Memoization and Chart Parsing

September 22, 2014

CPSC470/570 Artificial Intelligence
Prof. Drew McDermott
Classic Example

Fibonacci function:

```
define fib(i)
    if i < 3
        then 1
    else fib(i-1) + fib(i-2)
```

Looks harmless, but fib(i-1) calls fib(i-2): You end up with an exponential number of subroutine calls.
Fibonacci in Prolog

fib(I, 1) :- I < 3.

fib(I, N3) :-
    I >= 3, I1 is I-1, I2 is I-2,
    fib(I1,N1), fib(I2,N2),
    N3 is N1 + N2.
Cure: Add a table

Change \texttt{fib} so it (a) stores answers in a table; and (b) it consults the table before doing any computation:

\begin{verbatim}
define fib_table = empty hash table

define fib(i)
    if fib_table(i) exists
        then fib_table(i)
    else let result = fib(i-1) + fib(i-2)
        {
            fib_table(i) = result;
            result
        }
\end{verbatim}
fib Has Been *Memoized*

(Cf. *dynamic programming.*)

*But how would you do this in Prolog??*

Any change you attempt would be undone on backtracking.

Cure: Run-time use of Prolog “database” (where the rules are stored).
The Database

To add a fact to the database: `assertz(p)`.

To check it, just write `p`. (In extreme cases, `clause(p, true)`.)
Memoized Fibonacci in Prolog

fib_mem(I, 1) :- I < 3.

fib_mem(I, N) :-
    I >= 3, fib_table(I, N).
fib_mem, Part 2

fib_mem(I, N) :-
    I >= 3,
    \+ fib_table(I, _),
    I1 is I-1, I2 is I-2,
    fib_mem(I1, N1),
    fib_mem(I2, N2),
    N is N1 + N2,
    assertz(fib_table(I,N)).
Complication

\(\text{fib}\) can’t fail, and always returns exactly one value. (It’s deterministic.)

This is not true in general of Prolog programs. If all we do is record the answers we got for a goal, we can’t distinguish between not having seen the goal yet and having seen it but failing to find any answers.
Two Tables

So we need two tables, one for goals that have been attempted, and one for the answers obtained.

(What do we do if we try goal $G$, check the “attempts” table, and discover that an attempt is still in progress? This gets tricky....)
What Goes In The “Attempts” Table?

Suppose we have a goal `?- fib_mem(28, 317812)`. We don’t check the attempted-goal table for this goal literally, but for any goal of the form `fib_mem(28, _)`, because you can consider it to be shorthand for

`?- fib_mem(28, N), N is 317812.`
Tabling in Prolog

The study of general-purpose *tabling* is an active area of Prolog research. In XSB Prolog (but not in SWI-Prolog), one can write

```prolog
:- tabled friend/2.

friend(X,Y) :- friend(X,Z), friend(Z,Y).
friend(X,Y) :- friend(Y,X).
friend(pablo, maria).
friend(juan, maria).
```

and infer `friend(pablo, juan)` without further ado.
Application to Parsing

DCGs implement a *top-down* parsing strategy: Attempt to find a category $C$ spanning a string by looking for categories $C_1, \ldots, C_k$ licensed by a rule $C \rightarrow C_1, \ldots, C_n$.

A *bottom-up parser* concludes that a string is a $C$ after it has found substrings labeled $C_1$ to $C_n$. 
Chart Parsing

a.k.a. the Earley algorithm, the CYK algorithm, left-corner parsing, head-corner parsing, ....

Idea: Memoize the operations of a top-down, bottom-up, or some other kind of parser.