Finalized on 2009-10-12

Students
Aron HENRIKSSON
Andrei NECULAU

Split Infinitives in Prolog

Stockholm, Sweden
The Royal Institute of Technology
School of Information and Communication Technology

Logic Programming

Course led by Alf Thomas Sjöland

Split Infinitives in Prolog

Aron HENRIKSSON
Andrei NECULAU

Abstract

Henriksson, Aron. Neculau, Andrei. 2009. Split Infinitives in Prolog. Pages 2. Appendix A.
Keywords: split infinitives, logic programming, prolog

2009

Table of Contents

1. Introduction

1.1. Background

Spell checkers have today become integral parts of word processors. They make the user mindful of typos, spelling mistakes, as well as certain grammatical errors. However, when it comes to poor writing style, they usually offer little support. One such object of neglect is split infinitives.

Split infinitives – although not in a strict sense grammatical mistakes – are widely considered to constitute poor style in formal writing. A split infinitive entails the splitting of an infinitive *to <verb>*, typically by a single adverb, e.g. "to boldly go", but also by negation, "to not be" and a number of compounded splits, e.g. "to loudly and clearly speak". Although there are a number of exceptions, in most cases the adverb(s) should go after the infinitive and the negation before it (Oxford University Press, 2003).

The reasons behind the view that split infinitives should be avoided have been widely discussed. An interesting and plausible theory asserts that it is a legacy of Latin, where there is no equivalent to the English infinitive marker *to* – the infinitive is instead a single word, and thus cannot be split. As a result, some feel that the absence of a valid parallel makes the split infinitive concern null and void. However, in formal writing, split infinitives remain largely frowned upon (Wikimedia Foundation, Inc., 2009).

1.2. Problem

Given a chunk of English text, how does one identify the various cases of split infinitives, and how are they aptly corrected? This requires identification of the constituent parts of a sentence, rules to detect the presence of infinitives, rules to identify different types of split infinitive, and the correction of the split infinitives that have been identified.

1.3. Goal

The goal of this project is to identify and correct the most common instances of split infinitives. Given a string of intelligible English text, the Prolog program should:

- Identify the presence of split infinitives
- Identify the types of split infinitive
- Notify the user of the presence of split infinitives
- Provide the user with the corrected text, with the split infinitives removed

1.4. Limitations

Some split infinitives cannot be corrected without major rephrasing. For such cases it is difficult for a program to propose a generalized solution. Likewise, there are always exceptions, which by definition are difficult to create explicit rules for. For instance, in some cases, the adverb belongs before the infinitive rather than after it. Phrasal verbs are also less straightforward to deal with since they also should not be split, e.g. "to get rid of" – here an adverb would need to be placed at the end of the phrase rather than after the infinitive "to get". These cases will not be handled given the time frame of the project.

2. Implementation

2.1. Tokenizer

Given the goal of this project, the input had to be tokenized into words and punctuation marks in order to be identifiable as grammar items. A sentence is a string, and in Prolog a string is represented as a list of characters. Thus, this tokenizer implements a set of rules that parses this list and based on the current character it takes the decision either to append the character to another list (build a temporary word), or to append the temporary word to an output list. This output list in the end contains words and punctuation marks as items.

2.2. Database/Facts

In order to enable the identification of the constituent parts of a sentence, a database in the form of facts was built up. In addition to important elements such as the infinitive marker (“to”), negation (“not”), conjunctions (“and”, “or”, “but”), etc., there are 2377 verbs, 3739 adverbs, and 10 pronouns.

2.3. Split Infinitive Checker

The split infinitive checker assumes a tokenized sentence, where each word or punctuation mark is an element in the list. The rules identifying split infinitives apply only once the infinitive marker *to* is the first element of the list. To reach that condition, there is a recursive rule that simply outputs all words until an infinitive marker is detected. Depending on the number of elements remaining in the list, one of the identification rules will apply. These rules look at the words succeeding the infinitive marker in order to determine the split infinitive type. The words are reordered accordingly and subsequently output, while the rest of the list is recursively examined for further split infinitives.

