ?
creep
  Call: (14) is_determiner(some, _L295) ? creep
  Exit: (14) is_determiner(some, singular) ? creep
  Call: (14) _L302=[pacifist, people, are, evil] ? creep
  Exit: (14) [pacifist, people, are, evil]=[pacifist, people, are, evil] ? creep
  Exit: (13) determiner(singular, determiner(some), [some, pacifist, people, are,
evil], [pacifist, people, are, evil]) ? creep
  Call: (13) adjective(singular, _G467, [pacifist, people, are, evil], _L303) ? creep
  Call: (14) is_adjective(pacifist, singular) ? creep
  Exit: (14) is_adjective(pacifist, singular) ? creep
  Call: (14) _L303=[people, are, evil] ? creep
  Exit: (14) [people, are, evil]=[people, are, evil] ? creep
  Exit: (13) adjective(singular, adjective(pacifist), [pacifist, people, are, evil],
[people, are, evil]) ? creep
  Call: (13) noun(singular, _G468, [people, are, evil], _L211) ? creep
  Call: (14) is_noun(people, singular) ? creep
  Fail: (14) is_noun(people, singular) ? creep
  Fail: (13) noun(singular, _G468, [people, are, evil], _L211) ? creep
  Redo: (14) is_determiner(some, _L295) ? creep
  Exit: (14) is_determiner(some, plural) ? creep
  Call: (14) _L302=[pacifist, people, are, evil] ? creep
  Exit: (14) [pacifist, people, are, evil]=[pacifist, people, are, evil] ? creep
  Exit: (13) determiner(plural, determiner(some), [some, pacifist, people, are,
evil], [pacifist, people, are, evil]) ? creep
  Call: (13) adjective(plural, _G467, [pacifist, people, are, evil], _L303) ? creep
  Call: (14) is_adjective(pacifist, plural) ? creep
  Exit: (14) is_adjective(pacifist, plural) ? creep
  Call: (14) _L303=[people, are, evil] ? creep
  Exit: (14) [people, are, evil]=[people, are, evil] ? creep
  Exit: (13) adjective(plural, adjective(pacifist), [pacifist, people, are, evil],
[people, are, evil]) ? creep
  Call: (13) noun(plural, _G468, [people, are, evil], _L211) ? creep
  Call: (14) is_noun(people, plural) ? creep
  Exit: (14) is_noun(people, plural) ? creep
  Call: (14) _L211=[are, evil] ? creep
  Exit: (14) [are, evil]=[are, evil] ? creep
  Exit: (13) noun(plural, noun(people), [people, are, evil], [are, evil]) ? creep
  Exit: (12) noun_phrase(plural, noun_phrase(determiner(some),
adjective(pacifist), noun(people)), [some, pacifist, people, are, evil], [are, evil]) ?
creep
```

```
   Exit: (11) verb_phrase(plural, verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people))), [conscript,
some, pacifist, people, are, evil], [are, evil]) ? creep
   Exit: (10) rel_clause(plural, rel_clause(rel(that), verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people)))), [that,
conscript, some, pacifist, people, are, evil], [are, evil]) ? creep
   Exit: (9) noun_phrase(plural, noun_phrase(determiner(all),
noun(governments), rel_clause(rel(that), verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people))))), [all,
governments, that, conscript, some, pacifist, people, are|...], [are, evil]) ? creep
   Call: (9) verb_phrase(plural, _G446, [are, evil], []) ? creep
   Call: (10) verb(plural, _G476, [are, evil], []) ? creep
   Call: (11) is_verb(are, plural) ? creep
   Fail: (11) is_verb(are, plural) ? creep
   Fail: (10) verb(plural, _G476, [are, evil], []) ? creep
   Redo: (9) verb_phrase(plural, _G446, [are, evil], []) ? creep
   Call: (10) verb(plural, _G476, [are, evil], _L325) ? creep
   Call: (11) is_verb(are, plural) ? creep
   Fail: (11) is_verb(are, plural) ? creep
   Fail: (10) verb(plural, _G476, [are, evil], _L325) ? creep
   Redo: (9) verb_phrase(plural, _G446, [are, evil], []) ? creep
