Concurso Nacional de Programação Lógica e Funcional CeNPLf'06

Faculdade de Engenharia da Universidade do Porto

5–7 de Maio de 2006

Parte da manhã

(4 horas – 6 problemas)

Problemas

6 de Maio de 2006
09h00m – 13h00m

M1: Autómatos *My Way*

Introdução

Descobertos independentemente pelo Prof. Sinatra e por uns cientistas obscuros dum pequeno país (o laboratório Xutos e Pontapés Inc. e os seus intrigantes autómatos *à minha maneira*), os autómatos *my way* formam uma variedade exótica dos autómatos finitos que merecem a atenção da vossa ilustre comunidade.

Tratam-se de autómatos de estados finitos (eventualmente não deterministas) reactivos cujas transições são parametrizadas por uma fórmula lógica proposicional e por uma acção. Outra componente importante destes autómatos é o chamado *estado lógico*. Este estado consiste num conjunto de variáveis proposicionais e dos seus valores de verdade.

Neste contexto uma transição pode ser escolhida quando tem por rótulo o caracter analisado e quando a fórmula lógica em parâmetro é satisfeita. As acções (de facto as reacções) são associadas às transições e modificam o estado lógico do autómato. Isto é, alteram o valor lógico das variáveis proposicionais do estado.

Considerando $\mathcal{P}$ como o conjunto das variáveis proposicionais e $\mathcal{P}rop_{\neg,\wedge,\vee}$ como o conjunto das fórmulas da lógica proposicional que só envolvem as conectivas $\neg$, $\wedge$ e $\vee$, um autómato *my way* é definido mais formalmente como um $6-$tuplo $(\Sigma, S, S_0, State, \delta, F)$ onde:

Σ	: Alfabeto de entrada
S	: Conjunto dos estados do autómato
S_0	: Subconjunto de S dos estados iniciais
$State$	: Função parcial do conjunto de variáveis proposicionais $\mathcal{P}$, para o conjunto $\mathbb{B} = \{0, 1\}$, i.e. representa o estado lógico inicial do autómato
δ	: Relação de transição ($\subseteq (\Sigma \times S \times \mathcal{P}rop_{\neg,\wedge,\vee}) \times (S \times (\mathcal{P} \to \mathbb{B}))$)
F	: Conjunto dos estados finais

Veja por exemplo na figura 1 a representação gráfica de um autómato *my way* em que $\Sigma = \{a, b\}$, $S = \{1, 2, 3\}$, $S_0 = \{1\}$, $State = \{P \mapsto 1; Q \mapsto 0\}$, $F = \{3\}$. A relação δ é representada pelos rótulos nas transições em que `!` representa a negação, `|` a disjunção, `&` a conjunção, `X:=V` a actualização do estado lógico em que `X` passa a ter o valor lógico `V`.

A execução destes autómatos consiste no *consumo* de uma palavra de Σ^* (uma sequência eventualmente vazia de letras de Σ) fornecida em parâmetro. Este consumo é iniciado a partir dos estados de S_0 e consiste na exploração do grafo subjacente ao autómato: transita-se dum estado i para um estado j se existir uma transição entre i e j com o rótulo (a, f, e) tal que: a seja a primeira letra da palavra por consumir, f seja uma fórmula proposicional envolvendo as variáveis do estado lógico e cujo o valor de verdade é 1. Neste caso o valor do estado lógico é alterado de acordo com e e a primeira letra da palavra por consumir é removida. O processo continua novamente até não restar

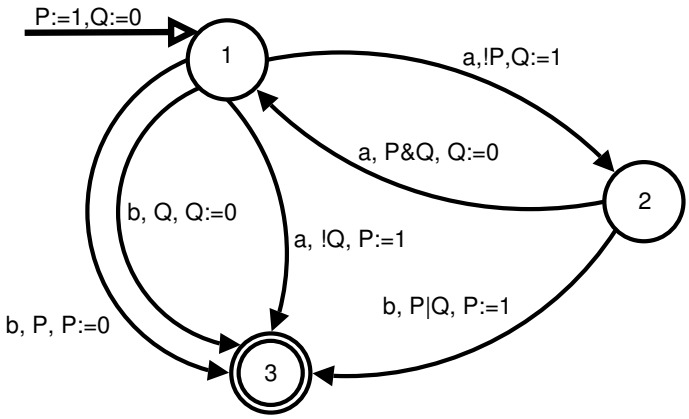

Figura 1: Um autómato *my way*

letras. Se, a partir de um estado, houver várias escolhas possíveis, todas elas têm de ser exploradas.

