Logica Computacional 23 – Novembro – 2023 LCC

Grupo 06:

- João Manuel Franqueira da Silva, A91638
- Eduardo Manuel Sousa Pereira, A70619

$TP3 - Problema\ 1$

In [1]:
```
from pysmt.shortcuts import *
import pysmt.typing as types
import random as rn
from pysmt.typing import BOOL, REAL, INT, BVType, STRING
from IPython.display import Latex
import itertools
```

O algoritmo estendido de Euclides (EXA) aceita dois inteiros constantes $a, b > 0$ e devolve inteiros r, s, t tais que $a * s + b * t = r$ e $r = \gcd(a, b)$. Para além das variáveis r, s, t o código requer 3 variáveis adicionais r', s', t' que representam os valores de r, s, t no "próximo estado".

```
INPUT  a, b : Int
0:     assume  a > 0 and b > 0 and a < N and b < N
       r, r', s, s', t, t' = a, b, 1, 0, 0, 1
1:     while r' != 0
         q = r div r'
2:       r, r', s, s', t, t' = r', r - q × r', s', s - q × s', t', t - q ×
t'
3:     OUTPUT r, s, t
```

Como o solver $msat$ não suporta operações de divisão e multiplicações não lineares, teremos que acrescentar mais 4 variávies e 2 transições:

1. aux que calcula recursivamente o valor de r div $r\_$ no estado 1, incrementando o valor de q e decrementando o seu valor por $r\_$ até não poder se efetuar mais subtrações, seguindo então para o estado 2.

2. $auxr$, $auxs$ e $auxt$ que calculam recursivamente no estado 2, o valor de $q\,r\_$, $q\,s\_$, $q * t\_$, respetivamente, decrementando q até este chegar a 0, e ao mesmo tempo, somando a estas novas variáveis $r\_$, $s\_$, $t\_$, respetivamente, q vezes, e voltando novamente ao estado 1.

Construção de um FOTS que descreva o comportamento do programa.

Vamos começar por declarar as variáveis relativas ao nosso problema.

Podemos ainda considerar uma variável pc que indicará a instrução em que nos encontramos.

In [2]:

```python
NN=40

def genState(vars,s,i):
    state = {}
    for v in vars:
        state[v] = Symbol(v+'!'+s+str(i),INT)
    return state

def declare1(i):
    state = {}
    state['pc'] = Symbol('pc'+str(i),INT)
    state['r'] = Symbol('r'+str(i),INT)
    state['s'] = Symbol('s'+str(i),INT)
    state['t'] = Symbol('t'+str(i),INT)
    state['q'] = Symbol('q'+str(i),INT)
    state['r_'] = Symbol('r_'+str(i),INT)
    state['s_'] = Symbol('s_'+str(i),INT)
    state['t_'] = Symbol('t_'+str(i),INT)
    state['aux'] = Symbol('aux'+str(i),INT)
    state['auxr'] = Symbol('auxr'+str(i),INT)
    state['auxs'] = Symbol('auxs'+str(i),INT)
    state['auxt'] = Symbol('auxt'+str(i),INT)
    return state
```

Definimos então o predicado $init$, que dado um estado e dois inteiros a e b verifica se este é um estado inicial válido, atribuindo os valores da a a r e de b a $r\_$ precisamos de observar a pré-condição do nosso programa:

$$a > 0 \quad \land \quad b > 0 \quad \land \quad r = a \quad \land \quad r\_ = b \quad \land \quad s = 1 \quad \land \quad s\_ = 0 \quad \land \quad t = 0 \quad \land \quad t\_ = 1$$

As variáveis nao referenciadas no início do programa podem ser inicializadas iguais a 0.

In [3]:

```python
def init1(state,a,b):
    A=Equals(state['r'],Int(a))
    B=Equals(state['r_'],Int(b))
    C=Equals(state['pc'],Int(0))
    D=Equals(state['s'],Int(1))
    E=Equals(state['t'],Int(0))
    F=Equals(state['s_'],Int(0))
    G=Equals(state['t_'],Int(1))
    H=Equals(state['q'],Int(0))
    K=Equals(state['aux'],Int(0))
    r=And(A,B,C,D,E,F,G,H,K)
    return r
```

Definimos agora a função transição que recebe dois estados como argumento, o atual $(curr)$ e o que pretendemos transitar para $(prox)$, e verifica se esses dois estados se referem a uma transição válida do programa.

