CT-200 Fundamentos de Automata e Linguagens Formais
Propostas de Projetos para o Primeiro período de 2010

Prof. Carlos Henrique Q. Forster
Estagiário Wesley M. Telles

Instruções Gerais

1o. Bimestre

Para o primeiro bimestre deverá ser entregue ao professor um artigo de até 6 páginas no formato IEEE Transactions http://www.ieee.org/web/publications/authors/transjnl/index.html#template

O artigo deve contemplar :

- Introdução
- Fundamentação do problema
- Revisão do estado da arte, indicando trabalhos relacionados ao projeto
- Breve descrição de trabalhos relacionados e sua conexão com o projeto
- Proposta da solução a ser implementada e diferencial em relação aos outros trabalhos
- Plano de execução do projeto para o bimestre seguinte

Deve ser criado um portal na web para o projeto. Nesse portal deverá ser descrito o projeto, listados seus participantes, apresentados links, materiais adicionais e código resultante das implementações. Deve ser criado um blog do projeto para acompanhamento pelo professor e pelo monitor da disciplina, bem como pelos colegas dos outros grupos, e deve haver um link da página do projeto para o blog. Sugere-se não disponibilizar o artigo para o público neste momento.

2o. Bimestre

Será exigido um artigo completo com 8 páginas, contendo além dos itens anteriores

- Descrição da solução
- Detalhamento da implementação
- Exemplos, experimentos e/ou resultados
- Conclusão

Serão cobrados da página do projeto

- Código fonte comentado
- Manual do usuário
- Código executável
- Exemplos

Devem ser preparadas apresentações de 50 minutos que deverão ocorrer durante as últimas semanas de aula.

Proposta I – Linux front-ends

O trabalho terá como produto um programa gerador de código. Esse programa recebe como entrada uma gramática que define as possíveis opções de linha de comando de um determinado aplicativo ou comando do UNIX acompanhadas de seus significados. A partir dessa gramática, o programa deve gerar como saída um script que permita a escolha dessas opções através de uma interface gráfica e exibição ou execução do comando criado.

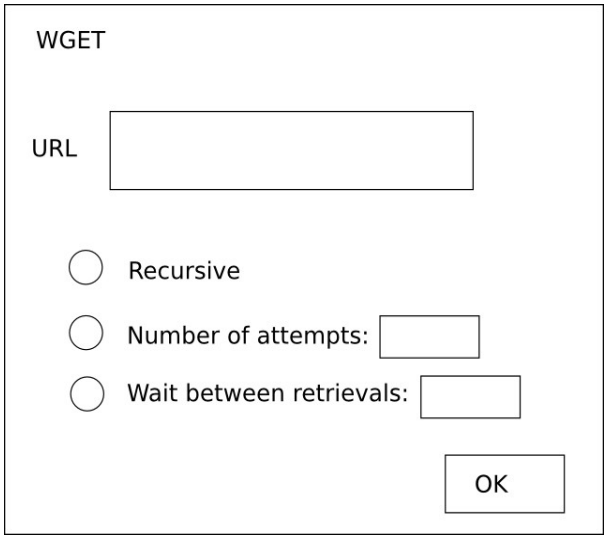
Caberá ao grupo de alunos

- Definir uma linguagem para a gramática de entrada
- Fazer a leitura dessa gramática no programa
- Gerar os scripts de interação pelo programa

Exemplos de linhas de comando complexas:

wget -r --tries=10 http://fly.srk.fer.hr/ -o log
wget --spider --force-html -i bookmarks.html
tar -xvvzf foo.tar.gz

Exemplo de tela para produzir os comandos



Exemplo não funcional em TCL/TK (só desenhando a tela):

```
frame .f
label .f.l1 -text "WGET"
label .f.l2 -text "URL"
entry .f.e1

radiobutton .f.b1 -text "Recursive"
radiobutton .f.b2 -text "Number of attempts"
entry .f.e2
radiobutton .f.b3 -text "Wait between retrievals"
entry .f.e3

button .f.pb -text "OK"
```

pack .f.l1 .f.l2 .f.e1 .f.b1 .f.b2 .f.e2 .f.b3 .f.e3 .f.pb .f

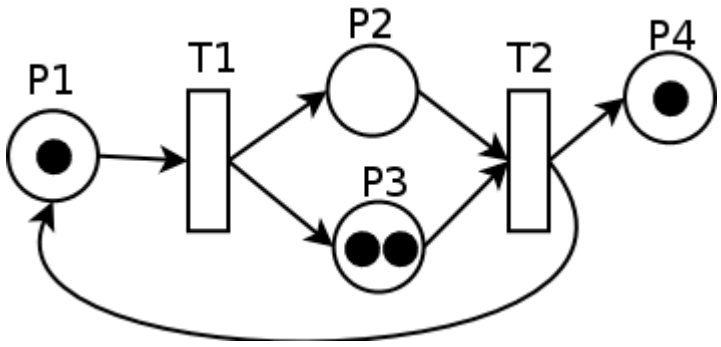
Proposta II – Rede de Petri

O produto deste projeto será um aplicativo que lê uma rede de Petri e instruções para sua simulação de um arquivo texto e apresenta visualmente seu funcionamento. As instruções para simulação devem permitir explorar adequadamente o não-determinismo inerente em algumas redes e devem explicitar o estado inicial da rede. A apresentação visual não necessariamente deve ser online, podendo ser gerados gráficos em formatos como SVG ou Graphviz.