Desta forma, diz-se que o autómato aceita uma palavra de Σ^* quando uma das execuções possíveis atingiu um estado final (i.e. pertencente a F).

Tarefa

A sua tarefa consiste em escrever um programa que aceite em entrada um autómato *my way* e uma palavra e que indique se a palavra é aceite ou não.

Se optar pela Programação Lógica, implemente o programa através de um predicado `my_way` que aceita em argumento o autómato e a palavra (lista de letras) por analisar. O predicado deverá ser verdade se a palavra é aceite pelo autómato.

Se escolher a Programação Funcional, desenvolva uma função binária `my_way` em que o primeiro argumento é o autómato e o segundo a palavra. Esta função deverá escrever "`yes`" se a palavra parâmetro é aceite, "`no`" senão.

Os Dados

Vamos considerar neste problema que $\mathcal{P} = \{p, q, r\}$ e que as fórmulas são dadas por termos (no paradigmo lógico) ou por árvores (no paradigma funcional). O autómato $(\Sigma, S, S_0, State, \delta, F)$ e a palavra são fornecidos da seguinte forma:

Programação lógica. Σ é representado por uma lista de átomos (por exemplo `[a,b,c]`), os conjuntos de estados S, S_0 e F por listas de inteiros, *State* por uma lista de 3 elementos de $\{0, 1\}$ onde cada um representa o valor lógico de p, q, e r (nesta ordem). As transições são representadas por uma lista de listas da forma

```
[estado_partida,estado_chegada,termo,actualizações]
```

onde

- `termo`, por exemplo `and(or(p,q),not(q))`, representa uma fórmula de $\mathcal{P}rop_{\neg,\wedge,\vee}$
- `actualizações` é uma lista de dois elementos. Por exemplo `[[p,1],[q,0]]` representa as actualizações ao estado $p := 1$ $q := 0$.

Finalmente, a palavra por reconhecer é representada por uma lista de átomos, como por exemplo `[a,b,b,a]`.

Programação funcional. Para começar assuma a existência do seguinte tipo `formula`.

```
type formula = AND of formula * formula | OR of formula * formula
             | NOT of formula | VAR of char
```

Σ é representado por uma lista de caracteres (por exemplo `['a';'b';'c']`), os conjuntos de estados S, S_0 e F por listas de inteiros, *State* por uma lista de 3 elementos de $\{0, 1\}$ onde cada um representa o valor lógico de p, q, e r (nesta ordem). As transições são representadas por uma lista de $4-$tuplos da forma

```
(estado_partida,estado_chegada,termo,actualizações)
```

onde

- `termo`, por exemplo `AND ( OR (VAR 'p', VAR 'q'), NOT 'q')`, representa uma fórmula de $\mathcal{P}rop_{\neg,\wedge,\vee}$
- `actualizações` é uma lista de pares *char∗int*. Por exemplo `[('p',1);('q',0)]` representa as actualizações ao estado $p := 1$ $q := 0$.

Finalmente, a palavra por reconhecer é representada por uma lista de caracteres, como por exemplo `['a';'b';'b';'a']`.

Os Resultados

As palavras `yes` ou `no`.

