|  |  |  |  |
| --- | --- | --- | --- |
| IST_logo | ***Industrial Processes Automation***  *MSc in Electrical and Computer Engineering*  *Scientific Area of Systems, Decision, and Control*  *Winter Semester 2016/2017* |  | *Group: A2*  *75167 - Francisco Oliveira*  *75322 - Catarina Gaspar*  *75637 - Inês Lourenço*  *84993 - Hugo Serieiro* |

***2nd Laboratory Assignment [[1]](#footnote-1)***

***Handling Faults in Keyboard Reading***

This laboratory assignment aims at studying Discrete Event Systems (DESs) in the aspects of modeling, analysis of properties and synthesis. Synthesis will be based on a recent methodology in the framework of supervised control. This assignment further develops the previous assignment in the keyboard reading component by introducing fault handling mechanisms.

The tools to be used in this work are MATLAB and a Petri Net editor. In the last part of the work the Schneider PLCs will be used once more to validate the proposed methodologies.

# Part C - DESs Synthesis, Supervised Control

The main purpose of the second laboratory assignment is the modeling and analysis of a discrete event system, based on a Petri net. That is in close relation with the first laboratory assignment.

The main objective of this last phase, part C, is the application of the *Supervisory Control* theory to diagnose and isolate a failure on the system. The failure to detect occurs when the user presses two keys simultaneously. In other words, the main objective of part C consists in designing and implementing a supervisor that detects the *multiple keys pressed error* and resumes the normal operation of the system.

Note: see in the course webpage tools helping this part of the assignment, namely the Matlab toolbox "spnbox" which allows, for example, showing invariant places of a Petri net. The toolbox is distributed as a ZIP file named "spnbox.zip".

**Q1.** Design a new Petri net that, instead of reading keyboard keys, as the net proposed in part A, simply detects there are multiple keys pressed at the same time. Draw the Petri using the editor PN\_Editor. Please indicate clearly the specific place of the net chosen to flag multiple keys detected. Simulate a sequence of events demonstrating the functionality of the net.

A petri net implementada para detectar várias teclas a serem pressionadas em simultâneo está descrita na Figura 1.

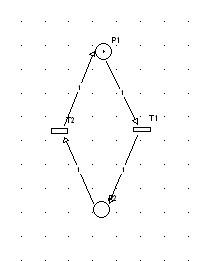


Figura 1 - Petri Net que detecta várias teclas a serem pressionadas em simultâneo

O lugar P1 é o lugar onde está o *token* na marcação inicial que representa a não existência de múltiplas teclas pressionadas num dado instante. A transição T1 é activa quando a variável “Linhas”, que detecta o número de linhas pressionadas, é maior que 1, passando assim para o estado P2. Este lugar vai funcionar como uma *flag* que detecta que várias teclas foram pressionadas em simultâneo. A transição T2 é activada quando a variável “Linhas” é menor que 2, detectando assim se 1 ou nenhuma tecla está a ser pressionada. Usando a seguinte sequência de teclas pode-se mostrar o funcionamento da Petri Net:

No primeiro e segundo instante o *token* mantém-se no lugar P1 pois o número de linhas activas nunca é maior que 1. No terceiro instante de tempo estão duas teclas a ser pressionadas na mesma coluna logo o número de linhas activas vai ser 2 activando assim a transição T1, que leva o *token* para o lugar P2. Com isto foram detectadas a existência de múltiplas teclas a serem pressionadas. No quarto instante de tempo o utilizador larga a tecla 7 ficando só a tecla 4, activando assim a transição T2 que leva o *token* de volta para o estado P1. Com isto a Petri está no estado que indica que não ocorreu nenhum problema de múltiplas teclas a serem pressionadas. No último instante de tempo o utilizador despressiona a tecla 4 mantendo-se assim o *token* no lugar P1.

**Q2.** Combine the Petri net designed in Q1 with the Petri net design in part A. Design a supervisor controller to handle the *multiple keys pressed error*. Specify the linear constraint(s) that must be used. Solve the problem resorting to the place invariant methodologies studied in this course.

Ao combinar as duas Petri nets, independentes uma da outra, a numeração dos *places* e das transições é feita começando na do detector de múltiplas teclas.