Referindo às variáveis do estado $curr$ por: pc, r, $r\_$, s, $s\_$, t, $t\_$, q, aux, $auxr$, $auxs$, $auxt$

E do estado $prox$ por: pc', r', $r\_'$, s', $s\_'$, t', $t\_'$, q', aux', $auxr'$, $auxs'$, $auxt'$

Temos então as seguintes transições possíveis:

Quando a variável pc tiver valor 0, no próximo estado do programa iremos estar na condição do ciclo while, ou seja, pc' vai ter o valor 1 e todas as variáveis mantêm o mesmo valor. Atribuimos também o valor de r a aux', para este poder calcular a divisão no próximo estado.

- $pc = 0 \quad \land \quad pc' = 1 \quad \land \quad r' = r \quad \land \quad r\_' = r\_ \quad \land \quad s' = s \quad \land \quad s\_' = s\_ \quad \land \quad t' = t$

Quando a variável pc tiver o valor 1 podemos prosseguir para 3 estados:

1. Se $r\_ \neq 0$ e $aux \neq 0$, sabemos que o valor total da divisão entre r e $r\_$ ainda não foi calculado, então continuamos no estado 1, subtraindo $r\_$ a aux e incrementado q, mantemos os valores das variáveis principais e efetuamos:

- $q' = q + 1 \quad \land \quad aux' = aux - r\_$

1. Se $r\_ \neq 0$ e $aux < r\_$, sabemos que neste momento, q tem o valor exato de $r//r\_$, então entramos dentro do ciclo, pc' terá o valor 2, os valores das variáveis são mantidas e inicializamos $auxr\_'$, $auxs\_'$ e $auxt\_'$ a 0.

- $pc = 1 \quad \land \quad pc' = 2 \quad \land \quad r\_ \neq 0 \quad \land \quad r' = r \quad \land \quad r\_' = r\_ \quad \land \quad s' = s \quad \land \quad s\_' = s\_$

1. Caso contrário, a condição do ciclo não é satisfeita, e pc' terá o valor 3 e os valores das variáveis são mantidos.

- $pc = 1 \quad \land \quad pc' = 3 \quad \land \quad r\_ = 0 \quad \land \quad r' = r \quad \land \quad r\_' = r\_ \quad \land \quad s' = s \quad \land \quad s\_' = s\_$

Quando pc tiver o valor 2, e $q \neq 0$ sabemos que ainda não efetuamos suficientes somas para guardar o valor de $q * r\_$ em $auxr$, $q * s\_$ em $auxs$, e $q * t\_$ em $auxt$, então decrementamos q , mantemo-nos no estado 2, guardamos o valor das restantes variáveis e efetuamos.

- $auxr' = auxr + r\_ \quad \land \quad auxs' = auxs + s\_ \quad \land \quad auxt' = auxt + t\_ \quad \land \quad q' = q - 1$

Quando pc tiver o valor 2 e q==0 sabemos que já temos os valores das multiplicações armazenados, entao:

- $pc = 2 \quad \land \quad pc' = 1 \quad \land \quad r\_! = 0 \quad \land \quad q = r/r\_ \quad \land \quad r\_' = r - auxr \quad \land \quad s\_' = s - a$

Por último podemos definir ainda uma transição do estado final para ele próprio, em que o pc, tal como as variavéis se mantêm.

- $pc = 3 \quad \land \quad pc' = 3 \quad \land \quad r' = r \quad \land \quad r\_' = r\_ \quad \land \quad s' = s \quad \land \quad s\_' = s\_ \quad \land \quad t' = t$