1º Exemplo (Prog. Lógica)

Segue-se um exemplo ilustrativo do programa pretendido:

```
?- my_way ([[a,b],[1,2,3,4],[1],[1,0,0],
           [[1,2,a,p,[q,1]],
            [2,3,b,and(p,q),[q,0]],
            [2,4,b,q,[p,0]],
            [3,4,a,or(p,q),[p,1]],
            [4,1,a,not(q),[p,1]]],
           [4]]
           [a,b,a]).

Yes
```

2º Exemplo (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido:

```
# my_way (['a';'b'],[1;2;3;4],[1],[1;0;0],
           [(1,2,'a',VAR 'p',[('q',1)]),
            (2,3,'b',AND (VAR 'p', VAR 'q'),[('q',0)]),
            (2,4,'b',VAR 'q',[('p',0)]],
            (3,4,'a',OR (VAR 'p', VAR 'q'),[('p',1)]),
            (4,1,'a',NOT (VAR 'q'),[('p',1)])],
           [4])
           ['a;'b';'a'];;

yes
- : unit = ()
```

M2: Baralhada

Baralhada

Num jogo de cartas, sem cartas repetidas, pretende-se construir uma sequência de cartas a partir de vários montes de cartas, não necessariamente de igual tamanho, de acordo com as seguintes regras:

- os montes estão colocados por uma certa ordem e são visitados circularmente nessa ordem.
- a primeira carta vem do primeiro monte.
- são retiradas por ordem as cartas de topo nos restantes montes que sejam do mesmo naipe da primeira, repetindo-se até nenhum monte ter no topo uma carta desse naipe.
- se ainda houver cartas, o processo é reiniciado com a primeira carta do monte (não vazio...) que se segue àquele de onde foi retirada a última carta na sequência.

Tarefa

Escreva um programa que faça a construção de uma sequência com base nas representações seguintes:

- a sequência de montes é dada por uma lista de comprimento ≥ 1.
- os montes e a sequência de cartas são listas de cartas.
- cada monte tem inicialmente pelo menos uma carta.
- cada carta é representada por um par `(V,N)`, em que `V` e `N` representam o valor e o naipe, respectivamente.

Os Resultados

Em Prolog o programa deve ser posto em execução chamando o predicado `baralha/2` cujos argumentos são a lista de montes (dada) e a sequência resultado, nesta ordem. O predicado deve dar apenas a solução correcta. Por exemplo:

```
?- baralha([[(4,e),(a,e),(k,e)],
            [(10,e),(q,p),(5,c),(6,p)],
            [(q,e),(2,e),(3,c),(2,c)]],S).

S = [(4,e),(10,e),(q,e),(a,e),(2,e),(k,e),
     (q,p),(3,c),(5,c),(2,c),(6,p)] ? ;
no
```

Numa linguagem funcional, o programa definirá uma função `baralha` que a partir da lista de montes fornece a lista que representa a sequência resultado. Por exemplo:

```
> (baralha (((4 e) (a e) (k o))
            ((10 o) (q p))
            ((k e) (2 c))))

((4 e) (k e) (a e) (10 o) (k o) (q p) (2 c))
```

M3: Códigos Ambíguos

Introdução

Por questões de segurança, o Departamento de Informática do Banco da Patagónia desenvolveu um sistema de códigos para guardar a informação relativa aos seus clientes. Neste sistema altamente sofisticado, a cada símbolo (letra, número, ...) é associada uma sequência de bits (`0`s e `1`s) e cada string é codificada pela concatenação das sequências correspondentes aos seus caracteres.

Tudo parecia estar a correr bem até ao dia em que a Sra. Java apareceu a reclamar porque a conta do cabelereiro da Sra. Pascal lhe tinha sido debitada. O gerente do banco foi investigar e descobriu que o código que estava a ser utilizado era o seguinte.

a	c	j	l	p	s	v
010	01	001	10	0	1	101

De facto, o nome `pascal` é representado pelo código `001010101010`, e infelizmente... este código também corresponde ao nome `java`.

O gerente do banco ficou com os cabelos em pé, despediu imediatamente o funcionário responsável pelo código e decidiu como medida de segurança contratar alguém que desenvolvesse um programa para detectar se um dado código é ou não ambíguo.

Tarefa

A sua tarefa consiste em escrever um programa que receba um código ambíguo e devolva a mensagem codificada mais curta (caso haja várias, a primeira por ordem lexicográfica) que pode ser interpretada de forma ambígua e duas das suas possíveis interpretações.