De modo a ignorar mais do que uma tecla primida ao mesmo tempo é necessário que os *token* não estejam simultaneamente num *place* correspondente a uma tecla (na Petri net original) e no *place* que corresponde a serem detectadas 2 ou mais teclas (na Petri net de detecção de múltiplas teclas). Deste modo, para a projecção do controlador supervisor as restrições linerares que este irá garantir são:

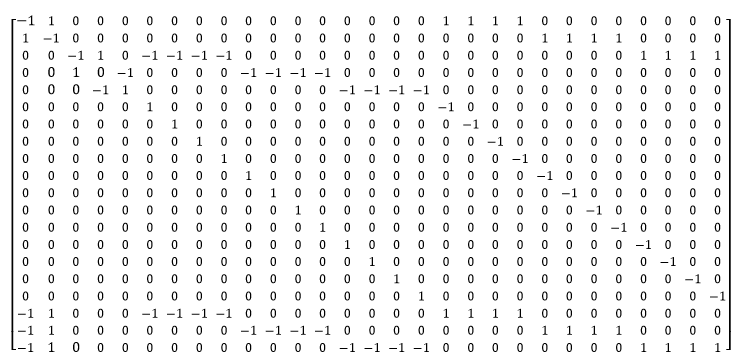
Estas restrições lineares correspondem a ter:

Onde é o vector correspondente ao estado,

De modo a descobrir o controlador supervisor a acrescentar é necessário descobrir a matriz incidência correspondente a ele, bem como o seu estado inicial:

Utilizando a função disponibilizada pelo professor *linenf.m* cujo *inputs* são as matrizes , da Petri net que inclui as duas máquinas independents (Petri net original e a da detecção de múltiplas teclas), bem com o estado inicial, a matriz *L* e o vector *b*.

Assim sendo o resultado obtido para a matriz de incidência da Petri net com controlador supervisor é:



E o estado inicial é:

Nestas duas matrizes, as linhas extra que foram acrescentadas à Petri net que incluia as duas petri nets independentes corresponde ao supervisor (são estas as últimas 3 linhas, pelo que o supervisor será composto por 3 novos *places*).

**Q3.** Design the complete resulting Petri net (part A and part C, Q1 and Q2). Identify clearly the supervisor.

A Petri net resultante, incluindo o supervisor a colorido, está representada na Figura 2.

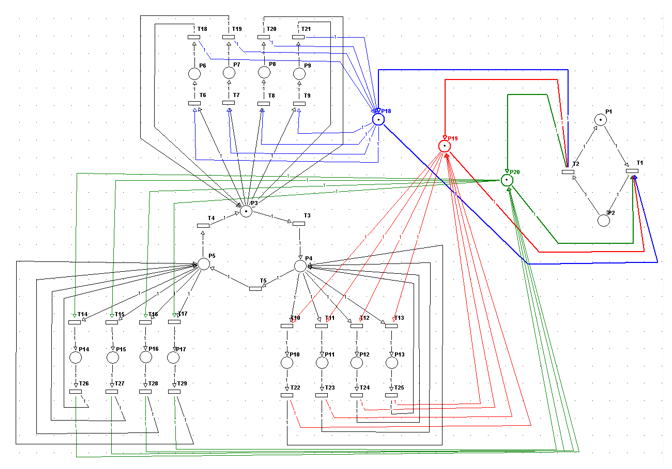


Figura 2 - Petri net com supervisor

**Q4.** Discuss briefly the properties relevant for the resulting DES based on the original Petri net and on this supervisor.

|  |  |
| --- | --- |
| **Order of importance** | **Property** |
| 1 | Temporal Invariance |
| 2 | Not Conservative |
| 3 | Reachability, Reachable set finito |
| 4 | Safeness |
| 5 | 1-Bounded |
| 6 | Liveness |
| 7 | Coverbility |

**Temporal Invariance**

A partir de um estado inicial há pelo menos uma sequência de acções que permite que o token volte ao estado inicial. Isto é fundamental em qualquer Petri net como a proposta neste trabalho, em que o programa deve ser cíclico tendo sempre o mesmo comportamento consoante a acção que é feita, pelo que foi a propriedade considerada mais importante.

