Lógica Computacional 2020-2021: Trabalhos Práticos

1. Os alunos participam na avaliação prática integrados em grupos. A constituição dos vários grupos de trabalho é feita individualmente por cada aluno na folha de cálculo aqui ("sheet" Grupos).
2. Cada grupo contém 2 alunos ou excepcionalmente 3. Nos grupos com 3 alunos, a nota do trabalho é penalizada em 5%. Devido ao elevado número de inscrições não é possível existir participação individual não integrado num grupo.
3. Os trabalhos têm a forma de num "notebook" Jupyter distinto para cada um dos problemas indicados. Cada notebook deve **resumidamente**
   a. descrever o problema e a abordagem usada para o resolver
   b. apresentar o código Python que resolve o problema
   c. apresentar exemplos e testes de aplicação.
4. A entrega do trabalho tem a forma de uma discussão oral de 30 minutos com todos os elementos do grupo, e inclui a demonstração da boa execução do código.
5. Os "notebooks" executáveis e uma cópia PDF de cada um, devem ser previamente enviados via e-mail ao responsável da disciplina (@José Manuel V) até às 23:59 do domingo imediatamente anterior à 1ª data de entrega desse trabalho.
6. A classificação do trabalho é específica de cada elemento do grupo segundo a perceção que o avaliador tem da contribuição de cada um.
7. A entrega dos trabalhos realiza-se nas tardes de 2ª, 4ª e 6ª feiras, nas datas abaixo indicadas. A inscrição no horário de entrega é feita pelos grupos na folha de cálculo aqui ("sheet" Slots)

		Observações
TP1	2, 4 e 6 de Novembro 2020	
TP2	23,25 e 27 de Novembro 2020	
TP3	14,16 e 18 de Dezembro 2020	
TP4	11, 13 e 15 de Janeiro 2021	

Trabalho 1

> *O propósito deste trabalho é a análise de problemas de alocação usando técnicas de SAT, em lógica proposicional, e IP em lógica linear inteira.*

1. Pretende-se construir um horário semanal para o plano de reuniões de projeto de uma "StartUp" de acordo com as seguintes condições:
   a. Cada reunião ocupa uma sala (enumeradas $1 \ldots S$) durante um "slot" $(\text{tempo}, \text{dia})$. Assume-se os dias enumerados $1..D$ e, em cada dia, os tempos enumerados $1..T$.
   b. Cada reunião tem associado um projeto (enumerados $1..P$) e um conjunto de participantes. Os diferentes colaboradores são enumerados $1..C$.
   c. Cada projeto tem associado um conjunto de colaboradores, dos quais um é o líder. Cada projeto realiza um dado número de reuniões semanais. São "inputs" do problema o conjunto de colaboradores de cada projeto, o seu líder e o número de reuniões semanais.
   d. O líder do projeto participa em todas as reuniões do seu projeto; os restantes colaboradores podem ou não participar consoante a sua disponibilidade, num mínimo ("quorum") de 50% do total de colaboradores do projeto. A disponibilidade de cada participante, incluindo o lider, é um conjunto de "slots" ("inputs" do problema).

2. O "pigeon hole principle" (PHP) é um problema clássico da complexidade. Basicamente

   > *Existem N pombos e $N-1$ poleiros de pombos. Cada pombo ocupa totalmente um poleiro. Pretende-se alocar cada pombo a um poleiro próprio.*

   a. Provar que não existe solução do problema, usando **Z3** em

    i. lógica proposional
    ii. lógica inteira linear
  b. Analisar a complexidade de cada uma das provas em função de N de forma empírica. Como sugestão pode começar por fazer um “plot” do tempo de execução.

Trabalho 2

> *O objetivo deste trabalho é a modelação de grafos, em Z3 e NetworkX.*

1. Um sistema de tráfego é representado por um grafo orientado ligado. Os nodos denotam pontos de acesso e os arcos denotam vias de comunicação só com um sentido .

   > *O grafo tem de ser ligado o que significa que entre cada par de nodos $\langle n_1, n_2 \rangle$ tem de existir um caminho $n_1 \rightsquigarrow n_2$ e um caminho $n_2 \rightsquigarrow n_1$.*

   a. Gerar aleatoriamente o grafo. Assuma $N = 32$ para o número de nodos.
      i. Cada nodo tem um número aleatório de descendentes no intervalo $1..d$ cujos destinos são também gerados aleatoriamente. Assume-se que não existem “loops” nem repetição de destinos.
   b. Pretende-se fazer manutenção interrompendo determinadas vias.

