TRABALHO 1: RESOLUÇÃO PARA LÓGICA DE PRIMEIRA ORDEM DRAFT 0

RAUL H.C. LOPES

1. Introdução

Continuando nosso exercício sobre formalização de tipos indutivos, de programas recursivos e prova de propriedades de funções, adapte seu provador do *Trabalho Prático 0* para provar teoremas da lógica de primeira ordem, **LPO**.

Você deverá implementar um módulo (ou conjunto de módulo, dependendo de como você projetar sua solução), contendo os tipos que definem sua lógica, tradutores da linguagem externa para os tipos internos do seu provador, as regras de inferência. Para efeitos deste documento, tudo isso será aqui identificado como seu **provador**.

2. Seu provador e o mundo

Seu provador entenderá teorias apresentadas usando **LPO** e provará teoremas por contradição. Uma teoria será sempre uma conjunção de disjunções de literais. Um literal representará uma verdade atômica ou a negação de uma verdade atômica. Um átomo será sempre um predicado n-ário aplicado a n termos. Seu provador entenderá teorias e teoremas expressos como disjunções de cláusulas. Por exemplo:

$$-man(?x) \,|\, mortal(?x)$$

representará o axioma:

$$\forall x : man(x) \Rightarrow mortal(x)$$

Note na cláusula:

- o quantificador universal fica implícito;
- na verdade, todas as variáveis são universalmente quantificadas;
- o uso de '**?**' para marcar o início do nome de uma variável: seu provador entenderá qualquer identificador cujo nome se inicia com um sinal de interrogação como sendo nome de variável.

Como antes, use:

- | para representar a disjunção;
- − para representar a negação;

Dado que a notação será toda clausal, seu provador não precisará usar nenhuma constante lógica, além das duas acimas.

2.1. **A linguagem do seu provador.** Em resumo, a linguagem do seu provador consistirá em:

- teorias, que serão conjunções de axiomas, por sua vez, disjunções de literais;
- um teorema a provar, que será representado por uma conjunção de disjunções de literais.

2.2. **Input/output do seu provador.** Seu provador aceitará na linha de comando os seguintes argumentos:

- nome do arquivo contendo o conjunto de cláusulas que formam os axiomas da teoria;
- nome do arquivo o contendo conjunto de clásulas que expressam o teorema a ser provado.
- outros argumentos que direcionem a busca e apresentação da prova.

IMPORTANTE: os dois primeiros argumentos da linha de comando serão, respectivamente, os nomes dos arquivos contendo a teoria e o teorema a ser provado. Violar essa restrição será considerado falta grave, digna da danação eterna nos domínios do *Senhor do Castelo.*

2.3. **A lógica do seu provador.** Seu provador construirá provas por contradição, usando fundamentalmente:

- resolução e fatoração para gerar novas cláusulas;
- eliminaçãode tautologias e cláusulas subjugadas para reduzir a explosão combinatória do espaço de pesquisa.

Essas regras estão formalmente apresentadas nas notas de aula sobre *Grafos, ârvores e termos em FOL.*

Lida uma teoria Γ e o teorema a provar Δ, ele tentará obter uma contradição a partir da conjunção de Γ e Δ.

$$\Gamma \& \Delta \vdash \bot$$

Ele deverá realizar a seguinte seqüência de passos:

1. Ler teoria;
2. converter teoria para representação interna;
3. ler teorema a provar;
4. converter teorema para representação interna;
5. tentar derivar a contradição.

6. exibir resposta.

2.4. A busca da prova. Cada cláusula do conjunto que compõe o teorema gerará uma árvore de pesquisa diferente. Dado um teorema expresso como conjugação de três cláusulas diferentes, $\{C_0, C_1, C_2\}$, seu provador iniciará três tentativas de prova, cada uma delas tendo uma das cláusulas do teorema como clásula inicial. Essa clúsula inicial será, antes da busca da prova começar, a única cláusula do conjunto de suporte.

2.4.1. *O uso de resolução.* Todo passo de resolução usará duas (resolução binária) ou mais (hiper-resolução, que só os fortes de coração implementarão) cláusulas. A primeira dessas cláusulas **será sempre** retirada do conjunto de suporte. A(s) outra(s) clúsula(s) será(ão) cópia(s) (via renomeação de variáveis) de qualquer cláusula que o provador tenha em um dado momento.

A cláusula gerada por um passo de resolução passará a pertencer ao conjunto de suporte.