Matematicamente isto pode ser visto vendo que transições ‘q’ levam a que o Petri net volte ao estado inicial, com . Para isto foi utilizada a função null(D, ‘r’) no matlab onde se obteve 14 soluções possíveis, ou seja há 14 sequências de transições que, dado um estado incial com um token no P1 e outro no P3, fazem voltar ao estado com um token no P1e P3. Uma das soluções apresentadas é a seguinte,

Em que é representada simplesmente a transição de ter duas ou mais linhas ao mesmo tempo e de seguida ser activada a transição de não ter nenhum ou uma linha activa.

**Conservation**

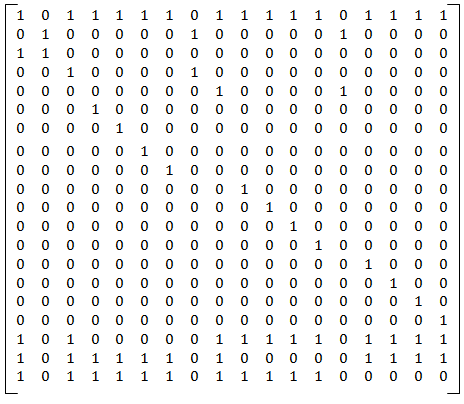
Uma Petri net é strictly conservative se na soma de todos os lugares em todos os estados o número de *tokens* for constante. No caso do nosso, ao se introduzido o supervisor, a Petri Net total deixa de ser conservativa, isto porque o *token* para passar para um *place* de uma tecla necessita que haja dois *tokens*: um no *place* da coluna corresondente e outro no *place* corresponte do supervisor. Quando passa para o p*lace* da tecla passa a haver menos um *token* na Petri Net nesses estados. A não conservação pode ser provada matematicamente através da seguinte propriedade: , em que a solução de *w* terá elementos nulos. Isto foi feito no matlab utilizando a função null(DT, ‘r’), onde se obteve o seguinte vários vectores de *w*, sendo que em todos eles se obteve pelo menos um elemento nulo, como se pode observar pelo seguinte exemplo,

Por análise da nossa Petri net é possível perceber que esta terá no máximo 5 *tokens* em cada estado (caso em que nenhuma linha esteja activa) e no mínimo 2 ( caso em estão activas duas ou mais linhas ao mesmo tempo). Pode ainda apresentar estados em que terá 4 *tokens* (caso em que apenas uma tecla é primida/uma linha está activa).

**Reachability**

Esta propriedade define se é possível garantir que, a partir de um certo lugar, se consegue aceder a qualquer outro existente na Petri net. Na Petri net proposta, de modo a garantir o correcto funcionamento do keyboard, é fundamental garantir que todos os lugares são acessíveis a partir de outro, não havendo nenhum dead lock. Esta Petri net tem um set de estados acessíveis finito (número de diferentes possíveis estados) uma vez que não são gerados *tokens*  livermente, e, apesar de não ser conservativa, é sempre possível repor o número de *tokens* incial.

Através das funções “graphnet” e “disp\_gr”, obteve-se a matriz RM indicada seguidamente, que é a matriz dos estados acessíveis, e criou-se a respectiva interpretação gráfica (Figura 3), que comprova que, como foi explicado, todos os estados são acessíveis.