Se optar pela Programação Lógica, implemente o programa através de um predicado `ambiguo/4` que receba no primeiro argumento o código e devolva no segundo argumento a mensagem codificada encontrada e nos dois restantes argumentos duas das suas possíveis interpretações.

Se escolher a Programação Funcional, desenvolva uma função `ambiguo` que receba como argumento o código e devolva uma lista contendo a mensagem codificada encontrada e duas das suas possíveis interpretações.

Os Dados

O código é fornecido como uma lista de pares cujo primeiro elemento é o símbolo e o segundo elemento é o código desse símbolo (lista de `0`s e `1`s).

Os Resultados

A mensagem codificada deve ser representada como uma lista de `0`s e `1`s; as mensagens descodificadas devem ser representadas como listas de símbolos.

1º Exemplo (Prog. Lógica)

Segue-se um exemplo ilustrativo do programa pretendido.

```
?- ambiguo([(a,[0,1,0]),
            (c,[0,1]),
            (j,[0,0,1]),
            (l,[1,0]),
            (p,[0]),
            (s,[1]),
            (v,[1,0,1])],M,T1,T2).

M = [0, 1]
T1 = [c]
T2 = [p, s]
```

Outro exemplo que o seu programa deverá ser capaz de resolver é o seguinte.

```
?- ambiguo([(a,[0,1,1,0]),
            (b,[0,1,1,1,1,1]),
            (c,[1,1,0,0,1,1,1,1]),
            (f,[1,0,1,1,1,0]),
            (j,[0,1,0]),
            (l,[0,1,0,0]),
            (r,[0,1,1,1,0])],M,T1,T2).
```

2º Exemplo (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido.

```
> ambiguo [(a,[0,1,0]),
           (c,[0,1]),
           (j,[0,0,1]),
           (l,[1,0]),
           (p,[0]),
           (s,[1]),
           (v,[1,0,1])]

[[0, 1], [c], [p, s]]
```

M4: Integrados

Introdução

A construção de circuitos integrados requer o desenho de conexões eléctricas sobre placas, entre pares de pontos distintos do circuito. Para representar essas ligações pode usar-se uma grelha rectangular, de N por M, onde se vão ligando segmentos horizontais e verticais, entre dois pontos adjacentes. Com esta modelação, cada ligação entre dois pontos, pode ser representada pelo conjunto de pontos do circuito por onde essa ligação passa. Para garantir que as ligações não se cruzam, algo que é exigível para os circuitos, basta garantir que nenhum ponto pertence a duas ligações distintas.

A figura 2 representa graficamente uma solução para o problema de encontrar ligações entre os pontos (1,4) e (2,1), entre os pontos (1,3) e (3,4), e entre os pontos (1,1) e (4,4),. Com a modelação acima sugerida, esta última ligação é representada pelo conjunto de pontos (1,1), (1,2), (2,2), (3,2), (4,2), (4,3), (4,4).

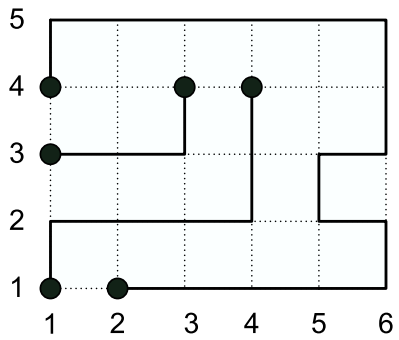

Figura 2: Circuito com 3 ligações

A figura 3 representa graficamente uma solução para ligar os pontos indicados com a mesma letra (A com A, B com B, etc), numa grelha de 10 por 10.