In [4]:
```
def trans1(curr,prox):  #r//r_ --- a//b
    #0 - 1
    A=Equals(curr['pc'],Int(0))
    B=Equals(prox['pc'],Int(1))
    C=Equals(prox['r'],curr['r'])
    D=Equals(prox['s'],curr['s'])
    E=Equals(prox['t'],curr['t'])
    F=Equals(prox['r_'],curr['r_'])
    G=Equals(prox['s_'],curr['s_'])
```

```
    H=Equals(prox['t_'],curr['t_'])
    I=Equals(prox['q'],curr['q'])
    J=Equals(prox['aux'],curr['r'])
    t01=And(A,B,C,D,E,F,G,H,I,J)

    #1 - 1                              #enquanto pode dividir (a calcular o q...) e so quan
    a=Equals(curr['pc'],Int(1))
    b=Equals(prox['pc'],Int(1))
    c=GE(curr['aux'],curr['r_'])
    d=Equals(prox['aux'],Minus(curr['aux'],curr['r_']))
    e=Equals(prox['q'],Plus(Int(1),curr['q']))
    f=Equals(prox['r'],curr['r'])
    g=Equals(prox['s'],curr['s'])
    h=Equals(prox['t'],curr['t'])
    i=Equals(prox['r_'],curr['r_'])
    j=Equals(prox['s_'],curr['s_'])
    k=Equals(prox['t_'],curr['t_'])
    l=Not(Equals(curr['r_'],Int(0)))
    t11=And(a,b,c,d,e,f,g,h,i,j,k,l)

    #1-2                                #quando o q ja esta calculado (passa para o 2)
    a=Equals(curr['pc'],Int(1))
    b=Equals(prox['pc'],Int(2))
    c=Not(GE(curr['aux'],curr['r_']))
    d=Equals(prox['r'],curr['r'])
    e=Equals(prox['s'],curr['s'])
    f=Equals(prox['t'],curr['t'])
    g=Equals(prox['r_'],curr['r_'])
    h=Equals(prox['s_'],curr['s_'])
    i=Equals(prox['t_'],curr['t_'])
    j=Equals(prox['q'],curr['q'])
    k=Equals(prox['auxr'],Int(0))
    l=Equals(prox['auxs'],Int(0))
    m=Equals(prox['auxt'],Int(0))
    t12=And(a,b,c,d,e,f,g,h,i,j,k,l,m)

    #2-2     #incrementa o auxr , auxs, auxt com r_,s_,t_ e decrementa o q, assim acab
    a=Equals(curr['pc'],Int(2))
    b=Equals(prox['pc'],Int(2))
    c=GE(curr['q'],Int(1))
    d=Equals(prox['q'],Minus(curr['q'],Int(1)))   #q decrementa
    e=Equals(prox['auxr'],Plus(curr['auxr'],curr['r_']))  #aumenta os auxs todos
    f=Equals(prox['auxs'],Plus(curr['auxs'],curr['s_']))
    g=Equals(prox['auxt'],Plus(curr['auxt'],curr['t_']))
    h=Equals(prox['r'],curr['r'])
    i=Equals(prox['s'],curr['s'])
    j=Equals(prox['t'],curr['t'])
    k=Equals(prox['r_'],curr['r_'])
    l=Equals(prox['s_'],curr['s_'])
    m=Equals(prox['t_'],curr['t_'])
    t22=And(a,b,c,d,e,f,g,h,i,j,k,l,m)

    #2-1     auxs tem os valores de q*r_ , q*s_, q*t_ agora precisamos de voltar ao ini
    a=Equals(curr['pc'],Int(2))
    b=Equals(prox['pc'],Int(1))
    c=Not(GE(curr['q'],Int(1)))
    d=Equals(prox['r'],curr['r_'])
    e=Equals(prox['s'],curr['s_'])
    f=Equals(prox['t'],curr['t_'])
    g=Equals(prox['r_'],Minus(curr['r'],curr['auxr']))
```

```
    h=Equals(prox['s_'],Minus(curr['s'],curr['auxs']))
    i=Equals(prox['t_'],Minus(curr['t'],curr['auxt']))
    j=Equals(prox['q'],Int(0))
    k=Equals(prox['aux'],prox['r'])
    t21=And(a,b,c,d,e,f,g,h,i,j,k)

    #1-3
    a=Equals(curr['pc'],Int(1))
    b=Equals(prox['pc'],Int(3))
    c=Equals(prox['r'],curr['r'])
    d=Equals(prox['s'],curr['s'])
    e=Equals(prox['t'],curr['t'])
    f=Equals(prox['r_'],curr['r_'])
    g=Equals(prox['s_'],curr['s_'])
    h=Equals(prox['t_'],curr['t_'])
    i=Equals(curr['r_'],Int(0))
    t13=And(a,b,c,d,e,f,g,h,i)

    #3-3
    a=Equals(curr['pc'],Int(3))
    b=Equals(prox['pc'],Int(3))
    c=Equals(prox['r'],curr['r'])
    d=Equals(prox['s'],curr['s'])
    e=Equals(prox['t'],curr['t'])
    f=Equals(prox['r_'],curr['r_'])
    g=Equals(prox['s_'],curr['s_'])
    h=Equals(prox['t_'],curr['t_'])
    t33=And(a,b,c,d,e,f,g,h)
    return Or(t01,t11,t12,t22,t21,t13,t33)
```

