MAC239 - Introdução à Lógica e Verificação de Programas
Primeiro Exercício Programa

1 O problema das n-Rainhas

Dado um tabuleiro n x n, uma solução para o problema das n-rainhas é uma configuração de n rainhas neste tabuleiro em que não haja duas em posição de ataque. Uma rainha ataca outras peças que estejam na mesma linha, coluna ou diagonal. Neste EP, vamos construir um programa que constrói uma fórmula da lógica propsicional e em seguida constrói um ORBDD correspondente que poderá ser usado para:

- verificar a existência de solução para tabuleiros de tamanho $n>2$ e, caso exista, imprimir uma configuração das rainhas que é solução.
- verificar a existência de solução para um tabuleiro de tamanho n>2, porém considerando agora que algumas rainhas já estão posicionadas.

2 Representando o tabuleiro com variáveis proposicionais

Para utilizarmos lógica na resolução deste problema, vamos transformar nosso tabuleiro n x n em um conjunto de n^2 variáveis. Denotaremos a variável correspondente à casa da i-ésima linha e j-ésima coluna por r_{ij}. Assim, $r_{ij} = T$ significa que há uma rainha nessa casa e $r_{ij} = F$ se a casa está vazia.

A primeira tarefa desse exercício é montar uma fórmula ϕ tal que uma valoração para as variáveis r_{ij} satisfaça ϕ se, e somente se, ela for uma solução para o problema das n rainhas.

(Dica: podem montar essa fórmula ϕ em CNF, criando um conjunto de cláusulas que precisam todas ser verdadeiras para que ϕ seja satisfeita)

2.1 Exemplo de CNF para o problema de 2 rainhas:

Vamos codificar o tabuleiro 2 x 2 com 4 variáveis: r_{11}, r_{12}, r_{21} and r_{22}. Cláusulas para a presença de uma rainha em cada linha:

- $r_{11} \vee r_{12}$ (c1)
- $r_{21} \vee r_{22}$ (c2)

Cláusulas de restrição de não ataque nas linhas:

- $\neg r_{11} \vee \neg r_{12}$ (c3)
- $\neg r_{21} \vee \neg r_{22}$ (c4)

Cláusulas de restrição de não ataque nas colunas:

- $\neg r_{11} \vee \neg r_{21}$ (c5)
- $\neg r_{12} \vee \neg r_{22}$ (c6)

Cláusulas de restrição de não ataque nas diagonais:

- $\neg r_{11} \vee \neg r_{22}$ (c7)
- $\neg r_{12} \vee \neg r_{21}$ (c8)

Solução total 2x2: $(c1) \wedge (c2) \wedge (c3) \wedge (c4) \wedge (c5) \wedge (c6) \wedge (c7) \wedge (c8)$

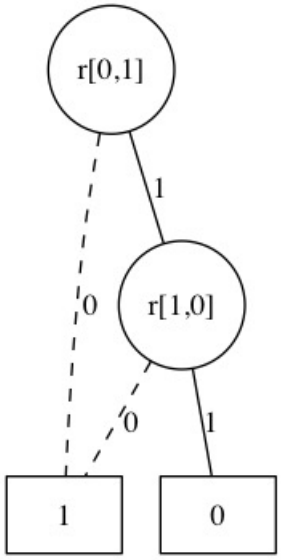
3 Algumas dicas sobre o uso da biblioteca de BDDs

Neste EP usaremos a biblioteca PyEDA (http://pyeda.readthedocs.org/en/latest/bdd.html) para a criação e manipulação dos diagramas de decisão binários. Neste link tem uma descrição sobre como instalar e utilizar a biblioteca. As principais funções que vamos usar:

- função bddvars("r",n,n): cria o array de variáveis proposicionais;
- operador $\sim$ para negação;
- operador | para ou;
- operador & para e;
- is_zero(): verifica se bdd é zero ou se a fórmula é insatisfazível;

- restrict(x=1): restringe um bdd ao subdiagrama correspondente ao caso de x=1;
- satisfy_one(): retorna uma valoração que satisfaz a fórmula desse diagrama;

A biblioteca tem uma função *to_dot*() que permite gerar um arquivo .dot que pode ser visualizado com o dot do GraphViz(http://www.graphviz.org), para gerar figuras como a seguinte:



O formato .dot é uma descrição em texto do grafo, e o programa dot transforma essa descrição em imagem, por exemplo em jpeg, com um comando: $dot -Tjpg BDD.dot -o BDD.jpg

Uma sugestão para apresentar a saída do seu programa é transformar a atribuição das n^2 variáveis em um tabuleiro com pontos ". " ou rainhas "Q ". Como o seguinte para 5x5:

```
. Q . . .
. . . . Q
. . Q . .
Q . . . .
. . . Q .
```

4 Entrada e saida do seu programa

O seu programa deve ler da entrada padrão dois números na primeira linha: N, o tamanho do tabuleiro e K, o número de rainhas já colocadas. Assim nas

próximas K linhas temos as posições da raínhas. Exemplo:

```
Nrainhas Ndadas=K
Xrainha-1 YRainha-1
Xrainha-2 YRainha-2
...
Xrainha-K YRainha-K
```

Para avaliar a primeira parte do EP, K será 0, assim basta criar as fórmulas sem considerar as restrições.

Na segunda parte haverão rainhas já colocadas e essa informação deve ser usada durante a geração da fórmula, para reduzir o tamanho das fórmulas intermediárias da solução. Programas que fizerem a restrição apenas após construir a fórmula completa serão ineficientes para resolver os maiores problemas e perderão pontos.

A primeira linha da saída de seu programa deve ser: "SAT", se existe solução para o tabuleiro ou "UNSAT" se não existe. Além disso se existir deve ser apresentado um tabuleiro com uma das soluções.

4.1 Exemplos:

Entrada:

```
3 0
```

Saída:

```
UNSAT
```

Entrada:

```
4 0
```

Saída:

```
SAT
. Q . .
. . . Q
Q . . .
. . Q .
```

Entrada:

```
4 2
1 2
3 3
```

Saída:

```
UNSAT
```

Entrada:

```
5 2
0 1
2 2
```

Saída:

```
SAT
. Q . . .
. . . . Q
. . Q . .
Q . . . .
. . . Q .
```

5 O que deverá ser entregue?

Um programa em python e um relatório curto que inclua até que instâncias o seu programa pode resolver no caso sem rainhas já colocadas.

Tarefa Bonus: ordenação das variáveis Vimos que, dependendo da ordem das variáveis, um BDD pode ser ainda mais reduzido. Você pode verificar isso modificando a ordem em que as fórmulas são criadas (uma vez que a ordem das variáveis no BDD é a mesma em que as variáveis são usadas no programa). Construa e entregue 3 versões do seu programa que realizam as mesmas operações, apenas em ordens diferentes. Comente no relatório sobre as diferenças no tamanho máximo do BDD gerado e tempo de execução entre as versões.