Segundo Trabalho de IA (Busca Heurística em Grafos *OU*) (Prof. Alexandre Direne - 2015/2)

Histórico das revisões deste enunciado
16/12 - Ampliado o prazo de entrega e flexibilizada a linguagem de implementação.
28/11 - Alterada a saída e o valor-verdade do predicado principal.
20/11 - Extensão do prazo de entrega.
18/11 - Simplicação dos formatos das fórmulas para sempre em CNF.
18/11 - Várias correções pequenas do texto.
06/11 - Simplicação dos nomes dos átomos (x1, x2, ..., xN).
04/11 - Divulgação do enunciado original.

Atenção:

Este trabalho é obrigatório e deverá ser entregue até o dia 21 de dezembro de 2015 (segunda-feira), com mais extensões possíveis. A solução é individual e deverá ser arquivada no diretório "~alexd/IA/" onde o nome do arquivo terá como prefixo o seu nome-de-usuário no sistema do laboratório e, como extensão, ".pl" para indicar que seu conteúdo possui um programa em Prolog. Assim, por exemplo, se o seu nome de usuário no sistema fosse "grs12" então o nome o arquivo seria "grs12.pl" (dentro do diretório "~alexd/IA/"). Não se esqueça de proteger completamente o arquivo criado, de maneira a permitir a leitura do mesmo apenas por você! Isso pode ser feito aplicando `chmod og-rwx grs12.pl` antes de efetuar a cópia com a preservação das permissões (`cp -p grs12.pl ~alexd/IA/`). Não se preocupe com as permissões do professor que irá corrigir o trabalho. A correção dos trabalhos será parcialmente automatizada, sendo assim, é importante que todos os arquivos com as soluções individuais estejam no diretório citado acima, dentro do prazo estipulado. Não será permitida a entrega do arquivo por e-mail.

Enunciado:

Implementar um predicado unário em linguagem Prolog, chamado `sat`, que realiza a tarefa computacional de decisão chamada de cálculo de satisfatibilidade buleana (boolean satisfiability), abreviadamente, `SAT`. Ver detalhes em:

`https://en.wikipedia.org/wiki/Boolean_satisfiability_problem`

O predicado deve ser implementado por meio de um dos algoritmos de busca heurística de Satisfação de Restrições (ver bibliografia abaixo). Em outras palavras, não é necessária a aplicaccão de um algorítmo específico para resolver com suas várias componentes de contribuição científica. No entanto, é recomendável que ao menos alguma calibragem heurística assim como a inclusão de retroação (`backtracking`) não cronológica sejam implementadas.

O predicado deverá responder apenas verdadeiro ou falso para qualquer instância de problema. A fórmula dada será sempre sintaticamente correta, ou seja, nenhuma verificação de erro sintático precisa ser feita. Além disso, a fórmula estará sempre no `forma normal conjuntiva` (CNF - Conjunctive Normal Form). Ver detalhes em:

`https://en.wikipedia.org/wiki/Conjunctive_normal_form`

Para ilustrar a aplicação do predicado `sat`, consedere a seguinte fórmula proposicional (que é insatisfatível):

$$(x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

Para isso, deve bastar a carga de seu código-fonte em Prolog e a execução da consulta assim:

```
?- sat("(x1#x2#x3)&(x1#x2#~x3)&(x1#~x2#x3)&(x1#~x2#~x3)&(~x1#x2#x3)&(~x1#x2#~x3)&(~x1#~x2#x3)&(~x1#~x2#~x3)").
Tempo de execucao = 0.37 segundo(s).
Instanciacoes = 4.
Unsat
yes
```

Um outro exemplo também insatisfatível seria para a seguinte fórmula proposicional:

$$x \wedge \neg x$$

Para essa, a consulta seria:

```
?- sat( " x1 & ~x1 " ).
Tempo de execucao = 0.03 segundo(s).
Instanciacoes = 2.
Unsat
yes
```

Veja mais um exemplo de fórmula:

$$(x \vee x \vee x) \wedge (\neg x \vee \neg x \vee \neg x)$$

Para essa, a consulta seria:

```
?- sat( " (x1#x1#x1) & (~x1#~x1#~x1) " ).
Tempo de execucao = 0.07 segundo(s).
Instanciacoes = 2.
Unsat
yes
```

Veja também um exemplo de fórmula satisfatível, cuja consulta é:

```
?- sat("(x1#x2#x3)&(x2#x3#x4)&(x3#x4#x5)&(x4#x5#x6)&(x5#x6#x7)&(x6#x7#x8)&(x7#x8#x9)&(x8#x9#x10)").
Tempo de execucao = 0.83 segundo(s).
Instanciacoes = 8.
Sat
yes
```

O único termo de `sat` é definido por meio de um `string` especial do Prolog o qual representa uma fórmula sintaticamente bem formada em Lógica Proposicional. Tal `string` é automaticamente convertido pelo Prolog para uma lista de números inteiros onde cada um representa o valor da escala ASCII de um elemento da fórmula. Tais elementos podem ser de 9 (nove) tipos:

126: operador de negação ($\neg$);

38: operador de interseção ($\wedge$);

35: operador de união ($\vee$);

62: operador de implicação ($\Rightarrow$);

42: operador de equivalência ($\Leftrightarrow$);

40: abre parênteses;

41: fecha parênteses;

32: espaço em branco;

120 letra "x" que forma o início do nome de cada âtomo;

48 a 57 algarismos de "0" a "9" usados na continuação do nome de cada âtomo.

O desempenho de seu predicado será avaliado por meio tanto do consumo de tempo como pela quantidade de instanciações de literais que ele realizar durante a execução. Para medir o consumo de tempo, veja o seguinte código aplicado ao cálculo do fatorial de um número. A implementação está no dialeto POPLOG-Prolog (em SWI-Prolog, a contagem de tempo pode ser feita de maneira semelhante usando o predicado `get_time(X)`):

```
:- prolog_language('pop11').
false -> popmemlim;
false -> pop_prolog_lim;
:- prolog_language('prolog').

fat(0,1).
fat(X,Y) :-
  Z is X-1,
  fat(Z,Fz),
  Y is X*Fz.

tempo(Total) :-
```

```
  Inicio is apply(valof(systime)),
  fat(20000, _ ),
  Fim is apply(valof(systime)),
  Total is (Fim - Inicio)/100.0,!.

?- tempo(T).
T = 4.23 ?
yes
```

Para o cálculo da quantidade de instanciações, veja o seguinte código genérico:

```
iniciadas_as_instanciacoes:-
  % retractall(instanciacoes(_)),
  assertz(instanciacoes(0)).

mais_uma_instanciacao :-
  instanciacoes(N),
  % retractall(instanciacoes(_)),
  N2 is N+1,
  assertz(instanciacoes(N2)).
```

Obsevações finais:

- Detalhes da buaca por Satisfação de Restrições (conhecidos como CSP) apresentados no Capítulo 6 da terceira edição do livro do Russell & Norvig. Ele equivale ao Capítulo 5 da segunda edição do mesmo livro, que está disponível de forma aberta em:

  `http://aima.cs.berkeley.edu/newchap05.pdf`

- Use apenas os compiladores POPLOG-Prolog ou SWI-Prolog;
- Não troque o nome do predicado `sat` pois isto dificultará a correção do trabalho.