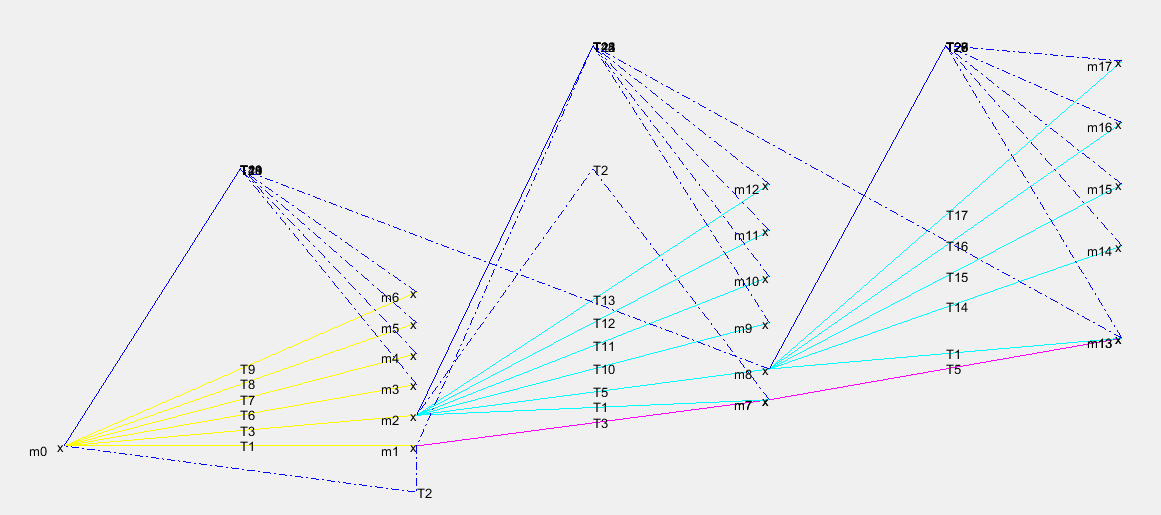


Figura 3 - Representação dos estados acessíveis

**-Bounded**

Uma Petri Net diz-se k-bounded quando cada lugar existente tem, no máximo, k *tokens* em todos os estados possíveis. De forma a garantir que uma Petri net seja bounded, é preciso que não haja em sítio nenhum a geração de *token* para garantir que o número *tokens* não aumente infinitamente. No caso da Petri net com supervisor não há geração de *tokens*, e não há nenhum peso associado a qualquer transição. Dito isto, e uma vez que o estado incial apenas inclui 1 *token* em cada *place* (com *token*), pode-se concluir que é 1-bounded.

**Safeness**

Esta propriedade é garantida sempre que a Petri net é 1-bounded uma vez que estas são equivalentes. Ou seja, em todos os estados há sempre no máximo 1 token em cada lugar.

**Liveness**

A Liveness de uma transição tem como objectivo classificar o número de vezes que uma transição pode ser acionada. No caso da Petri Net proposta neste laboratório, todas as transições podiam ser acionadas infinitas vezes. Para além disto, estando em qualquer estado, havia sempre uma sequência de eventos que levava a uma qualquer transição específica a ser acionada. Dado isto, pode-se concluir que todas as transições da Petri Net proposta são de nível 4.

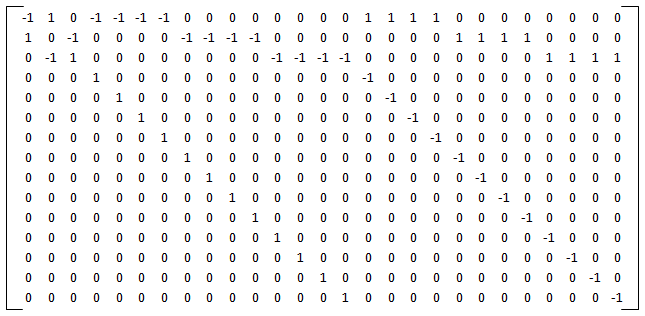
**Coverability**

Esta propriedade verifica-se analisando o número de *tokens* em cada lugar para dois estados consecutivos. Se, em qualquer lugar, o número de *tokens* no estado anterior for inferior ao número de *tokens* no estado seguinte, então o estado actual não é *coverable* por outros lugares alcansáveis. Na Petri net com supervisor, uma vez que ao se transitar de estado pelo menos um dos *tokens* é removido de um lugar para outro, nenhum estado é coverable por outro. Por esta razão, esta propriedade não foi tida em conta, pelo que a sua importância para o funcionamento do keyboard é reduzida e estando assim colocada em último lugar.

**Q5.** *[Aggregation of Petri nets]* Considering μp and Dp the state and incidence matrix of the Petri net designed in part A, and considering μd and Dd the state and incident matrix of the Petri net designed in part C (Q1), show that an enlarged net with state μ=[μpT μdT]T, running the two nets simultaneously, has an incidence matrix which is a function of the other two matrices, i.e. D=f(Dp, Dd). Make a Matlab program that illustrates the equivalence of using D versus using Dp and Dd.