The user is thus presented with the corrected text, along with a list of detected split infinitives and the corresponding type.

3. Conclusion & Discussion

3.1. Conclusion

Given an extensive dictionary of primarily verbs and adverbs, Prolog can be used to implement a program that effectively detects and corrects split infinitives. Such a program can come in handy when, say, unsure of where the adverb goes in conjunction with an infinitive, or merely to check a formal text for improper usage of infinitives.

3.2. Discussion

When faced with a new programming paradigm, one needs to get accustomed to implementing rules that can output information. For instance, when appending a list to another, the initial list cannot be used as an output list as well. That would imply overwriting the variable, which Prolog apparently does not do, unless the variable is yet to be assigned. Overall memory management falls under different rules compare to more conventional programming languages, and that affects not only the programming style but also the reasoning.

3.3. Future work

In addition to covering more types of split infinitive, a fairly straightforward extension would be to take a text file as input and outputting it with the correction of split infinitives. Incorporating a split infinitive feature in a word processor would be of great benefit for end users.

Bibliography

Oxford University Press. (2003, 05 12). *AskOxford: What is a split infinitive, and why should I avoid using one?* Retrieved 10 11, 2009, from AskOxford.com: http://www.askoxford.com/asktheexperts/faq/aboutgrammar/splitinfinitives

Sterling, L., & Shapiro, E. (1999). *The Art of Prolog.* Cambridge, MA: MIT Press.

Wikimedia Foundation, Inc. (2009, 09 23). *Split Infinitive*. Retrieved 10 11, 2009, from Wikipedia: http://en.wikipedia.org/wiki/Split_infinitive

Appendices

Appendix A – Source Code

Tokenizer

```
tokenize([], Out) :-
            Out = [].
tokenize([X|Xs], Out) :-
            tokenize([X|Xs], [], [], Out1),
            Out = Out1.
% if we ended up with a letter
tokenize([X|Xs], W, S, Out) :-
            letter(X),
            append(W, [X], W1),
            tokenize(Xs, W1, S, Out1),
            Out = Out1.
% if we ended up with a blank
tokenize([X|Xs], W, S, Out) :-
            blank(X),
            append_word(W, S, S1),
            tokenize(Xs, [], S1, Out1),
            Out = Out1.
% if we ended up with a punctuation marker
tokenize([X|Xs], W, S, Out) :-
            punctuation_marker(X),
            append_word(W, S, S1),
            append_word([X], S1, S2),
            tokenize(Xs, [], S2, Out1),
            Out = Out1.
% no character to process; whater word we have, store it, end
tokenize([], W, S, Out) :-
            append_word(W, S, S1),
            Out = S1.
```

```
/*
*
* Utility rules
*
*/
% whitespace
blank(C) :-
            C = 32.
% comma, period, exclamation mark, question mark, colon, semicolon
punctuation_marker(C) :-
            C = 44;
            C = 46;
            C = 33;
            C = 63;
            C = 58;
            C = 59.
% letter = non-whitespace non-punctuation character
letter(C) :-
            \+ blank(C),
            \+ punctuation_marker(C).
% append non-empty word (transforms from list of char codes to atom)
append_word([], S, S1) :-
            append([], S, S1).
append_word(W, S, S1) :-
            \+ W = [],
            atom_codes(G, W),
            append(S, [G], S1).
```