Temos a função $genTrace$, que, ao receber um a e um b, cria uma lista Xde $n+1$ estados em que:

- Aplicamos a função init ao primeiro estado: $init1(X[0], a, b)$
- Para todos estado consecutivos S, S', aplicamos a função transição: $\forall_{i=0}^{n}$ $trans1(X[i], X[i+1])$

In [5]:
```
def genTrace(vars,init1,trans1,n,a,b):
    with Solver(name="z3") as s:
        X = [genState(vars,'X',i) for i in range(n+1)]   # cria n+1 estados (com etiqu
        I = init1(X[0],a,b)
        Tks = [ trans1(X[i],X[i+1]) for i in range(n) ]

        if s.solve([I,And(Tks)]):      # testa se I /\ T^n  é satisfazível
            for i in range(n):
                print("Estado:",i)
                for v in X[i]:
                    print("          ",v,'=',s.get_value(X[i][v]))
```

1. Considere como propriedade de segurança

    `safety = (r > 0) and (r < N) and (r = a*s + b*t`)

    Prove usando k-indução que esta propriedade se verifica em qualquer traço do FOTS

Definimos então o predicado safety, que verifica se um estado S cumpre a relação de bezout, e se $0 < S(r) < NN$

In [6]:
```
def safety(s,x,y):
    return And(GT(s['r'],Int(0)),LT(s['r'],Int(NN)),Equals(s['r'],Plus(Times(s['s'],Ir
```

Queremos verificar este predicado é um invariante do sistema, para verificar esta propriedade, podemos utilizar o método da k_indução que consiste em:

- ϕ é válido nos estados iniciais, ou seja, $init(s) \rightarrow \phi(s)$
- Para qualquer estado, assumindo que ϕ é verdade, se executarmos uma transição, ϕ continua a ser verdade no próximo estado, ou seja, $\phi(s) \wedge trans(s, s') \rightarrow \phi(s')$.

Um processo parecido com este seria generalizar a indução assumindo no passo indutivo que o invariante é válido nos k estados anteriores.

In [7]:
```
def k_induction_always(vars,init1,trans1,safety,k,x1,y1):
    if(x1>0 and y1>0):
        with Solver(name="z3") as solver:
            X = [genState(vars,'X',i) for i in range(k+1)]
            solver.add_assertion(init1(X[0],x1,y1))
            for i in range(k-1):
                solver.add_assertion(trans1(X[i],X[i+1]))

            for i in range(k):
                solver.push()
                solver.add_assertion(Not(safety(X[i],x1,y1)))
                if solver.solve():
                    print(f"> Contradição! O invariante não se verifica nos k estados
                    for st in s:
                        print(f" pc = {solver.get_value(st['pc'])}",end=" ")
                        print(f" r = {solver.get_value(st['r'])}",end=" ")
                        print(f" s = {solver.get_value(st['s'])}",end=" ")
                        print(f" t = {solver.get_value(st['t'])}")
                        print(f" r_ = {solver.get_value(st['r_'])}",end=" ")
                        print(f" s_ = {solver.get_value(st['s_'])}",end=" ")
                        print(f" t_ = {solver.get_value(st['t_'])}",end=" ")
                        print(f" q = {solver.get_value(st['q'])}",end=" ")
                        print("------------------------------------------------------
                        print()

                    return
                solver.pop()


            X2 =[genState(vars,'X2',i) for i in range(k+2)]

            solver.add_assertion(Equals(X2[0]['r'],Int(x1)))
            solver.add_assertion(Equals(X2[0]['r_'],Int(y1)))
            for i in range(k+1):
                solver.add_assertion(safety(X2[i],x1,y1))
                solver.add_assertion(trans1(X2[i],X2[i+1]))

            solver.add_assertion(Not(safety(X2[-1],x1,y1)))
```

```
            if solver.solve():
                print(f"> Contradição! O passo indutivo não se verifica.")
                return
            print(f"> A propriedade verifica-se por k-indução (k={k}).")
```

1. Prove usando “Model-Checking” com interpelantes e invariantes prove também que esta propriedade é um invariante em qualquer traço de Σ.