Quando se agregam duas Petri nets independentes, a matriz de incidência, D, do conjuto será uma matriz diagonal por blocos, em que cada bloco representa a matriz incidência de cada uma das Petri nets independentes. Isto acontece pois nenhuma transição é partilhada pelas duas Petri net (são independentes entre elas). Assim sendo, tem-se que a rede alargada teria a seguinte matriz de incidência e o seguinte estado, respectivamente:

Ou seja, o funcionamento de ambas separadamente é igual ao funcionamento de ambas juntas numa só Petri net. Para mostrar este comportamento relativo à agregação de Petri nets foi feita uma função Matlab, *AgregacaoPetri.m* que mostra a matriz D das duas Petri Nets separadas, e depois a matriz de incidência de ambas as Petri nets agregadas numa só. Comecando primeiro pela matriz de incidência e o estado inicial da Petri net feita para a Parte A do laboratório, o resultado foi o seguinte:

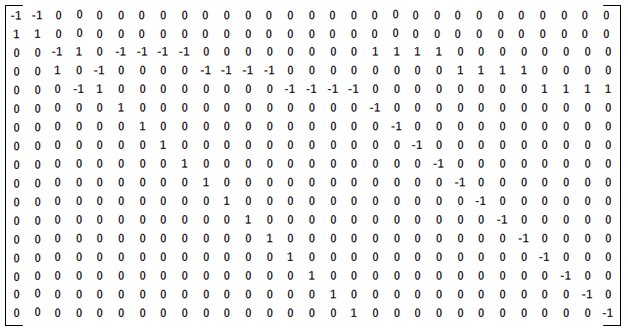


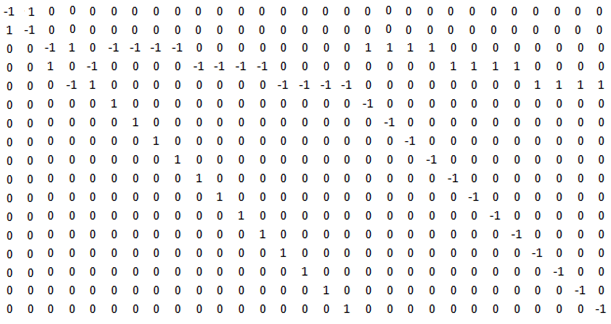
Dp:

A matriz de incidência e estado inicial obtido para a Petri net feita na Q1 deste laboratório foi a seguinte,

DD:

Ambas as Petri nets foram agregadas para formar a Petri net que iria efectuar o controlo do número de teclas que podem ser premidas num instante. Para a matriz de incidência da Petri net agregada obteve-se o seguinte resultado como o programa de matlab desenvolvido,



D =

Com isto verifica-se que a matriz de incidência e estado inicial do resultado da agregação das duas Petri nets é o esperado, ou seja,

.

De referir que quando foram agregadas as Petri nets da parte A e da parte C a que monitorizava múltiplas teclas a serem pressionadas foi dada prioridade, ou seja foi implementada primeiro. Com isto o resultado que se verifica no fim do programa matlab desenvolvido é,

.

**Q6.** *[Matlab simulation]* Simulate in Matlab the complete Petri net that reads keys (developed in part A), while detecting and reacting to multiple keys pressed simultaneously (developed in part C). As a starting point, consider using the code distributed in a zip file "kb\_sim\_lab2.zip" (made available in the webpage of the course). Note that are empty the functions implementing the conversion of the mapping of the state to the actuation/outputs (PN\_s2act.m, PN\_s2yout.m), and the conversion of keyboard line signals to transitions (PN\_tfire.m).

Para testar o sistema final com o supervisor implementado foi introduzida a seguinte sequência de teclas,

.