Split Infinitive Checker

```
/*
* Parsing sentence input as list.
*/
% 0 elements: To avoid 'no' when list has been processed with detected split infinitives.
is_split_infinitive([]).
% 1 element: When only one word is left to process, simply output it.
is_split_infinitive([X]) :-
            print(X).
% 2 elements: Deals with non-split infinitives - will actually be true when there is no split infinitive.
is_split_infinitive([X,Y]) :-
            infinitive_marker(X),
            (verb(Y);
            pronoun(Y)),
            print_list([X,Y]),
            print('\nDetected: correct use of infinitive/pronoun: '),
            print_list([X,Y]).
% 3 elements: detects split infinitive (negation and single adverb).
is_split_infinitive([X,Y,Z]) :-
            infinitive_marker(X),
            \+ verb(Y),
            \+ pronoun(Y),
            (negation(Y), verb(Z),
            print_negation_phrase([X,Y,Z]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z]),
            print('- type: negation'));
            (adverb(Y), verb(Z),
            print_adverb_phrase([X,Y,Z]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z]),
            print('- type: single ddverb')).
% 3 elements: Else case - if no split infinitive, output words.
is_split_infinitive([X,Y,Z]) :-
            infinitive_marker(X),
            (verb(Y);
            pronoun(Y)),
```

```
            print_list([X,Y,Z]),
            print('\nDetected: correct use of infinitive/pronoun: '),
            print_list([X,Y]).
% 4 elements: Detects split infinitive (above + combinations of negation and
adverb).
is_split_infinitive([X,Y,Z,A]) :-
            infinitive_marker(X),
            \+ verb(Y),
            \+ pronoun(Y),
            (negation(Y), verb(Z),
            print_negation_phrase([X,Y,Z]),
            print(A),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z]),
            print('- type: negation'));
            (adverb(Y), verb(Z),
            print_adverb_phrase([X,Y,Z]),
            print(A),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z]),
            print('- type: single adverb'));
            (negation(Y), adverb(Z), verb(A),
            print_negation_adverb_phrase([X,Y,Z,A]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A]),
            print(' - type: negation and adverb'));
            (adverb(Y), negation(Z), verb(A),
            print_adverb_negation_phrase([X,Y,Z,A]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A]),
            print('- type: adverb and negation')).
% 4 elements: Else case - if no split infinitive, output words.
is_split_infinitive([X,Y,Z,A]) :-
            infinitive_marker(X),
            (verb(Y);
            pronoun(Y)),
            print_list([X,Y,Z,A]),
            print('\nDetected: correct use of infinitive/pronoun: '),
            print_list([X,Y]).
% 5 elements: Detects split infinitive (above + adverbial conjunction).
```

```
is_split_infinitive([X,Y,Z,A,B]) :-
            infinitive_marker(X),
            \+ verb(Y),
            \+ pronoun(Y),
            % Outputs correction and keeps searching for more split infinitives
            (negation(Y), verb(Z),
            print_negation_phrase([X,Y,Z]),
            is_split_infinitive([A,B]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z]),
            print('- type: negation'));
            (adverb(Y), verb(Z),
            print_adverb_phrase([X,Y,Z]),
            is_split_infinitive([A,B]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z]),
            print('- type: single adverb'));
            (negation(Y), adverb(Z), verb(A),
            print_negation_adverb_phrase([X,Y,Z,A]),
            print(B),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A]),
            print('- type: negation and adverb'));
            (adverb(Y), negation(Z), verb(A),
            print_adverb_negation_phrase([X,Y,Z,A]),
            print(B),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A]),
            print('- type: adverb and negation'));
            (adverb(Y), conjunction(Z), adverb(A), verb(B),
            print_conjunction_phrase([X,Y,Z,A,B]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A,B]),
            print('- type: adverbial conjunction')).
% 5 elements: Else case - if no split infinitive, output infinitive and keep
searching for split infinitives.
is_split_infinitive([X,Y,Z,A,B]) :-
            infinitive_marker(X),
            (verb(Y);
            pronoun(Y)),
```

```
            print_list([X,Y]),
            is_split_infinitive([Z,A,B]),
            print('\nDetected: correct use of infinitive/pronoun: '),
            print_list([X,Y]).