   Determinar <u>o maior número de vias</u> que é possível remover mantendo o grafo ligado.

2. Considere circuitos aritméticos $N \times 1$ (N inputs e 1 output), com “wires” de 16 bits e “gates” de três tipos:
    i. a “gate” binária $\oplus$ implementa **`xor`** bit a bit
    ii. a “gate” binária $+$ implementa soma aritmética (**`add`**) de inteiros módulo 2^{16} ,
    iii. a “gate” unária $\gg_r$ implementa o “right-shift-rotate” do argumento um número de posições dado pela constante $0 < r < 16$.

   Os parâmetros do circuito são o número de inputs N, o número de “gates” M e a razão γ entre o número de “gates” **`add`** e o número total de “gates”.

   Neste problema
   a. É dado um circuito aleatoriamente gerado com parâmetros N, M e γ .
   b. São dados também o valor do output final e o “output” de todas as “gates” **`add`** .

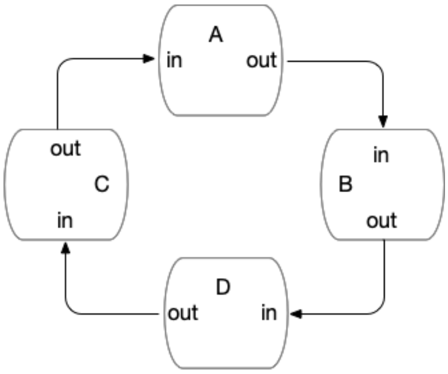
   Pretende-se usar Z3 para determinar se os dados são consistentes entre si e, se forem, determinar inputs que sejam compatíveis com tais outputs.

Estes circuitos costumam designar-se por XAR (“**`xor`** $-$ **`add`** $-$ **`rotate`**”) e são muito usados numa classe de cripto-esquemas designada por LWC (“lightweight cryptography ”). Nesses cripto-esquemas as “gates” $\oplus$ e $\gg_r$ são operações lineares enquanto que **`add`** é não linear; por isso as questões de eficiência fazem com que as “gates” **`add`** sejam só uma pequena parte do total de “gates”.

Trabalho 3

> *O objetivo deste trabalho é a utilização do sistema Z3 na análise de propriedades temporais de sistemas dinâmicos modelados por FOTS (“First Order Transition Systems” .*

1. O seguinte sistema dinâmico denota 4 *inversores* (A, B, C, D) que lêem um bit num canal input e escrevem num canal output uma transformação desse bit.



i. Cada inversor tem um bit s de estado, inicializado com um valor aleatório.
ii. Cada inversor é regido pelas seguintes transformações

$$\begin{array}{l}\mathbf{invert}(in, out)\\ \quad x \leftarrow \mathsf{read}(\mathtt{in})\\ \quad s \leftarrow \neg x \;\;\|\;\; s \leftarrow s\\ \quad \mathsf{write}(\mathtt{out}, s)\end{array}$$

iii. O sistema termina quando todos os inversores tiverem o estado $s = 0$.

a. Construa um FOTS que descreva este sistema e implemente este sistema, numa abordagem BMC ("bouded model checker") num traço com n estados.
b. Verifique usando k-lookahead se o sistema termina
ou, em alternativa,
c. Explore as técnicas que estudou para verificar em que condições o sistema termina.

2. Pretende-se construir um **autómato híbrido** que modele uma situação definida por 3 navios a navegar num lago infinito. Cada navio é caracterizado pela sua *posição* no plano (x, y), a sua *rota* medida num ângulo com o eixo horizontal em unidades de 15^o, e uma velocidade que assume apenas 2 valores: $1\,\mathtt{m/s}$ ("low") e $10\,\mathtt{m/s}$ ("high").
   a. Os navios conhecem o estado uns dos outros.
   b. Na presença de uma eminente colisão entre dois navios (ver notas abaixo) , ambos os navios passam à velocidade "low" e mudam a rota, para bombordo (esquerda) ou estibordo (direita) em não mais que uma unidade de 15^o. Após se afastarem para uma distância de segurança regressam à velocidade "high".
   c. Pretende-se verificar que realmente os navios navegam sem colisões.