O grupo de alunos deverá:

- Definir uma linguagem para especificar a estrutura das redes e sua simulação
- Construir o analisador sintático da linguagem proposta
- Executar a simulação da rede e apresentá-la graficamente

Exemplo de rede de Petri (Wikipedia)



Proposta III – Reescrita de Expressões Regulares

Deve ser projetado um aplicativo que leia uma lista de regras de reescrita e, a partir de uma cadeia inicial dada, faça a simulação da aplicação dessas regras. A lista de regras é inerentemente não-determinística, não havendo uma ordem específica para aplicação das regras. Assim, o programa deve gerar todas as possibilidades de cadeia final. O programa também deve ter uma opção para mostrar passo a passo a execução das regras. Caberá ao grupo definir a linguagem precisa para explicitar as regras, entretanto segue sugestão:

Para a expressão regular:

a	casa com o caractere ou a sub-expressão a
[abc]	casa com um caractere ou sub-expressão a, b ou c apenas
x*	casa com zero ou mais ocorrências da sub-expressão x
x+	casa com uma ou mais ocorrência de x
{x}	casa com uma sub-expressão x e associa um número à subcadeia correspondente
\\	casa com o símbolo \
\x	casa com o símbolo x, qualquer que seja x (exemplo \[ )

Para a expressão de substituição:

\1	substitui pela primeira expressão entre { }
\2	etc...
\\	gera o símbolo \

Exemplo
Cadeia inicial O:---|---|--- representa o jogo da velha e a vez de “O” jogar.

O:{[-|OX]*}-{[-|OX]*} → X:\1O\2
X:{[-|OX]*}-{[-|OX]*} → O:\1X\2

Aplicando a primeira regra à cadeia original, há 9 possibilidades de casamento, considerando \1=”---|” e \2=”--|---”, a cadeia gerada seria “X:---|O--|---”. Uma aplicação da segunda regra poderia originar a cadeia “O:-X-|O--|---” e assim por diante, até que nenhuma regra possa ser aplicada.

Estas regras substituem um - por O ou por X, alternadamente, até não haver mais -.
Todas as possibilidades do jogo são geradas, mas continua mesmo após a vitória de um dos lados.

Caberá ao grupo de alunos:

- Desenvolver um algoritmo para gerar todas as possibilidades de casamento de uma expressão regular
- Realizar as substituições da cadeia de símbolos fornecida
- Simular o sistema de regras, considerando o não-determinismo e gerando todas as possibilidades
- Criar novos exemplos aplicados

Proposta IV – Schema para JSON

O formato de dados JSON (Javascript Object Notation) vem ganhando popularidade e pode se tornar substituto do formato XML. Entretanto, nem toda funcionalidade presente no XML já existe para o JSON. Por exemplo, uma descrição do formato de dados semelhante ao XMLSchema ou o DTD ainda não está consolidada para o JSON. Assim, neste projeto, o grupo de alunos proporá um formato para validação de dados JSON e implementar o programa para fazer a validação.

Exemplo de DTD (Wikipedia):

```
<!ELEMENT people_list (person*)>
<!ELEMENT person (name, birthdate?, gender?, socialsecuritynumber?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT socialsecuritynumber (#PCDATA)>
```

Exemplo de XML associado:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE people_list SYSTEM "example.dtd">
<people_list>
  <person>
    <name>Fred Bloggs</name>
    <birthdate>27/11/2008</birthdate>
    <gender>Male</gender>
  </person>
</people_list>
```

Exemplo de JSON:

```
{
     "firstName": "John",
     "lastName": "Smith",
     "age": 25,
     "address": {
         "streetAddress": "21 2nd Street",
         "city": "New York",
         "state": "NY",
         "postalCode": "10021"
     },
     "phoneNumber": [
         { "type": "home", "number": "212 555-1234" },
         { "type": "fax", "number": "646 555-4567" }
     ],
     "newSubscription": false,
     "companyName": null
 }
```

Caberá ao grupo:

- Propor um formato para validar dados no formato JSON
- Fazer um analisador sintático para o JSON e para a especificação de formato
- Escrever um programa para ler a especificação de formato e uma cadeia JSON e retornar se aquela cadeia é válida para aquele formato de dados