% 6+ elements: Detects split infinitives (above + negations in adverbial conjunction; 'neither nor').
is_split_infinitive([X,Y,Z,A,B,C|Cs]) :-
            infinitive_marker(X),
            \+ verb(Y),
            \+ pronoun(Y),
            % Outputs correction and keeps searching for more split infinitives.
            (negation(Y), verb(Z),
            print_negation_phrase([X,Y,Z]),
            is_split_infinitive([A,B,C|Cs]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z]),
            print('- type: negation'));
            (adverb(Y), verb(Z),
            print_adverb_phrase([X,Y,Z]),
            is_split_infinitive([A,B,C|Cs]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z]),
            print('- type: single adverb'));
            (negation(Y), adverb(Z), verb(A),
            print_negation_adverb_phrase([X,Y,Z,A]),
            is_split_infinitive([B,C|Cs]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A]),
            print('- type: negation and adverb'));
            (adverb(Y), negation(Z), verb(A),
            print_adverb_negation_phrase([X,Y,Z,A]),
            is_split_infinitive([B,C|Cs]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A]),
            print('- type: adverb and negation'));
            (adverb(Y), conjunction(Z), adverb(A), verb(B),
            print_conjunction_phrase([X,Y,Z,A,B]),
            is_split_infinitive([C|Cs]),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A,B]),
```

```
            print('- type: adverbial conjunction'));
            (negation(Y), adverb(Z), conjunction(A), adverb(B), verb(C),
            print_negation_conjunction_phrase([X,Y,Z,A,B,C]),
            is_split_infinitive(Cs),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A,B,C]),
            print('- type: negation and adverbial conjunction'));
            (adverb(Y), conjunction(Z), negation(A), adverb(B), verb(C),
            print_conjunction_negation_phrase([X,Y,Z,A,B,C]),
            is_split_infinitive(Cs),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A,B,C]),
            print('- type: adverbial and negation conjunction'));
            (neither(Y), adverb(Z), nor(A), adverb(B), verb(C),
            print_neither_nor_conjunction_phrase([X,Y,Z,A,B,C]),
            is_split_infinitive(Cs),
            print('\nDetected: split infinitive: '),
            print_list([X,Y,Z,A,B,C]),
            print('- type: neither nor adverbial conjunction')).
% 6+ elements: Else case - if no split infinitive, output infinitive and keep
searching for split infinitives.
is_split_infinitive([X,Y,Z,A,B,C|Cs]) :-
            infinitive_marker(X),
            (verb(Y);
            pronoun(Y)),
            print_list([X,Y]),
            is_split_infinitive([Z,A,B,C|Cs]),
            print('\nDetected: correct use of infinitive/pronoun: '),
            print_list([X,Y]).
% Recursive function: look for infinitive marker ('to'); output preceding words.
is_split_infinitive([X|Xs]) :-
            \+ infinitive_marker(X),
            print(X),
            print(' '),
            is_split_infinitive(Xs).
```

```
/*
* Output rules
*/
% When all elements have been output.
print_list([]).
% Output all elements in the list recursively.
print_list([X|Xs]) :-
            print(X),
            print(' '),
            print_list(Xs).
% Correction of negation, e.g. "to not be" becomes "not to be".
print_negation_phrase([X,Y,Z]) :-
            print_list([Y,X,Z]).
% Correction of adverbial conjunctions (and/or), e.g. "to loudly and clearly speak" becomes "to speak loudly and clearly".
print_conjunction_phrase([X,Y,Z,A,B]) :-
            print_list([X,B,Y,Z,A]).
% Correction of single adverb, e.g. "to boldly go" becomes "to go boldly".

print_adverb_phrase([X,Y,Z]) :-
            print_list([X,Z,Y]).
% Correction of negation followed by single adverb, e.g. "to not rashly act".
print_negation_adverb_phrase([X,Y,Z,A]) :-
            print_list([Y,X,A,Z]).
% Correction of single adverb followed by negation, e.g. "to rashly not act".
print_adverb_negation_phrase([X,Y,Z,A]) :-
            print_list([Z,X,A,Y]).
% Correction of negation followed by adverbial conjunction, e.g. "to not rashly or stupidly act".
print_negation_conjunction_phrase([X,Y,Z,A,B,C]) :-
            print_list([Y,X,C,Z,A,B]).
% Correction of adverbial conjunction with a negation, e.g. "to rashly but not stupidly act".
print_conjunction_negation_phrase([X,Y,Z,A,B,C]) :-
            print_list([X,C,Y,Z,A,B]).
% Correction of neither nor in an adverbial conjunction, e.g. "to neither rashly nor stupidly act".
print_neither_nor_conjunction_phrase([X,Y,Z,A,B,C]) :-
            print_list([X,C,Y,Z,A,B]).
```