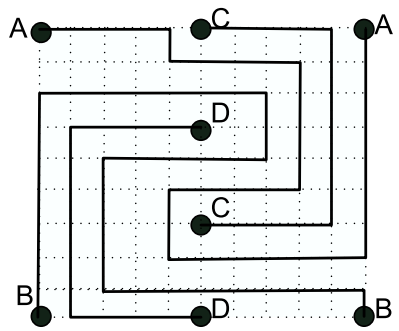

Figura 3: Circuito um pouco mais complexo

Tarefa

A sua tarefa consiste em fazer um programa que dada a dimensão da grelha, que no máximo terá 10 linhas e no máximo 10 colunas, e um conjunto de pontos a ligar determina se é possível fazer as ligações desses pontos nessa grelha sem que nenhuma das ligações se cruze. Para além disso, caso seja possível, o seu programa deverá ainda dizer como, ou seja por que pontos da grelha passa cada uma das ligações.

Se optar pela Programação Lógica, implemente o programa através de um predicado `integrado/4` em que os primeiros 3 argumentos contém a definição do problema (i.e. dimensão da grelha e pontos a ligar) e o segundo devolve o resultado, conforme descrito abaixo.

Se escolher a Programação Funcional, desenvolva uma função `integrado/3` que recebe como argumentos os dados descritos abaixo e devolve o resultado.

Os Dados

Quer use programação em lógica ou programação funcional, o input dum problema é dado em 3 argumentos, sendo o primeiro o número de colunas da grelha, o segundo o número de linhas. O 3º argumento tem uma lista de pares de pontos a ligar onde cada ponto é um par ordenado `(X,Y)`.

Os Resultados

Como resultado deverá ser devolvida uma lista (no caso da programação em lógica, no 4º argumento) de conjuntos (listas) de pontos. Cada elemento da lista resultado representa uma ligação, e contém todos os pontos da grelha que pertence a essa ligação.

Exemplos

Abaixo apresenta-se a chamada do predicado `integrado/4` e a solução obtidas relativamente ao circuito representado da figura 2:

```
?- integrado(6,5, [((1,4),(2,1)), ((1,3),(3,4)), ((1,1),(4,4))], Res).

Res = [ [(1,4),(1,5),(2,5),(3,5),(4,5),(5,5),(6,5),(6,4),(6,3),
         (5,3),(5,2),(6,2),(6,1),(5,1),(4,1),(3,1),(2,1))],
        [(1,3),(2,3),(3,3),(3,4)],
        [(1,1),(1,2),(2,2),(3,2),(4,2),(4,3),(4,4)]
      ]
```

Apresenta-se agora a chamada do predicado integrado/4 para o circuito da figura 3:

```
?- integrado(10,10,
             [  ((1,10),(10,10)), %As
```

```
    ((1,1),(10,1)),   %Bs
    ((5,10),(5,4)),   %Cs
    ((5,1),(5,7))],   %Ds
Res).
```

M5: Steinhaus

Introdução

Temos uma longa fila de 840 buracos, todos alinhados, todos iguais, todos vazios, numerados de 0 a 839. Temos também 8 bolas, numeradas de 1 a 8, e queremos metê-las nos buracos. Nada mais fácil, dirá você. Não é bem assim, digo eu, porque, neste exercício, queremos que as bolas fiquem distribuídas "uniformemente" pela fila de buracos. As bolas estarão distribuídas uniformemente pela fila de buracos se em cada uma das 8 secções com 105 buracos (a primeira secção tem os buracos 0 a 104, a segunda os buracos 105 a 209, etc.) houver exactamente uma bola, mas também se em cada uma das 7 secções com 120 buracos (a primeira destas secções tem os buracos numerados de 0 a 119, a segunda de 120 a 239, etc.), houver exactamente uma das 7 primeiras bolas. As 7 primeiras bolas são as bolas com números entre 1 e 7. E etc. Este "etc." significa que, além disso, em cada uma das 6 secções com 140 buracos deve haver uma das 6 primeiras bolas, em cada uma das 5 secções com 168 buracos deve haver uma das 5 primeiras bolas, e analogamente, quando dividimos a fila em 4, 3 e 2 secções do mesmo tamanho.