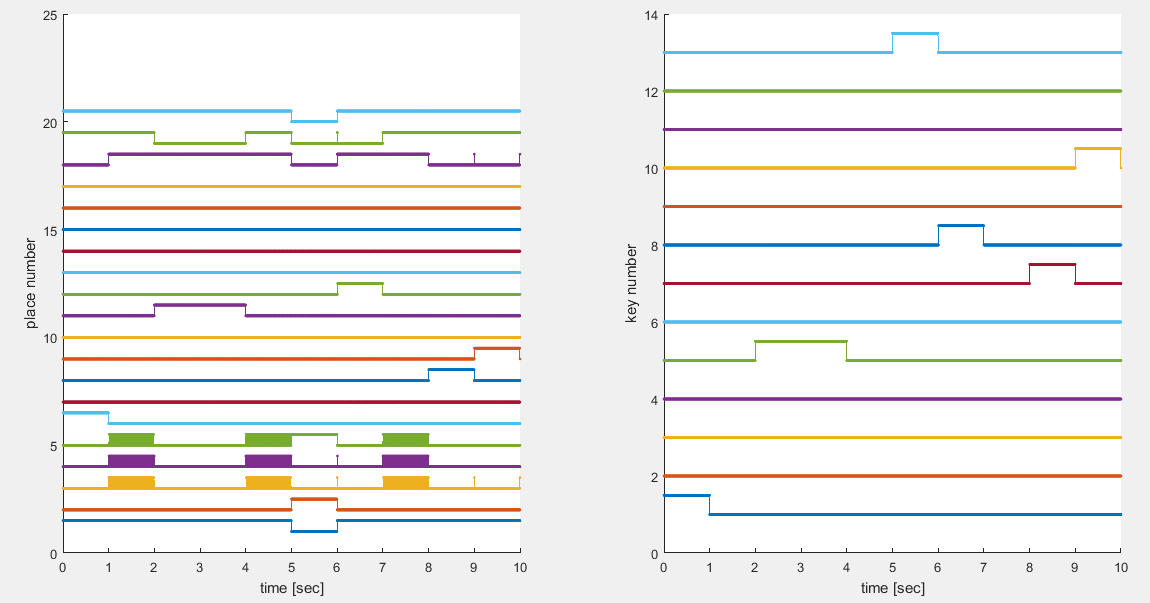
Com esta sequência de teclas obteve-se o resultado apresentado na Figura 4.

Figura 4 – Resultado do sistema do keyboard com multiple key correction implementado

Num primeiro instante a tecla 1 é pressionada, e verifica-se que de facto o *token* do lugar de controlo da coluna 1 (lugar 18) saiu, significando por isso que a tecla foi premida sem a existência de outra. De seguida nenhuma tecla é pressionada, logo todos os lugares do supervisor (lugar 18 a 20) têm um *token*. No instante de tempo em que são pressionadas duas teclas na mesma coluna (9 e 6) a transição T2 é activa pois todos os lugares do supervisor têm o *token* e a variável Linhas é maior que 1. Desta forma os *token* saiem dos lugares 18 a 20, tal como do lugar 1 e passa a existir um *token* no lugar 2. Logo, apesar de haver condições para as transições para as teclas 6, e 9 estarem activas, não há *token* no lugar 20 logo não há *tokens* nos lugares destas teclas. Comprova-se assim o correcto funcionamento do supervisor em termos de ignorar múltiplas teclas a serem pressionadas ao mesmo tempo, visto que o key “13” (não existe mas foi usado como flag de mútliplas teclas) está activo e nem a tecla 9 e 6 é detectada. No instante seguinte em que a tecla é largada a transição T1 é activa e volta um *token* para o lugar 1 e os *tokens* do supervisor voltam ao seu lugar original. O restante funcionamente corre da forma esperada sendo todas as teclas reconhecidas quando são pressionadas.

**Q7.** *[Discussion of the PLC implementation]* Discuss the implementation on the PLC of the supervisory control synthesized. Indicate eventual modifications needed in your key detection/validation program. Solve this question taking into account the Petri net to PLC code conversion tools introduced in the course classes.

Seguindo o exemplo do lab2\_B sobre o tst1\_blink\_on/off, é nescessário dar alguns dados importantes ao compilador, como a definição da nossa Petri Net, que inclui as matrizes Pre, Post e de estado inicial.