   Call: (10) be_verb(plural, _G476, [are, evil], _L325) ? creep
   Call: (11) is_be_verb(are, plural) ? creep
   Exit: (11) is_be_verb(are, plural) ? creep
   Call: (11) _L325=[evil] ? creep
   Exit: (11) [evil]=[evil] ? creep
   Exit: (10) be_verb(plural, be_verb(are), [are, evil], [evil]) ? creep
   Call: (10) adjective(plural, _G477, [evil], []) ? creep
   Call: (11) is_adjective(evil, plural) ? creep
   Exit: (11) is_adjective(evil, plural) ? creep
   Call: (11) []=[] ? creep
   Exit: (11) []=[] ? creep
   Exit: (10) adjective(plural, adjective(evil), [evil], []) ? creep
   Exit: (9) verb_phrase(plural, verb_phrase(be_verb(are), adjective(evil)), [are,
evil], []) ? creep
   Exit: (8) sentence(plural, sentence(noun_phrase(determiner(all),
noun(governments), rel_clause(rel(that), verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people))))),
verb_phrase(be_verb(are), adjective(evil))), [all, governments, that, conscript,
some, pacifist, people, are|...], []) ? creep
   Call: (8) s(sentence(noun_phrase(determiner(all), noun(governments),
rel_clause(rel(that), verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people))))),
verb_phrase(be_verb(are), adjective(evil))), _G263) ? creep
   Call: (9) gensym(x, _L323) ? creep
   Call: (12) increment_key('$gs_x', _G484) ? creep
   Exit: (12) increment_key('$gs_x', 42) ? creep
   Exit: (9) gensym(x, x42) ? creep
   Call: (9) np(x42, noun_phrase(determiner(all), noun(governments),
rel_clause(rel(that), verb_phrase(verb(conscript),
```

```
noun_phrase(determiner(some), adjective(pacifist), noun(people))))), _L324) ?
creep
  Call: (10) adj(x42, noun(governments), _L346) ? creep
  Call: (11) noun(governments)=..[adjective, _G489] ? creep
  Fail: (11) noun(governments)=..[adjective, _G489] ? creep
  Fail: (10) adj(x42, noun(governments), _L346) ? creep
  Redo: (9) np(x42, noun_phrase(determiner(all), noun(governments),
rel_clause(rel(that), verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people))))), _L324) ?
creep
  Call: (10) n(x42, noun(governments), _L346) ? creep
  Call: (11) noun(governments)=..[noun, _G489] ? creep
  Exit: (11) noun(governments)=..[noun, governments] ? creep
  Call: (11) _L346=..[governments, x42] ? creep
  Exit: (11) governments(x42)=..[governments, x42] ? creep
  Exit: (10) n(x42, noun(governments), governments(x42)) ? creep
  Call: (10) rc(x42, rel_clause(rel(that), verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people)))), _L347) ?
creep
  Call: (11) vp(x42, verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people))), _L347) ?
creep
  Call: (12) gensym(x, _L387) ? creep
  Call: (15) increment_key('$gs_x', _G501) ? creep
  Exit: (15) increment_key('$gs_x', 43) ? creep
  Exit: (12) gensym(x, x43) ? creep
  Call: (12) np(x43, noun_phrase(determiner(some), adjective(pacifist),
noun(people)), _L388) ? creep
  Call: (13) adj(x43, adjective(pacifist), _L410) ? creep
  Call: (14) adjective(pacifist)=..[adjective, _G506] ? creep
  Exit: (14) adjective(pacifist)=..[adjective, pacifist] ? creep
  Call: (14) _L410=..[pacifist, x43] ? creep
  Exit: (14) pacifist(x43)=..[pacifist, x43] ? creep
  Exit: (13) adj(x43, adjective(pacifist), pacifist(x43)) ? creep
  Call: (13) n(x43, noun(people), _L411) ? creep
  Call: (14) noun(people)=..[noun, _G520] ? creep
  Exit: (14) noun(people)=..[noun, people] ? creep
  Call: (14) _L411=..[people, x43] ? creep
  Exit: (14) people(x43)=..[people, x43] ? creep
  Exit: (13) n(x43, noun(people), people(x43)) ? creep
  Call: (13) det(x43, determiner(some), pacifist(x43), people(x43), _L388) ?
