PGENE503 – Sistemas em Tempo Real
Primeira Lista de Exercícios

1) Descreva com a suas próprias palavras o que você entende sobre verificação de modelos (*model checking*) e como isto difere de outras técnicas usadas para verificação de hardware e software. A sua resposta deve indicar quais são as entradas para verificar o software e quais são as suas saídas, e quais benefícios e implicações isto tem. Você também deve incluir uma explicação dos termos *modelo* e *propriedade de corretude*.

2) Mostre que as duas expressões ***if-then-else*** abaixo são equivalentes:

```
!(a||b)?h:!(a==b)?f:g          !(!a||!b)?g:(!a&&!b)?h:f
```

Você pode assumir que as variáveis têm somente um bit.

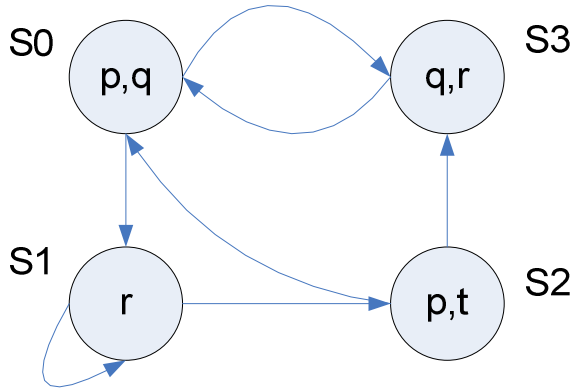
3) Considere três pessoas A, B e C que querem sentar em uma fileira, mas:

- A não quer sentar próximo do C.
- A não quer sentar na cadeira da esquerda.
- B não quer sentar no lado direito do C.

Escreva uma fórmula proposicional que é satisfeita se e somente se existe uma atribuição de assentos para as três pessoas que satisfaz todas as restrições. Utilize o SMT Solver Z3 para checar a satisfatibilidade da fórmula produzida. Caso a fórmula seja satisfeita, forneça o contraexemplo. Dica: Modele o problema usando variáveis do tipo $x_{i,j}$, onde $x_{i,j}$ é verdadeiro se e somente se a pessoa i estiver sentada no assento j.

4) Um sistema de redundância modular tripla (RMT) consiste de três processadores e um único eleitor. Cada componente deste sistema pode falhar. A confiabilidade é aumentada permitindo que todos os processadores executem o mesmo programa. O eleitor elege a maioria dos votos das saídas dos três processadores. Desta forma, se um único processador falhar, o sistema pode ainda produzir saídas confiáveis. Cada componente pode ser reparado. Assume-se que somente um componente de cada vez pode falhar e somente um componente de cada vez pode ser reparado. No caso de falha do eleitor, o sistema inteiro falha. No reparo do eleitor, assuma-se que o sistema inicia-se como novo, i.e., com três processadores e um eleitor. Modele este RMT usando estrutura Kripke. Dica: Considere estados da forma $s_{i,j}$ onde i denota o número de processadores que estão atualmente funcionando ($0<i\leq3$) e j é o número de eleitores operacionais ($j=0,1$).

5) Considere a seguinte estrutura *Kripke*:

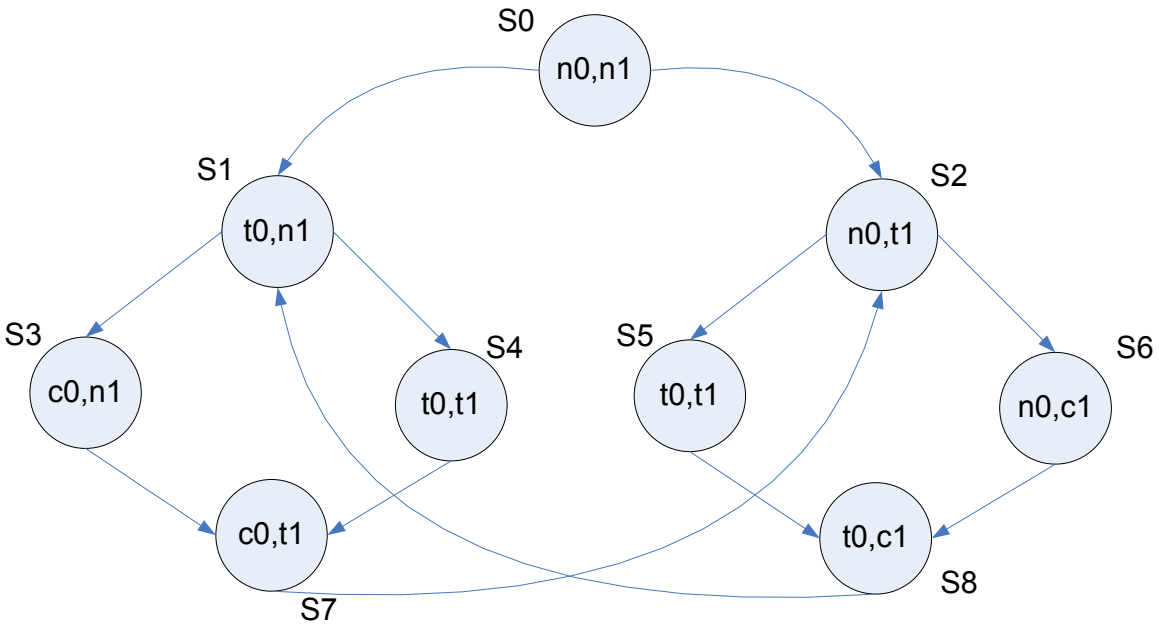


Verifique se S_0 satisfaz ϕ e S_2 satisfaz ϕ para as seguintes fórmulas CTL ϕ:

a) **AF** q
b) **AG EF** (p $\vee$ r)
c) **EX EX** r
d) **AG AF** q

Explique a sua resposta.

6) Considere o seguinte modelo de Kripke da exlcusão mútua:



onde as proposições atômicas n_i, t_i e c_i com i = 0,1 significa "processo *i* em uma seção não-crítica", "processo *i* tentando entrar na região crítica" e processo *i* na região crítica", respectivamente. Aplique o algoritmo de verificação de modelo do CTL para checar se as seguintes fórmulas se mantêm no estado S_0 deste modelo:

a) **AG** $\neg(C_0 \wedge C_1)$
b) **AG**$((t0 \Rightarrow$ **AF** $C_0) \wedge (t1 \Rightarrow$ **AF** $C_1))$

Note que você precisará transformar estas fórmulas para as aquelas que são semanticamente equivalentes para que você possa aplicar o algoritmo.

7) Expresse as seguintes propriedades em CTL sempre que possível:

a) Depois de *p*, *q* nunca é verdadeiro. (esta restrição é requerida em todos os possíveis caminhos de computação.)
b) O evento *p* precede ambos *s* e *t* em todos os caminhos de execução. (você pode achar mais fácil expressar primeiro a negação desta propriedade.)
c) Entre os eventos *q* e *r*, o evento *p* nunca é verdadeiro.

8) Construa uma estrutura Kripke para a implementação de uma versão simplificada da técnica de inserção de bytes (*byte stuffing*) usada na camada de enlance em redes de computadores.

```
#define DLE 16
#define STX 2
#define ETX 3
int main (void) {
 uchar in[6] = {DLE, STX, NUL, DLE, ETX, '\0'};
 uchar out[6];
 int i = 0;
 int j = 0;
 while (in[i] != '\0') {
  switch (in[i]) {
   case (DLE):
    if (in[i+1]==STX || in[i+1]==ETX) {
     out[j] = in[i];
    } else {
     out[j] = in[i];
     out[++j] = DLE;
    };
    break;
   default:
    out[j] = in[i];
    break;
  }
  i++;
  j++;
 }
```

```
  out[j] = '\0';
  return 0;
}
```

Considere que a proposição atômica *p* verifica se out[4] é igual a ETX e a proposição atômica q verifica se out[5] é igual a ETX. Deste modo, verifique se a estrutura Kripke do código acima satisfaz a propriedade **AG** ($p \wedge q$).

9) Um dado sistema de automação industrial consiste em monitorar e controlar um ambiente externo usando sensores, atuadores e outras interfaces de entrada/saída. Usualmente, tal sistema pode ser implementado através de um programa concorrente que consiste basicamente de uma coleção de processos computacionais que executam em paralelo e que podem interagir entre si. Para este caso em particular, considere que o sistema de automação foi implementando usando três processos (i.e., P_1, P_2 e P_3) independentes e cada processo consiste de dois comandos (e.g., a_1 e b_1 para o processo P_1):

P_1:	**P_2**	**P_3**
a_1	a_2	a_3
b_1	b_2	b_3

Note que um programa concorrente pode produzir diferentes intercalações (i.e., *interleavings*) dependendo do algoritmo de escalonamento. O número de intercalações é exponencial no número de processos e comandos. Exemplos de possíveis intercalações são:
$a_1, b_1, a_2, b_2, a_3, b_3$;
$a_3, b_3, a_2, b_2, a_1, b_1$;
$a_1, a_2, b_1, a_3, b_2, b_3$.
Você não encontrará um intercalação onde, por exemplo, b_1 executa antes de a_1. De posse destas informações, você deve responder as seguintes questões:

a) Implemente um algoritmo para imprimir todas as possíveis intercalações. Qual é a complexidade da sua solução?
b) Considere que as ações de todos os processos consistem basicamente em incrementar uma variável global "x", que é inicializada para zero antes de executar todos os processos. Neste caso, você geraria intercalações redundantes? (i.e., intercalações que sempre produzem o mesmo resultado) Caso positivo, implemente um algoritmo para eliminar as intercalações redundantes.

Data de entrega: 26 de setembro de 2013 (quinta-feira).
Após esta data será descontado 2 pontos por dia de atraso.
A lista de exercícios deve ser resolvida e entregue individualmente.

12/09/2013