Islamic University of Technology

Artificial Intelligence Lab

CSE 4712

Lab 4

Author:
Ishmam Tashdeed
CSE, IUT

Contents

1 Tasks

1.1 Task 1

Introduce a recursive definition named `sumto` that takes a limit to find the sum of all odd numbers upto the given limit.

Sample Input:

```
sumto(5, N).
```

Sample Output:

```
N=9
```

1.2 Task 2

In Artificial Intelligence, data often arrives in complex, nested structures (lists within lists), but algorithms often require a simple linear sequence. Your task is to write a Prolog predicate `flatten_list(NestedList, FlatList)`. This predicate should take a list of lists and return a single combined list.

Sample Input:

```
flatten_list([[a, b], [c], [b, d, e]], X).
```

Sample Output:

```
X=[a, b, c, b, d, e]
```

1.3 Task 3

Prolog naturally attempts to find every possible solution by backtracking. However, in mathematical functions like Factorial, once we reach the base case ($N = 1$), there is no need to search further. The standard definition of factorial below:

```
% Base case
factorial(1, 1).

factorial(N, Result) :-
N > 1,
N1 is N - 1,
factorial(N1, R),
Result is R * N.
```

Rewrite this predicate to include the Cut (!) operator in the base case. This should tell Prolog to stop searching for alternatives once it successfully reaches $N = 1$.

Sample Input:

```
factorial(5, X).
```

Sample Output:

```
X=120
```

1.4 Task 4

Pathfinding is a core AI concept. If given a simple directed graph such as:

```
edge(a, b).
edge(b, c).
edge(c, d).
edge(d, e).
edge(c, e).
```

Write a predicate `find_path` that finds the path from a start node to the goal node. It should return `false` if no path exists.

Sample Input:

```
find_path(a, c, X).
find_path(e, a, X)
```

Sample Output:

```
X=[a, b, c]
X=false
```

1.5 Task 5

Simple recursive definitions of pathfinding often fail when a **undirected graph** contains "cycles" (e.g., A connects to B, and B connects back to A), causing the program to enter an infinite loop. Write a predicate `path(Start, End, Path)` that avoids cycles by keeping track of visited nodes if given a simple directed graph such as:

```
edge(a, b).
edge(b, c).
edge(c, d).
edge(d, b).
edge(c, e).
```

It should return `false` if no path exists.

Sample Input:

```
path(a, e, X).
```

Sample Output:

```
X=[a, b, c, e]
```