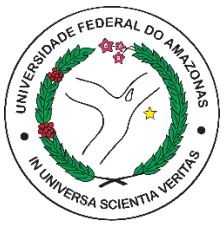

Universidade Federal do Amazonas

Faculdade de Tecnologia FT

Atividade assicrona valendo presença e nota

Manaus, outubro de 2024

Alunos:

- Thiago Rodrigo Monteiro Salgado – 21954456
- João Victor de Carvalho Marques – 21952894

Professor: Edjard

Introdução

Este trabalho tem como intuito resolver o exercício 21.1 (página 500) e da página 514 do livro de Ivan Bratko [1]. No primeiro exercício pede-se que realize um experimento modificando o conjunto de exemplos sobre a relação `has_daughter` e observar como essas modificações afetam os resultados. Já no segundo exercício. Já no segundo vamos utilizar o sistema `HYPER` para aprendizado de lógica indutiva, aprendendo os predicados odd (L) e even (L) simultaneamente.

O trabalho mostrará os passos detalhados para a execução do exercício e os resultados obtidos, servindo como uma introdução prática à linguagem.

Exercício 21.1

O exercício, visto na Figura 1, propõe que executemos o código disponibilizado alterando as informações iniciais. A ideia é de que possamos observar as mudanças que irão acarretar no resultado. Aqui estamos utilizando o conceito de `MINIHYPER e HYPER`, que são sistemas de aprendizado usados em Programação Lógica Indutiva (ILP) para gerar e refinar hipóteses lógicas a partir de exemplos. `HYPER` é mais avançado e eficiente, lidando com problemas complexos e múltiplos predicados, enquanto `MINIHYPER` é uma versão simplificada, adequada para tarefas introdutórias e regras básicas. Ambos ajudam a automatizar o aprendizado de regras lógicas em Prolog.

Para este primeiro exercício, será utiilzado `MINIHYPER`.

EXERCISE

21.1 Experiment with MINIHYPER with modified sets of examples about **has_daughter**. How do these modifications affect the results?

Let us now consider a slightly more complicated learning exercise: learning the predecessor relation, using the same background knowledge as in Figure 21.1. This is more difficult because it requires a recursive definition, but this is exactly what ILP enables. We may define some positive and negative examples as follows:

```
ex( predecessor( pam, bob)).
ex( predecessor( pam, jim)).
ex( predecessor( tom, ann)).
ex( predecessor( tom, jim)).
ex( predecessor( tom, liz)).

nex( predecessor( liz, bob)).
nex( predecessor( pat, bob)).
nex( predecessor( pam, liz)).
nex( predecessor( liz, jim)).
nex( predecessor( liz, liz)).
```

'Guessing' that our target hypothesis comprises two clauses, we may define a start hypothesis as:

```
start_hyp( [ [predecessor(X1,Y1)] / [X1,Y1],
             [predecessor(X2,Y2)] / [X2,Y2] ] ).
```

The relevant background predicates are:

```
backliteral( parent(X,Y), [X,Y]).
backliteral( predecessor(X,Y), [X,Y]).
prolog_predicate( parent(X,Y)).
```

Figura 1 - questão retirada do livro de Ian Bratko

Como solicitado na questão, foram adicionados os exemplos positivos e negativos, a hipótese incial e o background, juntamente do código disponibilizado pelo professor na sala de aula online. A inserção no código pode ser visto na Figura 2.

```
%-----------------------------------------------------------
% Exemplos positivos (ex/1)
ex(predecessor(pam, bob)).
ex(predecessor(pam, jim)).
ex(predecessor(tom, ann)).
ex(predecessor(tom, liz)).

% Exemplos negativos (nex/1)
nex(predecessor(liz, bob)).
nex(predecessor(pat, bob)).
nex(predecessor(pam, liz)).
nex(predecessor(liz, jim)).

%-----------------------------------------------------------
% Predicados de fundo
prolog_predicate(parent/2).

% Definições do predicado parent/2
parent(pam, ann).
parent(ann, jim).
parent(tom, liz).
parent(tom, ann).
parent(pam, bob).
parent(bob, jim).

%-----------------------------------------------------------
% Hipótese inicial
start_hyp([ [predecessor(X1,Y1)] / [X1,Y1],
            [predecessor(X2,Y2)] / [X2,Y2] ]).

%-----------------------------------------------------------
% Definir backliteral/2
```

Figura 2 - Inserção de elementos solicitados pelo exercício.

Resultados do exercício 21.1

O código realizado não chega a uma conclusão, como pode ser visto no console na Figura 3, e acaba em um aprendizado praticamente infinito, sem um resultado consistente de fato. Ou seja, não foi possível encontrar uma hipótese. O poderia ser feito ainda seria o de aumentar a complexidade da primeira hiótese ou adicionar mais predicados ao background. Adicionar esses componentes podem fazer com que o código ganhe uma maior robustez e gere um resultado conclusivo.

Figura 3 - Tentativa de encontrar um resultado

Exercício Learning 2 predicate odd and Even

Neste exercício utilizaremos `HYPER` pois lidaremos com um problema mais complexo. Em teoria devemos realizar o aprendizado simultãneo de dois predicados:

- Odd (L) e;
- Even (L).

Entenda ambos como lista de cumprimentos ímpares e pares que vão utilizar deste sistema mais robusto, que é o HYPER para lidar com a maior complexidade do nosso problema. Temos aqui basicamente duas listas: uma de exemplos positivos (ex/1) par ou ímpar e outra de exemplos negativos (nex/1) de tamanho diferente da primeira lista. O HYPER gera 85 hipóteses, das quais 16 são refinadas e 29 permanecem para serem refinadas.

Resultados do Learning 2

Na tentativa de execução do código, não foi possível estabelecer a totalidade de seu treino, ou seja, os predicados e fundos não estavam ajudando como um todo. Mesmo após checagens das listas, exige tempo para desenvolver o HYPE, que se mostraria mais eficiente em relação ao MINI. Segue na Figura 4 os resultados obtidos. Com podemos ver, ele supera os dados vazios indicando que é par mas para o próximo ele já não consegue ultrapassar e acaba ficando preso.

```
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Falhou ao provar: even([a,b])
Warning: /mnt/c/Users/jhonm/Documents/Codigos/Faculdade/IA/IA Atividade Bratku 21.1/IA-UFAM/trabalho_23_10/hyper.pl:190:
Warning:    Goal (directive) failed: user:(induce(_17590),write('Hipótese final aprendida: '),nl,show_hyp(_17590))
Warning: /mnt/c/Users/jhonm/Documents/Codigos/Faculdade/IA/IA Atividade Bratku 21.1/IA-UFAM/trabalho_23_10/hyper.pl:215:
Warning:    Singleton variables: [Type]
true.
```

Figura 4 - resultados do exercício 2

Referências

[1] – Bratko, I. (2011). Prolog Programming for Artificial Intelligence (4th ed.). Pearson Education.