2.4.2. *O uso de fatoração.* Todo passo de fatoração será aplicado a uma cláusula do conjunto de suporte.

A cláusula gerada por um passo de fatoração pertencerá ao conjunto de suporte.

2.5. Conjunto sem suporte. Seu provador trabalhará permanentemente com dois conjuntos de cláusulas:

- conjunto de suporte: que começa contendo apenas uma das cláusulas que definem o teorema a provar.
- conjunto **base** que conterá as cláusulas sem suporte, no início todas as cláusulas dos dois arquivos de entrada, com exceção da que já estará no conjunto de suporte.

2.6. O formato da resposta. Ao final da prova, seu provador exibirá no *stdout* mensagem com o resultado da prova.

- **Se teorema for provado,** ele exibirá uma linha contendo:
  Proved: LS
  onde **LS** será a lista de substituições associadas a variáveis da cláusula inicial da prova.
  Represente cada substituição como par variável, termo, e apresente a lista de substituições, usando notação da linguagem Haskell. Por exemplo: $[(?x, a), (?y, f(b, ?z, g(c))), (?u, e)]$
- **Se não for possível obter prova,** ele exibirá linha contendo:
  Fail

3. A execução trabalho

3.1. As etapas de construção. Decomponha seu trabalho de acordo com as etapas abaixo.

3.1.1. *Os tipos de dados.* Defina as classes indutivas e os tipos de dados que as representarão na linguagem de implementação.

3.1.2. *Lendo um axioma.* Defina e implemente procedimento para converter um string representando um axioma (ou uma conjectura) para a representação interna do seu sistema.

3.1.3. *Lendo uma teoria.* Defina e implemente procedimento para converter um conjunto de strings representando uma teoria para a representação interna da mesma no seu sistema.

3.1.4. *Apresentando output.* Defina e implemente procedimentos que permitam converter qualquer representação interna de fórmula para um string em acordo com a linguagem externa do provador.

3.1.5. *As regras de inferência.* Defina e implemente as regras de inferência acima.

3.1.6. *Estratégias.* Defina e implemente estratégias para melhorar o desempenho do seu provador, como:

- eliminação de tautologias;
- eliminação de cláusulas subjugadas;
- preferência por uso de cláusulas unitárias nos passos de resolução.

3.2. Linguagem de programação. Implemente em Haskell ou C. Se você, adicionalmente, implementar em Isabelle e provar propriedade de correção da sua implementação você estará aprovado com 10.

3.3. O que entregar. Você deverá entregar *tarball* contendo:

- A implementação do provador;
- Módulo de testes;
- Documentação em LaTeX;
- Makefile

Estando no diretório de trabalho, compacte seus arquivos com o comando:

```
tar -zhcf tp0.tar.gz *
```

Teste seu *tarball*, executando a seguinte seqüência de comandos em um diretório temporário **vazio**:

Provador	até 8.0
Módulo de testes	até 1.0
Documentação	até 1.0
Trabalho individual	$*1.2$
Trabalho em grupo de 2	$*1.0$
Trabalho em grupo de 3	$*0.8$
Trabalho em grupos de 4 ou mais	$*0$

TABELA 1. Pontuação do trabalho

```
tar -zxf tp0.tar.gz
make all
make test
```

IMPORTANTE: Vários trabalhos entregues da última vez não passaram por este teste. Qualquer trabalho que não sobreviva este teste mínimo terá nota será 0 (zero), independente do conteúdo do trabalho. Tão importante quanto: trabalhos submetidos após **dead**line serão anulados. Mesmo que o atraso seja de 12 segundos.

3.4. **Makefile.** Seu *Makefile* conterá dois *targets*:

- **all**: que ativará a compilação de todos os programas bem como a geração de documentação em **PDF** a partir dos fontes em LaTeX.
- **test**: que ativará o módulo de testes.

3.5. **O módulo de testes.** O módulo de testes consistirá neste trabalho simplesmente de um conjunto de aquivos contendo os axiomas de teoremas e não teoremas usados para testar seu provador. Além disso, seu *Makefile* conterá um *target* **test** que acionará a prova de cada um teoremas e não teoremas que constituem o teste.

3.6. **A avaliação.** A nota será atribuída de acordo com a tabela 1. Note que um item com pontuação $*0.8$ indica que a nota final do trabalho será multiplicada por 0.8.

Plágios anulam o trabalho atual e são remetidos imediatamente para direção do CT, que define punição que pode resultar em expulsão da universidade.