creep
  Exit: (13) det(x43, determiner(some), pacifist(x43), people(x43), exists(x43,
people(x43)&pacifist(x43))) ? creep
  Exit: (12) np(x43, noun_phrase(determiner(some), adjective(pacifist),
noun(people)), exists(x43, people(x43)&pacifist(x43))) ? creep
  Call: (12) v(x42, verb(conscript), exists(x43, people(x43)&pacifist(x43)),
_L347) ? creep
  Call: (13) verb(conscript)=..[verb, _G540] ? creep
  Exit: (13) verb(conscript)=..[verb, conscript] ? creep
```

```
  Call: (13) _L347=..[conscript, x42, exists(x43, people(x43)&pacifist(x43))] ?
creep
  Exit: (13) conscript(x42, exists(x43, people(x43)&pacifist(x43)))=..[conscript,
x42, exists(x43, people(x43)&pacifist(x43))] ? creep
  Exit: (12) v(x42, verb(conscript), exists(x43, people(x43)&pacifist(x43)),
conscript(x42, exists(x43, people(x43)&pacifist(x43)))) ? creep
  Exit: (11) vp(x42, verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people))),
conscript(x42, exists(x43, people(x43)&pacifist(x43)))) ? creep
  Exit: (10) rc(x42, rel_clause(rel(that), verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people)))),
conscript(x42, exists(x43, people(x43)&pacifist(x43)))) ? creep
  Call: (10) det(x42, determiner(all), conscript(x42, exists(x43,
people(x43)&pacifist(x43))), governments(x42), _L324) ? creep
  Exit: (10) det(x42, determiner(all), conscript(x42, exists(x43,
people(x43)&pacifist(x43))), governments(x42), all(x42,
governments(x42)&conscript(x42, exists(x43, people(x43)&pacifist(x43))))) ?
creep
  Exit: (9) np(x42, noun_phrase(determiner(all), noun(governments),
rel_clause(rel(that), verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people))))), all(x42,
governments(x42)&conscript(x42, exists(x43, people(x43)&pacifist(x43))))) ?
creep
  Call: (9) vp(x42, all(x42, governments(x42)&conscript(x42, exists(x43,
people(x43)&pacifist(x43)))), verb_phrase(be_verb(are), adjective(evil)),
_G263) ? creep
  Call: (10) gensym(x, _L456) ? creep
  Call: (13) increment_key('$gs_x', _G568) ? creep
  Exit: (13) increment_key('$gs_x', 44) ? creep
  Exit: (10) gensym(x, x44) ? creep
  Call: (10) np(x44, adjective(evil), _L457) ? creep
  Fail: (10) np(x44, adjective(evil), _L457) ? creep
  Redo: (9) vp(x42, all(x42, governments(x42)&conscript(x42, exists(x43,
people(x43)&pacifist(x43)))), verb_phrase(be_verb(are), adjective(evil)),
_G263) ? creep
  Call: (10) be_verb(are)=..[be_verb, are] ? creep
  Exit: (10) be_verb(are)=..[be_verb, are] ? creep
  Call: (10) adj(x42, adjective(evil), _G565) ? creep
  Call: (11) adjective(evil)=..[adjective, _G576] ? creep
  Exit: (11) adjective(evil)=..[adjective, evil] ? creep
  Call: (11) _G565=..[evil, x42] ? creep
  Exit: (11) evil(x42)=..[evil, x42] ? creep
  Exit: (10) adj(x42, adjective(evil), evil(x42)) ? creep
  Exit: (9) vp(x42, all(x42, governments(x42)&conscript(x42, exists(x43,
people(x43)&pacifist(x43)))), verb_phrase(be_verb(are), adjective(evil)),
all(x42, governments(x42)&conscript(x42, exists(x43,
people(x43)&pacifist(x43)))=>evil(x42))) ? creep
  Exit: (8) s(sentence(noun_phrase(determiner(all), noun(governments),
rel_clause(rel(that), verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people))))),
```

verb_phrase(be_verb(are), adjective(evil))), all(x42,
governments(x42)&conscript(x42, exists(x43,
people(x43)&pacifist(x43)))=>evil(x42))) ? creep
Tree = sentence(noun_phrase(determiner(all), noun(governments),
rel_clause(rel(that), verb_phrase(verb(conscript),
noun_phrase(determiner(some), adjective(pacifist), noun(people))))),
verb_phrase(be_verb(are), adjective(evil))),
X = all(x42, governments(x42)&conscript(x42, exists(x43,
people(x43)&pacifist(x43)))=>evil(x42)) .