Vamos então defenir como estado de erro, a negação do invariante $safety$, seja então:

In [8]:
```
def error(s,x,y,NN):
    return Or(Not(Equals(s['r'],Plus(s['s']*x,s['t']*y))),GE(s['r'],Int(NN)),LE(s['r']
```

Definimos a função invert que recebe a função que codifica a relação de transição e devolve a relação de transição inversa.

In [9]:
```
def invert(trans1):
    return lambda curr, prox: trans1(prox, curr)
```

In [10]:
```
def baseName(s):
    return ''.join(list(itertools.takewhile(lambda x: x!='!', s)))

def rename(form,state):
    vs = get_free_variables(form)
    pairs = [ (x,state[baseName(x.symbol_name())]) for x in vs ]
    return form.substitute(dict(pairs))

def same(state1,state2):
    return And([Equals(state1[x],state2[x]) for x in state1])
```

Usaremos então a algorítmo $model\_checking$, dado nas aulas práticas.

In [11]:
```
def model_checking(vars,init1,trans1,error,N,M,x1,y1,NN):
    with Solver(name="msat") as solver:

        # Criar todos os estados que poderão vir a ser necessários.
        X = [genState(vars,'X',i) for i in range(N+1)]
        Y = [genState(vars,'Y',i) for i in range(M+1)]
        transt = invert(trans1)

        # Estabelecer a ordem pela qual os pares (n,m) vão surgir. Por exemplo:
        order = sorted([(a,b) for a in range(1,N+1) for b in range(1,M+1)],key=lambda

        # Step 1 implícito na ordem de 'order' e nas definições de Rn, Um.
        for (n,m) in order:
            # Step 2.
            I = init1(X[0],x1,y1)
            Tn = And([trans1(X[i], X[i+1]) for i in range(n)])
            Rn = And(I, Tn)

            E = error(Y[0],x1,y1,NN)
            Bm = And([transt(Y[i], Y[i+1]) for i in range(m)])
            Um = And(E, Bm)

            Vnm = And(Rn, same(X[n], Y[m]), Um)
```

```
            if solver.solve([Vnm]):
                print("> O sistema é inseguro.")
                return
            else:
                # Step 3.
                A = And(Rn, same(X[n], Y[m]))
                B = Um
                C = binary_interpolant(A, B)

                # Salvaguardar cálculo bem-sucedido do interpolante.
                if C is None:
                    print("> O interpolante é None.")
                    break

                # Step 4.
                C0 = rename(C, X[0])
                T = trans1(X[0], X[1])
                C1 = rename(C, X[1])

                if not solver.solve([C0, T, Not(C1)]):
                    # C é invariante de T.
                    print("> O sistema é seguro.")
                    return
                else:
                    # Step 5.1.
                    S = rename(C, X[n])
                    while True:
                        # Step 5.2.
                        T = trans1(X[n], Y[m])
                        A = And(S, T)
                        if solver.solve([A, Um]):
                            #print("> Não foi encontrado majorante.")
                            break
                        else:
                            # Step 5.3.
                            C = binary_interpolant(A, Um)
                            Cn = rename(C, X[n])
                            if not solver.solve([Cn, Not(S)]):
                                # Step 5.4.
                                # C(Xn) -> S é tautologia.
                                print("> O sistema é seguro.")
                                return
                            else:
                                # Step 5.5.
                                # C(Xn) -> S não é tautologia.
                                S = Or(S, Cn)

    print("> Não foi provada a segurança ou insegurança do sistema.")
```