1. Na aritmética linear inteira (LIA) pode-se escrever uma relação $|x - z| \leq r$, sendo r uma constante, como duas relações lineares $(x \leq z + r) \wedge (z \leq x + r)$.
2. Para representar um ponto em movimento no plano deve-se usar um ponto $P \equiv (x, y, t)$: formado pelas coordenadas de posição no plano (x, y) e pelo tempo t.
3. Para detetar uma eventual colisão (dois pontos P_0, P_1 a ocuparem posições próximas em tempos próximos) temos de ver a distância entre os dois pontos que pode ser definida como
$$\mathsf{d}(P_0, P_1) \;\equiv\; \max\{|x_0 - x_1|\,,\, |y_0 - y_1|\,,\, v\,|t_0 - t_1|\}$$
sendo v uma constante apropriada (por exemplo, a velocidade média dos pontos).
Desta forma a condição de iminente colisão pode ser escrita como $\mathsf{d}(P_0, P_1) \leq r$; ou seja
$$|x_0 - x_1| \leq r \;\wedge\; |y_0 - y_1| \leq r \;\wedge\; |t_0 - t_1| \leq r/v$$

Trabalho 4

1. Considere o seguinte programa, em Python anotado, para multiplicação de dois inteiros de precisão limitada a 16 bits. Assume-se que os inteiros são representáveis na teoria `BitVecSort(16)` do Z3.

```
    assume m >= 0 and n >= 0 and r == 0 and x == m and y == n
 0: while y > 0:
 1:     if y & 1 == 1:
             y , r  = y-1 , r+x
```

```
2:    x , y = x<<1  ,  y>>1
3: assert r == m * n
```

1. Tenha em atenção a atribuição simultânea do Python : `x , y = f(x,y) , g(x,y)` implica que simultaneamente a `x` é atribuído o valor `f(x,y)` e a `y` é atribuído `g(x,y)` .
2. Recorde que anotações não modificam o estado, nomeadamente o "program counter"

a. Usando indução verifique a terminação deste programa.
b. Pretende-se verificar a correção parcial deste programa usando duas formas alternativas para lidar com programas iterativos: **havoc** e ***unfold.***
    i. Usando o comando **havoc** e a metodologia WPC (*weakest pre-condition)* gere a condição de verificação que garanta a correção parcial.
    ii. Usando a metodologia SPC (*strongest post-condition*), para um parâmetro inteiro N, gere o fluxo que resulta do *unfold* do ciclo N vezes e construa a respetiva condição de verificação.
c. Codifique, em SMT's e em ambos os casos, a verificação da correcção parcial.

2. Considere um **sistema híbrido** formado por 4 autómatos híbridos: três navios (análogos aos do trabalho TP3) e um controlador. Neste sistema cada autómato desconhece o estado dos restantes e comunica com eles exclusivamente através de **eventos**.
A propriedade de segurança é a mesma da do trabalho TP2: ausência de colisões entre navios.
Para isso a área de navegação é dividida em **setores** e o controlador assegura que, em qualquer instante, cada setor não contém mais do que um navio.
Assim
    a. Cada navio está, em cada estado, numa de três *velocidades* v possíveis: $10\,\mathbf{m/s}$ (high) , $1\,\mathbf{m/s}$ (low) e 0 (stop). As transições high $\leftrightarrow$ low têm uma duração mínima de $500\,\mathbf{seg}$; as transições low $\leftrightarrow$ **stop** têm uma duração mínima de $50\,\mathbf{seg}$.
    b. Cada navio está, em cada estado, numa *rota* $r \in \{0..23\}$; cada valor de r identifica um ângulo múltiplo de 15^o (também designado por $\mathtt{hora}$).
    c. A área de navegação é dividida numa matriz $N \times N$ de setores quadrados com $1\,\mathbf{km}$ de lado. Cada setor é identificado por um par de índices $0 \leq \mathtt{linha}, \mathtt{coluna} < N$. Cada navio está, em cada estado, num único setor.
    d. O estado do controlador incluiu o seu setor e a sua velocidade.

    A navegação é determinada pelas seguintes regras

    a. Um navio só muda de rota ou velocidade quando muda de setor.
    b. Quando um navio entra ou sai de um setor emite um evento, que identifica o navio e os setores envolvidos (de onde vem e para onde vai). Este evento sincroniza com o controlador que assim controla as mudanças de setor de cada navio.
    c. Se existir risco de dois navios estarem simultaneamente no mesmo setor, o controlador deve fazer que um deles mude de rota ou espere que o outro abandone esse setor.
    d. Dois navios em setores adjacentes estão ambos em velocidade low ou stop .

    Pretende-se verificar, dada uma determinada posição inicial dos três navios, a seguinte propriedade de segurança:

    > *Em qualquer traço e em qualquer estado, nenhum setor contém mais do que um navio.*