

**TRABALHO 0:
AMBIENTE PARA PROGRAMAÇÃO
DRAFT 0**

RAUL H.C. LOPES

1. INTRODUÇÃO

O objetivo fundamental deste trabalho consiste em exercitar simultaneamente:

- Sua capacidade de formulação de programas recursivos, implementando um programa de porte razoável em Haskell;
- seu conhecimento de lógica, implementando uma máquina de inferência;
- sua capacidade de formulação de soluções para problemas usando lógica, implementando algoritmos na sua máquina de inferência.

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:

$$-man(x) \mid mortal(x)$$

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 da mesma convenção do Otter: nomes iniciados com u, v, x, y, w, z são nomes de variáveis;

Como no Otter, use:

- \mid 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 acima.

2.1. A linguagem do seu provador. Em resumo, a linguagem do seu provador consistirá de um subconjunto da linguagem do Otter: todas as construções admissíveis na lógica do seu provador serão admissíveis no Otter.

2.2. Input/output do seu provador. A entrada e saída do seu provador serão compatíveis com entrada/saída do Otter.

Seu provador lerá do *stdin* o arquivo com o conjunto de cláusulas e produzirá em *stdout* a prova resultante.

2.3. A lógica do seu provador. Seu provador implementará no mínimo as seguintes regras de inferência e estratégias de busca, que estão presentes no Otter:

- resolução binária e fatoração para gerar novas cláusulas;
- eliminação de cláusulas subjugadas para reduzir a explosão combinatória do espaço de pesquisa;
- conjunto de suporte;
- busca em largura e profundidade;
- uso de literais de respostas, **\$ANS** literais no Otter, para extração de respostas.

Todas essas regras estão apresentadas nas notas de aulas, em referências disponíveis na biblioteca, como [2] ou [1], e no manual do Otter.

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 o arquivo de entrada.* 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 claramente as regras que pretende ter em seu provador e implemente-as.

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.

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

- A implementação do provador;
- Documentação em L^AT_EX;
- 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**:

```
tar -zxf tp0.tar.gz
make all
./prove < in > out
```

Na seqüência acima, **prove** é o nome do provador e essa última linha aciona a prova para teoria em arquivo de nome **in** e despeja resultado em arquivo de nome **out**.

IMPORTANTE: qualquer fuga dessa especificação implica em danoção eterna na terra do *Senhor do Castelo*. Tão importante quanto: trabalhos submetidos após **deadline** serão anulados. Mesmo que o atraso seja de 12 segundos e por culpa da menstruação atípica da avó do Bush.

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 L^AT_EX.
- **doc**: que converterá um artigo do formato L^AT_EX para **PDF**.

Provador	até 8.0
Artigo	até 2.0
Trabalho individual	*1.3
Trabalho em grupo de 2	*0.9
Trabalho em grupo de 3	*0.5
Trabalho em grupos de 4 ou mais	*0.000001

TABELA 1. Pontuação do trabalho

3.5. **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 à direção do CT, que define punição que pode resultar em expulsão da universidade.

4. AS DATAS

Cada grupo deverá entregar primeiro um plano de trabalho assinado propondo data de entrega final do trabalho que não poderá ir além do dia 10 de setembro. Note que você ainda realizará mais uma prova e opcionalmente um trabalho. Deixar este trabalho se alongar demais prejudicará seu desempenho nas outras avaliações.

Data de entrega do plano de trabalho: 05 de setembro de 2004.

O plano de trabalho deverá especificar claramente:

- o que será implementado;
- as etapas de implementação;
- a data da entrega.

Ao fazer seu plano de trabalho, inspire-se em tudo o que você deveria estar aprendendo em Engenharia de Software.

REFERÊNCIAS

- [1] C. Chang and R.C. Lee, *symbolic logic and mechanical theorem proving*, Academic Press, 1973.
- [2] Zohar Manna and Richard Waldinger, *The deductive foundations of computer programming*, Addison-Wesley, 1993.