Resultados

In [12]:

```
genTrace(['pc','r','r_','s','s_','t','t_','q','aux','auxr','auxs','auxt'],init1,trans1
```

```
Estado: 0
          pc = 0
          r = 14
          r_ = 14
          s = 1
          s_ = 0
          t = 0
          t_ = 1
          q = 0
          aux = 0
          auxr = 0
          auxs = 0
          auxt = 0
Estado: 1
          pc = 1
          r = 14
          r_ = 14
          s = 1
          s_ = 0
          t = 0
          t_ = 1
          q = 0
          aux = 14
          auxr = 0
          auxs = 0
          auxt = 0
Estado: 2
          pc = 1
          r = 14
          r_ = 14
          s = 1
          s_ = 0
          t = 0
          t_ = 1
          q = 1
          aux = 0
          auxr = 0
          auxs = 0
          auxt = 0
Estado: 3
          pc = 2
          r = 14
          r_ = 14
          s = 1
          s_ = 0
          t = 0
          t_ = 1
          q = 1
          aux = 0
          auxr = 0
          auxs = 0
          auxt = 0
Estado: 4
          pc = 2
          r = 14
          r_ = 14
          s = 1
          s_ = 0
          t = 0
          t_ = 1
```

```
        q = 0
        aux = 0
        auxr = 14
        auxs = 0
        auxt = 1
Estado: 5
        pc = 1
        r = 14
        r_ = 0
        s = 0
        s_ = 1
        t = 1
        t_ = -1
        q = 0
        aux = 14
        auxr = 0
        auxs = 0
        auxt = 0
Estado: 6
        pc = 3
        r = 14
        r_ = 0
        s = 0
        s_ = 1
        t = 1
        t_ = -1
        q = 0
        aux = 0
        auxr = 0
        auxs = 0
        auxt = 0
Estado: 7
        pc = 3
        r = 14
        r_ = 0
        s = 0
        s_ = 1
        t = 1
        t_ = -1
        q = 0
        aux = 0
        auxr = 0
        auxs = 0
        auxt = 0
Estado: 8
        pc = 3
        r = 14
        r_ = 0
        s = 0
        s_ = 1
        t = 1
        t_ = -1
        q = 0
        aux = 0
        auxr = 0
        auxs = 0
        auxt = 0
Estado: 9
        pc = 3
        r = 14
```

```
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 10
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 11
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 12
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 13
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
```

```
            auxs = 0
            auxt = 0
Estado: 14
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 15
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 16
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 17
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 18
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
```

```
          t = 1
          t_ = -1
          q = 0
          aux = 0
          auxr = 0
          auxs = 0
          auxt = 0
Estado: 19
          pc = 3
          r = 14
          r_ = 0
          s = 0
          s_ = 1
          t = 1
          t_ = -1
          q = 0
          aux = 0
          auxr = 0
          auxs = 0
          auxt = 0
Estado: 20
          pc = 3
          r = 14
          r_ = 0
          s = 0
          s_ = 1
          t = 1
          t_ = -1
          q = 0
          aux = 0
          auxr = 0
          auxs = 0
          auxt = 0
Estado: 21
          pc = 3
          r = 14
          r_ = 0
          s = 0
          s_ = 1
          t = 1
          t_ = -1
          q = 0
          aux = 0
          auxr = 0
          auxs = 0
          auxt = 0
Estado: 22
          pc = 3
          r = 14
          r_ = 0
          s = 0
          s_ = 1
          t = 1
          t_ = -1
          q = 0
          aux = 0
          auxr = 0
          auxs = 0
          auxt = 0
Estado: 23
```

```
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 24
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 25
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 26
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
            aux = 0
            auxr = 0
            auxs = 0
            auxt = 0
Estado: 27
            pc = 3
            r = 14
            r_ = 0
            s = 0
            s_ = 1
            t = 1
            t_ = -1
            q = 0
```

```
        aux = 0
        auxr = 0
        auxs = 0
        auxt = 0
Estado: 28
        pc = 3
        r = 14
        r_ = 0
        s = 0
        s_ = 1
        t = 1
        t_ = -1
        q = 0
        aux = 0
        auxr = 0
        auxs = 0
        auxt = 0
Estado: 29
        pc = 3
        r = 14
        r_ = 0
        s = 0
        s_ = 1
        t = 1
        t_ = -1
        q = 0
        aux = 0
        auxr = 0
        auxs = 0
        auxt = 0
```

In [13]:
```
k_induction_always(['pc','r','r_','s','s_','t','t_','q','aux','auxr','auxs','auxt'],ir
```

```
> A propriedade verifica-se por k-indução (k=40).
```

In [14]:
```
model_checking(['pc','r','r_','s','s_','t','t_','q','aux','auxr','auxs','auxt'], init1
```

```
> O sistema é seguro.
```