Assignment 2

Tutorial 8

Ali

The Cigarette-Smoker's Problem

Patil in 1971

- 3 smokers are sitting at a table
- Each one has a different ingredient to make a cigarette
  - Tobacco, Cigarette Paper, Matches
- 2 of the 3 ingredients will be placed on the table (by an agent)
- The smoker with the third ingredient should pick them
- A new set of ingredients wont be placed on the table until the action of making and smoking a cigarette is completed
- The other smokers must not interfere
- Cooperation is needed among smokers

The Actions of HORACIO

- **HORACIO** (the smoker with tobacco)
  - Pick up the paper
  - Pick up the match
  - Roll, light and smoke the cigarette
  - Stubs out the cigarette
  - Return to HORACIO

```
HORACIO =
        ( horacio.paper.down ->
          horacio.spark.down ->
          horacio.smokes ->
          horacio.stubs.out -> HORACIO ).
```

The Actions of ARTHUR

- **ARTHUR** (the smoker with paper)
  - Pick up the tobacco
  - Pick up the match
  - Roll, light and smoke the cigarette
  - Stubs out the cigarette
  - Return to ARTHUR

```
ARTHUR =
        ( arthur.spark.down ->
          arthur.tobacco.down ->
          arthur.smokes ->
          arthur.stubs.out -> ARTHUR ).
```

The Actions of EDGAR

- **EDGAR** (the smoker with match)
  - Pick up the tobacco
  - Pick up the paper
  - Roll, light and smoke the cigarette
  - Stubs out the cigarette
  - Return to EDGAR

```
EDGAR =
        ( edgar.tobacco.down ->
          edgar.paper.down ->
          edgar.smokes ->
          edgar.stubs.out -> EDGAR ).
```

The Actions of AGATHA

AGATHA (the agent) -- 3 possibilities :

- Put tobacco and paper
- Put tobacco and match
- Put paper and match

```
AGATHA =
        ( agatha.selects ->
          agatha.paper.up ->
          agatha.spark.up ->  AGATHA
        | agatha.selects ->
          agatha.spark.up ->
          agatha.tobacco.up -> AGATHA
        | agatha.selects ->
          agatha.tobacco.up ->
          agatha.paper.up -> AGATHA ).
```

Resources

```
||PAPER = ( paper:SEMAPHORE(0) )
/{ { horacio.paper.down, edgar.paper.down }/paper.down,
   agatha.paper.up/paper.up }.
```

```
||TOBACCO = ( tobacco:SEMAPHORE(0) )
/{ { arthur.tobacco.down, edgar.tobacco.down }/tobacco.down,
    agatha.tobacco.up/tobacco.up }.
```

```
||SPARK = ( spark:SEMAPHORE(0) )
/{ { horacio.spark.down, arthur.spark.down }/spark.down,
    agatha.spark.up/spark.up }.
```

```
SEMAPHORE(I=0) =

        SEMAPHORE[I],

        SEMAPHORE[ b : 0..1 ] = (
                 when b == 0 up -> SEMAPHORE[1]
                | when b == 1 down -> SEMAPHORE[0]
                                  ).
```

CONSTRAINTS

```
||CONSTRAINTS =
        ( agatha.select:SEMAPHORE(1) )
        /{
          agatha.selects/agatha.select.down,
          { horacio, arthur, edgar }.stubs.out/agatha.select.up
        }.
```

The solution is not deadlock free

Trace to DEADLOCK:

agatha.selects

agatha.paper.up

horacio.paper.down

agatha.spark.up

arthur.spark.down

Patil:1971 & Parnas:1972

- We need to use conditional statements
- The proof is based on Petri nets
  - How to represents interrelated process by Petri nets
  - The problem needs a 2/3 net
  - 2/3 nets cannot expressed using primitives V, P
  - We need conditional statements
- Parnas showed that he was wrong!
  - The proof does not consider semaphore arrays

the agent

initially s = 1 and a,b,c,X,Y and Z = 0

R_a

r_a: P[s]
V[b]
V[c]
<u>go to</u> r_a

R_b

r_b: P[s]
V[a]
V[c]
<u>go to</u> r_b

R_c

r_c: P[s]
V[a]
V[b]
<u>go to</u> r_c

β_x: P[X]
V[s]
<u>go to</u> β_x

β_y: P[Y]
V[s]
<u>go to</u> β_y

β_z: P[Z]
V[s]
<u>go to</u> β_z

the smokers

X

α_x: P[b]
P[c]
V[X]
<u>go to</u> α_x

Y

α_y: P[a]
P[c]
V[Y]
<u>go to</u> α_y

Z

α_z: P[a]
P[b]
V[Z]
<u>go to</u> α_z

The Solution (processes and semaphores additional to the agent)

semaphore mutex; (initially 1)

integer t; (initially 0)

semaphore array S[1:6]; (initially 0)

δ_a: P (a);
P (mutex);
t $\leftarrow$ t + 1;
V(S[t]);
V (mutex);
go to δ_a;

δ_b: P (b);
P (mutex);
t $\leftarrow$ t + 2;
V(S[t]);
V (mutex);
go to δ_b;

δ_c: P (c);
P (mutex);
t $\leftarrow$ t + 4
V(S[t]);
V (mutex);
go to δ_c;

X
bc

α_x: P(S[6]);
t $\leftarrow$ 0;
[redacted]
V (X);
go to α_x;

Y
ac

α_y: P(S[5]);
t $\leftarrow$ 0;
[redacted]
V (Y);
go to α_y;

Z
ab

α_z: P(S[3]);
t $\leftarrow$ 0;
[redacted]
V (Z);
go to α_z;