Tarefa

A sua tarefa consiste em escrever um programa que, dados os números dos buracos em que metemos as quatro primeiras bolas (numeradas de 1 a 4), calcule os números dos buracos em que devemos meter as outras quatro bolas (as numeradas de 5 a 8), de acordo com a regra da distribuição uniforme, se for possível. O resultado deve ser uma lista ou um vector com esses quatro números, por ordem do número de bola, se o problema tiver solução. Se o problema não tiver solução, o resultado deve ser um vector vazio ou uma lista vazia. Se houver várias soluções, qualquer uma serve. .... .

Se optar pela Programação Lógica, implemente o programa através de um predicado `steinhaus/2` em que o primeiro argumento é a lista com os dados e o segundo é a lista com o resultado. Se escolher a Programação Funcional, desenvolva uma função `steinhaus/1` que recebe como argumento o vector ou lista com os dados e devolve o vector ou lista com o resultado.

Os Dados

Os 4 números que compõem o vector ou lista fornecidos como dados, representam os números dos buracos onde estão já colocadas as bolas números 1, 2, 3 e 4, por esta ordem. São números entre 0 e 839 e não há repetições.

Os Resultados

Se o problema tiver solução, os quatro números que compõem o vector ou a lista resultado representam os números dos buracos onde devemos colocar as bolas números 5, 6, 7 e 8, por esta ordem, de maneira a que, em conjunto com as bolas colocadas inicialmente, a distribuição esteja de acordo com as regras. Se, para a configuração inicial das quatro bolas dadas, não for possível resolver o problema, o resultado deve ser um vector vazio ou uma lista vazia. Se houver mais do que uma solução, qualquer uma serve.

Nota

Este problema é um caso particular de um problema inventado pelo matemático polaco Hugo Steinhaus (1887-1972). O problema geral, expresso não com buracos e bolas mas com um segmento de recta e pontos sobre esse segmento, é interessantíssimo e muito difícil. Steinhaus descobriu uma solução com 14 pontos e um colega seu demonstrou que não existe solução com mais do que 17 pontos.

1º Exemplo (Prog. Lógica)

Seguem-se exemplos ilustrativos do programa pretendido:

```
?- steinhaus( [0 420 819 210], T ).

T = [560 315 630 105]

?- steinhaus( [500 100 800 210], T ).

T = []
```

2º Exemplo (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido:

```
> steinhaus [300 800 0 420]

[560 140 630 315]

> steinhaus [600 200 800 0]

[]
```

M6: Cheques Brinde

Introdução

Existe uma série de *cheques brinde*, cada um com um determinado valor. Para pagar uma determinada quantia pretende-se determinar qual a melhor combinação de cheques a usar (isto porque, ao contrário do que acontece com o dinheiro, não há troco). Esta deve ser tal que a soma dos seus valores seja superior ou igual à quantia em causa. Deve-se no entanto minimizar o valor desta soma.

Assuma que a série de cheques brinde é modelada por uma lista de números que representam os valores dos cheques.

Tarefa

A sua tarefa consiste em escrever um programa para resolver este problema, determinando a lista de cheques a serem usados para pagar uma determinada quantia.

Se optar pela Programação Lógica, implemente o programa através de um predicado `escolhe/3` de tal forma que o primeiro argumento é a lista de cheques existentes, o segundo a quantia a pagar e o terceiro uma lista contendo os cheques a serem usados para pagar a quantia.

Se escolher a Programação Funcional, desenvolva uma função `escolhe :: [Int] -> Int -> [Int]` em que o primeiro argumento é a lista de cheques existentes, o segundo a quantia a pagar e que devolve uma lista contendo os cheques a serem usados para pagar a quantia.

Os Dados

Os dados do problema, passados como argumentos ao programa a desenvolver, são a lista dos valores dos cheques e a quantia a ser paga.

Os Resultados

O resultado de invocar o predicado ou a função `escolhe` é uma lista contendo os cheques a serem usados para pagar a quantia.

1º Exemplo (Prog. Lógica)

Segue-se um exemplo ilustrativo do programa pretendido:

```
?- escolhe ([12,23,11,37,2,13],36,X).

X = [23,13]
```

2º Exemplo (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido:

```
> escolhe [12,23,11,37,2,13] 36

[23,13]
```