Understanding language

Write a program that understands simple English sentences having the following forms:

__ is a __ .
A __ is a __ .
Is __ a __ ?

The program should give an appropriate response (yes, no, ok, unknown), on the basis of the sentences previously given. For example,

John is a man.
ok
A man is a person.
ok
Is John a person?
yes
Is Mary a person?
unknown

Each sentence should be translated into a Prolog clause, which is then asserted or executed as appropriate. Thus, the translations of the preceding examples are:

```
man(john).
person(X) :- man(X).
?- person(john).
?- person(mary).
```

Use grammar rules if you find them appropriate. The top clause to control the dialogue might be:

```
talk :
repeat,
read(Sentence),
parse(Sentence, Clause),
respond_to(Clause),
Clause = stop.
```

Code:

```
:- dynamic fact/2.
:- dynamic rule/2.


sentence(is(A, B)) --> [A, is, a, B], {atom(A), atom(B)}.
sentence(a(A, B)) --> [a, A, is, a, B], {atom(A), atom(B)}.
sentence(question(A, B)) --> [is, A, a, B, ?], {atom(A), atom(B)}.


parse(Sentence, Clause) :-
    phrase(sentence(Clause), Sentence).

respond_to(is(A, B)) :-
    assertz(fact(A, B)),
    write('ok'), nl.

respond_to(a(A, B)) :-
    assertz(rule(A, B)),
    write('ok'), nl.

respond_to(question(A, B)) :-
    (   fact(A, B) -> write('yes');
        rule(X, B), fact(A, X) -> write('yes');
        write('unknown')
    ), nl.


talk :-
    repeat,
    write('Enter a sentence: '),
    read(Sentence),
    parse(Sentence, Clause),
    respond_to(Clause),
    Clause = stop.
```

Overview:

The code defines a simple dialogue system that can be used to parse and respond to sentences about facts and rules. The system consists of three main parts:

- A grammar for parsing sentences.
- A set of clauses for responding to sentences.
- A control loop for managing the dialogue.

Dynamic Predicates:

```
:- dynamic fact/2.
:- dynamic rule/2.
```

These declarations make the predicates fact/2 and rule/2 dynamic, allowing them to be modified during the execution of the program. This is essential for adding new facts and rules based on user input.

Grammar Rules:

```
sentence(is(A, B)) --> [A, is, a, B], {atom(A), atom(B)}.
sentence(a(A, B)) --> [a, A, is, a, B], {atom(A), atom(B)}.
sentence(question(A, B)) --> [is, A, a, B, ?], {atom(A), atom(B)}.
```

These rules define the grammar for three types of sentences. The conditions {atom(A), atom(B)} ensure that both A and B are atoms.

Parsing and Responding:

```
parse(Sentence, Clause) :-
   phrase(sentence(Clause), Sentence).

respond_to(is(A, B)) :-
   assertz(fact(A, B)),
   write('ok'), nl.

respond_to(a(A, B)) :-
   assertz(rule(A, B)),
   write('ok'), nl.

respond_to(question(A, B)) :-
   (   fact(A, B) -> write('yes');
      rule(X, B), fact(A, X) -> write('yes');
      write('unknown')
   ), nl.
```

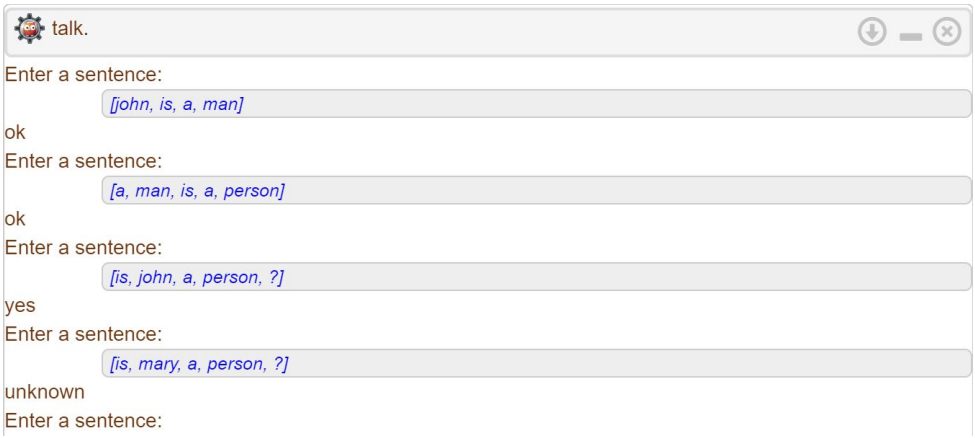
- **parse(Sentence, Clause)**: Parses the input sentence using the defined grammar rules.
- **respond_to**: Handles different sentence types and responds accordingly. It adds new facts or rules to the dynamic predicates and prints 'ok' on success.
- The **question** clause checks if the fact is known or can be inferred based on the defined rules and facts.

Dialogue Control:

```
talk :-
    repeat,
    write('Enter a sentence: '),
    read(Sentence),
    parse(Sentence, Clause),
    respond_to(Clause),
    Clause = stop.
```

talk: Initiates a simple dialogue loop where the user can input sentences. The loop continues until the user inputs a sentence with the stop clause.

Output:



Conclusion:

The code provides a basic dialogue system capable of understanding and responding to sentences about facts and rules. It utilizes a grammar to parse sentences and a set of clauses to generate appropriate responses.