

Exercise 7: Logic and Inference Variant 4

Introduction to Artificial Intelligence

Report

AAYUSH GUPTA

WYPYCH DAWID

Task :

Convert English words of a written number (up to a thousand) into numerical digits N , where $N \leq 1000$.

Description:

The solution to converting written numbers to digits in Prolog follows a logical flow that can be summarized as follows:

1. The program defines a set of facts representing the mapping between written numbers and their digit representations. For example, the fact `word(zero, 0)` states that the word "zero" corresponds to the digit 0.
2. The main predicate, `word_to_digit`, is defined to convert a given written number to its digit representation. It takes the written number as input and a variable to store the resulting digits.
3. The `word_to_digit` predicate first splits the written number into individual words using the `atomic_list_concat` predicate.
4. The `convert_words` predicate is called to recursively process the list of words and accumulate the digit representation. It takes

three arguments: the list of remaining words, the accumulated digit value, and the final digit representation.

5. The **convert_words** predicate has various rules to handle different cases based on the structure of the written number. It checks the words in the list and applies the corresponding conversion rules to update the digit value accordingly.
6. The conversion rules handle cases such as single words representing digits, compound words for numbers up to 99, the word "hundred" for hundreds, and the word "thousand" for thousands.
7. The digit representation is finally returned as the output of the **word_to_digit** predicate.

Main Components

```
% Rule to convert a written number to digits  
word_to_digit(WrittenNumber, Digits) :-  
    atomic_list_concat(Words, ' ', WrittenNumber), % Split the written number into individual words  
    convert_words(Words, 0, Digits). % Call the helper predicate to convert the words to digits
```

word_to_digit Predicate: This predicate is the entry point of the program. It takes the written number as input and the variable to store the resulting digits. It splits the written number into words and calls the **convert_words** predicate to perform the conversion.

```

% Rule to convert a list of words to digits
convert_words([], N, N). % Base case: if the list is empty, return the accumulated value
convert_words([Word], N, Digits) :-
    word(Word, Digit), % Lookup the numeric value associated with the word
    Digits is N + Digit. % Add the value to the accumulated sum
convert_words([hundred | Rest], N, Digits) :-
    convert_words(Rest, N * 100, Digits). % Multiply the accumulated sum by 100 (hundred)
convert_words([thousand | Rest], N, Digits) :-
    convert_words(Rest, N * 1000, Digits). % Multiply the accumulated sum by 1000 (thousand)
convert_words([Word1, and, Word2 | Rest], N, Digits) :-
    word(Word1, Value1), % Lookup the numeric value of the first word
    word(Word2, _), % Ignore the value of the second word
    NewN is N + Value1, % Add the value of the first word to the accumulated sum
    convert_words(Rest, NewN, Digits). % Recursively process the remaining words
convert_words([Word1, Word2 | Rest], N, Digits) :-
    word(Word1, Value1), % Lookup the numeric value of the first word
    word(Word2, _), % Ignore the value of the second word
    NewN is N + Value1, % Add the value of the first word to the accumulated sum
    convert_words([Word2 | Rest], NewN, Digits). % Recursively process the remaining word
convert_words([Word1, Word2, Word3 | Rest], N, Digits) :-
    word(Word1, Value1), % Lookup the numeric value of the first word
    word(Word2, Value2), % Lookup the numeric value of the second word
    word(Word3, _), % Ignore the value of the third word
    NewN is N + Value1 * 100 + Value2, % Add the product of the first word and 100 to the
    convert_words(Rest, NewN, Digits). % Recursively process the remaining words

```


convert_words/ Predicate: This predicate recursively processes the list of words and accumulates the digit representation. It applies different conversion rules based on the structure of the written number to update the digit value.

Test Cases and Results:

 `word_to_digit("eight hundred sixty one", Digits).`


Digits = 861

Next 10 100 1,000 Stop

 `word_to_digit("nine hundred ninety nine", Digits).`

Digits = 999

Next 10 100 1,000 Stop

 `word_to_digit("two hundred ninety two", Digits).`


Digits = 292

Next 10 100 1,000 Stop

 `word_to_digit("two", Digits).`


Digits = 2

Next 10 100 1,000 Stop

 `word_to_digit("two hundred thirty", Digits).`

Digits = 230

Next 10 100 1,000 Stop

 `word_to_digit("one hundred twenty five", Digits).`

Digits = 125

Next 10 100 1,000 Stop

?- `word_to_digit("one hundred twenty five", Digits).`

```
?- word_to_digit('twenty five', Digits).  
Digits = 25.
```

```
?- word_to_digit('ninety three', Digits).  
Digits = 93.
```

```
?- word_to_digit('eighty four', Digits).  
Digits = 84.  
?- word_to_digit('three hundred', Digits).  
Digits = 300.
```

```
?- word_to_digit('seven hundred fifty', Digits).  
Digits = 750.
```

Conclusion :

In conclusion, the `word_to_digit` predicate effectively converts written numbers into their corresponding digit representations in Prolog. The solution utilizes predefined facts for word-to-digit mappings and follows a logical flow to handle different cases, including compound numbers and conjunctions. Despite the challenges faced, such as ordering and special cases, the implementation successfully converts written numbers to digits. The provided test cases validate the functionality and correctness of the predicate. Overall, the solution offers a reliable and reusable approach for number conversion tasks in Prolog, making it a valuable tool for various applications.