Deve-se dar também a informação acerca das transições prioritárias, para o nosso caso, a T1 deverá ser a mais prioritária, pois não se quer que, se houver mais do que uma linha activa, se leia qualquer tecla, pelo que a transição 1 derá ter maior prioridade que as outras todas de modo a que se detecte primeiro que tudo que estão 2 ou mais linhas activas:

tprio= [1];

E ainda informação acerca do timer de controlo entre certos lugares e transições, no nosso caso interessa-nos que o *token* se mantenha nos lugares correspontentes à activação de cada uma das colunas por pelo menos 0,02 segundos para que dê tempo ao PLC de ler os inputs correspondentes às linhas, e conseguir verificar quais estão activas. Assim sendo, de acordo com a nossa Petri net, este será o tempo que cada um dos lugares P3, P4 e P5 ficarão ativos, correspondendo às respetivas colunas estarem activas.

Para garantir que o PLC lê correctamente cada tecla pressionada, deve-se temporizar um pequeno tempo entre as transições das colunas, em nenhuma coluna está activa. Desse modo garante-se que o PLC não lê teclas a pensar que estão na coluna errada. Para tal, deve-se associar as transições T3, T5 e T4 um tempo de espera nelas próprias. Ou então, à petri net deve-se acrescentar mais 1 lugar correspondente a essa pausa, que se deverá expecificar essa temporização, para que o token permança nesse lugar antes de passar para a próxima coluna.

Para além da Petri Net, este ficheiro tem ainda que mapear os inputs físicos correspondentes a transições da Petri net. Neste caso:

input\_map = define\_input\_mapping;

O input do programa serão cada uma das linhas, ou seja, no nosso PLC os inputs %i0.2.4, %i0.2.5, %i0.2.6 e %i0.2.7. Caso esteja activa a linha 1, correspondente ao input %i0.2.4, as transições T6, T10 e T14 são acionadas. Se este input estiver desactivo as transições T18, T22 e T26 são activadas. Caso esteja activa a linha 2, correspondente ao input %i0.2.5, as transições T7, T11 e T15 são acionadas. Se este input estiver desactivo as transições T19, T23 e T27 são activadas.Caso esteja activa a linha 3, correspondente ao input %i0.2.6, as transições T8, T12 e T16 são acionadas. Se este input estiver desactivo as transições T20, T24 e T28 são activadas. Caso esteja activa a linha 4, correspondente ao input %i0.2.7, as transições T9, T13 e T17 são acionadas. Se este input estiver desactivo as transições T21, T25 e T29 são activadas.

É ainda necessário dizer que caso %i0.2.4 e %i0.2.5

ou %i0.2.4 e %i0.2.6

ou %i0.2.4 e %i0.2.7

ou %i0.2.5 e %i0.2.6

ou %i0.2.5 e %i0.2.7

ou %i0.2.6 e %i0.2.7 estiverem activos deve-se acionar a T1 (sendo esta prioritaria a qualaquer uma das outras mencionadas). Estas consições incluiem todas as condições em que há 2 ou mais linhas activas simultaneamente.

Para acionar a transição T2, será quando todos os inputs das linhas estejam desactivos, ou quando apenas uma das linhas está activa, apenas um input correspondente a linhas está activo. (ou seja, quando T1 não está activa).

O mesmo acontece para os outputs:

output\_map = define\_output\_mapping;

Estes são acionados quando o programa está a correr em ciclo em cada um dos estados P3, P4 e P5, indo corresponder respetivamente aos outputs %q0.4.4 (coluna 1 activa), %q0.4.5 (coluna 2 activa) e %q0.4.6 (coluna 3 activa). Para além disso, o output %q0.4.4 deverá estar activo também quando o *token* está no lugar P6, P7, P8 ou P9. Para além disso, o output %q0.4.5 deverá estar activo também quando o *token* se está no lugar P10, P11, P12 ou P13. E ainda, o output %q0.4.6 deverá estar activo também quando o *token* se está no lugar P14, P15, P16 ou P17.

Um dos scripts do compilador tem a função de converter os índices em endereços físicos do PLC, sendo as transições e lugares guardadas em palavras de memória. Para além disso, outro script é responsável pelos temporizadores dos estados referidos, e outro codifica a lista dos lugares em pequenos conjuntos de bits.

Com toda esta informação o compilador escreve num ficheiro o programa, em structured text, para dar ao PLC.

1. Original guide by Prof. Paulo J. Oliveira. Revised by Prof. José Gaspar (2016). [↑](#footnote-ref-1)